

SRI RAAJA RAAJAN
COLLEGE OF ENGINEERING AND TECHNOLOGY
(Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE FILE

NAME	: V.MANJU
DESIGNATION	: ASSISTANT PROFESSOR
DEPARTMENT	: CSE
SUB. CODE	: CS8501
SUBJECT NAME	: TOC
EVEN SEMESTER	: 2021-22



SRI RAAJA RAAJAN

COLLEGE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai)

WILLINGNESS REPORT

From

Mrs. V. Manju,
Asst. Professor,
Department of computer science and Engineering,
SRR CET
Karaikudi-630301

TO

The Principal,
SRR CET,
Karaikudi-630301

Sir/ Madam,

Sub: Willingness Report for Subject: - reg.

I hereby express my willingness to handle the following subject in the following order of priority.

S.NO	Name of the Subject	YEAR	Reason for Selection
1	CS8501 – THEORY OF COMPUTATION	III	Interested

Thanking You

Date: 10.09.21


Signature of the Staff




PRINCIPAL
Sri Raaja Raajan College of Engg.
Amaravathipuram, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, New Delhi & Affiliated to Anna University)

146 /4B1, Amaravathi Village,
Amaravathipudur (Po.),
Karaikudi – 630 301.
Ph : 04565 – 234230 / 326132

Fax : 04565 – 234430
Mobile : 73737 11343, 73737 11333
E-mail : srrcet2010@gmail.com
Website: www.srirajaraajan.in

Minutes of Subject allocating meeting

Date :

I V.MANJU hereby submit the minutes of department meeting held for the subject allocating on 09/09/2021 at 10.00 am based on the annexure I & annexure II.

S.NO	NAME OF THE STAFF	NAME OF THE SUBJECT	CLASS	NO OF HRS	STAFF SIGN
1	MANJU.V	Cryptography & Network Security – CS8792	IV	8	
2		Theory of Computation – CS8501	III	6	



Sridharan
HOD SIGN

[Signature]
PRINCIPAL
Sri Raaja Raajan College of Engg. & Tech.
Amaravathipudur, Karaikudi

Trust Office : No. 24/63, T.T. Nagar Church 3rd Street, Opp. to Golden Singar Hotel, Karaikudi – 630 001.

Ph : 04565 – 234230, Mobile : 73737 11343, 73737 11339, 73737 11322



SRI RAAJA RAAJAN

COLLEGE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai)

146/14B1, Amaravathi Village, Amaravathiputhur Post,
Karaikudi – 630 301, Sivagangai Dt, Tamil Nadu

Website : www.srirajaraajan.in, E-mail : srrcet2010@gmail.com, Ph: 04565-234230

CIRCULAR

DATE : 10.09.2021

Intimation of course allotment for faculties in the Department of Computer Science and Engineering during ODD Semester for B.E., Computer Science and Engineering.

S.NO.	NAME OF THE FACULTY	TITLE OF THE COURSE	COURSE CODE	YEAR & CLASS
1.	V.Manju, Assistant Professor	Cryptography and Network Security	CS8792	IV CSE
		Theory of Computation	CS8501	III CSE

Note: Faculties are asked to follow the syllabus issued by Anna University, Chennai.

Copy to:

1. The HoD
2. All the faculties of CSE Dept.
3. File Copy.


PRINCIPAL
Sri Raaja Raajan College of Engg. & Tech.,
Amaravathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ACADEMIC YEAR-2020-2021

TIME TABLE

DAY	I	II	BREAK	III	IV	LUNCH	V	VI	VII
	9.30-10.20	10.20-11.10		11.20-12.10	12.10-1.00		1.35-2.20	2.20-3.05	3.15-4.00
MONDAY							TOC		
TUESDAY	TOC				TOC				
WEDNESDAY	TOC								
THURSDAY	TOC	TOC							
FRIDAY							TOC		



HOD

PRINCIPAL

PRINCIPAL
Sri Raaja Raajan College of Engineering and Technology
Aneravathipudur, Karaikudi
Sivagangai Dist. Tanjore

MASTER TIME TABLE

DAY	HOURS	I	II	III	IV	V	VI	VII
	CLASS	9.30-10.20	10.20-11.10					
MONDAY	II YR	OOPS	DS	DM	DM	CE	DS LAB	DS LAB
	III YR	CN	OOAD	APC	OOAD	TOC	MPMC	ANT
	IV YR	HCI	CC	CNS	SPM	CC	CNS	POM
TUESDAY	II YR	OOPS	CE	DS	OOPS	DS	DM	DM
	III YR	TOC	APC	OOAD	TOC	MPMC LAB		MPMC LAB
	IV YR	HCI	POM	CNS	HM	HCI	SPM	CNS
WEDNESDAY	II YR	DM	DPSD	OOPS LAB		OOPS	DS	CE
	III YR	TOC	OOAD	APC	CN	CN LAB		CN LAB
	IV YR	SECURITY LAB		HM	SPM	CC	CNS	POM
THURSDAY	II YR	DS	CE	DM	DM	OOPS	OOPS	CE
	III YR	TOC	TOC	ANT	ANT	OOAD	APC	MPMC
	IV YR	CC	POM	SPM	CNS	HM	SPM	HCI
FRIDAY	II YR	OOPS	DS	CE	DM	DPSD	DIGITAL LAB	DIGITAL LAB
	III YR	ANT	OOAD	APC	MPMC	TOC	CN LAB	CN LAB
	IV YR	SPM	CNS	HCI	HM	CC LAB		LIB



[Signature]
HOD

[Signature]
PRINCIPAL
Sri Raja Raajan College of Engineering
Amaravathipudur, Karaikudi - 626 002
Sivagangai Dist. Tamil Nadu



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

S.N O	PARTICULARS	PROPOSE D DATE	COMPLETE D DATE	TEACHIN G AIDS
UNIT I AUTOMATA FUNDAMENTALS				
1	Introduction to formal proof	06.09.2021	08.09.2021	BB
2	Additional forms of Proof	07.09.2021	09.09.2021	BB
3	Inductive Proofs	08.09.2021	10.09.2021	BB
4	Finite Automata	09.09.2021	13.09.2021	BB
5	DeterministicFiniteAutomata	10.09.2021	14.09.2021	BB
6	DeterministicFiniteAutomata problems	13.09.2021	15.09.2021	BB
7	Non-deterministicFiniteAutomata	14.09.2021	16.09.2021	BB
8	Non-deterministicFiniteAutomata problems	15.09.2021	20.09.2021	BB
9	FiniteAutomatawithEpsilonTransition s	16.09.2021	22.09.2021	BB
UNIT II REGULAR EXPRESSIONS AND LANGUAGES				
1	Regular Expressions	20.09.2021	23.09.2021	BB
2	FA and Regular Expressions	22.09.2021	24.09.2021	BB
3	Proving Languages not to be regular	23.09.2021	27.09.2021	BB
4	Regular languages properties	24.09.2021	28.09.2021	BB
5	Regular expression properties	27.09.2021	29.09.2021	BB
6	ClosureProperties	28.09.2021	30.09.2021	BB
7	ClosurePropertiesofRegularLanguage s	29.09.2021	30.09.2021	BB
8	Equivalence	30.09.2021	01.10.2021	BB
9	MinimizationofAutomata	30.09.2021	04.10.2021	BB
UNIT III CONTEXT FREE GRAMMAR AND LANGUAGES				
1	CFG – Parse Trees	01.10.2021	06.10.2021	BB
2	Ambiguity in Grammars and Languages	04.10.2021	07.10.2021	BB
3	Definition of the PushdownAutomata	06.10.2021	08.10.2021	BB
4	Languages of a Pushdown Automata	07.10.2021	11.10.2021	BB
5	Equivalence of Pushdown Automata and CFG	08.10.2021	13.10.2021	BB
6	PushdownAutomata	11.10.2021	15.10.2021	BB
7	PushdownAutomata problems	13.10.2021	18.10.2021	BB




PRINCIPAL
Sri Raaaja Raaajan College of Engg
Maravathipudur, Karaikudi - 606 001
Sivagangai Dist. Tamil Nadu

8	DeterministicPushdownAutomata	15.10.2021	20.10.2021	BB
9	DeterministicPushdownAutomata problems	18.10.2021	22.10.2021	BB

UNIT IV PROPERTIES OF CONTEXT FREE LANGUAGES

1	NormalFormsforCFG	20.10.2021	25.10.2021	BB
2	PumpingLemmaforCFL	22.10.2021	01.11.2021	BB
3	ClosurePropertiesofCFL	25.10.2021	02.11.2021	BB
4	TuringMachines	01.11.2021	11.11.2021	BB
5	TuringMachines properties	02.11.2021	15.11.2021	BB
6	TuringMachines problems	11.11.2021	16.11.2021	BB
7	ProgrammingTechniques	15.11.2021	22.11.2021	BB
8	Closure properties of TM	16.11.2021	24.11.2021	BB
9	ProgrammingTechniquesforTM	22.11.2021	07.12.2021	BB

UNIT V UNDECIDABILITY

1	Non Recursive Enumerable (RE) Language	24.11.2021	08.12.2021	BB
2	Undecidable Problem with RE	07.12.2021	13.12.2021	BB
3	UndecidableProblemsaboutTM	08.12.2021	22.12.2021	BB
4	Post'sCorrespondenceProblem Theorem	13.12.2021	29.12.2021	BB
5	TheClassP Theorem	15.12.2021	30.12.2021	BB
6	The Class NP Theorem	17.12.2021	31.12.2021	BB
7	TheClassP problems	20.12.2021	02.01.2022	BB
8	The Class NP problems	22.12.2021	03.01.2022	BB
9	Post'sCorrespondenceProblem	29.12.2021	04.01.2022	BB



[Signature]
PRINCIPAL

Sri Raaja Raajan College of Engg. & Tech.
Amaravathipudur, Karaikudi - 630 30
Sivagangai Dist. Tamil Nadu



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

S.NO	PARTICULARS	PROPOSED DATE	COMPLETED DATE	TEACHING AIDS
UNIT I AUTOMATA FUNDAMENTALS				
1	Introduction to formal proof	06.09.2021	08.09.2021	BB
2	Additional forms of Proof	07.09.2021	09.09.2021	BB
3	Inductive Proofs	08.09.2021	10.09.2021	BB
4	Finite Automata	09.09.2021	13.09.2021	BB
5	DeterministicFiniteAutomata	10.09.2021	14.09.2021	BB
6	DeterministicFiniteAutomata problems	13.09.2021	15.09.2021	BB
7	Non-deterministicFiniteAutomata	14.09.2021	16.09.2021	BB
8	Non-deterministicFiniteAutomata problems	15.09.2021	20.09.2021	BB
9	FiniteAutomatawithEpsilonTransitions	16.09.2021	22.09.2021	BB
UNIT II REGULAR EXPRESSIONS AND LANGUAGES				
1	Regular Expressions	20.09.2021	23.09.2021	BB
2	FA and Regular Expressions	22.09.2021	24.09.2021	BB
3	Proving Languages not to be regular	23.09.2021	27.09.2021	BB
4	Regular languages properties	24.09.2021	28.09.2021	BB
5	Regular expression properties	27.09.2021	29.09.2021	BB
6	ClosureProperties	28.09.2021	30.09.2021	BB
7	ClosurePropertiesofRegularLanguages	29.09.2021	30.09.2021	BB
8	Equivalence	30.09.2021	01.10.2021	BB
9	MinimizationofAutomata	30.09.2021	04.10.2021	BB
UNIT III CONTEXT FREE GRAMMAR AND LANGUAGES				
1	CFG – Parse Trees	01.10.2021	06.10.2021	BB
2	Ambiguity in Grammars and Languages	04.10.2021	07.10.2021	BB
3	Definition of the PushdownAutomata	06.10.2021	08.10.2021	BB
4	Languages of a Pushdown Automata	07.10.2021	11.10.2021	BB
5	Equivalence of Pushdown Automata and CFG	08.10.2021	13.10.2021	BB
6	PushdownAutomata	11.10.2021	15.10.2021	BB
7	PushdownAutomata problems	13.10.2021	18.10.2021	BB
8	DeterministicPushdownAutomata	15.10.2021	20.10.2021	BB
9	DeterministicPushdownAutomata problems	18.10.2021	22.10.2021	BB
UNIT IV PROPERTIES OF CONTEXT FREE LANGUAGES				



PRINCIPAL
Sri Raaja Raajan College of Engineering
Amaravathipuram, Karaikudi - 606 005
Civagangai Dist. Tamil Nadu

1	NormalFormsforCFG	20.10.2021	25.10.2021	BB
2	PumpingLemmaforCFL	22.10.2021	01.11.2021	BB
3	ClosurePropertiesofCFL	25.10.2021	02.11.2021	BB
4	TuringMachines	01.11.2021	11.11.2021	BB
5	TuringMachines properties	02.11.2021	15.11.2021	BB
6	TuringMachines problems	11.11.2021	16.11.2021	BB
7	ProgrammingTechniques	15.11.2021	22.11.2021	BB
8	Closure properties of TM	16.11.2021	24.11.2021	BB
9	ProgrammingTechniquesforTM	22.11.2021	07.12.2021	BB

UNIT V UNDECIDABILITY

1	Non Recursive Enumerable (RE) Language	24.11.2021	08.12.2021	BB
2	Undecidable Problem with RE	07.12.2021	13.12.2021	BB
3	UndecidableProblemsaboutTM	08.12.2021	22.12.2021	BB
4	Post'sCorrespondenceProblem Theorem	13.12.2021	29.12.2021	BB
5	TheClassP Theorem	15.12.2021	30.12.2021	BB
6	The Class NP Theorem	17.12.2021	31.12.2021	BB
7	TheClassP problems	20.12.2021	02.01.2022	BB
8	The Class NP problems	22.12.2021	03.01.2022	BB
9	Post'sCorrespondenceProblem	29.12.2021	04.01.2022	BB




PRINCIPAL
 Sri Raaja Raajan College of Engineering
 Amaravathipudur, Karaikudi
 Sivagangai Dist. Tamil Nadu

UNIT I- AUTOMATA FUNDAMENTALS

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. The field is divided into three major branches: automata theory, computability theory and computational complexity theory.

In order to perform a rigorous study of computation, computer scientists work with a mathematical abstraction of computers called a model of computation. There are several models in use, but the most commonly examined is the Turing machine.

Automata theory

In theoretical computer science, **automata theory** is the study of abstract machines (or more appropriately, abstract 'mathematical' machines or systems) and the computational problems that can be solved using these machines. These abstract machines are called automata.

This automaton consists of

- **states** (represented in the figure by circles),

As the automaton sees a symbol of input, it makes a *transition* (or *jump*) to another state, according to its **transition function** (which takes the current state and the recent symbol as its inputs).

Uses of Automata: compiler design and parsing.

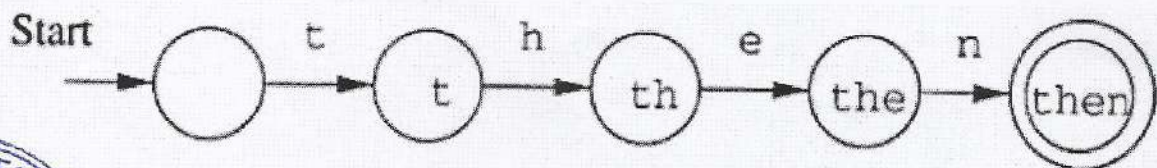


Figure 1.2: A finite automaton modeling recognition of then

Additive inverse: $a + (-a) = 0$

Multiplicative inverse: $a * 1/a = 1$

Universal set $U = \{1, 2, 3, 4, 5\}$




PRINCIPAL
Sri Raaja Raajan College of Engineering
Maravathipudur, Karaikudi
Sivagangai Dist. Tamil Nadu

Subset $A = \{1, 3\}$

$A' = \{2, 4, 5\}$

Absorption law: $A \cup (A \cap B) = A$, $A \cap (A \cup B) = A$

De Morgan's Law:

$(A \cup B)' = A' \cap B'$ $(A \cap B)' = A' \cup B'$ Double compliment

$(A')' = A$

$A \cap A' = \Phi$

Logic relations: $a \cap b = \neg(a \cup b)$ $\neg(a \cap b) = \neg a \cup \neg b$

Relations:

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

Reflexive: $a = a$

Symmetric: $aRb \Rightarrow bRa$

Transition: $aRb, bRc \Rightarrow aRc$

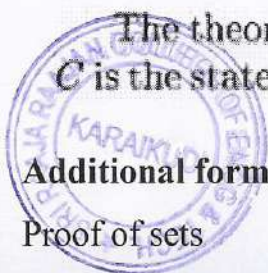
If a given relation is reflexive, symmetric and transitive then the relation is called equivalence relation.

Deductive proof: Consists of sequence of statements whose truth lead us from some initial

The theorem that is proved when we go from a hypothesis H to a conclusion C is the statement "if H then C ." We say that C is *deduced* from H .

Additional forms of proof:

Proof of sets




PRINCIPAL
Sri Raaja Raajan College of
Education, Karaikudi,
Sivagangai Dist. Tamil Nadu

Proof by contradiction

Proof by counter example

Direct proof (AKA) 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 $2n+1$

$$a=2x+1, b=2y+1$$

$$\text{product of } a \times b = (2x+1) \times (2y+1)$$

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

Proof by contrapositive:

The contrapositive of the statement "if H and C " is "if not C then not H ." A statement and its contrapositive are either both true or both false, so we can prove either to prove the other.

Theorem 1.10: $R \cup (S \cap T) = (R \cup S) \cap (R \cup T)$.

	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

Figure 1.5: Steps in the "if" part of Theorem 1.10




PRINCIPAL
Sri Raaja Raajan College of Engg. & Tech
Amaravathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

	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

Figure 1.6: Steps in the “only-if” part of Theorem 1.10

Figure 1.6: Steps in the “only-if” part of Theorem 1.10

To see why “If H then C ” and “If not C then not H ” are logically equivalent, first observe that there are four cases to consider:

1. H and C both true
2. H true and C false
3. C true and H false
4. H and C both false

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 rather than a

Alleged Theorem : All primes are odd. (More formally, we might say: if integer x is a prime, then x is odd.)



PRINCIPAL
Sri Raaja Raajan College,
Amaravathipuram, Karaikal - 605 004
Sivagangai Dist. Tamil Nadu

DISPROOF: The integer 2 is a prime, but 2 is even.

For any sets a, b, c if $a \cap b = \Phi$ and c is a subset of b the prove that $a \cap c$

$= \Phi$
Assume: $a \cap c \neq \Phi$

Then $a \cap c \neq \Phi \Rightarrow a \cap b \neq \Phi$

$\Rightarrow a \cap b = \Phi \Rightarrow a \cap c = \Phi$ (i.e., the assumption is

Proof by mathematical Induction:

Suppose we are given a statement $S(n)$, about an integer n , to prove. One common approach is to prove two things:

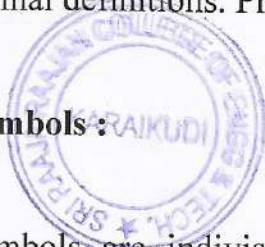
1. The *basis*, where we show $S(i)$ for a particular integer i . Usually, $i = 0$ or $i = 1$, but there are examples where we want to start at some higher i , perhaps because the statement S is false for a few small integers.
 2. The *inductive step*, where we assume $n \geq i$, where i is the basis integer and we show that "if $S(n)$ then $S(n + 1)$."
- *The Induction Principle:* If we prove $S(i)$ and we prove that for all $n \geq i$, $S(n)$ implies $S(n + 1)$, then we may conclude $S(n)$ for all $n \geq i$.

Languages :

The languages we consider for our discussion is an abstraction of natural languages. That is, our focus here is on formal languages that need precise and formal definitions. Programming languages belong to this category.

Symbols :

Symbols are indivisible objects or entity that cannot be defined. That is, symbols are atoms of the world of languages. A symbol is any single object such as \uparrow , a , 0 , 1 , $\#$, *begin*, or *do*. Usually, characters from a typical keyboard are only used as symbols.



PRINCIPAL
Sri Raaja Raajan College of Engineering
Amaravathipudur, Karaikudi - 630 001
Sivagangai Dist, Tamil Nadu

Alphabets :

An alphabet is a finite, nonempty set of symbols. The alphabet of a language is denoted by Σ . When more than one alphabets are considered for discussion, subscripts may be used (e.g. Σ_1 , Σ_2 etc) or sometimes other symbol like G may also be introduced.

$$\Sigma = \{0, 1\}$$

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

$$\Sigma = \{a, b, c, \&, z\}$$

$$\Sigma = \{\#, \nabla, \spadesuit, \beta\}$$

Example :

Strings or Words over Alphabet :

A string or word over an alphabet Σ is a finite sequence of concatenated symbols Σ .

Example : 0110, 11, 001 are three strings over the binary alphabet $\{0, 1\}$

aab, abcb, b, cc are four strings over the alphabet $\{a, b, c\}$

It is not the case that a string over some alphabet should contain all the symbols from the alphabet. For example, the string *cc* over the alphabet $\{a, b, c\}$ does not contain the symbols *a* and *b*. Hence, it is true that a string over an alphabet is also a string over an alphabet is also a string over any superset of that alphabet.

Length of a string :

The number of symbols in a string w is called its length, denoted by $|w|$

Example : $|011| = 4$, $|11| = 2$, $|b| = 1$




PRINCIPAL
Sri Raaja Raajan College
Amaravathipudur, Kar
Sivagangai Dist. Tamil Nadu

Convention : We will use small case letters towards the beginning of the English alphabet to denote symbols of an alphabet and small case letters towards the end denote strings over an alphabet. That $a, b, c, \in \Sigma$ (symbols) and u, v, w, x, y, z are strings.

Some String Operations :

Let $x = a_1a_2a_3 \in a_n$ and $y = b_1b_2b_3 \in b_m$ be two strings. The concatenation of string $a_1a_2a_3 \dots a_nb_1b_2b_3 \dots b_m$. That is, the concatenation of x and y denoted by xy is x followed by a copy of y without any intervening space between them.

Example : Consider the string 011 over the binary alphabet. All the prefixes, suffixes

Prefixes: $\epsilon, 0, 01, 011$. Suffixes:

$\epsilon, 1, 11, 011$. Substrings: $\epsilon, 0, 1, 01, 11, 011$.

Note that x is a prefix (suffix or substring) to x , for any string x and ϵ is a prefix (suffix or substring) to any string.

A string x is a proper prefix (suffix) of string y if x is a prefix (suffix) of y and $x \neq y$.

In the above example, all prefixes except 011 are proper

Powers of Strings : For any string x and $n \geq 0$, we use x^n to denote the string formed by sequentially concatenating n copies of x . We can also give an




PRINCIPAL
 Sri Raaja Raajan College
 Amaravathipudur, Karaik.
 Sivagangai Dist. Tamil Nadu

definition of x^n as follows:

$$x^n = e, \text{ if } n = 0; \text{ otherwise } x^n = xx^{n-1}$$

Example : If $x = 011$, then $x^3 = 011011011$, $x^1 = 011$ and $x^0 = e$

Powers of Alphabets :

We write Σ^k (for some integer k) to denote the set of strings of length k with symbols from Σ . In other words,

$\Sigma^k = \{ w \mid w \text{ is a string over } \Sigma \text{ and } |w| = k \}$. Hence, for any alphabet, Σ^0 denotes the set of all strings of length zero. That $\Sigma^0 = \{ e \}$. For the binary alphabet $\{ 0, 1 \}$ we is,

$$\Sigma^0 = \{ e \}$$

$$\Sigma^1 = \{ 0, 1 \}$$

$$\Sigma^2 = \{ 00, 01, 10, 11 \}$$

$$\Sigma^3 = \{ 000, 001, 010, 011, 100, 101, 110, 111 \}$$

The set of all strings over an alphabet Σ is denoted by Σ^* . That is,

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots \cup \Sigma^x \cup \dots$$

$$= \bigcup \Sigma^k$$

The set Σ^* contains all the strings that can be generated by iteratively symbols from any number of times.

Example : If $\Sigma = \{ a, b \}$, then $\Sigma^* = \{ e, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, \dots \}$.




PRINCIPAL
Sri Raja Raajan College of Engineering
Aravathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

Please note that if $\Sigma = \emptyset$, then Σ^* that is $\emptyset^* = \{e\}$. It may look odd that one can proceed from the empty set to a non-empty set by iterated concatenation. But there a

The set of all nonempty strings over an Σ is denoted by Σ^+ . That is,

$$\begin{aligned}\Sigma^+ &= \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots \cup \Sigma^k \cup \dots \\ &= \bigcup \Sigma^k\end{aligned}$$

Note that Σ^* is infinite. It contains no infinite strings but strings of arbitrary

Reversal :

For any string $w = a_1 a_2 a_3 \dots a_n$ the reversal of the string is $w^R = a_n a_{n-1} \dots a_3 a_2 a_1$.

An inductive definition of reversal can be given as

Languages :

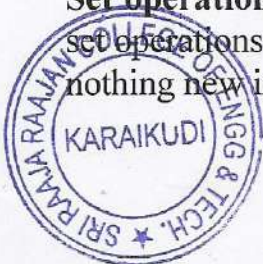
A language over an alphabet is a set of strings over that alphabet. Therefore, language L is any subset of Σ^* . That is, any $L \subseteq \Sigma^*$


Example :

1. \emptyset is the empty language.
2. Σ^* is a language for any Σ .
3. $\{e\}$ is a language for any Σ . Note that, $\emptyset \neq \{e\}$. Because the language \emptyset does not contain any string but $\{e\}$ contains one string of length zero.
4. The set of all strings over $\{0, 1\}$ containing equal number of 0's and 1's.
5. The set of all strings over $\{a, b, c\}$ that starts with a .

Convention : Capital letters A, B, C, L , etc. with or without subscripts are normally used

Set operations on languages : Since languages are set of strings we can apply set operations to languages. Here are some simple examples (though there is nothing new in it).




PRINCIPAL
Sri Raaja Raajan College of Engineering & Technology
Amaravathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

Union : A string $x \in L_1 \cup L_2$ iff $x \in L_1$ or $x \in L_2$

Example : $\{0, 11, 01, 011\} \cup \{1, 01, 110\} = \{0, 11, 01, 011, 111\}$

Intersection : A string, $x \in L_1 \cap L_2$ iff $x \in L_1$ and $x \in L_2$

Example : $\{0, 11, 01, 011\} \cap \{1, 01, 110\} = \{01\}$

Complement : Σ^* is the universe that a complement is taken with respect to. Thus for a language L , the complement is $L(\text{bar}) = \{x \in \Sigma^* \mid x \notin L\}$.

Example : Let $L = \{x \mid |x| \text{ is even}\}$. Then its complement is the language $\{x \in \Sigma^* \mid |x| \text{ is odd}\}$.

Similarly we can define other usual set operations on languages like relative complement, symmetric difference, etc.

Reversal of a language :

The reversal of a language L , denoted as L^R , is defined as: $L^R = \{w^R \mid w \in L\}$.

Example :

1. Let $L = \{0, 11, 01, 011\}$. Then $L^R = \{0, 11, 10, 110\}$.




HOD


PRINCIPAL

PRINCIPAL

Sri Raja Raajan College of Engineering & Technology
Amaravathipudur, Karaikudi - 606 001
Sivagangai Dist. Tamil Nadu

UNIT II - REGULAR EXPRESSIONS AND LANGUAGES

Regular Expressions: Formal

We construct REs from primitive constituents (basic elements) by repeatedly applying certain recursive rules as given below. (In the definition)

Definition: Let S be an alphabet. The regular expressions are defined recursively as

Basis:

- i) ϵ is a RE
- ii) a is a RE
- iii) $\forall a \in S, a$ is RE.

These are called primitive regular expression i.e. Primitive

Recursive Step :

If r_1 and r_2 are REs over, then so are

i) $r_1 + r_2$

ii) $r_1 r_2$

iii) r_1^*



Handwritten signature in green ink.

Closure : r is RE over only if it can be obtained from the basis elements (Primitive REs) by finite no. of applications of the recursive step (given in 2).

Sri Raaja Raajan College of Engineering & Technology
Amaravathipudur, Karaikudi
Sivagangai Dist. Tamil Nadu

Example : Let $\Sigma = \{ 0,1,2 \}$. Then $(0+21)^*(1+2)$ is a RE, because we can construct this expression applying the above rules as given in the following step.

Steps	RE Constructed	Rule Used
1	1	Rule 1(iii)
2	\emptyset	Rule 1(i)
3	$1+\emptyset$	Rule 2(i) & Results of Step 1, 2
4	$(1+\emptyset)$	Rule 2(iv) & Step 3
5	2	1(iii)
6	1	1(iii)
7	21	2(ii), 5, 6
8	0	1(iii)
9	0+21	2(i), 7, 8
10	$(0+21)$	2(iv), 9
11	$(0+21)^*$	2(iii), 10
12	$(0+21)^*$	2(ii), 4, 11

Language described by REs : Each describes a language (or a language is associated with every RE).

Notation : If r is a RE over some alphabet then $L(r)$ is the language associate with r . We can define the



PRINCIPAL

Sri Raaja Raajan College of E:
Amaravathipudur, Karaikudi -
Sivagangai Dist. Tamil Nadu.

1. ϕ is the RE describing the empty language i.e. $L(\phi) = \phi$.
2. ϵ is a RE describing the language $\{ \epsilon \}$ i.e. $L(\epsilon) = \{ \epsilon \}$.
3. $\forall a \in S$, a is a RE denoting the language $\{a\}$ i.e. $L(a) = \{a\}$.
4. If r_1 and r_2 are REs denoting language $L(r_1)$ and $L(r_2)$ respectively, then
 - i) $r_1 + r_2$ is a regular expression denoting the language $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
 - ii) $r_1 r_2$ is a regular expression denoting the language $L(r_1 r_2) = L(r_1) L(r_2)$
 - iii) r_1^* is a regular expression denoting the language $L(r_1^*) = (L(r_1))^*$
 - iv) (r_1) is a regular expression denoting the language $L((r_1)) = L(r_1)$

Example : Consider the RE $(0^*(0+1))$. Thus the language denoted by the RE is

$$L(0^*(0+1)) = L(0^*) L(0+1) \dots \text{by 4(ii)}$$

$$= L(0)^* L(0) \cup L(1)$$

$$= \{ \epsilon, 0, 00, 000, \dots \} \{0\} \cup \{1\}$$

$$= \{ \epsilon, 0, 00, 000, \dots \} \{0, 1\}$$

$$= \{ \epsilon, 0, 00, 000, 0000, \dots, 1, 01, 001, 0001, \dots \}$$

Precedence Rule

Consider the RE $ab + c$. The language described by the RE can be thought of either $L(a)L(b+c)$ or $L(ab) \cup L(c)$ as provided by the rules (of languages described by REs) given already. But this represents two different languages leading to ambiguity. To remove this ambiguity we can either

Amf



- 1) Use fully parenthesized expression- (cumbersome) or
- 2) Use a set of precedence rules to evaluate the options of REs in some order. Like other algebras mod in mathematics.

For REs, the order of precedence for the operators is as follows:

- i) The star operator precedes concatenation and concatenation precedes union (+) operator.
- ii) It is also important to note that concatenation & union (+) operators are associative and union operation is commutative.

Using these precedence rule, we find that the RE $ab+c$ represents the language $L(ab) \cup L(c)$ i.e. it should be grouped as $((ab)+c)$.

We can, of course change the order of precedence by using parentheses. For example, the language represented by the RE $a(b+c)$ is $L(a)L(b+c)$.

Example : The RE ab^*+b is grouped as $((a(b^*))+b)$ which describes the language $L(a)(L(b))^* \cup L(b)$

Example : The RE $(ab)^*+b$ represents the language $(L(a)L(b))^* \cup L(b)$.

Example : It is easy to see that the RE $(0+1)^*(0+11)$ represents the language of all strings over $\{0,1\}$ which are either ended with 0 or 11.

Example : The regular expression $r=(00)^*(11)^*1$ denotes the set of all strings with an even number of 0's followed by an odd number of 1's i.e.

Note: The notation r^+ is used to represent the RE rr^* . Similarly, r^2 represents the RE rr , r^3 denotes rrr , and so on.



PRINCIPAL
Sri Raaja Raajan College of Engg.
Amaravathipudur, Karaikal
Sivagangai Dist. Tamil Nadu

An arbitrary string over $\Sigma = \{0,1\}$ is denoted as $(0+1)^*$.

Exercise : Give a RE r over $\{0,1\}$ s.t.

$$L(r) = \{ \omega \in \Sigma^* \mid \omega$$

has at least one pair of consecutive 1's}

Solution : Every string in $L(r)$ must contain 00 somewhere, but what comes before and what goes after is completely arbitrary. Considering these observations we can write the REs as $(0+1)^*11(0+1)^*$.

Example : Considering the above example it becomes clear that the RE $(0+1)^*11(0+1)^* + (0+1)^*00(0+1)^*$ represents the set of strings over $\{0,1\}$ that contains the substring 11 or 00.

Example : Consider the RE $0^*10^*10^*$. It is not difficult to see that this RE describes the set of strings over $\{0,1\}$ that contains exactly two 1's. The presence of two 1's in the RE and any no of 0's before, between and after .

Example : Consider the language of strings over $\{0,1\}$ containing two or more

Solution : There must be at least two 1's in the RE somewhere and what comes before, between, and after is completely arbitrary. Hence we can write the RE as $(0+1)^*1(0+1)^*1(0+1)^*$. But following two REs also represent the same language, each ensuring presence of least two 1's somewhere in the string

i) $0^*10^*1(0+1)^*$

ii) $(0+1)^*10^*10^*$

Example : Consider a RE r over $\{0,1\}$ such that



PRINCIPAL
Sri Raaja Raajan College of Engineering
Amaravathipuram, Karaikudi
Sivagangai Dist. Tamil Nadu

$$L(r) = \{ \omega \in \{0,1\}^* \mid \omega$$

has no pair of consecutive 1's}

Solution : Though it looks similar to ex, it is harder to construct to construct. We observe that, whenever

a 1 occurs, it must be immediately followed by a 0. This substring may be preceded & followed by any no of 0's. So the final RE must be a repetition of strings of the form: 00...0100....00 i.e. 0^*100^* . So it looks like the these observations into consideration, the final RE is $r = (0^*100^*)(1 + \square)^+ + 0^*(1 + \square)^+$.

Alternative Solution :

The language can be viewed as repetitions of the strings 0 and 01. Hence get the RE as $r = (0+10)^*(1 + \square)$. This is a shorter expression but represents the same language.

Regular Expression:

FA to regular expressions:

FA to RE (REs for Regular Languages)



Lemma: If a language is regular, then there is a RE to describe it. i.e. if $L = L(M)$ for some DFA M, then there is a RE r such that $L = L(r)$.

Proof : We need to construct a RE r such $L(r) = \{w \mid w \in L(M)\}$. Since M is a DFA, it has a finite no of states. Let the set of states of M is $Q = \{1, 2, 3, \dots, n\}$ for some integer n. [Note : if the n states of M were denoted by some other symbols, we can always rename those to indicate as 1, 2, 3, ..., n].

ang

PRINCIPAL

Sri Raaja Raajan College of Engineering
Amaravathipudur, Karaikal
Sivagangai Dist. Tamil Nadu

Notations : $r_{ij}^{(k)}$ is a RE denoting the language which is the set of all strings w such that w is the label of path from state i to state j ($1 \leq i, j \leq n$) in M , and that path has no intermediate state whose number is greater than k . (i & j (begining and end pts) are not considered to be "intermediate" so i and j can be greater than k)

We now construct $r_{ij}^{(k)}$ inductively, for all $i, j \in Q$ starting at $k = 0$ and finally reaching $k = n$.

Basis : $k = 0$, $r_{ij}^{(0)}$ i.e. the paths must not have any intermediate state (since all states are numbered above). There are only two possible paths meeting the above condition :

1. A direct transition from state i to state j .

- $r_{ij}^{(0)} = a$ if there is a transition from state i to state j on symbol the single symbol a
- $r_{ij}^{(0)} = a_1 + a_2 + \dots + a_m$ if there are multiple transitions from state i to state j on symbols a_1, a_2, \dots, a_m .
- $r_{ij}^{(0)} = \emptyset$ if there is no transition at all from state i to state j .

2. All paths consisting of only one node i.e. when $i = j$. This gives the path of length 0 (i.e. the empty string ϵ) and all self loops. By simply adding ϵ to various cases above we get the corresponding REs i.e.

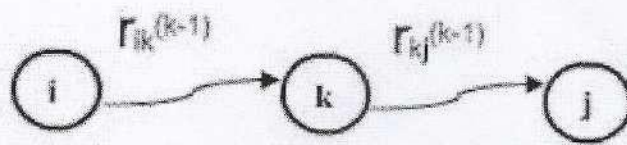
- $r_{ii}^{(0)} = \epsilon + a$ if there is a self loop on symbol a in state i
- $r_{ii}^{(0)} = \epsilon + a_1 + a_2 + \dots + a_m$ if there are self loops in state i as multiple symbols a_1, a_2, \dots, a_m .
- $r_{ii}^{(0)} = \epsilon$ if there is no self loop on state i

Induction :

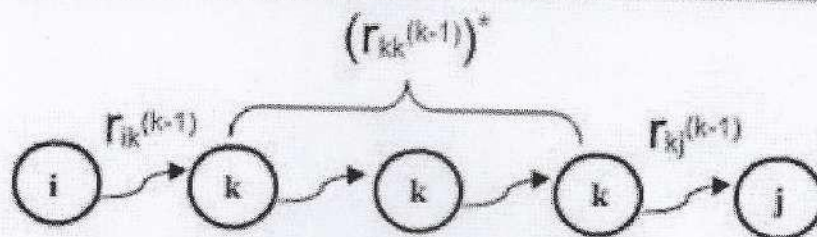
Assume that there exists a path from state i to state j such that there is no intermediate state whose number greater than k . The corresponding RE for the label of the path is $r_{ij}^{(k)}$. There are only two possible cases :

1. The path does not go through the state k at all i.e. number of all the intermediate states are less than k . So, the label of the path from state i to state j is the language described by the $r_{ij}^{(k-1)}$.
2. The path goes through the state k at least once. The path may go from i to j and k may appear more than once. We can break it into pieces as shown in the figure 7.

PRINCIPAL
 Sri Raaja Raajan College of Engineering
 Amaravathipuram, Karaikudi - 606 006
 Sivagangai Dist. Tamil Nadu



A path from i to j that goes through k exactly once



A path from i to j that goes through k more than once

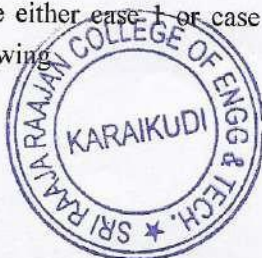
Figure 7

1. The first part from the state i to the state k which is the first recurrence. In this path, all intermediate states are less than k and it starts at i and ends at k. So the RE $r_{ik}^{(k-1)}$ denotes the language of the label of path.
2. The last part from the last occurrence of the state k in the path to state j. In this path also, no intermediate state is numbered greater than k. Hence the RE $r_{kj}^{(k-1)}$ denoting the language of the label of the path.
3. In the middle, for the first occurrence of k to the last occurrence of k, represents a loop which may be taken zero times, once or any no of times. And all states between two consecutive k's are numbered less than k.

Hence the label of the path of the part is denoted by the RE $r_{ij}^{(k-1)*}$. The label of the path from state i to state j is the concatenation of these 3 parts which is

$$r_{ik}^{(k-1)} (r_{kk}^{(k-1)})^* r_{kj}^{(k-1)}$$

Since either case 1 or case 2 may happen the labels of all paths from state i to j is denoted by the following



Handwritten signature

PRINCIPAL

Sri Raaja Raajan College of Engineering & Technology
Maravathipudur, Karaikudi - 606 001
Sivagangai Dist. Tamil Nadu

$$r_{ij}^{(k)} = r_{ij}^{(k-1)} + r_{ik}^{(k-1)} \left(r_{kk}^{(k-1)} \right)^* r_{kj}^{(k-1)}$$

We can construct $r_{ij}^{(k)}$ for all $i, j \in \{1, 2, \dots, n\}$ in increasing order of k starting with the basis $k=1$ since $r_{ij}^{(k)}$ depends only on expressions with a small superscript (and hence will be available). that state 1 is the start state and j_1, j_2, \dots, j_m are the m final states where $j_i \in \{1, 2, \dots, n\}$, $m \leq n$. According to the convention used, the language of the automata can be denoted by the

$$r_{1j_1}^{(n)} + r_{1j_2}^{(n)} + \dots + r_{1j_m}^{(n)}$$

Since $r_{ij}^{(n)}$ is the set of all strings that starts at start state 1 and finishes at final j following the transition of the FA with any value of the intermediate state $(1, 2, \dots, n)$ and hence accepted by the automata.

Regular Grammar:

A grammar $G = (N, \Sigma, P, S)$ is right-linear if each production has one of the following

- $A \rightarrow cB$,
- $A \rightarrow c$,
- $A \rightarrow \epsilon$

Where $A, B \in N'$ (with $A = B$ allowed) and $c \in \Sigma$. A grammar G is left-linear if each production has one of the following three forms.

$$A \rightarrow Bc, A \rightarrow c, A \rightarrow \epsilon$$

A right or left linear grammar is called a regular grammar.

Regular grammar and Finite Automata are equivalent as stated in the following

Theorem : A language L is regular iff it has a regular grammar. We use the following two lemmas to prove the above theorem.



PRINCIPAL
Sri Raaja Raajan College of Engg. & Tech.
Amaravathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

UNIT III – CONTEXT FREE GRAMMAR AND LANGUAGES

PUSHDOWN AUTOMATA

Grammar

A grammar is a mechanism used for describing languages. This is one of the most simple but yet powerful mechanism. There are other notions to do the same, of course.

In everyday language, like English, we have a set of symbols (alphabet), a set of words constructed from these symbols, and a set of rules using which we can group the words to construct meaningful sentences. The grammar for English tells us what are the words in it and the rules to construct sentences. It also tells us whether a particular sentence is well-formed (as per the grammar) or not. But even if one follows the rules of the english grammar it may lead to some sentences which are not meaningful at all, because of impreciseness and ambiguities involved in the language. In english grammar we use many other higher level

<sentence> -- >< noun-phrase >< predicate >

meaning that "a sentence can be constructed using a 'noun-phrase' followed by a predicate".

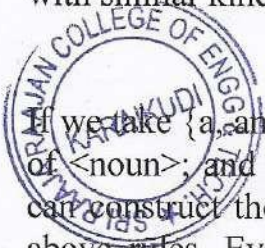
Some more rules are as follows:

< noun-phrase > -- > < article >< noun >

<predicate> -- > < verb >

with similar kind of interpretation given above.

If we take {a, an, the} to be <article>; cow, bird, boy, Ram, pen to be examples of <noun>; and eats, runs, swims, walks, are associated with <verb>, then we can construct the sentence- a cow runs, the boy eats, an pen walks- using the above rules. Even though all sentences are well-formed, the last one is not meaningful. We observe that we start with the higher level construct <sentence> and then reduce it to <noun-phrase>.



PRINCIPAL
Sri Raaja Raajan College of
Amaravathipudur, Karaiikal
Sivagangai Dist. Tamil Nadu

<article>, <noun>, <verb> successively, eventually leading to a group of words associated with these

These concepts are generalized in formal language leading to formal grammars. The word 'formal' here refers to the fact that the specified rules for the language are explicitly stated in terms of what strings or symbols can occur. There can be no ambiguity in it.

Formal definitions of a Grammar

A grammar G is defined as a quadruple.

$$G = (N, \Sigma, P, S)$$

N is a non-empty finite set of non-terminals or variables,

Σ is a non-empty finite set of terminal symbols such that $N \cap \Sigma = \phi$

S is a special non-terminal (or variable) called the start symbol, and P is a finite set of production rules.

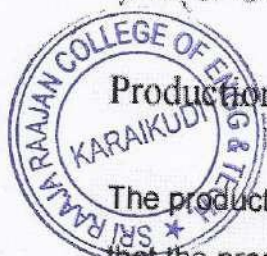
The binary relation defined by the set of production rules is denoted by \rightarrow , i.e. $\alpha \rightarrow \beta$ iff

In other words, P is a finite set of production rules of the form $\alpha \rightarrow \beta$, where $\alpha \in (N \cup \Sigma)^*$ and $\beta \in (N \cup \Sigma)^*$

Production rules:

The production rules specify how the grammar transforms one string to another. Given a string α that the production rule $\alpha \rightarrow \beta$ is applicable to this string, since it is possible to use the rule $\alpha \rightarrow \beta$ (in $\delta\alpha\gamma$) to β obtaining a new string $\delta\beta\gamma$. We say that $\delta\alpha\gamma$ derives $\delta\beta\gamma$ and is denoted by

$$\delta\alpha\gamma \Rightarrow \delta\beta\gamma$$



Handwritten signature in green ink.

PRINCIPAL
Sri Raaja Raajan College of Engg. & Tech
Amaravathipuram, Karaikudi - 630 301
Kanyakumari Dist. Tamil Nadu

Successive strings are derived by applying the productions rules of the grammar in any arbitrary order.

By applying the production rules in arbitrary order, any given grammar can generate many strings of terminal symbols starting with the special start symbol, S , of the grammar.

A

We write $\alpha \xRightarrow{*} \beta$ if the string β can be derived from the string α in zero or more steps; $\alpha \Rightarrow \beta$ if β can be derived from α in one or more steps.

By applying the production rules in arbitrary order, any given grammar can generate many strings starting with the special start symbol, S , of the grammar. The set of all such terminal strings is called the language generated by G .

Formally, for a given grammar $G = (N, \Sigma, P, S)$ the language generated by G is

$$L(G) = \{w \in \Sigma^* \mid S \xRightarrow{*} w\}$$

That is $w \in L(G)$ iff $S \xRightarrow{*} w$.



PRINCIPAL

Sri Raaja Raajan College of
Amaravathipuram, Karaikudi
Sivagangai Dist. Tamil Nadu

If $w \in L(G)$, we must have for some $n \geq 0$, $S = \alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n = w$, derivation sequence of w . The strings $S = \alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n = w$ are denoted as sentential derivation.

Example : Consider the grammar $G = (N, \Sigma, P, S)$, where $N = \{S\}$, $\Sigma = \{a, b\}$ and P is the production rules

$$\{ \rightarrow ab, S \rightarrow aSb \}$$

Some terminal strings generated by this grammar together with their derivation is given below

$$S \Rightarrow ab$$

$$S \Rightarrow aSb \Rightarrow aabb$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$$

It is easy to prove that the language generated by this grammar is

$$L(G) = \{a^i b^i \mid i \geq 1\}$$

PRINCIPAL

Shri Raja Raajan College of Engineering
Amaravathipuram, Karaikudi - 630 006
Sivagangai Dist. Tamil Nadu



If $w \in L(G)$, we must have for some $n \geq 0$, $S = \alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n = w$, den
 derivation sequence of w . The strings $S = \alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n = w$ are denoted as sentential
 derivation.

Example : Consider the grammar $G = (N, \Sigma, P, S)$, where $N = \{S\}$, $\Sigma = \{a, b\}$ and P is the
 production rules

$$\{ S \rightarrow ab, S \rightarrow aSb \}$$

Some terminal strings generated by this grammar together with their derivation is given below

$$S \Rightarrow ab$$

$$S \Rightarrow aSb \Rightarrow aabb$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbbb$$

It is easy to prove that the language generated by this grammar is

$$L(G) = \{ a^i b^i \mid i \geq 1 \}$$

By using the first production, it generates the string ab (for $i=1$).



Handwritten signature in green ink.

PRINCIPAL
 Karaja Rajan College of Engg. & Tech.
 Amaravathipudur, Karaikudi - 630 301
 Sivagangai Dist. Tamil Nadu

To generate any other string, it needs to start with the production $S \rightarrow aSb$ and then the non-terminal RHS can be replaced either by ab (in which we get the string $aabb$) or the same production used one or more times. Every time it adds an 'a' to the left and a 'b' to the right of S, thus giving form $a^i S b^i, i \geq 1$. When the non-terminal is replaced by ab (which is then only possibility for terminal string) we get a terminal string of the form $a^i b^i, i \geq 1$.

There is no general rule for finding a grammar for a given language. For many languages we have grammars and there are many languages for which we cannot find any grammar.

Example: Find a grammar for the language $L = \{a^n b^{n+1} \mid n \geq 1\}$.

It is possible to find a grammar for L by modifying the previous grammar since we need to get at the end of the string $a^n b^n, n \geq 1$. We can do this by adding a production $\rightarrow Bb$ where it generates $a^i b^i, i \geq 1$ as given in the previous example.

Using the above concept we devise the following grammar for L.

$G = (N, \Sigma, P, S)$ where, $N = \{S, B\}, P = \{S \rightarrow Bb, B \rightarrow ab, B \rightarrow aBb\}$

Parse Trees:



Handwritten signature in green ink.

PRINCIPAL
Sri Raaja Raajan College of Eng
Amaravathipudur, Karaikudi - 6
Sivagangai Dist. Tamil Nadu

There is a tree representation for derivations that has proved extremely useful. This tree shows us clearly how the symbols of a terminal string are derived into substrings, each of which belongs to the language of one of the non-terminals of the grammar. But perhaps more importantly, the tree, known as a parse tree, shows the construction of the string from the start symbol.

Construction of a Parse tree:

Let us fix on a grammar $G = (V, T, P, S)$. The *parse trees* for G are defined by the following conditions:


1. Each interior node is labeled by a variable in V .
2. Each leaf is labeled by either a variable, a terminal, or ϵ . If a leaf is labeled ϵ , then it must be the only child of its parent.
3. If an interior node is labeled A , and its children are labeled

$$X_1, X_2, \dots, X_k$$

respectively, from the left, then $A \rightarrow X_1 X_2 \dots X_k$ is a production of G . Note that the only time one of the X 's can be ϵ is if that is the only child, and $A \rightarrow \epsilon$ is a production of G .

Example 5.10: Figure 5.5 shows a parse tree for the palindrome 01010 in Fig. 5.1. The production used at the root is $P \rightarrow 0P0$, and at the middle child of the root it is $P \rightarrow 1P1$. Note that at the bottom is a use of the production $P \rightarrow \epsilon$. That use, where the node labeled by the head has one child, is the only time that a node labeled ϵ can appear in a parse tree.




 PRINCIPAL
 Raaja Raajan College of Engineering
 Amaravathipuram, Karaikudi
 Sivagangai Dist. Tamil Nadu

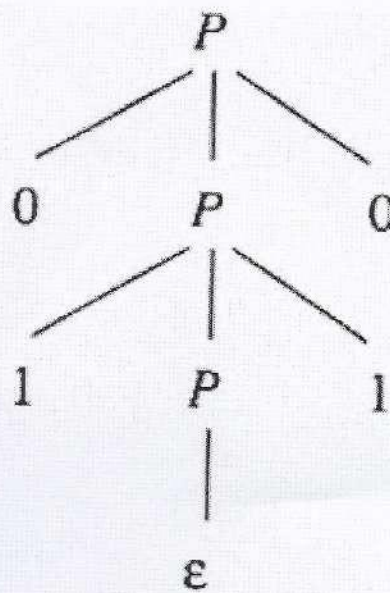


Figure 5.5: A parse tree showing the derivation $P \Rightarrow^* 0111$

Yield of a Parse tree:

If we look at the leaves of any parse tree and concatenate them from left to right, we get a string, called the *yield* of the tree, which is always a string that can be derived from the root variable. The fact that the yield is derived from the root variable can be proved shortly. Of special importance are those parse trees such that

1. The yield is a terminal string. That is, all leaves are labeled with a terminal or with ϵ .
2. The root is labeled by the start symbol.

Ambiguity in languages and grammars:



PRINCIPAL

Sri Raja Raajan College of Engineering & Technology
Maravathipudur, Karaikudi - 606 001
Sivagangai Dist. Tamil Nadu

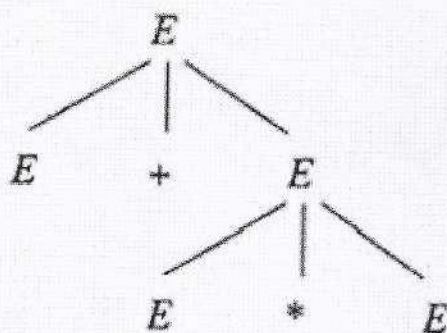
When a grammar fails to provide unique structures, it is sometimes possible to redesign the grammar to make the structure unique for each string in the language. Unfortunately, sometimes we cannot do so. That is, there are some CFL's that are "inherently ambiguous"; every grammar for the language produces more than one structure on some strings in the language. A grammar that lets us generate expressions with any sequence of $*$ and $+$ operators and the productions $E \rightarrow E + E \mid E * E$ allow us to generate these expressions in any order we choose.

Example 5.25: For instance, consider the sentential form $E + E * E$. It has two derivations from E :

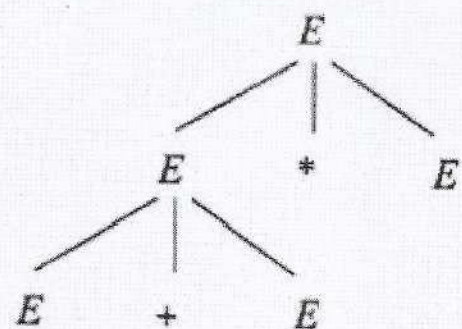
$$1. E \Rightarrow E + E \Rightarrow E + E * E$$

$$2. E \Rightarrow E * E \Rightarrow E + E * E$$

Notice that in derivation (1), the second E is replaced by $E * E$, while in derivation (2), the first E is replaced by $E + E$. Figure 5.17 shows the two parse trees, which we should note are distinct trees.



(a)



(b)

Figure 5.17: Two parse trees with the same yield. We say a CFG $G = (V, T, P, S)$ is *ambiguous* if there is at least one string w in T^* for which we can find two different parse trees, each with root labeled S and yield w . If each string has at most one parse tree in the grammar, then the grammar is *unambiguous*.

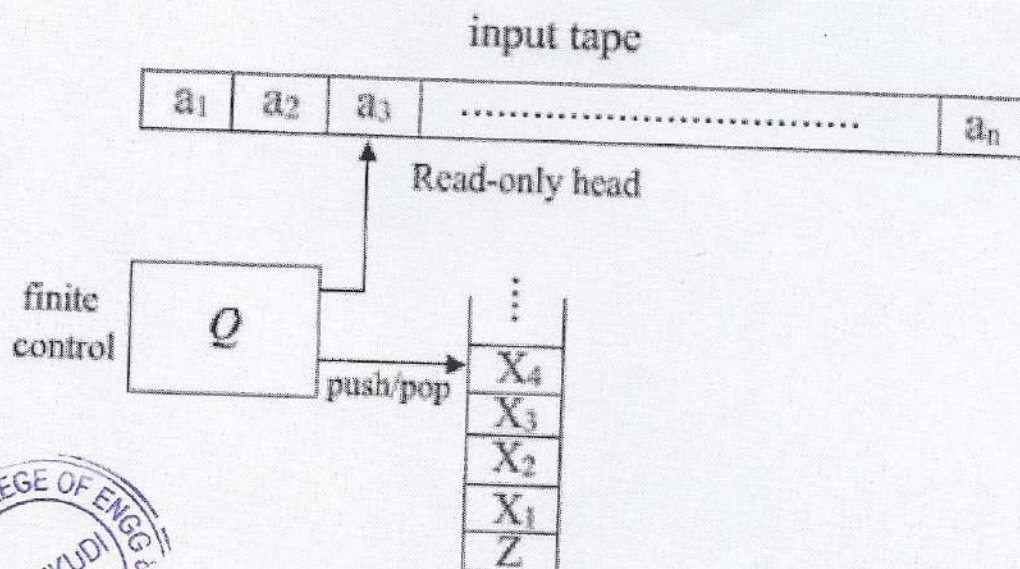
Push down automata:

Regular language can be characterized as the language accepted by finite automata. Similarly, we can characterize the context-free language as the

Principal
Sri Kaaja Raajan College of Engg.
Amaravathipuram, Karaikudi - 630 005
Sivagangai Dist. Tamil Nadu

language accepted by a class of machines called "Pushdown Automata" (PDA). A pushdown automation is an extension of the NFA.


It is observed that FA have limited capability. (in the sense that the class of languages accepted or characterized by them is small). This is due to the "finite memory" (number of states) and "no external involved with them". A PDA is simply an NFA augmented with an "external stack memory". The addition of a stack provides the PDA with a last-in, first-out memory management capability. This "Stack" or "pushdown store" can be used to record a potentially unbounded information. It is due to this memory management accept many interesting languages like $\{a^nb^n | n \geq 0\}$. Although, a PDA can store an unbounded amount of information on the stack, its access to the information on the stack is limited. It can push an element onto the top of the stack and pop off an element from the top of the stack. To read down into the stack the top elements must be popped off and are lost. Due to this limited access to the information on the stack, a PDA still has some limitations and cannot accept some other interesting languages.



As shown in figure, a PDA has three components: an input tape with read only head, a finite control and a pushdown store.

The input head is read-only and may only move from left to right, one symbol (or cell) at a time. In each step, the PDA pops the top symbol off the stack; based on this symbol, the input symbol it is currently reading, its present state, it can push a sequence of symbols onto the stack, move its read-only head one


HOD


PRINCIPAL
Bala Rajan College of Engineering
Kavathipudur, Karaikudi - 626 005

UNIT IV - PROPERTIES OF CONTEXT FREE LANGUAGES

TURING MACHINES

Empty Production Removal

The productions of context-free grammars can be coerced into a variety of forms without affecting the expressive power of the grammars. If the empty string does not belong to a language, then there is a way to eliminate the productions of the form $A \rightarrow \lambda$ from the grammar. If the empty string belongs to a language, then we can eliminate λ from all productions save for the single production $S \rightarrow \lambda$. In this case we can also eliminate any occurrences of S from the right-hand side of productions.

Procedure to find CFG with out empty Productions

Step (i): For all productions $A \rightarrow \lambda$, put A into V_N .

Step (ii): Repeat the following steps until no further variables are added to V_N
For all productions|

$$B \rightarrow A_1 A_2 \dots A_n.$$

Step (i): For all productions $A \rightarrow \lambda$, put A into V_N .

Step (ii): Repeat the following steps until no further variables are added to V_N .
For all productions|

$$B \rightarrow A_1 A_2 \dots A_n.$$

where $A_1, A_2, A_3, \dots, A_n$ are in V_N , put B into V_N .

To find P , let us consider all productions in P of the form

$$A \rightarrow x_1 x_2 \dots x_m, m \geq 1$$

for each $x_i \in V \cup T$.

Unit production removal



Conf

PRINCIPAL

Sri Raajan College of Engineering & Technology
Maravathipudur, Karaikudi - 606 005
Sivagangai Dist. Tamil Nadu

Any production of a CFG of the form

$$A \rightarrow B$$

where $A, B \in V$ is called a "Unit-production". Having variable one on either side of a production is sometimes undesirable.

"Substitution Rule" is made use of in removing the unit-productions.

Given $G = (V, T, S, P)$, a CFG with no λ -productions, there exists a CFG $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$ that does not have any unit-productions and that is equivalent to G .

Let us illustrate the procedure to remove unit-production through example 2.4.6.

Procedure to remove the unit productions:

Find all variables B , for each A such that

$$A \overset{*}{\Rightarrow} B$$

This is done by sketching a "depending graph" with an edge (C, D)

whenever the grammar has unit-production $C \rightarrow D$, then $A \overset{*}{\Rightarrow} B$ holds whenever there is a walk between A and B .

The new grammar \hat{G} , equivalent to G is obtained by letting into \hat{P} all non-unit productions of P .

Then for all A and B satisfying $A \overset{*}{\Rightarrow} B$, we add to \hat{P}

$$A \rightarrow y_1 | y_2 | \dots | y_n$$

where $B \rightarrow y_1 | y_2 | \dots | y_n$ is the set of all rules in \hat{P} with B on the left.

Left Recursion Removal



Handwritten signature in green ink.

PRINCIPAL

Raja Raajan College of E.
Amaravathipudur, Karaikudi -
Sivagangai Dist. Tamil Nadu

A variable A is left-recursive if it occurs in a production of the form

$$A \rightarrow Ax$$

for any $x \in (V \cup T)^*$.

A grammar is left-recursive if it contains at least one left-recursive variable.

Every context-free language can be represented by a grammar that is not left-recursive.

NORMAL FORMS

Two kinds of normal forms viz., Chomsky Normal Form and Greibach Normal Form (GNF) are

Chomsky Normal Form (CNF)

Any context-free language L without any λ - production is generated by a grammar in which productions are of the form $A \rightarrow BC$ or $A \rightarrow a$, where $A, B \in VN$, and $a \in V$

T. Procedure to find Equivalent Grammar in CNF

- (i) Eliminate the unit productions, and λ -productions if any,
- (ii) Eliminate the terminals on the right hand side of length two or more.

(iii) Restrict the number of variables on the right hand side of productions to two. Proof:

For Step (i): Apply the following theorem: "Every context free language can be generated by a grammar with no useless symbols and no unit productions"



Handwritten signature

PRINCIPAL
Sri Raaja Raajan College of
Amaravathipuram, Karaikal
Sivagangai Dist. Tamil Nadu

At the end of this step the RHS of any production has a single terminal or two or more symbols. Let us assume the equivalent resulting grammar as $G = (V_N, V_T, P, S)$.

For Step (ii): Consider any production of the form

$$A \rightarrow y_1 y_2 \dots y_m, \quad m \geq 2.$$

If y_1 is a terminal, say ' a ', then introduce a new variable B_a and a production

$$B_a \rightarrow a$$

Repeat this for every terminal on RHS.

Let P' be the set of productions in P together with the new productions

$B_a \rightarrow a$. Let V'_N be the set of variables in V_N together with B_a 's introduced for every terminal on RHS.

The resulting grammar $G_1 = (V'_N, V_T, P', S)$ is equivalent to G and every production in P' has either a single terminal or two or more variables.

For step (iii): Consider $A \rightarrow B_1 B_2 \dots B_m$

where B_i 's are variables and $m \geq 3$.

If $m = 2$, then $A \rightarrow B_1 B_2$ is in proper form.

The production $A \rightarrow B_1 B_2 \dots B_m$ is replaced by new productions

$$\begin{aligned} A &\rightarrow B_1 D_1, \\ D_1 &\rightarrow B_2 D_2, \\ &\dots \dots \dots \\ &\dots \dots \dots \\ D_{m-2} &\rightarrow B_{m-1} B_m \end{aligned}$$

where D_i 's are new variables.

The grammar thus obtained is G_2 , which is in CNF.



Handwritten signature

PRINCIPAL
Sri Raaja Raajan College of
Amaravathipuram, Karaikudi
Kanyakumari Dist. Tamil Nadu

Example

Obtain a grammar in Chomsky Normal Form (CNF) equivalent to the grammar G with

$$S \rightarrow aAbB$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bB \mid b.$$

Solution

- (i) There are no unit productions in the given set of P .
- (ii) Amongst the given productions, we have

$$A \rightarrow a,$$

$$B \rightarrow b$$

which are in proper form.

For $S \rightarrow aAbB$, we have

$$S \rightarrow B_a AB_b B,$$

$$B_a \rightarrow a$$

$$B_b \rightarrow b.$$

For $A \rightarrow aA$, we have

$$A \rightarrow B_a A$$

For $B \rightarrow bB$, we have

$$B \rightarrow B_b B.$$



PRINCIPAL
Sri Raja Raajan College of Engineering & Technology
Amaravathipudur, Karaikudi - 630 005
Sivagangai Dist. Tamil Nadu

(iii) In P' above, we have only

$$S \rightarrow B_a A B_b B$$

not in proper form.

Hence we assume new variables D_1 and D_2 and the productions

$$S \rightarrow B_a D_1$$

$$D_1 \rightarrow A D_2$$

$$D_2 \rightarrow B_b B$$

Therefore the grammar in Chomsky Normal Form (CNF) is G_2 with the productions given by

$$S \rightarrow B_a D_1,$$

$$D_1 \rightarrow A D_2,$$

$$D_2 \rightarrow B_b B,$$

$$A \rightarrow B_a A,$$

$$B \rightarrow B_b B,$$

$$B_a \rightarrow a,$$

$$B_b \rightarrow b,$$

$$A \rightarrow a,$$

$$B \rightarrow b.$$

and



Pumping Lemma for CFG

A "Pumping Lemma" is a theorem used to show that, if certain strings belong to a language, then certain other strings must also belong to the language. Let us discuss a Pumping Lemma for CFL. We will show that, if L is a context-free language, then strings of L that are at least ' m ' symbols long can be "pumped" to produce additional strings in L . The value of ' m ' depends on the particular

PRINCIPAL
Sri Raaja Raajan College of Engineering & Technology
Amaravathipudur, Karaikudi - 606 004
Sivagangai Dist. Tamil Nadu

language. Let L be an infinite context-free language. Then there is some positive integer 'm' such that, if S is a string of L of Length at least 'm', then

(i) $S = uvwxy$ (for some u, v, w, x, y)

(ii) $|vwx| \leq m$

(iii) $|vx| \geq 1$

(iv) $uv^iwx^iy \in L$.

for all non-negative values of i .

It should be understood that

(i) If S is sufficiently long string, then there are two substrings, v and x , somewhere in S . There is stuff (u) before v , stuff (w) between v and x , and stuff (y), after x .

(ii) The stuff between v and x won't be too long, because $|vwx|$ can't be larger than m .

(iii) Substrings v and x won't both be empty, though either one could be.

(iv) If we duplicate substring v , some number (i) of times, and duplicate x the same number of times, the resultant string will also be in L .

Definitions

A variable is useful if it occurs in the derivation of some string. This requires that

(a) the variable occurs in some sentential form (you can get to the variable if you start from S), and (b) a string of terminals can be derived from the sentential form (the variable is not a "dead end"). A variable is "recursive" if it can generate a string containing itself. For example, variable A is recursive if



bnf
PRINCIPAL
Sri Raja Raman College of Engineering & Technology
Amaravathipuram, Karaikudi - 626 002
Sivagangai Dist. Tamil Nadu

$$S \Rightarrow^* uAy$$

for some values of u and y .

A recursive variable A can be either

- (i) "Directly Recursive", i.e., there is a production

$$A \rightarrow x_1 Ax_2$$

for some strings $x_1, x_2 \in (T \cup V)^*$, or

- (ii) "Indirectly Recursive", i.e., there are variables x_i and productions

$$A \rightarrow X_1 \dots$$

$$X_1 \rightarrow \dots X_2 \dots$$

$$X_2 \rightarrow \dots X_3 \dots$$

$$X_N \rightarrow \dots A \dots$$

Proof of Pumping Lemma

- (a) Suppose we have a CFL given by L . Then there is some context-free Grammar G that generates

L . Suppose

- (i) L is infinite, hence there is no proper upper bound on the length of strings belonging to L .

- (ii) L does not contain ϵ .

There are only a finite number of variables in a grammar and the productions for each variable have finite lengths. The only way that a grammar can generate arbitrarily long strings is if one or more variables is both useful and recursive. Suppose no variable is recursive. Since the start symbol is non recursive, it must be defined only in terms of terminals and other variables. Then since those variables are non recursive, they have to be defined in terms of terminals and still other variables and so on.

After a while we run out of "other variables" while the generated string is still finite. Therefore there is an upper bound on the length of the string which can be generated from the start symbol. This contradicts our statement that the language is finite.



PRINCIPAL
Sri Raja Rajaan College
Amaravathipuram, Karaikal
Kavagangai Dist. Tamil Nadu

Proof: Suppose L is a context-free language.

If string $X \in L$, where $|X| > m$, it follows that $X = uvwx$, where $|vwx| \leq m$.

Choose a value i that is greater than m . Then, wherever vwx occurs in the string $a^i b^i c^i$, it cannot contain more than two distinct letters it can be all a 's, all b 's, all c 's, or it can be a 's and b 's, or it can be b 's and c 's.

Therefore the string vx cannot contain more than two distinct letters; but by the "Pumping Lemma" it cannot be empty, either, so it must contain at least one letter.

Now we are ready to "pump".

Since $uvwx$ is in L , uv^2wx^2y must also be in L . Since v and x can't both be empty,

$$|uv^2wx^2y| > |uvwx|,$$

so we have added letters.

Both since vx does not contain all three distinct letters, we cannot have added the same number of each letter.

Therefore, uv^2wx^2y cannot be in L .

Thus we have arrived at a "contradiction".

Hence our original assumption, that L is context free should be false. Hence the language L is not context-free.

Example

Check whether the language given by $L = \{a_m b_m c_n : m \leq n \leq 2m\}$ is a CFL or not.

Solution



PRINCIPAL

Sri Raaja Raajan College of Engineering & Technology
Amaravathipudur, Karaikudi - 618 001
Sivagangai Dist. Tamil Nadu

Let $s = a^n b^n c^{2n}$, n being obtained from Pumping Lemma.

Then $s = uvwxy$, where $1 \leq |vx| \leq n$.

Therefore, vx cannot have all the three symbols a, b, c .

If you assume that vx has only a 's and b 's then we can choose i such that uv^iwx^iy has more than $2n$ occurrence of a or b and exactly $2n$ occurrences of c .

Hence $uv^iwx^iy \notin L$, which is a contradiction. Hence L is not a CFL.

Closure properties of CFL —

Let Σ be an alphabet, and suppose that for every symbol a in Σ , we choose a language L_a . These chosen languages can be over any alphabets, not necessarily Σ and not necessarily the same. This choice of languages defines a function s (a substitution) on Σ , and we shall refer to L_a as $s(a)$ for each symbol a .

If $w = a_1 a_2 \dots a_n$ is a string in Σ^* , then $s(w)$ is the language of all strings $x_1 x_2 \dots x_n$ such that string x_i is in the language $s(a_i)$, for $i = 1, 2, \dots, n$. Put another way, $s(w)$ is the concatenation of the languages $s(a_1)s(a_2)\dots s(a_n)$. We can further extend the definition of s to apply to languages: $s(L)$ is the union of $s(w)$ for all strings w in L .

Theorem 7.23: If L is a context-free language over alphabet Σ , and s is a substitution on Σ such that $s(a)$ is a CFL for each a in Σ , then $s(L)$ is a CFL.

PROOF: The essential idea is that we may take a CFG for L and replace each terminal a by the start symbol of a CFG for language $s(a)$. The result is a single CFG that generates $s(L)$. However, there are a few details that must be gotten right to make this idea work.

More formally, start with grammars for each of the relevant languages, say $G = (V, \Sigma, P, S)$ for L and $G_a = (V_a, T_a, P_a, S_a)$ for each a in Σ . Since we can choose any names we wish for variables, let us make sure that the sets of variables are disjoint; that is, there is no symbol A that is in two or more of V and any of the V_a 's. The purpose of this choice of names is to make sure that when we combine the productions of the various grammars into one set of productions, we cannot get accidental mixing of the productions from two grammars and thus have derivations that do not resemble the derivations in any of the given grammars.

We construct a new grammar $G' = (V', T', P', S)$ for $s(L)$, as follows:



Handwritten signature

PRINCIPAL

Sri Raaja Raajan College of Engineering
Amaravathipuram, Karaikudi - 606 001
Vagangai Dist. Tamil Nadu

- V' is the union of V and all the V_a 's for a in Σ .
- T' is the union of all the T_a 's for a in Σ .
- P' consists of:
 1. All productions in any P_a , for a in Σ .
 2. The productions of P , but with each terminal a in their bodies replaced by S_a everywhere a occurs.

Thus, all parse trees in grammar G' start out like parse trees in G , but instead of generating a yield in Σ^* , there is a frontier in the tree where all nodes have labels that are S_a for some a in Σ . Then, dangling from each such node is a parse tree of G_a , whose yield is a terminal string that is in the language $s(a)$.

Applications of substitution



PRINCIPAL
Raaja Raajan College of Engineering
Amaravathipudur, Karaikudi - 626 002
Sivagangai Dist. Tamil Nadu

Theorem 7.24: The context-free languages are closed under the following operations:

1. Union.
2. Concatenation.
3. Closure (*), and positive closure (+).
4. Homomorphism.

PROOF: Each requires only that we set up the proper substitution. The proofs below each involve substitution of context-free languages into other context-free languages, and therefore produce CFL's by Theorem 7.23.

1. *Union:* Let L_1 and L_2 be CFL's. Then $L_1 \cup L_2$ is the language $s(L)$, where L is the language $\{1, 2\}$, and s is the substitution defined by $s(1) = L_1$ and $s(2) = L_2$.
2. *Concatenation:* Again let L_1 and L_2 be CFL's. Then $L_1 L_2$ is the language $s(L)$, where L is the language $\{12\}$, and s is the same substitution as in case (1).
3. *Closure and positive closure:* If L_1 is a CFL, L is the language $\{1\}^*$, and s is the substitution $s(1) = L_1$, then $L_1^* = s(L)$. Similarly, if L is instead the language $\{1\}^+$, then $L_1^+ = s(L)$.
4. Suppose L is a CFL over alphabet Σ , and h is a homomorphism on Σ . Let s be the substitution that replaces each symbol a in Σ by the language consisting of the one string that is $h(a)$. That is, $s(a) = \{h(a)\}$, for all a in Σ . Then $h(L) = s(L)$.

Reversal




PRINCIPAL

Sri Raja Raajan College of Engineering & Technology
Amaravathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

Theorem 7.25: If L is a CFL, then so is L^R .

PROOF: Let $L = L(G)$ for some CFL $G = (V, T, P, S)$. Construct $G^R = (V, T, P^R, S)$, where P^R is the "reverse" of each production in P . That is, if $A \rightarrow \alpha$ is a production of G , then $A \rightarrow \alpha^R$ is a production of G^R . It is an easy induction on the lengths of derivations in G and G^R to show that $L(G^R) = L^R$. Essentially, all the sentential forms of G^R are reverses of sentential forms of G , and vice-versa. We leave the formal proof as an exercise. \square

Inverse Homomorphism:

Theorem 7.30: Let L be a CFL and h a homomorphism. Then $h^{-1}(L)$ is a CFL.

PROOF: Suppose h applies to symbols of alphabet Σ and produces strings in T^* . We also assume that L is a language over alphabet T . As suggested above, we start with a PDA $P = (Q, T, \Gamma, \delta, q_0, Z_0, F)$ that accepts L by final state. We construct a new PDA

$$P' = (Q', \Sigma, \delta', (q_0, \epsilon), Z_0, F \times \{\epsilon\}) \quad (7.1)$$

where:

1. Q' is the set of pairs (q, x) such that:

- (a) q is a state in Q , and
- (b) x is a suffix (not necessarily proper) of some string $h(a)$ for some input symbol a in Σ .



PRINCIPAL
Sri Raaja Raajan College of Engg. & Tech
Maravathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

That is, the first component of the state of P' is the state of P , and the second component is the buffer. We assume that the buffer will periodically be loaded with a string $h(a)$, and then allowed to shrink from the front, as we use its symbols to feed the simulated PDA P . Note that since Σ is finite, and $h(a)$ is finite for all a , there are only a finite number of states for P' .

2. δ' is defined by the following rules:

(a) $\delta'((q, \epsilon), a, X) = \{((q, h(a)), X)\}$ for all symbols a in Σ , all states q in Q , and stack symbols X in Γ . Note that a cannot be ϵ here. When the buffer is empty, P' can consume its next input symbol a and place $h(a)$ in the buffer.

(b) If $\delta(q, b, X)$ contains (p, γ) , where b is in T or $b = \epsilon$, then

$$\delta'((q, bx), \epsilon, X)$$

contains $((p, x), \gamma)$. That is, P' always has the option of simulating a move of P , using the front of its buffer. If b is a symbol in T , then the buffer must not be empty, but if $b = \epsilon$, then the buffer can be empty.

3. Note that, as defined in (7.1), the start state of P' is (q_0, ϵ) ; i.e., P' starts in the start state of P with an empty buffer.

4. Likewise, the accepting states of P' , as per (7.1), are those states (q, ϵ) such that q is an accepting state of P .

The following statement characterizes the relationship between P' and P :

- $(q_0, h(w), Z_0) \vdash_P^* (p, \epsilon, \gamma)$ if and only if $((q_0, \epsilon), w, Z_0) \vdash_{P'}^* ((p, \epsilon), \epsilon, \gamma)$.

Turing machine:

Informal Definition:

We consider here a basic model of TM which is deterministic and have one-tape. There are many variations, all



PRINCIPAL
Sri Raaja Raajan College of
Amaravathipudur, Karaikudi.
Sivagangai Dist. Tamil Nadu

The basic model of TM has a finite set of states, a semi-infinite tape that has a leftmost cell but is infinite to the right and a tape head that can move left and right over the tape, reading and writing symbols.

For any input w with $|w|=n$, initially it is written on the n leftmost (contiguous) tape cells. The infinitely many cells to the right of the input all contain a blank symbol, B which is a special tape symbol that is not an input symbol. The machine starts in its start state with its head scanning the leftmost symbol of the input w .

Depending upon the symbol scanned by the tape head and the current state the machine makes a move which consists of the following:

- writes a new symbol on that tape cell,
- moves its head one cell either to the left or to the right and
- (possibly) enters a new state.

The action it takes in each step is determined by a transition functions. The machine continues computing (i.e.

- it decides to "accept" its input by entering a special state called accept or final state or On some inputs the TM may keep on computing forever without ever accepting or rejecting the input, in which case it is said to "loop" on that input

Formal Definition :



A handwritten signature in green ink, appearing to be "Ranjit".

PRINCIPAL

Sri Raaja Raajan College of Engg & Tech
Amaravathipuram, Karaikudi - 630 001
Sivagangai Dist. Tamil Nadu

Formally, a deterministic turing machine (DTM) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where

- Q is a finite nonempty set of states.
- Σ is a finite non-empty set of tape symbols, called the tape alphabet of M .
- Γ is a finite non-empty set of input symbols, called the input alphabet of M .
- δ is the transition function of M ,
- $q_0 \in Q$ is the initial or start state.
- $B \in \Gamma \setminus \Sigma$ is the blank symbol
- $F \subseteq Q$ is the set of final state.

So, given the current state and tape symbol being read, the transition function describes the next state, symbol to be written on the tape, and the direction in which to move the tape head.

Transition function :

Transition function : δ

- The heart of the TM is the transition function, δ because it tells us how the machine gets one step to the next.
- when the machine is in a certain state $q \in Q$ and the head is currently scanning the tape $X \in \Gamma$, and if $\delta(q, X) = (p, Y, D)$, then the machine

1. replaces the symbol X by Y on the tape
2. goes to state p , and
3. the tape head moves one cell (i.e. one tape symbol) to the left (or right) if D is L (or R).

The ID (instantaneous description) of a TM capture what is going out at any moment i.e. it contains all the information to exactly capture the "current state of the computations".

It contains the following:

- The current state, q



PRINCIPAL
Sri Raaja Raajan College of
Engineering, Karaikudi - 630 301,
Sivagangai Dist. Tamil Nadu

- The position of the tape head,
- The constants of the tape up to the rightmost nonblank symbol or the symbol to the left of the head, whichever is rightmost.

Note that, although there is no limit on how far right the head may move and write nonblank symbols on the tape, at any finite time, the TM has visited only a finite prefix of the infinite

An ID (or configuration) of a TM M is denoted by $\alpha q \beta$ where $\alpha, \beta \in \Gamma^*$ and

- α is the tape contents to the left of the head
- q is the current state.
- β is the tape contents at or to the right of the tape head

That is, the tape head is currently scanning the leftmost tape symbol β . (Note that if $\beta = \epsilon$, then the tape is empty.)

If q_0 is the start state and w is the input to a TM M then the starting or initial configuration of M is denoted by $q_0 w$

Moves of Turing Machines



PRINCIPAL
Sri Raaja Raajan College of Engineering & Technology
Amaravathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

To indicate one move we use the symbol \vdash . Similarly, zero, one, or more moves will be represented by \vdash^* .

M is defined as follows.

Let $\alpha Z q X \beta$ be an ID of M where $X, Z \in \Gamma$, $\alpha, \beta \in \Gamma^*$ and $q \in Q$.

Let there exists a transition $\delta(q, X) = (p, Y, L)$.

Then we write $\alpha Z q X \beta \vdash_M \alpha q Z Y \beta$ meaning that ID $\alpha Z q X \beta$ yields $\alpha q Z Y \beta$.

- Alternatively, if $\delta(q, X) = (p, Y, R)$ is a transition of M , then we write $\alpha Z q X \beta \vdash \alpha q Z Y \beta$ which means that the ID $\alpha Z q X \beta$ yields $\alpha q Z Y \beta$.
- In other words, when two IDs are related by the relation \vdash , we say that the first one yields the second (or the second is the result of the first) by one move.
- If ID_j results from ID_i by zero, one or more (finite) moves then we write $ID_i \vdash^* ID_j$ (If the TM M is understood then the subscript M can be dropped from \vdash or \vdash^*).

Special Boundary Cases




HOD


PRINCIPAL

PRINCIPAL
Sri Raajan College of Engg. & Tech.
Mavathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

UNIT V- UNDECIDABILITY

UNSOLVABLE PROBLEMS AND COMPUTABLE

Design a Turing machine to add two given integers.

Solution:

Assume that m and n are positive integers. Let us represent the input as $0^m B 0^n$.

If the separating B is removed and 0's come together we have the required output, $m + n$ is unary.

- (i) The separating B is replaced by a 0.
- (ii) The rightmost 0 is erased i.e., replaced by B .

Let us define $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0\}, \{0, B\}, \delta, q_0, \{q_4\})$. δ is defined by Table shown below.

State	Tape Symbol	
	0	B
q_0	$(q_0, 0, R)$	$(q_1, 0, R)$
q_1	$(q_1, 0, R)$	(q_2, B, L)
q_2	(q_3, B, L)	—
q_3	$(q_3, 0, L)$	(q_4, B, R)

M starts from ID $q_0 0^m B 0^n$, moves right until seeking the blank B . M changes state to q_1 . On reaching the right end, it reverts, replaces the rightmost 0 by B . It moves left until it reaches the beginning of the input string. It halts at the final state q_4 .

Some unsolvable Problems are as follows:

- (i) Does a given Turing machine M halts on all input?
- (ii) Does Turing machine M halt for any input?
- (iii) Is the language $L(M)$ finite?
- (iv) Does $L(M)$ contain a string of length k , for some given k ?
- (v) Do two Turing machines M_1 and M_2 accept the same language?



PRINCIPAL
Sri Raajan College of Engg.
Kavathipudur, Karaikudi - 630
Sivagangai Dist. Tamil Nadu

It is very obvious that if there is no algorithm that decides, for an arbitrary given Turing machine M and input string w , whether or not M accepts w . These problems for which no algorithms exist are called "UNDECIDABLE" or "UNSOLVABLE".

Code for Turing Machine:

Our next goal is to devise a binary code for Turing machines so that each TM with input alphabet $\{0, 1\}$ may be thought of as a binary string. Since we just saw how to enumerate the binary strings, we shall then have an identification of the Turing machines with the integers, and we can talk about "the i th Turing machine, M_i ." To represent a TM $M = (Q, \{0, 1\}, \Gamma, \delta, q_1, B, F)$ as a binary string, we must first assign integers to the states, tape symbols, and directions L and R .

- We shall assume the states are q_1, q_2, \dots, q_r for some r . The start state will always be q_1 , and q_2 will be the only accepting state. Note that, since we may assume the TM halts whenever it enters an accepting state, there is never any need for more than one accepting state.
- We shall assume the tape symbols are X_1, X_2, \dots, X_s for some s . X_1 always will be the symbol 0, X_2 will be 1, and X_3 will be B , the blank. However, other tape symbols can be assigned to the remaining integers arbitrarily.
- We shall refer to direction L as D_1 and direction R as D_2 .



PRINCIPAL
Sri Raajan College of Engr
Amaravathipudur, Karaikudi - 6
Sivagangai Dist. Tamil Nadu

Since each TM M can have integers assigned to its states and tape symbols in many different orders, there will be more than one encoding of the typical TM. However, that fact is unimportant in what follows, since we shall show that no encoding can represent a TM M such that $L(M) = L_d$.

Once we have established an integer to represent each state, symbol, and direction, we can encode the transition function δ . Suppose one transition rule is $\delta(q_i, X_j) = (q_k, X_l, D_m)$, for some integers i, j, k, l , and m . We shall code this rule by the string $0^i 10^j 10^k 10^l 10^m$. Notice that, since all of i, j, k, l , and m are at least one, there are no occurrences of two or more consecutive 1's within the code for a single transition.

A code for the entire TM M consists of all the codes for the transitions, in some order, separated by pairs of 1's:

$$C_1 11 C_2 11 \cdots C_{n-1} 11 C_n$$

where each of the C 's is the code for one transition of M .



PRINCIPAL
Sri Raja Raajan College of Eng-
Arinaravathipudur, Karaikudi -
Sivagangai Dist. Tamil Nadu

Diagonalization language:

- The language L_d , the *diagonalization language*, is the set of strings w_i such that w_i is not in $L(M_i)$.

That is, L_d consists of all strings w such that the TM M whose code is w does not accept when given w as input.

The reason L_d is called a "diagonalization" language can be seen if we consider Fig. 9.1. This table tells for all i and j , whether the TM M_i accepts input string w_j ; 1 means "yes it does" and 0 means "no it doesn't."¹ We may think of the i th row as the *characteristic vector* for the language $L(M_i)$; that is, the 1's in this row indicate the strings that are members of this language.

		$j \rightarrow$				
		1	2	3	4	...
$i \downarrow$	1	0	1	1	0	...
	2	1	1	0	0	...
	3	0	0	1	1	...
	4	0	1	0	1	...

Diagonal

Proof that L_d is not recursively enumerable:



PRINCIPAL
Sri Raja Raajan College of Engg. &
Amaravathipudur, Karaikudi - 630 00
Sivagangai Dist, Tamil Nadu

Theorem 9.2: L_d is not a recursively enumerable language. That is, there is no Turing machine that accepts L_d .

PROOF: Suppose L_d were $L(M)$ for some TM M . Since L_d is a language over alphabet $\{0, 1\}$, M would be in the list of Turing machines we have constructed, since it includes all TM's with input alphabet $\{0, 1\}$. Thus, there is at least one code for M , say i ; that is, $M = M_i$.

Now, ask if w_i is in L_d .

- If w_i is in L_d , then M_i accepts w_i . But then, by definition of L_d , w_i is not in L_d , because L_d contains only those w_j such that M_j does not accept w_j .
- Similarly, if w_i is not in L_d , then M_i does not accept w_i . Thus, by definition of L_d , w_i is in L_d .

Since w_i can neither be in L_d nor fail to be in L_d , we conclude that there is a contradiction of our assumption that M exists. That is, L_d is not a recursively enumerable language.

Recursive Languages:

We call a language L *recursive* if $L = L(M)$ for some Turing machine M such that:

1. If w is in L , then M accepts (and therefore halts).
2. If w is not in L , then M eventually halts, although it never enters an accepting state.

A TM of this type corresponds to our informal notion of an "algorithm," a well-defined sequence of steps that always finishes and produces an answer. If we think of the language L as a "problem," as will be the case frequently, then problem L is called *decidable* if it is a recursive language, and it is called *undecidable* if it is not a recursive language.



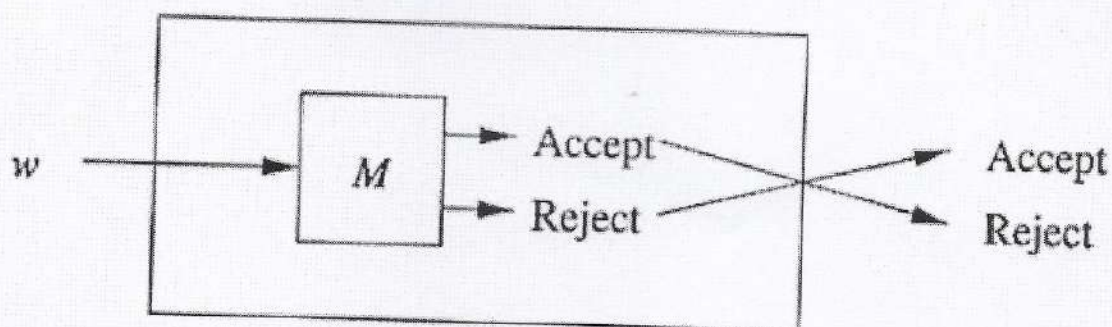
PRINCIPAL

S. Raja Raajan College of Engineering
Amaravathipuram, Karaikudi - 630 006
Sivagangai Dist. Tamil Nadu

Theorem 9.3: If L is a recursive language, so is \bar{L} .

PROOF: Let $L = L(M)$ for some TM M that always halts. We construct a TM \bar{M} such that $\bar{L} = L(\bar{M})$ by the construction suggested in Fig. 9.3. That is, \bar{M} behaves just like M . However, M is modified as follows to create \bar{M} :

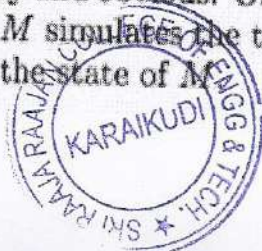
1. The accepting states of M are made nonaccepting states of \bar{M} with no transitions; i.e., in these states \bar{M} will halt without accepting.
2. \bar{M} has a new accepting state r ; there are no transitions from r .
3. For each combination of a nonaccepting state of M and a tape symbol of M such that M has no transition (i.e., M halts without accepting), add a transition to the accepting state r .



Since M is guaranteed to halt, we know that \bar{M} is also guaranteed to halt. Moreover, \bar{M} accepts exactly those strings that M does not accept. Thus \bar{M} accepts \bar{L} . \square

Theorem 9.4: If both a language L and its complement are RE, then L is recursive. Note that then by Theorem 9.3, \bar{L} is recursive as well.

PROOF: The proof is suggested by Fig. 9.4. Let $L = L(M_1)$ and $\bar{L} = L(M_2)$. Both M_1 and M_2 are simulated in parallel by a TM M . We can make M a two-tape TM, and then convert it to a one-tape TM, to make the simulation easy and obvious. One tape of M simulates the tape of M_1 , while the other tape of M simulates the tape of M_2 . The states of M_1 and M_2 are each components of the state of M .



Universal Language:

PRINCIPAL

Raaja Raajan College of Engg. & Tech.
Amaravathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

We define L_u , the *universal language*, to be the set of binary strings that encode, in the notation of Section 9.1.2, a pair (M, w) , where M is a TM with the binary input alphabet, and w is a string in $(0+1)^*$, such that w is in $L(M)$. That is, L_u is the set of strings representing a TM and an input accepted by that TM. We shall show that there is a TM U , often called the *universal Turing machine*, such that $L_u = L(U)$. Since the input to U is a binary string, U is in fact some M_j in the list of binary-input Turing machines we developed in

Undecidability of Universal Language:

Theorem 9.6: L_u is RE but not recursive.

PROOF: We just proved in Section 9.2.3 that L_u is RE. Suppose L_u were recursive. Then by Theorem 9.3, $\overline{L_u}$, the complement of L_u , would also be recursive. However, if we have a TM M to accept $\overline{L_u}$, then we can construct a TM to accept L_d (by a method explained below). Since we already know that L_d is not RE, we have a contradiction of our assumption that L_u is recursive.

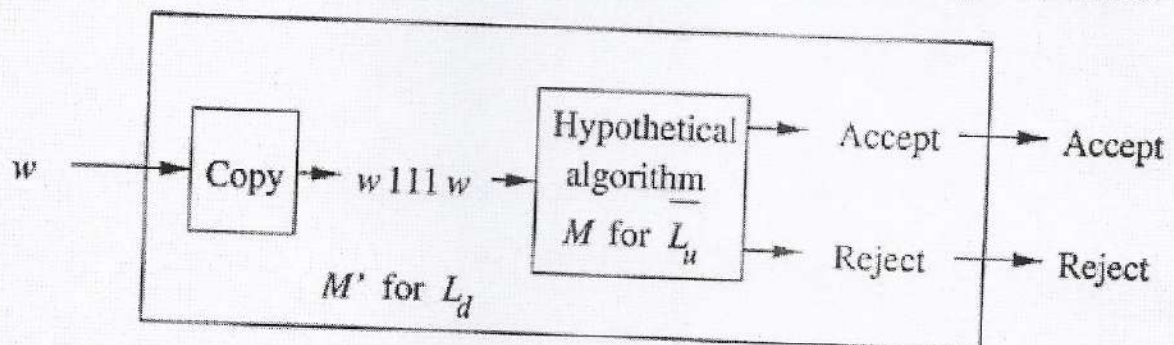


Figure 9.6: Reduction of L_d to $\overline{L_u}$

Suppose $L(M) = \overline{L_u}$. As suggested by Fig. 9.6, we can modify TM M into a TM M' that accepts L_d as follows.



PRINCIPAL
Sri Raaja Raajan College of Engg. & Tech.
Amaravathipudur, Karaikudi - 630 303
Sivagangai Dist. Tamil Nadu

1. Given string w on its input, M' changes the input to $w111w$. You may, as an exercise, write a TM program to do this step on a single tape. However, an easy argument that it can be done is to use a second tape to copy w , and then convert the two-tape TM to a one-tape TM.
2. M' simulates M on the new input. If w is w_i in our enumeration, then M' determines whether M_i accepts w_i . Since M accepts $\overline{L_u}$, it will accept if and only if M_i does not accept w_i ; i.e., w_i is in L_d .

Thus, M' accepts w if and only if w is in L_d . Since we know M' cannot exist by Theorem 9.2, we conclude that L_u is not recursive.

Problem -Reduction : If P_1 reduced to P_2 ,

Then P_2 is at least as hard as P_1 . Theorem: If P_1 reduces to P_2 then,

- If P_1 is undecidable then so is P_2 .
- If P_1 is Non-RE then so is P_2 .

Post's Correspondence Problem (PCP)



[Signature]
HOD

[Signature]
PRINCIPAL

PRINCIPAL
Sri Raaja Raajan College of Engg. & Tech.
Amaravathipuram, Karaikudi - 630 302
Sivagangai Dist. Tamil Nadu.

SRI RAAJA RAAJAN COLLEGE OF ENGINEERING AND TECHNOLOGY
AMARAVATHIPUDUR, KARAİKUDI
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SUBJECT: THEORY OF COMPUTATION
GIVEN DATE: 10.10.2021
SUBMITTED DATE: 15.10.2021
III Year B.E - V Semester

Sl. No.	Reg Num	Name of the Student	ASSIGNMENT TOPIC	MARKS
1	912519104001	AARTHI.K	DETERMINISTIC FINITE AUTOMATA	8
2	912519104002	ABIKSHA.S	DETERMINISTIC FINITE AUTOMATA	10
3	912519104003	ABINAYA.K	DETERMINISTIC FINITE AUTOMATA	7
4	912519104004	ABINAYAN.C	DETERMINISTIC FINITE AUTOMATA	7
5	912519104005	ANIMUTHU.K	DETERMINISTIC FINITE AUTOMATA	7
6	912519104006	ARIVAZHAGAN.S	DETERMINISTIC FINITE AUTOMATA	7
7	912519104008	ARJUN KUMAR.C	DETERMINISTIC FINITE AUTOMATA	7
8	912519104009	AZARUDEEN.S	DETERMINISTIC FINITE AUTOMATA	7
9	912519104010	BENITTA. A	DETERMINISTIC FINITE AUTOMATA	8
10	912519104011	DEENA JASMINE.A	DETERMINISTIC FINITE AUTOMATA	8
11	912519104012	DINESH.M	DETERMINISTIC FINITE AUTOMATA	8
12	912519104013	JENCY.C	NON-DETERMINISTIC FINITE AUTOMATA	8
13	912519104014	KAYALVIZHI.K	NON-DETERMINISTIC FINITE AUTOMATA	9
14	912519104015	MANIMEGALAI.R	NON-DETERMINISTIC FINITE AUTOMATA	8
15	912519104016	MUGESHKANNA.R	NON-DETERMINISTIC FINITE AUTOMATA	7
16	912519104017	MUTHUKAVIYA.T	NON-DETERMINISTIC FINITE AUTOMATA	9
17	912519104018	MUTHUKUMAR.A	NON-DETERMINISTIC FINITE AUTOMATA	7
18	912519104019	NITHIYASRIBHUVAN	NON-DETERMINISTIC FINITE AUTOMATA	10
19	912519104020	NIVETHITHA.K	NON-DETERMINISTIC FINITE AUTOMATA	10
20	912519104021	PAVITHRA.S	NON-DETERMINISTIC FINITE AUTOMATA	8
21	912519104022	PIRIYADHARSHINI.A	NON-DETERMINISTIC FINITE AUTOMATA	8

PRINCIPAL
 Sri Raja Raajan College of Engg
 Amaravathipudur, Karaikudi - 630 5
 Sivagangai Dist. Tamil Nadu

22	912519104023	RAMJI.B	EQUIVALENCE AND MINIMIZATION OF AUTOMATA	10
23	912519104025	SATHISH.S	EQUIVALENCE AND MINIMIZATION OF AUTOMATA	9
24	912519104026	SENTHOORADEVI.P	EQUIVALENCE AND MINIMIZATION OF AUTOMATA	8
25	912519104027	SOWNTHARYA.N	EQUIVALENCE AND MINIMIZATION OF AUTOMATA	7
26	912519104028	SRINIDHI.A	EQUIVALENCE AND MINIMIZATION OF AUTOMATA	10
27	912519104029	SWATHI.J	EQUIVALENCE AND MINIMIZATION OF AUTOMATA	8
28	912519104030	SWETHA.A	EQUIVALENCE AND MINIMIZATION OF AUTOMATA	8
29	912519104031	VELLAIYAMMAL.K	EQUIVALENCE AND MINIMIZATION OF AUTOMATA	7

HOD

PRINCIPAL

Sri Raaja Raajan College of Engg
Amaravathipuram, Karaikudi - 630 006
Sivagangai Dist. Tamil Nadu





SRI RAAJA RAAJAN COLLEGE OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NAME OF THE STAFF: V.MANJU

CLASS: III -CSE

NAME OF THE SUBJECT: CS8501 THEORY OF COMPUTATION

SEMINAR GIVEN DATE: 18.10.21

SEMINAR TAKEN DATE: 22.10.21

S.NO	NAME OF THE STUDENT	TOPIC
1	ABIKSHA.S	NFA TO DFA
2	RAMJI.B	REGULAR LANGUAGE



PRINCIPAL
Sri Raaja Raajan College of Eng.
Amaravathipudur, Karaikudi - 630 001
Sivagangai Dist. Tamil Nadu



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY



AMARAVATHIPUTHUR POST, KARAİKUDI - 630301

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NAME OF THE STAFF : Mrs.V.Manju, AP/CSE

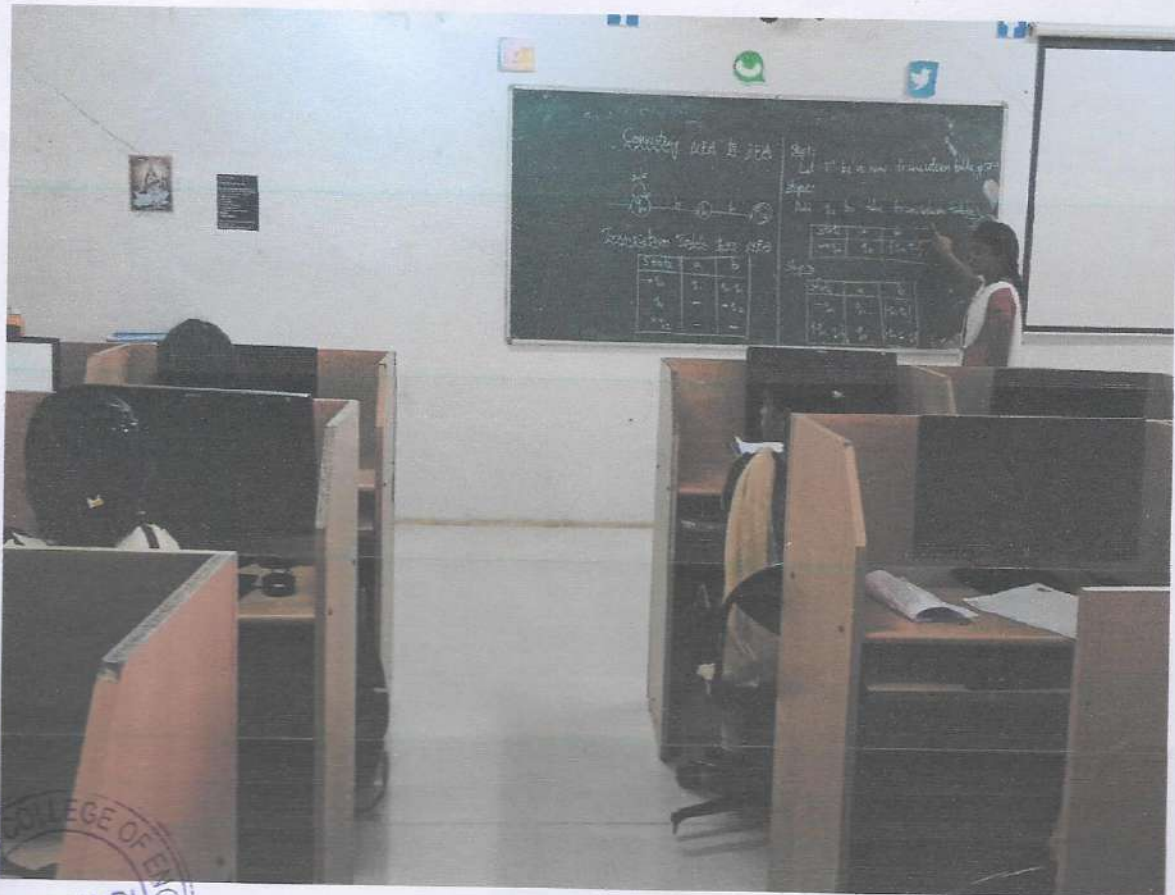
CLASS : III CSE

NAME OF THE SUBJECT : CS8501 THEORY OF COMPUTATION

SEMINAR GIVEN DATE : 18.10.2021

SEMINAR TAKEN DATE : 22.10.2021

S.NO.	NAME OF THE STUDENT	TOPIC
1.	S.ABIKSHA	NFA TO DFA
2.	B.RAMJI	REGULAR LANGUAGE



PRINCIPAL
Sri Raaja Raajan College of Engg.
Amaravathipudur, Karaikudi - 630
Sivagangai Dist. Tamil Nadu



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
Department of Computer Science & Engineering
CS8501-TOC INTERNAL I

PART-A

(10*2=20)

1. Define inductive proof.
2. Define the term epsilon transition.
3. Create a FA which checks whether the given binary number is even.
4. Differentiate between DFA and NFA.
5. What is proof by contradiction?
6. What are the applications of context free language?
7. What is a Regular expression?
8. Construct a DFA for the regular expression aa^*bb^* .
9. Construct NFA for all strings over alphabet $Z=\{a,b\}$ that contains a substring 'ab'.
10. What is $\{10, 11\}$? Write at least first seven terms.

PART-B

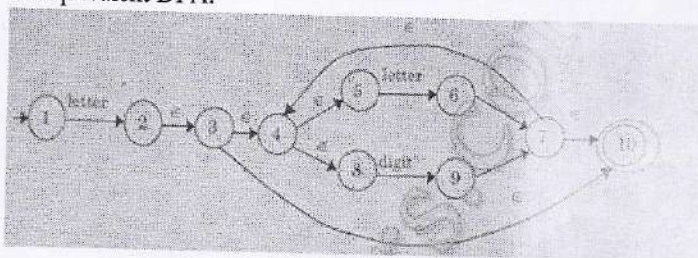
(13*5=65)

11. a) Construct a DFA equivalent to the NFA. $M = (\{p, q, r\}, \{0, 1\}, \delta, p, \{q, s\})$ Where δ is defined in the following table.

	0	1
p	{q,s}	{q}
q	{r}	{q,r}
r	{s}	{p}
s	-	{p}

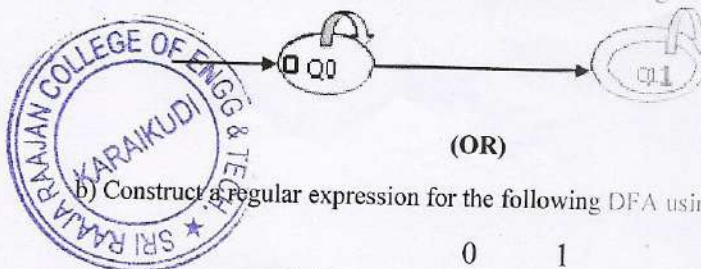
(OR)

- b) Prove that "A language accepted by some DFA iff L is accepted by NFA"
12. a) Consider the following ϵ -NFA for an identifier. Consider the ϵ -closure of each state and give its equivalent DFA.



(OR)

- b) Explain about Finite Automata and Regular Expressions.
13. a) Construct an NFA without ϵ -transitions for the NFA given below.



(OR)

- b) Construct a regular expression for the following DFA using Kleene's theorem.

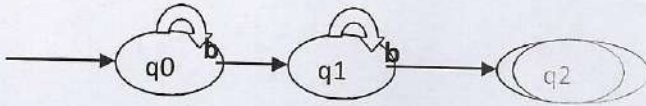
	0	1
$\rightarrow^* A$	A	B
B	C	B

PRINCIPAL
Sri Raaaja Raaajan College of Eng
Amaravathipudur, Karaikudi -
Sivagangai Dist. Tamil Nadu

C A B

14. a) Construct finite automata to accept the string $\{0,1\}$ that always ends with 00.
(OR)

b) Obtain the regular expression for the finite automata.



15. a) Convert the NFA with ϵ with its equivalent DFA.



(OR)

b) Examine whether the language $L = (0^n 1^n | n \geq 1)$ is regular or not? Justify your answer.

PART-C

(1*15=15)

16. a) Construct minimized automata for the following automata to define the same language.

$\rightarrow q_0$	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
$*q_3$	q_3	q_0
q_4	q_3	q_5
q_5	q_6	q_4
q_6	q_5	q_6
q_7	q_6	q_3

(OR)

- b) Construct the following ϵ NFA to DFA.

states	ϵ	a	b	c
p	Φ	{p}	{q}	{r}
q	{p}	{q}	{r}	{p,q}
*r	{q}	{r}	Φ	Φ

MSF
STAFF ITC

PRINCIPAL
HOD
Sri Raaja Raajan College of E
Amaravathipuram, Karaikudi - 630 001
Sivagangai Dist. Tamil Nadu



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
Department of Computer Science & Engineering
CS8501-TOC INTERNAL II

PART-A

(10*2=20)

1. What is Pushdown Automata?
2. Define Deterministic Pushdown Automata?
3. What is context free language derived from the following grammar?
 $S \rightarrow aSbS \rightarrow ab$
4. State the pumping lemma for CFL.
5. What is instantaneous description of PDA (ID)?
6. What is Turing machine?
7. Define moves of a Turing machine.
8. What is regular grammar?
9. Define Linear Bounded Automata.
10. What is Halting Problem?

PART-B

(5*3=15)

11. a) Discuss about the basic definitions of Turing Machine.

(OR)

- b) Prove that "If L is a context free Language, then there exists a PDA M , such that $L = N(M)$ " and "If $L \in N(M)$ for some PDA M , then L is a context free language".

12. a) Explain the programming techniques for Turing Machine Construction.

(OR)

- b) State the pumping lemma for CFL and Show that the language $L = \{a^n b^n c^n \mid n \geq 1\}$ is not a CFL.

13. a) Construct the PDA to accept the following $L = \{ww^R\}$ by empty stack.

(OR)

- b) Discuss about the Multihead and MultiTape Turing Machine.

14. a) i) Convert the Grammar

$S \rightarrow OSI \mid A$

$A \rightarrow IAO \mid SE$

- ii) Design a PDA that accepts the same language by empty stack.

(OR)

- b) Describe the Chomskian hierarchy of Languages.



PRINCIPAL

Sri Raaja Raajan College of Engineering
Amaravathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

15. a) Design a PDA accepting $L = \{a^n b^n \mid n \geq 1\}$ by final state.
(OR)

b) Explain Halting Problem. Is it solvable or unsolvable problem? Discuss.

PART-C

(1*15=15)

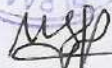
16. a) i) Design a Turing Machine to accept the Palindromes in an


Alphabet set. Trace the strings "0101" and "1001".

ii) Design a Turing Machine to perform addition and proper subtraction.
(OR)

b) Construct a Turing Machine to perform proper subtraction.




STAFF IIC


HOD



PRINCIPAL
Sri Raaja Raajan College of
Amaravathipuram, Karaikudi - 606 001
Sivagangai Dist. Tamil Nadu



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
Department of Computer Science & Engineering
CS8501-TOC MODEL I

PART-A

(10*2=20)

1. What is relation?
2. What is equivalence?
3. What are the types of derivation?
4. What is CFL?
5. State the Pumping lemma for CFL.
6. What are the closure properties of CFL's
7. What is Turing machine?
8. Define instantaneous description of turing machine.
9. What is recursive enumerable and recursive set?
10. What is decidable problem or decidability?

PART-B

(5*13=65)

11. a) i). Explain if L is accepted by an NFA with ϵ -transition then show that L is accepted by an NFA without ϵ -transition. (6)
 ii) Construct a DFA equivalent to the NFA $M = (\{p, q, r\}, \{0, 1\}, \delta, p, \{q, s\})$ Where δ is defined in the following table. (7)

	0	1
p	{q, s}	{q}
q	{r}	{q, r}
r	{s}	{p}
s	-	{p}

(OR)

- b) Prove for every $n \geq 1$ by mathematical induction $\sum(i) = \{n(n+1)/2\}$.
12. a) Demonstrate how the set $L = \{ab^n / n \geq 1\}$ is not a regular.
- (OR)
- b) Express that the regular languages are closed under: (a) union (b) intersection (c) Keene Closure (d) Complement (e) Difference
13. a) i). Discuss about PDA and CFL Prove the equivalence of PDA and CFL. (6)
 ii). If L is Context free language then prove that there exists PDA M such that $L = N(M)$.

(OR)

- b) Solve the following grammar

$S \rightarrow AaBbBB$

$A \rightarrow CA \mid A$

$C \rightarrow \epsilon$

Derive the string abaaba. Give Left most derivation (3) ii) Rightmost derivation (3) iii)

Derivation Tree (3) iv) For the string abaabbba, find the rightmost derivation. (4)

14. a) Express the following grammar G into Greibach Normal Form (GNF).

$S \rightarrow XA \mid BB$

$B \rightarrow b \mid SB$

$X \rightarrow b$

$A \rightarrow a$

(OR)

Sri RAAJA RAAJAN College of Engineering & Technology
 Armaravathipudur, Karaikudi - 630 301
 Sivagangai Dist. Tamil Nadu

- b) Illustrate the Turing machine for computing $f(m, n) = m - n$ (proper subtraction).
15. a) i). Describe about the tractable and intractable problems. (7)
 ii) Identify that "MPC reduce to PCP". (6)
- (OR)
- b) Discuss post correspondence problem. Let $\Sigma = \{0, 1\}$. Let A and B be the lists of three strings each, defined as

	ListA	ListB
i	wi	xi
1	1	111
2	10111	10
3	10	0

- i) Does the PCP have a solution? (7)
 ii) Prove that the universal language is recursively enumerable. (6)

PART-C

(1*15=15)

16. a) Tabulate the difference between the NFA and DFA. Convert the following ϵ -NFA to DFA.

states	ϵ	a	b	c
p	Φ	{p}	{q}	{r}
q	{p}	{q}	{r}	Φ
*r	{q}	{r}	Φ	{p}

(OR)

- b) Explain the DFA Minimization algorithm with an example.



STAFF IIC

HOD

PRINCIPAL

Sri Raaja Raajan College of Engg
 Amaravathipudur, Karaikudi - 621
 Sivagangai Dist. Tamil Nadu



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
Department of Computer Science & Engineering
CS8501-TOC MODEL 2

PART-A

(10*2=20)

1. State the Principle of induction.
2. Construct a DFA that accepts input string of 0's and 1's that end with 11.
3. Differentiate regular expression and regular language.
4. Define free grammar.
5. Define parse tree with an example.
6. What is ambiguous grammar?
7. State pumping lemma for CFL.
8. What is Turing machine?
9. What is recursively enumerable language?
10. State Rice's theorem.

PART-B

(5*13=65)

11. a) Construct a DFA equivalent to the NFA. $M = (\{p, q, r\}, \{0, 1\}, \delta, p, \{q, s\})$ Where δ is defined in the following table.

	0	1
p	{q,s}	{q}
q	{r}	{q,r}
r	{s}	{p}
s	-	{p}

(OR)

- b) Prove that "A language accepted by some DFA if L is accepted by NFA."

12. a) Construct a regular expression for the following DFA using Kleene's theorem.

(OR)

- b) (i) Discuss on regular expressions.
(ii) Discuss in detail about the closure properties of regular languages.

13. a) Describe the Chomskian hierarchy of Languages.

(OR)

- b) Construct the PDA to accept the following $L = \{ww^R\}$ by empty stack.

- a) Discuss about the Multihead and MultiTape Turing Machine.

(OR)

- b) Explain the programming techniques for Turing Machine Construction.



PRINCIPAL
Sri Raaja Raajan College of Engg. &
Amaravathipuram, Karaikudi - 630 003
Sivagangai Dist. Tamil Nadu

15. a) When we say a problem is decidable? Give an example of undecidable problem

(OR)

b) How to prove that the Post Correspondence problem is Undecidable.

PART-C

(1*15=15)

16. a) Construct minimized automata for the following automata to define the same language.

□q0	q1	q0
q1	q0	q2
q2	q3	q1
*q3	q3	q0
q4	q3	q5
q5	q6	q4
q6	q5	q6
q7	q6	q3

(OR)

Construct the following ϵ NFA to DFA.

states	E	a	b	c
p	Φ	{p}	{q}	{r}
q	{p}	{q}	{r}	{p,q}
*r	{q}	{r}	Φ	Φ



STAFF I/c

HOD

PRINCIPAL

Sri Raaja Raajan College of Engg. & Tech.
Amaravathipudur, Karaikudi - 630 001
Sivagangai Dist. Tamil Nadu



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
Department of Computer Science & Engineering
CS8501-TOC

INTERNAL1 - ANSWER KEY

PART-A

(10*2=20)

1. Statement $P(n)$ follows from

(a) $P(0)$ and

(b) $P(n-1)$ implies $P(n)$ for $n \geq 1$

Condition (a) is an inductive proof is the basis and Condition (b) is called the inductive step.

2. An **epsilon transition** (also **epsilon move** or **lambda transition**) allows an automaton to change its state spontaneously, i.e. without consuming an input symbol. It may appear in almost all kinds of nondeterministic automaton in formal language theory.

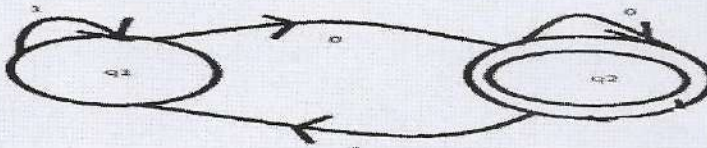
3. correct automation needs to remember whether the last character read was 10 so we need to state the initial state and accepting state 10 is heat transferred accepting state are remain there if we wear in the accepting state already.

have designed the following



A Binary String is even if it is ending with 0 and odd if its ending with 1. I have applied this. Am I right?

UPDATE:



4.

Deterministic Finite Automata	Non-Deterministic Finite Automata
Each transition leads to exactly one state called as deterministic	A transition leads to a subset of states i.e. some transitions can be non-deterministic.
Accepts input if the last state is in Final	Accepts input if one of the last states is in Final.
Backtracking is allowed in DFA.	Backtracking is not always possible.
Requires more space.	Requires less space.

PRINCIPAL

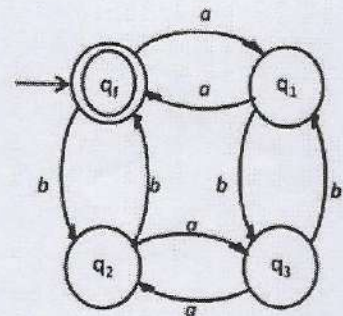
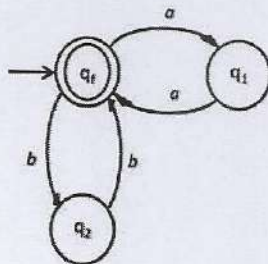
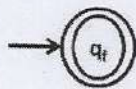
S. Raja Raajan College of Eng
Amaravathipudur, Karaikudi -
Sivagangai Dist. Tamil Nadu

5. In logic and mathematics, proof by contradiction is a form of proof that establishes the truth or the validity of a proposition, by showing that assuming the proposition to be false leads to a contradiction. Proof by contradiction is also known as indirect proof, proof by assuming the opposite.

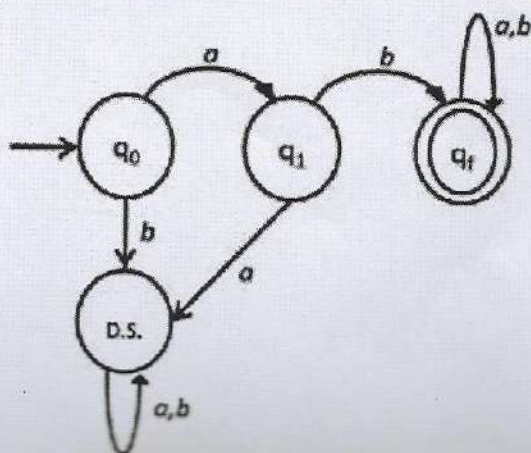
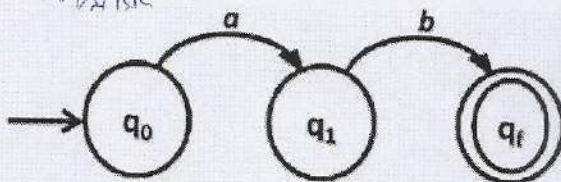
6. Context Free Grammars are used in Compilers (like GCC) for parsing. ... Context Free Grammars are used to define the High Level Structure of a Programming Languages.

7. A regular expression can also be described as **a sequence of pattern that defines a string**. Regular expressions are used to match character combinations in strings. String searching algorithm used this pattern to find the operations on a string.

8. ----- $L = \{ \epsilon, aabb, abab, baba, baab, bbaa, aabbaa, \dots \}$



9. ----- $L = \{ ab, aba, abb, abab, abaa, abbb, abba \}$



Amaravathipudur, Karaikal - 605 001
Sivagangai Dist. Tamil Nadu

10. Let $a = 5$ and $b = 10$... Write atleast first seven terms. ... 11) Write a RE which contains L having the strings which have at least one 0 and 1.

PART-B

(13*5=65)

11. a) The equivalent DFA is defined by $(\{\emptyset, p, q, r, s, pq, pr, ps, qr, qs, rs, pqr, pqs, prs, qrs, pqrs\}, \{0,1\}, \delta', p, \{\emptyset, pq, pr, ps, qr, qs, rs, pqr, pqs, prs, qrs, pqrs\})$, where δ' is defined by the chart below.

	0	1
\emptyset	\emptyset	\emptyset
p	pq	p
q	r	r
r	s	\emptyset
s	s	s
pq	pqr	pr
pr	pqs	p
ps	pqs	ps
qr	rs	r
qs	rs	rs
rs	s	s
pqr	pqrs	pr
prs	pqs	ps
qrs	rs	rs
pqrs	pqrs	prs

- b) For any DFA D, there is an NFA N such that $L(N) = L(D)$, and
For any NFA N, there is a DFA D such that $L(D) = L(N)$.

12. a) The method for converting the NFA with ϵ to DFA is explained below –

Step 1 – Consider $M = \{Q, \Sigma, \delta, q_0, F\}$ is NFA with ϵ . We have to convert this NFA with ϵ to equivalent DFA denoted by

$$M_0 = (Q_0, \Sigma, \delta_0, q_0, F_0)$$

Then obtain,

$$\epsilon\text{-closure}(q_0) = \{p_1, p_2, p_3, \dots, p_n\}$$

then $[p_1, p_2, p_3, \dots, p_n]$ becomes a start state of DFA

$$\text{now } [p_1, p_2, p_3, \dots, p_n] \in Q_0$$

Step 2 – We will obtain δ transition on $[p_1, p_2, p_3, \dots, p_n]$ for each input.

$$\delta_0([p_1, p_2, p_3, \dots, p_n], a) = \epsilon\text{-closure}(\delta(p_1, a) \cup \delta(p_2, a) \cup \dots \cup \delta(p_n, a))$$

$$= \bigcup_{i=1}^n \epsilon\text{-closure}(\delta(p_i, a))$$

Where a is input $\in \Sigma$

PRINCIPAL

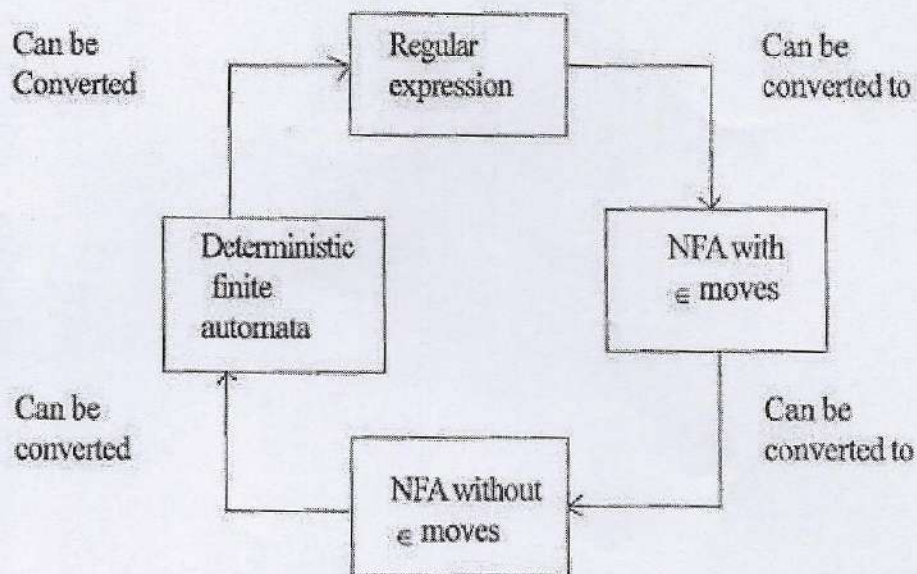
Sri Raaja Raajan College of Engg. & Tech.
Amaravathipuram, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

Step 3 – The state obtained $[p_1, p_2, p_3, \dots, p_n] \in Q_0$.

The states containing final state in p_i is a final state in DFA

b) **Finite automata** are formal (or abstract) machines for *recognizing* patterns. These machines are used extensively in compilers and text editors, which must recognize patterns in the input.

Regular expressions are a formal notation for *generating* patterns. This notation is used extensively in programming language manuals (used to describe legal patterns of input) and in command languages (such as the UNIX shell, where it is used to describe patterns for naming files, etc.)



13. a) **Step 1** – Find out all the ϵ -transitions from each state from Q . That will be called as ϵ -closure(q_i) where, $q_i \in Q$.

Step 2 – Then, δ_1 transitions can be obtained. The δ_1 transitions means an ϵ -closure on δ moves.

Step 3 – Step 2 is repeated for each input symbol and for each state of given NFA.

Step 4 – By using the resultant status, the transition table for equivalent NFA without ϵ can be built.

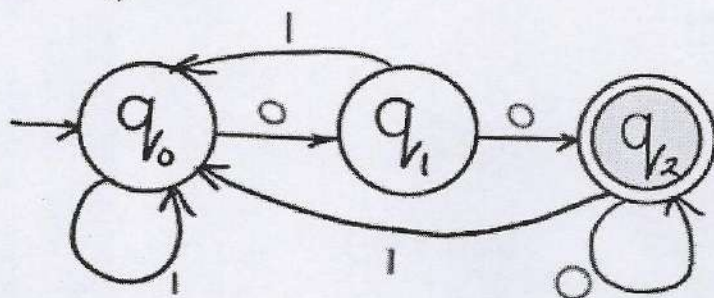
NFA with ϵ to without ϵ is as follows –

$$\delta_1(q, a) = \epsilon\text{-closure}(\delta(\delta^*(q, \epsilon), a)) \text{ where, } \delta^*(q, \epsilon) = \epsilon\text{-closure}(q)$$

b). To convert a regular expression to an NFA, we first convert it to an ϵ -NFA, then convert that to a DFA.

An ϵ -NFA is like an NFA, except that we are allowed to include "epsilon transitions". In a normal NFA or DFA, every character in the string causes a single transition in the machine, and each transition in the machine "consumes" one character. Epsilon transitions allow the machine to transition *without* consuming a character. They make it more convenient to build machines.

14. a)



b) To find the regular expression for the given automata, we first create equations of the given form for all the states as follows –

$$q_1 = q_1 a_1 1 + q_2 a_2 1 + \dots + q_n a_n 1 + \epsilon$$

$$q_2 = q_1 a_1 2 + q_2 a_2 2 + \dots + q_n a_n 2$$

-

-

-

-

$$q_n = q_1 a_1 n + q_2 a_2 n + \dots + q_n a_n n.$$

By repeatedly applying substitutions and Arden's Theorem we can express q_i in terms of a_{ij} 's. To get the set of strings recognized by FSA we have to take the union of all q_i 's corresponding to final states.

15. a) Step 1: We will take the **ϵ -closure** for the starting state of NFA as a starting state of DFA. ...

Step 2: If we found a new state, take it as current state and repeat step 1.

Step 3: Repeat Step 1 and Step 2 until there is no new state present in the transition table of DFA.

b) To prove if a language is a regular language, one can **simply provide the finite state machine that generates it**. If the finite state machine for a given language is not obvious (and this might certainly be the case if a language is, in fact, non-regular), the pumping lemma for regular languages is a useful tool.



PART-C

(1*15=15)

16. a) Step 1: Remove all the states that are unreachable from the initial state via any set of the transition of DFA.

Step 2: Draw the transition table for all pair of states.

Step 3: Now split the transition table into two tables T1 and T2. T1 contains all final states, and T2 contains non-final states.

Step 4: Find similar rows from T1 such that:

PRINCIPAL
Sri Raaja Raajan College of Engg.
Amaravathipuram, Karaikudi - 630 003
Sivagangai Dist. Tamil Nadu

1. $\delta(q, a) = p$

2. $\delta(r, a) = p$

That means, find the two states which have the same value of a and b and remove one of them.

Step 5: Repeat step 3 until we find no similar rows available in the transition table T1.

Step 6: Repeat step 3 and step 4 for table T2 also.

Step 7: Now combine the reduced T1 and T2 tables. The combined transition table is the transition table of minimized DFA.

16. b) **Step 1** – Consider $M = \{Q, \Sigma, \delta, q_0, F\}$ is NFA with ϵ . We have to convert this NFA with ϵ to equivalent DFA denoted by. ...

Step 2 – We will obtain δ transition on $[p_1, p_2, p_3, \dots p_n]$ for each input. ...

Step 3 – The state obtained $[p_1, p_2, p_3, \dots p_n] \in Q_0$



MSP
STAFF IIC

HOD

PRINCIPAL

Sri Raaja Raajan College of Engg
Amaravathipudur, Karaikudi -
Sivagangai Dist. Tamil Nadu



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
Department of Computer Science & Engineering
CS8501-TOC

INTERNAL2 - ANSWER KEY

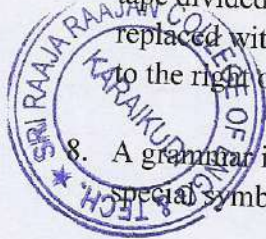
PART-A

(10*2=20)

1. Pushdown Automata is a finite automaton with extra memory called stack which helps Pushdown automata to recognize Context Free Languages. A Pushdown Automata (PDA) can be defined as : Q is the set of states. Σ is the set of input symbols. Γ is the set of pushdown symbols.
2. A deterministic pushdown automaton has at most one legal transition for the same combination of input symbol, state, and top stack symbol. This is where it differs from the nondeterministic pushdown automaton.
3. $S \rightarrow aSb$ $S \rightarrow ab$

 $S \rightarrow aabb$ $S \rightarrow ab$

 $S \rightarrow aabb$
4. Pumping Lemma for CFL states that for any Context Free Language L , it is possible to find two substrings that can be 'pumped' any number of times and still be in the same language.
5. The Instantaneous description is called as an informal notation, and explains how a Push down automata (PDA) computes the given input string and makes a decision that the given string is accepted or rejected.
6. A Turing machine is a mathematical model of computation that defines an abstract machine that manipulates symbols on a strip of tape according to a table of rules.
7. A Turing Machine (TM) is a mathematical model which consists of an infinite length tape divided into cells on which input is given. ... After reading an input symbol, it is replaced with another symbol, its internal state is changed, and it moves from one cell to the right or left.
8. A grammar is regular if it has rules of form $A \rightarrow a$ or $A \rightarrow aB$ or $A \rightarrow \epsilon$ where ϵ is a special symbol called NULL.
9. A linear bounded automaton is a multi-track non-deterministic Turing machine with a tape of some bounded finite length.
Length = function (Length of the initial input string, constant c)



10. In computability theory, the halting problem is the problem of determining, from a description of an arbitrary computer program and an input, whether the program will finish running, or continue to run forever.

PART-B

(13*5=65)

11. a) A Turing machine is a theoretical machine that manipulates symbols on a tape strip, based on a table of rules. Even though the Turing machine is simple, it can be tailored to replicate the logic associated with any computer algorithm. It is also particularly useful for describing the CPU functions within a computer.

Alan Turing invented the Turing machine in 1936, and he referred to it as an "a-machine" or automatic machine.

- Tape: A tape that is split into cells, one beside the other. Every cell includes a symbol from a certain finite alphabet. The alphabet includes a unique blank symbol as well as one or more other symbols. The volume of tape required for the computation is always included in the Turing machine.
- Head: A head that is able to write and read symbols on the tape. In certain models, the head moves while the tape is fixed.
- State register: A state register to store the Turing machine's state. There is a special start state through which the state register is initialized.
- Finite table: A finite table (sometimes referred to as a transition function or an action table) of instructions, which are generally quintuples, but occasionally quadruples.

b) Proof:

Let $G = (V, T, P, S)$ be a CFG in Greibach Normal form generating L

★ Let M be defined as

$$M = (\{q\}, T, V, \square, q, s, \square)$$

Where

$$\square(q, a, A) \square(q, \square)$$

Whenever $A \square a \square$ is in p



★ The PDA M simulates leftmost derivations of G in GNF each sentential form in a leftmost derivation consists of a string of terminals x followed by a string of variables.

★ M stores the suffix on the left sentential form on its stack after processing the prefix.

12. a) The Turing machine mathematically models a machine that mechanically operates on a tape. On this tape are symbols, which the machine can read and write one at a

time, using a tape head. Operation is fully determined by a finite set of elementary instructions such as "in state 42, if the symbol seen is 0, write a 1; if the symbol seen is 1, change into state 17; in state 17, if the symbol seen is 0, write a 1 and change to state 6;" etc. In the original article ("On computable numbers, with an application to the Entscheidungsproblem", see also references below), Turing imagines not a mechanism, but a person whom he calls the "computer", who executes these deterministic mechanical rules slavishly

b) Lemma

Let L be any CFL. Then there is a constant n , depending only on L , such that if $z \in L$ and

$|z| \geq n$ then

we may

write $z =$

$uvwxy$

such

that

1) $|Vx| \leq 1$

1) $|Vwx| \leq n$

2) for all $i \geq 0$, $uv^iwx^iy \in L$

Proof:

- Let G be a CFG in CNF generating $L \setminus \{\epsilon\}$
- If $Z \in L(G)$ and Z is long, then any parse tree for z must contain a long path
- We prove this by mathematical induction on i , that path of length i for z .
- If the word generated by a CNF grammar has no path of length greater than i , then the word length is not greater than 2^{i-1}

Basis:

$i=1$ is trivial

Since the tree must be of the form

Induction step:

Let $i > 1$

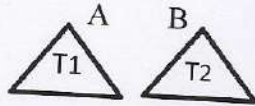
If there are no paths of length greater than $i-1$ in trees T_1 and T_2 , then the trees generate words of 2^{i-1} or fewer symbols

- Thus the entire tree generates a word no longer than 2^i



PRINCIPAL

Sri Raja Rajaan College of Engineering
Amaravathipuram, Karaikudi - 606 006
Sivagangai Dist. Tamil Nadu



- Let G have k variables and let $n = 2^k$
- If z is in $L(G)$ and $|z| \leq n$, then since $|z| > k-1$, any parse tree for z must have a path of length at least $k+1$.
- But such a path has $k+2$ vertices, and all except last vertex are labeled by variables.
- Thus there must be some variables that appear twice on the path.
- Some variables must appear twice near the bottom of the path.
- Then there must be two variables V_1 and V_2 on the path satisfying the following conditions.
 1. The vertices V_1 and V_2 both have the same label say A
 2. Vertex V_1 is closer to the root than vertex V_2
 3. The portion of the path from V_1 to the leaf is of length at most $k+1$
- Now the subtree T_1 and T_2 is the subtree generated by vertex V_2 and let Z_2 is the yield of subtree T_2

Then we can write

$$Z_1 = Z_3 Z_2 Z_4$$

- Z_3 and Z_4 cannot both be
 - Since the production used in the derivation of z_1 must be of the form $A \rightarrow BC$, for some variable B and C
- The Subtree T_2 must be completely within either the subtree generated by B or the subtree generated by C .
- Example:

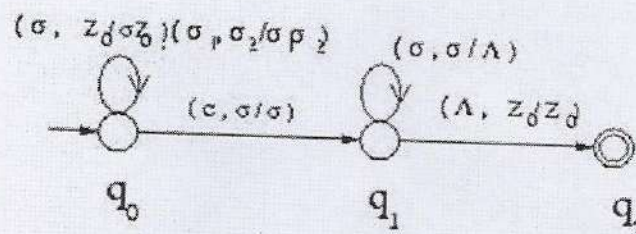
$G = (\{A, B, C\}, \{a, b\}, P, A)$

$P: \{A \rightarrow BC, B \rightarrow BA, A \rightarrow a, B \rightarrow b, \{C \rightarrow BA\}$



PRINCIPAL
Raaja Raajan College of Engineering & Technology
Amaravathipuram, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

13. a)



PDA accepting $w w^r$

b) Multi-tape Turing Machines have multiple tapes where each tape is accessed with a separate head. Each head can move independently of the other heads. Initially the input is on tape 1 and others are blank. At first, the first tape is occupied by the input and the other tapes are kept blank. Next, the machine reads consecutive symbols under its heads and the TM prints a symbol on each tape and moves its heads.

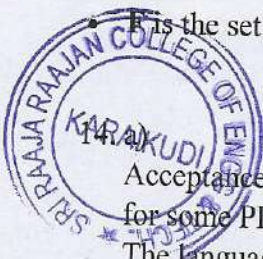
each tape and moves its heads.

A Multi-tape Turing machine can be formally described as a 6-tuple $(Q, X, B, \delta, q_0, F)$ where –

- Q is a finite set of states
- X is the tape alphabet
- B is the blank symbol
- δ is a relation on states and symbols where

$$\delta: Q \times X^k \rightarrow Q \times (X \times \{\text{Left_shift}, \text{Right_shift}, \text{No_shift}\})^k$$
 where there is k number of tapes
- q_0 is the initial state
- F is the set of final states

Acceptance by Empty Stack: On reading the input string from the initial configuration for some PDA, the stack of PDA gets empty. Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ be a PDA. The language acceptable by empty stack can be defined as: $N(\text{PDA}) = \{w \mid (q_0, w, Z) \vdash^* (p, \epsilon, \epsilon), q \in Q\}$



b) Chomsky Hierarchy represents the class of languages that are accepted by the different machines.

Chomsky hierarchy

Hierarchy of grammars according to Chomsky is explained below as per the grammar types –

Type 0 – It is an Unrestricted grammars

Unrestricted grammar – an **unrestricted grammar** is a 4-tuple (T, N, P, S) , which consisting of –

T = set of terminals

N = set of nonterminal

P = as set of productions, of the form –

$v \rightarrow w$

where v and w are strings consisting of nonterminal and terminals.

S = is called the **start symbol**.

Grammar type	Grammar accepted	Language accepted	Automaton
Type 0	unrestricted grammar	recursively enumerable language	Turing Machine
Type 1	context-sensitive grammar	context-sensitive language	linear-bounded automata
Type 2	Context-free grammar	Context-free language	Push down automata
Type 3	regular grammar	regular language	finite state automaton

15. a) Lemma

Let L be any CFL. Then there is a constant n , depending only on L , such that if $z \in L$ and

1) $|Vx| \leq 1$

2) $|Vwx| \leq n$

PRINCIPAL

Sri Raaja Raajan College of Engg. & Tech
Amaravathipuram, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

3) for all $i \geq 1$ $uv^i x^i y$ is in L

Proof:

- Let G be a CFG in CNF generating $L = \{ \square \}$
- If Z is in $L(G)$ and Z is long, then any parse tree for z must contain a long path
- We prove this by mathematical induction on i , that path of length i for z .
- If the word generated by a CNF grammar has no path of length greater than i , then the word length is no greater than 2^{i-1}

¹ Basis:

$i=1$ is trivial

Since the tree must be of the form

Induction step:

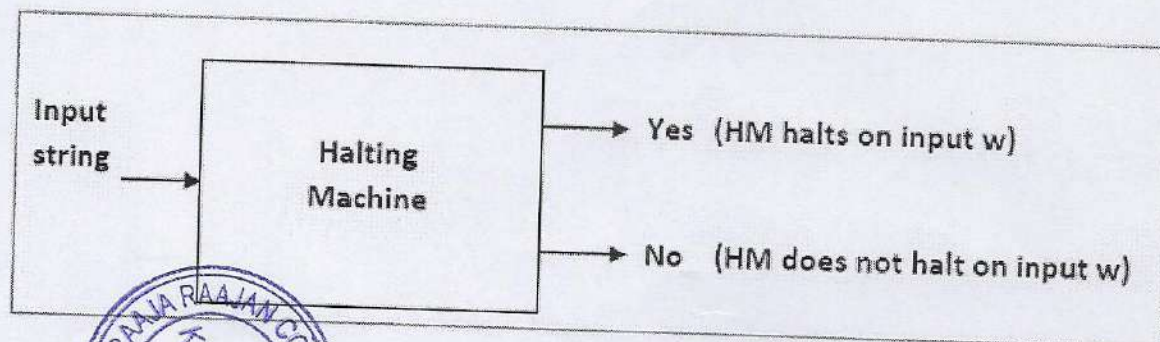
Let $i > 1$

- If there are no paths of length greater than $i - 1$ in trees T_1 and T_2 , then the trees generate words of 2^{i-1} or fewer symbols
- Thus the entire tree generates a word no longer than 2^{i-1}

b) **Input** – A Turing machine and an input string w .

Problem – Does the Turing machine finish computing of the string w in a finite number of steps? The answer must be either yes or no.

Proof – At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself. We will call this Turing machine as a **Halting machine** that produces a 'yes' or 'no' in a finite amount of time. If the halting machine finishes in a finite amount of time, the output comes as 'yes', otherwise as 'no'. The following is the block diagram of a Halting machine –

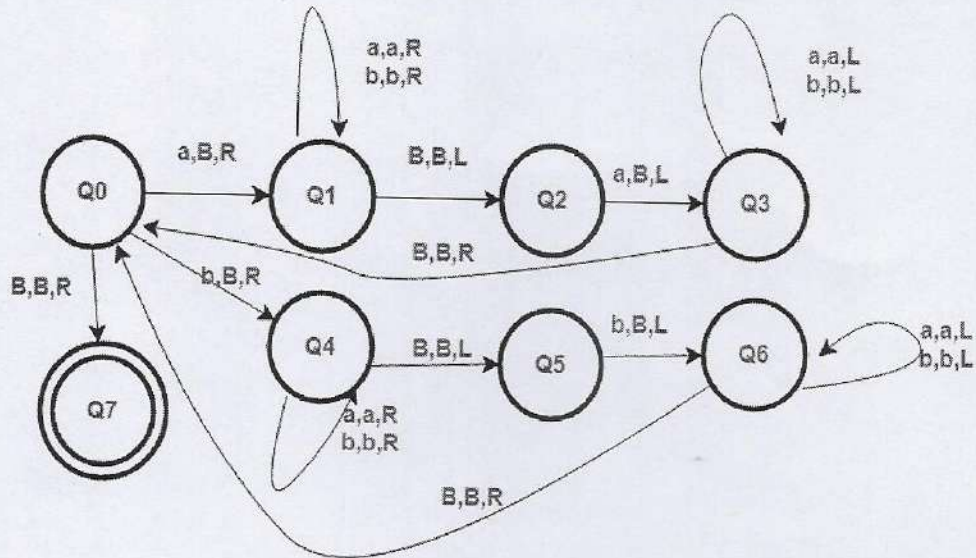


Sri Raaja Raajan College of Engg & Tech
Amaravathipuram, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

PART-C

(1*15=15)

16. a) We start with Q0 state if we get a symbol "a" as input then there should also be "a" at the ending of the string then only the string is palindrome and we have to verify that. We first make the current input "a" to B blank and go to state Q1 move rightwards to traverse the string till we reach the end.



b) Subtraction of two unary integers

$$3-2=1$$

In Turing Machine 3 represents - 111

2 represents: 11

Let 'M' be a symbol used to separate two integers

B	1	1	1	M	1	1	B
---	---	---	---	---	---	---	---

Here B= blank

M= Symbol used two separate two integers



↑ = Head

[Signature]
STAFF IIC

[Signature]
HOD

[Signature]



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
Department of Computer Science & Engineering
CS8501-TOC

MODEL1 - ANSWER KEY

PART-A

(10*2=20)

1. A relation between two sets is a **collection of ordered pairs containing one object from each set**. If the object x is from the first set and the object y is from the second set, then the objects are said to be related if the ordered pair (x,y) is in the relation. A function is a type of relation.
2. An equivalence relation is a relationship on a **set**, generally denoted by " \sim ", that is reflexive, symmetric, and transitive for everything in the set. ... Example: The relation "is equal to", denoted " $=$ ", is an equivalence relation on the set of real numbers since for any $x, y, z \in \mathbb{R}$:
3. **There are three types of Derivation trees;**
Leftmost Derivation tree.
Rightmost derivation tree.
Mixed derivation tree.
4. context-free language (CFL) is a **language generated by a context-free grammar (CFG)**. Context-free languages have many applications in programming languages, in particular, most arithmetic expressions are generated by context-free grammars.
5. Pumping Lemma for CFL states that for any Context Free Language L , it is possible to find two substrings that can be 'pumped' any number of times and still be in the same language. For any language L , we break its strings into five parts and pump second and fourth substring.
6. CFL's are **closed under union, concatenation, and Kleene closure**. Also, under reversal, homomorphisms and inverse homomorphisms. But not under intersection or difference.
7. A Turing machine is a **mathematical model of computation that defines an abstract machine that manipulates symbols on a strip of tape** according to a table of rules. ... The Turing machine was invented in 1936 by Alan Turing, who called it an "a-machine"
8. **All symbols to left of head, State of machine, symbol head is scanning and all symbols to right of head.** Programming Turing machine can be done entirely in finite state logic, but can also be done with information on tape.
9. A recursively enumerable language is a **recursively enumerable subset in the set of all possible words over the alphabet of the language**. A recursively enumerable language is a formal language for which there exists a Turing machine which will enumerate all valid strings of the language.

PRINCIPAL
Sri RaaJa RaaJan College of En
Amaravathipudur, Karaikudi -
Sivagangai Dist. Tamil Nadu

10. A problem is said to be **Decidable** if we can always construct a corresponding algorithm that can answer the problem correctly.

PART-B

(5*13=65)

11. a)i) In this method, we try to remove all the ϵ -transitions from the given Non-deterministic finite automata (NFA) –

The method is mentioned below stepwise –

- **Step 1** – Find out all the ϵ -transitions from each state from Q. That will be called as ϵ -closure(q_i) where, $q_i \in Q$.
- **Step 2** – Then, δ_1 transitions can be obtained. The δ_1 transitions means an ϵ -closure on δ moves.
- **Step 3** – Step 2 is repeated for each input symbol and for each state of given NFA.
- **Step 4** – By using the resultant status, the transition table for equivalent NFA without ϵ can be built.

NFA with ϵ to without ϵ is as follows –

$$\delta_1(q,a) = \epsilon\text{-closure}(\delta(\delta\epsilon^+(q,\epsilon),a)) \text{ where, } \delta\epsilon^+(q,\epsilon) = \epsilon\text{-closure}(q)$$

ii) The equivalent DFA is defined by

($\{\{\}, p, q, r, s, pq, pr, ps, qr, qs, rs, pqr, pqs, prs, qrs, pqrs\}, \{0,1\}, \delta', p, \{s, ps, qs, rs, pqs, prs, qrs, pqrs\}$), where δ' is defined by the chart below.

	0	1
$\{\}$	$\{\}$	$\{\}$
p	pq	p
q	r	r
r	s	$\{\}$
s	s	s
pq	pqr	pr
pr	pqs	p
ps	pqs	ps
qr	rs	r
qs	rs	rs
rs	s	s
pqr	pqrs	pr
prs	pqs	ps
qrs	rs	rs
pqrs	pqrs	prs



PRINCIPAL
Sri Raaja Raajan College of Engineering & Technology
Amaravathipudur, Karaikudi - 626 102
Sivagangai Dist. Tamil Nadu

b) Let,

$$p(n) = 1 + 2 + 3 + \dots + n$$

$$\text{Now, } p(1) = 1 = 2 \cdot 1 \cdot (1+1) = 1$$

So, $p(1)$ is true

Let, us assume that $p(k)$ is true that is

$$1 + 2 + \dots + k = 2k(k+1) \dots \dots \dots (1)$$

We shall prove that $p(k+1)$ is true

Now,

$$p(k+1)$$

$$= (1 + 2 + \dots + k) + (k+1)$$

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

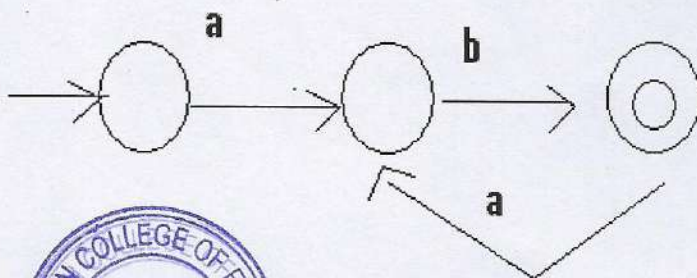
$$= 2(k+1)(k+2)$$

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

So, $p(k+1)$ is true.

Now by principle of mathematical induction curve $p(n)$ is true.

12. a) The language $a^n b^n$ where $n \geq 1$ is **not regular**, and it can be proved using the pumping lemma. Assume there is a finite state automaton that can accept the language. This finite automaton has a finite number of states k , and there is string x in the language such that $n > k$



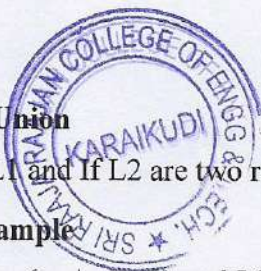
b) Union

If L_1 and L_2 are two regular languages, their union $L_1 \cup L_2$ will also be regular.

Example

$$L_1 = \{a^n \mid n > 0\} \text{ and } L_2 = \{b^n \mid n > 0\}$$

$$L_3 = L_1 \cup L_2 = \{a^n \cup b^n \mid n > 0\} \text{ is also regular.}$$



PRINCIPAL
Sri Raaja Raajan College of Engg
Amaravathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

Intersection

If L_1 and L_2 are two regular languages, their intersection $L_1 \cap L_2$ will also be regular.

Example

$$L_1 = \{a^m b^n \mid n > 0 \text{ and } m > 0\} \text{ and}$$

$$L_2 = \{a^m b^n \cup b^n a^m \mid n > 0 \text{ and } m > 0\}$$

$$L_3 = L_1 \cap L_2 = \{a^m b^n \mid n > 0 \text{ and } m > 0\} \text{ are also regular.}$$

Concatenation

If L_1 and L_2 are two regular languages, their concatenation $L_1.L_2$ will also be regular.

Example

$$L_1 = \{a^n \mid n > 0\} \text{ and } L_2 = \{b^n \mid n > 0\}$$

$$L_3 = L_1.L_2 = \{a^m . b^n \mid m > 0 \text{ and } n > 0\} \text{ is also regular.}$$

Kleene Closure

If L_1 is a regular language, its Kleene closure L_1^* will also be regular.

Example

$$L_1 = (a \cup b)$$

$$L_1^* = (a \cup b)^*$$

Complement

If $L(G)$ is a regular language, its complement $L'(G)$ will also be regular. Complement of a language can be found by subtracting strings which are in $L(G)$ from all possible strings.

Example

$$L(G) = \{a^n \mid n > 3\} \quad L'(G) = \{a^n \mid n \leq 3\}$$

13. a) i) Equivalence of PDA's and CFG's The goal is to prove that the following three classes of the languages are all the same class. 1. The context-free languages (The language defined by CFG's). 2. The languages that are accepted by empty stack by some PDA. 3. The languages that are accepted by final state by some PDA. Grammar PDA by empty stack PDA by final state Figure 1: Organization of constructions showing equivalence of three ways of defining the CFL's we have already shown that (2) and (3) are the same. Now, we prove that (1) and (2) are same.

ii) L is a deterministic context-free language (DCFL) if and only if there exists a deterministic PDA M such that $L=L(M)$. The language $L=\{a^n b^n \mid n \geq 0\}$ is a deterministic CFL. accepts the given language.

b) S $a^n a^n b^n b^n$

A

14. a) Step 1: Convert the grammar into CNF.



PRINCIPAL

Raja Raajan College of Engg. & Tech.
Maravathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

Step 2: If the grammar exists left recursion, eliminate it.

Step 3: In the grammar, convert the given production rule into GNF form.

b) Subtraction of two unary integers

$$3-2=1$$

In Turing Machine 3 represents – 111

2 represents: 11

Let 'M' be a symbol used to separate two integers

B	1	1	1	M	1	1	B
---	---	---	---	---	---	---	---

Here B= blank

M= Symbol used to separate two integers

↑ = Head

15. a) i) **Tractable Problem:** A problem that is solvable by a polynomial-time algorithm. The upper bound is polynomial. Here are **examples** of tractable problems (ones with known polynomial-time algorithms):

- Searching an unordered list
- Searching an ordered list
- Sorting a list
- Multiplication of integers (even though there's a gap)
- Finding a minimum spanning tree in a graph (even though there's a gap)

Intractable Problem: a problem that cannot be solved by a polynomial-time algorithm. The lower bound is exponential.

From a computational complexity stance, intractable problems are problems for which there exist no efficient algorithms to solve them.

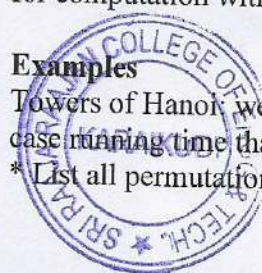
Most intractable problems have an algorithm that provides a solution, and that algorithm is the brute-force search.

This algorithm, however, does not provide an efficient solution and is, therefore, not feasible for computation with anything more than the smallest input.

Examples

Towers of Hanoi: we can prove that any algorithm that solves this problem must have a worst-case running time that is at least $2^n - 1$.

* List all permutations (all possible orderings) of n numbers.



PRINCIPAL
Sri Raaja Raajan College of Engineering
Amaravathipuram, Karaikudi -
Sivagangai Dist. Tamil Nadu

ii) Introduce two new symbol * and \$ that are not in input alphabet.

- For upper string of each domino, put * to left of every symbol.
- For lower string of each domino, put * to right of every symbol.
- Add a new domino based on the first domino of MPCP using the above rules, except that the lower string has an extra * at its left.
- Add a new domino whose upper string is *\$ and lower string is \$.

b) i) This PCP has a solution $M=4$

$i_1=2, i_2=1, i_3=1, i_4=3,$

$101111110=101111110$

The solution,

$I_1=2$

$I_2=1$

$I_3=1$

$I_4=3$

ii) L_u is recursively enumerable but not recursive. L_u is the set of binary strings that consist of encoded pairs (M, w) such that M is an encoding of a Turing machine and w is an encoding of a binary input string accepted by that Turing machine. This PCP has



PRINCIPAL
Sri Raaja Raajan College of Engg. & Tech.
Thipudur, Karaikudi - 630 301
Kavagangai Dist. Tamil Nadu

PART-C**(1*15=15)**

16. a)

Deterministic Finite Automata	Non-Deterministic Finite Automata
Each transition leads to exactly one state called as deterministic	A transition leads to a subset of states i.e. some transitions can be non-deterministic.
Accepts input if the last state is in Final	Accepts input if one of the last states is in Final.
Backtracking is allowed in DFA.	Backtracking is not always possible.
Requires more space.	Requires less space.

Step 1 – Consider $M = \{Q, \Sigma, \delta, q_0, F\}$ is NFA with ϵ . We have to convert this NFA with ϵ to equivalent DFA denoted by. ...

Step 2 – We will obtain δ transition on $[p_1, p_2, p_3, \dots, p_n]$ for each input. ...

Step 3 – The state obtained $[p_1, p_2, p_3, \dots, p_n] \in Q_0$...

The DFA diagram is as follows

b) DFA minimization stands for converting a given DFA to its equivalent DFA with minimum number of states. Suppose there is a DFA $D = \{Q, \Sigma, q_0, \delta, F\}$ which recognizes a language L . Then the minimized DFA $D = \{Q', \Sigma, q_0, \delta', F'\}$ can be constructed for language L as: Step 1: We will divide Q (set of states) into two sets.

Input – DFA

Output – Minimized DFA

Step 1 – Draw a table for all pairs of states (Q_i, Q_j) not necessarily connected directly [All are unmarked initially]

Step 2 – Consider every state pair (Q_i, Q_j) in the DFA where $Q_i \in F$ and $Q_j \notin F$ or vice versa and mark them. [Here F is the set of final states]

Step 3 – Repeat this step until we cannot mark anymore states –

If there is an unmarked pair (Q_i, Q_j) , mark it if the pair $\{\delta(Q_i, A), \delta(Q_j, A)\}$ is marked for some input alphabet.

Step 4 – Combine all the unmarked pair (Q_i, Q_j) and make them a single state in the reduced DFA.



MSP
STAFF ITC

HOD **PRINCIPAL**
Sri Raaja Raajan College of Engineering
Amaravathipuram, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
Department of Computer Science & Engineering
CS8501-TOC

MODEL2 - ANSWER KEY

PART-A

(10*2=20)

1. The principle of induction is **a way of proving that $P(n)$ is true for all integers $n \geq a$** Then we may conclude that $P(n)$ is true for all integers $n \geq a$. This principle is very useful in problem solving, especially when we observe a pattern and want to prove it.

2. $L = \{11, 111, 1111, \dots\}$,



Regular expression	Regular language
Regular Expressions are an algebraic way to describe languages	A regular language is a language that can be expressed with a regular expression or a deterministic or non-deterministic finite automata or state machine
Regular Expressions describe exactly the regular languages.	A language is a set of strings which are made up of characters from a specified alphabet, or set of symbols.

4. Context free grammar is a formal grammar which is used to generate all possible strings in a given formal language.

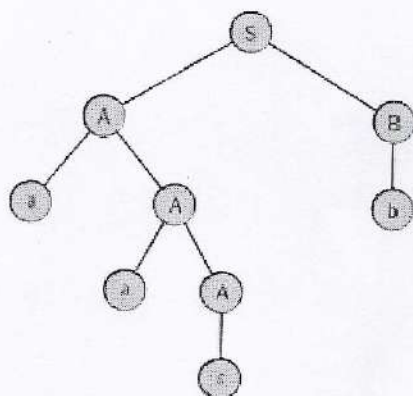
Context free grammar G can be defined by four tuples as:

$$G = (V, T, P, S)$$

5. Parse tree is **the hierarchical representation of terminals or non-terminals**. These symbols (terminals or non-terminals) represent the derivation of the grammar to yield input strings. In parsing, the string is derived using the beginning symbol.



PRINCIPAL
Sri RaaJa RaaJan College of Engg. & Tech
Kannuravathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu



6. A context free grammar G such that some word has two parse trees is said to be ambiguous.

An equivalent definition of ambiguity is that some word has more than one left most

derivation or more than one rightmost derivation.

7. The pumping lemma for context-free languages, also known as the Bar-Hillel lemma, is a lemma that gives a property shared by all context-free languages and generalizes the pumping lemma for regular languages.
8. a mathematical model of a hypothetical computing machine which can use a predefined set of rules to determine a result from a set of input variables.
9. It means **TM can loop forever for the strings which are not a part of the language**. RE languages are also called as Turing recognizable languages.
10. Rice's theorem states that **all non-trivial semantic properties of programs are undecidable**. A property is non-trivial if it is neither true for every partial computable function, nor false for every partial computable function.

PART-B

(5*13=65)

11. a) The equivalent DFA is defined by $(\{\{\}, p, q, r, s, pq, pr, ps, qr, qs, rs, pqr, pqs, prs, qrs, pqrs\}, \{0, 1\}, \delta', p, \{s, ps, qs, rs, pqs, prs, qrs, pqrs\})$, where δ' is defined by the chart below.

	0	1
$\{\}$	$\{\}$	$\{\}$
p	pq	p
q	r	r
r	s	$\{\}$
s	s	s
pq	pqr	pr
pr	pqs	p
ps	pqs	ps



Principal
Sri Raaja Raajan College of Engg. & Tech
Amaravathipuram, Karaikudi - 630 001
Sivagangai Dist. Tamil Nadu

qr|rs r
 qs|rs rs
 rs|s s
 pqr|pqrs pr
 prs|pqs ps
 qrs|rs rs
 pqrs|pqrs prs

b) For any DFA D , there is an NFA N such that $L(N) = L(D)$, and

For any NFA N , there is a DFA D such that $L(D) = L(N)$.

12. a) The set of regular languages, **the set of NFA-recognizable languages, and the set of DFA-recognizable languages** are all the same.

Kleene's Theorem states the equivalence of the following three statements

- A language accepted by Finite Automata can also be accepted by a Transition graph.
- A language accepted by a Transition graph can also be accepted by Regular Expression.
- A language accepted by Regular Expression can also be accepted by finite Automata.

b) i) A regular expression is a sequence of characters that specifies a search pattern in text. Usually such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation.

ii) Union

If L_1 and L_2 are two regular languages, their union $L_1 \cup L_2$ will also be regular.

Example

$L_1 = \{a^n \mid n > 0\}$ and $L_2 = \{b^n \mid n > 0\}$

$L_3 = L_1 \cup L_2 = \{a^n \cup b^n \mid n > 0\}$ is also regular.

Intersection

If L_1 and L_2 are two regular languages, their intersection $L_1 \cap L_2$ will also be regular.

Example

$L_1 = \{a^m b^n \mid n > 0 \text{ and } m > 0\}$ and

$L_2 = \{a^m b^n \cup b^n a^m \mid n > 0 \text{ and } m > 0\}$

$L_3 = L_1 \cap L_2 = \{a^m b^n \mid n > 0 \text{ and } m > 0\}$ are also regular.

Concatenation

If L_1 and L_2 are two regular languages, their concatenation $L_1.L_2$ will also be regular.

Example

$L1 = \{an \mid n > 0\}$ and $L2 = \{bn \mid n > 0\}$

$L3 = L1.L2 = \{am, bn \mid m > 0 \text{ and } n > 0\}$ is also regular.

Kleene Closure

If $L1$ is a regular language, its Kleene closure $L1^*$ will also be regular.

Example

$L1 = (a \cup b)$

$L1^* = (a \cup b)^*$

Complement

If $L(G)$ is a regular language, its complement $L'(G)$ will also be regular. Complement of a language can be found by subtracting strings which are in $L(G)$ from all possible strings.

Example

$L(G) = \{an \mid n > 3\}$ $L'(G) = \{an \mid n \leq 3\}$

13. a) Chomsky Hierarchy represents the class of languages that are accepted by the different machines.

Chomsky hierarchy

Hierarchy of grammars according to Chomsky is explained below as per the grammar types –

Type 0 – It is an Unrestricted grammars

Unrestricted grammar – an **unrestricted grammar** is a 4-tuple (T, N, P, S) , which consisting of –

T = set of terminals

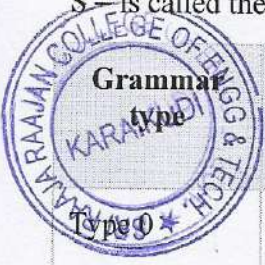
N = set of nonterminal

P = as set of productions, of the form –

$v \rightarrow w$

where v and w are strings consisting of nonterminal and terminals.

S = is called the **start symbol**.



Grammar type	Grammar accepted	Language accepted	Automaton
Type 0	unrestricted grammar	recursively enumerable language	Turing Machine
Type 1	context-sensitive	context-sensitive	linear-bounded

PRINCIPAL
Raja Raajan College of Eng
Maravathipudur, Karaikudi - 6
Sivagangai Dist. Tamil Nadu

Grammar type	Grammar accepted	Language accepted	Automaton
	grammar	language	automata
Type 2	Context-free grammar	Context-free language	Push down automata
Type 3	regular grammar	regular language	finite state automaton

b) Acceptance by Empty Stack: On reading the input string from the initial configuration for some PDA, the stack of PDA gets empty. Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ be a PDA. The language acceptable by empty stack can be defined as: $N(PDA) = \{w \mid (q_0, w, Z) \vdash^* (p, \epsilon, \epsilon), q \in Q\}$

14. a) Multi-tape Turing Machines have multiple tapes where each tape is accessed with a separate head. Each head can move independently of the other heads. Initially the input is on tape 1 and others are blank. At first, the first tape is occupied by the input and the other tapes are kept blank. Next, the machine reads consecutive symbols under its heads and the TM prints a symbol on each tape and moves its heads.

each tape and moves its heads.

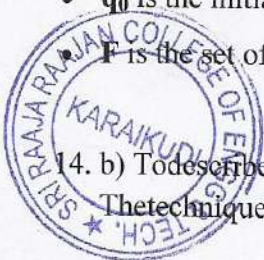
A Multi-tape Turing machine can be formally described as a 6-tuple $(Q, X, B, \delta, q_0, F)$ where –

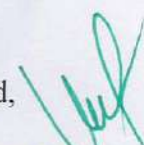
- Q is a finite set of states
- X is the tape alphabet
- B is the blank symbol
- δ is a relation on states and symbols where

$$\delta: Q \times X^k \rightarrow Q \times (X \times \{\text{Left_shift}, \text{Right_shift}, \text{No_shift}\})^k$$
 where there is k number of tapes

- q_0 is the initial state
- F is the set of final states

14. b) To describe the complicated TM constructions, some techniques are used. The techniques are




PRINCIPAL
 Sri Raaja Raajan College of Engineering
 Karavathipudur, Karaikudi - 626 002
 Sivaqangai Dist. Tamil Nadu



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CLASS: III -CSE

TEST PERIOD ANALYSIS - INTERNAL MARKS STATEMENT

S.N O	REG.NO	NAME OF THE STUDENT	IM2	IM3	IM4
1	912519104001	AARTH.I.K	95	87	93
2	912519104002	ABIKSHA.S	94	94	95
3	912519104003	ABINAYA.K	95	86	94
4	912519104004	ABINAYAN.C	96	83	95
5	912519104005	ANIMUTHU.K	88	88	93
6	912519104006	ARIVAZHAGAN.S	90	81	92
7	912519104008	ARJUN KUMAR.C	92	80	93
8	912519104009	AZARUDEEN.S	93	78	94
9	912519104010	BENITTA. A	94	89	91
10	912519104011	DEENA JASMINE.A	95	86	92
11	912519104012	DINESH.M	95	90	93
12	912519104013	JENCY.C	94	94	92
13	912519104014	KAYALVIZHI.K	95	93	94
14	912519104015	MANIMEGALAI.R	95	94	93
15	912519104016	MUGESHKANNA.R	95	86	93
16	912519104017	MUTHUKAVIYA.T	96	94	92
17	912519104018	MUTHUKUMAR.A	92	87	92
18	912519104019	NITHIYASRIBHUVANIK A.K	93	94	91
19	912519104020	NIVETHITHA.K	95	91	93
20	912519104021	PAVITHRA.S	94	88	94
21	912519104022	PIRIYADHARSHINI.A	96	78	92
22	912519104023	RAMJI.B	90	98	94
23	912519104025	SATHISH.S	94	88	96
24	912519104026	SENTHOORADEVI.P	95	91	93
25	912519104027	SOWNTHARYA.N	96	86	93
26	912519104028	SRINIDHI.A	96	97	93
27	912519104029	SWATHI.J	96	90	94
28	912519104030	SWETHA.A	96	94	95
29	912519104031	VELLAIYAMMAL.K	95	88	93

IM	All clear	75-80	81-90	91-100
IM2	29	-	03	26
IM3	29	05	13	11
IM4	29	-	-	29




PRINCIPAL
Sri Raaja Raajan College of Engg. & Tech
Amaravathipudur, Karaikudi - 62
Sivagangai Dist. Tamil Nadu

1. Storage in the finite control
2. Multiple tracks
3. Checking of symbols
4. Shifting over
5. Subroutines

15. a) A problem whose language is recursive is said to be decidable. Otherwise the problem is said to be undecidable. Decidable problems have an algorithm that takes as input an instance of the problem and determines whether the answer to that instance is "yes" or "no". (eg) of undecidable problems are (1) Halting problem of the TM.

b) If we are able to reduce Turing Machine to PCP then we will prove that PCP is undecidable as well. Consider Turing machine M to simulate PCP's input string w can be represented as . If there is match in input string w, then Turing machine M halts in accepting state.

PART-C

(1*15=15)

16. a) **Step 1:** Remove all the states that are unreachable from the initial state via any set of the transition of DFA.

Step 2: Draw the transition table for all pair of states.

Step 3: Now split the transition table into two tables T1 and T2. T1 contains all final states, and T2 contains non-final states.

Step 4: Find similar rows from T1 such that:

$$1. \delta(q, a) = p$$

$$2. \delta(r, a) = p$$

That means, find the two states which have the same value of a and b and remove one of them.

Step 5: Repeat step 3 until we find no similar rows available in the transition table T1.

Step 6: Repeat step 3 and step 4 for table T2 also.

Step 7: Now combine the reduced T1 and T2 tables. The combined transition table is the transition table of minimized DFA.

16. b) **Step 1** – Consider $M = \{Q, \Sigma, \delta, q_0, F\}$ is NFA with ϵ . We have to convert this NFA with ϵ to equivalent DFA denoted by. ...

Step 2 – We will obtain δ transition on $[p_1, p_2, p_3, \dots, p_n]$ for each input. ...

Step 3 – The state obtained $[p_1, p_2, p_3, \dots, p_n] \in Q_0$...



MSR
STAFF I/C

[Signature]
HOD

[Signature]
PRINCIPAL
Sri Raaja Raajan College of
Amaravathipudur, Karaikudi,
Sivagangai Dist. Tamil Na



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
AMARAVATHIPUTHUR POST, KARAUKUDI - 630301



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE FILE

SUB CODE : CP5001

SUB NAME : PRINCIPLES OF PROGRAMMING LANGUAGE

DEGREE/BRANCH : M.E/CSE

STAFF INCHARGE : Mr.P.PONVASAN,AP/CSE



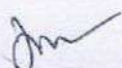
SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
AMARAVATHIPUTHUR POST, KARAİKUDI - 630301

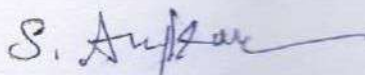



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Subject Code / Name : CP5001 PPL
Year / Dept. / Semester : I/ CSE /II Sem
Academic Year : 2020-21
Staff in-Charge : Mr.P.Ponvasan AP/CSE

S.NO.	INDEX	REMARKS
1.	Time table	
2.	Name List	
3.	Syllabus	
4.	Lesson Plan	
5.	Course Plan	
6.	Handwritten Notes	
7.	University Question Papers	
8.	Internal 1 Question Paper	
	Internal 2 Question Paper	
	Model Question Paper with key	
9.	Internal & Model Marks Statement	
10.	Internal 1 Result Analysis	
11.	Internal 2 Result Analysis	
12.	Model Result Analysis	
13.	Answer Sheets – Internal 1, Internal 2 & Model Exam	


Staff in-charge


HOD


Principal
PRINCIPAL

Sri Raaja Raajan College of Engg. & Tech
Amaravathipudur, Karaikudi - 630 301
Sivagangai Dist. Tamil Nadu

WILLINGNESS REPORT

From

P.Ponvasan M.E,
Asst. Professor,
Department of Computer Science and Engineering,
SRRCET,
Karaikudi-630 301.

TO

The Principal,
SRRCET,
Karaikudi-630301

Sir/ Madam,

Sub: Willingness Report for Subject: - Reg. I hereby express my willingness to handle the following subject in the following order of priority.

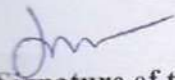
S.NO	Name of the Subject	Class	Reason for Selection
1	CP5292 –Internet of Things	M.E -I	Interested
2	CP5001-Principles of Programming Language	M.E-I	Interested

Thanking You

Date: 11.04.2021

Time: 10.30AM




Signature of the Staff

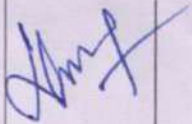
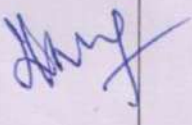


SRI RAAJA RAAJAN COLLEGE OF ENGINEERING AND TECHNOLOGY
AMARAVATHIPUTHUR POST, KARAİKUDI – 630 301.
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

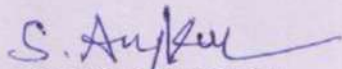


Minutes of Subject Allocating Meeting

I P.Ponvasan hereby submit the minutes of department meeting held for the subject allocating on 31/01/20223 at 10.00 am based on the annexure I & annexure II.

S.NO	NAME OF THE STAFF	NAME OF THE SUBJECT	CLASS	NO OF HRS	STAFF SIGN
1	P.Ponvasan	CP5292 – Internet of Things	I	7	
2		CP5001- Principles of Programming Language	I	8	




HOD SIGNATURE

CIRCULAR

DATE: 10.01.2021

Intimation of course allotment for faculties in the department of Computer Science and Engineering during EVEN SEMESTER for M.E-COMPUTER SCIENCE AND ENGINEERING.

S.NO	NAME OF THE FACULTY	TITLE OF COURSE	COURSE CODE	CLASS & YEAR
1	(PRINCIPAL)	Internet of Things	CP5292	I
		Principles of Programming Language	CP5001	I
2	P.PONVASAN ASSISTANT PROFESSOR	Internet of Things	CP5292	I
		Principles of Programming Language	CP5001	I

Note: Faculties are asked to follow the syllabus issued by Anna University, Chennai.

Copy to

1. The HOD
2. All the faculties of Computer Science Dept.
3. File Copy





SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
AMARAVATHIPUTHUR POST, KARAİKUDI - 630301



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

TIME TABLE

Name of the Staff : MR. P.Ponvasan **Designation** : AP / CSE
Degree/Branch : M.E/CSE **Year/Semester** : I/ II
Academic Year : 2020-21 **Subject code & Name** : CP5001 PPL

DAY	1 (9.30- 10.20)	2 (10.20- 11.10)	3 (10.50- 11.40)	4 (11.20- 12.25)	5 (1.05- 12.10)	6 (12.10- 01.00)	7 (3.15- 4.00)
Mon	PPL						
Tue			PPL				
Wed		PPL					
Thu					PPL		
Fri							PPL

Faculty in Charge





SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY

AMARAVATHIPUTHUR POST, KARAİKUDI – 630 301.



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

STUDENTS NAME LIST

Degree/Branch : M.E/CSE

Year/Semester : I/ II

Academic Year : 2020-21

Subject code & Name : CP5001 PPL

S.NO.	REG.NO.	NAME
1	912520405001	DEVI ABARNA SRI A
2	912520405002	GOWTHAMI R
3	912520405003	PANDIMUTHU C

Faculty in Charge



CP5001

PRINCIPLES OF PROGRAMMING LANGUAGES

L	T	P	C
3	0	0	3

OBJECTIVES:

- To understand and describe syntax and semantics of programming languages.
- To understand Data, Data types, and Bindings.
- To learn the concepts of functional and logical programming.
- To explore the knowledge about concurrent Programming paradigms.

UNIT I ELEMENTS OF PROGRAMMING LANGUAGES

9

Reasons for studying, concepts of programming languages, Language Evaluation Criteria, influences on Language design, Language categories. Programming Language Implementation – Compilation, Hybrid Implementation, Pure Interpretation and Virtual Machines. Describing Syntax and Semantics -Introduction - The General Problem of Describing Syntax-Formal Methods of Describing Syntax - Attribute Grammars - Describing the Meanings of Programs: Dynamic Semantics.

UNIT II DATA TYPES-ABSTRACTION

9

Introduction - Primitive Data Types- Character String Types- User-Defined Ordinal Types- Array types- Associative Arrays-Record Types- Tuple Types-List Types -Union Types - Pointer and Reference Types -Type Checking- Strong Typing -Type Equivalence - Theory and Data Types-Variables-The Concept of Binding -Scope - Scope and Lifetime - Referencing Environments - Named Constants- The Concept of Abstraction- Parameterized Abstract Data Types- Encapsulation Constructs- Naming Encapsulations

UNIT III FUNCTIONAL PROGRAMMING

9

Introduction- Mathematical Functions- Fundamentals of Functional Programming Languages- The First Functional Programming Language: LISP- An Introduction to Scheme- Common LISP- Haskell-F# - ML : Implicit Types- Data Types- Exception Handling in ML. Functional Programming with Lists- Scheme, a Dialect of Lisp- The Structure of Lists- List Manipulation- A Motivating Example: Differentiation- Simplification of Expressions- Storage Allocation for Lists.

UNIT IV LOGIC PROGRAMMING

9

Relational Logic Programming- Syntax- Basics- Facts- Rules- Syntax- Operational Semantics- Relational logic programs and SQL operations- Logic Programming- Syntax- Operational semantics- Data Structures-Meta-tools: Backtracking optimization (cuts); Unify; Meta-circular interpreters- The Origins of Prolog- Elements- of Prolog-Deficiencies of Prolog- Applications of Logic Programming.

UNIT V CONCURRENT PROGRAMMING

9

Parallelism in Hardware- Streams: Implicit Synchronization-Concurrency as Interleaving- Liveness Properties- Safe Access to Shared Data- Concurrency in Ada- Synchronized Access to Shared Variables- Synthesized Attributes- Attribute Grammars- Natural Semantics- Denotational Semantics -A Calculator in Scheme-Lexically Scoped Lambda Expressions- An Interpreter-Recursive Functions.

TOTAL: 45 PERIODS**OUTCOMES:**

Upon completion of this course, the students will be able to

- Describe syntax and semantics of programming languages
- Explain data, data types, and basic statements of programming languages
- Design and implement subprogram constructs, Apply object - oriented, concurrency, and event handling programming constructs
- Develop programs in LISP, ML, and Prolog.



REFERENCES:

1. Ghezzi, "Programming Languages", 3rd Edition, John Wiley, 2008
2. John C. Mitchell, "Concepts in Programming Languages", Cambridge University Press, 2004.
3. Louden, "Programming Languages", 3rd Edition, 2012.
4. Ravi Sethi, "Programming Languages: Concepts and Constructs", 2nd Edition, Addison Wesley, 1996.
5. Robert .W. Sebesta, "Concepts of Programming Languages", 10th Edition, Pearson Education, 2002.





SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
AMARAVATHIPUTHUR POST, KARAİKUDI - 630301



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Lecture Plan 2020-21 (Even Semester)

Name of the Faculty : Mr.P.Ponvasan

Department : CSE

Subject Code & Title : CP5001 PPL

Year / Semester : I / II

Degree/Branch : M.E/CSE

Individual Time Table

Days \ Period	I	II	III	IV	V	VI	VII
Monday	PPL						
Tuesday			PPL				
Wednesday		PPL					
Thursday					PPL		
Friday							PPL

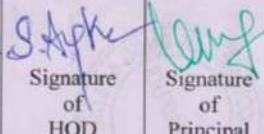

Tentative dates for Unit Tests & Model Exam

Unit test I : 10.3.21

Unit Test II : 15.4.21

Model Exam : 1.5.21

SRR CET/CSE/LP/SEM

Sl. No.	Proposed Lecture		Topics to be covered	Text / Reference Book Page No	Teaching Aids to be used			Actual Lecture		Remarks by the faculty	Steps taken to cover the incomplete portions, if any & signature of the Staff Member	Signature of Principal
	Date	Period			BB	OHP	PPT	Date	Period			
Unit – I												
ELEMENTS OF PROGRAMMING LANGUAGES												
1	1.2.21	1	Reasons for Studying Concepts of Programming Languages	R1/5								
2	2.2.21	3	Language Evaluation Criteria	R1/9								
3	3.2.21	2	Influences on Language design, Language categories	R1/14								
4	4.2.21	5	Programming Language Implementation — Compilation, Hybrid Implementation, Pure Interpretation and Virtual Machines	R1/23								
5	5.2.21	7	Describing Syntax and Semantics - Introduction	R1/27								
6	8.2.21	1	The General Problem of Describing Syntax	R1/29								
7	9.2.21	3	Formal Methods of Describing Syntax	R1/50								
8	10.2.21	2	Attribute Grammars	R1/34								
9	11.2.21	5	Describing the Meanings of Programs: Dynamic Semantics.	R1/57								

Signature
of
HOD

Signature
of
Principal

PRINCIPAL

Sri Raaja Rajan College of Engg. &
SRRCEIT/CSE/II/SEM
Amaravathipudur, Karaikudi - 630
Sivagangai Dist. Tamil Nadu



Sl. No.	Proposed Lecture		Topics to be covered	Text / Referenc e Book Page No	Teaching Aids to be used			Actual Lecture		Remarks by the faculty	Steps taken to cover the incomplete portions, if any & signature of the Staff Member	Signature of Principal
	Date	Perio d			BB	OHP	PPT	Date	Perio d			
Unit – II												
DATA TYPES-ABSTRACTION												
1	15.2.21	1	Introduction - Primitive Data Types- Character String Types	R1/45								
2	16.2.21	3	User-Defined Ordinal Types- Array types- Associative Arrays	R1/49								
3	17.2.21	2	Record Types- Tuple Types-List Types -Union Types -	R1/56								
4	18.2.21	5	Pointer and Reference Types -Type Checking- Strong Typing	R1/68								
5	19.2.21	7	Type Equivalence - Theory and Data Types	R1/75								
6	22.2.21	1	- Variables-The Concept of Binding - Scope - Scope and Lifetime -	R1/78								
7	23.2.21	3	Referencing Environments - Named Constants	R1/83								
8	24.2.21	2	The Concept of Abstraction- Parameterized Abstract Data Types	R1/84								
9	25.2.21	6	Encapsulation Constructs- Naming Encapsulations	R1/86								
											Signature of HOD	Signature of Principal

Signature of HOD

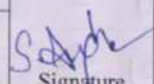

Signature of Principal

PRINCIPAL

Sri Raaja Raajan College of Engineering
Amaravathipudur, Karaikudi -
Sivagangai Dist. Tamil Nadu



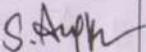

SRR CET/CSE/LP/SEM

Sl. No.	Proposed Lecture		Topics to be covered	Text / Reference Book Page No	Teaching Aids to be used			Actual Lecture		Remarks by the faculty	Steps taken to cover the incomplete portions, if any & signature of the Staff Member	Signature of Principa
	Date	Period			BB	OHP	PPT	Date	Period			
Unit – III												
FUNCTIONAL PROGRAMMING												
1	16.3.21	3	Introduction-Mathematical Functions	R2/68							 Signature of HOD	 Signature of Principal
2	17.3.21	2	Fundamentals of Functional Programming Languages	R2/74								
3	18.3.21	5	The First Functional Programming Language: LISP- An Introduction to Scheme- Common LISP	R2/78								
4	19.3.21	7	Haskell-F# - ML : Implicit Types- Data Types	R2/83								
5	22.3.21	1	Exception Handling in ML	R2/85								
6	23.3.21	3	Functional Programming with Lists- Scheme, a Dialect of Lisp	R2/88								
7	24.3.21	2	The Structure of Lists- List Manipulation	R2/97								
8	25.3.21	5	A Motivating Example: Differentiation- Simplification of Expressions	R2/99								
9	26.3.21	7	Storage Allocation for Lists Introduction-Mathematical Functions	R2/103								



Sri Raaja Raajan College of Engg. & Tech.
Amaravathipudur, Karaikudi - 630 300
Sivagangai Dist. Tamil Nadu

SRR CET/CSE/LP/SEM

Sl. No.	Proposed Lecture		Topics to be covered	Text / Reference Book Page No	Teaching Aids to be used			Actual Lecture		Remarks by the faculty	Steps taken to cover the incomplete portions, if any & signature of the Staff Member	Signature of Principal
	Date	Period			BB	OHP	PPT	Date	Period			
Unit – IV LOGIC PROGRAMMING												
1	1.5.21	1	Relational Logic Programming- Syntax- Basics	R1/169							 Signature of HOD	 Signature of Principal
2	2.3.21	3	Facts- Rules- Syntax	R1/116								
3	3.3.21	2	Operational Semantics	R1/121								
4	4.3.21	5	Relational logic programs and SQL operations	R1/125								
5	5.3.21	7	Logic Programming- Syntax- Operational semantics	R1/138								
6	6.3.21	1	Data Structures-Meta-tools: Backtracking optimization (cuts); Unify;	R1/145								
7	9.3.21	3	Meta-circular interpreters- The Origins of Prolog	R1/149								
8	10.3.21	2	Elements- of Prolog-Deficiencies of Prolog	R1/162								
9	11.3.21	5	Applications of Logic Programming	R1/163								

S. Anish
Signature
of
HOD

Signature
of
Principal

PRINCIPAL

Sri Raaja Raajan College of Engg. & Tech.
Amaravathipuram, Karaikudi - 630 002
Sivagangai Dist. Tamil Nadu

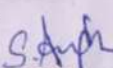
SRR CET/CSE/LP/SEM



Sl. No.	Proposed Lecture		Topics to be covered	Text / Reference Book Page No	Teaching Aids to be used			Actual Lecture		Remarks by the faculty	Steps taken to cover the incomplete portions, if any & signature of the Staff Member	Signature of Principal
	Date	Period			BB	OHP	PPT	Date	Period			
Unit-V												
CONCURRENT PROGRAMMING												
1	1.4.21	7	Parallelism in Hardware- Streams: Implicit Synchronization-Concurrency as Interleaving	R2/105								
2	2.4.21	7	Liveness Properties- Safe Access to Shared Data	R2/110								
3	8.4.21	5	Concurrency in Ada	R2/115								
4	9.4.21	7	Synchronized Access to Shared Variables	R2/150								
5	12.4.21	1	Synthesized Attributes- Attribute Grammars	R2/152								
6	13.4.21	3	Natural Semantics- Denotational Semantics	R2/153								
7	14.4.21	2	A Calculator in Scheme	R2/159								
8	15.4.21	8	Lexically Scoped Lambda Expressions	R2/172								
9	16.4.21	7	An Interpreter-Recursive Functions	R2/180								
											Signature of HOD	Signature of Principal



PRINCIPAL
 Sri Raaja Raajan College of Engg.
 Amaravathipudur, Karaikudi - 60
 SRR CET/CSE/EP/SEM

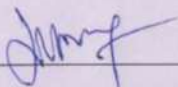

 Signature
 of
 HOD


 Signature
 of
 Principal

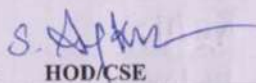
REFERENCES:

1. Ghezzi, "Programming Languages", 3rd Edition, John Wiley, 2008
2. John C. Mitchell, "Concepts in Programming Languages", Cambridge University Press, 2004.
3. Louden, "Programming Languages", 3rd Edition, 2012.
4. Ravi Sethi, "Programming Languages: Concepts and Constructs", 2nd Edition, Addison Wesley, 1996.
5. Robert W. Sebesta, "Concepts of Programming Languages", 10th Edition, Pearson Education, 2002.

Prepared by:



Verified by:


HOD/CSE

Approved by:


PRINCIPAL

Sri Raaja Raajan College of Engg. & Tech.
Amaravathipuram, Karaikudi - 630 30
Sivagangai Dist. Tamil Nadu



SRR CET/CSE/LP/SEM

UNIT-1

ELEMENTS OF PROGRAMMING LANGUAGES

Background

- Frankly, we didn't have the vaguest idea how the thing [FORTRAN language and compiler] would work out in detail. ...We struck out simply to optimize the object program, the running time, because most people at that time believed you couldn't do that kind of thing. They believed that machined-coded programs would be so inefficient that it would be impractical for many applications.
- John Backus, unexpected successes are common – the browser is another example of an unexpected success

1.1 Reasons for Studying Concepts of Programming Languages

- Increased ability to express ideas
- Improved background for choosing appropriate languages
- Increased ability to learn new languages
- Better understanding of significance of implementation
- Overall advancement of computing

Programming Domains

- Scientific applications
 - Large number of floating point computations
 - Fortran
- Business applications
 - Produce reports, use decimal numbers and characters
 - COBOL
- Artificial intelligence
 - Symbols rather than numbers manipulated
 - LISP
- Systems programming
 - Need efficiency because of continuous use
 - C
- Web Software
 - Eclectic collection of languages: markup (e.g., XHTML), scripting (e.g., PHP), general-purpose (e.g., Java)

1.2 Language Evaluation Criteria

- Readability : the ease with which programs can be read and understood
- Writability : the ease with which a language can be used to create programs
- Reliability : conformance to specifications (i.e., performs to its specifications)
- Cost : the ultimate total cost

Readability

- Overall simplicity
 - A manageable set of features and constructs
 - Few feature multiplicity (means of doing the same operation)
 - Minimal operator overloading
- Orthogonality
 - A relatively small set of primitive constructs can be combined in a relatively



- small number of ways
 - Every possible combination is legal
- Control statements
 - The presence of well-known control structures (e.g., while statement)
- Data types and structures
 - The presence of adequate facilities for defining data structures
- Syntax considerations
 - Identifier forms: flexible composition
 - Special words and methods of forming compound statements
 - Form and meaning: self-descriptive constructs, meaningful keywords

Writability

- Simplicity and Orthogonality
 - Few constructs, a small number of primitives, a small set of rules for combining them
- Support for abstraction
 - The ability to define and use complex structures or operations in ways that allow details to be ignored
- Expressivity
 - A set of relatively convenient ways of specifying operations
 - Example: the inclusion of for statement in many modern languages

Reliability

- Type checking
 - Testing for type errors
- Exception handling
 - Intercept run-time errors and take corrective measures
- Aliasing
 - Presence of two or more distinct referencing methods for the same memory location
- Readability and writability
 - A language that does not support "natural" ways of expressing an algorithm will necessarily use "unnatural" approaches, and hence reduced reliability

Cost

- Training programmers to use language
- Writing programs (closeness to particular applications)
- Compiling programs
- Executing programs
- Language implementation system: availability of free compilers
- Reliability: poor reliability leads to high costs
- Maintaining programs

Others

- Portability
 - The ease with which programs can be moved from one implementation to another
- Generality
 - The applicability to a wide range of applications
- Well-definedness
 - The completeness and precision of the language's official definition

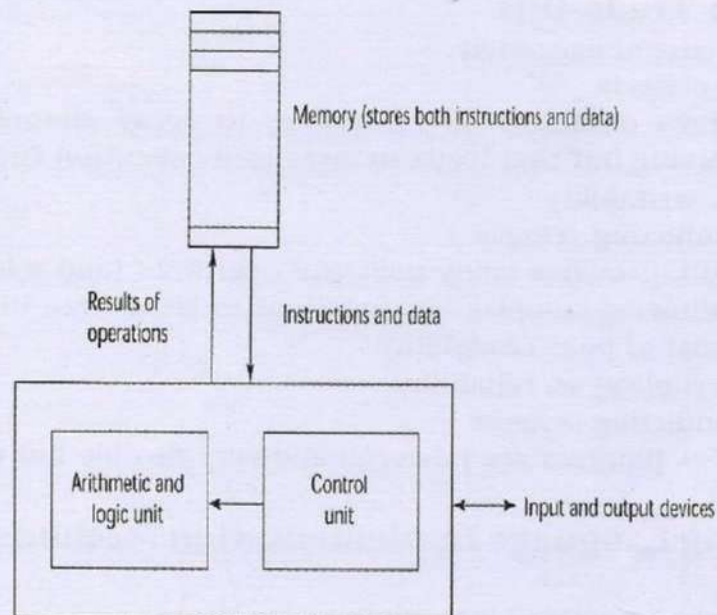


1.3 Influences on Language Design

- Computer Architecture
 - Languages are developed around the prevalent computer architecture, known as the *von Neumann* architecture
- Programming Methodologies
 - New software development methodologies (e.g., object-oriented software development) led to new programming paradigms and by extension, new programming languages

Computer Architecture

- Well-known computer architecture: Von Neumann
- Imperative languages, most dominant, because of von Neumann computers
 - Data and programs stored in memory
 - Memory is separate from CPU
 - Instructions and data are piped from memory to CPU
 - Basis for imperative languages
- Variables model memory cells
- Assignment statements model piping
- Iteration is efficient



Central processing unit
Figure 1.1 The von Neumann Computer Architecture

Programming Methodologies

- 1950s and early 1960s: Simple applications; worry about machine efficiency
- Late 1960s: People efficiency became important; readability, better control structures
 - structured programming
 - top-down design and step-wise refinement
- Late 1970s: Process-oriented to data-oriented
 - data abstraction
- Middle 1980s: Object-oriented programming
 - Data abstraction + inheritance + polymorphism



1.4 Language Categories

- Imperative
 - Central features are variables, assignment statements, and iteration
 - Examples: C, Pascal
- Functional
 - Main means of making computations is by applying functions to given parameters
 - Examples: LISP, Scheme
- Logic
 - Rule-based (rules are specified in no particular order)
 - Example: Prolog
- Object-oriented
 - Data abstraction, inheritance, late binding
 - Examples: Java, C++
- Markup
 - New; not a programming per se, but used to specify the layout of information in Web documents
 - Examples: XHTML, XML

Language Design Trade-Offs

- Reliability vs. cost of execution
 - Conflicting criteria
 - Example: Java demands all references to array elements be checked for proper indexing but that leads to increased execution costs
- Readability vs. writability
 - Another conflicting criteria
 - Example: APL provides many powerful operators (and a large number of new symbols), allowing complex computations to be written in a compact program but at the cost of poor readability
- Writability (flexibility) vs. reliability
 - Another conflicting criteria
 - Example: C++ pointers are powerful and very flexible but not reliably used

1.5 Programming Language Implementation Methods

- Compilation
 - Programs are translated into machine language
- Pure Interpretation
 - Programs are interpreted by another program known as an interpreter
- Hybrid Implementation Systems
 - A compromise between compilers and pure interpreters

1.5.1 Compilation

- Translate high-level program (source language) into machine code (machine language)
- Slow translation, fast execution
- Compilation process has several phases:
 - lexical analysis: converts characters in the source program into lexical units
 - syntax analysis: transforms lexical units into *parse trees* which represent the syntactic structure of program
 - Semantics analysis: generate intermediate code
 - code generation: machine code is generated

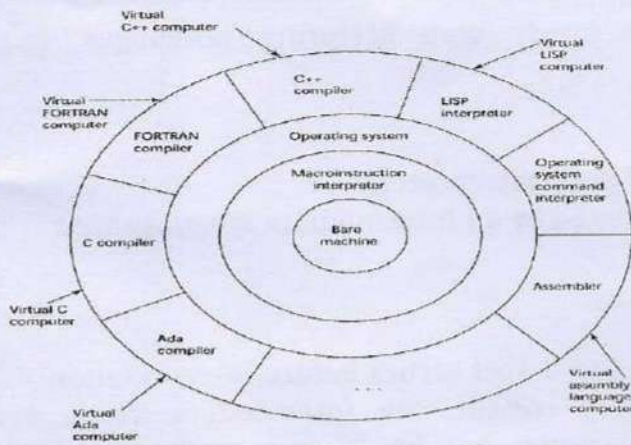


Figure Layered View of Computer: The operating system and language implementation are layered over Machine interface of a computer

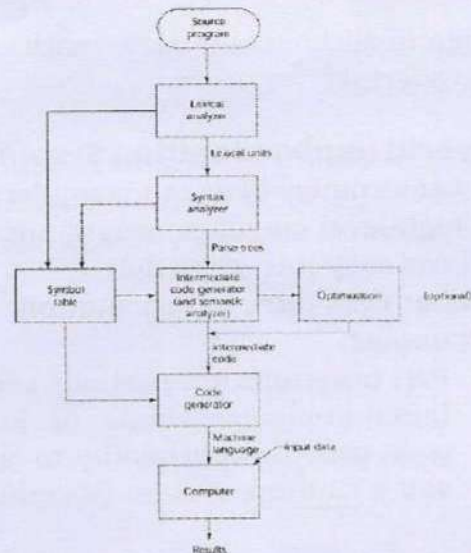


Figure The Compilation Process

1.5.2 Virtual Machines-Additional Compilation Terminologies

- Load module (executable image): the user and system code together
- Linking and loading: the process of collecting system program and linking them to user program

Execution of Machine Code

- Fetch-execute-cycle (on a von Neumann architecture)

initialize the program counter

repeat forever

fetch the instruction pointed by the counter

increment the counter

decode the instruction

execute the instruction

end repeat

Von Neumann Bottleneck

- Connection speed between a computer's memory and its processor determines the speed of a computer
- Program instructions often can be executed a lot faster than the above connection speed; the connection speed thus results in a *bottleneck*
- Known as von Neumann bottleneck; it is the primary limiting factor in the speed of computers

1.5.3 Pure Interpretation

- No translation
- Easier implementation of programs (run-time errors can easily and immediately displayed)
- Slower execution (10 to 100 times slower than compiled programs)
- Often requires more space
- Becoming rare on high-level languages



- Significantly comeback with some latest web scripting languages (e.g., JavaScript)

1.5.4 Hybrid Implementation Systems

- A compromise between compilers and pure interpreters
- A high-level language program is translated to an intermediate language that allows easy interpretation
- Faster than pure interpretation
- Examples
 - Perl programs are partially compiled to detect errors before interpretation
 - Initial implementations of Java were hybrid; the intermediate form, *byte code*, provides portability to any machine that has a byte code interpreter and a runtime system (together, these are called *Java Virtual Machine*)

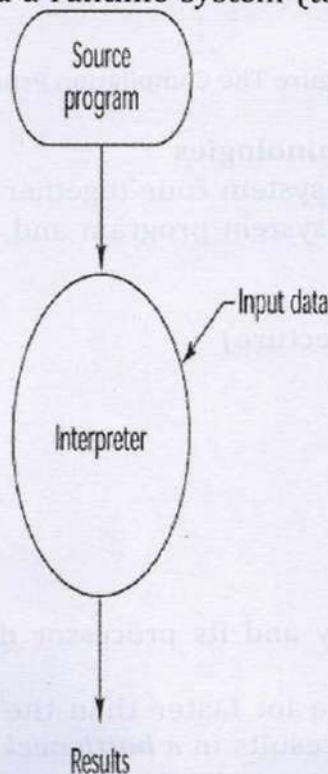


Figure Pure Interpretation

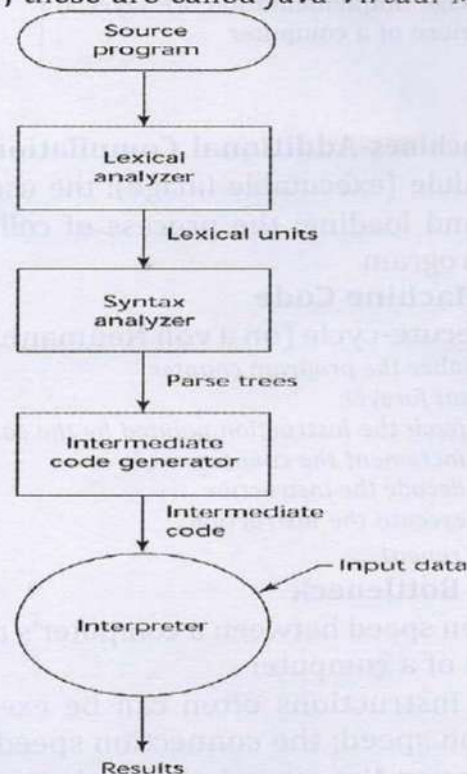


Figure Hybrid Implementation

Just-in-Time Implementation Systems

- Initially translate programs to an intermediate language
- Then compile intermediate language into machine code
- Machine code version is kept for subsequent calls
- JIT systems are widely used for Java programs
- .NET languages are implemented with a JIT system

Preprocessors

- Preprocessor macros (instructions) are commonly used to specify that code from another file is to be included
- A preprocessor processes a program immediately before the program is compiled to expand embedded preprocessor macros
- A well-known example: C preprocessor
 - expands #include, #define, and similar macros

1.6 Describing Syntax and Semantics - Introduction

- i. **Syntax:** the form or structure of the expressions, statements, and program units
- ii. **Semantics:** the meaning of the expressions, statements, and program units
- iii. Syntax and semantics provide a language's definition
 1. Users of a language definition
 2. Other language designers
 3. Implementers
 4. Programmers (the users of the language)

1.7 The General Problem of Describing Syntax

- A *sentence* is a string of characters over some alphabet
- A *language* is a set of sentences
- A *lexeme* is the lowest level syntactic unit of a language (e.g., *, sum, begin)
- A *token* is a category of lexemes (e.g., identifier)
- **Languages Recognizers**
 - A recognition device reads input strings of the language and decides whether the input strings belong to the language
 - Example: syntax analysis part of a compiler
- **Languages Generators**
 - A device that generates sentences of a language
 - One can determine if the syntax of a particular sentence is correct by comparing it to the structure of the generator

1.8 Formal Methods of Describing Syntax

- Backus-Naur Form and Context-Free Grammars
 - Most widely known method for describing programming language syntax
- Extended BNF
 - Improves readability and writability of BNF
- Grammars and Recognizers

Backus-Naur Form and Context-Free Grammars

- Context-Free Grammars
- Developed by Noam Chomsky in the mid-1950s
- Language generators, meant to describe the syntax of natural languages
- Define a class of languages called context-free languages

Backus-Naur Form (BNF)

- Backus-Naur Form (1959)
 - Invented by John Backus to describe ALGOL 58
 - BNF is equivalent to context-free grammars
 - BNF is a *metalanguage* used to describe another language
 - In BNF, abstractions are used to represent classes of syntactic structures-- they act like syntactic variables (also called *nonterminal symbols*)

BNF Fundamentals

- Non-terminals: BNF abstractions
- Terminals: lexemes and tokens
- Grammar: a collection of rules
 - Examples of BNF rules:



$\langle \text{ident_list} \rangle \rightarrow \text{identifier} \mid \text{identifier}, \langle \text{ident_list} \rangle$
 $\langle \text{if_stmt} \rangle \rightarrow \text{if } \langle \text{logic_expr} \rangle \text{ then } \langle \text{stmt} \rangle$

BNF Rules

- A rule has a left-hand side (LHS) and a right-hand side (RHS), and consists of *terminal* and *nonterminal* symbols
- A grammar is a finite nonempty set of rules
- An abstraction (or nonterminal symbol) can have more than one RHS

$\langle \text{stmt} \rangle \rightarrow \langle \text{single_stmt} \rangle$
 $\mid \text{begin } \langle \text{stmt_list} \rangle \text{ end}$

Describing Lists

- Syntactic lists are described using recursion

$\langle \text{ident_list} \rangle \rightarrow \text{ident}$
 $\mid \text{ident}, \langle \text{ident_list} \rangle$

- A derivation is a repeated application of rules, starting with the start symbol and ending with a sentence (all terminal symbols)

An Example Grammar

$\langle \text{program} \rangle \rightarrow \langle \text{stmts} \rangle$
 $\langle \text{stmts} \rangle \rightarrow \langle \text{stmt} \rangle \mid \langle \text{stmt} \rangle ; \langle \text{stmts} \rangle$
 $\langle \text{stmt} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expr} \rangle$
 $\langle \text{var} \rangle \rightarrow a \mid b \mid c \mid d$
 $\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle + \langle \text{term} \rangle \mid \langle \text{term} \rangle - \langle \text{term} \rangle$
 $\langle \text{term} \rangle \rightarrow \langle \text{var} \rangle \mid \text{const}$

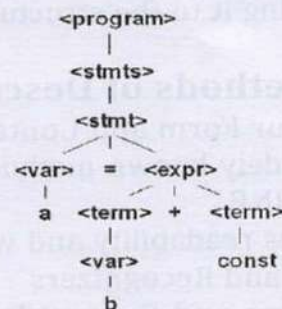
Parse Tree

A hierarchical representation of a derivation

An example derivation

$\langle \text{program} \rangle \Rightarrow \langle \text{stmts} \rangle$
 $\Rightarrow \langle \text{stmt} \rangle$
 $\Rightarrow \langle \text{var} \rangle = \langle \text{expr} \rangle$
 $\Rightarrow a = \langle \text{expr} \rangle$
 $\Rightarrow a = \langle \text{term} \rangle + \langle \text{term} \rangle$
 $\Rightarrow a = \langle \text{var} \rangle + \langle \text{term} \rangle$
 $\Rightarrow a = b + \langle \text{term} \rangle$
 $\Rightarrow a = b + \text{const}$

Figure 2.1 Parse Tree



Derivation

- Every string of symbols in the derivation is a sentential form
- A sentence is a sentential form that has only terminal symbols
- A leftmost derivation is one in which the leftmost nonterminal in each sentential form is the one that is expanded
- A derivation may be neither leftmost nor rightmost

Ambiguity in Grammars

- A grammar is *ambiguous* iff it generates a sentential form that has two or more distinct parse trees

An Unambiguous Expression Grammar

If we use the parse tree to indicate precedence levels of the operators, we cannot have ambiguity

$\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle - \langle \text{term} \rangle \mid \langle \text{term} \rangle$
 $\langle \text{term} \rangle \rightarrow \langle \text{term} \rangle / \text{const} \mid \text{const}$



1.9 Attribute Grammars

- Context-free grammars (CFGs) cannot describe all of the syntax of programming languages
- Additions to CFGs to carry some semantic info along parse trees
- Primary value of attribute grammars (AGs):
 - Static semantics specification
 - Compiler design (static semantics checking)

Definition

- An attribute grammar is a context-free grammar $G = (S, N, T, P)$ with the following additions:
 - For each grammar symbol x there is a set $A(x)$ of attribute values
 - Each rule has a set of functions that define certain attributes of the nonterminals in the rule
 - Each rule has a (possibly empty) set of predicates to check for attribute consistency
 - Let $X_0 X_1 \dots X_n$ be a rule
 - Functions of the form $S(X_0) = f(A(X_1), \dots, A(X_n))$ define *synthesized attributes*
 - Functions of the form $I(X_i) = f(A(X_0), \dots, A(X_n))$, for $i \leq j \leq n$, define *inherited attributes*
 - Initially, there are *intrinsic attributes* on the leaves

Example

- Syntax
 - $\langle \text{assign} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expr} \rangle$
 - $\langle \text{expr} \rangle \rightarrow \langle \text{var} \rangle + \langle \text{var} \rangle \mid \langle \text{var} \rangle$
 - $\langle \text{var} \rangle \rightarrow A \mid B \mid C$
- actual_type: synthesized for $\langle \text{var} \rangle$ and $\langle \text{expr} \rangle$
- expected_type: inherited for $\langle \text{expr} \rangle$
- Syntax rule: $\langle \text{expr} \rangle \rightarrow \langle \text{var} \rangle[1] + \langle \text{var} \rangle[2]$
- Semantic rules: $\langle \text{expr} \rangle.\text{actual_type} \rightarrow \langle \text{var} \rangle[1].\text{actual_type}$
- Predicate: $\langle \text{var} \rangle[1].\text{actual_type} == \langle \text{var} \rangle[2].\text{actual_type}$
 $\langle \text{expr} \rangle.\text{expected_type} == \langle \text{expr} \rangle.\text{actual_type}$
- Syntax rule: $\langle \text{var} \rangle \rightarrow \text{id}$
- Semantic rule: $\langle \text{var} \rangle.\text{actual_type} \leftarrow \text{lookup}(\langle \text{var} \rangle.\text{string})$
- How are attribute values computed?
 - If all attributes were inherited, the tree could be decorated in top-down order.
 - If all attributes were synthesized, the tree could be decorated in bottom-up order.
 - In many cases, both kinds of attributes are used, and it is some combination of top-down and bottom-up that must be used.
 - $\langle \text{expr} \rangle.\text{expected_type} \leftarrow \text{inherited from parent}$
 - $\langle \text{var} \rangle[1].\text{actual_type} \leftarrow \text{lookup}(A)$
 - $\langle \text{var} \rangle[2].\text{actual_type} \leftarrow \text{lookup}(B)$
 - $\langle \text{var} \rangle[1].\text{actual_type} = ? \langle \text{var} \rangle[2].\text{actual_type}$
 - $\langle \text{expr} \rangle.\text{actual_type} \leftarrow \langle \text{var} \rangle[1].\text{actual_type}$
 - $\langle \text{expr} \rangle.\text{actual_type} = ? \langle \text{expr} \rangle.\text{expected_type}$



1.10 Describing the Meanings of Programs: Dynamic Semantics

- There is no single widely acceptable notation or formalism for describing semantics
- Operational Semantics

- Describe the meaning of a program by executing its statements on a machine, either simulated or actual. The change in the state of the machine (memory, registers, etc.) defines the meaning of the statement
- To use operational semantics for a high-level language, a virtual machine is needed
- A hardware pure interpreter would be too expensive
- A software pure interpreter also has problems:
 - The detailed characteristics of the particular computer would make actions difficult to understand
 - Such a semantic definition would be machine- dependent

operational Semantics

- A better alternative: A complete computer simulation
- The process:
 - Build a translator (translates source code to the machine code of an idealized computer)
 - Build a simulator for the idealized computer
- Evaluation of operational semantics:
 - Good if used informally (language manuals, etc.)
 - Extremely complex if used formally (e.g., VDL), it was used for describing semantics of PL/I.
- Axiomatic Semantics
 - Based on formal logic (predicate calculus)
 - Original purpose: formal program verification
 - Approach: Define axioms or inference rules for each statement type in the language (to allow transformations of expressions to other expressions)
 - The expressions are called assertions

Axiomatic Semantics

- An assertion before a statement (a precondition) states the relationships and constraints among variables that are true at that point in execution
- An assertion following a statement is a postcondition
- A weakest precondition is the least restrictive precondition that will guarantee the postcondition
- Pre-post form: $\{P\}$ statement $\{Q\}$
- An example: $a = b + 1 \{a > 1\}$
- One possible precondition: $\{b > 10\}$
- Weakest precondition: $\{b > 0\}$
- Program proof process: The postcondition for the whole program is the desired result. Work back through the program to the first statement. If the precondition on the first statement is the same as the program spec, the program is correct.
- An axiom for assignment statements
 $(x = E):$
 $\{Qx \rightarrow E\} x = E \{Q\}$
- An inference rule for sequences
 - For a sequence $S1; S2:$
 - $\{P1\} S1 \{P2\}$
 - $\{P2\} S2 \{P3\}$
 - An inference rule for logical pretest loops
 For the loop construct:
 $\{P\} \text{ while } B \text{ do } S \text{ end } \{Q\}$

Characteristics of the loop invariant

I must meet the following conditions:

- $P \Rightarrow I$ (the loop invariant must be true initially)
- $\{I\} B \{I\}$ (evaluation of the Boolean must not change the validity of I)
- $\{I \text{ and } B\} S \{I\}$ (I is not changed by executing the body of the loop)
- $(I \text{ and } (\text{not } B)) \Rightarrow Q$ (if I is true and B is false, Q is implied)
- The loop terminates (this can be difficult to prove)
- The loop invariant I is a weakened version of the loop postcondition, and it is also a precondition.
- I must be weak enough to be satisfied prior to the beginning of the loop, but when combined with the loop exit condition, it must be strong enough to force the truth of the postcondition.

Evaluation of Axiomatic Semantics:

- Developing axioms or inference rules for all of the statements in a language is difficult
- It is a good tool for correctness proofs, and an excellent framework for reasoning about programs, but it is not as useful for language users and compiler writers
- Its usefulness in describing the meaning of a programming language is limited for language users or compiler writers

Denotational Semantics

- Based on recursive function theory
- The most abstract semantics description method
- Originally developed by Scott and Strachey (1970)
- The process of building a denotational spec for a language (not necessarily easy):
- Define a mathematical object for each language entity
- Define a function that maps instances of the language entities onto instances of the corresponding mathematical objects
- The meaning of language constructs are defined by only the values of the program's variables
- The difference between denotational and operational semantics: In operational semantics, the state changes are defined by coded algorithms; in denotational semantics, they are defined by rigorous mathematical functions
- The state of a program is the values of all its current variables
 $s = \{ \langle i1, v1 \rangle, \langle i2, v2 \rangle, \dots, \langle in, vn \rangle \}$
- Let VARMAP be a function that, when given a variable name and a state, returns the current value of the variable
 $\text{VARMAP}(ij, s) = vj$
- Decimal Numbers
 - The following denotational semantics description maps decimal numbers as strings of symbols into numeric values
 $\langle \text{dec_num} \rangle \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
 $\Rightarrow \langle \text{dec_num} \rangle (0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)$
 $M_{\text{dec}}('0') = 0, M_{\text{dec}}('1') = 1, \dots, M_{\text{dec}}('9') = 9$
 $M_{\text{dec}}(\langle \text{dec_num} \rangle '0') = 10 * M_{\text{dec}}(\langle \text{dec_num} \rangle)$
 $M_{\text{dec}}(\langle \text{dec_num} \rangle '1') = 10 * M_{\text{dec}}(\langle \text{dec_num} \rangle) + 1$
 \dots
 $M_{\text{dec}}(\langle \text{dec_num} \rangle '9') = 10 * M_{\text{dec}}(\langle \text{dec_num} \rangle) + 9$



UNIT-2

DATA TYPES-ABSTRACTION

Introduction

- A *data type* defines a collection of data objects and a set of predefined operations on those objects
- A *descriptor* is the collection of the attributes of a variable
- An *object* represents an instance of a user-defined (abstract data) type
- One design issue for all data types: What operations are defined and how are they specified?

2.1 Primitive Data Types

- Almost all programming languages provide a set of *primitive data types*
- Primitive data types: Those not defined in terms of other data types
- Some primitive data types are merely reflections of the hardware
- Others require only a little non-hardware support for their implementation

Integer

- Almost always an exact reflection of the hardware so the mapping is trivial
- There may be as many as eight different integer types in a language
- Java's signed integer sizes: byte, short, int, long

Floating Point

- Model real numbers, but only as approximations
- Languages for scientific use support at least two floating-point types (e.g., float and double; sometimes more)
- Usually exactly like the hardware, but not always
- IEEE Floating-Point Standard 754

Complex

- Some languages support a complex type, e.g., Fortran and Python
- Each value consists of two floats, the real part and the imaginary part
- Literal form (in Python):
(7 + 3j), where 7 is the real part and 3 is the imaginary part

Decimal

- For business applications (money)
 - Essential to COBOL
 - C# offers a decimal data type
- Store a fixed number of decimal digits, in coded form (BCD)
- *Advantage*: accuracy
- *Disadvantages*: limited range, wastes memory

Boolean

- Simplest of all
- Range of values: two elements, one for "true" and one for "false"
- Could be implemented as bits, but often as bytes
 - Advantage: readability

Character

- Stored as numeric coding
- Most commonly used coding: ASCII
- An alternative, 16-bit coding: Unicode

- Includes characters from most natural languages
- Originally used in Java
- C# and JavaScript also support Unicode

2.2 Character String Types

- Values are sequences of characters
- Design issues:
 - Is it a primitive type or just a special kind of array?
 - Should the length of strings be static or dynamic?

Operations

- Typical operations:
 - Assignment and copying
 - Comparison (=, >, etc.)
 - Catenation
 - Substring reference
 - Pattern matching

Character String Type in Certain Languages

- C and C++
 - Not primitive
 - Use **char** arrays and a library of functions that provide operations
- SNOBOL4 (a string manipulation language)
 - Primitive
 - Many operations, including elaborate pattern matching
- Fortran and Python
 - Primitive type with assignment and several operations
- Java
 - Primitive via the String class
- Perl, JavaScript, Ruby, and PHP
 - Provide built-in pattern matching, using regular expressions

Character String Length Options

- Static: COBOL, Java's String class
- *Limited Dynamic Length*: C and C++
 - In these languages, a special character is used to indicate the end of a string's characters, rather than maintaining the length
- *Dynamic* (no maximum): SNOBOL4, Perl, JavaScript
- Ada supports all three string length options

Evaluation

- Aid to writability
- As a primitive type with static length, they are inexpensive to provide—why not have them?
- Dynamic length is nice, but is it worth the expense?

Implementation

- Static length: compile-time descriptor
- Limited dynamic length: may need a run-time descriptor for length (but not in C and C++)
- Dynamic length: need run-time descriptor; allocation/de-allocation is the biggest implementation problem

Static string
Length
Address

Figure 3.1 Compile-Time Descriptor

Limited dynamic string
Maximum length
Current length
Address

Figure 3.2 Run-Time Descriptors

2.3 User-Defined Ordinal Types - CO2

- An ordinal type is one in which the range of possible values can be easily associated with the set of positive integers
- Examples of primitive ordinal types in Java
 - integer
 - char
 - boolean

Enumeration Types

- All possible values, which are named constants, are provided in the definition
- C# example


```
enum days {mon, tue, wed, thu, fri, sat, sun};
```
- Design issues
 - Is an enumeration constant allowed to appear in more than one type definition, and if so, how is the type of an occurrence of that constant checked?
 - Are enumeration values coerced to integer?
 - Any other type coerced to an enumeration type?

Evaluation of Enumerated Type

- Aid to readability, e.g., no need to code a color as a number
- Aid to reliability, e.g., compiler can check:
 - Operations (don't allow colors to be added)
 - No enumeration variable can be assigned a value outside its defined range
 - Ada, C#, and Java 5.0 provide better support for enumeration than C++ because enumeration type variables in these languages are not coerced into integer types

Subrange Types

- An ordered contiguous subsequence of an ordinal type
 - Example: 12..18 is a subrange of integer type
- Ada's design

```
type Days is (mon, tue, wed, thu, fri, sat, sun);
subtype Weekdays is Days range mon..fri;
subtype Index is Integer range 1..100;
```

```
Day1: Days;
Day2: Weekday;
Day2 := Day1;
```

Subrange Evaluation

- Aid to readability
 - Make it clear to the readers that variables of subrange can store only certain range of values
- Reliability
 - Assigning a value to a subrange variable that is outside the specified range is detected as an error

Implementation

- Enumeration types are implemented as integers
- Subrange types are implemented like the parent types with code inserted (by the compiler) to restrict assignments to subrange variables

2.4 Array Types

- An array is an aggregate of homogeneous data elements in which an individual element is identified by its position in the aggregate, relative to the first element.

Array Design Issues

- What types are legal for subscripts?
- Are subscripting expressions in element references range checked?
- When are subscript ranges bound?
- When does allocation take place?
- What is the maximum number of subscripts?
- Can array objects be initialized?
- Are any kind of slices supported?

Array Indexing

- *Indexing* (or subscripting) is a mapping from indices to elements
 $\text{array_name}(\text{index_value_list}) \rightarrow \text{an element}$
- Index Syntax
 - FORTRAN, PL/I, Ada use parentheses
- Ada explicitly uses parentheses to show uniformity between array references and function calls because both are *mappings*
 - Most other languages use brackets

Arrays Index (Subscript) Types

- FORTRAN, C: integer only
- Ada: integer or enumeration (includes Boolean and char)
- Java: integer types only
- Index range checking
 - C, C++, Perl, and Fortran do not specify range checking
 - Java, ML, C# specify range checking
 - In Ada, the default is to require range checking, but it can be turned off

Subscript Binding and Array Categories

- *Static*: subscript ranges are statically bound and storage allocation is static (before run-time)
 - Advantage: efficiency (no dynamic allocation)
- *Fixed stack-dynamic*: subscript ranges are statically bound, but the allocation is done at declaration time
 - Advantage: space efficiency
- *Stack-dynamic*: subscript ranges are dynamically bound and the storage allocation is dynamic (done at run-time)
 - Advantage: flexibility (the size of an array need not be known until the array is to be used)
- *Fixed heap-dynamic*: similar to fixed stack-dynamic: storage binding is dynamic but fixed after allocation (i.e., binding is done when requested and storage is allocated from heap, not stack)
- *Heap-dynamic*: binding of subscript ranges and storage allocation is dynamic and can change any number of times
 - Advantage: flexibility (arrays can grow or shrink during program execution)
- C and C++ arrays that include static modifier are static
- C and C++ arrays without static modifier are fixed stack-dynamic
- C and C++ provide fixed heap-dynamic arrays
- C# includes a second array class `ArrayList` that provides fixed heap-dynamic
- Perl, JavaScript, Python, and Ruby support heap-dynamic arrays

Array Initialization

- Some language allow initialization at the time of storage allocation

- C, C++, Java, C# example
`int list [] = {4, 5, 7, 83}`
- Character strings in C and C++
`char name [] = "freddie";`
- Arrays of strings in C and C++
`char *names [] = {"Bob", "Jake", "Joe"};`
- Java initialization of String objects
`String[] names = {"Bob", "Jake", "Joe"};`

Heterogeneous Arrays

- A *heterogeneous array* is one in which the elements need not be of the same type
- Supported by Perl, Python, JavaScript, and Ruby

Arrays Operations

- APL provides the most powerful array processing operations for vectors and matrixes as well as unary operators (for example, to reverse column elements)
- Ada allows array assignment but also catenation
- Python's array assignments, but they are only reference changes. Python also supports array catenation and element membership operations
- Ruby also provides array catenation
- Fortran provides *elemental* operations because they are between pairs of array elements
 - For example, + operator between two arrays results in an array of the sums of the element pairs of the two arrays

Rectangular and Jagged Arrays

- A rectangular array is a multi-dimensioned array in which all of the rows have the same number of elements and all columns have the same number of elements
- A jagged matrix has rows with varying number of elements
 - Possible when multi-dimensioned arrays actually appear as arrays of arrays
- C, C++, and Java support jagged arrays
- Fortran, Ada, and C# support rectangular arrays (C# also supports jagged arrays)

Slices

- A slice is some substructure of an array; nothing more than a referencing mechanism
- Slices are only useful in languages that have array operations

Slice Examples

- Fortran 95

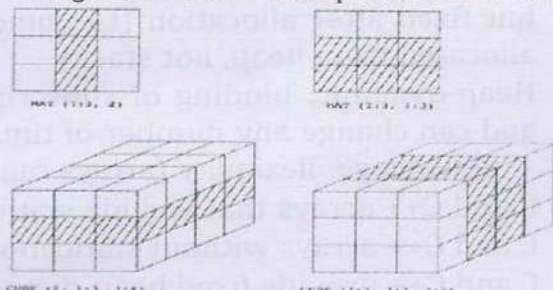
Integer, Dimension (10) :: Vector

Integer, Dimension (3, 3) :: Mat

Integer, Dimension (3, 3) :: Cube

Vector (3:6) is a four element array

Figure 3.3 Slices Examples in Fortran 95



Implementation of Arrays

- Access function maps subscript expressions to an address in the array
- Access function for single-dimensioned arrays:
$$\text{address}(\text{list}[k]) = \text{address}(\text{list}[\text{lower_bound}]) + ((k - \text{lower_bound}) * \text{element_size})$$

Accessing Multi-dimensioned Arrays

- Two common ways:
 - Row major order (by rows) – used in most languages
 - Column major order (by columns) – used in Fortran

2.5 Associative Arrays

- An *associative array* is an unordered collection of data elements that are indexed by an equal number of values called *keys*
 - User-defined keys must be stored
- Design issues:
 - What is the form of references to elements?
 - Is the size static or dynamic?

Associative Arrays in Perl

- Names begin with %; literals are delimited by parentheses
`%hi_temps = ("Mon" => 77, "Tue" => 79, "Wed" => 65, ...);`
- Subscripting is done using braces and keys
`$hi_temps{"Wed"} = 83;`
- Elements can be removed with delete
`delete $hi_temps{"Tue"};`

2.6 Record Types - Tuple Types & List Types

- A *record* is a possibly heterogeneous aggregate of data elements in which the individual elements are identified by names
- Design issues:
 - What is the syntactic form of references to the field?
 - Are elliptical references allowed?

Definition of Records in COBOL

- COBOL uses level numbers to show nested records; others use recursive definition

```
01 EMP-REC.
  02 EMP-NAME.
    05 FIRST PIC X(20).
    05 MID PIC X(10).
    05 LAST PIC X(20).
  02 HOURLY-RATE PIC 99V99.
```

Definition of Records in Ada

- Record structures are indicated in an orthogonal way

```
type Emp_Rec_Type is record
  First: String (1..20);
  Mid: String (1..10);
  Last: String (1..20);
  Hourly_Rate: Float;
end record;
Emp_Rec: Emp_Rec_Type;
```

References to Records

- Record field references
 - COBOL
`field_name OF record_name_1 OF ... OF record_name_n`
 - Others (dot notation)
`record_name_1.record_name_2 record_name_n.field_name`
- Fully qualified references must include all record names
- Elliptical references allow leaving out record names as long as the reference is unambiguous, for example in COBOL
`FIRST, FIRST OF EMP-NAME, and FIRST of EMP-REC` are elliptical



references to the employee's first name

Operations on Records

- Assignment is very common if the types are identical
- Ada allows record comparison
- Ada records can be initialized with aggregate literals
- COBOL provides MOVE CORRESPONDING
 - Copies a field of the source record to the corresponding field in the target record

Evaluation and Comparison to Arrays

- Records are used when collection of data values is heterogeneous
- Access to array elements is much slower than access to record fields, because subscripts are dynamic (fieldnames are static)
- Dynamic subscripts could be used with record field access, but it would disallow type checking and it would be much slower

2.7 Unions Types

A *union* is a type whose variables are allowed to store different type values at different times during execution

- Design issues
 - Should type checking be required?
 - Should unions be embedded in records?

Discriminated vs. Free Unions

- Fortran, C, and C++ provide union constructs in which there is no language support for type checking; the union in these languages is called *free union*
- Type checking of unions require that each union include a type indicator called a *discriminant*
 - Supported by Ada

Ada Union Types type Shape is (Circle, Triangle, Rectangle); type Colors is (Red, Green, Blue);
 type Figure (Form: Shape) is record Filled: Boolean;
 Color: Colors; case Form is
 when Circle => Diameter: Float;
 when Triangle => Leftside, Rightside: Integer; Angle: Float;
 when Rectangle => Side1, Side2: Integer; end case;
 end record;

Evaluation of Unions

- Free unions are unsafe
 - Do not allow type checking
 - Java and C# do not support unions
- Reflective of growing concerns for safety in programming language
- Ada's discriminated unions are safe

2.8 Pointer and Reference Types

- A *pointer* type variable has a range of values that consists of memory addresses and a special value, *nil*
- Provide the power of indirect addressing
- Provide a way to manage dynamic memory
- A pointer can be used to access a location in the area where storage is dynamically created (usually called a *heap*)

Design Issues of Pointers

- What are the scope of and lifetime of a pointer variable?
- What is the lifetime of a heap-dynamic variable?
- Are pointers restricted as to the type of value to which they can point?
- Are pointers used for dynamic storage management, indirect addressing, or both?
- Should the language support pointer types, reference types, or both?

Pointer Operations

- Two fundamental operations: assignment and dereferencing
- Assignment is used to set a pointer variable's value to some useful address
- Dereferencing yields the value stored at the location represented by the pointer's value
 - Dereferencing can be explicit or implicit
 - C++ uses an explicit operation via `*`

`j = *ptr`

sets `j` to the value located at `ptr`

Pointer Assignment Illustration

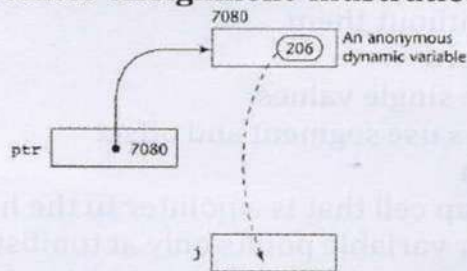


Figure The assignment operation `j = *ptr`

Problems with Pointers

- Dangling pointers (dangerous)
 - A pointer points to a heap-dynamic variable that has been deallocated
- Lost heap-dynamic variable
 - An allocated heap-dynamic variable that is no longer accessible to the user program (often called *garbage*)
- Pointer `p1` is set to point to a newly created heap-dynamic variable
- Pointer `p1` is later set to point to another newly created heap-dynamic variable
- The process of losing heap-dynamic variables is called *memory leakage*

Pointers in Ada

- Some dangling pointers are disallowed because dynamic objects can be automatically deallocated at the end of pointer's type scope
- The lost heap-dynamic variable problem is not eliminated by Ada (possible with `UNCHECKED_DEALLOCATION`)

Pointers in C and C++

- Extremely flexible but must be used with care
- Pointers can point at any variable regardless of when or where it was allocated
- Used for dynamic storage management and addressing
- Pointer arithmetic is possible
- Explicit dereferencing and address-of operators
- Domain type need not be fixed (**`void *`**)
 - `void *` can point to any type and can be type checked (cannot be de-referenced)

Pointer Arithmetic in C and C++

```
float stuff[100];
float *p;
```



$p = \text{stuff};$

$*(p+5)$ is equivalent to $\text{stuff}[5]$ and $p[5]$

$*(p+i)$ is equivalent to $\text{stuff}[i]$ and $p[i]$

Reference Types

- C++ includes a special kind of pointer type called a *reference type* that is used primarily for formal parameters
 - Advantages of both pass-by-reference and pass-by-value
- Java extends C++'s reference variables and allows them to replace pointers entirely
 - References are references to objects, rather than being addresses
- C# includes both the references of Java and the pointers of C++

Evaluation of Pointers

- Dangling pointers and dangling objects are problems as is heap management
- Pointers are like goto's--they widen the range of cells that can be accessed by a variable
- Pointers or references are necessary for dynamic data structures--so we can't design a language without them

Representations of Pointers

- Large computers use single values
- Intel microprocessors use segment and offset

Dangling Pointer Problem

- *Tombstone*: extra heap cell that is a pointer to the heap-dynamic variable
 - The actual pointer variable points only at tombstones
 - When heap-dynamic variable de-allocated, tombstone remains but set to nil
 - Costly in time and space
- *Locks-and-keys*: Pointer values are represented as (key, address) pairs
 - Heap-dynamic variables are represented as variable plus cell for integer lock value
 - When heap-dynamic variable allocated, lock value is created and placed in lock cell and key cell of pointer

Heap Management

- A very complex run-time process
- Single-size cells vs. variable-size cells
- Two approaches to reclaim garbage
 - Reference counters (*eager approach*): reclamation is gradual
 - Mark-sweep (*lazy approach*): reclamation occurs when the list of variable space becomes empty

Reference Counter

- Reference counters: maintain a counter in every cell that store the number of pointers currently pointing at the cell

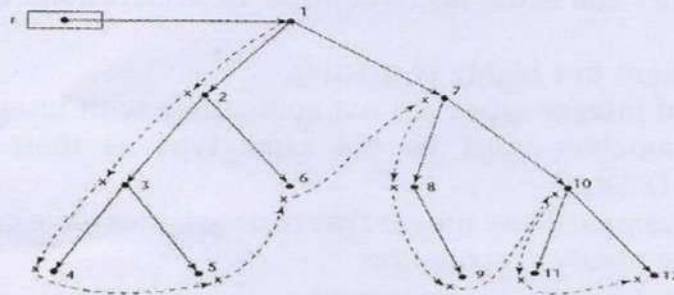
Disadvantages: space required, execution time required, complications for cells connected circularly

Advantage: it is intrinsically incremental, so significant delays in the application execution are avoided

Mark-Sweep

- The run-time system allocates storage cells as requested and disconnects pointers from cells as necessary; mark-sweep then begins
 - Every heap cell has an extra bit used by collection algorithm
 - All cells initially set to garbage
 - All pointers traced into heap, and reachable cells marked as not garbage
 - All garbage cells returned to list of available cells

- Disadvantages: in its original form, it was done too infrequently. When done, it caused significant delays in application execution. Contemporary mark-sweep algorithms avoid this by doing it more often—called incremental mark-sweep



Dashed lines show the order of node marking
Figure 3.10 Marking Algorithm

Variable-Size Cells

- All the difficulties of single-size cells plus more
- Required by most programming languages
- If mark-sweep is used, additional problems occur
 - The initial setting of the indicators of all cells in the heap is difficult
 - The marking process is nontrivial
 - Maintaining the list of available space is another source of overhead

2.8 Type Checking

- Generalize the concept of operands and operators to include subprograms and assignments
- Type checking is the activity of ensuring that the operands of an operator are of compatible types
- A compatible type is one that is either legal for the operator, or is allowed under language rules to be implicitly converted, by compiler-generated code, to a legal type. This automatic conversion is called as coercion.
- A type error is the application of an operator to an operand of an inappropriate type
- If all type bindings are static, nearly all type checking can be static
- If type bindings are dynamic, type checking must be dynamic
- Def: A programming language is strongly typed if type errors are always detected

2.9 Strong Typing -

- Advantage of strong typing: allows the detection of the misuses of variables that result in type errors
- Language examples:
 - ☐ FORTRAN 77 is not: parameters, *EQUIVALENCE*
 - Pascal is not: variant records
 - C and C++ are not: parameter type checking can be avoided; unions are not type checked
 - Ada is, almost (*UNCHECKED CONVERSION* is loophole)
(Java is similar)
- Coercion rules strongly affect strong typing—they can weaken it considerably (C++ versus Ada)
- Although Java has just half the assignment coercions of C++, its strong typing is still far less effective than that of Ada



2.10 Type Equivalence

- Our concern is primarily for structured types
- Def: Name type compatibility means the two variables have compatible types if they are in either the same declaration or in declarations that use the same type name
- Easy to implement but highly restrictive:
 - Subranges of integer types are not compatible with integer types
 - Formal parameters must be the same type as their corresponding actual parameters (Pascal)
- Structure type compatibility means that two variables have compatible types if their types have identical structures
- More flexible, but harder to implement
- Consider the problem of two structured types:
 - Are two record types compatible if they are structurally the same but use different field names?
 - ☐ Are two array types compatible if they are the same except that the subscripts are different?
(e.g., [1..10] and [0..9])
 - Are two enumeration types compatible if their components are spelled differently?
 - With structural type compatibility, you cannot differentiate between types of the same structure (e.g., different units of speed, both float)
- Language examples:
 - ☐ Pascal: usually structure, but in some cases name is used (formal parameters)
 - C: structure, except for records
 - Ada: restricted form of name
 - o Derived types allow types with the same structure to be different
 - o Anonymous types are all unique, even in:
A, B : array (1..10) of INTEGER;

2.11 Theory and Data Types -Names

- Design issues for names:
 - Maximum length?
 - Are connector characters allowed?
 - Are names case sensitive?
 - Are special words reserved words or keywords?
- Length
 - If too short, they cannot be connotative
- Language examples:
 - ☐ FORTRAN I: maximum 6
 - COBOL: maximum 30
 - FORTRAN 90 and ANSI C: maximum 31
 - o Ada and Java: no limit, and all are significant
 - o C++: no limit, but implementors often impose one
- Connectors
 - Pascal, Modula-2, and FORTRAN 77 don't allow
 - Others do
- Case sensitivity
 - Disadvantage: readability (names that look alike are different)
 - o worse in C++ and Java because predefined names are mixed case (e.g.,

IndexOutOfBoundsException)

- C, C++, and Java names are case sensitive
- The names in other languages are not
- Special words
 - An aid to readability; used to delimit or separate statement clauses
 - Def: A keyword is a word that is special only in certain contexts
 - o i.e. in Fortran:
 - Real VarName (*Real is data type followed with a name, therefore Real is a keyword*)
 - Real = 3.4 (*Real is a variable*)
 - Disadvantage: poor readability
 - Def: A reserved word is a special word that cannot be used as a user-defined name

2.12 Variables

- A variable is an abstraction of a memory cell
- Variables can be characterized as a sextuple of attributes: (name, address, value, type, lifetime, and scope)
- Name - not all variables have them (anonymous)
- Address - the memory address with which it is associated (also called *l*-value)
 - A variable may have different addresses at different times during execution
 - A variable may have different addresses at different places in a program
 - If two variable names can be used to access the same memory location, they are called aliases
 - Aliases are harmful to readability (program readers must remember all of them)
- How aliases can be created:
 - Pointers, reference variables, C and C++ unions
 - Some of the original justifications for aliases are no longer valid; e.g., memory reuse in FORTRAN
 - Replace them with dynamic allocation
- Type - determines the range of values of variables and the set of operations that are defined for values of that type; in the case of floating point, type also determines the Precision
- Value - the contents of the location with which the variable is associated
- Abstract memory cell - the physical cell or collection of cells associated with a variable

2.13 The Concept of Binding- Scope, Scope and Lifetime

- The *l*-value of a variable is its address
- The *r*-value of a variable is its value
- Def: A binding is an association, such as between an attribute and an entity, or between an operation and a symbol
- Def: Binding time is the time at which a binding takes place
- Possible binding times:
 - Language design time--e.g., bind operator symbols to operations
 - Language implementation time--e.g., bind floating point type to a representation
 - Compile time--e.g., bind a variable to a type in C or Java
 - Load time--e.g., bind a FORTRAN 77 variable to a memory cell (or a C static

variable)

- Runtime--**e.g.**, bind a nonstatic local variable to a memory cell
- Def: A binding is static if it first occurs before run time and remains unchanged throughout program execution.
- Def: A binding is dynamic if it first occurs during execution or can change during execution of the program.
- Type Bindings
 - How is a type specified?
 - When does the binding take place?
 - If static, the type may be specified by either an explicit or an implicit declaration
- Def: An explicit declaration is a program statement used for declaring the types of variables
- Def: An implicit declaration is a default mechanism for specifying types of variables (the first appearance of the variable in the program)
- FORTRAN, PL/I, BASIC, and Perl provide implicit declarations
 - Advantage: writability
 - Disadvantage: reliability (less trouble with Perl)
- Dynamic Type Binding (JavaScript and PHP)
- Specified through an assignment statement **e.g.**, JavaScript


```
list = [2, 4.33, 6, 8];
list = 17.3;
```

 - Advantage: flexibility (generic program units)
 - Disadvantages:
 - High cost (dynamic type checking and interpretation)
 - Type error detection by the compiler is difficult
- Type Inferencing (ML, Miranda, and Haskell)
 - Rather than by assignment statement, types are determined from the context of the reference
- Storage Bindings & Lifetime
 - Allocation - getting a cell from some pool of available cells
 - Deallocation - putting a cell back into the pool
- ef: The lifetime of a variable is the time during which it is bound to a particular memory cell
- Categories of variables by lifetimes
 - Static--bound to memory cells before execution begins and remains bound to the same memory cell throughout execution.
 - e.g.**, all FORTRAN 77 variables, C static variables
 - Advantages: efficiency (direct addressing), history-sensitive subprogram support
 - Disadvantage: lack of flexibility (no recursion)
- Categories of variables by lifetimes
 - Stack-dynamic--Storage bindings are created for variables when their declaration statements are elaborated.
 - If scalar, all attributes except address are statically bound
 - e.g.**, local variables in C subprograms and Java methods
 - Advantage: allows recursion; conserves storage
 - Disadvantages:
 - Overhead of allocation and deallocation
 - Subprograms cannot be history sensitive
 - Inefficient references (indirect addressing)

3.6 ML

- A static-scoped functional language with syntax that is closer to Pascal than to LISP
- Uses type declarations, but also does *type inferencing* to determine the types of undeclared variables
- It is strongly typed (whereas Scheme is essentially typeless) and has no type coercions
- Includes exception handling and a module facility for implementing abstract data types
- Includes lists and list operations

ML Specifics

- Function declaration form:

$$\text{fun name (parameters) = body;}$$
 e.g., $\text{fun cube (x : int) = x * x * x;}$
 - The type could be attached to return value, as in

$$\text{fun cube (x) : int = x * x * x;}$$
 - With no type specified, it would default to
 $\text{int (the default for numeric values)}$
 - User-defined overloaded functions are not allowed, so if we wanted a cube function for real parameters, it would need to have a different name
 - There are no type coercions in ML
- ML selection

$$\text{if expression then then_expression}$$

$$\text{else else_expression}$$

where the first expression must evaluate to a Boolean value

- Pattern matching is used to allow a function to operate on different parameter forms

```
fun fact(0) = 1
  | fact(n : int) : int = n * fact(n - 1)
```

- Lists

Literal lists are specified in brackets

[3, 5, 7]

[] is the empty list

CONS is the binary infix operator, ::

4 :: [3, 5, 7], which evaluates to [4, 3, 5, 7]

CAR is the unary operator hd

CDR is the unary operator tl

fun length([]) = 0

| length(h :: t) = 1 + length(t);

fun append([], lis2) = lis2

| append(h :: t, lis2) = h :: append(t, lis2);

The val statement binds a name to a value (similar to DEFINE in Scheme)

val distance = time * speed;

As is the case with DEFINE, val is nothing like an assignment statement in an imperative language

3.7 Exception Handling in ML

- The following code prompts the user to enter a numeric literal, and stores the corresponding real number in num:

```
while True:
```

```
try:
```

```
    response = raw_input("Enter a numeric literal: ")
```

```
    num = float(response)
```



```

break
except ValueError:
    print "Your response was ill-formed."

```

This while-command keeps prompting until the user enters a well-formed numeric literal. The library procedure `raw_input(...)` displays the given prompt and returns the user's response as a string. The type conversion `"float(response)"` attempts to convert the response to a real number. If this type conversion is possible, the following break sequencer terminates the loop. If not, the type conversion throws a `ValueError` exception, control is transferred to the `ValueError` exception handler, which displays a warning message, and finally the loop is iterated again.

3.8 Haskell-F#

- Similar to ML (syntax, static scoped, strongly typed, type inferencing, pattern matching)
- Different from ML (and most other functional languages) in that it is *purely* functional (e.g., no variables, no assignment statements, and no side effects of any kind)

Syntax differences from ML

```

fact 0 = 1
fact n = n * fact (n - 1)
fib 0 = 1
fib 1 = 1
fib (n + 2) = fib (n + 1) + fib n

```

3.9 Functional Programming with Lists

- List notation: Put elements in brackets
e.g., `directions = ["north", "south", "east", "west"]`
- Length: `#`
e.g., `#directions` is 4
- Arithmetic series with the `..` operator
e.g., `[2, 4..10]` is `[2, 4, 6, 8, 10]`
- Catenation is with `++`
e.g., `[1, 3] ++ [5, 7]` results in `[1, 3, 5, 7]`
- `CONS`, `CAR`, `CDR` via the colon operator (as in Prolog)
e.g., `1:[3, 5, 7]` results in `[1, 3, 5, 7]`

Factorial Revisited

```

product [] = 1
product (a:x) = a * product x
fact n = product [1..n]

```

List Comprehension

- Set notation
- List of the squares of the first 20 positive integers: `[n * n | n <- [1..20]]`
 - All of the factors of its given parameter:
 - `factors n = [i | i <- [1..n div 2], n mod i == 0]`

Lazy Evaluation

- A language is *strict* if it requires all actual parameters to be fully evaluated
- A language is *nonstrict* if it does not have the strict requirement
- Nonstrict languages are more efficient and allow some interesting capabilities
 - *infinite lists*
- Lazy evaluation - Only compute those values that are necessary
- Positive numbers
`positives = [0..]`
- Determining if 16 is a square number


```

member [] b = False
member(a:x) b=(a == b)||member x b
squares = [n * n | n ← [0..]]
member squares 16

```

Member Revisited

- The member function could be written as:

```

member [] b = False
member(a:x) b=(a == b)||member x b

```
- However, this would only work if the parameter to squares was a perfect square; if not, it will keep generating them forever. The following version will always work:

```

member2 (m:x) n
| m < n = member2 x n
| m == n = True
| otherwise = False

```

3.10 List Structures

- Other basic data structure (besides atomic propositions we have already seen):
list
- List is a sequence of any number of elements
- Elements can be atoms, atomic propositions, or other terms (including other lists)

```

[apple, prune, grape, kumquat]
[] (empty list)
[X | Y] (head X and tail Y)

```

Append Example

```

append([], List, List).
append([Head | List_1], List_2, [Head | List_3]) :-
  append(List_1, List_2, List_3).

```

Reverse Example

```

reverse([], []).
reverse([Head | Tail], List) :-
  reverse(Tail, Result),
  append(Result, [Head], List).

```

3.11 List Manipulation

A software system often consists of a number of subsystems controlled or connected by a script. **Scripting** is a paradigm characterized by:

- Use of scripts to glue subsystems together.
- Rapid development and evolution of scripts.
- Modest efficiency requirements.
- Very high-level functionality in application-specific areas.

Key Concepts

The following concepts are characteristic of scripting languages:

- Very high-level string processing.
- Very high-level graphical user interface support.
- Dynamic typing.

3.11 Simplification of Expressions

PYTHON was designed in the early 1990s by Guido van Rossum. It has been used to help implement the successful Web search engine GOOGLE, and in a variety of other application areas ranging from science fiction (visual effects for the *Star Wars* series) to real science (computer-aided design in NASA).

VALUES AND TYPES

- PYTHON has a limited repertoire of primitive types: integer, real, and complex numbers.
- It has no specific character type; single-character strings are used instead. Its boolean values (named False and True) are just small integers.
- PYTHON has a rich repertoire of composite types: tuples, strings, lists, dictionaries and objects. A PYTHON list is a heterogeneous sequence of values.
- A *dictionary* (sometimes called an associative array) is a heterogeneous mapping from keys to values, where the keys are distinct immutable values.
- The following code illustrates tuple construction:

```
date = 1998, "Nov", 19
```

Now `date[0]` yields 1998, `date[1]` yields "Nov", and `date[2]` yields 19.

- The following code illustrates two list constructions, which construct a homogeneous list and a heterogeneous list, respectively:

```
primes = [2, 3, 5, 7, 11]
```

```
years = ["unknown", 1314, 1707, date[0]]
```

Now `primes[0]` yields 2, `years[1]` yields 1314, `years[3]` yields 1998, "`years[0] = 843`" updates the first component of `years`, and so on. Also, "`years.append(1999)`" adds 1999 at the end of `years`.

3.12 Storage allocation for Lists

- PYTHON supports global and local variables.
- Variables are not explicitly declared, simply initialized by assignment. After initialization, a variable may later be assigned any value of any type.
- PYTHON's repertoire of commands include assignments, procedure calls, conditional (if- but *not* case-) commands, iterative (while- and for-) commands and exception-handling commands.
- However, PYTHON differs from C in not allowing an assignment to be used as an expression.
- PYTHON additionally supports simultaneous assignment.
- For example:

```
y, m, d = date
```

assigns the three components of the tuple `date` to three separate variables.

Also:

```
m, n = n, m
```

concisely swaps the values of two variables `m` and `n`. (Actually, it first constructs a pair, then assigns the two components of the pair to the two left-side variables)

- PYTHON if- and while-commands are conventional.
- PYTHON for-commands support definite iteration.
- We can easily achieve the conventional iteration over a sequence of numbers by using the library procedure `range(m,n)`, which returns a list of integers from `m` through `n-1`.
- PYTHON supports break, continue, and return sequencers. It also supports exceptions, which are objects of a subclass of `Exception`, and which can carry values.



UNIT-4

LOGIC PROGRAMMING

4.1 Logic Programming Introduction

BASICS

- Logic programming languages, sometimes called *declarative* programming languages
- Express programs in a form of symbolic logic
- Use a logical inferencing process to produce results
- *Declarative* rather than *procedural*:
 - Only specification of *results* are stated (not detailed *procedures* for producing them)

FACTS

Proposition

- A logical statement that may or may not be true
 - Consists of objects and relationships of objects to each other

Symbolic Logic

- Logic which can be used for the basic needs of formal logic:
 - Express propositions
 - Express relationships between propositions
 - Describe how new propositions can be inferred from other propositions
- Particular form of symbolic logic used for logic programming called *predicate calculus*

SYNTAX

Object Representation

- Objects in propositions are represented by simple terms: either constants or variables
- *Constant*: a symbol that represents an object
- *Variable*: a symbol that can represent different objects at different times
 - Different from variables in imperative languages

Compound Terms

- *Atomic propositions* consist of compound terms
- *Compound term*: one element of a mathematical relation, written like a mathematical function
 - Mathematical function is a mapping
 - Can be written as a table

Parts of a Compound Term

- Compound term composed of two parts
 - *Functor*: function symbol that names the relationship
 - *Ordered list of parameters (tuple)*

Examples:

student(jon)
like(seth, OSX)
like(nick, windows)
like(jim, linux)

Forms of a Proposition

- Propositions can be stated in two forms:
 - *Fact*: proposition is assumed to be true

- Query: truth of proposition is to be determined
- Compound proposition:
 - Have two or more atomic propositions
 - Propositions are connected by operators

RULES

Logical Operators

Name	Symbol	Example	Meaning
Negation	\neg	$\neg a$	a not b
Conjunction	\cap	$a \cap b$	a and b
Disjunction	\cup	$a \cup b$	a or b
Equivalence	\equiv	$a \equiv b$	a is equivalent to b
Implication	\supset	$a \supset b$	a implies b
	\subset	$a \subset b$	b implies a

Quantifiers

Name	Example	Meaning
universal	$\forall X.P$	For all X, P is true
existential	$\exists X.P$	There exists a value of X such that P is true

Clausal Form

- Too many ways to state the same thing
- Use a standard form for propositions
- *Clausal form*:
 - $B_1 \cup B_2 \cup \dots \cup B_n \subset A_1 \cap A_2 \cap \dots \cap A_m$
 - means if all the A s are true, then at least one B is true
- *Antecedent*: right side
- *Consequent*: left side

Predicate Calculus and Proving Theorems

- A use of propositions is to discover new theorems that can be inferred from known axioms and theorems
- *Resolution*: an inference principle that allows inferred propositions to be computed from given propositions

Resolution

- *Unification*: finding values for variables in propositions that allows matching process to succeed
- *Instantiation*: assigning temporary values to variables to allow unification to succeed
- After instantiating a variable with a value, if matching fails, may need to *backtrack* and instantiate with a different value

OPERATIONAL SEMANTICS

- Basis for logic programming
- When propositions used for resolution, only restricted form can be used
- *Horn clause* - can have only two forms
 - *Headed*: single atomic proposition on left side
 - *Headless*: empty left side (used to state facts)
- Most propositions can be stated as Horn clauses



4.2 An Overview of Logic Programming

- Declarative semantics
 - There is a simple way to determine the meaning of each statement
 - Simpler than the semantics of imperative languages
- Programming is nonprocedural
 - Programs do not state how a result is to be computed, but rather the **form of the result**

4.3 The Origins of Prolog

- University of Aix-Marseille
 - Natural language processing
- University of Edinburgh
 - Automated theorem proving

4.5 The Basic Elements of ProLog CO4

- Edinburgh Syntax
- *Term*: a constant, variable, or structure
- *Constant*: an atom or an integer
- *Atom*: symbolic value of Prolog
- Atom consists of either:
 - a string of letters, digits, and underscores beginning with a lowercase letter
 - a string of printable ASCII characters delimited by apostrophes

Terms: Variables and Structures

- *Variable*: any string of letters, digits, and underscores beginning with an uppercase letter
- *Instantiation*: binding of a variable to a value
 - Lasts only as long as it takes to satisfy one complete goal
- *Structure*: represents atomic proposition
`functor(parameter list)`

Fact Statements

- Used for the hypotheses
- Headless Horn clauses
`female(shelley).`
`male(bill).`
`father(bill, jake).`

Rule Statements

- Used for the hypotheses
- Headed Horn clause
- Right side: *antecedent* (**if** part)
 - May be single term or conjunction
- Left side: *consequent* (**then** part)
 - Must be single term
- *Conjunction*: multiple terms separated by logical AND operations (implied)

Example Rules

`ancestor(mary,shelley):- mother(mary,shelley).`

• Can use variables (*universal objects*) to generalize meaning:

`parent(X,Y):- mother(X,Y).`

`parent(X,Y):- father(X,Y).`

`grandparent(X,Z):- parent(X,Y), parent(Y,Z).`

`sibling(X,Y):- mother(M,X), mother(M,Y),`

`father(F,X), father(F,Y).`

Goal Statements

- For theorem proving, theorem is in form of proposition that we want system to

prove or disprove – *goal statement*

- Same format as headless Horn
man(fred)
- Conjunctive propositions and propositions with variables also legal goals
father(X,mike)

Inferencing Process of Prolog

- Queries are called goals
- If a goal is a compound proposition, each of the facts is a subgoal
- To prove a goal is true, must find a chain of inference rules and/or facts.

For goal Q:

B :- A

C :- B

...

Q :- P

- Process of proving a subgoal called matching, satisfying, or resolution

Approaches

- *Bottom-up resolution, forward chaining*
 - Begin with facts and rules of database and attempt to find sequence that leads to goal
 - Works well with a large set of possibly correct answers
- *Top-down resolution, backward chaining*
 - Begin with goal and attempt to find sequence that leads to set of facts in database
 - Works well with a small set of possibly correct answers
- Prolog implementations use backward chaining

Subgoal Strategies

- When goal has more than one subgoal, can use either
 - Depth-first search: find a complete proof for the first subgoal before working on others
 - Breadth-first search: work on all subgoals in parallel
- Prolog uses depth-first search
 - Can be done with fewer computer resources

Backtracking

- With a goal with multiple subgoals, if fail to show truth of one of subgoals, reconsider previous subgoal to find an alternative solution: *backtracking*
- Begin search where previous search left off
- Can take lots of time and space because may find all possible proofs to every subgoal

Simple Arithmetic

- Prolog supports integer variables and integer arithmetic
- *is* operator: takes an arithmetic expression as right operand and variable as left operand
A is B / 17 + C
- Not the same as an assignment statement!

Example

speed(ford,100).
speed(chevy,105).
speed(dodge,95).
speed(volvo,80).
time(ford,20).
time(chevy,21).
time(dodge,24).
time(volvo,24).
distance(X,Y) :-

speed(X,Speed),



$time(X, Time),$
 $Y \text{ is } Speed * Time.$

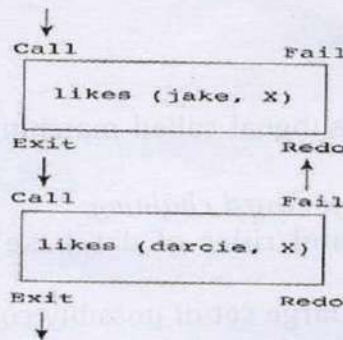
Trace

- Built-in structure that displays instantiations at each step
- *Tracing model* of execution - four events:
 - *Call* (beginning of attempt to satisfy goal)
 - *Exit* (when a goal has been satisfied)
 - *Redo* (when backtrack occurs)
 - *Fail* (when goal fails)

Example

likes(jake, chocolate).
likes(jake, apricots).
likes(darcie, licorice).
likes(darcie, apricots).

trace.
likes(jake, X),
likes(darcie, X).



4.6 Deficiencies of Prolog

- Resolution order control
- The closed-world assumption
- The negation problem
- Intrinsic limitations

4.7 Applications of Logic Programming - CO4

- Relational database management systems
- Expert systems
- Natural language processing



UNIT V

CONCURRENT PROGRAMMING

5.1 Parallelism in Hardware-Streams

Implicit Synchronization

- Implicit synchronization is done at the start and may be done at the end of a test case by the MTC. The corresponding **CREATE** constructs and **DONE** events can be generated automatically by a tool.
- Further synchronization is needed if it has to be guaranteed that the first send event happens after the creation of all PTCs or if the PTCs should indicate their termination to the MTC. For these cases, one of the explicit synchronization mechanisms has to be used.

5.2 Concurrency as interleaving

- Concurrency can occur at four levels:
 - Machine instruction level
 - High-level language statement level
 - Unit level
 - Program level
- Because there are no language issues in instruction- and program-level concurrency, they are not addressed here

Multiprocessor Architectures

- Late 1950s - one general-purpose processor and one or more special purpose processors for input and output operations
- Early 1960s - multiple complete processors, used for program-level concurrency
- Mid-1960s - multiple partial processors, used for instruction-level concurrency
- Single-Instruction Multiple-Data (SIMD) machines
- Multiple-Instruction Multiple-Data (MIMD) machines
 - Independent processors that can be synchronized (unit-level concurrency)

Categories of Concurrency

- A *thread of control* in a program is the sequence of program points reached as control flows through the program
- Categories of Concurrency:
 - *Physical concurrency* - Multiple independent processors (multiple threads of control)
 - *Logical concurrency* - The appearance of physical concurrency is presented by time-sharing one processor (software can be designed as if there were multiple threads of control)
- Coroutines (*quasi-concurrency*) have a single thread of control

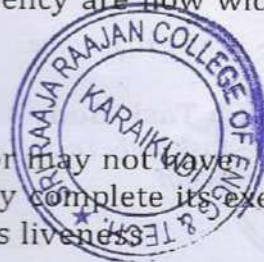
Motivations for Studying Concurrency

- Involves a different way of designing software that can be very useful— many real-world situations involve concurrency
- Multiprocessor computers capable of physical concurrency are now widely used

5.3 Liveness Properties

Liveness and Deadlock

- *Liveness* is a characteristic that a program unit may or may not have
 - In sequential code, it means the unit will eventually complete its execution
- In a concurrent environment, a task can easily lose its liveness



- If all tasks in a concurrent environment lose their liveness, it is called *deadlock*

Design Issues for Concurrency

- Competition and cooperation synchronization
- Controlling task scheduling
- How and when tasks start and end execution
- How and when are tasks created

Methods of Providing Synchronization

- Semaphores
- Monitors
- Message Passing

5.4 Safe Access to Shared Data

1) Immutable objects are by default thread-safe because their state can not be modified once created.

Since String is immutable in Java, it's inherently thread-safe.

2) Read-only or **final variables in Java** are also thread-safe in Java.

3) Locking is one way of achieving thread-safety in Java.

4) **Static variables** if not synchronized properly become a major cause of thread-safety issues.

5) Example of thread-safe class in Java: Vector, Hashtable, ConcurrentHashMap, String, etc.

6) Atomic operations in Java are thread-safe like reading a 32-bit int from memory because it's an atomic operation it can't interleave with other threads.

7) local variables are also thread-safe because each thread has there own copy and using local variables is a good way to write thread-safe code in Java.

8) In order to avoid thread-safety issues minimize the sharing of objects between multiple threads.

9) **Volatile keyword in Java** can also be used to instruct thread not to cache variables and read from main memory and can also instruct JVM not to reorder or optimize code from threading perspective.

5.5 Concurrency in Ada

- The Ada 83 Message-Passing Model
 - Ada tasks have specification and body parts, like packages; the spec has the interface, which is the collection of entry points:

```
task Task_Example is
  entry ENTRY_1 (Item : in Integer);
end Task_Example;
```

Task Body

- The body task describes the action that takes place when a rendezvous occurs
- A task that sends a message is suspended while waiting for the message to be accepted and during the rendezvous
- Entry points in the spec are described with accept clauses in the body accept

```
entry_name (formal parameters) do
```

```
...
end entry_name
```

Example of a Task Body

```
task body Task_Example is
begin
  loop
    accept Entry_1 (Item: in Float) do
```



```

...
end Entry_1;
end loop;
end Task_Example;

```

5.6 Synchronized Access to Shared Variables

- A *task* or *process* is a program unit that can be in concurrent execution with other program units
- Tasks differ from ordinary subprograms in that:
 - A task may be implicitly started
 - When a program unit starts the execution of a task, it is not necessarily suspended
 - When a task's execution is completed, control may not return to the caller
- Tasks usually work together

Two General Categories of Tasks

- *Heavyweight tasks* execute in their own address space
- *Lightweight tasks* all run in the same address space
- A task is *disjoint* if it does not communicate with or affect the execution of any other task in the program in any way

Task Synchronization

- A mechanism that controls the order in which tasks execute
- Two kinds of synchronization
 - *Cooperation* synchronization
 - *Competition* synchronization
- Task communication is necessary for synchronization, provided by:
 - Shared nonlocal variables
 - Parameters
 - Message passing

Kinds of synchronization

- *Cooperation*: Task A must wait for task B to complete some specific activity before task A can continue its execution, **e.g.**, the producer-consumer problem
- *Competition*: Two or more tasks must use some resource that cannot be simultaneously used, **e.g.**, a shared counter
 - Competition is usually provided by mutually exclusive access (approaches are discussed later)

Scheduler

- Providing synchronization requires a mechanism for delaying task execution
- Task execution control is maintained by a program called the *scheduler*, which maps task execution onto available processors

Task Execution States

- *New* - created but not yet started
- *Ready* - ready to run but not currently running (no available processor)
- *Running*
- *Blocked* - has been running, but cannot now continue (usually waiting for some event to occur)
- *Dead* - no longer active in any sense



5.7 Synthesized Attributes, Attribute Grammar

Attribute Grammar

Attribute grammar is a special form of context-free grammar where some additional information (attributes) are appended to one or more of its non-terminals in order to provide context-sensitive information. Each attribute has well-defined domain of values, such as integer, float, character, string, and expressions.

Attribute grammar is a medium to provide semantics to the context-free grammar and it can help specify the syntax and semantics of a programming language. Attribute grammar (when viewed as a parse-tree) can pass values or information among the nodes of a tree.

Example:

```
E → E + T
{
  E.value = E.value + T.value
}
```

The right part of the CFG contains the semantic rules that specify how the grammar should be interpreted. Here, the values of non-terminals E and T are added together and the result is copied to the non-terminal E.

Semantic attributes may be assigned to their values from their domain at the time of parsing and evaluated at the time of assignment or conditions. Based on the way the attributes get their values, they can be broadly divided into two categories : synthesized attributes and inherited attributes.

Synthesized attributes

These attributes get values from the attribute values of their child nodes. To illustrate, assume the following production:

$$S \rightarrow ABC$$

If S is taking values from its child nodes (A,B,C), then it is said to be a synthesized attribute, as the values of ABC are synthesized to S.

As in our previous example ($E \rightarrow E + T$), the parent node E gets its value from its child node. Synthesized attributes never take values from their parent nodes or any sibling nodes.

5.8 Natural Semantics- Denotational Semantics

Semantics of state

- State (such as a heap) and simple imperative features can be straightforwardly modeled in the denotational semantics described above
- The key idea is to consider a command as a partial function on some domain of states. The meaning of " $x:=3$ " is then the function that takes a state to the state with 3 assigned to x. The sequencing operator ";" is denoted by composition of functions. Fixed-point constructions are then used to give a semantics to looping constructs, such as "while".

Denotations of data types

- Many programming languages allow users to define recursive data types. For example, the type of lists of numbers can be specified by

datatype list = Cons of nat * list | Empty



Denotational semantics of sequentiality

- The problem of full abstraction for the sequential programming language PCF was, for a long time, a big open question in denotational semantics. The difficulty with PCF is that it is a very sequential language. For example, there is no way to define the parallel-or function in PCF. It is for this reason that the approach using domains, as introduced above, yields a denotational semantics that is not fully abstract.
- This open question was mostly resolved in the 1990s with the development of game semantics and also with techniques involving logical relations.^[16] For more details, see the page on PCF.

Denotational semantics as source-to-source translation

- It is often useful to translate one programming language into another. For example, a concurrent programming language might be translated into a process calculus; a high-level programming language might be translated into byte-code.

5.9 Lexically Scoped Lambda Expressions

- This section deals only with functional data structures that cannot change. Conventional imperative programming languages would typically allow the elements of such a recursive list to be changed.

- For another example: the type of denotations of the untyped lambda calculus is

datatype D = D of (D \rightarrow D)

- The problem of *solving domain equations* is concerned with finding domains that model these kinds of datatypes. One approach, roughly speaking, is to consider the collection of all domains as a domain itself, and then solve the recursive definition there. The textbooks below give more details.
- Polymorphic data types are data types that are defined with a parameter. For example, the type of α lists is defined by

datatype α list = Cons of $\alpha * \alpha$ list | Empty

- Lists of natural numbers, then, are of type `nat list`, while lists of strings are of `string list`.
- Some researchers have developed domain theoretic models of polymorphism. Other researchers have also modeled parametric polymorphism within constructive set theories. Details are found in the textbooks listed below.
- A recent research area has involved denotational semantics for object and class based programming

semantics for programs of restricted complexity

- Following the development of programming languages based on linear logic, denotational semantics have been given to languages for linear usage (see e.g. proof nets, coherence spaces) and also polynomial time complexity.

5.10 A Calculator in Scheme

- **Syntax.** Legal Calculator expressions are either numbers or well-formed Scheme lists that have an operator symbol as their first element. The latter are interpreted as call expressions. Legal operator symbols include `+`, `-`, `*`, and `/`.
- **Evaluation.** The evaluation procedure for each operator is described in the lecture notes section about the Scheme-Syntax Calculator.



- **Read-Eval-Print.** When run interactively, the interpreter reads Scheme expressions (without quotation or dotted lists), evaluates them, and prints the results.

> 2

2

> (+ 1 2 (* 3 4))

15

5.11 An Interpreter-Recursive Functions

- An interpreter is the simplest way to have actions expressed by a source program to be performed. This is done by processing the AST.
- An interpreter considers nodes of the AST in the correct order and performs the prescribed actions for those nodes by the language semantics. Unlike compiling, this requires inputs to be present.
- Ideally interpreters work just like CPUs, the difference is that while CPUs work on instruction sets, interpreters work on ASTs.
- Interpreters can either be:
 - *recursive* - working directly on the AST thus less preprocessing
 - *iterative* - working on a linearized AST.

Recursive Interpreters.

- A recursive interpreter will have a routine for each node type in the AST. These routines call other similar routines.
- This is possible since the meaning of language constructs is defined as a function of the meaning of its components. i.e The meaning of an *if-statement* is defined by the meaning of the condition, the *then-part* and the *else-part* including a short paragraph in the manual tying them together.





SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
AMARAVATHIPUTHUR POST, KARAUKUDI - 630301



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INTERNAL MARKS STATEMENT

Degree/Branch : M.E/CSE Year/Semester : I/ II
Academic Year : 2020-21
Sub Code & Name : CP5001/Principles of Programming Language

S.NO.	REG.NO.	NAME	IT1	IT2	MODEL
		DATE	17.3.21	21.4.21	10.5.21
1	912520405001	DEVI ABARNA SRI A	88	83	80
2	912520405002	GOWTHAMI R	72	67	71
3	912520405003	PANDI MUTHU C	55	48	63



Faculty in Charge

HOD Signature



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
AMARAVATHIPUTHUR POST, KARAIKUDI - 630301



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INTERNAL TEST I - EXAM RESULT ANALYSIS

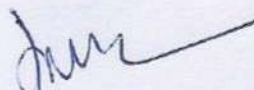
Department : CSE Date of Exam : 17.03.21
Year / Semester : I/ II Date of Evaluation : 19.03.21
Subject Code/Title : CP5001/Principles of Programming Language
Faculty Name : MR.Ponvasan.P Designation : AP/CSE
Total Strength : 3 Present : 3 Absent : NIL
Passed : 3 Failed : NIL % of Pass : 100%

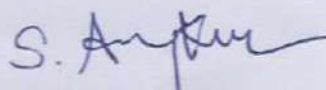
Description	Below 50	50 to 59	60 to 74	75 to 89	90 and Above
No. of Students		1	1	1	

Register Number and Name of the Failed Students:

S. No	Reg. No.	Name of Student
NIL		




Faculty in-Charge


HOD Signature



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
AMARAVATHIPUTHUR POST, KARAUKUDI - 630301



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

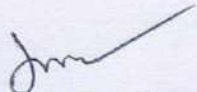
INTERNAL TEST II - EXAM RESULT ANALYSIS

Department	: CSE	Date of Exam	: 21.04.21
Year / Semester	: I / II	Date of Evaluation	: 23.04.21
Subject Title	: Internet Programming	Subject Code	: CP5001
Faculty Name	: MR.Ponvasan.P	Designation	: AP/CSE
Total Strength	: 3	Present	: 3
		Absent	: NIL
Passed	: 2	Failed	: NIL
		% of Pass	: 100%

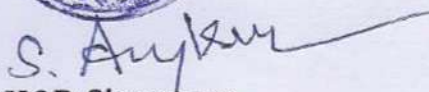
Description	Below 50	50 to 59	60 to 74	75 to 89	90 and Above
No. of Students	1		1	1	

Register Number and Name of the Failed Students:

S. No	Reg. No.	Name of Student
1.	-	NIL


Faculty in-Charge




HOD Signature



SRI RAAJA RAAJAN COLLEGE OF ENGINEERING & TECHNOLOGY
AMARAVATHIPUTHUR POST, KARAUKUDI - 630301



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

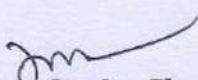
MODEL- EXAM RESULT ANALYSIS

Department	: CSE	Date of Exam	: 10.05.21
Year / Semester	: I/ II	Date of Evaluation	: 13.05.21
Subject Title	: Internet Programming	Subject Code	: CP5001
Faculty Name	: MR.Ponvasan.P	Designation	: AP/CSE
Total Strength	: 3	Present	: 3
		Absent	: NIL
Passed	: 3	Failed	: NIL
		% of Pass	: 100%

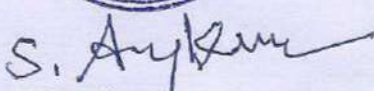
Description	Below 50	50 to 59	60 to 74	75 to 89	90 and Above
No. of Students			2	1	

Register Number and Name of the Failed Students:

S. No	Reg. No.	Name of Student
NIL		


Faculty in-Charge




HOD Signature

Case Study: Design Engineering for Ear Biometric Model

Angayarkanni N¹, Minarva Devi K², Ashaboshini R³, Devi Abarna Sri A⁴, Aanjana Devi S⁵,
Aanjankumar S⁶, Anitha Thangasamy⁷

^{1,2,3,4,5,6}Sri Raaja Raajan College of Engineering and Technology, Karaikudi, Tamilnadu, India

⁵Alagappa University, Karaikudi, Tamilnadu, India

⁷Anna university-CEG Campus, Tamilnadu, India

Angayarkanni n13@gmail.com¹, minarvadevi88@gmail.com², ashaboshini@gmail.com³, abarnasri2404@gmail.com⁴, devisuresh94@gmail.com⁵, itsec1990@gmail.com⁶, anithathangasamy@gmail.com⁷

Abstract:

Biometric based authentication is the wide spread concept of identifying the personal characteristics while offering certain services. There are huge varieties of biometric authentication approaches are proposed by various scientific researches and developers like fingerprints, face biometric, iris, DNA, palm and Ear. While comparing with other biometric technique Ear biometric gets more attention than others. Because the ear-based authentication has various advantages and offers security and reliability than the other biometrics. This paper investigates the various existing models in ear based biometric and tabulate its performance analysis for easier implementations.

Keywords: Ear, Biometrics, Ear Biometric, Review, Performance Analysis, Evaluation of Ear Biometrics, PCA.

I. INTRODUCTION

In this modern era, biometric concepts are introduced in wide variety of applications and it is the best replacement models for our traditional user authentication called Login. The usual login models included the user name and passwords. These models are replaced by variety of security features because of the security weakness. The Biometric technology is the modern approach which can be utilized in user authentication to offer high security. The biometrics can test the users and authenticate the users by their physical characteristics with the help of computers, sensors and machine learning devices. The Biometric identification is separated into various models like Iris, Palm, Fingerprint, Face, DNA and Ear. Ear based authentication is the newest trend in human authentication and it gets more attention because of its unique features and security strength [1].



[2850]

PRINCIPAL

Sri Raaja Raajan College of Engg. & Tech
Amaravathipudugai - 630 301
Sivagangai Dist. Tamil Nadu

IJARSE

INTERNATIONAL JOURNAL OF
ADVANCE RESEARCH IN SCIENCE
AND ENGINEERING

ISSN(O) : 2319-8354, ISSN(P) : 2319-8346

Certificate

This is to certify that
N.Rajeswari

has published a paper title

**Maximum Power Point Tracking Control Solar Power
Generation Using Fuzzy Network**

in

International Journal of Advance Research in Science and Engineering

Volume No.06, Special Issue No.(01), Dec 2017, ICASES-17

This paper can be downloaded from the following link: www.ijarse.com

Longa

Editor in Chief

International Journal of Advance Research in Science and Engineering

website: www.ijarse.com

E-mail: submission@ijarse.com

IJARSE Team wishes all the best for your bright future

Principal

PRINCIPAL

**Sri Raaja Raajan College of Engg. & Tech.
Amaravathipudur, Karaikudi - 630 001
Sivagangai Dist. Tamil Nadu**



An Efficient Honey Badger Optimization Based Solar MPPT Under Partial Shading Conditions

N. Rajeswari^{1,*} and S. Venkatanarayanan²

¹Department of Electrical and Electronics Engineering, Sri Rāja Raajan College of Engineering and Technology, Sivaganga, Tamilnadu, 630301, India

²Department of Electrical and Electronics Engineering, K. L. N. College of Engineering, Sivagangai, Tamilnadu, 630612, India

*Corresponding Author: N. Rajeswari. Email: rajisugumar22@gmail.com

Received: 12 February 2022; Accepted: 24 March 2022

Abstract: Due to the enormous utilization of solar energy, the photovoltaic (PV) system is used. The PV system is functioned based on a maximum power point (MPP). Due to the climatic change, the Partial shading conditions have occurred under non-uniform irradiance conditions. In the PV system, the global maximum power point (GMPP) is complex to track in the P - V curve due to the Partial shading. Therefore, several tracking processes are performed using various methods like perturb and observe (P & O), hill climbing (HC), incremental conductance (INC), Fuzzy Logic, Whale Optimization Algorithm (WOA), Grey Wolf Optimization (GWO) and Flying Squirrel Search Optimization (FSSO) etc. Though, the MPPT is not so efficient when the partial shading is increased. To increase the efficiency and convergences in MMPT, the Honey Badger optimization (HBO) algorithm is presented. This HBO model is motivated by the excellent foraging behaviour of honey badgers. This HBO model is used to achieve the best solution in GMPP tracking and speed convergence. The HBO methodology is also compared with prior P&O, WOA and FSSO methods using MATLAB. Therefore, the experiment shows that the HBO method is performed a higher tracking than all prior methods.

Keywords: PV system; gmpp tracking; convergence; honey badger optimization; digging and honey phase

1 Introduction

Solar energy is the most essential source to provide a clean environment and a better gain in economic. Nowadays photovoltaic (PV) systems are acted as a main solar source for electricity generation. But in the PV system, the conversion of insolation into electricity is more difficult and has minimum efficiency [1]. Solar radiation and atmospheric temperature are the environmental factors that are used for power generation in a PV system. The characteristics of power-voltage (P - V) and current-voltage (I - V) are affected by these environmental factors.



This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Sri Rāja Raajan College of Engineering and Technology
Amaravathi Road, Sivagangai - 630 301
Sivagangai Dist. Tamil Nadu

CONTENTS

Year: 2018, Volume: 4, Issue: 10

Research Articles

Power Quality Analysis in a Bulk High-tension Industry at Theni, Tamil Nadu	1
<i>Sahayaraj Arokyasamy, S. Venkatanarayanan</i>	
Design and Implementation of Maximum Point Power Tracking controller for Grid-connected Photovoltaic System and Analyzing its Performances	6
<i>C. Karuppasamy, V. M. Ravi Kumar, S. Venkatanarayanan</i>	
Optimum Location of Distributed System Using Shuffled Frog Leaping Algorithm	10
<i>N. Rajeswari, S. Venkatanarayanan, R. Karpaga Priya, S. Kannan</i>	
Internet of Things Based Monitoring of Fertilizer and Moisture in Agriculture.....	15
<i>M. Seema, S. Venkatanarayanan, S. M. Kannan</i>	
A Novel Method of Detection and Classification of Diabetic Retinopathy in Fundus Imagery	18
<i>P. Bhuvaneshwari, R. Banumathi, S. Venkatanarayanan, S. M. Kannan</i>	
Analysis of Various Types of Balanced Voltage Sags.....	24
<i>K. R. Ramela, S. Parameswari, S. Venkatanarayanan, S. M. Kannan</i>	
Industrial Energy Audit and Improve Power Quality in Solar Energy System	29
<i>S. Sivakumar, S. Venkatanarayanan, S. M. Kannan</i>	
Performance Analysis of Fuzzy Logic Controller-based dc-Link Shunt Compensator in Single-Phase Grid-Connected Mode	33
<i>M. Nagajothi, S. Venkatanarayanan, N. Rajeswari, S. M. Kannan</i>	
Transform-Based Texture Feature Extraction Techniques	38
<i>C. Udhaya, R. Banumathi, S. Venkatanarayanan, S. M. Kannan</i>	
Internet of Things-Based Health Monitoring System Using Microelectromechanical Systems' Sensor	44
<i>C. Gayathri, S. Venkatanarayanan, R. Karpaga Priya, M. Kannan</i>	
Sequential Loading of Backup Generator with Load Sharing	56
<i>P. K. Arun Kumar, S. Venkatanarayanan, R. Pragadeesh, S. Nikhil, R. Prasanna, A. Prasanna Venkatesh, S. M. Kannan</i>	
Gender Classification Using Histogram of Oriented Gradients-Support Vector Machine Classifier.....	61
<i>S. Gomathi Meena, K. G. Srinivasagan</i>	
Analysis and Development of Music Rhythm for LED Flashlight Using Microcontrollers	69
<i>M. Muthumeenakshi, B. Ishuvarya, N. Andar, A. M. Gayathri, S. Venkatanarayanan, S. M. Kannan</i>	
Multi-constraint Fuzzy-based Wireless Sensor Node Communication for Smart Grid.....	72
<i>R. Karpaga Priya, S. Venkatanarayanan, N. Rajeswari, S. M. Kannan</i>	
Real-Time Detection of Retinal Detachment and Exudates Using Digital Fundus Image with Superpixel Multi-Feature Classification.....	76
<i>R. Priyanka, J. Kanimozhi, Vengat Narayanan, P. Vasuki</i>	
Investigations on Severity Level for Diabetic Maculopathy Based on the Location of Lesions	82
<i>A. Priyanga, D. Rajesh Kumar, S. Venkatanarayanan, S. M. Kannan</i>	



Research Article

Optimum Location of Distributed System Using Shuffled Frog Leaping Algorithm

N. Rajeswari¹, S. Venkatanarayanan², R. Karpaga Priya³, S. Kannan²

¹Department of Electrical and Electronics Engineering, Sri Raaja Raajan College of Engineering and Technology, Karaikudi, Tamil Nadu, India, ²Department of Electrical and Electronics Engineering, K.L.N. College of Engineering, Sivagangai, Tamil Nadu, India, ³Department of Electronics and Communication Engineering, Vaigai College of Engineering, Madurai, Tamil Nadu, India



CrossMark

ABSTRACT

This paper presents a novel optimization algorithm for optimizing the distributed generation (DG) parameters in deregulated power system which improves the stability, reduces the losses, and also expands the cost of age. The shuffled frog leaping algorithm used to optimize the various DG parameters simultaneously. The various parameters taken into consideration are their type, location, and size of the DG devices. The simulation was performed on a distribution system and modeled for steady-state studies.

Address for correspondence:

N. Rajeswari, Sri Raaja Raajan College of Engineering and Technology, Amaravathipuram, Karaikudi, Sivaganga, Tamil Nadu, India.

Keywords:

Optimisation Technique, Active Power, Statcom, Distributed Generation

Received: 05th February 2018
Accepted: 20th September 2018
Published: 13th October 2018

INTRODUCTION

Nowadays, the power electricity demand is growing fast, and one of the main tasks for power engineers is to generate electricity from renewable energy sources to overcome this increase in the energy consumption and at the same time reduces environmental impact of power generation. The utilization of renewable sources of vitality has achieved more noteworthy significance as it advances practical living and with a few exemptions (biomass burning) does not contaminant. Sustainable sources can be utilized as a part of either little scale applications far from the substantial estimated age plants or in expansive scale applications in areas where the asset is plentiful and extensive change frameworks are utilized.

In any case, issues emerge when the new age is coordinated with the power dispersion arrange, as the customary dissemination frameworks have been intended to work radially, without considering the mix of this new age later on. In outspread frameworks, the power streams from upper terminal voltage levels down to clients arranged along the spiral feeders. In this manner, over current insurance in spiral frameworks is very clear as the blame current can just stream one way. With the expansion of infiltration

of distributed generation (DG), circulation systems are getting to be plainly like transmission systems where age and load hubs are blended ("work" framework) and more intricate assurance configuration is required. In this new configuration, design considerations regarding the number, size location, and technology of the DG connected must be taken into account as the short-circuit levels are affected and miss coordination problems with protection devices may arise.

The term DG is frequently used to portray a small-scale electricity generation. However, what exactly is small-scale electricity generation? Currently, there is no consensus on how the DG should be exactly defined. As shown by the survey conducted by CIRED, there is no consensus on the definition of this term. Some countries define DG on the basis of the voltage level, whereas others start from the principle that DG is connected to circuits from which consumer loads are supplied directly. Other countries define DG as having some basic characteristic (for example, using renewables, cogeneration, being non-dispatch able, etc.).

This paper presents survey works and a methodology for optimizing a utility-owned DG size and location on the basis of economic considerations under existing loading

Copyright ©2017. The Author(s). Published by Arunai publications private Ltd.



This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>)