St. Joseph's College of Engineering & Technology Palai

Department of Computer Science & Engineering

S4 CS

CS010 406 Theory of Computation

Module 1

Theory of Computation - Module 1

Brought to you by
http://nutlearners.blogspot.com

Syllabus

Proving techniques- Mathematical induction - Diagonalization principle - Pigeonhole principle - Functions - Primitive recursive and partial recursive functions - Computable and non computable functions - Formal representation of languages - Chomsky classification

Contents

Part I. Introduction

The subject Theory of Computation is a core part of Computer Science. The aim of this subject is

1. to familiarize students with the foundation and principles of computer science,
2. to teach material that is useful in subsequent courses (compiler design, algorithm analysis),
3. to strengthen students' ability to carry out formal and rigorous mathematical arguments.

This subject includes topics such as

automata theory,
languages,
grammars,
computability and
complexity.

These constitute the theoretical foundation of Computer Science.

The field of Computer Science includes a wide range of topics from machine design to programming. But all these have some common underlying principles. To study these principles, we construct abstract models of computers and computation.

The ideas we discuss in this subject have some immediate and important applications. For example, the fields of digital design, programming languages and compilers are the important examples. Also it has applications in operating systems to pattern recognition.

This subject provides many challenging, puzzle like problems that can lead to many sleepless nights.

1 Origin of Theory of Computation

Always man was interested in creating machines to ease human labour. In 16th century, Pascal invented a computing device. In 19th century, Charles Babbage built an analytical engine that performed computations without human intervention. In 1930s, mathematicians and philosophers thought of an ideal model which has features of computation as an intelligent activity.

A. M. Turing brought the idea of a machine called Turing Machine. This turing machine is a model of automatic computation. This led turing to propose a thesis called Turing thesis.

Turing thesis states that any computable function can be solved by a Turing machine. Then only in 1950, it was able to create a computer based on this model.

In 1943, Prof. McCullough brought the model of finite automata. [this will be learned in detail in Module 2]. finite automata has a large number of applications including lexical analysis in compilers, text editors (example: notepad, kwrite), special purpose hardware design, control unit design etc..

Prof. Noam Chomsky proposed a mathematical model to specify the syntax of natural languages. This resulted in formal language theory. A grammar called context free grammar was proposed for programming languages.

Languages, Grammars and automata form three fundamental ideas of theory of Computation.

Part II. Proving Techniques

[Pandey2006]

An important requirement for learning this subject is the ability to follow proofs. There are three fundamental techniques to proof. They are,

1. Principle of mathematical induction,
2. Pigeonhole principle,
3. Diagonalization principle.

2 Principle of Mathematical Induction

Let we want to show that property P holds for all natural numbers. To prove this property, P using mathematical induction, following are the steps:

Basic Step:

First show that property P is true for 0 or 1.

Induction Hypotheis:

Assume that property P holds for n.

Induction step:

Using induction hypothesis, show that P is true for n+1.

Then by the principle of mathematical induction, P is true for all natural numbers.

Examples

Example 1:

Prove using mathematical induction, $n^4 - 4n^2$ is divisible by 3 for n>=0.

Basic step:

For n=0,

$n^4 - 4n^2$ =0, which is divisible by 3.

Induction hypothesis:

Let $n^4 - 4n^2$ is divisible by 3.

Induction step:

$(n+1)^4 - 4(n+1)^2$

$= [(n+1)^2]^2 - 4(n+1)^2$

$= (n^2 + 2n + 1)^2 - (2n+2)^2$

$= (n^2 + 2n + 1 + 2n + 2)(n^2 + 2n + 1 - 2n - 2)$

$= (n^2 + 4n + 3)(n^2 - 1)$

$= n^4 + 4n^3 + 3n^2 - 3 - 4n - n^2$

$= n^4 + 4n^3 + 2n^2 - 4n - 3$

$= n^4 + 4n^3 - 4n^2 + 6n^2 - 4n - 3$

$= n - 4n^2 + 6n^2 - 3 + 4n^3 - 4n$

$= (n^2 - 4n^2) + (6n^2) - (3) + 4(n^3 - n)$

$(n^2 - 4n^2)$ is divisible by 3 from our hypothesis.

$6n^2, 3$ are divisible by 3.

We need to prove that $4(n^3 - n)$ is divisible by 3.

Again use mathematical induction.

Basic step:

For n = 0,

4(0-0) = 0 is divisible by 3.

Induction hypothesis:

Let $4(n^3 - n)$ is divisible by 3.

Induction step:

$4[(n+1)^3 - (n+1)]$

$= 4[(n^3 + 3n^2 + 3n + 1) - (n+1)]$

$= 4[n^3 + 3n^2 + 3n + 1 - n - 1]$

$= 4[n^3 + 3n^2 + 2n]$

$= 4[n^3 - n + 3n^2 + 3n]$

$= 4(n^3 - n) + 4.3n^2 + 4.3n$

$4(n^3 - n)$ is divisible by 3 from our hypothesis.

$4.3n^2$ is divisible by 3.

$4.3n$ is divisible by 3.

Thus we can say that

$= (n^2 - 4n^2) + (6n^2) - (3) + 4(n^3 - n)$ is divisible by 3.

That is,

$n^4 - 4n^2$ is divisible by 3.

Example 2:

Prove using mathematical induction:

$1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$

Let $P(n) = 1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$

Basic Step:

For n=1,

LHS = 1

RHS = 1(1+1)/2=1

Induction hypothesis;

Assume that P(n) is true for n=k,

Then,

$1 + 2 + 3 + ... + k = \frac{k(k+1)}{2}$

Induction step:

Show that P(n) is true for n=k+1.

$1 + 2 + 3 + ... + k + (k + 1)$

$= \frac{k(k+1)}{2} + k + 1$

$= (k + 1)[\frac{k}{2} + 1]$

$= (k + 1)(\frac{k+2}{2})$

$= \frac{(k+1)(k+2)}{2}$

That is, P(n) is true for n=k+1.

Thus by using the principle of mathematical induction, we proved

$1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$

Example 3:

Prove using the principle of mathematical induction,

$\sum_{i=0}^{n} n^2 = \frac{n(n+1)(2n+1)}{6}$

Let $P(n) = \sum_{i=0}^{n} n^2 = \frac{n(n+1)(2n+1)}{6}$

Basic step:

For n=1,

LHS = $1^2 = 1$

RHS = $\frac{n(n+1)(2n+1)}{6} = \frac{1.2.3}{6} = 1$

P(n) is true for n=1.

Induction hypothesis:

Assume that result is true for n=k.

That is,

$1^2 + 2^2 + ... + k^2 = \frac{k(k+1)(2k+1)}{6}$

Induction step:

Prove that result is true for n=k+1.

$1^2 + 2^2 + ... + k^2 + (k + 1)^2 = \frac{k(k+1)(2k+1)}{6} + (k + 1)^2$

$= (k + 1)[\frac{k(2k+1)}{6} + (k + 1)]$

$= (k + 1)(\frac{2k^2+k+6k+6}{6})$

$= (k + 1)(\frac{2k^2+7k+6}{6})$

$= (k + 1)(\frac{2k^2+4k+3k+6}{6})$

$= (k + 1)(\frac{2k(k+2)+3(k+2)}{6})$

$= \frac{(k+1)(k+2)(2k+3)}{6}$

Thus it is proved.

Exercise:

1. Prove the following by principle of induction:

a. $\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$

b. $\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$

c. 2i>i for all i>1.

d. $1 + 4 + 7 + ... + (3n - 2) = \frac{n(3n-1)}{2}$

e. $2^x \geq x^2$,if $x \geq 4$

f. $10^{2n} - 1$ is divisible by 11 for all n>1

g. $2^n > n$, for all n>1.

3 Diagonalization Principle

Let s be a non empty set and R any relation on S.

Let

$D = \{a \epsilon A | (a, a) \notin r\}$

For each $a \notin A$, let $R_a = \{b | (a, b) \in R\}$

Then diagonalization principle states that D is different from each R_a.

OR

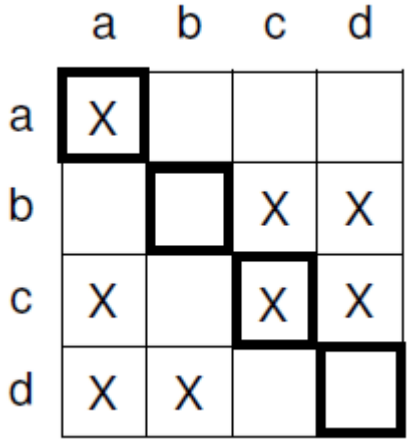
Diagonalization principle states that the complement of the diagonal is different from each row.

For example,

Let S = {a, b, c, d}

R = { (a,a), (b,c), (b,d), (c,a), (c,c), (c,d), (d,a), (d,b) }

The above relation R is shown in matrix from as follows:



Diagonal elements are marked.

From the figure, $R_a = \{a\}$

$R_b = \{c, d\}$

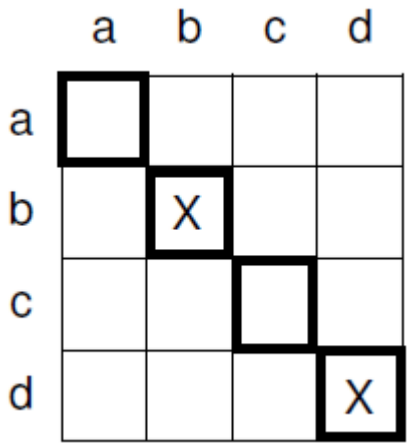
$R_c = \{a, c, d\}$

$R_d = \{a, b\}$

Complement of the diagonal is,

D = {b, d}

That is,



If we compare each of the above R_a, R_b, R_c, R_d with D, we can see that D is different from each R_a. Thus complement of the diagonal is distinct from each row.

Note:

Following information is needed to solve the example given below:

9's complement of a number

9's complement of 276 is 723.

9's complement of 425 is 574.

9's complement of 793 is 206.

Example 1:

Prove that the set of real numbers between 0 and 1 is uncountable.

An example for a real number between 0 and 1 is .34276

Let we represent a real number between 0 and 1 as

$x = .x_0x_1x_2x_3.....$

where each x_i is a decimal digit.

Let f(k) be an arbitrary function from natural numbers to the set [0,1].

We can arrange the elements in a 2d array as,

f(0): . x_{00} x_{01} x_{02} x_{03} _ _ _

f(1): . x_{10} x_{11} x_{12} x_{13} _ _ _

f(2): . x_{20} x_{21} x_{22} x_{23} _ _ _

_

_

f(n): . x_{n0} x_{n1} x_{n2} x_{n3} _ _ _

where x_{ni} is the ith digit in the decimal expansion of f(n).

Next we find the complement of the diagonal (9's complement) as follows:

$Y = .y_0y_1y_2.....$

where y_i =9's complement of x_{ii}

[Find out the 9's complement of $x_{00}, x_{11}, x_{22}, x_{33}.....x_{nn}$]

From the diagonalisation principle, it is clear that the complement of the diagonal is different from each row.

Here it is clear that Y is different from each f(i) in at least one digit. $Y \neq f(i)$. Hence Y cannot be present in the above array.

This means that the set of real numbers between 0 and 1 is countably infinite or not countable.

For instance suppose we arrange the real numbers as,

f(0):	.	9	4	2	4	_	_	_
f(1):	.	6	3	6	2	_	_	_
f(2):	.	2	8	6	4	_	_	_
f(3)	.	6	5	3	2	_	_	_
_	.	_	_	_	_	_	_	_
f(n):	.	4	1	5	7	_	_	_

Here the diagonal is 9 3 6 2.....

The complement (9's complement) of the diagonal is 0 6 3 7.....

The real number .0637... is not in the above table.

Thus It is clear that this value is distinct from each row, f(0), f(1), f(2)...f(n).

4 Pigeonhole Principle

A pigeonhole is a hole for pigeons to nest.

From the above diagram, it is clear that if 10 pigeons are put into 9 pigeonholes, then one pigeonhole must contain

more than one item.

The pigeonhole principle states that if n pigeons are put into m pigeonholes with n>m, then at least one pigeonhole must contain more than one pigeon.

Thus if S1 and S2 are two non empty finite sets and $|S1| > |S2|$, then there is no one-to-one function from S1 to S2.

Pigeonhole principle can be used to show that certain languages are not regular. We will use pigeonhole principle in the topic pumping lemma later.

Part III. Recursive Function Theory

Functions

The concept of a function is a fundamantal topic in mathematics.

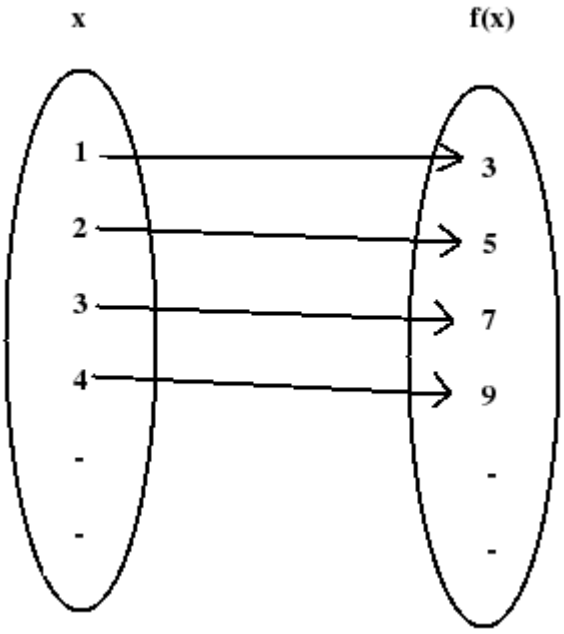
A function or a total function $f : X \longrightarrow Y$ is a rule that assigns to all the elements of one set, X, a unique element of another set, Y.

The first set, X is called the domain of the function, and the second set, Y is called its range.

For example,

$$f(x) = 2x + 1$$

is a function defined on one variable, x. Imagine we say that f is over all natural numbers only. Then the following diagram shows the function:



Here the domain of f is {1, 2, 3, 4, ...}

Range of f is {3, 5, 7, 9, ...}.

$$f(x, y) = x^2 + 2y$$

is a function on 2 variables, x and y.

$$f(x, y, z) = x + 2y^3 + 7z$$

is a function on 3 variables, x, y and z.

Total Function

The above three examples were total functions. A total function from X to Y is a rule that assigns to every element of X, a unique element of Y.

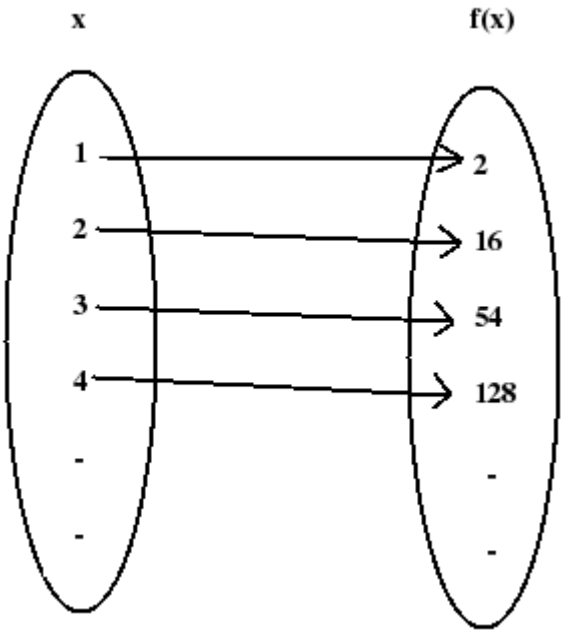
$$f : X \longrightarrow Y$$

For example,

$$f(x) = 2x^3$$

In all our discussions, we assume that f is over all natural numbers,

then



Here the domain of f is {1, 2, 3, 4 .....}.

That is, the domain contains all the elements of x. So f is a total function.

Partial Function

Here, to extend the class of computable functions, we use partial functions.

A partial function,

$$f : X \longrightarrow Y$$

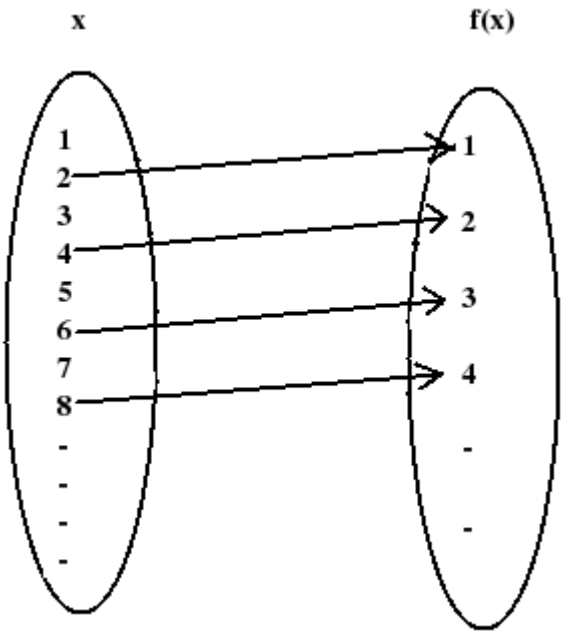
is a rule that assigns to some elements of X, a unique element of Y. For other elements, there may not be any corresponding element in Y.

For example,

Example 1:

$$f(x) = \frac{x}{2}$$

Suppose f is defined over natural numbers. Then f is a partial function. this can be seen from the following diagram.



In the above diagram, domain of f is {2, 4, 6, 8, ...}

Here the domain is not equal to the set X. So f is a partial function.

Example 2:

$$f(x, y) = x - y$$

f is defined over N.

f is a partial function.

This is because when x=6, y=8, f(x,y) = -2

-2 is not a natural number. So f is not defined for x=6, y=8. f is a partial function.

Recursive Functions

Consider the example,

$$f(n) = n^2 + 1$$

Here, given any value for n, multiply that value by itself and then add one. Here function is defined in a recursive way. Value of f can be computed in a mechanical fashion.

Example 2:

Factorial of a number is defined as,

n! = n (n-1) (n-2) . . . . . 1.

This is an explicit definition.

Another way to define factorial is,

0! = 1

n! = n (n-1)!

for $n \epsilon N$

This definition is recursive because to find the factorial at an argument n, we need to find the factorial at some simpler argument (n-1).

Example 3:

Exponentiation can be defined as,

$x^n = x.x.x.x.....x$ (n times)

This exponentiation function can be recursively defined as,

$x^0 = 1$

$x^n = x.x^{n-1}$

for $n \epsilon N$.

This is a recursive definition for exponentiation.

In all our discussions, a function, f is defined over natural numbers (N) only.

5 Primitive Recursive Functions

A function, f is called a primitive recursive function,

i) If it is one of the three basic functions, or,

ii) If it can be obtained by applying operations such as composition and recursion to the set of basic functions.

The basic functions and operations are explained below;

5.1 Basic Functions

We define three basic functions. They are,

Zero function,

Successor function, and

Projector function.

Zero Function

Z(x) = 0

is called Zero function.

Example:

We may write

Z(8) = 0.

Successor Function

The successor function is S(x), defined as

S(x) = x+1

Thus the value of S(x) is the integer next in sequence to x.

For example,

S(4) = 5

S(29) = 30

Projector Function

Projector function, P_k is defined as,

$$P_k(x_1, x_2, ...x_k, ...x_n) = x_k$$

For example,

$$P_3(8, 5, 6, 4) = 6$$

$$P_6(6, 34, 7, 2, 45, 23, 22) = 23$$

5.2 Operations

We can build complicated functions from the above basic functions by performing operations such as,

composition, and

recursion.

Composition

If functions, f_1, f_2 and g are given, then the composition of g with f_1, f_2 is given by,

$$h = g(f_1, f_2).$$

In general,

If functions $f_1, f_2.....f_k$ and g are given, then the composition of g with $f_1, f_2.....f_k$ is

$$h = g(f_1, f_2, f_3.....f_k)$$

Example 1:

Let

$$g(n) = n^2$$

$$h(n) = n + 3$$

Find the composition of h with g.

The composition of h with g is,

$$\begin{aligned} f(n) &= h[g(n)] \\ &= h[n^2] \\ &= n^2 + 3 \end{aligned}$$

Example 2:

Let

$g(n) = n^2$

$h(n) = n + 3$

Find the composition of g with h.

$$\begin{aligned} f(n) &= g[h(n)] \\ &= g[n+3] \\ &= (n+3)^2 \end{aligned}$$

Example 3:

Let

$f_1(x, y) = x + y$

$f_2(x, y) = 2x$

$f_3(x, y) = xy$, and

$g(x, y, z) = x + y + z$

be functions over N.

Find the composition of g with f_1, f_2, f_3.

$$\begin{aligned} h(x, y) &= g(f_1, f_2, f_3) \\ &= g\,(x + y,\ 2x,\ xy) \\ &= x + y + 2x + xy \\ &= 3x + y + xy \end{aligned}$$

Recursion

A function f can be constructed using recursion from the functions g and h as,

$f(x,\ 0) = g(x)$

$f(x,\ y + 1) = h\,[x,\ y,\ f(x, y)]$

In the above, f is of two variables.

g is of one variable and h is of 3 variables.

In general,

a function of n+1 variables is defined by recursion if there exists a function g of n variables and a function h of n+2 variables.

f is defined as,

$f(x_1, x_2, ...x_n, 0) = g(x_1, x_2, ...x_n)$

$f(x_1, x_2, ...x_n,\ y + 1) = h\,[x_1, x_2, ...x_n,\ y,\ f(x_1, x_2, ...x_n, y)\,]$

Example 1:

Addition of integers can be defined using recursion as,

$add(x, 0) = x$

$add(x, y + 1) = add(x, y) + 1$

Example 2:

The function multiplication can be defined using recursion operation as,

$prod(x, 0) = 0$

$prod(x, y + 1) = add\,[x,\ prod(x, y)\,]$

Above examples show how recursion operation is applied to a set of functions.

Primitive Recursive functions

Now we learned basic functions such as zero function, successor function and projector function, and operations such as composition and recursion.

Again, a function, f is a primitive recursive function if either,

i. it is one of the basic functions, or

ii. it is produced by performing operations such as composition and recursion to the basic functions.

Example 1:

The addition function, add(x,y) is a primitive recursive function because,

$$\begin{aligned} add(x, 0) &= x \\ &= P_1(x) \\ add(x, y + 1) &= add(x, y) + 1 \\ &= S[add(x, y)] \\ &= S[P_3(x, y, add(x, y))] \end{aligned}$$

Thus add(x, y) is a function produced by applying composition and recursion to basic functions, P_k and S.

Example 2:

The multiplication function mult(x, y) is a primitive recursive function because,

$$\begin{aligned} mult(x, 0) &= 0 \\ &= Z(x) \\ mult(x, y + 1) &= add\,[x,\ mult(x, y)\,] \\ &= add[P_1(x, y, mult(x, y)), P_3(x, y, mult(x, y))] \end{aligned}$$

From the previous example, add() is a primitve recursive function.

From the above, mult(x,y) is produced by performing operations on basic functions and add() function.

So mult(x,y) is a primitive recursive function.

Example 3:

The factorial function,

f(n) = n!

is a primitive recursive function. This can be proved using mathematical induction.

Basic Step:

For n = 0,

f(0) = 0! = 1

= $P_1(1)$

Induction Hypothesis:

Assume that f(p) is a primitive recursive function.

Induction Step:

f(p+1) = (p+1)!

= p! (p+1)

= f(p) (p+1)

= mult [f(p), p+1]

= mult [f(p), S(p)]

In the above, f(p) is primitive recursive from induction hypothesis.

S(p) is primitive recursive, since it is the successor function.

mult() is primitve recursive from the previous example.

So we can conclude that

f(n) = n!

is primitive recursive.

Example 4:

Show that function $f_1(x, y) = x + y$ is primitive recursive.

$$f_1(x, 0) = x + 0$$
$$= x$$
$$= P_1(x)$$
$$f_1(x, y + 1) = x + (y + 1)$$
$$= (x + y) + 1$$
$$= f_1(x, y) + 1$$
$$= S[f_1(x, y)]$$
$$= S[P_3(x, y, f_1(x, y))]$$

Since $f_1(x, 0)$ and $f_1(x, y + 1)$ are primitive recursive, $f_1(x, y) = x + y$ is primitive recursive.

Example 5:

Show that the function, $f_1(x, y) = x * y$ is primitive recursive.

$$f_2(x, 0) = x * 0$$
$$= 0$$
$$= Z(x)$$
$$f_1(x, y + 1) = x * (y + 1)$$
$$= (x * y) + x$$
$$= f_2(x, y) + x$$

$= f_1[f_2(x,y), x]$ from Example 4.

$= f_1[P_3(x,y,f_2(x,y)),\ P_1(x,y,f_2(x,y))]$

Since $f_2(x,0)$ and $f_2(x,y+1)$ are primitive recursive, $f_2(x,y) = x * y$ is primitive recursive.

Example 6:

Show that the function, $f(x,y) = x^y$ is a primitive recursive function.

$f(x,0) = x^0$

$= 1$

$= P_1(1)$

$f(x,y+1) = x^{y+1}$

$= x^y.x^1$

$= x.x^y$

$= x.f(x,y)$

$= P_1(x,y,f(x,y)) * P_3(x,y,f(x,y))$

$= mult\ [P_1(x,y,f(x,y)),\ P_3(x,y,f(x,y))\ ]$

mult() is a primitive recursive function as we found earlier.

Since $f(x,0)$ and $f(x,y+1)$ are primitive recursive, $f(x,y) = x^y$ is primitive recursive.

6 Partial Recursive Functions

A function, f is a partial recursive function if either,

i. it is one of the basic functions, or

ii. it is produced by performing operations such as composition, recursion and minimization to the basic functions.

Basic Functions

We learned three basic functions such as,

Zero function, Z(x),

Successor function, S(x), and

Projector function, P_k.

Operations

We learned the operations composition and recursion earllier.

A new operation now comes, minimization.

Minimization

μ is the minimization operator.

Let a function g(x,y) is defined over two variables, x and y.

Then minimization operator, μ is defined over g (x, y) is as follows:

$\mu_y[g(x,y)]$ =smallest y such that g (x, y) = 0.

Using the minimization operation, a function f(x) can be defined for some y;

we can write, f(x) = $\mu_y[g(x,y)]$

Example 1:

Let g(x,y) is given in the following table:

g(0,0)=5	g(1,0)=5	g(2,0)=8	g(3,0)=1
g(0,1)=4	g(1,1)=6	g(2,1)=5	g(3,1)=2
g(0,2)=6	g(1,2)=0	g(2,2)=undefined	g(3,2)=0
g(0,3)=0	g(1,3)=3	g(2,3)=0	g(3,3)=4
g(0,4)=1	g(1,4)=0	g(2,4)=7	g(3,4)=undefined

Then,

f(0) = 3,

[because g(0,3) = 0]

Minimization operation is denoted by $\mu[g(0,3)=0]=3$

f(1) = 2

[2 is the minimum y that g(1,2)=0; Though g(1,4)=0 also;]

Minimization operation is denoted by $\mu[g(1,2)=0]=2$

f(2) = Not defined

[because g(2,3)=0, but g(2,2) is undefined, where y=2 < y=3]

f(3) = 2

[because g(3,2)=0, though g(3,4) is undefined for y=4, but 4<2]

Minimization operation is denoted by $\mu[g(3,2)=0]=2$

For some values g(x,y) is undefined, but we can define minimization operation.

Partial Recursive Functions

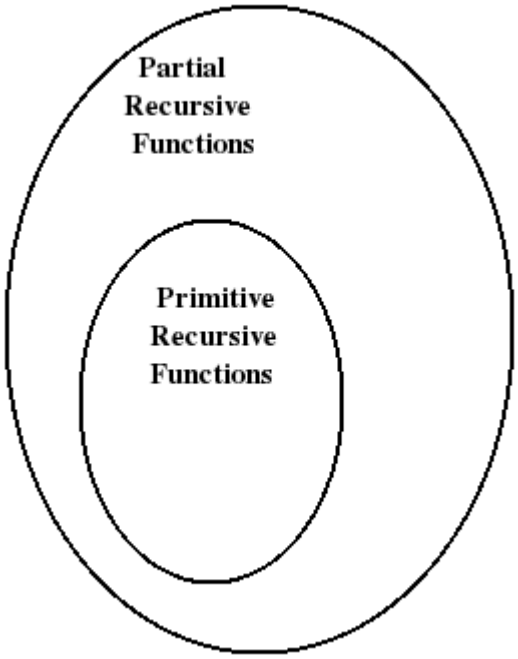
A function, f is a partial recursive function if either,

i. it is one of the basic functions, or

ii. it is produced by performing operations such as composition, recursion and minimization to the basic functions.

From the definition, we can say that,

primitive recursive functions are a subset of partial recursive functions. This is shown below:



Thus all primitive recursive functions are partial recursive functions.

Note that all partial recursive functions are computable functions.

Example 1:

Show that $f(x) = \frac{x}{2}$ is a partial recursive function over N.

$f(x) = \frac{x}{2}$

We may write

$y = \frac{x}{2}$

Then,

$x = 2y$

$2y - x = 0$

Let

$g(x, y) = |2y - x|$

Let

$g(x, y) = 0$, for some x and y.

Let

$f_1(x) = \mu[|2y - x| = 0]$

Only for even values of x , $f_1(x)$ is defined. When x is odd, $f_1(x)$ is not defined. So f_1 is partial recursive.

Since, $f(x) = \frac{x}{2} = f_1(x)$, f is a partial recursive function.

Part IV. Computable and Non-Computable Functions

7 Computable Functions

Computable functions are the basic objects of study in Theory of Computation. Computable functions are the functions that can be calculated using a mechanical calculation device given unlimited time and memory space. Also any function that has an algorithm is computable.

Each computable function f takes a fixed finite number of natural numbers as arguments. The computable function f returns an output for some inputs.

For some inputs, it may not return an output. Due to this, a computable function is a paertial recursive function.

If the computable function is defined for all possible arguments, it is called a total computable function or total recursive function.

The notation $f(x_1, x_2, ...x_k)$ indicates that the partial function f is defined on arguments $x_1, x_2, ...x_k$.

We say that a function f on a certain domain is said to be computable, if there exists a Turing machine that computes the value of f for all arguments in its domain. A function is uncomputable if no such Turing machine exists.

The basic characteristic of a computable function is that there is a finite procedure or algorithm telling how to compute the function.

The characterisitics of a computable function are as follows:

1. There must be exact instructions (ie, a program, finite in length).

2. If the procedure is given a k tuple x in the domain of f, then after a finite number of steps, the procedure must terminate and produce f(x).

3. If the procedure is given a k-tuple x which is not in the domain of f, then the procedure must never halt, or it may get stuck at some point.

Following are some examples for computable functions:

1. Each function with a finite domain.

eg. Any finite sequence of natural numbers.

2. Each constant function $f : N^k \longrightarrow N$, $f(n_1, n_{2,}.....n_k) = n$.

3. Addition $f : N^2 \longrightarrow N$, $f(n_1, n_2) = n_1 + n_2$.

4. The function which gives the list of prime factors of a number.

5. The gcd of two numbers is a computable function.

8 Non-Computable Functions

The rela numbers are uncountable so most real numbers are not computable. Most subsets of natural numbers are not countable.

A non-computable function corresponds to an undecidable problem and it consists of a family of instances for which there is no computer program that given any problem instance as input terminates and outputs the required answer after a finite number of steps.

For an undecidable problem, it is impossible to construct a single algorithm that always leads to a correct yes/ no answer.

We say that a function f on a certain domain is said to be non-computable, if no Turing machine exists that computes the value of f for all arguments in its domain.

Examples for undecidable problems (non-computabel functions) are halting problem, post correspondence problem.

Part V. Formal Representation of Languages

[Kavitha2004]

Here we introduce the concept of formal languages and grammars.

We know that a grammar is specified for a natural language such as English. To define valid sentences and to give structural descriptions of sentences a grammar is used. Linguistics is a branch to study the theory of languages. Scientists in this branch of study have defined a formal grammar to describe English that would make language translation using computers very easy.

English Grammar

First we will consider how grammars can be specified for English language. Then we will extend this model to our programming language grammar specification.

An example for grammar is,

ARTICLE $\longrightarrow$ a | an | the

NOUN_PHRASE $\longrightarrow$ ARTICLE NOUN

NOUN $\longrightarrow$ boy | apple | ε

Productions

The above are called productions or rules of the grammar. Two productions are given above.

First production is,

ARTICLE $\longrightarrow$ a | an | the

It can also be written as,

ARTICLE $\longrightarrow$ a

ARTICLE $\longrightarrow$ an

ARTICLE $\longrightarrow$ the

Second production is,

NOUN_PHRASE $\longrightarrow$ ARTICLE NOUN

Third production is,

NOUN $\longrightarrow$ boy | apple | ε

It can also be written as,

NOUN $\longrightarrow$ boy

NOUN $\longrightarrow$ apple

NOUN $\longrightarrow \varepsilon$

Non-terminals

In the above, ARTICLE, NOUN_PHRASE, NOUN are called non-terminal symbols. A non-terminal is specified using upper case letters.

Note that the left hand side of a production always contains a single non-terminal.

Terminals

In the above, 'a', 'an', 'the', 'boy', 'apple' are called terminal symbols. A terminal is written using lowercase letters.

ε means an empty string.

Start Symbol

In a grammar specification, one non-terminal symbol is called the start symbol. Here, NOUN_PHRASE is the start symbol.

Thus a grammar consists of a set of productions, terminals, non-terminals and start symbol.

If the matching left hand side of a certain rule can be found to occur in a string, it may be replaced by the corresponding right hand side.

8.1 Formal Definition of a Grammar

A grammar is $(V_N, \sum, P, S)$ where

V_N is a finite non empty set whose elements are called non-terminals.

$\sum$ is a finite non empty set whose elements are called terminals.

$V_N \cap \sum = \phi$

S is a special non terminal called start symbol.

P is a finite set whose elements are $\alpha \longrightarrow \beta$, whose α and β are terminals or non terminals. Elements of P are called productions or production rules.

For example,

Consider the productions,

$S \longrightarrow aB|bA$

$A \longrightarrow aS|bAA|a$

$B \longrightarrow bS|aBB|b$

where S is the start symbol.

For the above productions the grammar is defined as,

$V_N = \{S, A, B\}$

$\sum = \{a, b\}$

$S$is the start symbol S

$P = \{S \longrightarrow aB|bA, A \longrightarrow aS|bAA|a, B \longrightarrow bS|aBB|b\}$

Derivations

Productions are used to derive various sentences.

Example 1:

Consider the grammar given below:

SENTENCE ⟶ NOUN_PHRASE VERB_PHRASE

NOUN_PHRASE ⟶ ARTICLE NOUN

VERB_PHRASE ⟶ VERB NOUN_PHRASE

ARTICLE ⟶ a | an | the

NOUN ⟶ boy | apple

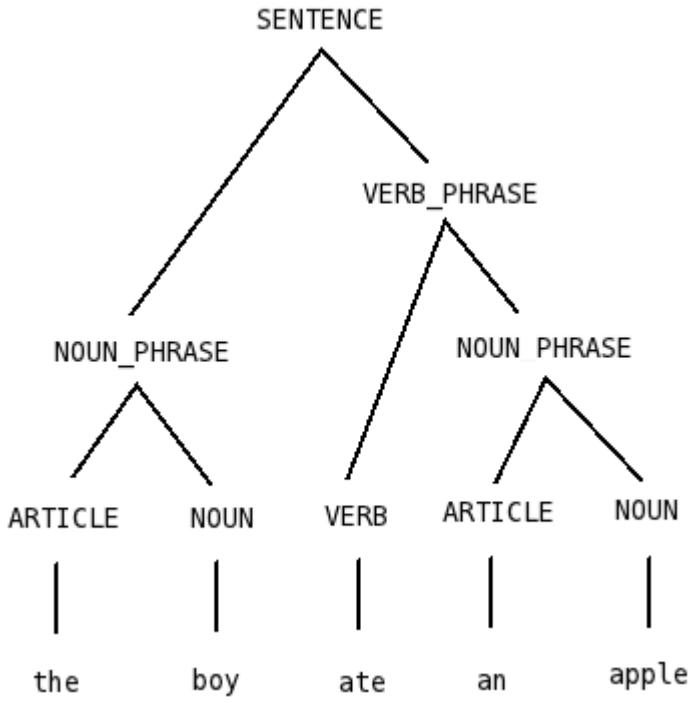
VERB ⟶ apple

where SENTENCE is the start symbol.

Using this grammar, the sentence

'the boy ate an apple'

is derived as shown below:



Example 2:

Consider the grammar given below:

$S \longrightarrow baaS \mid baa$

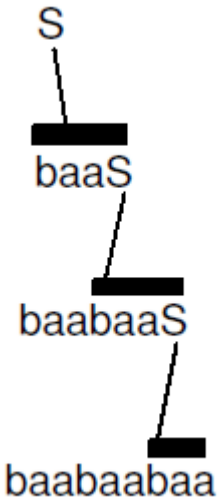
In the above, S is the start symbol, S is a non-terminal and a, b are terminals.

To derive a sentence, we begin with the start symbol, S.

From the above, we can rewrite, S as baa.

Thus the sentence, baa is a valid sentence according to the grammar.

From the above rule, rewrite S as baaS. Then rewrite S in baaS as baaS. Then we get baabaaS. Again rewrite S as baa. We get baabaabaa.



Thus baabaabaa is a valid sentence.

Example 3:

Following is a grammar for expressions:

Expr ⟶ Expr Op num | num

Op ⟶ + | - | * | /

In the above grammar, Expr, Op are non-terminals.

num, +, -, *, / are terminals.

Using the above grammar, a large set of expressions can be derived.

Example 4:

The following is a grammer for if..else construct in C.

Stmt ⟶ if (Expr) Stmt else Stmt | if (Expr) Stmt

Example 5:

The following is a grammar for a set of statements in C.

Stmt ⟶ id = Expr;

| if (Expr) Stmt

| if (Expr) Stmt else Stmt

| while (Expr) stmt

| do Stmt while (Expr);

| { Stmts }

Stmts ⟶ Stmts Stmt

| ε

8.2 Formal Definition of a Language

The language generated by a grammar G is L(G) is defined as

$L(G) = \{ w \epsilon \sum * \mid S* \Longrightarrow_G w \}$

The elements of L(G) are called sentences.

In a simple way, L(G) is the set of all sentences derived from the start symbol S.

For example, for the grammar

$S \longrightarrow baaS \mid baa$

L(G) = {baa, baabaa, baabaabaa, ... }

Example 1:

Consider the grammar given below:

ARTICLE ⟶ a | an | the

NOUN_PHRASE ⟶ ARTICLE NOUN

NOUN ⟶ boy | apple

where NOUN_PHRASE is the start symbol.

Find the language L(G) generated by the grammar.

i)

NOUN_PHRASE ⟶ ARTICLE NOUN

$\Longrightarrow$a boy

ii)

NOUN_PHRASE ⟶ ARTICLE NOUN

$\Longrightarrow$an apple

iii)

NOUN_PHRASE ⟶ ARTICLE NOUN

$\Longrightarrow$the boy

iv)

NOUN_PHRASE ⟶ ARTICLE NOUN

$\Longrightarrow$the apple

v)

NOUN_PHRASE ⟶ ARTICLE NOUN

$\Longrightarrow$a apple

NOUN_PHRASE $\longrightarrow$ ARTICLE NOUN

$\Longrightarrow$an boy

Hence L(G) = { a boy, an apple, the apple, the boy, a apple, an boy}

Note that 'a apple' and 'an boy' are in the set even if they are not valid English phrases.

Exercises:

1. Let grammar G = [{S,C}, {a,b}, P, S]

where P consists of

$S \longrightarrow aCa$

$C \longrightarrow aCa|b$. Find L(G).

2. Let $G = [\{S, A_1, A_2\}, \{a, b\}, P, S]$, where P consists of

$S \longrightarrow aA_1A_2a$

$A_1 \longrightarrow baA_1A_2b$

$A_2 \longrightarrow A_1ab$

$aA_1 \longrightarrow baa$

$bA_2b \longrightarrow abab$

Check whether w = baabbabaaabbaba is in L(G).

2. Let grammar G = [{S}, {0,1}, {S $\longrightarrow$0S1, S $\longrightarrow \epsilon$}, S]. Find L(G).

9 Chomsky Classification of Languages

[Kavitha2004]

A number of language families are there. Noam Chomsky, a founder of formal language theory, provided an initial classification into four language types, type 0 to type 3.

A grammar represents the structure of a language as we saw earlier.

Four types of languages and their associated grammars are defined in Chomsky Hierarchy (defined by Noam Chomsky).

The languages are

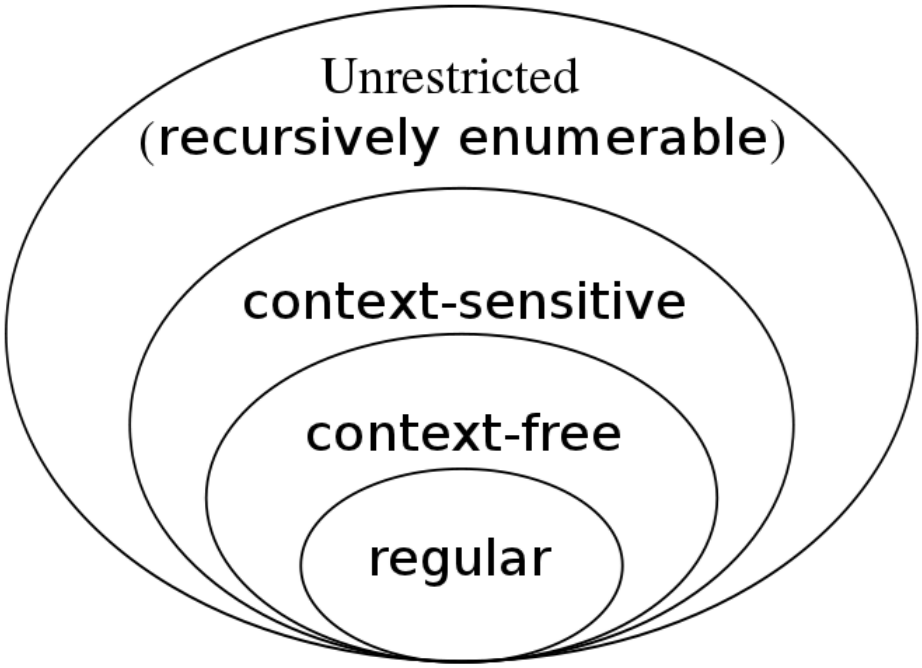
Type- 0 languages or Unrestricted languages,

Type- 1 languages or Context sensitive languages,

Type- 2 languages or Context free languages,

Type- 3 languages or regular languages.

Each language family is a proper subset of another one. The following figure shows the relationship.



Type-0 languages are generated by Type-0 grammars, Type-1 languages by Type-1 grammars, Type-2 languages by Type-2 grammars and Type-3 languages by Type-3 grammars.

Thus the corresponding grammars are

Type- 0 grammars or Unrestricted grammars,

Type- 1 grammars or Context sensitive grammars,

Type- 2 grammars or Context free grammars,

Type- 3 grammars or regular grammars.

Type- 0 or Unrestricted languages vs Type- 0 grammars

Type- 0 languages are defined by arbitrary grammars (Type-0).

Type- 0 grammars have productions of the form, $\alpha \longrightarrow \beta$, such that $\alpha \neq \varepsilon$.

Type- 0 languages are recognized using Turing Machines (TM).

The following is an example for an Unrestricted grammar:

S $\longrightarrow$ ACaB

Ca $\longrightarrow$ aaC

CB $\longrightarrow$ DB

aD $\longrightarrow$ Da

aE $\longrightarrow$ Ea

AE $\longrightarrow \varepsilon$

Type- 1 or Context- sensitive languages vs Type- 1 grammars

Type- 1 languages are described by Type- 1 grammars.

LHS of a Type- 1 grammar is not longer than the RHS.

Type- 1 grammars are represented by the productions $\alpha \longrightarrow \beta$, such that $|\alpha| \leq |\beta|$.

Type- 1 languages are recognized using Linear Bounded Automata (LBA).

The following is an example for a Context Sensitive grammar:

S $\longrightarrow$ SBC

S $\longrightarrow$ aC

B $\longrightarrow$ a

CB $\longrightarrow$ BC

Ba $\longrightarrow$ aa

C $\longrightarrow$ b

Type- 2 or Context- free languages vs Type- 2 grammars

Type- 2 languages are described by Type- 2 grammars or context free grammars.

LHS of a Context free grammar (Type-2) is a single non-terminal. The RHS consists of an arbitrary string of terminals and non-terminals.

Context free grammars are represented by the productions A $\longrightarrow \beta$.

Type 2 languages are recognized using Pushdown Automata (PDA).

The following is an example for a Context free grammar:

S $\longrightarrow$ aSb

S $\longrightarrow \varepsilon$

The English grammar given below is a context free grammar.

ARTICLE $\longrightarrow$ a | an | the

NOUN_PHRASE $\longrightarrow$ ARTICLE NOUN

NOUN $\longrightarrow$ boy | apple

Type- 3 or Regular languages vs Type- 3 grammars

Type- 3 languages are described by Type- 3 grammars.

The LHS of a Type- 3 grammar is a single non-terminal. The RHS is empty, or consists of a single terminal or a single terminal followed by a non- terminal.

Type- 3 grammars are represented by the productions A $\longrightarrow$ aB or A $\longrightarrow$ a.

Type- 3 languages are recognized using Finite State Automata (FA).

The following is an example for a regular grammar:

A $\longrightarrow$ aA

A $\longrightarrow \varepsilon$

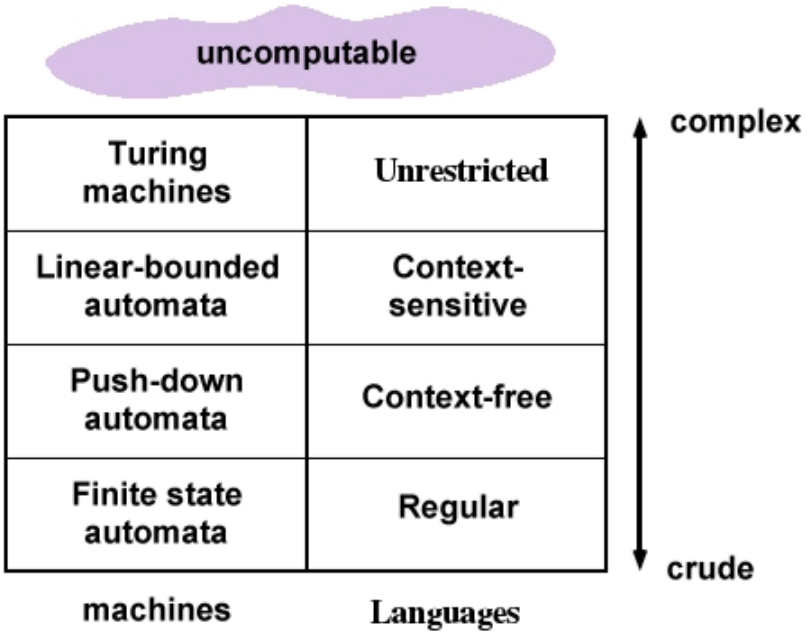
A $\longrightarrow$ bB

B $\longrightarrow$ bB

B $\longrightarrow \varepsilon$

A $\longrightarrow$ c

Following figure shows the relation between the four types of languages and machines.



Questions (from Old syllabus of S7CS TOC)

MGU/Nov2011

1. What is meant by a formal proof? (4marks).

2. a. Differentiate between primitive recursive and partial recursive functions with examples (12marks).

OR

b. Write notes on: ii) diagonalization principle. (12marks)

MGU/April2011

2. What is a primitive recursive function (4marks)?

3. a. Show that

ii) For n is an integer >=0, $n^3 + 2n$ is divisible by 3 (12marks).

OR

b. i) Show that a problem whose language is recursive is undecidable.

ii) If Z is the decimal expansion of pi, then show that the function f(x) = 1 for exactly x consecutive 3's in z and f(x) = 0, otherwise (12marks).

MGU/Nov2010

1. What is meant by computable and non-computable? (4marks)

2. Define a grammar and what are various types of grammars (4marks).

3 a. show that factorial of a number is a primitive recursive function.

OR

b. i) Show that multiplication is a computable function on the set of natural numbers.

ii). Discuss the computability of f(x) = 1 for xth digit of Z = 3 and f(x) = 0 otherwise Z is a decimal expansion of $\pi = 3.14285714.....$(12marks)

MGU/May2010

1. Define primitive recursive function (4marks).

2. Differentiate cmputable function from non-computable function (4marks).

3 a. Explain what is meant by the following statements:-

i) f: N1N is a total recursive (TR) function.

ii) The sequence $\{f_n : N1N*\}n2N$ of TR functions of a single variable is recursively enumerable.

iii) Show that no recursive enumeration can include the set of all TR functions of a single variable (12marks).

MGU/Nov2009

1. What does it mean for a subset "s" of the set "N" of natural numbers to be register machine decidable? (4marks).

2. What is diagonalisation principle? (4marks)

3a. Prove that the union and intersection of two recursive languages are also recursive.

OR

b. Discuss the proof that the closure properties of regular sets are closed (12mraks).

MGU/Nov2008

1. Give the formal definition of context sensitive grammar (4marks).

2. Show that $\Sigma_{k=0}^{n} k = \frac{n(n+1)}{2}$. (4marks)

3a. Prove that 2^n is uncountable using diagonalisation principle (12marks).

OR

b. Compare recursive function, primitive recursive function and partial recursive function (12marks).

4a

ii) Show that $f(r) = \frac{x}{2}$ is a partial recursive function (12marks).

MGU/May2008

1. What is primitive recursive function (4marks).

2. Show that $\Sigma_{i=0}^{n} i = \frac{n(m+1)}{2}$ by induction (4marks).

3a i) For any finite set A, $|2^A| = 2^{|A|}$, is cardinality of any power set a is 2 raised to a power equal to cardinality of A.

OR

b ii) Explain Chomsky classification (12marks).

MGU/JDec2007

2 a i) Check whether the language $L = \{a^n b^n / n \geq 1\}$ is regular or not. Justify your answer.

ii) Explain diagonalization principle (12marks).

OR

b ii) Explain primitive recursive and partial recursive function (12marks).

MGU/July2007

1. Define primitive recursive function (4marks).

2. Explain composition of functions (4marks).

3a. i) Prove that the function $f_{add}(x, y) = x + y$ is primitive recursive.

ii) Prove that the function $g(x, y) = x^y$ is primitive recursive (12marks).

OR

b. ii) Explain the significance of Theory of Computation in computer science (12marks).

MGU/Jan2007

1. Prove that for any set A having n>=0 elements, the power set of a has 2^n elements (4marks).

2. Explain the types of functions- injection, surjection, bijection and invertible function (4marks).

3b. i) Explain the Chomsky hierarchy of formal languages.

ii) Define formal language. Give examples for finite and infinite languages (12marks).

MGU/July2006

1. Define a partial recursive function (4marks).

2. Differentiate between deterministic and non deterministic algorithms (4marks).

4a. i)

ii) Define context sensitive and regular grammars. Give examples.

OR

b. i) Find a function f(x) such that f(2) = 3, f(4) = 5, f(7) =2 and f(x). assume any arbirtrary value for other arguments. Show that f(x) is primitive recursive.

MGU/Nov2005

1. Give the formal definition of a grammar (4marks).

2b. i) Show that $f(x) = x/2$ is a partial recursive function.

ii) Explain the different types of grammars with suitable examples (12marks).

References

.

Linz, P (2006). An Introduction to Formal Languages and Automata. Narosa.

Moret, B, M (2007). The Theory of Computation. Pearson Education.

Mishra, K, L, P; Chandrasekaran, N (2009). Theory of Computer Science. PHI.

Pandey, A, K (2006). An Introduction to automata Theory and Formal Languages. Kataria & Sons.

Kavitha,S; Balasubramanian,V (2004). Theory of Computation. Charulatha Pub.

website: http://sites.google.com/site/sjcetcssz