Theory of Computation

Prof. Kamala Krithivasan

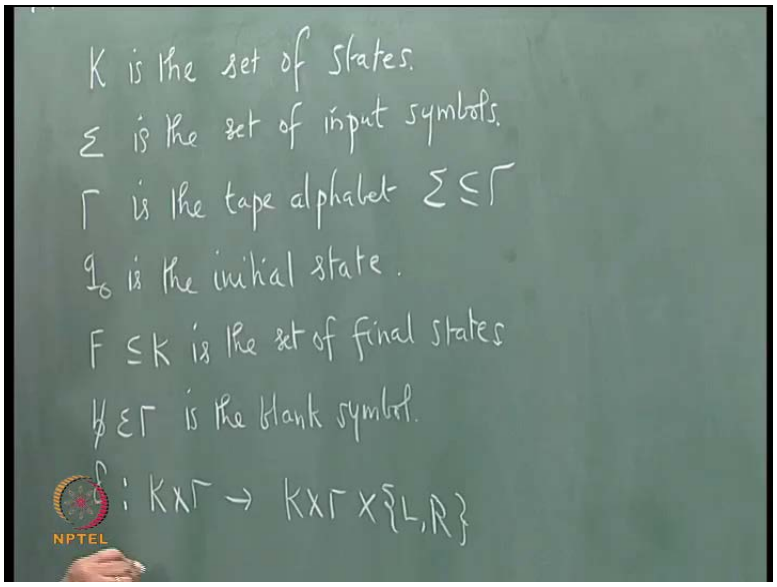
Department of Computer Science and Engineering

Indian Institute of Technology, Madras

Lecture No. # 28

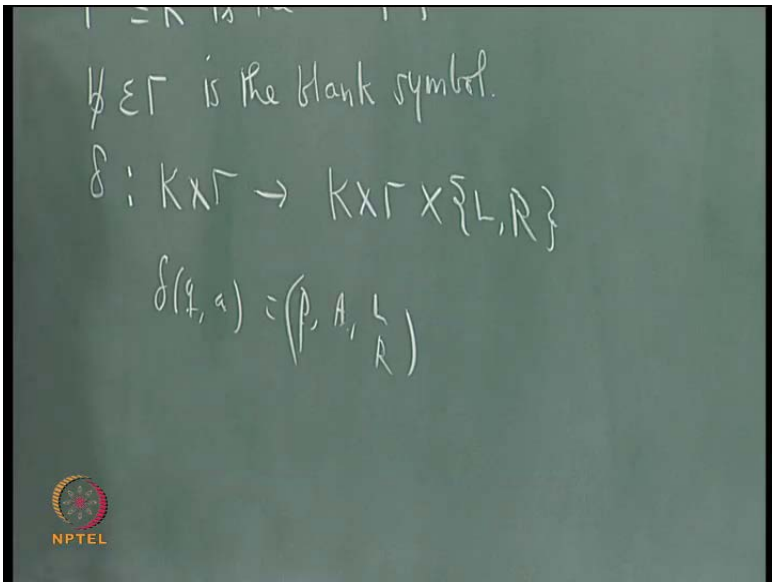
Turing Machine as Acceptor, Techniques for TM Construction

(Refer Slide Time: 00:24)



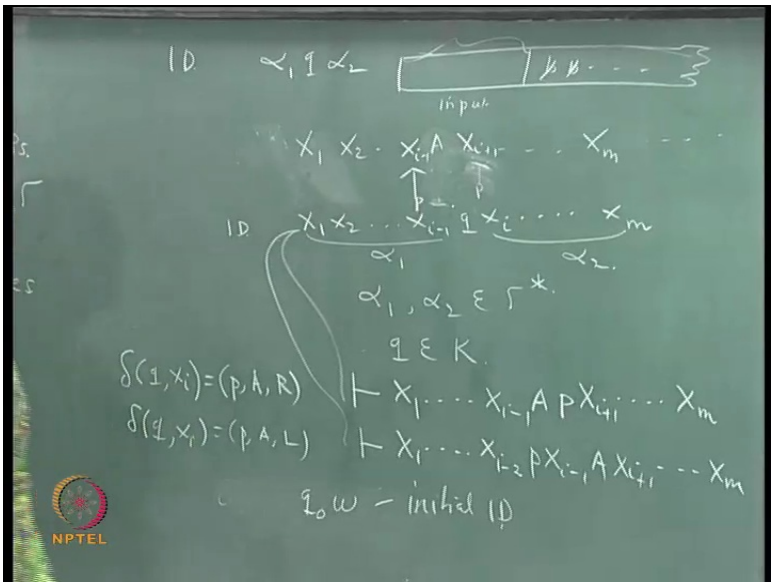
Today we shall consider the Turing machine as an acceptor, we have considered the Turing machine as an input output device; we will see how it can accept a language. So, in this case, the Turing machine say M is taken as a 7 tuple K, sigma, gamma, delta q 0, blank F; seven tuple. Sometimes it is given as a 6 tuple, that blank is not given separately, where K is the set of states, sigma is the set of input symbols or it is input alphabet, gamma is the tape alphabet and sigma is contained in gamma, q 0 is the initial state, F contained in K is the set of final states, the blank symbol belongs to gamma is the blank symbol, it is the blank symbol the symbol denoting the blank. And delta is the mapping delta it is a mapping from K into gamma into K into gamma into L, R.

(Refer Slide Time: 02:18)



Now, we are considering deterministic machine, deterministic tuning machine. If it is nondeterministic it will be K into gamma into finite subsets of K into gamma into L, R. So, in a formal way we denote the Turing machine by the seven tuple to the set of states set of tape alphabets, portion of it is in input alphabet q 0 is the initial state F contained in K is a set of final states and one symbol in gamma is denoted as the blank symbol and delta is a mapping from K into gamma into K into gamma into L, R that is the mappings are given in this form delta of q, a is equal to p, A, L R that means, if it is in state q reading the symbol a, it can go to state p rewrite over a the capital A and move left or right.

(Refer Slide Time: 02:28)



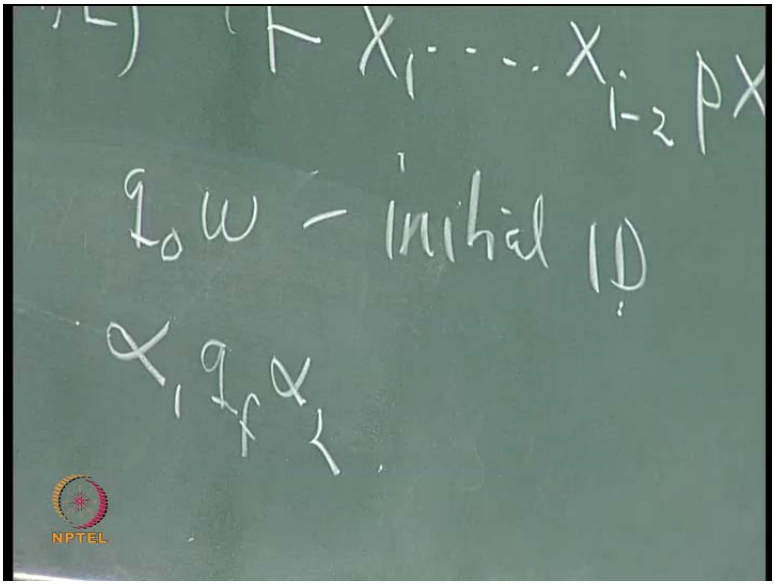
And I D are the instantaneous description is a string of the form alpha 1, q, alpha 2. See when you consider it as an acceptance device, the left end is fixed and the right end is infinite the input is initially given here faded up with blanks for accepting the Turing machine can write over the blank cells and use as many cells as it wants. So, the I D is described as a string suppose, the non-blank portion contains something like know x 1, x 2, sum x m. So, the non-blank portion see you must remember that, in between it may print a blank also so, but whatever follows this is blank on the tape this is left most symbol right. The left most symbol and this the nonblank portion followed by many blanks infinite number of blanks to the right.

If this is a situation and head is reading, the x i th symbol in state q head is pointing to this symbol and the state is q, the I D is written as x 1, x 2, x i minus 1 q, x i, x m this is the I D this is a string alpha 1, this is the string alpha 2. So, alpha 1 and alpha 2 they belong to gamma star, q belongs to K. So, the I D of the Turing machine or instantaneous description is given by a string, which denote the non-blank portion of the head position is marked here q, x i means the head is pointing to x i and the state is q. Now, suppose this is an I D, this is the I D current I D and you have the mapping delta of q, x i is equal to say p sum A, R it is a right move.

Delta of q, x i it is at reading x i it is going to state p, rewriting this as A and going to state p and it is going to read next symbol, that is the next I D then from this I D the next

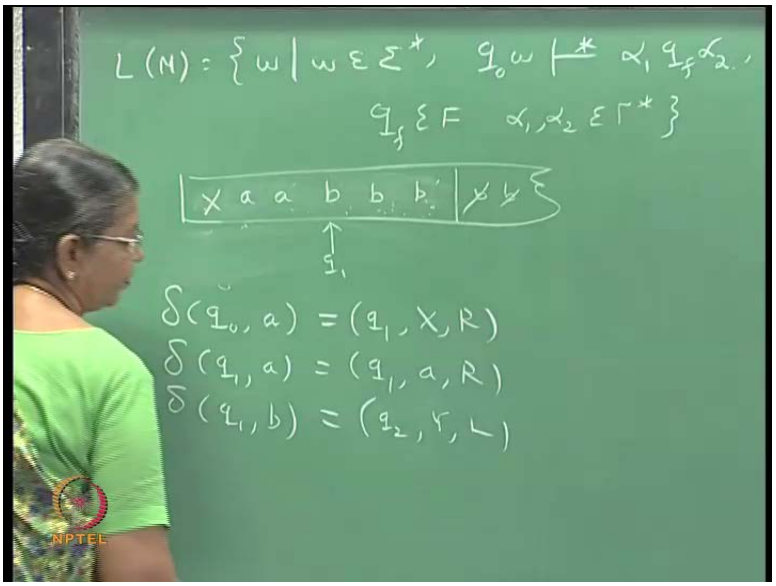
I D is written with the symbol the next I D is x 1, x 2, x i minus 1, x i has been replaced by A the next symbol is p and it is reading x i plus 1 head is reading x i plus 1 and this will be the next I D. If it is a left move delta of q of x i is p, A, L the left move from this I D it will go to the I D x 1 up to x i minus 2. Now, this will move left move means it will go here. In state p so p, x i minus 1 a, x i plus 1 x m this is the next I D is that clear, it cannot move left of the end. So, you cannot have a if you if you try delta of sum q, x 1 is equal to sum p, A, L this is not allowed it has to stop there it will halt.

(Refer Slide Time: 09:07)



And what will be the initial I D, initially the put is given here so, initial I D will be q 0, w, w is the initial input then the initial I D will be q 0, w this is the initial I D without loss of generality, you can assume that once it reaches a final state it halts. The machine, once it accepts it halts there is no loss of generality in assuming that we can do that. So, at the end there will be some configuration alpha 1, q of alpha 2 it will halt with some such I D where q f is a final state.

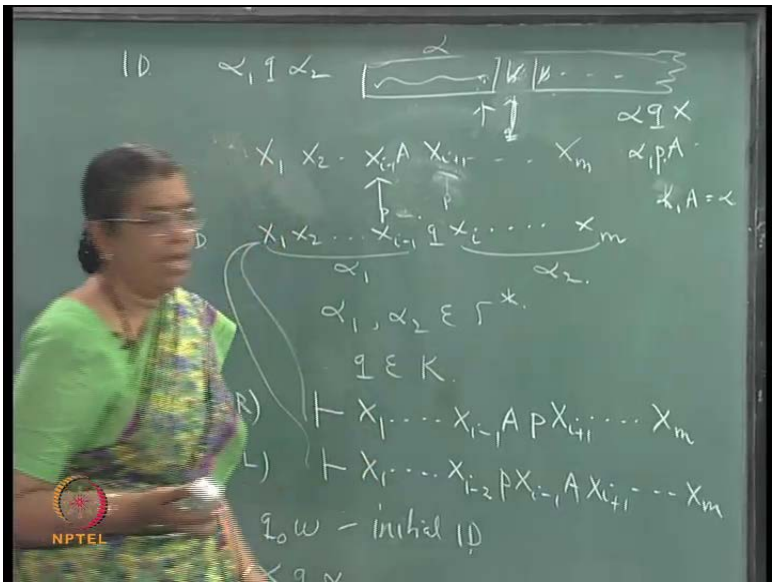
(Refer Slide Time: 09:54)



The language accepted is denoted by L of m or t of m. L of m is the set of strings of the form w, w belongs to sigma star and the initial I D is q 0, w because what does that symbol and then star means, if you get in one step you use this, this is as usual is the reflexive transitive closure. If you get in many steps you use the star. So, q 0, w is sum alpha 1, q, q f alpha 2.

Where q f belongs to F. alpha 1, alpha 2 there are strings over gamma star. So, if it is reading this, it can read this symbol, it can it can read here. Suppose, it is here in state q, this is the non-blank portion. Say some alpha after sometime initially, what is given is input after some time, some non-blank portion is there and the first blank symbol it is reading in q, this is alpha. Say then, the I D will be alpha q it is alpha 2 is 2 is empty epsilon, alpha 2 is epsilon, alpha 1 is alpha.

(Refer Slide Time: 11:37)



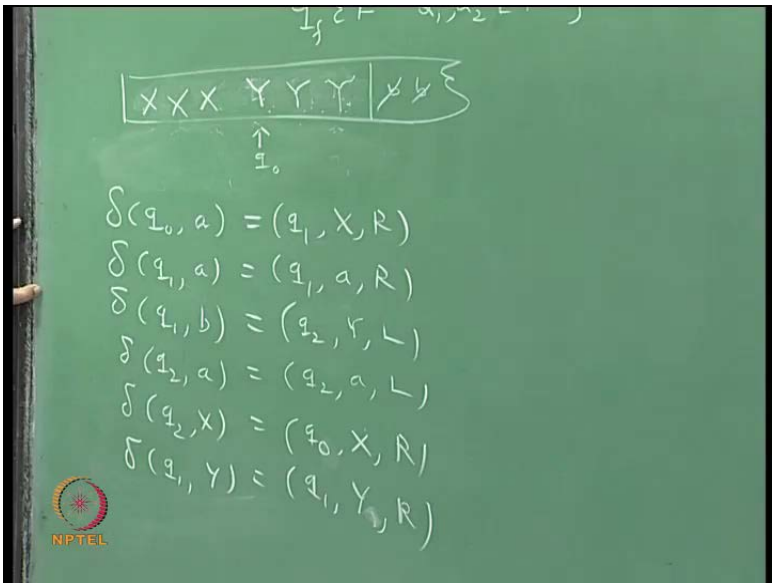
Now, it can print something here and move right or it would just there is another way suppose, the last symbol is sum x it is here in q so, the I D is this I D is alpha q, this is alpha say alpha x is the content of the tape. It is reading the last symbol x in state q. Now, suppose it prints a blank and moves left then alpha, whatever symbol is there it will be like this alpha 1, A is alpha, it will be reading this. So, alpha 1, A is alpha last symbol of alpha is A same thing only how this manipulated towards the end I am just mentioning.

So, the language accepted is given by this now, the way the definition is given there is no condition that it should read the whole input, it is not at all given that it should read the whole input, but for particular languages when you design the Turing machine, when you construct the Turing machine, you always make sure that the whole input is read.

Let us construct a simple example, a power n, b power n. How it will accept a power n, b power n, n greater, this can be accepted by a pushdown automaton, but not by a finite state automaton. So, as a Turing machine, how will you accept it? This is the one example, where I will write the whole moves in completely later informally, we will describe this can be written something and so on.

So, formal write in the delta move it will take lot of time because it is not difficult, but it takes time. So, this example, will arise the all the mappings.

(Refer Slide Time: 17:02)



The idea is like this, the input is given like this, a, a, a, b. Suppose, I am taking a particular example, followed by blanks, please note that this is different from the b. Now, this has to be accepted the machine is started in state q 0 reading a leftmost symbol.

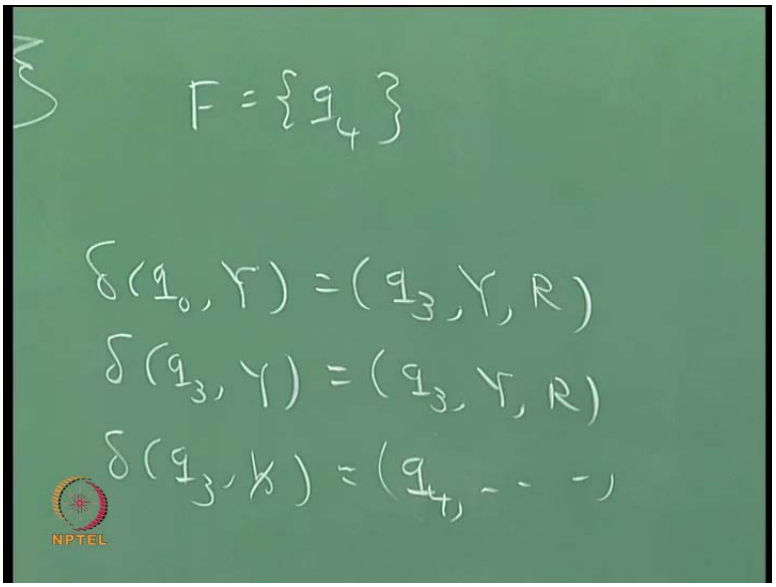
What it does is, it changes this into x goes here, changes the b into a, y comes back, changes this a into x, goes back, changes this into a, y comes back, changes this into x, then goes back, changes this into a, y mean while coming back it sees that there are no more a's. It also sees it goes back and checks that it is not there, there are no more b's remaining. It sees the blank symbol it will accept. So, how do you write the mappings? This is what you start with. So, delta of q 0, a is equal to q 1, X, R. So, first move a is changed into x and you move right. Now, in q 1, it moves right, it has to move right and it has to see the b, when it sees the b it will change state and start moving to the left. So, delta of q 1, a is q 1, a right this moves right and delta of q 1, b. When it sees this b in state q 1, it makes it into a, y and starts moving left .So, in state q 2 it starts moving left so, delta of q 2, a is q 2, a l it starts moving like that and then when it sees the x in state q 2, it has to start the process all over again. So, it moves right in state q 0 delta of q 2 x is q 2 sorry q 0 X, R.

So, it moves right instead q 0, the whole thing will be repeated again. So, in q 0 when it sees a it will change this into x and move right in state q 1. In q 1 when it sees a it will

move right, I am sorry this has been changed into y I D, not mark that now, in q 1 it looks for b right? It has to pass through y also, so delta of q 1, y is q 1, y R. So, when it comes here it will replace this by y and move left in state q 2. It will come up to this point when it sees this x it will move right in state q 0, another pass is over. right Now, again the same thing it will change this into x go to state q 1 and in q 1 it will pass through all these things and reach here in state q 1.

In q 1 when it is sees b, it will change that into y and come back, it will come back here in q 0, q 2 and then move right in state q 0. Now, in q 0, it is suppose to see a to start the next pass but it is seeing. What it is seeing? It is seeing y that means, it has exhausted all a's. So, delta of q 0, Y is it goes to another state q 3, Y, R. Now, it must see that there are no more b's remaining. So, in q 3 it just passes over the y's. So, delta of q 3, Y is q 3, Y, R and in delta, in q 3 if it sees a blank it will halt q 4. It will go to q 4 again. Say, q 4 and halt and q 4 is the final state. F is q to the final state so, for q 4 no moves are defined when it goes to q 4 it halts.

(Refer Slide Time: 19:30)



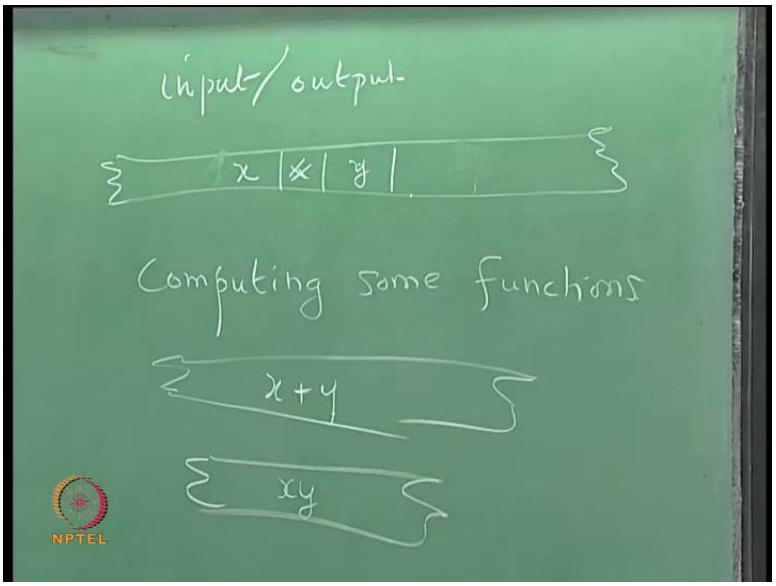
Now, what will happen if you have more a's or more b's? Suppose, I have one more a. What will happen? It will come here, in state q, 0 replace this by x and it will move in state q1 looking for b and in q1 it will see a blank. Move is not defined for that q1 blank so, when it reaches here in q 1 it has to halt. Now, move is specified, it has to halt but, it will not accept that. What will happen if you have more b's?

Suppose I have one more b here. So, three passes are over and you see here this y in q0. So, it will go to q3 and start moving right. So, in q3 it can pass through over the y's, but if it sees a, b; if it is a blank it will halt and accept it and will go to q4 and accept. If it sees b, no move is specified isn't it. It has to halt, but did not accept the input. So, here the acceptance, you can see that it has to see the first blank symbol and then only accept. That way we are making sure that it is seeing the whole input by the definition, it is not necessary.

(Refer Slide Time: 22:34)

So, what happens if something is accepted without going through the whole input after seeing this portion it accept say, what does that mean? It could be there, is some there are some symbols, not blanks that means anything followed sigma star that into sigma star, any string will be accepted, that is the main, but for all examples, you make sure that the whole input is read and it is the first non, first blank symbol, leftmost blank symbol and then only accepts.

(Refer Slide Time: 23:24)



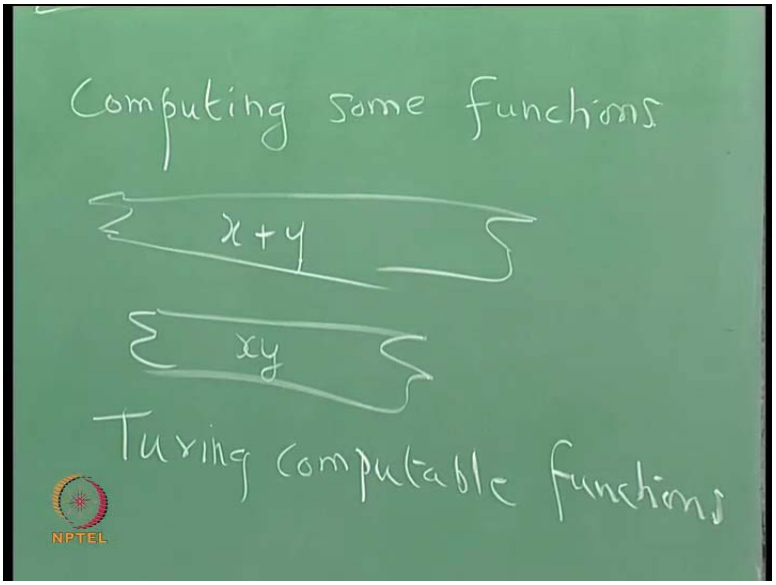
Now we can, we have already seen the Turing machine as input, output device. When you consider it as input output usually, you are not bothered about the left hand being fixed that is not really the tape, we can take as infinite, initially when you start it will be input, will be given and when end whatever, is given is the output you can look at is as computing some functions, computing some functions they are called Turing computable functions. So, for example I can be given x with a separator and y and you may end up with x plus y, answer x plus y.

So, ultimately this is the input and ultimately you end up with x plus y on the tape this is possible so, it is performing the operation of addition it could perform the operation of multiplication. So, you may end up with x, y on the tape. Now, if the numbers x and y are represented in unary. It is very simple, you have to just remove that hash symbol shift them shift the all the symbols one position, it will be over addition will be performed like, if it is binary you can do in some other manner.

But anyway we have seen that converting from one base to another base is not a very big problem, we can always we are not talking about efficiency, how much time it is going to take at this stage. Later on, we will talk about that also, but at this stage we are only thinking about, how the function is computed, not really how many steps it takes to compute because unary means something will be easy, but conversion from unary to decimal, decimal to unary, all those it will take time.

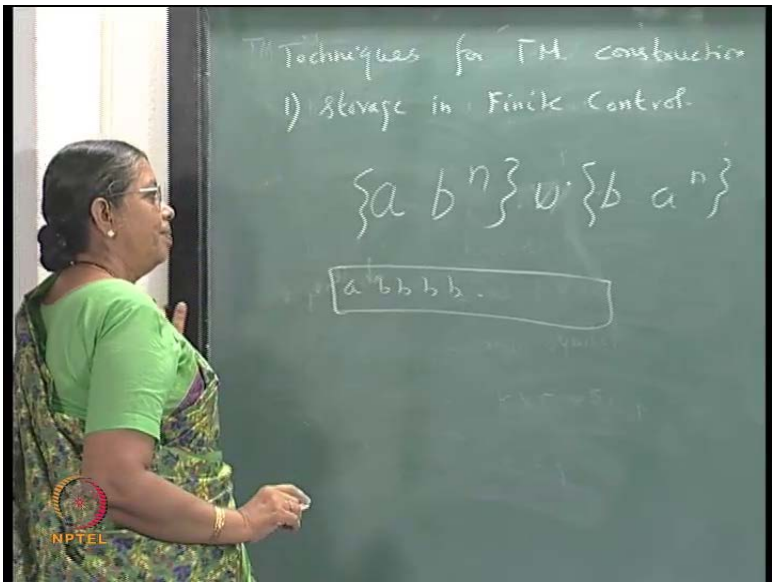
Then multiplication again you can do then at x power y, x and given x and y. How to compute x power y, all those things can be done. So, any function I am looking at them as natural numbers to natural numbers are some two, three arguments of natural numbers computing some function anything, anything for which you can write a programme for that matter you can compute with the Turing machine .So, these are called Turing computable functions.

(Refer Slide Time: 26:12)



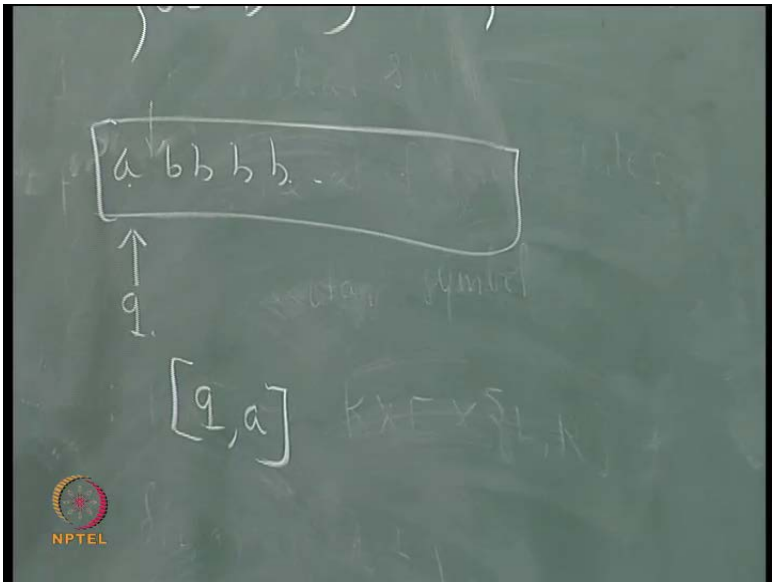
So, as I mentioned anything for which you can write a programme, you can do with the Turing machine and when you learn programming, you learn certain techniques. How to write a do loop, how to write a repeat until loop, how to, all those things when, where you have to use loop, where you have to (( )) so on. Similarly, there are techniques for Turing machine constructions. So, we will see I will not write the mappings one by one, but I will informally describe the techniques.

(Refer Slide Time: 27:20)



Techniques for Turing machine constructions, one is storage in finite control. What is finite control? Re-state really some amount of information you can store in the state. So, what does that mean? The state itself you have to take as a tuple, 3 tuple, 4 tuple or something like that, then you can store some information in that. Of course, that has to be finite, you cannot it if the state can be taken as a 2 tuple or 5 tuple or something like that, that has to be finite, you cannot have keep on increasing that.

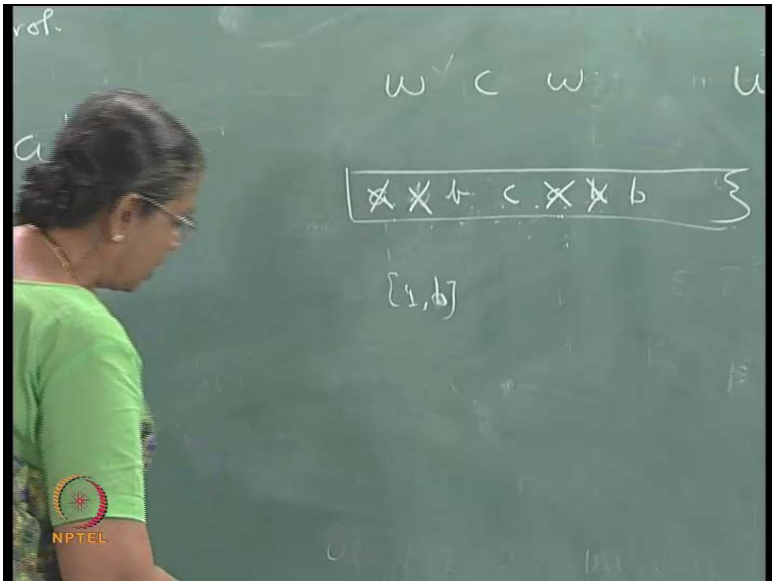
(Refer Slide Time: 28:00)



For example, suppose I want to accept a, b power n or b, at least first symbol should not

be repeated again or it can be over another alphabet, also a, b, c you can take first symbol should not occur afterwards. So, if it is like that in the store, when you treats the a the tape will contain something like this left. You can start here a, b, b, it has to accept such a language. So, when it reads this a the state becomes q, a it shows that it has read a in the state then it can move and at any time this should not match with this like that, you can have another thing is suppose, I want to accept w, c, w. W is a string of a's and b's.

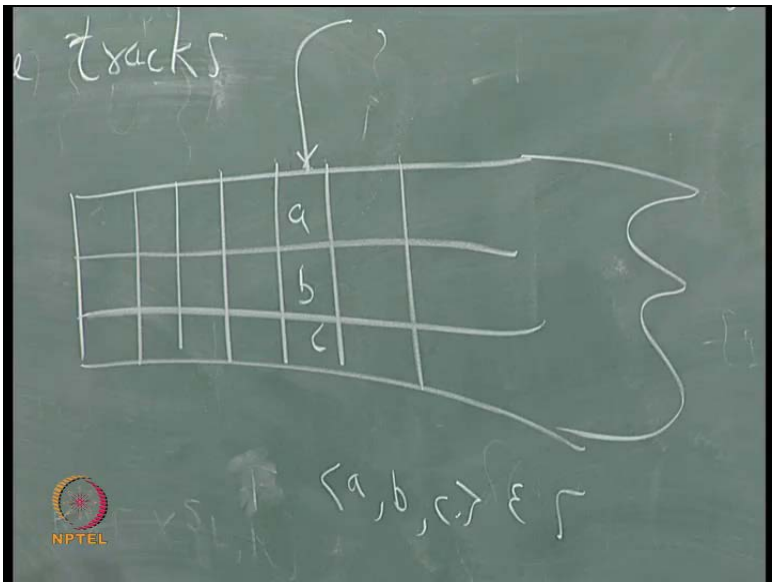
(Refer Slide Time: 29:32)



So, what we can do is left and suppose a, b, b, c, a, b, b. Now, this is not context free, this context sensitive, it is not context free, you cannot have a pushdown automaton accepting that. Now, the idea is, it can read this changes into x and store that information as q, a. So, that it remembers that it has read a and after moving through see the first symbol, it has to the first symbol has to be the same a, if it is not it will stop rejecting otherwise, it also changes this into x comes back.

Now, when it sees this b, it can changes into y, but store that b in the state then go back after the c, it will leave out the x's and the first symbol has to be again b it will change that into x, come back then, when it reads this change store that b in the component and so on. So, it can store some amount of information in the states and that that technique is called storage in finite control.

(Refer Slide Time: 31:02)



Then you can have multiple tracks. The tape so far, we considering only one tape, we are assuring that the Turing machine has got only one tape. In fact multi-tape Turing machines are there, you can have multiple tapes for the Turing machine finite number of tapes, but at present we are considering only one tape, but this tape can have multiple tracks, any tape can have multiple tracks.

So, for example, it can be like this, it may be divided into say three or four. Something like this, there is only one tape and so, only one head if it is multi-tape, for each tape there will be a tape head. Here there is only one tape and this head is pointing to this and suppose I have say a, b, c here in a sense it means. What does that mean?

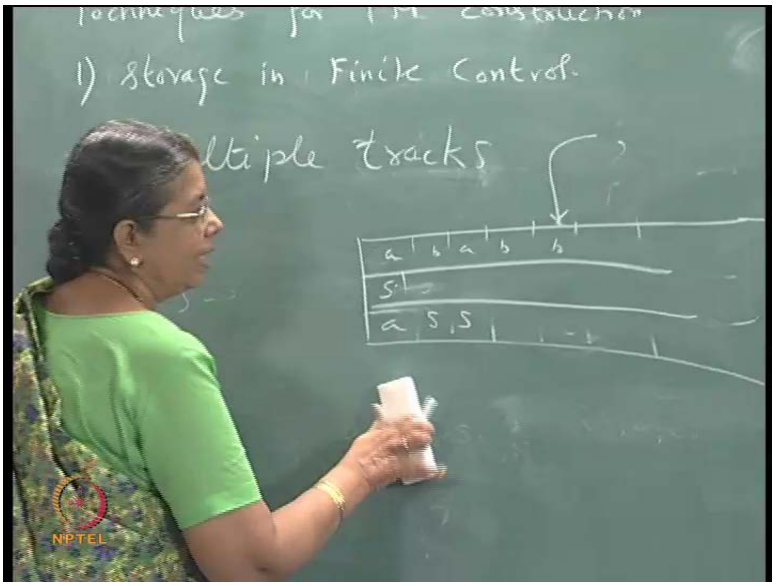
Gamma itself is taken as a triple, the symbol, this is a symbol of gamma. Here we have taken the first one storage in finite control. What we have done is, we have taken the state as a tuple and only one component will be really the state, the rest of them is used to store some information. Here multiple track means, the tape symbol sometimes it is, instead of using, so many symbols, it is convenient to use it as a tuple. If you want to change a middle one alone, you can do it. Suppose there are a, b, c three symbols in three places, there will be 27 symbols know, instead of using a 1, a 2, a 27 I use it as a triple, it is convenient.

Where you are going to change and this technique is really useful because sometimes, what you do is you may have something on the first track, you want to do something and

then check whether, at the end you get the same thing as the first one. For example, I want to some, this is, I want and I want to see whether this grammar, this string is accepted by the grammar, what we consider s goes to a s b or s goes to b, s or something like that. So, what you do is, in another track you keep you start with s, then start with this, start again replacing this and so on.

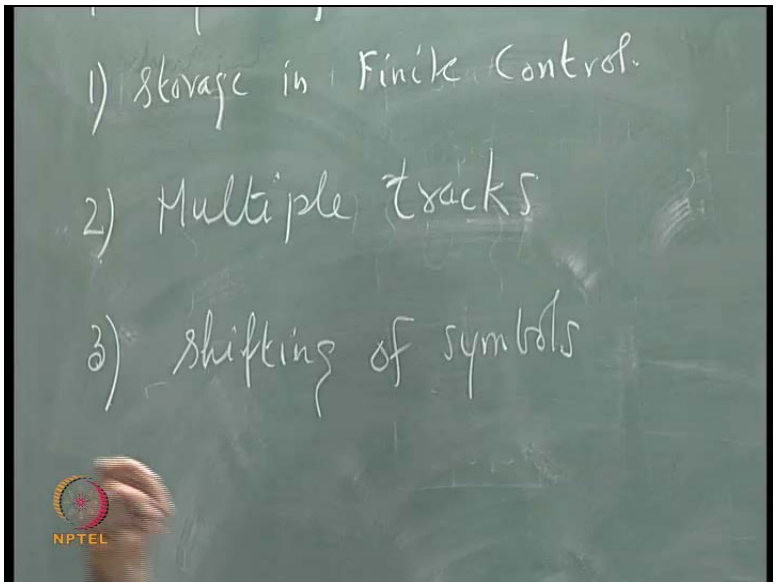
So, finally when you end with terminals, is it matching with that you can check. So, me sort of calculation or computation, we can use for the second and the third one and then the input compare with the input and so on. In a sense multiple tracks means the tape alphabet is taken as a tuple that is all. And this will be useful in many cases.

(Refer Slide Time: 33:24)

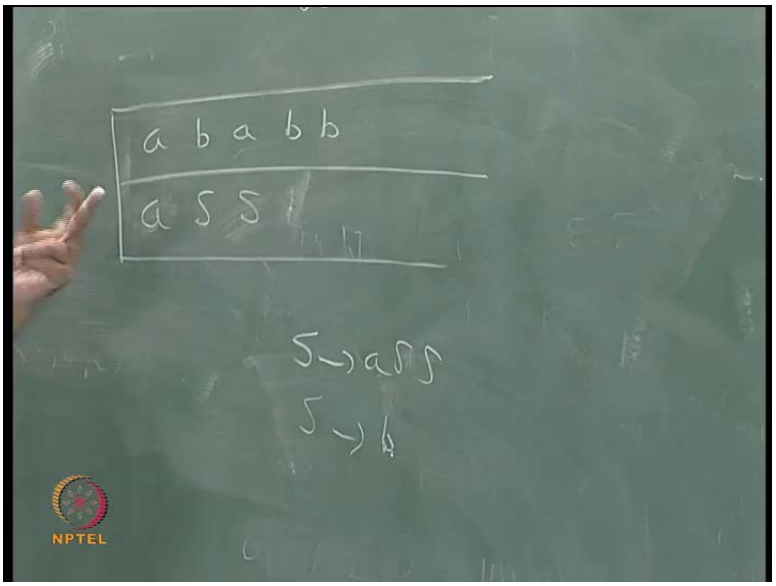


Shifting of symbols, again I will take the example of grammar, I want to simulate. I want to check whether something is a, b, a, b, b is derivable using the grammar. S goes to a, s, s goes to b. So, I will start with s and the second track replace it by a, s, s. Now, I want to change this, in the change this into b you can do that. Now, this s I have to replace by a, s, s. For this, I have to shift here, actually it is all blank.

(Refer Slide Time: 34:44)

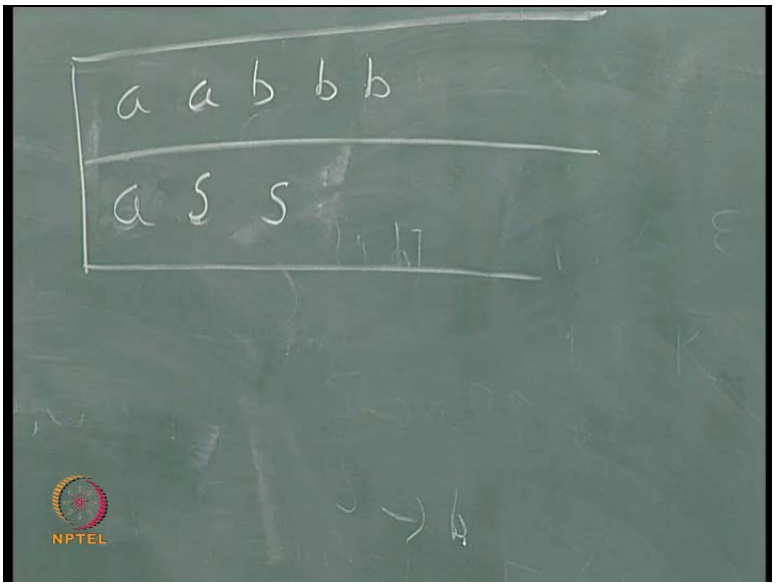


(Refer Slide Time: 35:06)

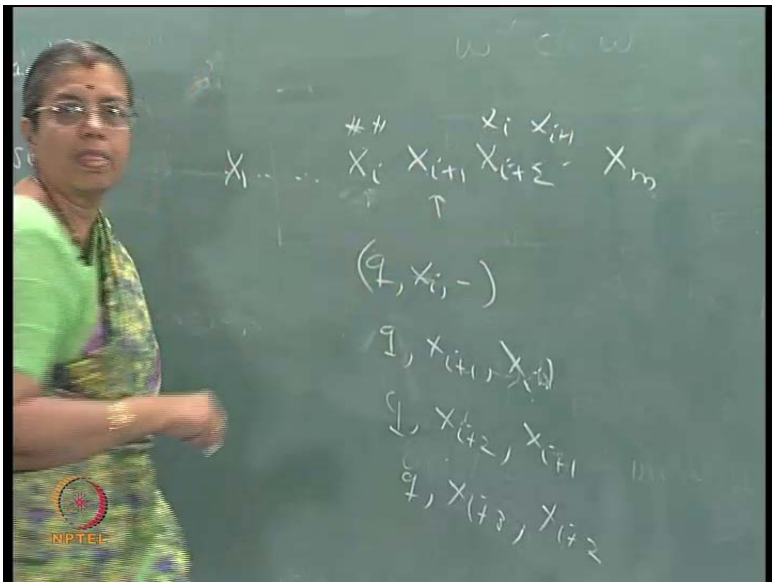


So, it is not very difficult I will take this string this is more, I will take this string it will illustrate in a better manner. So, s has been replaced by s. Now, this s has to be written as a, s, s but this I have to shift the s, this s cannot be written as a, s, s because there is no space here. So, I have to move these two places and then rewrite s. So many a times, I may have to create some space. So, How do you do this?

(Refer Slide Time: 35:35)



(Refer Slide Time: 36:36)



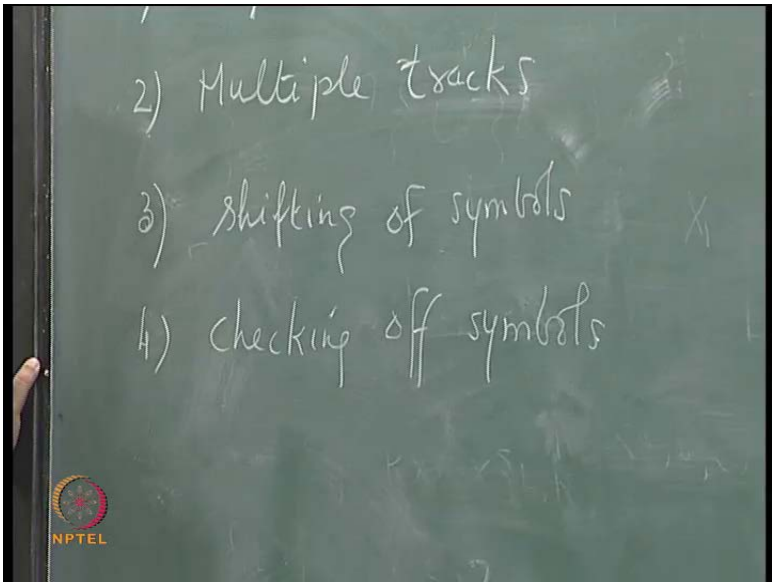
As a mapping, the delta mapping, how will you do this again? Suppose, I want some tape contains x 1 to x i, x i plus 1 x m. Now, I want to create two blank places here and shift from x i to this portion. I want to shift two places to the right so, that I can do something here so, what I do not know the length of this?

This fixed length, if it is fixed length, you can do in one way, but I do not know how long it is going to be, but whole thing I want to shift two places. So that I can write something. So, what you do is, you take this state as a tuple 3, 3 tuple, triple and then

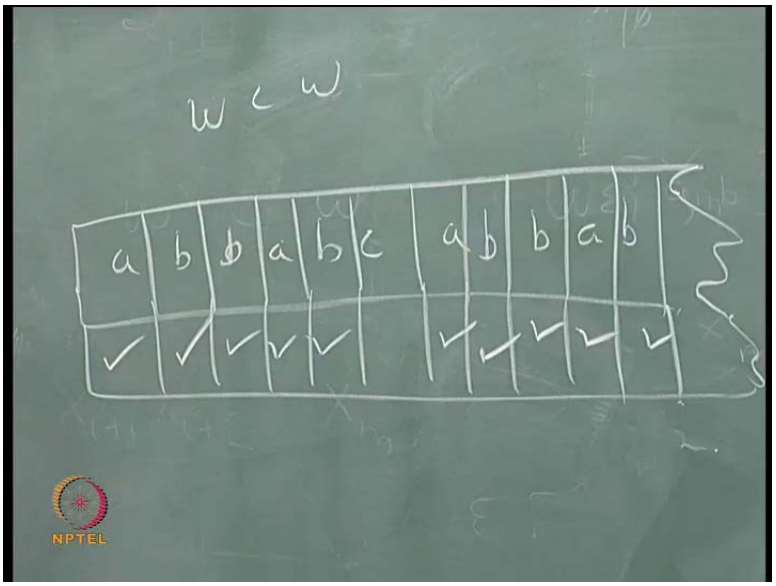
when you read this store, this x I here move right. When you read the x i plus 1 store the x i plus 1 here and move the x i to the third from second to third component, you move then next you will be seeing x i plus 2.

When you see x i plus 2 in an automatic manner, you store x i plus 2 in the second component. Move this to the third component and deposit this x I here. Systematically, you can do that, then next step x i plus 3 will be there that you store here, move this to the third component and whatever is there, you deposit here, that way you can shift everything, two places to the right and when you do the initially, when you store this you create two blank portions, here two blank portions, you create and later on you use it for some purpose, see this is shifting towards the right, you can also shift towards the left.

(Refer Slide Time: 39:02)



(Refer Slide Time: 39:18)

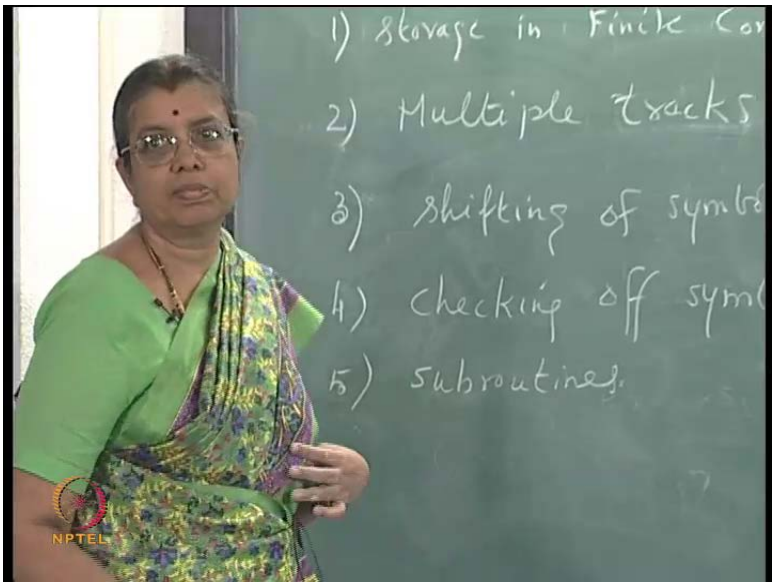


Checking of symbols. What does that mean? The same example which we which I considered w, c, w suppose I have a, b, b, a, b, c, a, b, b, a, b followed by blanks. As I said you can store some information in the state that is necessary again, but change everything into x and so on. So, I am losing the information, you can use x and y and then also you can some information you can keep. Another way of doing it, I do not want to do anything with this I want to keep them as a, b, b, a, b etc.

I do not want to do anything then have two tracks, the tape we have two tracks, what you do is, start from here, mark it off and that it has read one a. You remember, in the state then move pass this, then the first symbol check it is off. Again come back and when you come here, check the symbols, you leave out, uncheck the symbol first, uncheck the symbol you have to see. So, b store that information that it is b on the state move to the right, check this off, then move to the right and check this off.

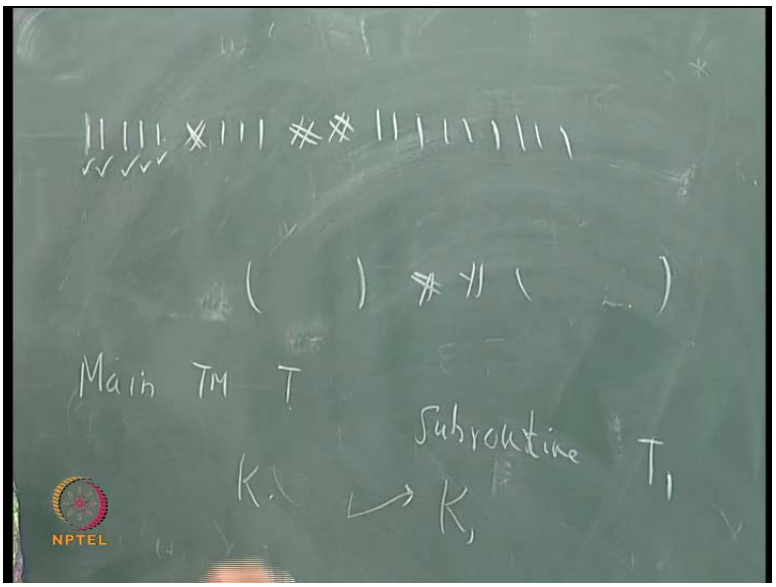
So, what essentially you are doing is, which symbol you are matching that you are storing in the state a has to match with a, b has to match, but whichever symbols you have marked, you have matched, you check, you put a tick mark and check. Finally, you make sure that everything has been checked. So, this is, you are using two tracks. The second track is putting a tick mark, that you have checked the symbol state also you are taking as a tuple and some information you are storing. In the state, this is another thing.

(Refer Slide Time: 42:11)



Subroutines this is a technique which usually do in programming, the same technique you can follow for Turing machines also. One Turing machine can be considered as a subroutine for another Turing machine.

(Refer Slide Time: 42:37)

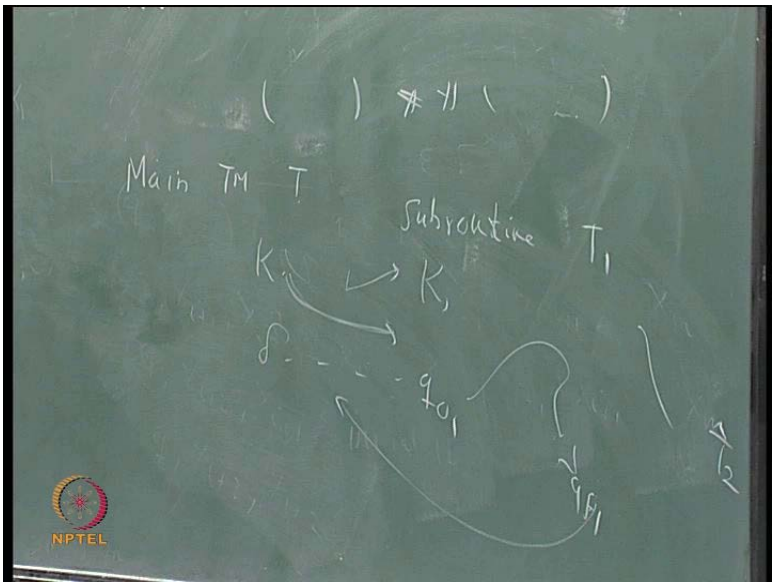


For example, if you want to multiply two numbers m and n given in unary m into n, 5 into 3 to end up with 15. So, what do you do? You make a copy of it answer I want here numbers are given. If you are given in binary, you can convert them into unary vice versa or something like that so, 2 unary numbers are given I want to multiply them.

So, in the end, I have to end up with 15 once. So, first the thing is you check this off for and then copy this here, then tick this off, copy this here, tick this off, copy this here and so on. We will end up with later on, you can erase this portion. So, what are the things which you are doing? You are copying. So, copying one portion here, this you are doing several times isn't it, each time when you check off a symbol here. So, for copying alone, you can write a Turing machine. You can write the mappings for delta mappings for copying one string here, then the main one it will check of a symbol and then copy this. So, copy will be called five times in this example, right.

Now, what are the things which you have to take care of, when you write the subroutine. Generally, when you have a programme, what do you do? Subroutine it will written a value and so on, So, here the state's main, main Turing machine is T subroutine is subroutine is T 1, the states should be designed, the the states of this and the states of this, they should be designed, they should obvious is it and then when you want to call the subroutine T 1 from here, delta mapping should take you to the initial state of this, if you go to the initial state of this that means you are calling the subroutine.

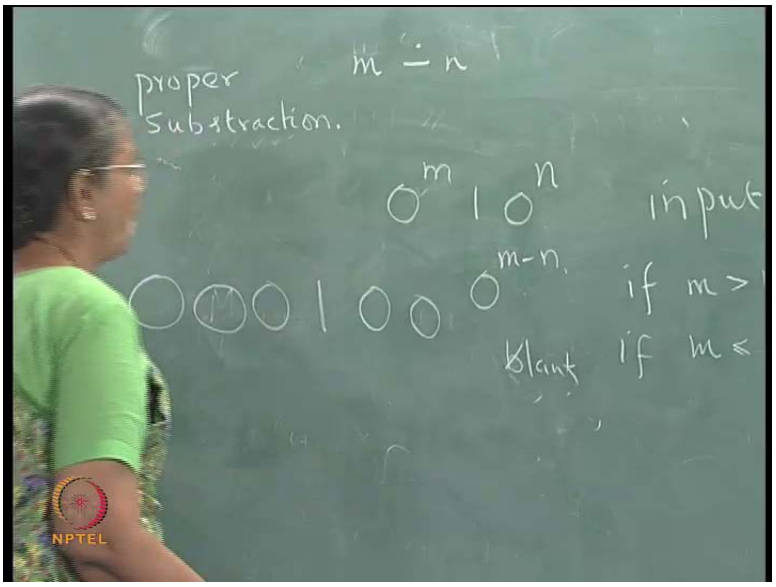
(Refer Slide Time: 45:22)



So, when you reach a final state here, you must get back to one of the states here or return to the value. So, a main routine and subroutine, you can also have this subroutine calling another subroutine T 2 and so on. Just as in programming, main routine can call subroutine, that subroutine can call another subroutine and so on. Again the same things,

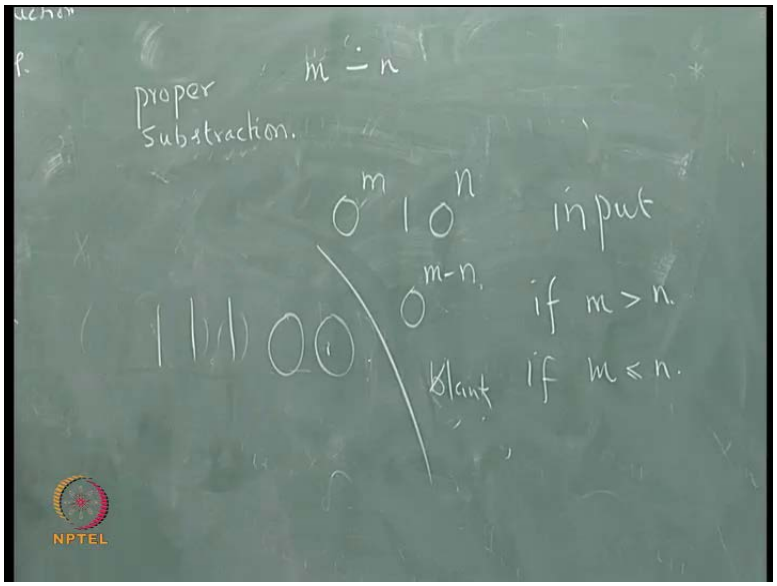
the states have to be designed and from the initial state you have to look into the other details.

(Refer Slide Time: 46:25)



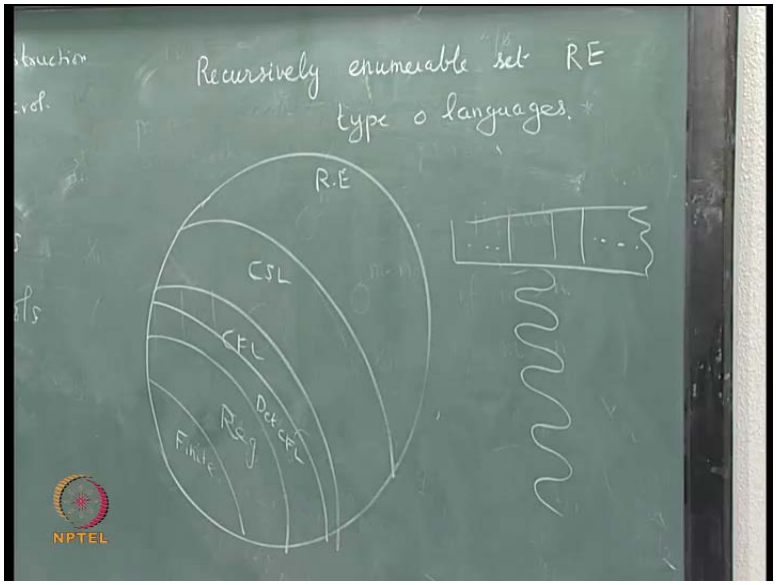
So, these are some techniques for Turing machine construction. One more example, suppose, I want to compute a function m minus n, this is proper subtraction. So, another how do you compute this? Input is given as 0 power m, n, 0 power n, this is the input you should end up with 0 power m n. If m is greater than or n blank tape, how can you do this? Suppose, I have this blank tape blank. So, what you can do is, change this into a blank, go here, change this into a 1, come back, change this into a blank, go and change this into a 1, come back and change this into a blank.

(Refer Slide Time: 47:45)



Now, looking ahead you see that there are no more 0s. So, what you do is, you make all of them blank, come back here and here you print two 0s, you will end up with two 0s. Now, if you have less for example, if you have less, you should end up with a blank tape. What will you do? Change this into a blank, then go and change this into a1, change this into a blank go and change this into a1. Now, when you are moving left, you find that no more 0s are left. So, erase everything. You end up with a blank tape. So, each problem, you have to look at in a separate way and decide, what is the procedure to be followed?

(Refer Slide Time: 49:38)



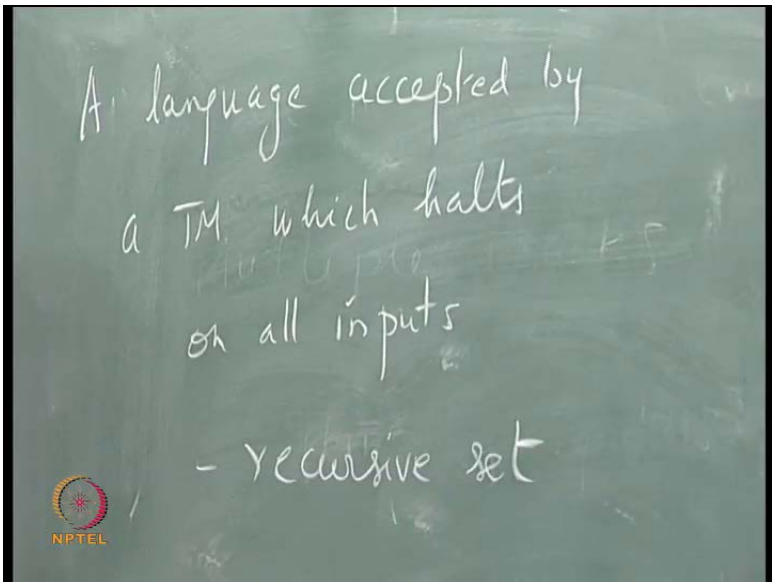
Now, the language accepted by the Turing machine is called the recursively enumerable set or R E, it is equivalent to type 0 languages. In the Chomsky hierarchy, finite sets are included in regular and regular sets are included in C F L isn't it and C F L is included in C S L, context sensitive language are type 1 and R E is up side this. Now, here comes deterministic C F L it is higher than regular, but less than C F L languages accepted deterministic pushdown automata.

Now, we have seen about acceptance by a Turing machine, but if a string is not accepted, then the Turing machine may go to a state from where it, it can go to a state where it will not be able to proceed further and it is not a final state. That is possible or in Turing machine, there is another thing possible, it may get into a loop. For example, it does not want to accept an input. So, after reading this is the input followed by blanks, after doing something, it starts doing the same thing again and again. Gets into a loop and it does not go to a final states none of the states is final.

So, maybe it is writing or just reading it may, move left and right left and right or just a few symbols and do the same thing again and again, none of the states is the final states. So, the string cannot be accepted by the Turing machine, but the machine does not halt, so for accepting it has to halt and accept for accepting string. A Turing machine has to stop, it has to halt and accept for rejecting string. It may go to a position, where it will not be able to make further moves.

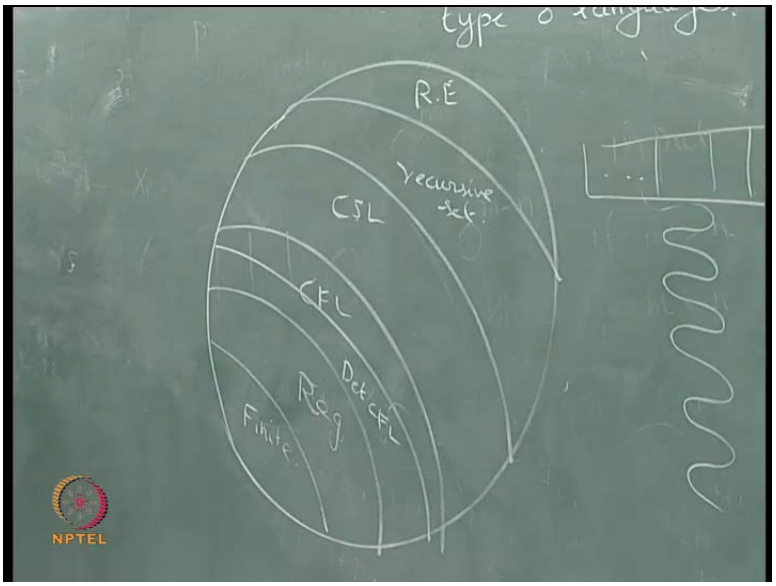
So, it halts in a non-final state. If you have to say, it in a that way or it can get into a loop, this is a very important point you have to note. Now, a set accepted by a Turing machine which halts on all inputs is called a recursive set.

(Refer Slide Time: 53:02)



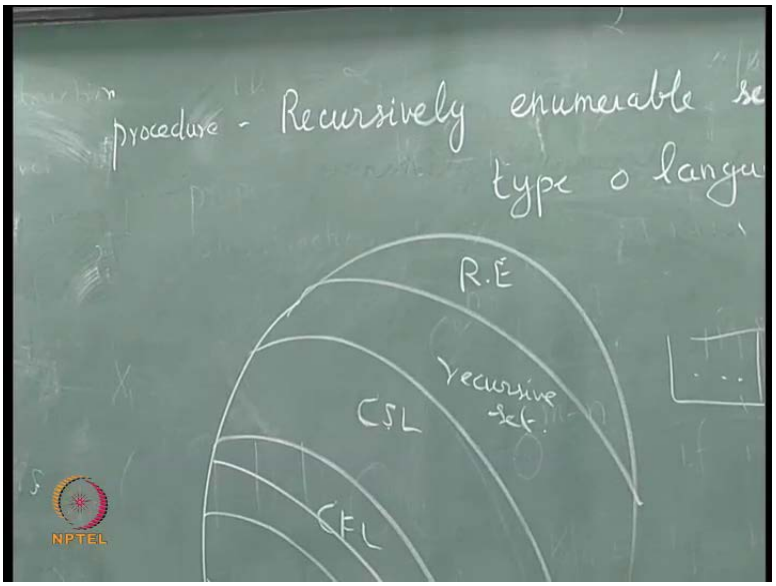
A language accepted by a Turing machine which halts on all inputs is called a recursive set.

(Refer Slide Time: 53:39)



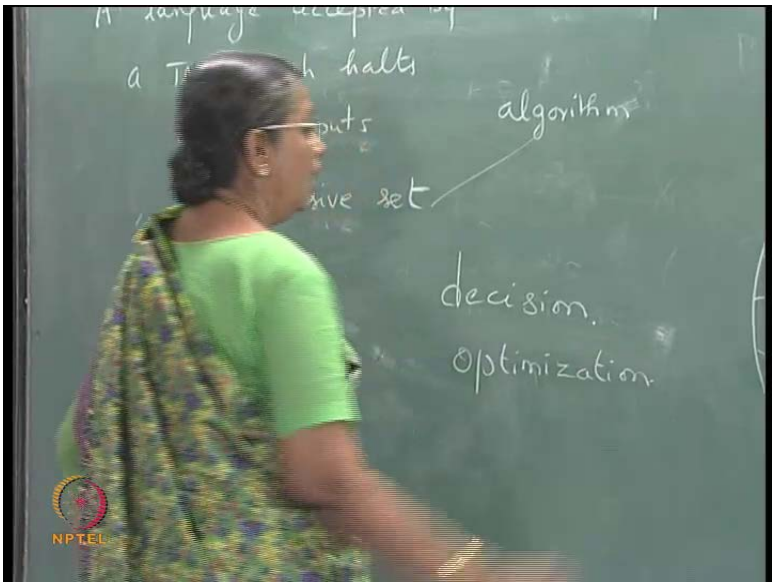
So, recursive sets are higher than this, they correspond to Turing machine which halt on all inputs. I told you, when earlier that there is a small difference between a procedure and algorithm.

(Refer Slide Time: 54:02)



A procedure corresponds to this, if the answer is yes it will give the answer, but if the answer is no, it may get into a loop. It may go on trying, trying and it may never end up.

(Refer Slide Time: 54:18)



Whereas, an algorithm corresponds to a recursive set that is, an algorithm always halts and tells you an answer yes or no. So, when you say decidable, I told you given context free grammar is it ambiguous or not? You cannot give an algorithm for that, that is called an undecidable problems. We shall study about that in next one or two classes. So, that means, you cannot write an algorithm for that, even if you are able to write, I mean

procedure it will be undecidable, I mean if you are not able to give an algorithm, it is undecidable. It ultimately you must say, yes or no and the problem generally, decision problem and optimization problems.

Decision problems are the answer is yes or no, does the graph have a Hamiltonian circuit or the travelling salesman problem that is it have a path a solution that is yes or no. Find the one which has the least cost or something like that that is an optimization problem. So, you can look at the problems or optimization problem or decision problem. Mainly, we will be considering only about decision problems. Yes or No answer of course, you can look at optimization problem as a decision problem by converting it in a little bit slight different way, but generally, we are more concern with decision problems.

And the one problem which we will study two of them, we will study in detail. One is the post correspondence problem and another is the Cook's theorem, that is Boolean satisfiability np-complete, both we will see how the idea of Turing machine is use that. So, we shall consider more generalizations of Turing machines and restricted versions of Turing machine, later in the next class.