

**Theory of Computation**  
**Professor. Subrahmanyam Kalyanasundaram**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Hyderabad**  
**Formal Definition of a Turing Machine**

(Refer Slide Time: 00:17)

Computability Theory

Now we see Turing Machines - which are an abstract model of modern day computers. The main features of Turing machines are:

- Read/write head
- Head can move left or right.
- Infinite tape (to one side)
- Special accept/reject states which immediately come into effect.

Diagram: A box labeled "State Control" is connected by an arrow to a horizontal tape containing the sequence "0110011011...".

Now we see Turing Machines - which are an abstract model of modern day computers. The main features of Turing machines are:

- Read/write head
- Head can move left or right.
- Infinite tape (to one side)
- Special accept/reject states which immediately come into effect.

Diagram: A box labeled "State Control" is connected by an arrow to a horizontal tape containing the sequence "0110011011...". This diagram is circled in red.

Consider the language  $D$ .

Hello and welcome to lesson 27 of the course Theory of Computation. So, we completed chapters 1 and 2, which talked about regular languages and context free languages. In this lecture, we will start with Turing Machines. So, we saw regular languages and context free languages. So, we saw that regular languages used machine models like deterministic finite automata and non-deterministic finite automata. In context free languages we saw pushdown automata.

In chapter 3, we see Turing machines. So, which are actually an abstraction of modern-day computers. As we will see, these will be more powerful than the models that we have seen so far. And so, this is the beginning of the computability theory part, so, at the beginning of the course, you may recall that I had promised that the goal of the course is to understand computation and to see what can be computed and what cannot be computed.

Surprising as it may seem, there are things that even computers cannot do, which we think are quite powerful and can do all sorts of computations, there are questions that even computers cannot answer even if you give them all the information and all the time in the world. So, let us see how the Turing machines differ from the automata that we have seen so far.

So, they differ in 3, 4 important aspects which I have listed down here. So, in many other cases we have a state control and a tape which contains the input. So, if you recall in DFAs NFAs, PDAs, etcetera. just one input symbol is read in each step or sometimes there is an  $\epsilon$  transition, but it is just, the input is just read and at the end if we are an accepting state we accept if you are not an accepting state we do not accept. So, the first difference is that we have a read and write head.

So, there is an input which is given in some tape and the head can read as well as write. So, when you read the input, it can also write something onto the tape, it could overwrite the contents of the tape, it could erase the contents and write something there. So, the first difference is that we have a read/write head. The second difference is that in PDAs and NFAs and DFAs, we just read the input symbols one by one till we reach the end and then that is it. If we are at an accepting state at the end, we accept. Here, the head could move left or right.

So, there was no concept of the head moving to the left and right in the DFA or in PDA. So, in Turing machines, the head could go to the left or to the right it could... so, it could take any trajectory based on how the Turing machine is programmed. The next thing is that we have an infinite tape. So, in DFAs, NFAs, etc. we had just the input given and that was it, there was no notion of extra space or anything. Here we have an infinite tape and a tape that extends to infinity on one side.

So, there are different models, which can extend to infinity in both directions, etc. But this is the first model that we will see and we will stick with this model for most of the time. So, the left side is bounded and the right side extends to infinity. And finally, we had accept states in DFAs and NFAs and PDAs.

So, it was like this: we do the processing, you may be jumping across many states, you may go to the accept state, but maybe there is more string remaining. What really mattered is where we are at the end of processing the string. So, if at the end of the processing the string, if we are at accepting state, we would accept that string. It does not matter if in any intermediate state we happen to move to an accepting state. It does not matter, all that counts is whether we are at an accepting state at the end of processing.

However, in Turing machines, there is a difference, we have special accept /reject states, which immediately come into effect, meaning if you ever move to an accept state, that is it. The computation is over and we have accepted the input. If we move to a reject state, again, that is it. The computation is over and we reject the input. So, these are the 3, 4 distinctions that we have in Turing machines.

(Refer Slide Time: 05:38)

| 0 | 1 | 0 | 0 | 1 | 0 | 1 | ...

Consider the language  $D$ .

$$D = \{ w \# w \mid w \in \{0,1\}^* \}$$

$D$  is not a CFL. Can a Turing machine recognize  $D$ ?

On tape:  $\phi \ x \ y \ \phi \ \phi \ \# \ \phi \ x \ x \ \phi \ \phi \ \_ \ \_ \ \_ \ \dots$

Reject  $\begin{cases} \phi \ x \ x \ \phi \ \# \ \phi \ x \ x \ \phi \ \_ & \text{blank} \\ \phi \ x \ y \ \phi \ \_ \ \# \ \phi \ x \ x \ \phi & \text{reject} \end{cases}$

TM can do the following:  $\phi \ y \ 1 \ 0 \ 0 \ \# \ \phi \ 0 \ 1 \ 0 \ 0$

1. "Strike off" the first position from the left side, which is not already struck off. Remember this symbol, go past # and cross check with the first



TM can do the following:  $01100\#01100$

1. "Strike off" the first position from the left side, which is not already struck off. Remember this symbol, go past # and cross check with the first symbol in the right of # that is not already struck off. If not the same, reject. Else continue, or repeat. If no symbol is found after #, or no # symbol is found, then reject.

2. When all the symbols before # has been struck off, check if any more symbols remain after # (that are not struck off). If yes, reject. Else, accept.

We usually do not go into this level of detail.

or repeat. If no symbol is found after #, or no # symbol is found, then reject.

2. When all the symbols before # has been struck off, check if any more symbols remain after # (that are not struck off). If yes, reject. Else, accept.

We usually do not go into this level of detail. Only the high level idea will be explained. Most of the things can be accomplished, though the details could be tedious.

Q: How can a TM recognize  $\{w\#w \mid w \in \{0,1\}^*\}$ ?



$$D = \{w\#w \mid w \in \{0,1\}^*\}$$

So, at the end of the previous lecture, we mentioned this language  $w\#w$ , where  $w$  is a binary string. And we said that it is not a CFL. So now let us try to see how a Turing machine can recognize this string. So, what the Turing machine will do is the following. So, I have written the string here. So, one string is  $01100\#01100$ . And as you can see, this string is in the language because it is  $01100$  which is  $w$ , hash, followed by the same string. And these things over here are blank symbols, because there is an infinite tape.

After where we have the string, it is entirely blank symbols. But that does not matter, it is like blank symbols, which are distinct from the inputs. Let us see how a Turing machine might try to see whether this is part of the language that it has to recognize. So, what it will do is, it will

start from the left side and see the first symbol in the left side. So, the first symbol in the left side is a 0. So, it can strike off the 0.

So, what do I mean by strike off the 0? So, it can replace the 0 with some symbol let us say like, like this is 0 with a strike off or a you can replace it with an X or something like that. So, when I say strike off, it means that it has to replace it with something which says that it has already been read and then the Turing machine has to remember that it has struck off a 0 and then it starts moving to the right side. Slowly, it starts moving to the right side and it will keep moving till it encounters the hash symbol.

So, the hash symbol is this, #. So, once it reaches the hash symbol, it remembers that it had struck off a 0, then it sees the next symbol. So, the next symbol is 0, and the struck off symbol also was 0. So, now it strikes this 0 off also and it continues. So, now it comes back to the left side. Now, from the left side, it checks for the first symbol which is not struck off and the first symbol that is not struck off is 1.

So, now it strikes off 1. It remembers that it has struck off 1 and then it keeps going to the right side till it encounters the hash symbol. Once it reaches the hash symbol, it now starts checking for the next symbols and it sees a 0. It sees something that was struck off and the next symbol that is not struck off is a 1. So again, it strikes up this one. So, basically the first 2 symbols 01 and the 01 after the hash have been kind of matched.

Now, it comes back to the left side, the first symbol that is not struck off is 1, it crosses the hash the first symbol that is not struck off is 1, so things are fine. It comes back, the first symbol that is not struck off is 0, fine. Suppose it happens so that the symbol that we have here was a 1. So, now it will come to the left, the first symbol that is not struck off is 0, it remembers that it struck off a 0. The first symbol that is not struck off on the right side of the hash is a 1, so now it knows that the symbol that it struck off on the left side was 0, and the symbol that we are seeing now is a 1.

We know they are not equal. So, then it will be rejecting this. So, it knows they are not the same. And for that matter, it did not wait up to like. So now, if it is indeed a 0, then it will check whether it is 0 on the left side and check whether it is a 0 on the right side and strike off and then it will come back to the left and see whether there is any symbol to the right that is not struck off. And it sees that everything is struck off till the hash so the first symbol that it sees from the left side that is not struck off as a hash and then it goes to the right side and sees if

there is any leftover symbol that is not struck off. So, suppose there was a symbol here which is 0.

So, that means that the string to the left string to the left of the hash is not the same string to the right to the hash. So, this means this will get rejected because there is something to the left of the hash. Suppose, there was no 0 here and if it was just blank. In this case it will get accepted, because everything to the left is struck off and it matched with everything to the right and everything to the right is also struck off.

But, if you had some other things like let us say if you had 0110#01100 this will not be accepted because it will strike off 00, 11, 11, 00 but then it will see a 0 that is left, not struck off. And same thing, if it found something like let us say 01101#0110. Suppose this is the entire string. Again, it will strike off 0 and 0, 1 and 1, again 1 and 1, 0 and 0 and then it will come and strike off this 1, the last 1 in the left side, but then when it comes to the right side, there is nothing to match it with. So, even in this case, it will get rejected. So, these two will get rejected. And an even simpler string like I will just write, let us say, 01100#00100.

So, even this will get rejected because the first 0 will get struck off with this. This one, the second symbol one will not get struck off because the second symbol after the hash is a 0 which does not match. So, this will also get rejected. So, this is how a Turing machine may operate on a language like this. So, if the Turing machine is trying to identify that language, this is what it has to do. And the same thing I have written in a bit of detail. So, what does it do?

It strikes off the first position from the left side which is not already struck off. It remembers a symbol that was struck off, moves to the right, goes past the hash symbol and cross checks with the first symbol to the right of the hash that is not struck off. If it is not the same then it rejects else it continues or repeats. Suppose it struck off something in the left, but there is no symbol to the right side, then it rejects.

Suppose it strikes off something on the left, but there is no hash symbol at all. If there is not even a hash symbol, then it is not even of this form. Even then you reject. Now, finally, if all the symbols to the left side have been struck off and it has matched with whatever is struck off on the right side, it is still possible that it is not in the language because there could be extra symbols on the right side.

So, check if there are extra symbols. If there are extra symbols you reject. If there are no extra symbols, it means whatever was struck off from the left matched with whatever was there in

the right. In that case you accept. So, these are the kind of intricate details on how a Turing machine may operate.

So, we can formally define the Turing machine and all of this. Like we saw the transition function in DFAs, and NFAs, we can define a transition function for the Turing machines as well. And so that these kinds of rules, so, here I have written the proceedings or the way the Turing machine has to operate in English, but this is kind of high level and we can encode that into the rules of the transition function in a bit of detail. But that is kind of extremely intricate and cumbersome to do.

The point is that almost anything that we want to do like this, where we give instructions, can be accomplished. So, we will not get into those kinds of details. We will not get into minute details as to how we accomplish these rules in a Turing machine. We will just give high level description such as this. So, the details are tedious, the main point that I want you to know is that all of this can be accomplished.

(Refer Slide Time: 15:10)

(that are not stuck off). If yes, reject. Else, accept.

We usually do not go into this level of detail. Only the high level idea will be explained. Most of the things can be accomplished, though the details could be tedious.

Q: How can a TM recognize  $\{w \mid w \in \{0,1\}^*\}$ ?

Formal Definition of TM (Def 3.3): A Turing machine is a 7-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$  where  $Q, \Sigma, \Gamma$  are finite sets, such that

$q_0 \in Q, q_{acc} \in Q, q_{rej} \in Q$





State Control

0 1 0 0 1 0 1 1 ...

Consider the language  $D$ .

$D = \{w\#w \mid w \in \{0,1\}^*\}$

$D$  is not a CFL. Can a Turing machine recognize  $D$ ?

On tape:  $\phi x y \phi \# \phi x x \phi \phi w w \dots$

Reject  $\left\{ \begin{array}{l} \phi x x \phi \# \phi x x \phi \_ \text{blank} \\ \phi x y \phi \_ \# \phi x x \phi \text{ reject} \end{array} \right.$

TM can do the following:  $\phi x 1 0 0 \# \phi 0 1 0 0$

1. "Strike off" the first position from the left side,



So, one small question at this stage. Suppose we want to recognize this language,  $\{ww \mid w \in \{0,1\}^*\}$ . So, it is the same as  $D$  here, but there is no hash symbol. So, in our algorithm here, we try to go past the hash and then look for the hash etcetera. If there is no hash, how will it recognize this? How can a TM recognize this? So, it has to identify the midpoint. How will it identify the midpoint? So, this is one thing that you can think about. Perhaps it can do some kind of counting or something like that.

This is something that I want you to think about. So, what it can do is: one is that it can use a counter of some sort, it can count how many symbols are there overall, and then it can identify which should be the midpoint. If the count is an odd number, then you know that it is not of this form, because  $ww$  means a string is repeating. So, the length is even. If the length is even, let us say it is of length 20, then we know that the symbol, the point after the tenth symbol, is the midpoint. Once we identify the midpoint, it is pretty much the same.



(Refer Slide Time: 16:43)

Q: How can a TM recognize  $\{ww \mid w \in \{0,1\}^*\}$ ?

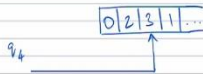
Deterministic

Formal Definition of TM (Def 3.3): A Turing machine is a 7-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$  where  $Q, \Sigma, \Gamma$  are finite sets, such that

$$\bullet \Sigma \subseteq \Gamma, \perp \in \Gamma, \perp \notin \Sigma.$$

$$\bullet q_{acc} \neq q_{rej}$$

$$\bullet \delta: (Q, \Gamma) \rightarrow Q \times \Gamma \times \{L, R\}$$



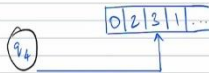
$$\delta(q_4, 3) = (q_6, 0, R)$$

where  $Q, \Sigma, \Gamma$  are finite sets, such that

$$\bullet \Sigma \subseteq \Gamma, \perp \in \Gamma, \perp \notin \Sigma.$$

$$\bullet q_{acc} \neq q_{rej}$$

$$\bullet \delta: (Q, \Gamma) \rightarrow Q \times \Gamma \times \{L, R\}$$



$$\delta(q_4, 3) = (q_6, 0, R)$$

$$\delta(q_6, 3) = (q_{10}, 1, L)$$

\* The input is placed on the tape to the left most end.  $w = w_1 w_2 \dots w_n \in \Sigma^*$ . The rest of the cells are blank.  $\perp$  indicates the blank symbol.

Reject  $\{ \overset{\sim}{0} \overset{\sim}{1} \overset{\sim}{0} \overset{\sim}{1} \# \overset{\sim}{0} \overset{\sim}{1} \overset{\sim}{0} \}$  reject  
 TM can do the following:  $01100 \# 00100$

1. "Strike off" the first position from the left side, which is not already struck off. Remember this symbol, go past # and cross check with the first symbol in the right of # that is not already struck off. If not the same, reject. Else continue, or repeat. If no symbol is found after #, or no # symbol is found, then reject.

2. When all the symbols before # has been struck off, check if any more symbols remain after # (that are not struck off). If yes, reject. Else, accept.



So, formal definition of the Turing machine. This is in fact a Deterministic Turing machine. There are many variants of the Turing machine that we will see. There is a Deterministic Turing machine. And again, I am not writing the definition in full detail, but just the main parts that I want you to focus on.

So, Turing machine is a 7 tuple,  $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ .  $\delta$  is a transition function, then  $q_0$  which is the start state, then  $q_{acc}$  which is a single accepting state. Again, we do not have multiple accepting states. When we have to accept, we just move to an accepting state. Then  $q_{rej}$  which is a single reject state.  $Q, \Sigma, \Gamma$  are finite sets.  $Q$  is the set of states,  $\Sigma$  is the input alphabet. Input alphabet means, the strings that are computed or processed in the Turing machine, what are they comprised of?  $\Sigma$ . And  $\Gamma$  is the tape alphabet.

So, in the tape, we may use more symbols. So, everything that is in the input alphabet has to be in the tape alphabet. But there could be other symbols in the tape alphabet.  $Q, \Sigma, \Gamma$  are finite sets such that  $\Sigma \subseteq \Gamma$ , because input alphabet, we should be able to write all the input on the tape. And this blank symbol is part of the tape alphabet. So, this blank symbol which is used to denote that a tape location is empty and the blank symbol is not part of the input alphabet. We cannot have a blank as an input symbol. Then  $q_{accept}$  and  $q_{reject}$  have to be distinct.

$$\begin{aligned} \Sigma &\subseteq \Gamma, \quad \sqcup \in \Gamma, \quad \sqcup \notin \Sigma \\ q_{accept} &\neq q_{reject} \\ \delta(Q, \Gamma) &\rightarrow q \times \Gamma \times \{L, R\} \end{aligned}$$

Because both of them are different. And this is how the transition is defined. So again, it is a deterministic Turing machine. So, it is a function. If you are at a certain state  $q$ , and you read something from the tape in  $\Gamma$ , then we go to another state, you write something onto the tape, something like PDA because in PDA also you can read the input and the stack but here there are not two inputs with a stack. There is only one input, there is no stack. But you could also write onto the stack.

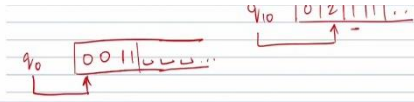
So, like that you could also write onto the tape. And then L, R which denotes left or right. So, this indicates how the head should move. The head could move left or right. So, suppose you were in state  $q_4$  and let us say this is the input tape. The input tape is let us say 0 2 3 1. And its head is currently pointing at 3. Suppose we have the rule  $\delta(q_4, 3) = (q_6, 0, R)$ . In that case what happens is we will use a different color. In that case, what happens is 0 remains as it is, 2 remains as it is. So, the 3 is replaced by 0. So, it reads a 3 and replaces it by 0 and then the next cell is 1 that is all the same. And if the tape head moves to the right.

So, which means the tape head is to the right and the state is  $q_6$ . So, if the rule  $\delta(q_4, 3)$  was to write 0 move right and then go to state  $q_6$  then this would be the next situation. If instead the rule was this,  $(q_{10}, 1, L)$  which means to go to state  $q_{10}$ , write 1 and move left then it would have been 0 2 1 1 and  $q_{10}$  and you move left.

So, you were pointing to the third location at the number 3 now, you move left and you write 1 here which is written here this 1 is new, and then you move to the left and you are at state  $q_{10}$ . So, suppose the transition rule was  $\delta(q_4, 3) = (q_{10}, 1, L)$  then this is how the Turing Machine tape will move. So, this is how we define the transitions.

And how does Turing Machine... so again, as I said, whatever we mentioned here, strike off the first position and then go to the hash then go to the right side and see if they are the same etcetera. All these things, all these things can be accomplished by states and transitions, but we will usually not be getting into that kind of detail, because it is getting too cumbersome and difficult to manage.

(Refer Slide Time: 22:10)



The diagram shows a Turing Machine tape with a state  $q_0$  above the first cell and  $q_{10}$  above the third cell. The tape contains the sequence 0 0 1 1 followed by blank symbols. A transition rule  $\delta(q_4, 3) = (q_{10}, 1, L)$  is indicated with arrows pointing to the state  $q_{10}$ , the symbol 1, and the direction L.

- \* The input is placed on the tape to the left most end...  $w = w_1 w_2 \dots w_n \in \Sigma^*$ . The rest of the cells are blank ( $\sqcup$ ). The blanks indicate the end of the input.
- \*  $\delta$  indicates what to write and where to move to.
- \* Computation starts from the left most end of the tape. If the tape head ever tries to move to the left from the left most end, it remains there.
- \* Computations end when  $q_{accept}$  /  $q_{reject}$  is reached. Else it can also go on forever without halting.



$\delta = \{ \delta_1, \delta_2, \dots, \delta_n \}$   
 $\delta: Q_{\text{accept}} \neq Q_{\text{reject}}$   
 $\delta: (Q, \Gamma) \rightarrow Q \times \Gamma \times \{L, R\}$

$\delta(q_4, 3) = (q_6, 0, R)$   $q_6$  | 0 | 2 | 0 | 1 | ...  
 $\delta(q_4, 3) = (q_{10}, 1, L)$   $q_{10}$  | 0 | 2 | 1 | 1 | ...  
 $q_0$  | 0 | 0 | 1 | 1 | ...

\* The input is placed on the tape to the left most end.  $w = w_1 w_2 \dots w_n \in \Sigma^*$ . The rest of the cells are blank ( $\sqcup$ ). The blanks indicate the end of the input.

\* The input is placed on the tape to the left most end.  $w = w_1 w_2 \dots w_n \in \Sigma^*$ . The rest of the cells are blank ( $\sqcup$ ). The blanks indicate the end of the input.

\*  $\delta$  indicates what to write and where to move to.

\* Computation starts from the left most end of the tape. If the tape head ever tries to move to the left from the left most end, it remains there.

\* Computations end when accept/reject is reached. Else it can also go on forever without halting. (looping).

Configuration: A configuration of a TM consists of



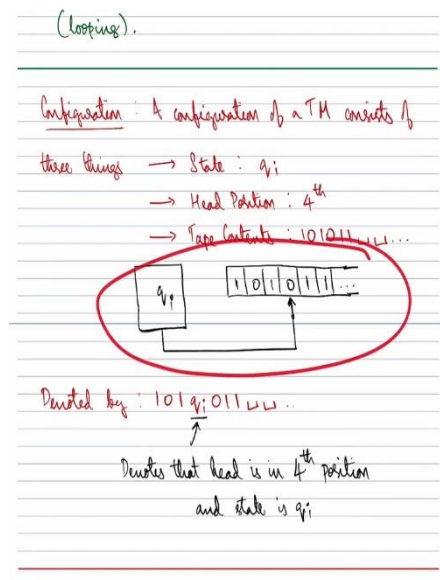
So, some more things. So, at the beginning of computation, let us say you want to give something as an input to the Turing machine, just like you give an input to the DFA or an NFA. So, then you keep the input in the left side of the Turing machine. So, suppose the input is 0011. So, then, you write 0011 on the tape, followed by blanks.

So, the rest of the tape is empty. And the head points to the leftmost location and you are at the starting state, which is  $q_0$ . And then it may read, it may write, it may move to the left or right or whatever till it goes to an accept or a reject, based on the transition rules. So,  $\delta$  does all this. And as I said, computation starts from the leftmost end of the tape. And there is one small point. So, here, we explained if it tries to move right from this point 0, you go here. If it goes left we go here, but if we try to go left from the leftmost location, we just remain there because there is nothing more to the left side.

And what like I said before the computations end, when you reach the accept or reject state, but it is also possible that neither state is reached neither accept nor reject, this is called an infinite loop situation. So, just like in a computer program, you could have an infinite loop, which you can never come out of. Like that, even in a Turing machine, you may end up in a situation where you just are doing something again and again or something repeatedly without ever going to accept or reject.

So, there are 3 possible ways a Turing Machine computation may proceed. One is that it accepts and ends. Another one is that it rejects and ends. Third one is that it neither accepts nor rejects it could just continue computation forever. So, this is called looping, like an infinite loop and accept and reject are called halt, because once you reach accept or reject you stop the computation and you end.

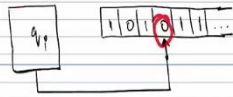
(Refer Slide Time: 24:41)



(looping).

Configuration : A configuration of a TM consists of

- three things → State :  $q_i$
- Head Position : 4<sup>th</sup>
- Tape Contents : 101011□□...



Denoted by : 101 $q_i$ 011□□...

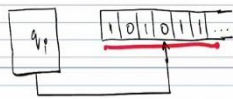
↑  
Denotes that head is in 4<sup>th</sup> position  
and state is  $q_i$



(looping).

Configuration : A configuration of a TM consists of

- three things → State :  $q_i$
- Head Position : 4<sup>th</sup>
- Tape Contents : 101011□□...



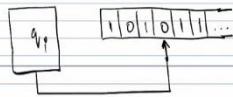
Denoted by : 101 $q_i$ 011□□...

↑  
Denotes that head is in 4<sup>th</sup> position  
and state is  $q_i$



Configuration : A configuration of a TM consists of

- three things → State :  $q_i$
- Head Position : 4<sup>th</sup>
- Tape Contents : 101011□□...



Denoted by : 101 $q_i$ 011□□...

↑  
Denotes that head is in 4<sup>th</sup> position  
and state is  $q_i$



We say  $C_1$  yields  $C_2$  if the TM can go from configuration  $C_1$  to configuration  $C_2$  in one single

So, the next thing is a definition called configuration. So, suppose there is a Turing machine and suppose it has some rules. I am computing and I reached a certain point of computation. Now I just want to tell you where I have reached so that you can resume the computation. So, what are the information that I have to give you so that you can continue the computation.

So, I cannot tell you the input, because I do not want you to start from the beginning. But now the tape contents may have changed. I may have overwritten on the input. I may have reached some other symbols. I may have like written over things multiple times. So, it would be simplest if I transfer the contents of the tape to you. Not just that, I should also tell you which state I am in and which position of the tape the head is pointing to.

So, if I can tell you these things, then you can continue from wherever I have stopped. So, the configuration contains exactly this information. It contains the state, the head position and the tape contents. So, once I tell you all these 3, you can continue the computation from this point. So, in this picture over here, the state is  $q_i$  head position is 4th, it is pointing to the 4th symbol and the tape contents is 101011 followed by blanks. Once I tell you this, you can start computation from this particular point and then see where it goes. It does not matter how you reached how I reached here.

So, these 3 pieces of information together are called configuration- state, head position, and tape contents. Maybe I will just put a box because these 3 things are called configuration and it is basically the information that I need to give you so that you can resume the computation. Usually, we denote configuration in this kind of notation. Basically, we write the state in between the input string. So, we know 101011 is the input. So, I write the state in between.

So,  $q_i$  is the state I write it in the 4th position, this denotes that  $q_i$  is the state and the head is pointing to the 4th location. So, this is one way to denote the configuration instead of telling state  $q_i$ , head position fourth and tape content 101011, I can just give this string in which the state is inserted into the fourth position. So, then you know the state you know the head position and also you the rest of it gives you the tape content.

(Refer Slide Time: 27:58)

and state is  $q_i$

---

We say  $C_1$  yields  $C_2$  if the TM can go from configuration  $C_1$  to configuration  $C_2$  in one single step.

Let  $u a q_i b v$  and  $u q_j a c v$  be two different configurations, where  $u, v \in \Gamma^*$ ,  $a, b, c \in \Gamma$  and  $q_i, q_j \in Q$ .

The diagram shows two configurations,  $C_1$  and  $C_2$ , represented as horizontal boxes.  $C_1$  contains the string  $u a b v$  and has state  $q_i$  above the  $a$ .  $C_2$  contains the string  $u a c v$  and has state  $q_j$  above the  $a$ . A red arrow points from  $q_i$  to  $q_j$ , and a blue arrow points from  $b$  in  $C_1$  to  $c$  in  $C_2$ .



and state is  $q_i$

---

We say  $C_1$  yields  $C_2$  if the TM can go from configuration  $C_1$  to configuration  $C_2$  in one single step.

Let  $u a q_i b v$  and  $u q_j a c v$  be two different configurations, where  $u, v \in \Gamma^*$ ,  $a, b, c \in \Gamma$  and  $q_i, q_j \in Q$ .

The diagram shows two configurations,  $C_1$  and  $C_2$ , represented as horizontal boxes.  $C_1$  contains the string  $u a b v$  and has state  $q_i$  above the  $a$ .  $C_2$  contains the string  $u a c v$  and has state  $q_j$  above the  $a$ . A red arrow points from  $q_i$  to  $q_j$ , and a blue arrow points from  $b$  in  $C_1$  to  $c$  in  $C_2$ .



Let  $u a q_i b v$  and  $u q_j a c v$  be two different configurations, where  $u, v \in \Gamma^*$ ,  $a, b, c \in \Gamma$  and  $q_i, q_j \in Q$ .

The diagram shows two configurations,  $C_1$  and  $C_2$ , represented as horizontal boxes.  $C_1$  contains the string  $u a b v$  and has state  $q_i$  above the  $a$ .  $C_2$  contains the string  $u a c v$  and has state  $q_j$  above the  $a$ . A red arrow points from  $q_i$  to  $q_j$ , and a blue arrow points from  $b$  in  $C_1$  to  $c$  in  $C_2$ .

So we say  $u a q_i b v$  yields  $u q_j a c v$ .

This happens when  $\delta(q_i, b) = (q_j, c, L)$

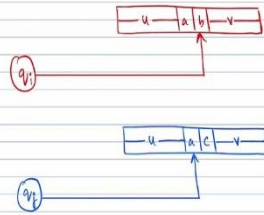






exp.

Let  $u a q_i b v$  and  $u q_j a c v$  be two different configurations, where  $u, v \in \Gamma^*$ ,  $a, b, c \in \Gamma$  and  $q_i, q_j \in Q$ .



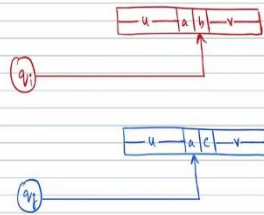
So we say  $u a q_i b v$  yields  $u q_j a c v$ .

This happens when  $\delta(q_i, b) = (q_j, c, L)$



exp.

Let  $u a q_i b v$  and  $u q_j a c v$  be two different configurations, where  $u, v \in \Gamma^*$ ,  $a, b, c \in \Gamma$  and  $q_i, q_j \in Q$ .



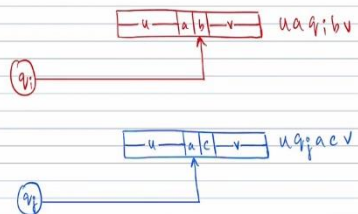
So we say  $u a q_i b v$  yields  $u q_j a c v$ .

This happens when  $\delta(q_i, b) = (q_j, c, L)$



exp.

Let  $u a q_i b v$  and  $u q_j a c v$  be two different configurations, where  $u, v \in \Gamma^*$ ,  $a, b, c \in \Gamma$  and  $q_i, q_j \in Q$ .



So we say  $u a q_i b v$  yields  $u q_j a c v$ .

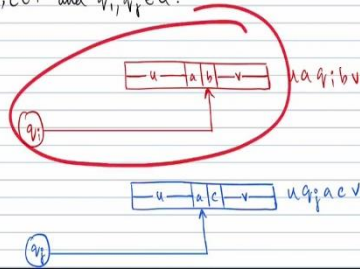
This happens when  $\delta(q_i, b) = (q_j, c, L)$



and state is  $q_i$

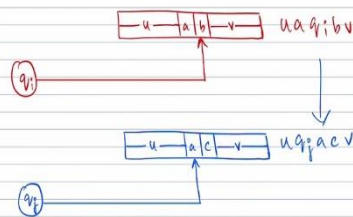
We say  $C_1$  yields  $C_2$  if the TM can go from configuration  $C_1$  to configuration  $C_2$  in one single step.

Let  $u a q_i b v$  and  $u q_j a c v$  be two different configurations, where  $u, v \in \Gamma^*$ ,  $a, b, c \in \Gamma$  and  $q_i, q_j \in Q$ .

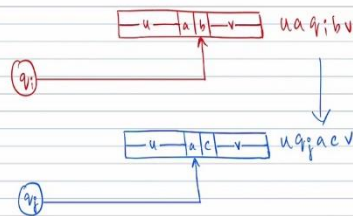


We say  $C_1$  yields  $C_2$  if the TM can go from configuration  $C_1$  to configuration  $C_2$  in one single step.

Let  $u a q_i b v$  and  $u q_j a c v$  be two different configurations, where  $u, v \in \Gamma^*$ ,  $a, b, c \in \Gamma$  and  $q_i, q_j \in Q$ .



Let  $u a q_i b v$  and  $u q_j a c v$  be two different configurations, where  $u, v \in \Gamma^*$ ,  $a, b, c \in \Gamma$  and  $q_i, q_j \in Q$ .



So we say  $u a q_i b v$  yields  $u q_j a c v$ .

This happens when  $\delta(q_i, b) = (q_j, c, L)$

If instead we had  $\delta(q_i, b) = (q_j, c, R)$



So, we say that a configuration C1 yields a configuration C2 if from the configuration C1 you apply the transition rule and go to C2. So, suppose, you have this configuration,  $u a q_i b v$ , which means, you have  $u$ , you have  $a$ , you have  $b$  you have  $v$  and  $q_i$  is the state and you are pointing it at the location of  $b$ .

Now, suppose the rule is that, if you are in state  $q_i$  and you are reading  $b$  from the tape, then you have to write  $c$ , you have to go to state  $q_j$ , and move left. Suppose this was a rule, if you were in  $q_i$  and you are reading  $b$  you have to move to the left go to state  $q_j$  and write  $c$ . Suppose this was the transition function this configuration, we will move to this configuration.

So, where  $b$  is overwritten by  $c$  and you move left and the state is  $q_j$  instead of  $q_i$ . So,  $b$  is changed into  $c$  and the tape head has moved one step to the left. So, we can say that  $u a q_i b v$  yields  $u q_j a c v$ . So, notice that the state in the configuration came one step to the left because the head has moved one step. So here  $u$  and  $v$  are some strings and  $a b c$  are single symbols of the tape alphabet.

Now one question. Suppose the rule was, instead of  $\delta(q_i, b) = (q_j, c, L)$ . If it gave  $(q_j, c, R)$ , then everything would have been the same. But this is what I have written here. Instead of  $\delta(q_i, b)$  being  $(q_j, c, L)$  if it was  $(q_j, c, R)$  so, this  $L$  became  $R$ , everything would be the same it would override it with  $c$ , it would enter a state  $q_j$ , but instead of pointing at  $a$  it would point at the first symbol of this place that I have depicted here. If the rule was of  $\delta(q_i, b) = (q_j, c, R)$  then this would be the configuration  $u a c q_j v$ .

So, we say that the configuration C1 yields C2 if C2 is a successor configuration of the configuration C1. And notice that we are in deterministic Turing machines. There is no non-determinism. So, given this particular situation, when I said this particular situation, the next step is clearly defined. So, we know that  $u a q_i b v$  we know that this yields  $u q_j a c v$ . And that is the unique configuration that it can yield. In a deterministic Turing machine there is only one clear configuration that it yields, so we say this.

(Refer Slide Time: 31:57)

$u a c q_1 v$

---

The TM accepts  $w$  if there is a sequence of configurations  $C_1, C_2, \dots, C_k$  such that

1.  $C_1$  is the start configuration:  $q_0 w$
2.  $C_i \rightarrow C_{i+1}$  for all  $i$
3.  $C_k$  is an accepting configuration (the state in  $C_k$  is  $q_{\text{accept}}$ ).

The language recognized by  $M$  is denoted  $L(M)$ .

$$L(M) = \{w \mid w \text{ is accepted by } M\}$$


\*  $\delta(Q, \Gamma) \rightarrow Q \times \Gamma \times \{L, R\}$

$q_4$  0 | 2 | 3 | 1 | ...  
 $\uparrow$

$\delta(q_4, 3) = (q_6, 0, R)$   $q_6$  0 | 2 | 0 | 1 | ...  
 $\delta(q_4, 3) = (q_{10}, 1, L)$   $q_{10}$  0 | 2 | 1 | 1 | ...

$q_0$  0 | 0 | 1 | 1 | 0 | 0 | ...  
 $\uparrow$

\* The input is placed on the tape to the left most end.  $w = w_1 w_2 \dots w_n \in \Sigma^*$ . The rest of the cells are blank ( $\sqcup$ ). The blanks indicate the end of the input.

\*  $\delta$  indicates what to write and where to move to.



u | a | c | v  
 $q_1$   $\uparrow$   
 $u a c q_1 v$



The TM accepts  $w$  if there is a sequence of configurations  $C_1, C_2, \dots, C_k$  such that

1.  $C_1$  is the start configuration:  $q_0 w$
2.  $C_i \rightarrow C_{i+1}$  for all  $i$
3.  $C_k$  is an accepting configuration (the state in  $C_k$  is  $q_{\text{accept}}$ ).





The TM accepts  $w$  if there is a sequence of configurations  $C_1, C_2, \dots, C_k$  such that

1.  $C_1$  is the start configuration:  $q_0 w$
2.  $C_i \rightarrow C_{i+1}$  for all  $i$
3.  $C_k$  is an accepting configuration (the state in  $C_k$  is accept).

The language recognized by  $M$  is denoted  $L(M)$ .



The TM accepts  $w$  if there is a sequence of configurations  $C_1, C_2, \dots, C_k$  such that

1.  $C_1$  is the start configuration:  $q_0 w$
2.  $C_i \rightarrow C_{i+1}$  for all  $i$
3.  $C_k$  is an accepting configuration (the state in  $C_k$  is accept).

The language recognized by  $M$  is denoted  $L(M)$ .

$$L(M) = \{ w \mid w \text{ is accepted by } M \}.$$



The TM accepts  $w$  if there is a sequence of configurations  $C_1, C_2, \dots, C_k$  such that

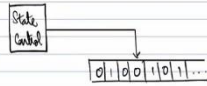
1.  $C_1$  is the start configuration:  $q_0 w$
2.  $C_i \rightarrow C_{i+1}$  for all  $i$
3.  $C_k$  is an accepting configuration (the state in  $C_k$  is accept).

The language recognized by  $M$  is denoted  $L(M)$ .

$$L(M) = \{ w \mid w \text{ is accepted by } M \}.$$



- Read/write head
- Head can move left or right.
- Infinite tape (to one side)
- Special accept/reject states which immediately come into effect.



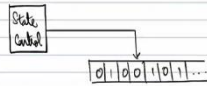
Consider the language  $D$ .

$$D = \{w\#w \mid w \in \{0,1\}^*\}$$

$D$  is not a CFL. Can a Turing machine recognize  $D$ ?



- Special accept/reject states which immediately come into effect.



Consider the language  $D$ .

$$D = \{w\#w \mid w \in \{0,1\}^*\}$$

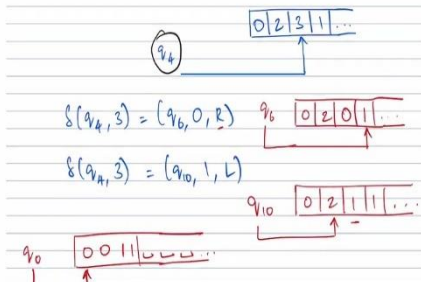
$D$  is not a CFL. Can a Turing machine recognize  $D$ ?



On tape:  $\phi \ x \ y \ \phi \ \phi \ # \ \phi \ x \ x \ \phi \ \phi \ \dots$   
 Reject  $\begin{cases} \phi \ x \ x \ \phi \ # \ \phi \ x \ x \ \phi \ \phi & \text{blank} \\ \phi \ x \ y \ \phi \ # \ \phi \ x \ x \ \phi & \text{reject} \end{cases}$

Formal Definition of TM (Def 3.3): A Turing machine is a 7-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$  where  $Q, \Sigma, \Gamma$  are finite sets, such that

- $\Sigma \subseteq \Gamma, \perp \in \Gamma, \perp \notin \Sigma$ .
- $q_{\text{accept}} \neq q_{\text{reject}}$
- $\delta: (Q, \Gamma) \rightarrow Q \times \Gamma \times \{L, R\}$



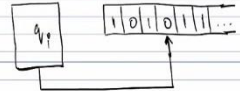
\* Computations end when accept/reject is reached.  
Else it can also go on forever without halting.  
(looping).



Configuration: A configuration of a TM consists of

three things

- State :  $q_i$
- Head Position : 4<sup>th</sup>
- Tape Contents : 101011□□...



Denoted by :  $101q_i011□□...$

↑  
Denotes that head is in 4<sup>th</sup> position

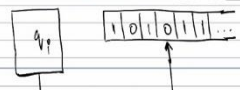
(looping).



Configuration: A configuration of a TM consists of

three things

- State :  $q_i$
- Head Position : 4<sup>th</sup>
- Tape Contents : 101011□□...



Denoted by :  $101q_i011□□...$

↑  
Denotes that head is in 4<sup>th</sup> position  
and state is  $q_i$



Now finally, maybe I will just conclude by saying when does a Turing machine accept a string. If we say the Turing machine accepts a string if there are a series of configurations such that... this sort of thing we have seen in DFAs, NFAs, PDAs, etcetera. So, the starting should be proper ending should be proper and all the steps in between should be proper. We say that Turing machine accepts  $w$  if  $C_1$  is a starting configuration. Starting configuration means the entire input string should be in the input tape and we should be at the starting state.

So, the input starting configuration is  $q_0 w$ , I think we mentioned this earlier. Here we had mentioned it, if the input was 0011 and  $q_0$  is the starting state, the head should point to the leftmost and the input should be the only thing in the tape. So,  $C_1$  should be the start configuration and for all these pairs  $C_1 C_2, C_2 C_3$  etcetera. We want  $C_1$  to yield  $C_2, C_2$  to yield  $C_3$  and so on. So, this should be a valid sequence of configuration so, that they are all

valid successes of each other.  $C_i \rightarrow C_{i+1}$  for all  $i$  and finally,  $C_k$  should be an accepting configuration. What do I mean by accepting configuration? It is a configuration which has an accepting state.

So, every configuration contains a state. So, the state in the configuration  $C_k$  should be the accepting state. So, once we have these 3 you start correctly all your moves are proper and end with an accepting state then the string is accepted and the language recognized by the Turing Machine  $M$  like before it be denoted  $L(M)$ , it is a set of all strings which are accepted by the Turing machine.

So, once again, a Turing machine accepts a string if it starts with the starting configuration, which has the starting state in the state and the input string in the input tape and that is the start configuration. And for each of the,  $C_2, C_3$  etcetera are valid successors of the previous one. And finally,  $C_k$  is an accepting configuration. And the language recognized by the Turing machine is a set of all strings that are accepted by the Turing Machine. So that is it as far as this lecture is concerned. So, we defined what is Turing machine. And in fact, what we saw is a deterministic Turing machine.

So, later we will see many, many variants. So, we will see non deterministic Turing Machine we will see Turing machine with multiple tapes, etcetera. But right now, what we saw was a deterministic Turing machine. And we saw the distinctions with respect to PDAs and NFAs etcetera. It has read write head, which can move left or right, we have infinite tape and special accept, reject states. We saw what is the configuration, we saw a formal definition. We saw what is the configuration, it contains 3 things- state, head position, and the tape contents.

We saw this notation to denote the configuration, which is the tape content where the state is inserted in the head position. And we saw the conditions for a string to be accepted in a Turing machine. So, this computation has to continue. Initially, you start with the input in the tape, and the head to the leftmost point, and the start state, and then you just let it run.

If it reaches an accept state it accepts, if it reaches a reject state, it rejects. And if it does not reach either, it does not accept or reject, but it just continues, it is a loop and the set of all strings that are accepted is called the language recognized by the Turing machine. And that completes this lecture, lecture 27. That also completes week 5 lectures. So, in week 5, we saw the equivalence of PDAs and context free grammars. We saw pumping lemma for context free languages. And we saw the beginning of introduction to computability theory, which is the



definition of Turing machine. And we will see more about computability theory and Turing machines in the upcoming week, which is week 6. So, see you in week 6. Thank you.