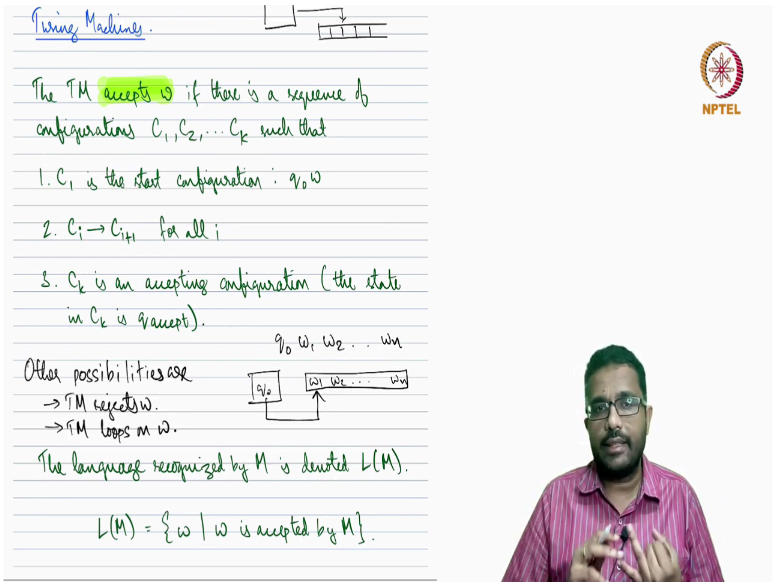
Theory of Computation
Professor Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture: 28
Turing Recognizable & Decidable Languages and TM Examples

(Refer Slide Time: 00:15)



Turing Machines.

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. $C_i \to C_{i+1}$ for all i
3. C_k is an accepting configuration (the state in C_k is q_{accept}).

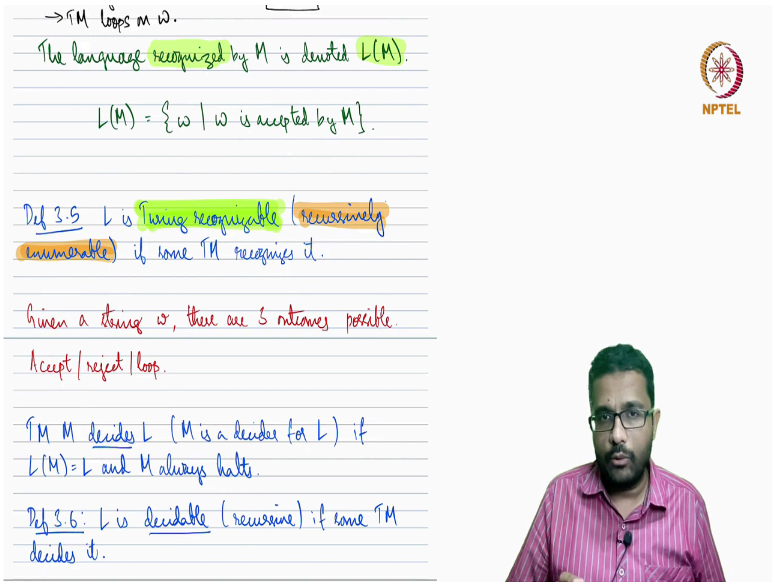
$q_0\, w_1\, w_2 \dots w_n$

Other possibilities are
→ TM rejects w.
→ TM loops on w.

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

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

q_0
$w_1\ w_2 \dots w_n$



→ TM loops on w.

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

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

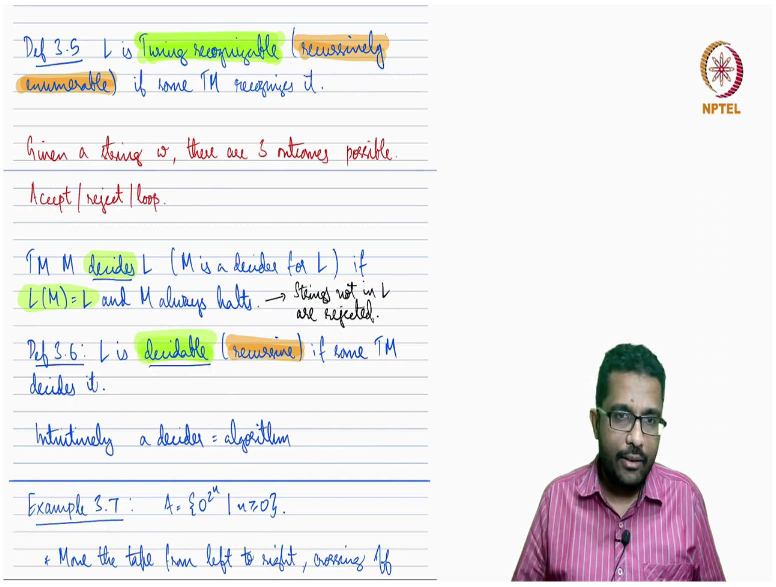
Def 3.5 L is Turing recognizable (recursively enumerable) if some TM recognizes it.

Given a string w, there are 3 outcomes possible.

Accept / reject / loop.

TM M decides L (M is a decider for L) if $L(M) = L$ and M always halts.

Def 3.6: L is decidable (recursive) if some TM decides it.



Def 3.5 L is Turing recognizable (recursively enumerable) if some TM recognizes it.

Given a string w, there are 3 outcomes possible.

Accept / reject / loop.

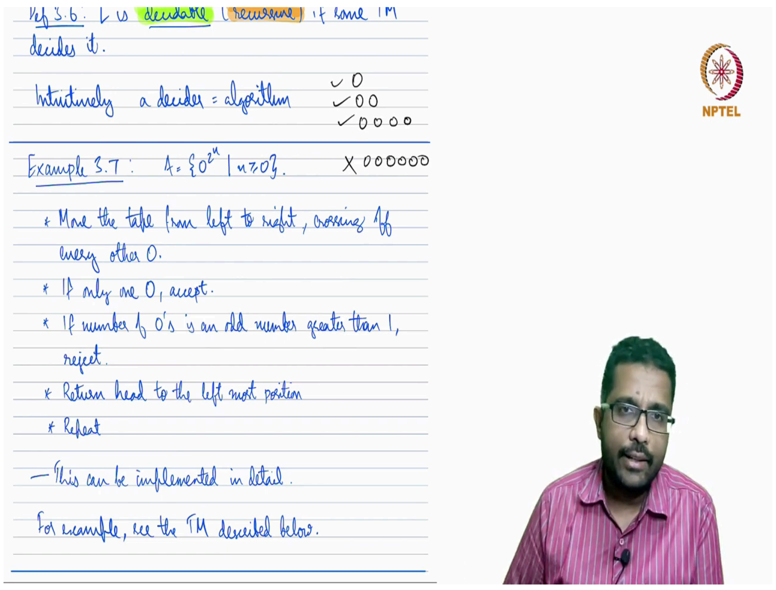
TM M decides L (M is a decider for L) if L(M) = L and M always halts. $\rightarrow$ Strings not in L are rejected.

Def 3.6: L is decidable (recursive) if some TM decides it.

Intuitively a decider = algorithm

Example 3.7: $A = \{0^{2^n} \mid n \geq 0\}$.

* Move the tape from left to right, crossing off



Def 3.6: L is decidable (recursive) if some TM decides it.

Intuitively a decider = algorithm

✓ 0

✓ 00

✓ 0000

Example 3.7: $A = \{0^{2^n} \mid n \geq 0\}$. X 000000

* Move the tape from left to right, crossing off every other 0.

* If only one 0, accept.

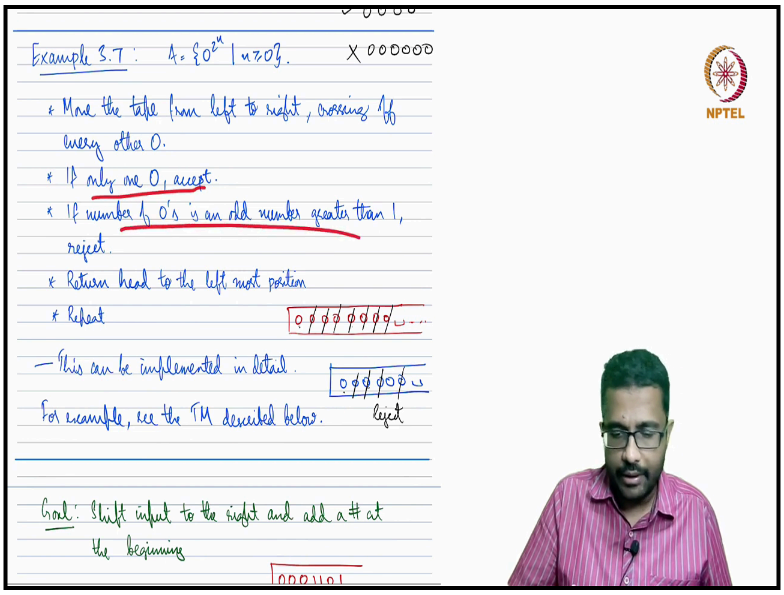
* If number of 0's is an odd number greater than 1, reject.

* Return head to the left most position

* Repeat

— This can be implemented in detail.

For example, see the TM described below.



Example 3.7: $A = \{0^{2^n} \mid n \geq 0\}$. X 000000

* Move the tape from left to right, crossing off every other 0.

* If only one 0, accept.

* If number of 0's is an odd number greater than 1, reject.

* Return head to the left most position

* Repeat

$\boxed{0\cancel{0}\cancel{0}\cancel{0}\cancel{0}\cancel{0}\cancel{0}\cancel{0}\sqcup\ldots}$

— This can be implemented in detail.

$\boxed{0\cancel{0}\cancel{0}\cancel{0}0\cancel{0}\sqcup}$

For example, see the TM described below. Reject

Goal: Shift input to the right and add a # at the beginning

$\boxed{000110\,1}$

Hello and welcome to Lecture 28 of the course theory of computation. So, also the first lecture of week 6, in week 5, we completed context free languages. And then we started talking about Turing machines. So, we defined what a Turing machine is and we explained what when does a Turing Machine accept a certain string.

So, let me just repeat so, the Turing machine has a state control and there is a tape which is an infinite tape and which is initially filled with the input. And the main differences of a Turing machine is that differences from a DFA NFA or a PDA is that there are special accept and reject states there is one accept one reject state and which so once you reach the accept or reject state, you immediately accept or reject, then the input tape is a read write tape, it is an infinite tape.

And that is these are the main differences between Turing machines and the previous automata that we saw. So, we say that a Turing machine accepts a string if there is a sequence of configurations C1, C2 up to Ck such that C1 is a start configuration. So, start configuration means, you have the start state start state being q0 followed by the input w. So, w itself is a string. So, this means that the input will be less something like this.

So, q0, w1, w2 et cetera wn, what does this mean? This is the configuration, what this means is that in the input tape you have w1, w2 et cetera wn and the head is pointing to the first location and the state is q0 this is the start configuration and then every successive like C2 is a valid successor of C1, C3 is a valid successor of C2 and so on.

And finally Ck must be a valid successor of Ck minus 1 and Ck must be accepting configuration meaning the state corresponding to Ck must be the accept state. So, basically this is the situation when a Turing machine accepts a string. So, recall that when you are given a string a Turing machine can, there are three possible eventualities one is that the machine accepts the string, another one is it rejects the string. So, rejection will happen exactly the same way.

So, this is another place where Turing Machine differs from DFA, NFA and all is that rejection. So, in DFA, NFA, PDA rejection was just defined as not accepting here there are two possible ways of string not getting accepted, one is that exactly the same kind of thing can happen, but instead of Ck being an accepting configuration, it could be rejecting configuration meaning the state in the Ck could be q reject.

So, TM can reject so, the other possibilities are two possibilities TM rejects w or Turing Machine loops on w. So, loops by loops on w I mean something where the Turing machine is not able to accept it is not able to reject it just continues computation forever. So, just like if you write a computer program and if there is an infinite loop it will never end, but it will also never give an output it never accepts and never makes a decision.

This similar situation is what I mean by the Turing Machine loops on w. So, given a string to a Turing machine, there are three (possible) possibilities of what can happen: one is a string gets accepted, two strings gets rejected, three is that the string just continues computation and neither gets accepted nor gets rejected.

So, that is something that we need to understand and this is an important difference of how Turing machines are different from DFAs or NFAs. So, the next definition actually the next set of definitions actually, this plays a role in the fact that there are two ways in which a string cannot get accepted.

So, first of all, we define the language that is recognized by a Turing Machine. So, notice the word recognized is, so, Turing Machine M is denoted by L(M) just like we did for DFA, NFA et cetera, it is the set of all strings that are accepted by the Turing Machine. So, this is exactly similar like how we did for NFA, PDA and so on. L of M is the set of all strings that are accepted by the machine.

So, which means, there are three again I want to say that three possibilities, so, there are all the set of all strings in Σ^* some sets some strings are accepted and that is called L(M), the remaining strings may not be all rejected, there could be strings that are rejected and there could be some strings that are neither accepted nor rejected where the machine just loops on them.

So, L(M) is a set of accepted strings and what is important is from or what is different here from PDAs or NFAs or DFAs is that the remaining strings that are not accepted, Σ^* minus L(M) the remaining strings there are two possibilities it could be rejected or it could be looped. Anyway the language recognizable Turing machine is denoted L(M) and we say that a language is Turing recognizable or the other terminology used is recursively enumerable if there is some Turing machine that recognizes it.

So, just like regular languages are the set of languages that are recognized by some DFAs or context free languages are those languages that are recognized by some PDA. Similarly, Turing recognizable languages are those languages that can be recognized by some Turing Machine. So, which means L is a Turing recognizable if there is some Turing Machine M such that L is a language recognized by M and another word for Turing recognizable.

So, I will maybe use it is recursively enumerable. So, this is another terminology for Turing recognizable. So, in Sipser they predominantly use during recognizable, but if you look at some other books like Hop croft and so on, they use the word recursively enumerable. So, I will be following Sipser so, I will be mostly referring to it as Turing recognizable, but, I am making this remark so, that in case in the future, keep this in mind.

So, in case you are referring to some resource which is the other definition you should or the other terminology, you should be able to you should be familiar with it. So, L is Turing recognizable or L is recursively enumerable if some Turing Machine recognizes it. And this is something that I already said given a string, there are three outcomes possible: accept, reject or loop.

So, now, based on this there is another definition. So, we say that a Turing Machine M decides L. Or in other words we say that M is a decider for the language the Turing machine is a decider for the language, if the language recognized by the Turing machine is L so L(M) is equal to L means, the language recognized by the Turing machine is L meaning the set of all strings accepted by the Turing machine is L.

This is same as M is the language recognized by M as L and the second condition is that M always halts meaning looping is not an option. This means that M decides L if there is a Turing Machine sorry, M decides L. Sorry, M decides L if M accepts all the strings in L and M rejects all the strings not in L.

So, if you want M to recognize L, if you want M to merely recognize L, you just want the strings in L to be accepted and the strings not in L to be not accepted. So not accepted could be in two possibilities: reject or loop. However, when you are talking about a decider, you want all the strings in L to be accepted and all the strings not in L to be rejected meaning you are not allowed to loop on any input.

So M always halts means, strings not in L are rejected. No looping is allowed. So decides means you are not supposed to loop on any input. This is what you say. This is when you so, notice that we say M recognizes L and we say M decides L. So, deciding is actually so, when you M, when M decides L M is also a recognizer for L, but decider, if it has to be decider there are more constraints on it.

So, a decider is also a recognizer, but a recognizer need not be a decider because there could be strings, they are not in L on which M is looping. So, we say that L is decidable the language is decidable if there is some Turing Machine M that decides it. And just like I said recursively enumerable another terminology for decidable is recursive.

So, we say that, so, some notes may refer to it as recursive. So, L is decidable if some Turing Machine decides it. So, once again three possibilities exist: accept, reject and loop if a Turing machine the set of strings accepted by a Turing machine is L we say that the Turing Machine recognizes L, further if no string is rejected by the Turing machine we say that the Turing Machine decides that language.

So, I am sorry if L is a language recognized by the Turing machine we say the Turing Machine recognizes L, further if no string loops on the Turing Machine meaning all the strings that are not in L are rejected, we say that L, the Turing Machine decides L, so, we do not we want the Turing machine to halt on all the inputs no looping is allowed.

So, this is a difference between recognizing and deciding a Turing machine that decides a language also recognizes that language, but then a Turing machine recognizes a language need not be a decider because there could be some strings that loop on it. So, intuitively, since some of you may be familiar with the course algorithms, to decide it like an algorithm so, when you run an algorithm you always want an answer.

So, you want to know if a graph is connected or something is three-colorable. You want a yes or no answer. So, the answer could be. You want a yes or no answer. So, you do not want to give an input and just not hear back from the algorithm. So, a decider can be intuitively thought, thought of as an algorithm. So, this is a, these are the main definitions L being Turing recognizable, and L being decidable. So, now, let us see some examples of Turing machines. Or some examples in the way Turing machines can operate.

So the first example is a language. It is a unary language, 0 power 2 power n (0^{2^n}). So, 0 power 2 power n means it is a string of 2 power n 0s. So, it is a unary language that is so it is entirely 0s, but the number of 0s in the string is going to be a power of 2. So, the strings in the language are something like 0 is in the language 00 is in the language 0000 is in the language.

But so, all these are in the language but that is 000000 this is not in the language, because 6 zeros is 6 is not a power of 2. So how can a Turing Machine be programmed to decide this language? So, we will talk about how the Turing machine operates at a high level and then we will get into the details or will not get into the entire details but we will just say that everything that we say here can be implemented in detail.

So, what we do is let us say I will just draw a tape here. Something like 00000000 and that is it. This is my tape. So the 8 this 8 zeros this is in the language 8 is a power of 2. So, what we do is move from left to right. And crossing off every other 0, so you mean you start with 0 the next 0 is crossed off, next 0 then the next 0 is crossed off, next 0 next 0 is crossed off, next 0 next 0 is crossed off.

So, now you are, now you come to the blank. So now you see that you are at the right most end of the string. Now, what you do is, once you are at the end, you return to the left side. And if so at this stage, if there was an odd number of 0s, meaning if, if instead of the space, sorry, if instead of the space here we had a ninth 0, then we would have rejected so 9, because 9 is not a power of 2.

But anyway, now we have a space here. And we will continue. So now we come back to the left. So, now you start with the first 0, then one is the crossed of one, we can just skip over, so the next 0, you cross off, then the fourth the 0, which is in the originally in the fifth position, then the 7th, 0 is crossed off, then you see that you are again, coming to the end, then you so you come back to the right, left side.

So now you have only two 0s, the first one you see, and the second one you cross off. And, again, you come back, and now you see that there is only one 0. So because it is only one 0, you accept because it is the conditions set. If there is only one 0, you accept, and if at any point you see that the number of 0s is an odd number greater than 1 you reject. So in this case, we did not encounter that because 8 was a power of 2.

So 8 became 4, 4 became 2, 2 became 1. However, if you had started off with some other number, let us say you have, we had started off with 6 000000 what would have happened is first you cross off this you cross off this and you cross off this, then you come back to the left side then you see the first 0 then the next 0 is crossed off, then you see the 0 and the 5th spot and then there is no other 0 to cross off this is so, now we had three 0s that were remaining 3 is an odd number greater than 1.

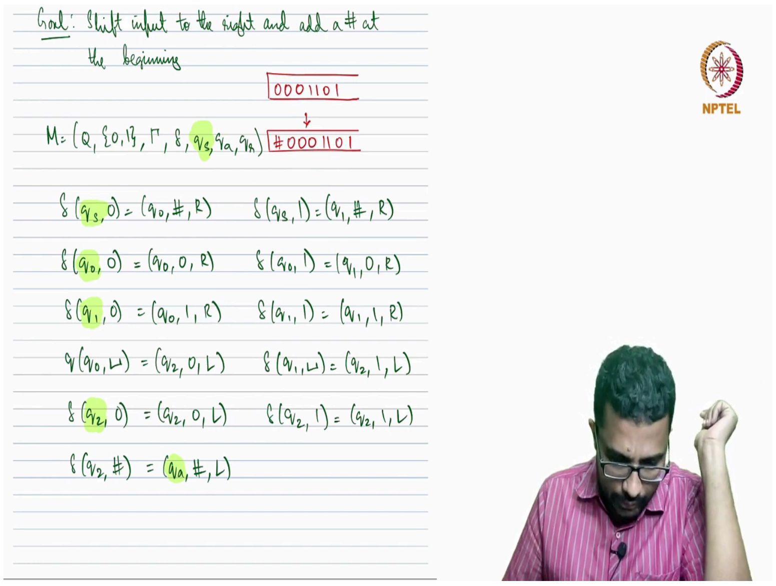
So which means at this point it is rejected. So, this is what you do, this is how you can check whether the number of zeros is a power of 2. And the algorithm is written in a bit of detail over here. But this is still a high level. Because I am saying things like move the tape from left to right.

But the definition of a Turing machine, if you remember, involves states and each tape head reading something and then each move going left or right. So, now I am talking about moving from left to right entirely. So how will that be programmed? So, the point I want to say is that all that can be done.

So, most of what we say, here all of this can be accomplished using states and appropriate rules. So, just to illustrate, but then if we go into that kind of detail, we will end up spending too much time and too much effort. So, what I would rather do is to explain one small task using such rules.

Instead of explaining everything to the detail, I will just take up one small task and explain how that can be accomplished using the Turing machine rules. And now, you will just have to take it from me that any of these rules like if the number of, if only one 0 you accept number of 0s an odd number you reject all these kinds of rules can be accomplished return head to the leftmost position, all of this can be accomplished.

(Refer Slide Time: 18:26)



Goal: Shift input to the right and add a # at the beginning

$\boxed{0001101}$

$\downarrow$

$M = (Q, \{0,1\}, \Gamma, \delta, q_s, q_a, q_r)$ $\boxed{\#0001101}$

$\delta(q_s, 0) = (q_0, \#, R)$ $\delta(q_s, 1) = (q_1, \#, R)$

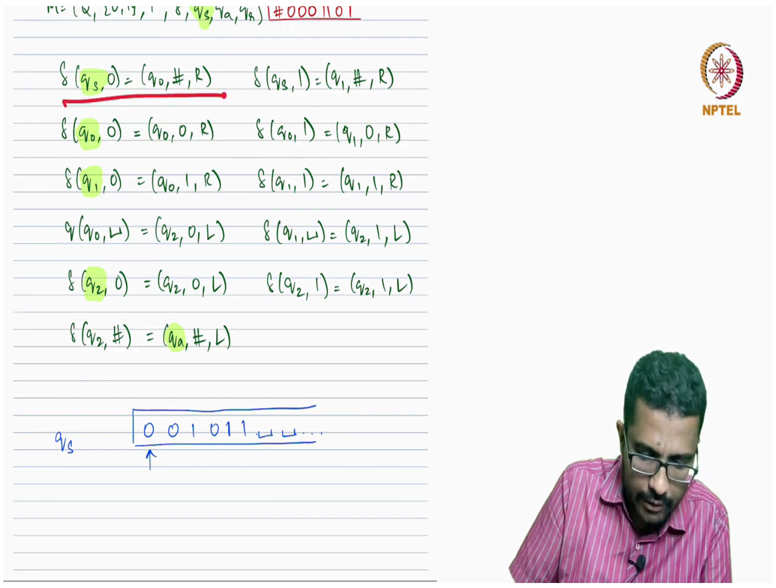
$\delta(q_0, 0) = (q_0, 0, R)$ $\delta(q_0, 1) = (q_1, 0, R)$

$\delta(q_1, 0) = (q_0, 1, R)$ $\delta(q_1, 1) = (q_1, 1, R)$

$\delta(q_0, \sqcup) = (q_2, 0, L)$ $\delta(q_1, \sqcup) = (q_2, 1, L)$

$\delta(q_2, 0) = (q_2, 0, L)$ $\delta(q_2, 1) = (q_2, 1, L)$

$\delta(q_2, \#) = (q_a, \#, L)$



$M = (Q, \{0,1\}, \Gamma, \delta, q_s, q_a, q_r)$ $\boxed{\#0001101}$

$\underline{\delta(q_s, 0) = (q_0, \#, R)}$ $\delta(q_s, 1) = (q_1, \#, R)$

$\delta(q_0, 0) = (q_0, 0, R)$ $\delta(q_0, 1) = (q_1, 0, R)$

$\delta(q_1, 0) = (q_0, 1, R)$ $\delta(q_1, 1) = (q_1, 1, R)$

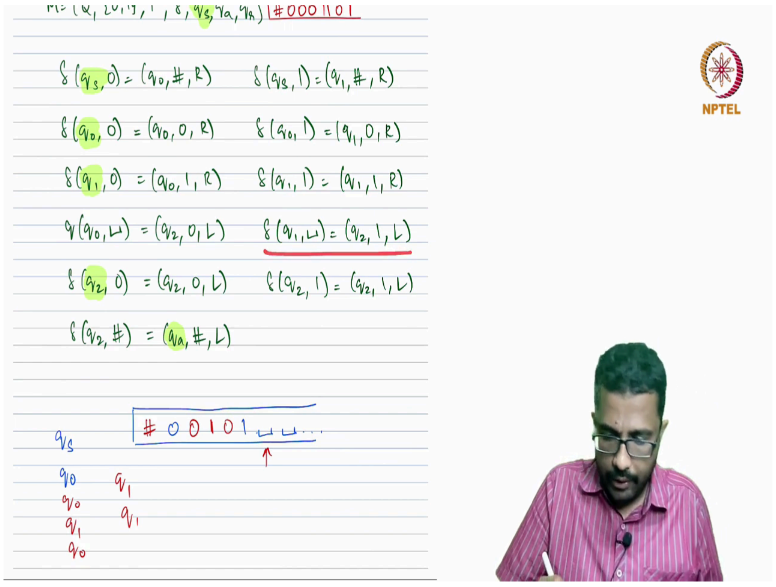
$\delta(q_0, \sqcup) = (q_2, 0, L)$ $\delta(q_1, \sqcup) = (q_2, 1, L)$

$\delta(q_2, 0) = (q_2, 0, L)$ $\delta(q_2, 1) = (q_2, 1, L)$

$\delta(q_2, \#) = (q_a, \#, L)$

q_s $\boxed{0\ 0\ 1\ 0\ 1\ 1\ \sqcup\ \sqcup \ldots}$

$\uparrow$



$M = (Q, \{0,1\}, \Gamma, \delta, q_s, q_a, q_R)$ | #0001101

$\delta(q_s, 0) = (q_0, \#, R)$ $\delta(q_s, 1) = (q_1, \#, R)$

$\delta(q_0, 0) = (q_0, 0, R)$ $\delta(q_0, 1) = (q_1, 0, R)$

$\delta(q_1, 0) = (q_0, 1, R)$ $\delta(q_1, 1) = (q_1, 1, R)$

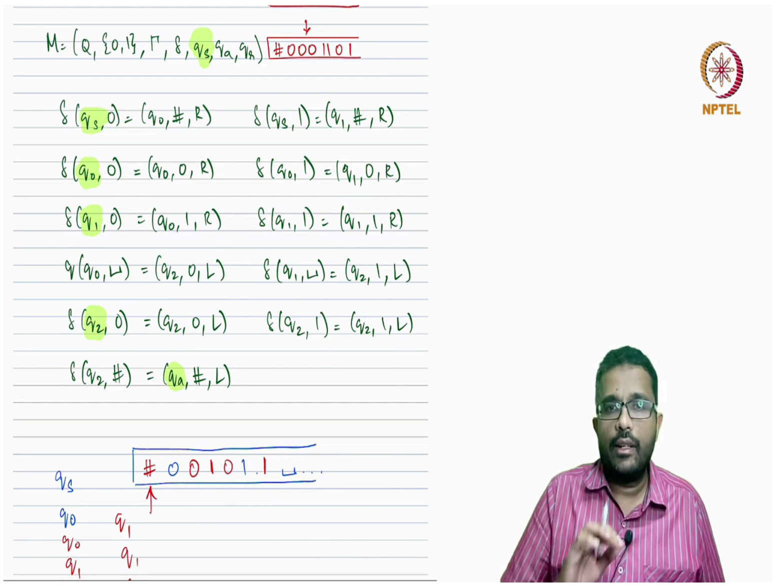
$\delta(q_0, \sqcup) = (q_2, 0, L)$ $\underline{\delta(q_1, \sqcup) = (q_2, 1, L)}$

$\delta(q_2, 0) = (q_2, 0, L)$ $\delta(q_2, 1) = (q_2, 1, L)$

$\delta(q_2, \#) = (q_a, \#, L)$

q_s | # 0 0 1 0 1 ⊔ ⊔ ... |

q_0 q_1
q_0 q_1
q_1
q_0



$M = (Q, \{0,1\}, \Gamma, \delta, q_s, q_a, q_R)$ | #0001101

$\delta(q_s, 0) = (q_0, \#, R)$ $\delta(q_s, 1) = (q_1, \#, R)$

$\delta(q_0, 0) = (q_0, 0, R)$ $\delta(q_0, 1) = (q_1, 0, R)$

$\delta(q_1, 0) = (q_0, 1, R)$ $\delta(q_1, 1) = (q_1, 1, R)$

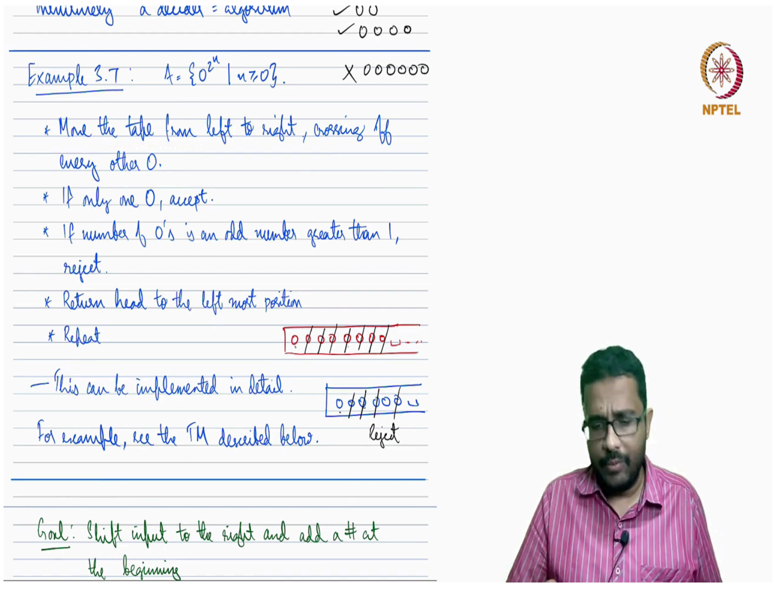
$\delta(q_0, \sqcup) = (q_2, 0, L)$ $\delta(q_1, \sqcup) = (q_2, 1, L)$

$\delta(q_2, 0) = (q_2, 0, L)$ $\delta(q_2, 1) = (q_2, 1, L)$

$\delta(q_2, \#) = (q_a, \#, L)$

q_s | # 0 0 1 0 1 1 ⊔ ... |

q_0 q_1
q_0 q_1
q_1



minimizing a detailed algorithm ✓00
✓0000

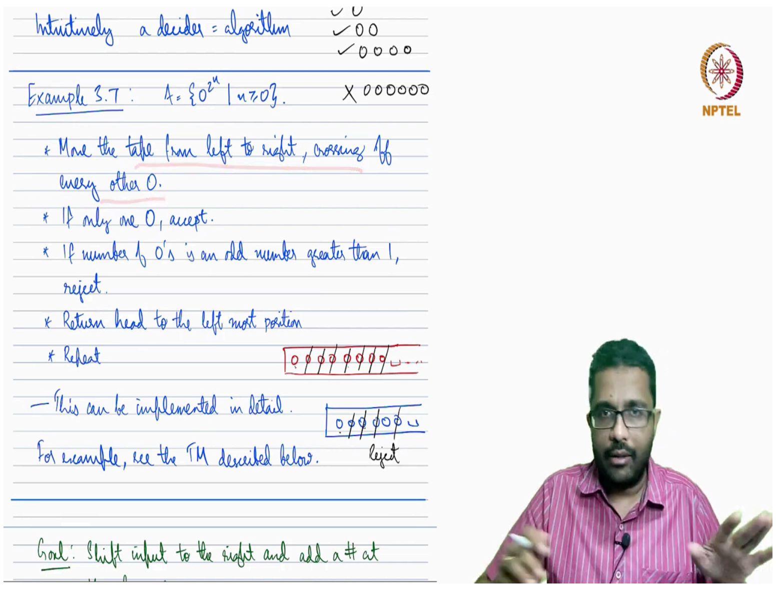
Example 3.7: $A = \{0^{2^n} \mid n \geq 0\}$. X 000000

* Move the tape from left to right, crossing off every other 0.
* If only one 0, accept.
* If number of 0's is an odd number greater than 1, reject.
* Return head to the left most position
* Repeat | $0 \cancel{0} \cancel{0} \cancel{0} \cancel{0} \cancel{0} \cancel{0} \cancel{0} \sqcup \ldots$ |

— This can be implemented in detail. | $0 \cancel{0} \cancel{0} \cancel{0} 0 \cancel{0} \sqcup$ |

For example, see the TM described below. reject

Goal: Shift input to the right and add a # at the beginning



Intuitively a decider = algorithm

✓ 0
✓ 00
✓ 0000

Example 3.7 : $A = \{0^{2^n} \mid n \geq 0\}$.

X 000000

* Move the tape from left to right, crossing off every other 0.
* If only one 0, accept.
* If number of 0's is an odd number greater than 1, reject.
* Return head to the left most position
* Repeat

$0\cancel{0}0\cancel{0}0\cancel{0}0\cancel{0}\sqcup\ldots$

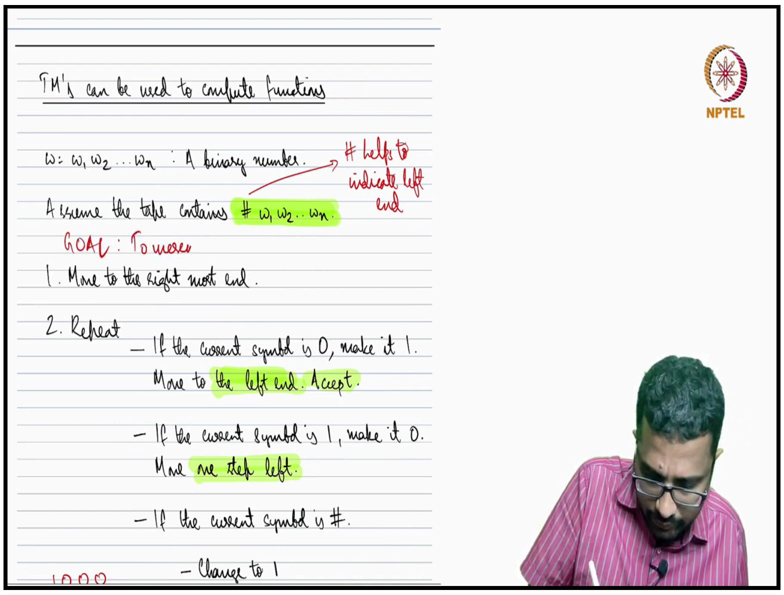
— This can be implemented in detail.

$0\cancel{0}\cancel{0}\cancel{0}0\cancel{0}\sqcup$

For example, see the TM described below.

reject

Goal: Shift input to the right and add a # at



TM's can be used to compute functions

$w = w_1 w_2 \ldots w_n$: A binary number.

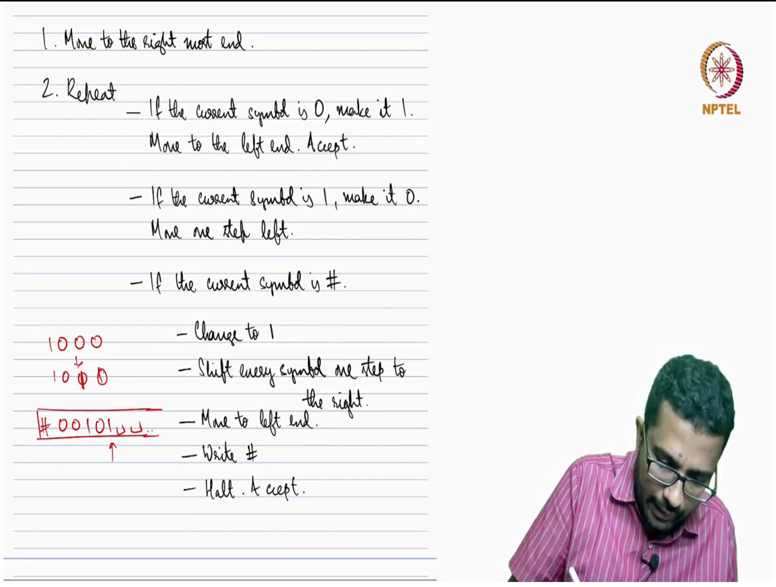
helps to indicate left end

Assume the tape contains $\# w_1 w_2 \ldots w_n$.

GOAL : To increment

1. Move to the right most end.
2. Repeat
   - If the current symbol is 0, make it 1. Move to the left end. Accept.
   - If the current symbol is 1, make it 0. Move one step left.
   - If the current symbol is #.
     - Change to 1

1000



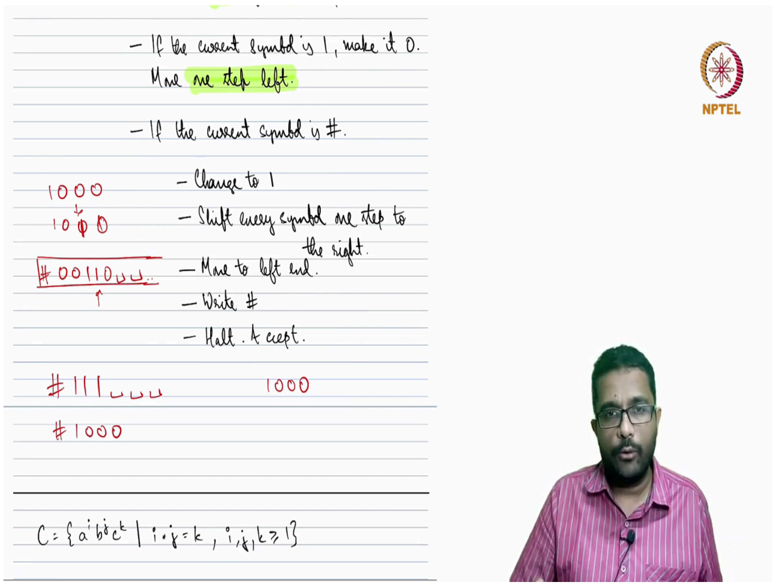
1. Move to the right most end.
2. Repeat
   - If the current symbol is 0, make it 1. Move to the left end. Accept.
   - If the current symbol is 1, make it 0. Move one step left.
   - If the current symbol is #.
     - Change to 1
     - Shift every symbol one step to the right.
     - Move to left end.
     - Write #
     - Halt. Accept.

1000

$10\cancel{0}0$

$\#00101\sqcup\sqcup\ldots$

↑


- If the current symbol is 1, make it 0.
  Move one step left.

- If the current symbol is #.

1000
10ϕ0

- Change to 1
- Shift every symbol one step to the right.

#00110⊔⊔..

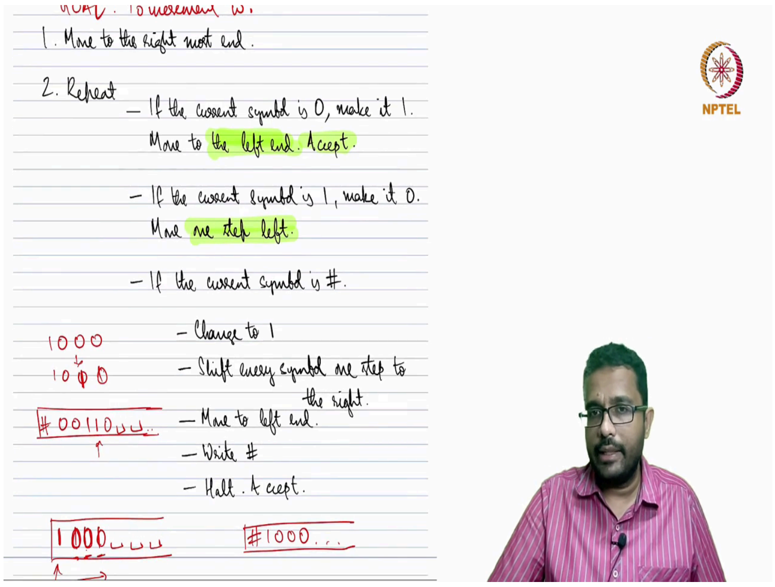
- Move to left end.
- Write #
- Halt. Accept.

#111⊔⊔⊔

1000

#1000

$C = \{a^i b^j c^k \mid i \cdot j = k,\ i,j,k \geq 1\}$


1. Move to the right most end.

2. Repeat
   - If the current symbol is 0, make it 1.
     Move to the left end. Accept.
   - If the current symbol is 1, make it 0.
     Move one step left.
   - If the current symbol is #.

1000
10ϕ0

- Change to 1
- Shift every symbol one step to the right.

#00110⊔⊔..

- Move to left end.
- Write #
- Halt. Accept.

$\boxed{1000⊔⊔⊔}$ $\boxed{\#1000...}$

So, what I want to just explain in full detail is given a string like this 0001101. How can we using Turing machine rules, put a hash symbol to the left side and then shift everything to the one step to the right. So, this is what I am talking about. So, given a string given a binary string, I want to shift everything to one side, one step to the right and put a hash symbol on the left side.

So, this is what I want to do. So, this is a Turing machine that will just do this operation it will not really check something. So here it was checking with a number of 0s a power of 2 or not, but here there is no checking because it is just asked to execute something or do some operations.

So it will just do the operation and accept there will be no reject because there is no scope for rejection. So, now, we have a Turing machine, it has two symbols, two input symbols

0001101 and the tape symbol there is a hash. And at the end of the operations, the tape will be, the tape will have the required thing meaning the original input shifted by 1 step to the right and the hash symbol.

And there are four states or maybe more than four states. There is a start state q s, there is a state 0, there is a state q 0, there is a state q 1, there is a state q 2 and there is an accept state. So start state as q s so node the small difference instead of q 0 I am calling it q s the start state qs, so, 1 2 3 4 5 states, qs, q0, q 1, q 2 and q 8.

So, let us see what these states do. So originally, maybe we will see this through an example. Let us try to work out some examples. Let us take a different colour, maybe let us say 001011. So this is the string and this is blank, blank and so on. So this is a string that we want to and originally we are here. And we are at the start state q s.

So, let us see what happens. So initially is q s and 0 ? So we will use this rule q s 0 is q 0 hash, or, meaning what do we do. So we move to state q 0, so be moved to state q 0. So I am just writing the states below, and the 0 is replaced by hash. So, 0 is replaced by maybe I will use a different colour 0 is replaced by hash and one step to the right.

So, basically what this q 0 indicates, so this is 0 and the q 0 indicates that the last symbol that saw was a 0. So that is why we have q0 and q 1 so we will come to q2 later, q 0 means that so now I have erased a symbol 0 and put a hash. And I need to remember that it was 0 that I erased and that is what we are accomplishing through q0 if it was 1 that was erased we would have gone to q1, q s if the first symbol was 1 we would have gone to q1 see this is q 1.

Anyway now we are at q0 and at the second symbol. So because we erased a 0 we are at q0 and because we are at q0 now we will write 0 also. So, let us see what so now we are at q0 and we are reading 0, so we have to follow these rules q0 0. If you add again going to q0, so you are again going to q0, but you write 0 also. So thankfully, it is already 0 there. So you write 0 and then you move one step to the right.

So again at q0 and you move one step to the right, because it was 0 earlier, I am not going to erase and write 0 again. So, now you are at q0 and reading a 1. So, the rule to be used is this one q0 1. So, now you have to move to state q 1 and write 0 and move on separately. So, you go to state q 1 and write 0 moves one step to the right.

So, now you are in state q1. Now at q1 you are seeing a 0. So, you have to use this rule. Now, you have to write 1 and you have to again move to q0. So, you have to write 1, and again move to q0. So, let us see you have to write 1 and again move to q0 which I have written here and then move one step to the right.

So, now you are at q0 and reading a 1. So which means this is the rule to be followed which means you will move to q1 maybe I will write here you will move to q1 and write 0 and then move one step to the right. So because you were at q1, now you are at q1 and looking at 1. So, now this rule has to be imposed which means you will remain at q1.

So, you will remain at q1 but you will write 1 which is already 1 anyway there and then you will move one step to the right. So now, you can see how using the states I am keeping track of what I just erased. So now we come to a blank but you are at q1. So, what is a rule to be applied is this one you are at q1 and then reading a blank.

So q1 means the last symbol you erase is a 1. So which means this blank should be erased and 1 should be written here. And because we just erased a blank, we know it was the last spot. So then we move to q2. So q2 means that now we have come to the right most endpoint. This is the meaning of q2, which means now it is a signal for us to move to the left, because at the end, I want to come back to the left side and stop, I could have stopped here.

But just for completeness, I want to come to the left side and stop. So, now let us see what q2 does. The rules for q2 are, if it sees a 0, it writes a 0 and most left, if it sees a 1, it writes a 1 and then moves left, meaning it leaves the symbol unchanged. So now what will happen is that it is a q2 and in the last position, it will come to one step before leaving the 1.

And again, it is here. So it will come here leaving the 1, it will come here, it will always be in q2. So q2, it will go one step to the left, one step to the left, and so on. And finally, it reaches the hash symbol.

So when it reaches a hash symbol, we know we put the hash symbol at the leftmost point. So when we reach the hash symbol, we know that we have reached the leftmost point. So then this rule applies. So q2, at state q2 reading the hash symbol means now you are done with whatever you set out to do, which means you can move to qa, which is the accept state. So then you complete the execution.

So once you come back to the leftmost point you accept and you are done. So there is a left move that is a set here. But this does not do anything because you are in the leftmost point. So trying to move left does not do anything. So now you see how we were able to do some simple things like this, like shifting everything to one step to the right.

But writing a hash at the beginning. This was accomplished using a Turing machine. So like this, this is a fairly simple task. But if you see the kinds of things that we talked about here, move everything from left to right crossing, all of this can be accomplished, because these are fairly simple tasks.

And these can all be accomplished using states. So the point here is that I will not or we will not be going through things in this level of machine level detail or the state level rule level detail. The point is that from now on, we will just take it for granted that these kinds of simple things can be done. And we will not bother with the details.

If you want, we can get out the details, but we will not be doing so just for the sake of time, we will point to see other things that we can learn about Turing machines. So, we will resort to high level language, when I say high level language, I mean language like this. So depending on how high this is not that high, actually, there is still quite a lot of detail, but it is not at the level of details of states and transitions.

That is what I mean by a little bit high level so some maybe medium level detail. So this, we saw. And the next thing I want to say is we can use Turing machines to compute functions. So suppose we want to increment a function. Increment a number sorry, suppose a Turing machine there is a binary number w1, w2 up to wn. And let us say we want to increment this number. Let us see how to do this. And just for the sake of simplicity, let us assume that the tape contains a hash symbol. At the left most endpoint why this assumption helps is because this hash symbol helps us to indicate the left end.

So this tells us that we have reached the left endpoint otherwise, there is no way the Turing machine can know that it is at the left endpoint. So let us see how to increment something. So maybe, we will try to write some numbers. Let us say hash 00101, blank, blank. Let us see what happens. So first, we move to the right most end.

So basically, we look for a blank and we get to the right most end, so we come to the right most end. And then, so we come to the blank, and then we can come back one step, once we see the blank. If the current symbol is 0, we make it a 1. And that is it. But here the current symbol is not 0, it is a 1. So then basically the point is this, if the number was something like 1000, the incrementing is 1001.

So once we make the 0 to 1, we are done, but if we want to increment 1001, we actually have to change this to 0 and do something to the bit to the left side of it because there is a carry operation happening. So, that is why, if the current symbol is 0, we move to the left and accept. This is done. But if the current symbol is 1, it is kind of you just move one step to the left and do not accept, you continue.

So maybe I should here, the goal is to increment 1 to increment w. So this increment means adding 1 to it. So now, the current symbol is 1, so 00101, the current symbol is 1, so you make it a 0, and then you move one step to the left, and now we current symbol is 0. So now you just make it a 1.

And if since the current symbol was 0 was made into a 1, we can follow this rule, the first rule, we just moved to the left end, and you accept so you come here and you accept and the hash symbol will tell us that we have reached the left end, so, we can come back to the left one step one step until we stop when we reach a hash.

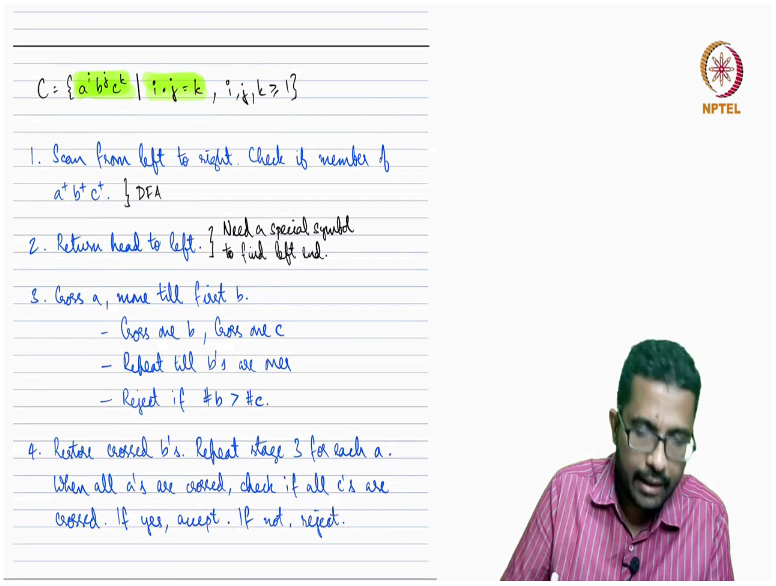
That is why the hash itself is so, that is these two rules will take care of almost everything, just that, in some cases, we need to do something more. So, let us tell you, let us see why. Suppose the number was 111. So, basically a string of all, ones. So now when we increment this, the resulting number is going to be 1000. So basically, we need to get to hash 1000 meaning we need to go once and push everything one step to the right from a three digit three bit number, I am going to a four bit number. So, that requires some work.

So, let us see how this is done. So, we first come to the leftmost point this 1 and this is first made into a 0, and then you come to the one step to the left further left and you have a 1 so, this is made into a 0. So, you change it into a 0 and you come here and this also you make it to a 0 and then you come to the left, but now we see a hash symbol. So, if you see a hash symbol this part kicks in.

So, if the current symbol is hash, what we do is you change it to 1. So, you change this hash to 1. So, now, this is a tape you have 1000 and we are kind of done. We have incremented the thing but we started with this hash on the left side. So the output that we desire is a hash followed by the incremented value. So, how do we do that?

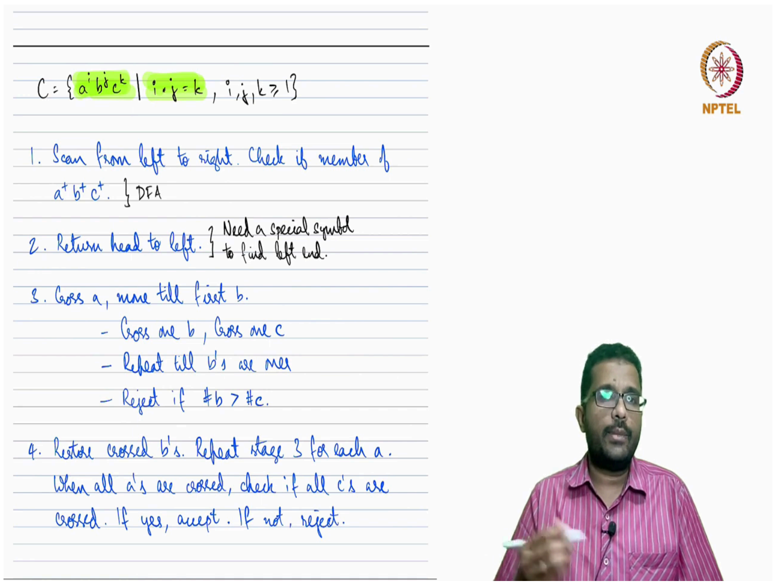
We have to move everything by right to one bit and put a hash in the insert a hash here and shift everything by one side. So, that is exactly what we described about you shifting everything to the right and adding a hash at the beginning and we can do that and then you accept and stop. So this is how you would implement an increment function. So, again, things are not in full detail here. But the point is that you can implement it in full detail if you want.

(Refer Slide Time: 34:08)



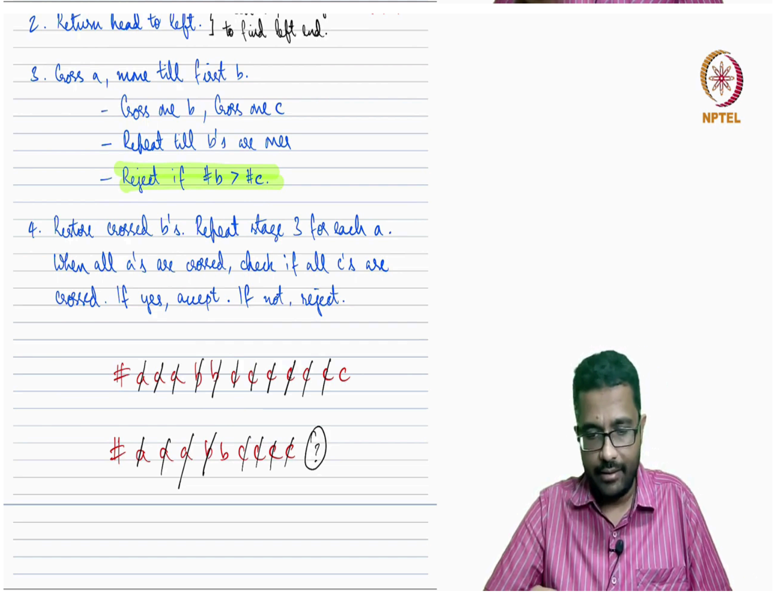
$C = \{a^i b^j c^k \mid i \cdot j = k, \; i, j, k \geq 1\}$

1. Scan from left to right. Check if member of $a^+ b^+ c^+$. } DFA
2. Return head to left. } Need a special symbol to find left end.
3. Cross a, move till first b.
   - Cross one b, Cross one c
   - Repeat till b's are over
   - Reject if $\#b > \#c$.
4. Restore crossed b's. Repeat stage 3 for each a. When all a's are crossed, check if all c's are crossed. If yes, accept. If not, reject.



$C = \{a^i b^j c^k \mid i \cdot j = k,\ i,j,k \geq 1\}$

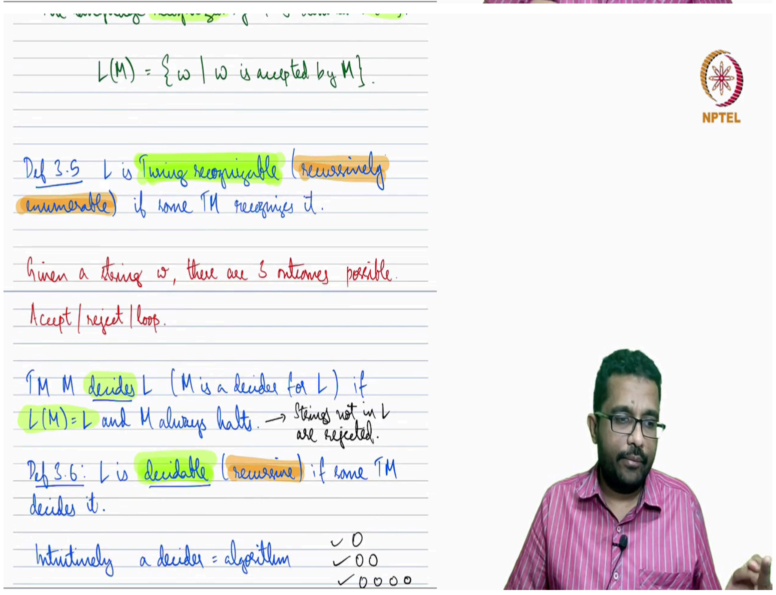
1. Scan from left to right. Check if member of $a^+ b^+ c^+$. } DFA
2. Return head to left. } Need a special symbol to find left end.
3. Cross a, move till first b.
   - Cross one b, Cross one c
   - Repeat till b's are over
   - Reject if #b > #c.
4. Restore crossed b's. Repeat stage 3 for each a. When all a's are crossed, check if all c's are crossed. If yes, accept. If not, reject.



2. Return head to left. } to find left end.
3. Cross a, move till first b.
   - Cross one b, Cross one c
   - Repeat till b's are over
   - Reject if #b > #c.
4. Restore crossed b's. Repeat stage 3 for each a. When all a's are crossed, check if all c's are crossed. If yes, accept. If not, reject.

$\# \cancel{a} \cancel{a} \cancel{a} \cancel{b} \cancel{b} \cancel{c} \cancel{c} \cancel{c} \cancel{c} \cancel{c} \cancel{c}\, c$

$\# \cancel{a} \cancel{a} \cancel{a} \cancel{b}\, b\, \cancel{c} \cancel{c} \cancel{c} \cancel{c}\ \textcircled{?}$



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

Def 3.5 L is Turing recognizable (recursively enumerable) if some TM recognizes it.

Given a string w, there are 3 outcomes possible.

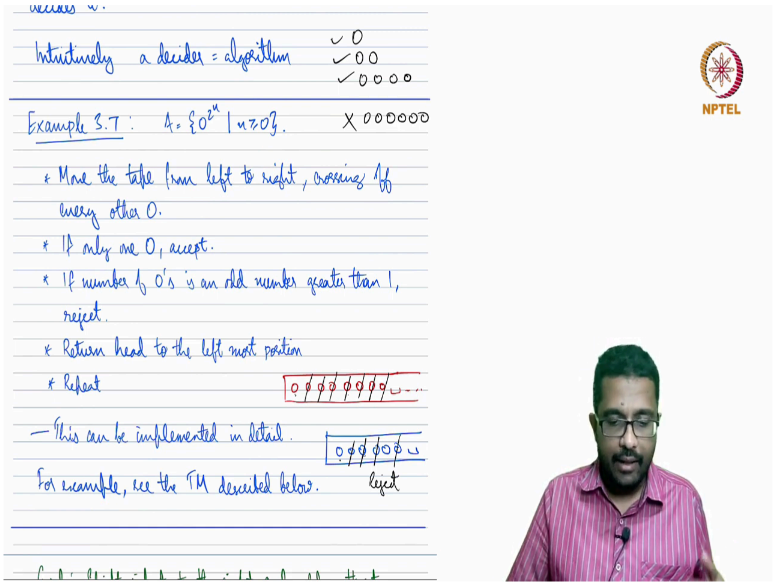
Accept / reject / loop.

TM M decides L (M is a decider for L) if L(M) = L and M always halts. → Strings not in L are rejected.

Def 3.6: L is decidable (recursive) if some TM decides it.

Intuitively a decider = algorithm

✓ 0
✓ 00
✓ 0000


Intuitively a decider = algorithm

✓ 0

✓ 00

✓ 0000

Example 3.7 : $A = \{0^{2^n} \mid n \geq 0\}$. X 000000

* Move the tape from left to right, crossing off every other 0.
* If only one 0, accept.
* If number of 0's is an odd number greater than 1, reject.
* Return head to the left most position
* Repeat

— This can be implemented in detail.

For example, see the TM described below.

Reject

Finally, one more example. So, this is kind of like testing multiplication. So, this is like trying to see how to do multiplication. So the language here is C language is called C, it is $a^ib^jc^k$ where k is the product of i and j, k is i multiplied by j. So, let us see how to do this. So, you have let us say, we have a a a b b c c c c c c.

So, you first check whether the input is of the, a star b star c star. In fact, I am saying that i j k are all at least 1 so, a plus b plus c plus. So, check whether the input is of this format. In fact, this is something simple even though these are actually regular expressions; this is something that the DFA can itself handle. So, you want a star b star c star but at least 1 a, 1 b, 1 c, a plus, b plus, c plus.

So, this is something a DFA can check and then you come back to the left side. So, this is something simple I will not spend much time and coming back to the left we may have a, we may have to have a hash symbol. So, this is something like the dollar that we used to put at the bottom of the stack like that we put a hash symbol to the left of the input, because you keep coming left, left, left, left, left, left, left and finally you reach the hash and then you know that you have reached the leftmost otherwise how do there is no way to tell.

So, now we do that, let me just get that out of the way, maybe I will put the hash. So, now, what we do is we do the following: we crossed the a, you cross an a, you cross one a and then you go and check you go till you see the b's. So, first you see a b you cross one b and then you cross one c, then you come back, you cross another b you cross another c, then you come back, then you see that there are no b's. So, all the b's are crossed out. So, now you decide to erase this. So now, basically we want to check if the number of a's multiplied by the number of b’s is a good number of c's. So here number of a's is 3, number of b’s is 2 and number of c's is 6, in this case, it is an instance that should be accepted.

So, for each a you cross out b’s and cross out c’s and after which you uncrossed the b’s you uncrossed the b’s and now you strike out the next a and then you cross out a b you cross out a c, you cross out a b you cross out a c. Now you again remove the crossing of the b’s you strike the a, you cross out a b, you cross out a c you cross out a b you cross out a c. And now you see that all the a’s are crossed out.

And now you check whether any c’s are remaining if any c’s are remaining you reject. If no c's are remaining you accept. So, in this case everything is crossed out. So, you accept. So, suppose let us say there was one more c, in this case you would have rejected. So, here the number of c's is more than the number of a multiplied by the number of b.

So, on the other hand, we could have had some situations like this a a a, b b, c c c c, something like this. So here, you would run out of c’s first. So in this case, you would have struck out an a then this b, this b this c this c and then you would unstrikes the b’s. The strike sorry, sorry, you strike out the a, this b this b this c this c and then you again unstrike the b, strike out the a, this b you strike out, but then there are no c's here.

So if there are no c's here, again, you will reject. If at any point you do not have you strikeout of b. But you do not have the c’s you reject that is what is happening that is what is happening here. If at some point you have struck out a, b but there are no c’s to strike out, you reject otherwise, if all b's everything is fine, but at the end, you see everything is matching up then you accept.

So this is how you can use the Turing machine to kind of do multiplication or kind of check verify multiplication. So here that is what we are doing. You are checking whether the number of c’s is equal to the number of a's multiplied by the number of c’s, sorry, number of b’s. So, these are some examples of Turing machines. So, just to quickly summarise, we said what are the three possibilities for a Turing machine on a string: one is to accept one is to reject and one is to loop.

If a Turing machine. So, what are the and two main definitions two important definitions I will repeat a language is Turing recognizable or recursively enumerable if some Turing Machine recognizes it, meaning, it accepts all the string in the language and does not accept any string that is not in the language does not accept can mean two things reject or loop.

We say a language is decidable or a Turing Machine decides that language if it accepts every string in the language and rejects every string not in the language meaning it does not loop on any string at all, all the strings in the language are accepted, all the strings not in the language are rejected.

So we say language is decidable if or recursive decidable or recursive, if there is some Turing machine that decides it. And then we saw some examples like how to check how a Turing machine could do some simple things like check some simple languages, like a being 0 to the 2 power n, or do simple tasks, in which we went into some detail.

And some computing functions like incrementing a string, incrementing a bit binary number and checking some simple languages. So that completes this lecture, which is lecture number 28. In the next lecture we will see variants of different types of Turing machines. So what we saw is a deterministic Turing machine with a single tape. We will see different types of other types of Turing machines. Thank you.