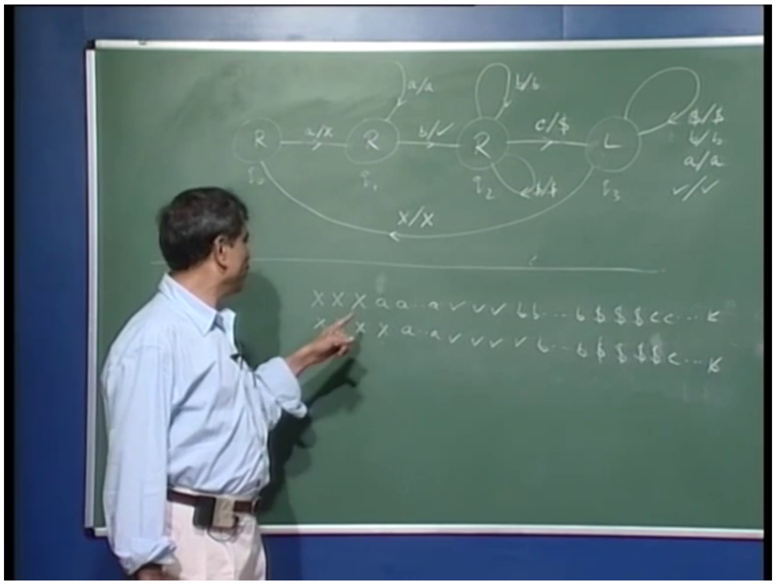
Course on Theory of Computation
By Professor Somenath Biswas
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Lecture 35
Module 1
Execution trace, another example (unary to binary conversion)

(Refer Slide Time: 0:25)

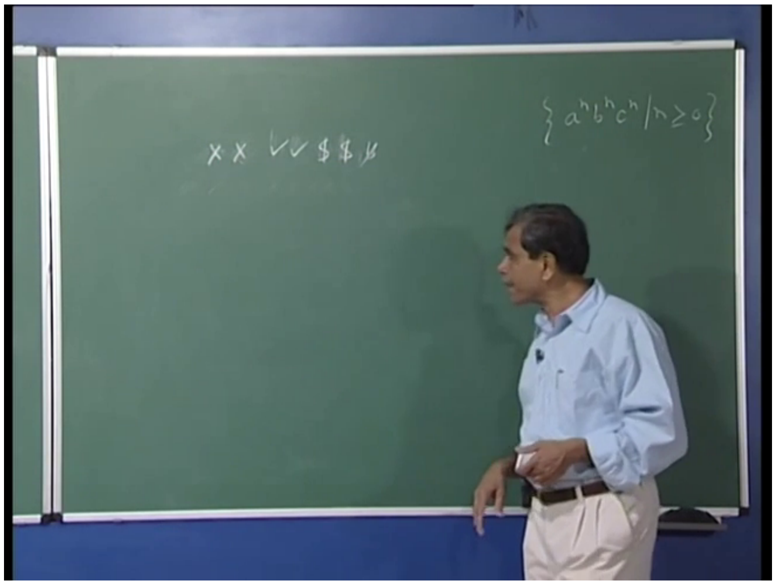
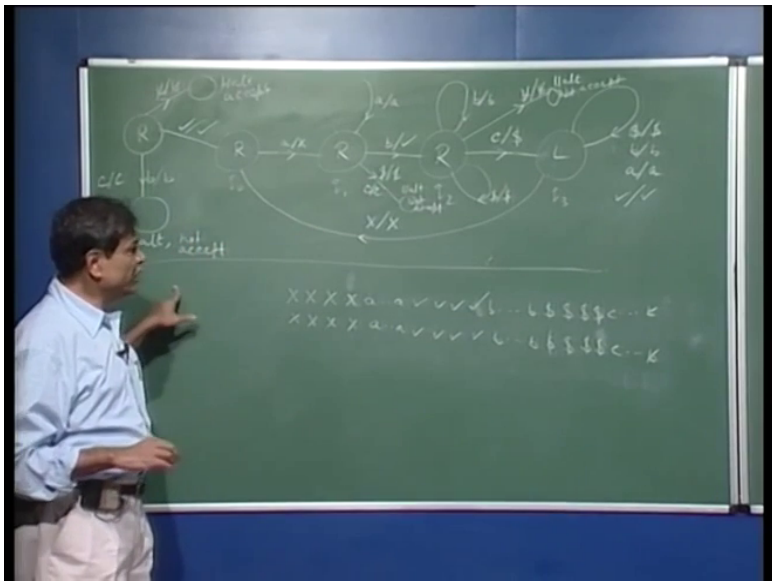


To reiterate one iteration of the machine that we are trying to build is that it will consider 1 a and will find a corresponding b and corresponding c, then it will come back and possession itself to the a which was just checked. So what we wanted an iteration once more that this is the next a the left most available a so this will be crossed off and then the machine will move to the right and in this process what it is trying is that it will find a corresponding b it will put a tick and still go on towards right find a corresponding c and then it will put a dollar.

So that means corresponding to that a it has found a b and c and now the iteration should start all over again but now it should possession its head here so that the next iteration can start alright? And we have seen that these four states when you are when the machine is first is in this state starting the iteration it will come back to this state for the next stage for (())(1:59) it will possession itself correctly for the next iteration.

Now so this iterations are going on then what happens? At the end several things can happen that it will find that indeed all the a's have been checked off with b's and c's should then the machine come to the conclusion that the number of a's is equal to number of b's equal to number of c's?

(Refer Slide Time: 2:38)

Not really you see it is like this suppose you have number of a's and then slightly smaller number of b's or number of b's which is less than the number of a's and then some c's. So you will be able to check off for each a 1 b this a also you will be able to check off 1 b and 1 c but in this particular example corresponding to this a we will not be able to find the b so that is one possible way that the machine will find that something is wrong.

So there are several cases the right case is of course the number of a’s is equal to number of b’ and equal to number of c's. So the head will be come all the way here it finds a next and move to the right. So long the iterations are going on it will find an a, but now if the block of a’s are over what is it going to find? It is going to find a tick as it moves from the X to right. So it has found an X it of course does not change that X and then moves to the right in this case you can see that it will find the tick.

So it is possible that it will find a tick in this place what does that signify? It signifies that the block of a’s is such that for every particular a we have been able to tick off a corresponding b and a corresponding c, right? So in that case when this tick comes we know that block of a’s have been you know taken care of, now the question is the input is of the correct form provided no b’s or c’s are left.

So how can we do that? We will move to the right and as we move to the right if we find any b or any c left then of course we know that number of a’s was less than either number of b’s or number of c's. So for example in this case if we are moving to the right see we have come to this state moving now moving to the right to check if there is a b or a c still left with as not been you know either in case of b it has been ticked off or in case of a c we had put a corresponding dollar. So in case we find a b what we know we do not have to change that b but we know that this input that we have has more b’s than the number of a’s.

So in this particular case what we can do we can go to a state where we can halt and with the understanding that the string input string is not a correct one so we can say halt and not accept. Similarly if we find c again you can write a c there and say the number of c’s is more than the number of a’s such was the input. So in that case again we should halt and not accept, however as we are moving to the right in case we find you see what is going to happen look at this case so it came here it found the tick this is where we reach we entered this state we found this state and moving to the right and then here what we are going to find if there is no b’s or no c’s left? We are going to find a blank.

So in this state as it is moving to the right if it finds a blank does not change that blank symbol but it knows that the number of a’s of course is exactly equal to the number of b's and the number of c's. So in this case it halt it goes to a state which is halt and accept, right? So have you taken care of all the possibilities? Not really, you see what may happen number of a’s could have been more than either number of b’s or number of c's that particular possibility we have not yet taken care of in this diagram in this transition diagram.

How does the machine know that the number of b's or number of c's is less than the original number of a's, so for example here you see it is in this state what it is looking for? It is looking for a b to put a tick but in this state if it finds that the c block had already started how would that happen that in this state if it sees either a c or a dollar then it knows that the b block is over and yet we are short of at least 1 b. So in this case for example if it finds a dollar or a c it knows that the number of a’s is definitely more than more than number of b's.

So let me just put it this way dollar of course it does not need to change it and similarly if it finds a c then again it is a c or you know it has gone you know it has found a not a b but either a dollar or a c. So in that case what should happen this case is again reaching a state where it is halt and not accept. So this takes care of the case where the input is such that number of a’s is more than the number of b's but it could have been the number of a’s is more than number of c's.

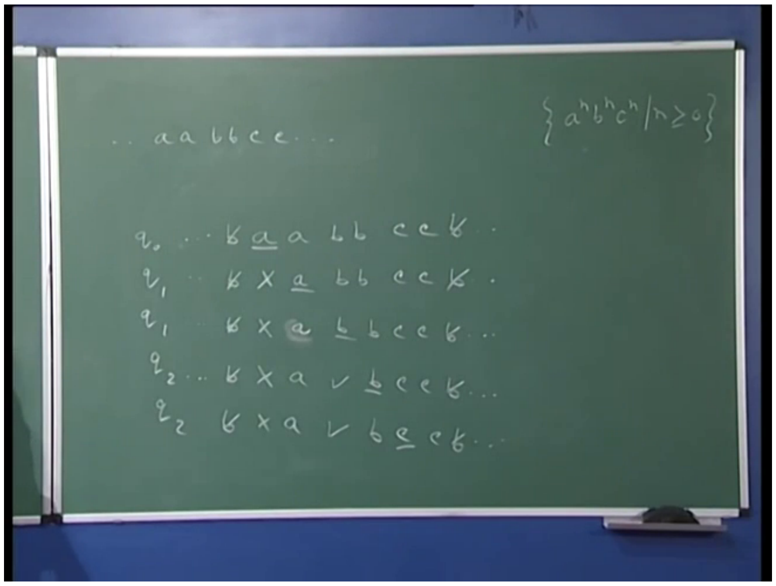
So that situation will be discovered here you see like in this state it is looking for a c and in case of course it finds a c it puts a dollar but if it does not find a c supposing there all c’s have been already taken care of then what is it going to find as it moves to the right keeps on moving to the right, it is going to find a blank. So in case it finds a blank then again it knows that the number of a’s is more than the number of c’s. So here again it should halt in this state and not accept, alright?

So you can see the basic idea is that with the capability Turing machine has that is being in a state depending on the input it can decide to go to some other state writing a particular symbol and move either left to the right this simple you know mechanism in defining a Turing machine at least in this particular example we can see that it (())(11:39) to distinguish between all strings which are of the form equal number of a’s followed by equal number of b’s followed by equal number of c's this set of strings can be differentiated from those string where number of a’s and number of b’s and number of c's did not match.

So we can therefore see that this device that is Turing machine can indeed do something which no pushdown automata could do which no context free grammar could generate could define that particular language of a n, b n, c n remember this particular language cannot be either generated by a context free grammar or equivalently cannot be recognized by a pushdown automata but here we see that a Turing machine does have the capability to recognize these strings and not those strings where number of a's and b's and c's do not match.

Let us just to be clear about the working of a Turing machine what we can do the way we can you know given a program we can trace the execution of a program in a similar manner we can trace the execution of a Turing machine, right? So let us just trace the execution of this Turing machine on a very simple input string.

(Refer Slide Time: 13:55)



$\ldots a\,a\,b\,b\,c\,c \ldots$

$\{a^n b^n c^n / n \geq 0\}$

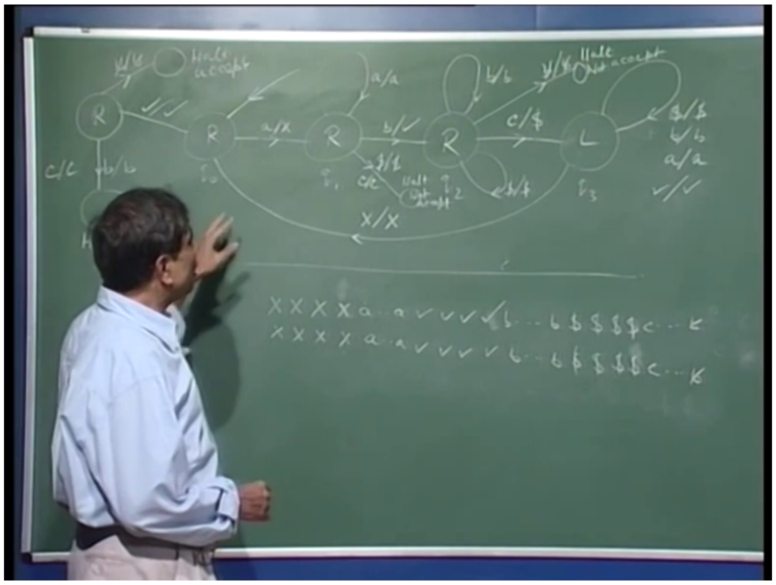
$q_0 \ldots \not b\, \underline{a}\, a\, b\, b\, c\, c\, \not b \ldots$

$q_1 \ldots \not b\, X\, \underline{a}\, b\, b\, c\, c\, \not b \ldots$

$q_1 \ldots \not b\, X\, a\, \underline{b}\, b\, c\, c\, \not b \ldots$

$q_2 \ldots \not b\, X\, a\, \checkmark\, \underline{b}\, c\, c\, \not b \ldots$

$q_2\ \not b\, X\, a\, \checkmark\, b\, \underline{c}\, c\, \not b \ldots$



So let us say our input string is at least I will carry out a few steps so that you understand what does it mean to trace the execution of a Turing machine. So let us say you had and I am not writing the blanks symbols so these are here, remember the Turing machines starts in this state scanning the left most a that is what we said.

In this diagram really we have not if you look at it there is no indication where does the Turing machine begin and by convention what we do I mean we know that we want this state is this state in which the Turing machine is supposed to begin, so what we do is we put an arrow coming from nowhere into this state to indicate that this is the initial state. So this is the initial state and we have the name for this state is q0.

So one way of doing this tracing the execution will be that will write the basically we will describe the tape remember all this is on tape and there are some blank symbols flanking both sides and the machine is in state q0 in the beginning and scanning this particular symbol one way of indicating all that will be the machine is in state q0 and scanning this, what we are going to do is going to put an underline to put a small line below the symbol where the head of the Turing machine is.

So this is describing our initial scenario, and then we write in the beginning machine is the idea is if it sees an a it puts an X and goes to this state. So the next snapshot or after the state change has been affected what are you going to see? You will see that the Turing machine is in state q1 and the head has moved one cell to the right, right? So we can describe that by saying the machine is in state q1 and the head is here, but you see before that what did it do? The machine as it change the state it changed the symbol it was scanning which was this a into an X so this is now an X and the head is here everything else remains of course as before, alright?

So the machine is in state q1 so this is the state and now what is what we can see that in this state what is it scanning? It is scanning an a what will happen it will remain in the same state and it will change that particular symbol you see. So the next time instance the situation will look like this, I am sorry so this is it is it was scanning an a it will write the same a, right? And move to the right, so it will now look at this symbol. In q1 now if you see a b you are supposed to put a tick and go state q2 this is the state q2, right?

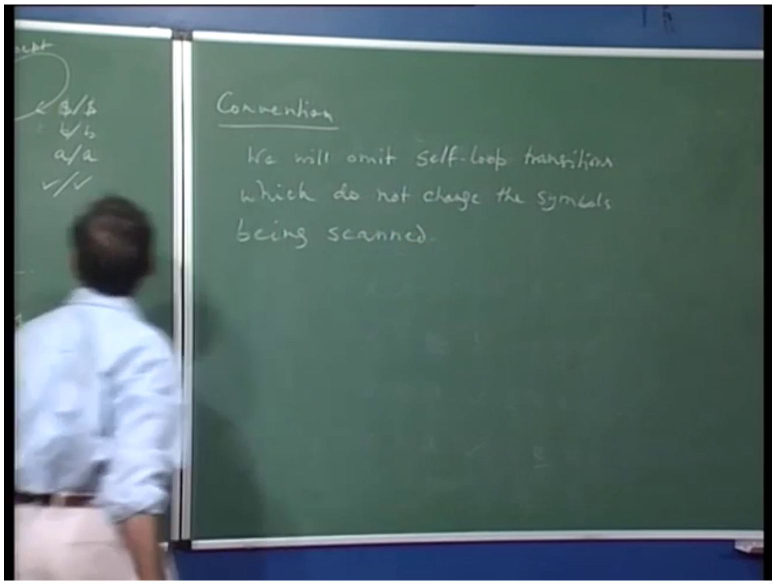
So therefore next step is going to be q2 and this b is changed to tick which we have shown there and what happens now you go to state q2 by moving to the right so the head has moved one cell to the right now it will scan a b and you know we can see that at least look at one more instance of time what is going to happen the machine is in state q2 scanning the symbol b so machine is in state q2 this is q2 remember scanning b it will remain in the same state and move to because see this is the diagram in this diagram the arrow is coming back to the same state and this state is associated with right move so the head will move one step to the right.

So really it will remain in the same state therefore without changing the symbol and it will move one cell to the right so therefore the machine will be in state q2 scanning the symbol c and this goes on. So although it is laborious process but then tracing execution of any program even a simple program you must have done it sometime it is fairly, what should we say it is a kind of tedious and laborious process but at the same time what I am trying to show you by means of this that it is possible to very methodically trace the execution of a Turing machine step by step.

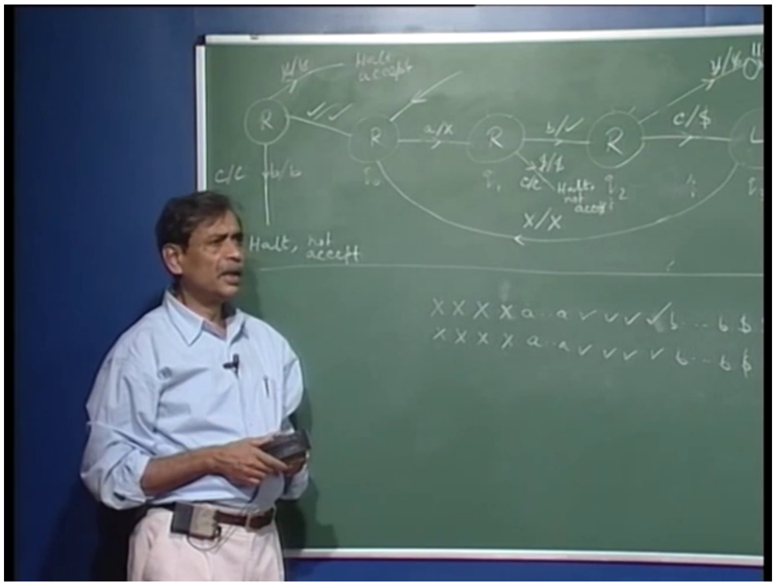
So just in case we have doubts about the working of a Turing machine we can follow an execution sequence to ensure to be sure that what exactly does the Turing machine do on the given input, alright. So now there are certain things in this diagram which is a little messy which we can take care of so that you know it does not distract us so much, see I mean if you look at this what does it doing, we are saying that if you are in state q3 and if you see a dollar write the same symbol there and remain in the same state.

So similar such things will be there all over we can avoid so much of clattering by a very simple convention that is we will not show on the diagram all those self-loops in which the symbol does not change, alright?

(Refer Slide Time: 21:38)


Convention

We will omit self-loop transitions which do not change the symbols being scanned.



So let me be very clear what we mean is so it is a convention that we will omit self-loop transition which do not change the symbol being scanned. So there are actually 1, 2, 3, 4 transitions which are like this here by once more by the self-loop transition we mean a transition from a state to itself and if in that transition the symbol that was being scanned at that time if it remains that it the Turing machine does not change that symbol, so how do we specify that? It is seeing a dollar and it writes back the same dollar, right?

So the symbol really does not change on the input tape. So in such case we say you know it is by convention we will not show this such transition, if we do that how will this figure look so we will remove this transition basically we are not removing few transition but we are not drawing the convention being that if you are in a state and nothing is specified that means you remain in the state without changing the symbol and the move is according to the move which is associated with the state.

So this is these are all also transitions which do not change the symbols and into the same state, so I can actually remove showing them explicitly and this is another such transition which also I do not draw it is understood and similarly this one, okay is there any other such thing not really and this looks neater because right for example here you are in state q3 and in q3suppose you are seeing a something let us say dollar, what would happen it is nothing is mentioned here this diagram it is not explicitly mentioned what the Turing machine does when the it is in state q3 scanning dollar since nothing is mentioned by this convention we will assume that it remains in the same state writing the same symbol, okay.

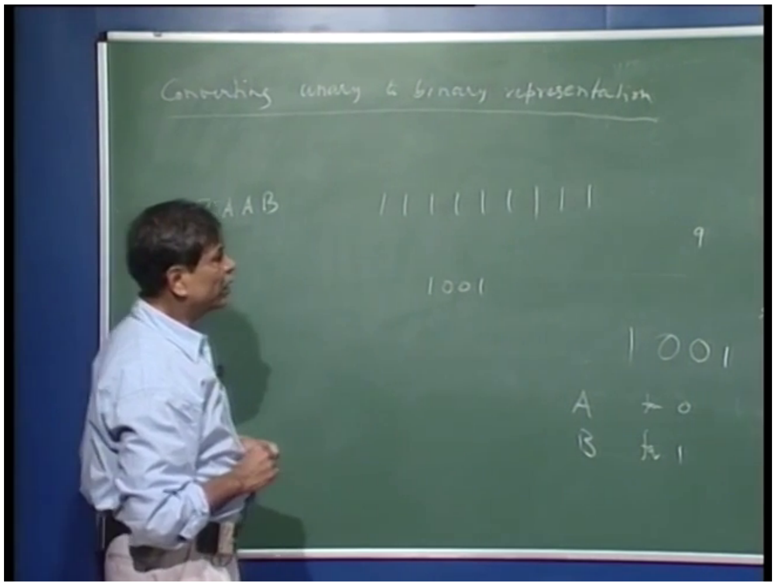
So we will see an another example where we will right away right from the beginning we will use this convention and only thing is of course if we follow this convention like here nothing is shown but then these are states where you would like to halt, so what we can do therefore instead of drawing a particular you know symbol or a circle for the state we will just say that we halt and do not accept and similarly we here will remove this and we will simple say that we halt an accept and so on, right? You understood what is the reason that we are removing this explicit showing of these states.

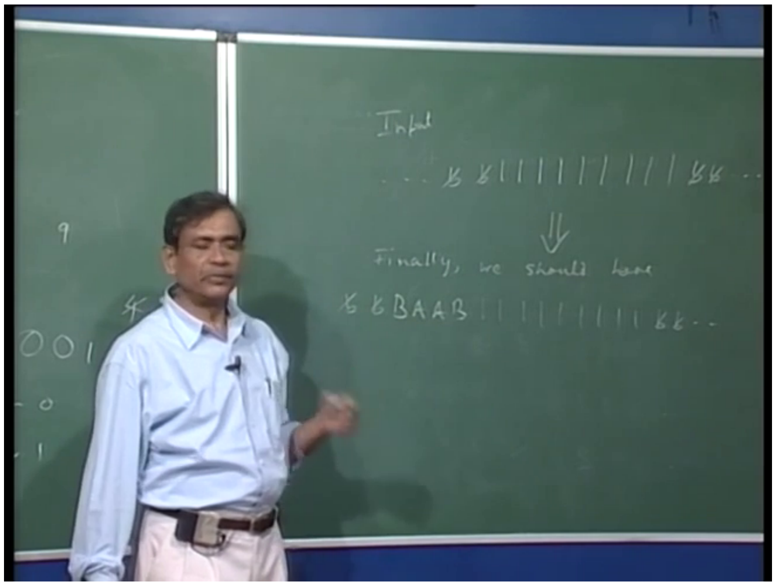
Reason is because if we follow that convention if we are here then it means that all those if I just draw like this it means that all those self-loop transition where there and then it is contradicting that we are saying it is halt, so we will just say we will remove this and we will say like here halt and not accept, okay. So at least we have one example of a Turing machine which does something which is in a way fairly complex because something this task is something which no finite state machine or no PDA could do.

Let us now consider one more example just to get our ideas well fixed about Turing machine and these transition diagrams.

The example that we consider now is converting a not let us put it this way that we will be given a block of 1’s in the input and corresponding to this block of ones if we see these block of 1's as coding a unary number what is the corresponding binary number representation, okay.

(Refer Slide Time: 27:54)





So we of course this task is we are all familiar with that suppose I am given 1, 2, 3, 4, 5, 6, 7, 8, 9 1’s let us say and in unary this block of 1's represents the number 9, so 1, 2, 3, 4, 5, 6, 7, 8, 9.

In binary we would say that this being the binary representation of the same number 9 or the task that we would like to carry out is that given a block of 1's we would like see it as a unary number and the corresponding binary code for that number we would like to write down by means of a Turing machine, so therefore it is the task of you can say the problem is converting unary representation to binary.

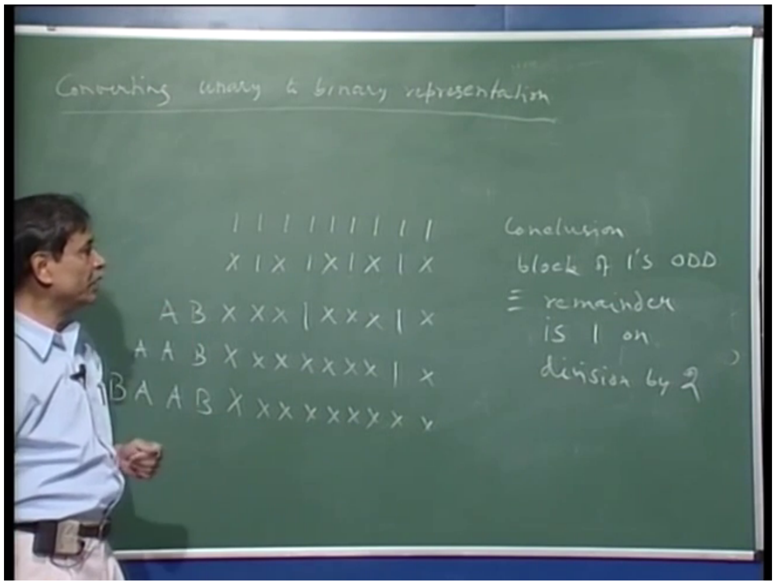
So how do you do it? Of course we know all of us know the algorithm for obtaining the binary representation from a unary representation what we do we divide it by successively we keep diving by 2 and if the remainder is 1 that is we know that the one more bit that we have found of the binary representation and then we work with the result of the division by 2 for the rest. So for this 9 what would happen that if you divide it by 2 you will be left with remainder so that becomes the least significant digit and the result of the division by 2 is quotient is 4.

So 4 when you divide by 2 you are going to get quotient will be 2 and the remainder will be 0 that is the next significant bit of the representation that we would find then 2 if you divide by 2 you will get the remainder to be 0 and the quotient to be 1 result of the division to be 1 and this 1 when you divide by 0 if the result is going to be 0 the result of the division is 0 in the sense the quotient is 0 and the remainder is 1, so that we write here and this is indeed the and then this is become 0 and that means there is no more nothing more to be done in the conversion.

So this is what we would like to carry out by means of a Turing machine. Now only thing is we see that we have a block of 1's as well as if we use again 1 for the binary representation there is a kind of clash just it is simpler to use two other symbols for 1 and 0 the conventional 1 and 0. So let us say we will use A for 0 and B for 1 B to represent 1 in the binary representation. So in that case what we get corresponding to this we should get BAAB to fix our convention, ideas about the input itself remember what we have is to begin with a blank tape with some 1’s and a of course these are blank and we would like to get corresponding to this block of 1's the binary representation but we are coding 1 as B and A as 0 and it will convenient as you will see that to write the result at the left.

So basically this being the input finally what we should have finally we should have BAAB and block of these block of 1’s remember that these b with a slash stands for blank, alright. So this is the conversion that we would like to affect by means of a Turing machine. We have said divide by 2, take the remainder how do we carry out these activities by means of the kind of primitives which are available with Turing machine. In this case that can be done quite simply.

(Refer Slide Time: 34:50)



Converting unary to binary representation

```
      1 1 1 1 1 1 1 1 1
      X 1 X 1 X 1 X 1 X
  A B X X X 1 X X X 1 X
 A A B X X X X X X X 1 X
B A A B X X X X X X X X X
```

Conclusion
block of 1's ODD
≡ remainder
is 1 on
division by 2

So let me first show you the idea you remember we had this 9 block is that example we had 9 1’s. Now suppose what you do is you make a scan and so what the idea is put a cross for this and then leave it like that, then so what is happening the first 1 that you get you put a cross the next 1 you do not change however the next 1 you put a cross, next 1 you do not change, next 1 similarly we will put a cross, this 1 you do no change, this you put a cross, this you do not change, this you put a cross, now you see what we are seeing what we are doing¸ we are it is like the previous example that we are trying to check for every 1 another corresponding 1.

So for this 1 which we crossed I found the 1 which I am not really changing, then we are doing this for this 1 found this 1 for this 1 we found this 1, for this 1 found this 1, for this 1 we what happened we did not find another 1. So once more that for this 1 we found another 1, for this 1 we found another 1, for this 1 we found another 1, for this we found another 1, for this we put a cross but we did not find another 1, did you see what is the conclusion that you can come to? The conclusion is that the original block of 1’s must have been odd because it is of course obvious that if you have an odd number of 1’s then as you are pairing off 1, 1 with another 1 one particular 1 will be left off and which is this one which got left off.

So we know that this is odd another way of saying is so conclusion in this case is block of 1’s odd but if we have a block of 1's which is odd that means what that if we divide by 2 the remainder is going to be 1. So this is equivalent to this conclusion that the original block of 1's is odd that is equivalent to that the remainder is 1 on division by. So we have found indeed the least significant bit of the binary representation, right?

We have found the least significant bit of the binary representation which because the remainder is 1 that least significant bit is going to be 1 so we should write a B here but remember that now we should carry out with the iteration and we should have had somewhere the result of the division, when you divide 9 by 2 result of the division is 4 but you see the way we have done the number of 1's which are left on this block if you ignore the crosses the number of 1’s which are left is indeed the result of the division.

So you see so again the next iterations suppose you write this B that is the result the remainder of the division by 2 then you had this, now again you do the same thing this 1 you put a cross and of course ignore any previous crosses and then this 1 so the next 1 that you have ignoring crosses you leave that crosses you ignore this 1 you put a cross, cross and this 1 is left as such and this one is left as such and then it is cross.

So now you see what is happened that this 1 you could check off, this 1 you could check off and there were no other 1. So in that case we know the result of the division by 2 of the block of 1's remaining here that is 0 because that because the number of 1 in this case is even we checked off we could manage to check off each we could pair up like this paired up with this this particular 1 was paired up with this and therefore the result is of the division is 0 result in the sense the remainder of the division by 2 is 0 so here we should write A and start the next iteration because you can see what is going to happen after this we remember we are ignoring crosses so this 1 will put a cross and ignoring crosses the next 1 we will leave it as such, this is cross.

So again the result if the remainder of the division by 2 is 0, so BA so this will again be A you know we are writing getting the next significant digit starting from the least significant bit and finally this case we have 1 so just the block of 1’s has 1 so we ignore crosses and we make it a cross and we find that there is no 1 to pair this particular 1 with so the result of the division is by 2 is odd so we know that this would be B and this what we see there are no more 1’s left if you recall what we did for this example when we did the usual kind of division by 2 take the remainder when finally successively after dividing by 2 when the you know number that was finally left was 0 that means we have taken care of the entire number and whatever we have is the binary representation.

What we are going to do now is to see how this can be carried out by means of a Turing machine and we will see that Turing machine is going to be fairly simple.