Theory of Computation

Prof. Kamala Krithivasan

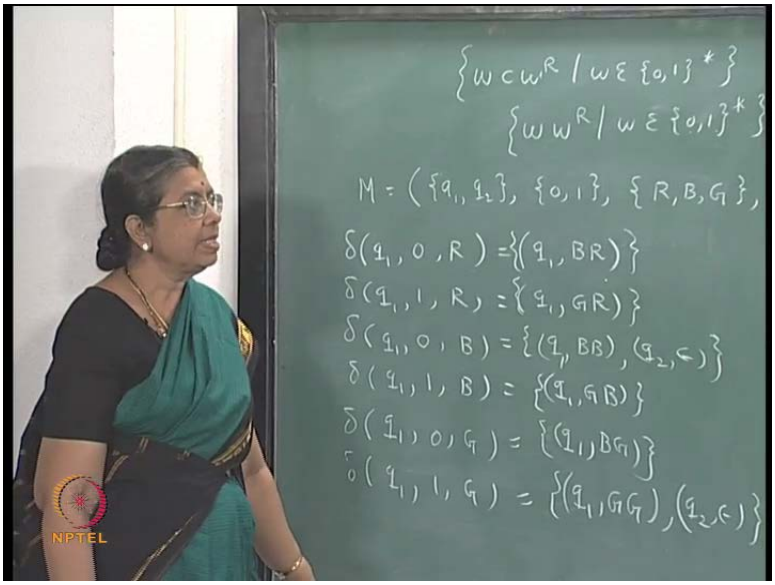
Department of Computer Science and Engineering

Indian Institute of Technology, Madras

Lecture No. # 21

Pushdown Automata Equivalence Between Acceptance By Empty Store and Acceptance by Final State

(Refer Slide Time: 00:17)



We were considering pushdown automaton. So, we consider one example w c w R, where w belongs to 0 1 star. For this we constructed a pushdown automaton, we wrote down the mapping and so on. And in pushdown automaton there are two ways of acceptance, one is acceptance by empty store, another is acceptance by final state, and by definition the pushdown automaton is nondeterministic in nature. Nondeterministic pushdown automata they are equivalent to context free grammars; that is what we are going to study.

So, when you just say finite state automata generally you mean deterministic finite state automata, because nondeterministic finite state automata and deterministic finite state automata are equivalent, whereas in the case of pushdown automata, when you say pushdown automata, you mean nondeterministic pushdown automata. Deterministic

pushdown automata and nondeterministic pushdown automata are not equivalent. Deterministic automata accept only a sub class which is important from passing point of view, so instead of considering this language.

Let us consider w w R where w belongs to 0 1 star. That c is not there. In the earlier case if you remember what we did in the last class? When you are reading w, when you see a 0 you are putting a blue plate and when you see a one you are putting a green plate or in the stack you are adding a symbol B when you see a 0 and adding a G when you see a 1. And then when you see a c you change the state from q 1 to q 2 and when you; in state q 2 when you see a blue and 0 combination you remove the blue symbol.

When you see G and 1 combination you remove the G symbol. Like that you are doing and the end the bottom most stack symbol, red symbol was removed, R was removed. What happen when you have 0 1 star, Ssppose you have string of this form or a string of this form w w R. In that case if you want to simulate the same move after reading this 1, when you read the second one it may be the continuation of w R, the beginning of w R. You do not know. When you read this 1 you do not know whether it is a continuation of the w R, the beginning of the w R.

And then suppose you take it as w at this point again you do not know when you see this 0 whether it is a continuation of w R, the beginning of w R. That is not known. So, you have to allow for both possibilities. So, you must have a nondeterministic machine, so the same thing if you try to do the same concept of acceptance as for w c w R.

If you write the machine it will have two states q 1 and q 2 input symbols are no c. Now, 0 1, pull down symbols are red, blue, green. Again you can have the same symbols and delta is the mapping which you have to define. Then the other components initial push down symbols is red acceptance by empty store empty, initial state is q 1. All these components you know.

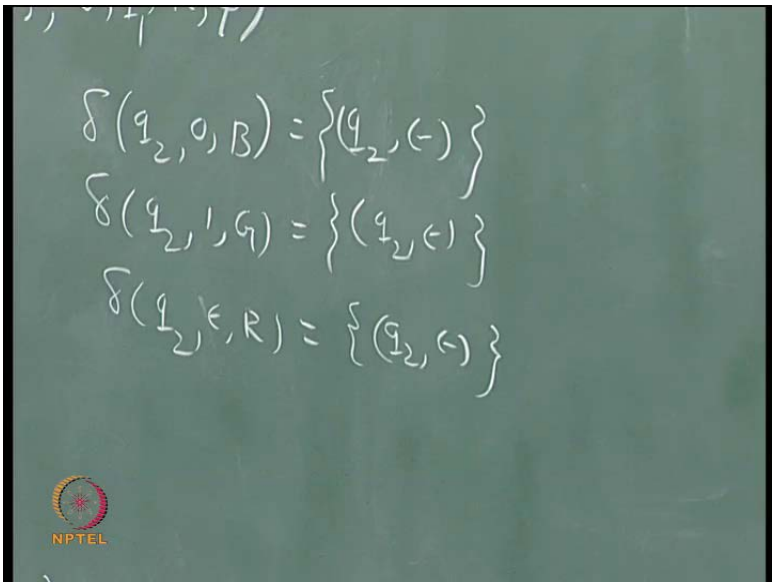
Now, you have to write the delta mappings. So, what happens delta of q 1 R when you see a 0? You add a blue plate or blue symbol is adder. In q 1 when you see a with R as the symbol you again add a G. So, q 1 G R. (No Audio Time: 04:43 to 04:48) In q 1 the top place is blue, if you see a 0 that means a last symbol your red is 0, there next symbol you are going to see is a 0. So, that accounts have two possibilities it may be the continuation of w R, the beginning of w R.

So, here actually I should write a flower parenthesis here because it is nondeterministic. Here you can either add a blue plate, and stay in q 1 or you can go to q 2 and remove the blue plate. This accounts for the case when it is a continuation of w and this accounts for the case when it is the beginning of w R, but in q 1 if you see a 1 with the B on the top, last symbol red is B is on the top means what is the last symbol red? 0 in q 1 and when you add a 1, when you see a 1 you can add a green plate. So, give q 1 G B.

In q 1 if you have a G and a 0 in q 1 you are adding symbols and in q 2 you are removing them. In q 1 if you have a G means last symbol red is 1 you are going to read a 0. So, you will add the blue symbol corresponding to 0 you are adding B corresponding to 1 you are adding G.

So, that I will be q 1 B G, but in q 1 the top plate, top symbol is G and you see a 1. That means the last symbol you have red is G and next symbol you are going to see is also last symbol you have seen is a 1. Then next symbol you have seen you are going to see is also a 1 is not it. So, the next one may be the continuation of w R, the beginning of w R. So, both possibilities you must consider.

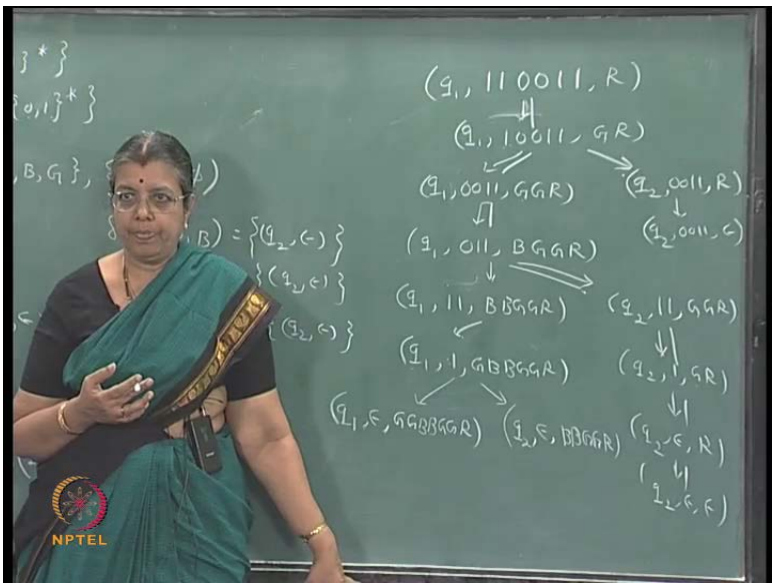
(Refer Slide Time: 07:43)



So, you could have q 1 G G or q 2 epsilon. Both possibilities you have to consider. Then in q 2 if you have a 0 and a B combination you remove that and in q 2 if you have a 1 and G combination you remove that. In q 2 if you get a red symbol without waiting for input you remove it q 2 epsilon R is q 2 epsilon. This is very similar to what we have

written in the last class, but in two of the cases you are allowing for two possibilities.

(Refer Slide Time: 08:41)



Now, let us see how it was on an input 1 1 0 0 1 1? So, consider the input 1 1 0 0 1 1, this is the input. It has to be accepted. When it accepts the push down store has to be empty. We are considering acceptance by empty store now. Acceptance by binary state is also allowed and we will show the equivalence. Generally push down automata you consider acceptance by empty store. The equivalence to context free grammar is shown very easily using acceptance by empty store.

Now, what is the initial ID? q 1 is the initial state and you are having R as a initial push down symbol. Now, when you read a 1 what happens? q 1 this 1 has been red, next ID will be this is the portion to be red and you are going to add a G. Now, in q 1 top you are going to read a 1 and the top symbol is G. q 1 1 G, q 1 1 G there are two possibilities either you stay in q 1 and add a green symbol or you go to q 2 and remove the green symbol.

So, there are two possibilities one is q 1, 1 has been red. This is the portion remaining and you can add a G symbol. The other possibility is you go to q 2 read a 1, but this is a remaining remove the G symbol. So, you get R. When you get a q 2 and R combination, when you are in state q 2 and see a R symbol without looking into the input you remove the R symbol. So, this will go to q 2 0 0 1 1 epsilon. Now, once the stack is emptied further move is not possible. See if any move has to take place then there should be at

least one symbol on the push down store. Only if there is at least one push down symbol a move can be defined. If there are no symbols it cannot be defined.

So, in this case you cannot proceed further, but at same time the whole of the input has not been red. The string is accepted if and only if the whole input is red. If you stop in the middle it has to be rejected. So, continuing in this you are having a G and a 0, when you see a 0 you have to add a blue symbol.

So, q 1 0 has been red. This is the portion remaining you are having adding a blue symbol on the stack. Please note that we are following the convention that the left most symbol is the top of the stack. Now, in q 1 you are seeing a 0 and the top plate is blue or top symbol is B. There are again two possibilities it can be the continuation of w R, the beginning of w R.

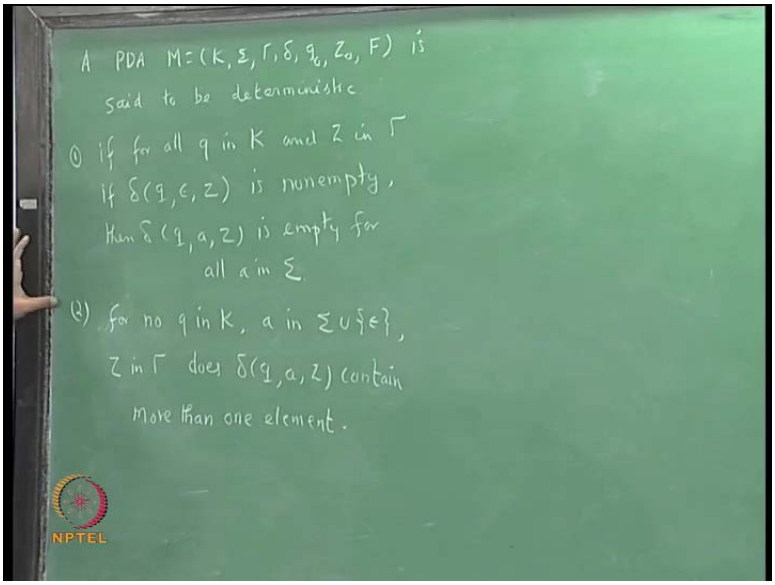
So, you can remain in q 1 read the 0 add a blue symbol. This is possible or you can go to q 2 B has to be removed 0 and B combination you remove the blue symbol. So, portion of which is remaining is 1 1. On the input you have to still read 1 1 and you have G G R. Once you go to q 2, if there is a 0 B combination you remove the B, 1 G combination you remove the G. If there is another combination G 1 b or 0 G you have to halt. Now, here you find that you have a 1 G combination, remove it, again 1 G combination remove it. And in q 2 when you have a R on the top without looking for the input you can remove it.

So you go to q 2 epsilon epsilon and this is the accepting configuration for empty store. Now, in this case what happens when you have a 1 in the B you have to add G. So, next thing will be q 1 1 G B B G G R now, you are in q 1 and you are getting a 1 and top symbol is G. So, again it is can be the beginning of w R or the continuation of w.

So one possibility is, you go to q 1 read this symbol add 1 G (No Audio Time: 14:26 to 14:32) another thing is you go to q 2 this has been red and you will remove the G for one so end up with B B G G R. We cannot proceed further here because in q 2 and B no symbol is there you have red the whole input in this case. But the stack has not been emptied push down stores still contain something. So, this is not accepting configuration. Here, you are again, you are in state q 1 whole input has been red but something is there on the stack so this is also not an accepting configuration. The accepting configuration is this.

So, how do you go to the accepting configuration? Make this move, this move, this, this, this, this there may be several possibilities that is why it is nondeterministic. After which if one sequence takes you to an accepting ID, accepting configuration the string will be accepted. The string will not be accepted if there are no sequence of more which leads to a accepting ID. None of it is accepting then only it is string will be rejected. Here one sequence leads you to an accepting ID, other sequences may lead you to non accepting IDs does not matter. If there is at least one sequence which takes you to a accepting ID, the string will be accepted, other ways it will not be accepted.

(Refer Slide Time: 16:27)



So, this is the difference between deterministic automata and deterministic PDA, and non deterministic PDA. The definition of deterministic PDA is like this, a PDA M is equal to K sigma gamma delta q naught z naught F if it is acceptance by empty store it is empty, is said to be deterministic. (No Audio Time: 16:52 to 16:58) There are two conditions, if for all q in K and z in gamma; if delta of q epsilon z is non empty then delta of q a z is empty for all a in sigma. What does that mean?

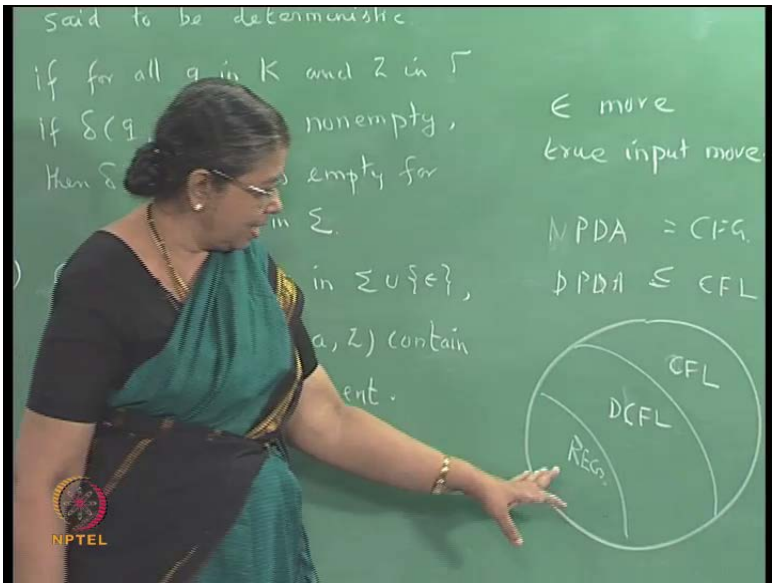
This is the first condition I will write down the second condition also then we will see what it is. If for no q in k, a in sigma union epsilon, z in gamma, does delta of q a z contain more than one element. What does that mean? The second is easy, the next unique that is, what it means in deterministic case? The next will be unique.

What is the first one mean? For all q in K and z in gamma if delta of q epsilon z is non

empty, then delta of q a z is empty for all a in sigma. That means, that for a particular state and push down symbol combination q z combination either you use epsilon move when you do not read any input symbol that is called an epsilon move either you use an epsilon move or you use a true input move, you read a symbol.

For a particular q and z combination both you do not allow for a deterministic question. Look at this one this is because of this it is nondeterministic. But here you see the earlier example of w c w R also we had this move. When you are getting a q 2 and R combination, when you are in state q 2 and the top symbol is R then you remove it. When you are in state q 2 and R no move is defined for a or b, I mean 0 or 1.

(Refer Slide Time: 20:59)



So, for this combination q 2 R combination only epsilon move is allowed. Similarly, for some q and z combination, if you allow for epsilon move, you do not allow for true input move. You call them as; generally you call them as epsilon move, true input move. For a deterministic push down automata for a particular state and a push down symbol combination either you allow for epsilon move or you allow for true input move, both you do not allow, that is the first condition.

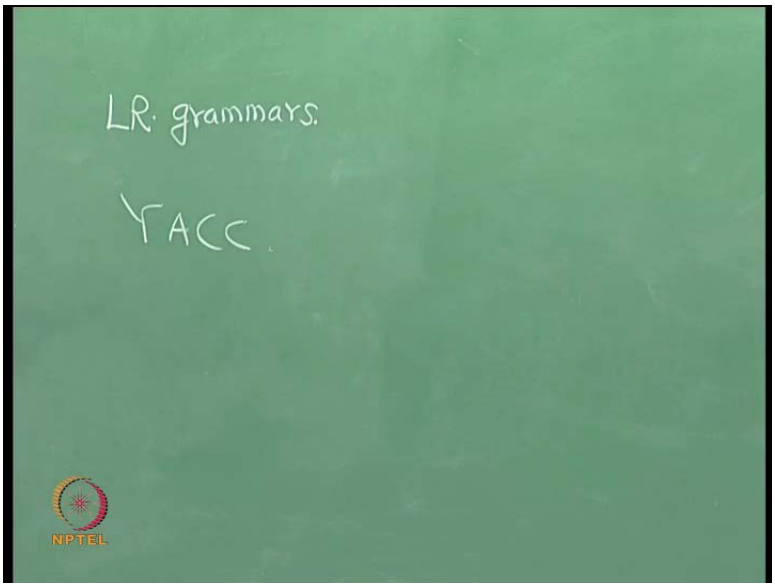
The second condition is when you say delta of q a z is p gamma or something like that, that is unique, at most you can have only one possibility. It is not necessary, that you should define for every each and every combination. Like in the finite state automata you draw a state table everything you fill and whenever there is no possibility you put a dead

state. See you make the move machine move go to the dead state and once it goes to dead state it remains there. Here we do not specify like that, sometimes delta q a z need not be specified at all.

But, whenever you specify that, either a can be epsilon or a true input. Whenever you specify it has got only one ordered pack p gamma, you cannot have p 1 gamma 1, p 2 gamma 2 like that. So, in the first example, when we consider for w c w R, we never had two possibilities, only one possibility was there. Here in this one there are two possibilities, that is why it is non deterministic. Here this first condition is not violated but the second condition is violated, you have two choices here. In some cases when you allow for both a true input to be red or an epsilon to be red with the for the same push down symbol and the state combination it becomes non deterministic.

So, non deterministic push down automata are equivalent to you do not say NPDA you say generally PDA, PDA means non deterministic. This is equivalent to CFG and d PDA form a subclass it is just contained in. So, if you draw the class like this regular sets will be like this, CFL includes regular set, context free languages this is the family of deterministic context free languages. What is DCFL? Languages accepted by deterministic push down automata, they form a sub class. One example is this w w R inherently ambiguous context free languages cannot be accepted by DPDA, you have to have nondeterministic in there and so on.

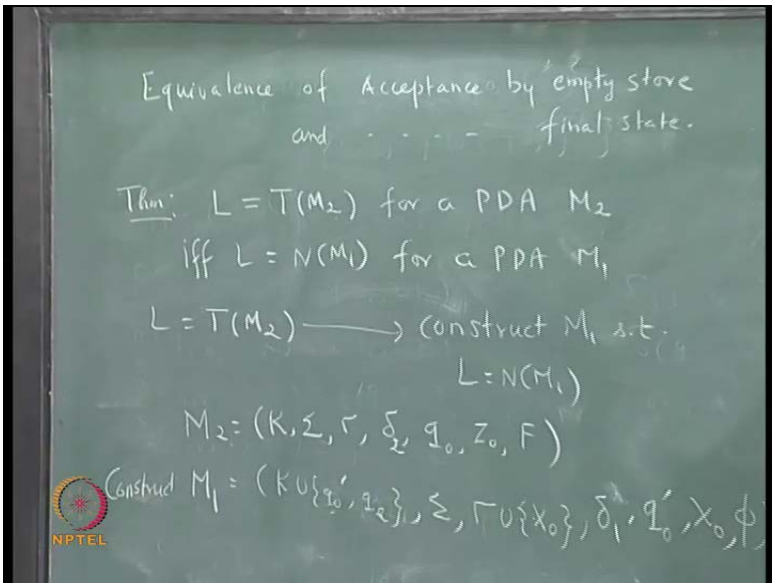
(Refer Slide Time: 24:28)



What is the grammar corresponding to this, this is finite state automata; this is PDA correspondence to context free grammar. For deterministic push down automata the corresponding grammars, LR grammars, these are very useful in compiles you have LR one passing table, LR two passing table and so on. And there is an automatic generator yak given a grammar it will try to generate the parcel automatically.

So, those concepts are used if there is enough time we will consider in detailed about LR 0 tables, passing tables, how do we construct the tables and so on. Depending upon the time we have, we will have that. The next thing, we have to consider is the equivalence of acceptance by empty store and acceptance by final state.

(Refer Slide Time: 25:42)



(No Audio Time: 25:28 to 25:42) Equivalence of acceptance by empty store and acceptance by (No Audio Time: 26:03 to 26:15) so we have a result. L is a context free language; L is equal to T of M 2 for a PDA M 2. T of M 2 means acceptance by final state, we use the symbol N for acceptance by empty store. If and only if L is equal to N of M 1 for some PDA, for a PDA M 1, this is the result we want to prove. That is if there is a language which is accepted by a push down automata by final state, then there is a another push down automata which will accept the same language by empty store and vice versa.

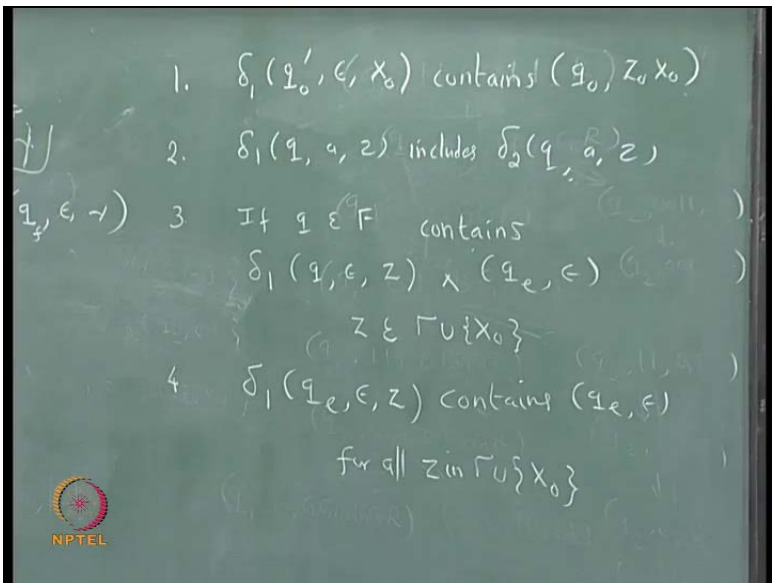
So, the first part of the proof is L is equal to T of M 2, then you have to construct M 1 such that L is equal to N of M 1 given M 2 you have to construct M 1. Now, how do you

do this? M 2 is say K sigma, gamma, delta, q naught, z naught, F and M 2 accepts L by final states set of final states are given by F.

Now, you construct M 1, construct you add two more states to this q naught dash and q e, what does q naught dash mean? I mean this is new as a state q e means erasing state; we want it to have it a erasing state, so we call it as q e. Same sigma, gamma union a new push down symbol x naught which is not already there in gamma. Add a new push down symbol and mapping is delta dash or if you want to have del, you call this as delta 2, call this as delta 1, that is more easy. The new state which we have added q naught dash is the initial state here, and the new push down symbol which you have added is the initial push down symbol. And because it is acceptance by empty store you need not have to right the final states.

So, what we do is we want to construct m one which will accept the same language by empty store. And for that what we do is? The set of states you add two more states q naught dash, and q e, and this q naught dash is a new initial state. The symbol which you have added new push down symbol, you add to gamma and that is the initial push down symbol and so on.

(Refer Slide Time: 30:14)

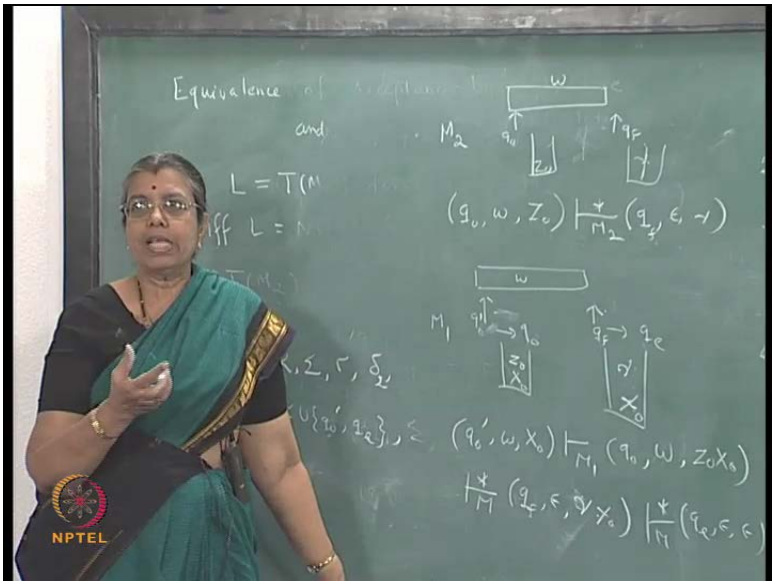


Now we have to define delta 1. How do you define delta 1? (No Audio Time: 30:01 to 30:13) delta 1, q naught dash epsilon X naught contains q naught z naught X naught, the first move is like this. Then delta 1 q a z whatever you have I mean whatever delta 2 is

equal to whatever mapping you have there you have here also, includes this whatever mappings are there in delta 2 all those are in delta 1 also.

Then if q belongs to F, you have final state, then you have delta 1 q epsilon z is equal to q e epsilon where z belongs to gamma. Gamma union X naught (( )) 4 delta 1 q e epsilon z, it should not write a equal to because rather contains is the proper word, contains q e epsilon for all z in gamma union X naught.

(Refer Slide Time: 32:55)



(No Audio Time: 32:21 to 32:37)

Now, how M 1 simulates M 2, let us see how M 1 simulates M 2, how will M 2 accept a string? You have input you have a push down store initially; initially it will be in q naught and have z naught on the stack. And when it accepts this it goes to a final state q f some final state something may be there on the stack, that is how this input is accepted by M 2, M 2 accepts in this manner. What does M 1 do? M 1, I has to accept the same string.

So, this one if you write in the formal ID way q naught say some w is accepted this is w say, w z naught in M 2 this M 2 means in a the move is M 2, you go to some q to denoted as a final state I will write as q f epsilon some gamma. Now, what does M 1 do here? M 1 it starts with this w and the initial state q naught dash and with X naught as the bottom most push down symbol, this is a situation with which it starts. The first moves

raise that delta 1, q naught dash epsilon X, naught contains q naught, z naught, X naught.

So, without reading any input, symbol epsilon move, without reading anything the state changes to, it changes to, q naught from q naught dash it changes to q naught and it writes a z naught on top of X naught. Now, you have almost come to this situation is not it except that there is a X naught below the z naught it is exactly similar to this. Then M 1 will simulate the behaviour of M 2 they behave just like M 2 that is what is given by delta 1 q a z includes delta 2 q a z, so it will have all the moves.

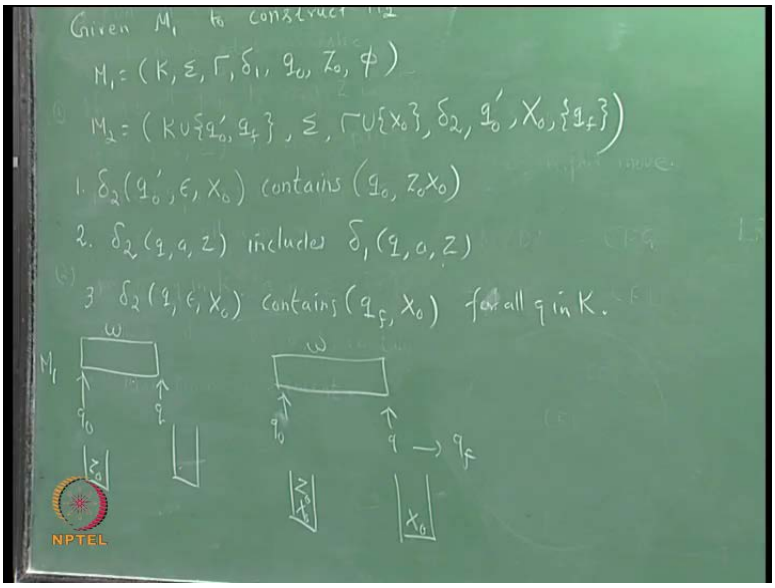
So, it will read the entire thing and at the end what happens, it goes to a final state q f. And start in this case it had a string gamma here it will have gamma below that X naught. Now, afterwards once you reach a final state and the whole input has been read. What it is going to do is? The third move says that you go to the erasing state. So, from q f it will go to the erasing state and start erasing one by one all these symbols, it will erase all the symbols. So, you get acceptance by empty store.

If you write in this way starting from q naught dash w X naught, M 1 will make one move, it will go to q naught, it is an epsilon move. So, no input has been read z naught X naught, then by making more moves, it will go to some q f whole input will be read, this portion is exactly similar to this gamma. But below that you have the X naught and afterwards it starts erasing and you get q e epsilon.

So, if a string is accepted by M 2, it will be accepted by M 1. If it is accepted by M 1, it should be accepted by M 2 how is that possible? See look at this the only move where the start state is used is this, you have to start with this, and once you go to the erasing state it is only epsilon, you cannot read any input, you have to keep on erasing that is all. Once you go to q e the whole stack has to be erased you cannot do anything else.

So, you have to use this in the end, you have to use this in the beginning and in between you have to do this, that is the only way you can proceed. So, in between when you do this you are simulating M 2 is not it. So, if it is accepted by M 1 that means the first move you make like this and the end you erase the start symbols so, in between you would have gone from this to this, that means it is accepted by M 2. So, the language accepted by M 1 and M 2 are the same. M 2 is accepting it by final state, M 1 is accepting by empty state.

(Refer Slide Time: 38:34)



Now, the second part of it given M 2 to construct M 1. So, M 1 is given as K, sigma, delta, a sigma, gamma, gamma, delta 1, q naught, z naught, phi. It is acceptance by empty store, it is given like this. So, M 2 to get M 2 what you do is? To the set of states you add two more states q naught and q f, q naught dash and q f, two more states you add, sigma is a same, you add one more push down symbol X naught, and the mapping is given by delta 2, the new state q naught dash is the initial state, X naught is the initial pushdown symbol, and the only finite state is q f.

So, as in the earlier case, we add two states q naught dash which will act as a new initial state, q f which will act as the only final state. And you push down symbol, you add one more which will act as the initial push down symbol and the only final state is q f, the mapping we have right now delta 2. Delta 2 q naught dash epsilon X naught contains q naught, z naught, X naught as the earlier case you can understand that this move is from the initial ID of M 1 you go to, the initial ID of M 2 you go to, the initial ID of M 1 that is why this mapping is written.

Second delta 2 q, a, z includes delta 1 q, a, z, for all q, for all a, for z, etcetera whatever mapping is there in M 1 all those mappings are there in M 2 also. Then delta 2 q epsilon X naught contains q f X naught. (No Audio Time: 41:43 to 41:53) For all (No Audio Time: 41:58 to 42:08) a the idea is similar how does M 1 accept a string w, you have a string w how is this accepted by M 1 start with q naught with z naught on the stack after

reading this you go to some state q and the stack is empty. Now, how will M 2 accept the same w, you have w, you are in the initial state q naught, dash and you are having X naught. You first move says, q naught dash epsilon X naught contains q naught z naught X naught. So, make one epsilon move that is no do not read any symbol from q naught, you go to q naught dash, you go to q naught, and write z naught over the stack.

Now, you have come to the initial configuration of M 1, simulate M 1 at the end you go to some state q. And the stack is emptied but the bottom most symbol X naught remains. It is stack is empty up to z naught you are marking the stack with X naught now. So, X naught will remain so finally, when you end up with just X naught and the stack that means the whole stack has been emptied in the case of M 1. So, you make one more move go to the straight q f. This is what it means when you just have X naught alone on the stack from any state you go to the final state. So, in this way you can simulate the behaviour of M 1 by M 2 the languages accepted are the same.

So, if a string is accept by M 1, it will be accepted by M 2 and if a string is that you can very easily see. If a string is accepted by M 2, the way you proceed is first you have to go to the initial, this move can be used only in the beginning and this move can be used only in the end. Once you go to q f there is no move defined for q f, after once you go to q f, you halt you stop there is no move with delta q f etcetera.

So, you have to use this move in the beginning and you have to use this move in the end, in between you have to use the other moves. So, that means from here to here, first you have to use, then you have to use this, then you have to use this. That means when you use this you are really simulating M 1 so, if a string is accepted by M 2 it is accepted by making this move then some moves here and then this move. That means it makes all these moves so, it will be accepted by M 1. So, if a string is accepted by M 2 it is accepted by M 2 and vice versa so they all are same.

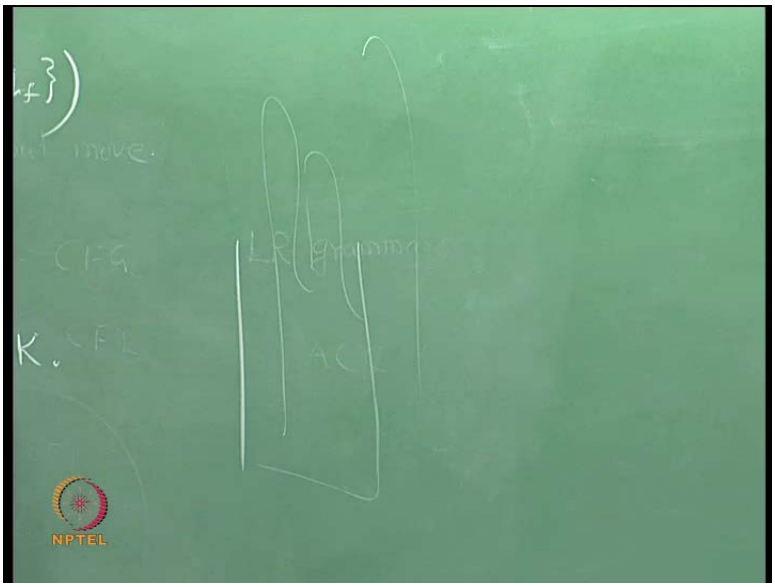
So, acceptance by final state and acceptance by empty store are equivalent. But please remember that for a same machine it is not like that; see you may define a push down automaton by final state. Then in between it may for reading when it reads a string it may empty the stack so that string is not accepted by the machine. It is not true that N of M is equal to L of M, T of M for same machine. If some language L is accepted by one push down automaton M 1 by empty store, it can be accepted by another push down

automata by final state and so on. But for the same push down automata suppose you define a finite state then acceptance by empty store in final state need not be the same they will be different. You when you define acceptance where empty store you do not specify any final state you say 5.

And the way why we are having this X naught in the beginning, when we want to construct M 1 from M 2, we are having a new push down symbol. If you do not have that push down symbol, what may happen is? While trying to simulate, it may empty the stack and accept something else, which you do not want to accept such a thing may happens.

So, this is about equivalence between empty store and final state. Usually, we take only acceptance by empty store in the case of push down automata that is very convenient actually for proving results and all for especially for proving the result with context free grammars, equivalence with context free grammars, it is very convenient to take acceptance by empty store.

(Refer Slide Time: 48:05)



And there are so many results about making, the push down store you have a push down store this push down store can grow come back and so on, when it grows and comes back, which is called one turn. So, how many turns does it make? Such things even today there are papers coming on such things, turn bounded push down automata, when how many turns it makes and so on.

If it makes just one turn it will accept what are known as linear languages. What are linear languages? a power n, b power n, w w R, w c w R, the just stack grows and comes back. If it just makes one turn the language accepted will be a linear language. So, even today, research papers are being published on this turn push down automata. How many turns and when it comes back should it erase the bottom most symbol or just keep the symbol or go to the initial state and things like that. So, next we will consider the equivalence of with the grammars in the next class.