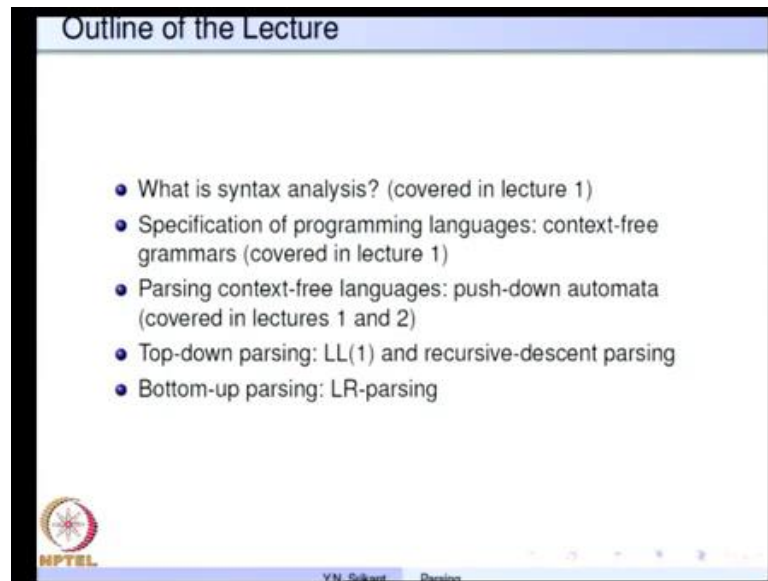


Principles of Compiler Design
Prof. Y. N. Srikant
Department of Computer Science and Automation
Indian Institute of Science, Bangalore

Lecture - 7

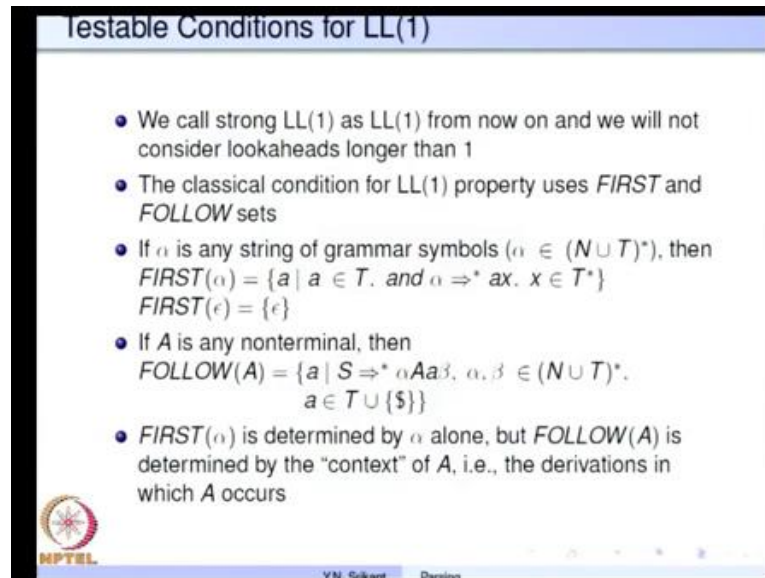
Syntax Analysis: Context-free Grammars, Pushdown Automata and Parsing Part-3

(Refer Slide Time: 00:23)



Welcome to the lecture on Syntax Analysis part 3. So far we covered the principles of syntax analysis context free grammars and pushdown automata, and we also saw some definitions regarding LL (1). Today, we will continue with LL (1) parsing go on to cover recursive descent parsing as well.

(Refer Slide Time: 00:47)



Testable Conditions for LL(1)

- We call strong LL(1) as LL(1) from now on and we will not consider lookaheads longer than 1
- The classical condition for LL(1) property uses *FIRST* and *FOLLOW* sets
- If α is any string of grammar symbols ($\alpha \in (N \cup T)^*$), then
 $FIRST(\alpha) = \{a \mid a \in T, \text{ and } \alpha \Rightarrow^* aX, X \in T^*\}$
 $FIRST(\epsilon) = \{\epsilon\}$
- If A is any nonterminal, then
 $FOLLOW(A) = \{a \mid S \Rightarrow^* \alpha A a \beta, \alpha, \beta \in (N \cup T)^*, a \in T \cup \{\$\}\}$
- $FIRST(\alpha)$ is determined by α alone, but $FOLLOW(A)$ is determined by the "context" of A , i.e., the derivations in which A occurs

MPTEL
Y.N. Saha et al. Design

Some recap on LL (1), so we just use LL (1) instead of a strong LL (1), because strong LL (1) and LL (1) are equivalent as far as look ahead one is concerned. And since it is not easy to implement look aheads longer than one, we will not worry about LL (2), LL (3) and so on. We also defined first and flow sets last time let us do a recap on the definitions.

The first of alpha is a set consisting of the symbols A, which can be derived from alpha, so $A \in T$ and alpha derives a X and X is also a term string. First of Epsilon by definition is Epsilon and the follow is although symbols here, which actually are after the non terminal A in any sentential form. So, that is why it is defined as although symbols A such that S derives alpha A, A beta, thereby signifying that A occurs in the sentential form.

And we collect all these symbols little a, which or after this non terminal A, A can be either A terminal symbol or the end of file symbol dollar. So, what we need to need to you know understand is that first of all first determine by alpha alone, whereas follow is determined by the context, that is we require that S derived a sentential form containing A.

(Refer Slide Time: 02:31)

FIRST and FOLLOW Computation Example

- Consider the following grammar
 $S' \rightarrow S\$$, $S \rightarrow aAS \mid c$, $A \rightarrow ba \mid SB$, $B \rightarrow bA \mid S$
- $FIRST(S') = FIRST(S) = \{a, c\}$ because
 $S' \Rightarrow S\$ \Rightarrow c\$$, and $S' \Rightarrow S\$ \Rightarrow aAS\$ \Rightarrow abaS\$ \Rightarrow abac\$$
- $FIRST(A) = \{a, b, c\}$ because
 $A \Rightarrow ba$, and $A \Rightarrow SB$, and therefore all symbols in $FIRST(S)$ are in $FIRST(A)$
- $FOLLOW(S) = \{a, b, c, \$\}$ because
 $S' \Rightarrow S\$$,
 $S' \Rightarrow^* aAS\$ \Rightarrow aSBS\$ \Rightarrow aSbAS\$$,
 $S' \Rightarrow^* aSBS\$ \Rightarrow aSSS\$ \Rightarrow aSaASS\$$,
 $S' \Rightarrow^* aSSS\$ \Rightarrow aScS\$$
- $FOLLOW(A) = \{a, c\}$ because
 $S' \Rightarrow^* aAS\$ \Rightarrow aAaAS\$$,
 $S' \Rightarrow^* aAS\$ \Rightarrow aAc$

MPTEL
Y.N. Saha, Design

Here is an example that I presented last time, so for this grammar S' going to S dollar, S going to aAS or c , A going to ba or SB , B going to bA or S , the first and follow are computed here. We begin you know very simply with first of S' which is nothing but the first of S , because all the strings from S' actually are derived by S , and what does S derive all strings you know derived by S begin with either A or c , so first of S would be a comma c .

Similarly, for first of A we have all the strings derivable from A , you know that they begin with little b and because there is a non terminal S here, all the strings which are derivable by S or also include in the first of A ; and for B , so that gives us you know the first of A as a , b and the c . So, now as far as B is concerned the symbols b and the first of S are included in the first of B , so that is how these are all computed.

Now, what about the follow, follow of S , it says is a, b, c dollar, so let us see where S occurs, so S' derives S dollar, so this is the first sentential form where S occurs and dollars follows it, so this dollar is because of that. And S' derives S dollar and if we replace S by aAS dollar, then you know finally, A can be replaced by SB and now B is a non terminal, so let us make it derive some string all these strings will follow this non terminal S .

So, if you apply B going to bA little b follows S , so that b is included in the follow of S and then we have in a similar way, this sentential form aS aAS where little a follows

S, so A also gets into the follow set of S. Finally we derive a S c S dollar from S prime, where the little c follows S, so c is also included in the follow set of S, follow of A in a similar way is a comma c, because S prime derives a A a A S dollar, where little a follows capital A and in the other one S prime derives a A c where little c follows capital A, so these two are in the follow of A.

(Refer Slide Time: 05:42)

Computation of *FIRST*: Terminals and Nonterminals

```

{
  for each (a ∈ T) FIRST(a) = {a}; FIRST(ε) = {ε};
  for each (A ∈ N) FIRST(A) = ∅;
  while (FIRST sets are still changing) {
    for each production p {
      Let p be the production A → X1X2...Xn;
      FIRST(A) = FIRST(A) ∪∅ (FIRST(X1) - {ε});
      i = 1;
      while (ε ∈ FIRST(Xi) && i ≤ n - 1) {
        FIRST(A) = FIRST(A) ∪ (FIRST(Xi+1) - {ε}); i++;
      }
      if (i == n) && (ε ∈ FIRST(Xn))
        FIRST(A) = FIRST(A) ∪ {ε}
    }
  }
}

```

So, this is as far as the example is concerned, to compute the first for terminals and non terminals it is fairly straight forward, terminal symbols the first set does not change, it is the symbol itself. And first of Epsilon is always Epsilon for a non terminals we begin with phi. And then we consider and then we need to iterate the computation this entire thing, until even every first set stops changing.

So, even if one of them is changing then we need to iterate once more, this is because the first set computation, you know even if one of them changes it may actually make a few others changes as well. So, then the computation is quite straight forward, we look at the first you know production A going to X 1, X 2 etcetera X n, now first of A obviously, includes the first of X 1, so first of A equal to first of A union, first of X 1 minus Epsilon.

Now, first of X 1 produces Epsilon in that case whatever is derived by X 2 is also derived by A, so that is what is being checked here is Epsilon in first of X i, so if so include the first of X 2 also into first of A. Similarly, if there is X 3 and X 1 and X 2 both

of them produce Epsilon then we include you know the first of X 3 also into the first of A.

So, this keeps going and if X 1, X 2 etcetera X n all of them, first of all these symbols contain Epsilon, then Epsilon itself has to be included in the first of A. So, if we reach I equal to n that is all the symbols have been looked at that means, all of them have produced Epsilon and Epsilon is in the first of X n, so that is the last one we include Epsilon into the first set of A.

(Refer Slide Time: 07:56)

```
Computation of  $FIRST(\beta)$ :  $\beta$ , a string of Grammar Symbols

/* It is assumed that FIRST sets of terminals and nonterminals
are already available */
FIRST( $\beta$ ) =  $\emptyset$ ;
while (FIRST sets are still changing) {
  Let  $\beta$  be the string  $X_1X_2...X_n$ ;
  FIRST( $\beta$ ) = FIRST( $\beta$ )  $\cup$  (FIRST( $X_1$ ) -  $\{\epsilon\}$ );
   $i = 1$ ;
  while ( $\epsilon \in FIRST(X_i)$  &&  $i \leq n - 1$ ) {
    FIRST( $\beta$ ) = FIRST( $\beta$ )  $\cup$  (FIRST( $X_{i+1}$ ) -  $\{\epsilon\}$ );  $i++$ ;
  }
  if ( $i == n$ ) && ( $\epsilon \in FIRST(X_n)$ )
    FIRST( $\beta$ ) = FIRST( $\beta$ )  $\cup$   $\{\epsilon\}$ 
}
```

So, now suppose we have a string beta for which we want to compute the first set, it is not different from computing the first set of non terminal, if the string is certain beta is X 1 to X n, we do exactly what we did before for the non terminals, except that this algorithm has placed beta in place of instead of A in all designs. So, we consider X 1, X 2 etcetera X n, so first of X 1 is obviously in first of beta, then if X n, we can produce Epsilon that is Epsilon in first of X 1.

(Refer Slide Time: 08:52)

FIRST Computation: Algorithm Trace - 1

- Consider the following grammar
 $S' \rightarrow S\$, S \rightarrow aAS \mid \epsilon, A \rightarrow ba \mid SB, B \rightarrow cA \mid S$
- Initially, $FIRST(S) = FIRST(A) = FIRST(B) = \emptyset$
- Iteration 1
 - $FIRST(S) = \{a, \epsilon\}$ from the productions $S \rightarrow aAS \mid \epsilon$
 - $FIRST(A) = \{b\} \cup FIRST(S) - \{\epsilon\} \cup FIRST(B) - \{\epsilon\} = \{b, a\}$
from the productions $A \rightarrow ba \mid SB$
(since $\epsilon \in FIRST(S)$, $FIRST(B)$ is also included;
since $FIRST(B) = \emptyset$, ϵ is not included)
 - $FIRST(B) = \{c\} \cup FIRST(S) - \{\epsilon\} \cup \{\epsilon\} = \{c, a, \epsilon\}$
from the productions $B \rightarrow cA \mid S$
(ϵ is included because $\epsilon \in FIRST(S)$)

MPTCL
Y.N. Subramanian, Professor

Then first of X 2 goes into first of beta and so on and so forth, exactly the way it was before. Now, let us look at an example to understand how this algorithm works, so S prime going to S dollar, S going to a A S or Epsilon. So, this is the difference between this grammar and the previous one the other two productions are the same A going to b a or SB dot B going to c A or S. So, to begin with we initialized the first sets of the non terminals to phi the null set and iteration one first of S is a comma Epsilon, because it derives A and there is S going to Epsilon as well.

So, that includes Epsilon into the first set of S, now the first set of A, here is b, so which begins the string b a here and then we have the then we have the non terminals. So, b union first of S, of course minus Epsilon and union the first of b minus Epsilon why S you know derives Epsilon. So, first of S we see here has produced and Epsilon, so therefore, both the first of S and first of V have to be included in A and what about B itself, B produces c A and S now the c is obviously non null, but B produces S and first of S contains Epsilon, so finally A going to SB will also produce Epsilon.

And therefore, we need to you know include it, but not immediately at this point of time we do not know that first of B is phi, from the analysis you know if we apply B going to c A and B going to S, then we know that you know B is a first of B includes first of S and therefore, it includes Epsilon, but at this point in iteration one first of B has been

initialized to phi and therefore, we cannot include Epsilon into the first set of A at this point of time, it will included in the next iteration.

(Refer Slide Time: 11:51)

FIRST Computation: Algorithm Trace - 2

- The grammar is
 $S' \rightarrow S\$$, $S \rightarrow aAS \mid \epsilon$, $A \rightarrow ba \mid SB$, $B \rightarrow cA \mid S$
- From the first iteration,
 $FIRST(S) = \{a, \epsilon\}$, $FIRST(A) = \{b, a\}$, $FIRST(B) = \{c, a, \epsilon\}$
- Iteration 2
 (values stabilize and do not change in iteration 3)
 - $FIRST(S) = \{a, \epsilon\}$ (no change from iteration 1)
 - $FIRST(A) = \{b\} \cup FIRST(S) - \{\epsilon\} \cup FIRST(B) - \{\epsilon\}$
 $= \{b, a, c, \epsilon\}$ (changed!)
 - $FIRST(B) = \{c, a, \epsilon\}$ (no change from iteration 1)

MPTEL
Y.N. Srikant - Director

First of B is c, so at the now we know what first of B will be it will be different, it is c of course and the first of S, first of S actually includes an Epsilon and therefore, you know we include Epsilon into the first of B, so c A Epsilon is the first set of B. So, this is as far as the first iteration is concerned, in iteration 2, now first of S has A Epsilon, first of A has b a, first of B has c a Epsilon. First of S does not change it is just A Epsilon, but first of a changes.

So, from here we see the same b union first of S and then since S includes an Epsilon first of S includes an Epsilon first of B as well and since first of B also includes Epsilon, we need to include Epsilon as well. So, B union first of S minus Epsilon, union first of B minus Epsilon and then we have to include Epsilon, because first of you know both S and B derive Epsilon. So, that means, this becomes b a c epsilon which is different, first of B is c a Epsilon you know has no has not changed from the last iteration. So, these are the values which stabilize and do not change in iteration 3. So, these are the first values.

(Refer Slide Time: 13:02)

```
Computation of FOLLOW

{ for each  $(X \in N \cup T)$  FOLLOW( $X$ ) =  $\emptyset$ ;
  FOLLOW( $S$ ) =  $\{S\}$ ; /*  $S$  is the start symbol of the grammar */
  repeat {
    for each production  $A \rightarrow X_1X_2...X_n$  { /*  $X_i \neq \epsilon$  */
      FOLLOW( $X_n$ ) = FOLLOW( $X_n$ )  $\cup$  FOLLOW( $A$ );
      REST = FOLLOW( $A$ );
      for  $i = n$  downto 2 {
        if ( $\epsilon \in \text{FIRST}(X_i)$ ) { FOLLOW( $X_{i-1}$ ) =
          FOLLOW( $X_{i-1}$ )  $\cup$  (FIRST( $X_i$ ) -  $\{\epsilon\}$ )  $\cup$  REST;
          REST = FOLLOW( $X_{i-1}$ );
        } else { FOLLOW( $X_{i-1}$ ) = FOLLOW( $X_{i-1}$ )  $\cup$  FIRST( $X_i$ );
          REST = FOLLOW( $X_{i-1}$ ); }
      }
    }
  } until no FOLLOW set has changed
```

Now, let us look at the algorithm for computing follow of a non terminal, what we saw was a definition before, but now let us see how its computed, initialize the follow set of every non terminal to phi and follow of S is initialized to dollar, because S is the start symbol, then we have S prime going to S dollars. So, dollar will be always present in follow of S, so we do this for every production, until no follow set has changed.

Exactly, the way we computed the first set, let the production be $X_1 A$ going to X_1, X_2, X_n , so the assumption is not all these symbols are Epsilons, so in other words it is not a production a going to Epsilon follow of X_n is so we are now we cannot compute the follow of A. We need to actually compute the follow of X_n , you know and then the other symbols as we go along, but to compute the follow of a we need to look at the production in which the right hand side contains an A.

Until, that time we cannot compute the follow of A, because follow of a requires A context derivable from these start symbol. So, for the non terminal X_n it is easy to see that in a sentential form, when we replace here by the right hand side X_1 to X_n , the symbols which was following a will also follow X_n . Therefore, follow of X_n is follow of X_n union or whatever, is the previous value from the previous iterations union follow of A.

So, now let us call rest as follow of A, we will see why it is requires, now we are going to look at the symbols X_1 to X_n in the reverse order from X_n downwards to X_1 , so n

down to 2, so in such a case after X_n for which we have computed the follow in the first iteration. We go to X_{n-1} , X_{n-2} etcetera, so let us take just two symbols that is n equal to two, let us say we have computed the follow of X_2 .

Now, we are looking at the follow of X_1 follow of X_1 contains all the symbols in the first of X_2 , that is obvious because X_2 derives a certain number of things and those are all following X_1 , but suppose X_2 also produces Epsilon. In such a case, when we replace a by X_1 , X_2 and make X_2 derive the Epsilon, all the symbols which follow A in a sentential form will also follow X_1 in later derivations.

Therefore the follow of A which is nothing but you know the symbols which follow A will also be included in the follow of X_1 , so here is the generalization for the X a production A going to X_1 to X_n . So, it says for i equal to n down to 2, if Epsilon is in the first of X_i , so that is i is n to begin with so for first of X_n , so if this is going to produce an Epsilon. Then follow of the symbol which is prior to this, that is X_{n-1} X_{i-1} will be follow of X_{i-1} union the first of X_i obviously, this particular symbol first of X_n will be included in the follow of X_{n-1} .

Union, the rest so in the first instance rest will be follow A , but later it will become different, now rest is again made follow of X_{i-1} , then we go to the next symbol and so on, and so forth. So, suppose Epsilon was not in X_n at all the first of X_n , then you know follow of a will never be following the symbols of X_{n-1} for as the symbol X_{n-1} . So, we do not have to worry about the Epsilon part, we simply say follow of X_{i-1} is follow X_{i-1} .

(Refer Slide Time: 18:05)

FOLLOW Computation: Algorithm Trace

- Consider the following grammar
 $S' \rightarrow S\$$, $S \rightarrow aAS \mid \epsilon$, $A \rightarrow ba \mid SB$, $B \rightarrow cA \mid S$
- Initially, $follow(S) = \{\$\}$; $follow(A) = follow(B) = \emptyset$
 $first(S) = \{a, \epsilon\}$; $first(A) = \{a, b, c, \epsilon\}$; $first(B) = \{a, c, \epsilon\}$
- Iteration 1 /* In the following, $x \cup y$ means $x = x \cup y$ */
 - $S \rightarrow aAS$: $follow(S) \cup = \{\$\}$; $rest = follow(S) = \{\$\}$
 $follow(A) \cup = (first(S) - \{\epsilon\}) \cup rest = \{a, \$\}$
 - $A \rightarrow SB$: $follow(B) \cup = follow(A) = \{a, \$\}$
 $rest = follow(A) = \{a, \$\}$
 $follow(S) \cup = (first(B) - \{\epsilon\}) \cup rest = \{a, c, \$\}$
 - $B \rightarrow cA$: $follow(A) \cup = follow(B) = \{a, \$\}$
 - $B \rightarrow S$: $follow(S) \cup = follow(B) = \{a, c, \$\}$
 - At the end of iteration 1
 $follow(S) = \{a, c, \$\}$; $follow(A) = follow(B) = \{a, \$\}$

Union, first of X the rest part does not come into picture and of course, rest is initialized to follow of X i minus 1, so this iteration continuous until the follow sets have not changed. So, let us take an example so here it is the same grammar that we considered for the first computation to begin with follow of S is dollar and the others are all phi, the first competition let us assume has been completed. So, first of A is Epsilon you know first of S is A Epsilon, first of A is a b c Epsilon, first of B is a c Epsilon, we completed in the example prior to this.

In iteration one, we used the symbol X union equal to y to mean X equal to X union y this is very similar to X plus equal to y in c, it is is used to you know reduce the amount of material present on this slide S going to a A S. So, here this is the S that we consider now, follow of S from the previous iterations and union dollar, so why is that follow of this symbol is the S and this is the S, that we are considering to contain dollar.

So, follow of this S is follow of this S into a zero. So, that is dollar and rest would be follow of this s which is again dollar the next symbol is here. So, follow of A is follow of A union first of S minus Epsilon union rest the reason is this S can you know first of S contains Epsilon here. So, whatever follows this S will also follow this A and therefore, follow of a will include first of S union the rest is nothing but the follow of S.

So, that computes the follow of S and a here the for this production, so now let us take the production A going to SB here, first of B also contains Epsilon, so follow of S will

contain follow of first of B will contain follow of B, union follow of A. So, that is because b derives, you know this is the last symbol, so once a replaces is replaced by SB whatever is following A will also follow V and the rest is recorded as follow A. Now, the fact that B produces Epsilon is useful in computing the follow of this S, follow of S will be first of B minus Epsilon union rest is nothing but follow of A, that is because B produces Epsilon.

So, all the symbols which follow a will also follow this S, so that gives us a c dollar from this production follow of A will be follow of A union follow of b and from this production follow of S will be follow of S, union follow of b at the end of iteration one these are values we got.

(Refer Slide Time: 21:40)

FOLLOW Computation: Algorithm Trace (contd.)

- $first(S) = \{a, \epsilon\}$; $first(A) = \{a, b, c, \epsilon\}$; $first(B) = \{a, c, \epsilon\}$;
- At the end of iteration 1
 $follow(S) = \{a, c, \$\}$; $follow(A) = follow(B) = \{a, \$\}$
- Iteration 2
- $S \rightarrow aAS$: $follow(S) \cup = \{a, c, \$\}$;
 $rest = follow(S) = \{a, c, \$\}$
 $follow(A) \cup = (first(S) - \{\epsilon\}) \cup rest = \{a, c, \$\}$ (changed!)
- $A \rightarrow SB$: $follow(B) \cup = follow(A) = \{a, c, \$\}$ (changed!)
 $rest = follow(A) = \{a, c, \$\}$
 $follow(S) \cup = (first(B) - \{\epsilon\}) \cup rest = \{a, c, \$\}$ (no change)
- At the end of iteration 2
 $follow(S) = follow(A) = follow(B) = \{a, c, \$\}$;
- The follow sets do not change any further

And in iteration two, the production a A S gives us follow of S equal to a c dollar, so that is there is no change their, whereas follow of A equal to follow of A union first of S minus Epsilon, union rest that is because S produces an Epsilon. So, that gives us a c dollar now the follow set has changed, so follow of A was different, now it has changed and the production A to SB, we compute follow of b. So, that again changes to a c dollar.

Follow of S does not change and at the end of iteration two, we have follow of S equal to follow of A, equal to follow of B, equal to a c dollar and the follow sets do not change any further. So, here in these cases we required two iterations, but it is possible that more than two will be required that is easy to see.

(Refer Slide Time: 22:43)

LL(1) Conditions

- Let G be a context-free grammar
- G is LL(1) iff for every pair of productions $A \rightarrow \alpha$ and $A \rightarrow \beta$, the following condition holds
 - $\text{dirsymp}(\alpha) \cap \text{dirsymp}(\beta) = \emptyset$, where
 - $\text{dirsymp}(\gamma) = \{ \epsilon \in \text{first}(\gamma) \}$ then $((\text{first}(\gamma) - \{ \epsilon \}) \cup \text{follow}(A))$ else $\text{first}(\gamma)$
 - (γ stands for α or β)
 - dirsymp stands for "direction symbol set"
 - An equivalent formulation (as in ALSU's book) is as below
 - $\text{first}(\alpha, \text{follow}(A)) \cap \text{first}(\beta, \text{follow}(A)) = \emptyset$
 - Construction of the LL(1) parsing table
 - for each production $A \rightarrow \alpha$
 - for each symbol $s \in \text{dirsymp}(\alpha)$
 - /* s may be either a terminal symbol or $\$$ */
 - add $A \rightarrow \alpha$ to $\text{LLPT}[A, s]$
 - Make each undefined entry of LLPT as *error*

MPTCL
Y.N. Saha
Design

So, let us consider the LL (1) conditions now, so far we computed the first and follow, so now, we are going to define the LL (1) grammar condition based on the first and follow the reason, we want to do that is we want to show an algorithm for computing the parsing table for LL (1) grammars and those are based on first and follow. Suppose, G is our grammar G is LL (1), if every pair of productions A to α and A to β , the following conditions satisfies.

So, the point is during LL (1) parsing, when there is a choice of productions A going to α or A going β , we should be able to say that one of them applies by looking at the next symbol in the input. So, this can be asserted if for every choice of productions, we are able to satisfy this particular condition which is stated here the condition says direction symbol, set of α intersection directions symbol, set of β is ϕ . So, what is direction symbol set direction symbol set of γ is if ϵ is in the first of γ , then it is first of γ minus ϵ union of follow here.

Otherwise, it is just first of γ , so if you know γ stands for either α or β , so in other words you know whatever A symbols α derives and β derives the first symbols of these should not be the same. Otherwise, it is the choice cannot be made by looking at the next input symbol, so there is an equivalent formulation in the you know Sethi and Ullman's book, Sethi and Ullman, it says first of α dot follow a

intersection first of beta dot follow A equal to phi for the same productions c_2 alpha and A to beta.

These conditions are identical it is easy to see that, because suppose alpha does not produce any Epsilon, in that case Epsilon in first of alpha is false, so the direction symbol set simply becomes first of alpha. So, the follow is inconsequential in that case suppose alpha does produce Epsilon, so first of alpha contains Epsilon in such a case, if this is Epsilon then the follow set of a will also be included in the first computation. So, we also do that here you know if Epsilon is in the first of gamma, first of gamma minus Epsilon union follow of A.

So, all the elements in the follow of A will also be included in direction symbol set of alpha, in this case the same is true for this as well first of beta dot follow here. So, direction symbol intersection or this intersection or identical as far as the LL (1) condition holds. Now, this is the condition for the grammar to satisfy the LL (1) parsing property, suppose we condition is satisfied can we build a parsing table LL (1) parsing table from the grammar definitely.


The process is quite simple for each production A to alpha, we check each symbol S in direction symbol alpha. So, S may be either a terminal symbol or the end of file symbol dollar just add the production A to alpha to the parse table A at the point, A comma S, we will see an example of this, so and make each undefined entry has error. So, with the other formulation first and follow of formulation the table construction is very similar consider the first of alpha add the productions.

(Refer Slide Time: 27:11)

LL(1) Table Construction using *FIRST* and *FOLLOW*

```
for each production  $A \rightarrow \alpha$ 
  for each terminal symbol  $a \in \text{first}(\alpha)$ 
    add  $A \rightarrow \alpha$  to  $\text{LLPT}[A, a]$ 
  if  $\epsilon \in \text{first}(\alpha)$  {
    for each terminal symbol  $b \in \text{follow}(A)$ 
      add  $A \rightarrow \alpha$  to  $\text{LLPT}[A, b]$ 
    if  $\$ \in \text{follow}(A)$ 
      add  $A \rightarrow \alpha$  to  $\text{LLPT}[A, \$]$ 
  }
Make each undefined entry of  $\text{LLPT}$  as error
```

- After the construction of the LL(1) table is complete (following any of the two methods), if any slot in the LL(1) table has two or more productions, then the grammar is NOT LL(1)





YN, Sakant, Deshpande

And if Epsilon is in first of alpha, then add for all follow of A as well and if dollar is in the follow of A add it for the dollar as well. So, after the construction of the table, so of course, we could test the productions for the LL (1) property and then build the table or we could build the table and check if any slot in the LL (1) table has more than one production to or more then the grammar is not LL (1), so these two conditions are identical.

(Refer Slide Time: 27:58)

Simple Example of LL(1) Grammar

- P1: $S \rightarrow \text{if } (a) S \text{ else } S \mid \text{while } (a) S \mid \text{begin } SL \text{ end}$
- P2: $SL \rightarrow S S'$
- P3: $S' \rightarrow ; \mid \epsilon$
- {if, while, begin, end, a, (,), ;} are all terminal symbols
- Clearly, all alternatives of P1 start with distinct symbols and hence create no problem
- P2 has no choices
- Regarding P3, $\text{dirstymb}(;SL) = \{ ; \}$, and $\text{dirstymb}(\epsilon) = \{ \text{end} \}$, and the two have no common symbols
- Hence the grammar is LL(1)



YN, Sakant, Deshpande

So, let us take up a few examples and understand what exactly this LL (1) property means, so here is a very simple grammar the sentences or the statements are either if A, S else S or while a S or begin SL end, where SL is a statement list. So, we derive S prime from SL and S prime derives semicolon SL or Epsilon as far as production one is concerned each of the three alternatives begin with a different token here is if here is while and here is begin.

So, these three terminal symbols are all different, so as far as P 1 is concerned there is the LL (1) property is satisfied for P 2 there is no choice. So, there is nothing to that LL (1) property is trivially satisfied for P 3, we have s prime going to a semicolon SL or Epsilon. So, we need to compute the direction symbol set for this side and direction symbol set for this epsilon as well, direction symbol set of semicolon SL all the strings begin with semicolon.

So, it similarly just non null symbol, so it is just semicolon and direction symbol set of epsilon to compute that the symbol of Epsilon kit produces an Epsilon of course, therefore, it is nothing but the follow of S prime to compute, the follow of S prime, we look at the production where S prime is placed. So, this production has S prime of the right hand side and since this is the last symbol here it tells us that follow of S prime you know is nothing but follow of SL.

Now, look at SL is contained here and what follows SL is this token end, so direction symbol set of Epsilon is end and again semicolon and end are two different symbols they do not intersect to non phi therefore, the LL (1) indeed LL (1).

(Refer Slide Time: 30:19)

LL(1) Table Construction Example 1

LL(1) Parsing Table for the original grammar

	if	id	else	a	\$
S'	S' → S\$				S' → S\$
S	S → if id S S → if id S else S			S → a	

Original Grammar

Grammar is not LL(1)

Original Grammar

S' → S\$

S → if id S |
if id S else S |
a


tokens: if, id, else, a

dirsyb(S\$) = {if,a}; dirsyb(a) = {a}

dirsyb(if id S) = {if}

dirsyb(if id S else S) = {if}

$dirsyb(if\ id\ S) \cap dirsyb(a) = \emptyset$
 $dirsyb(if\ id\ S\ else\ S) \cap dirsyb(a) = \emptyset$
 $dirsyb(if\ id\ S) \cap dirsyb(if\ id\ S\ else\ S) \neq \emptyset$



Here is a different example, so if you observe in this example I carefully avoided the two possibilities of if statements if a then S, if A then, S else, S these are the two possibilities. So, I included only one of them what happens if we include both, so let the grammar be S prime going to dollar S going to if id S or if id S else S or A. So, A is any other you know statement now trivially you can see that these two alternatives begin with the same symbol if and therefore, our hunch is that the LL (1) test will be not satisfied.

So, let us formally check it out for this you know s prime are producing S dollar, so direction symbols are tough S dollar is if comma A because direction symbol set is nothing but the first set, if there is no Epsilon involved. So, S produces if and A as the first symbols or the first tokens, so if and A are included in the direct symbol of S dollar direct symbol of the terminal symbol A is of course just a direct symbol of or symbol.

If i d S is if because the first symbol is non null if symbol here also it is non null and if so this is also if so if you look at the alternatives these two these two. And then this and this for the first pair it is indeed five the intersection is phi for the second pair the intersection is phi, but for the third pair if i d S and if i d S else, the sets contain the symbols if and therefore, it is not equal to phi.

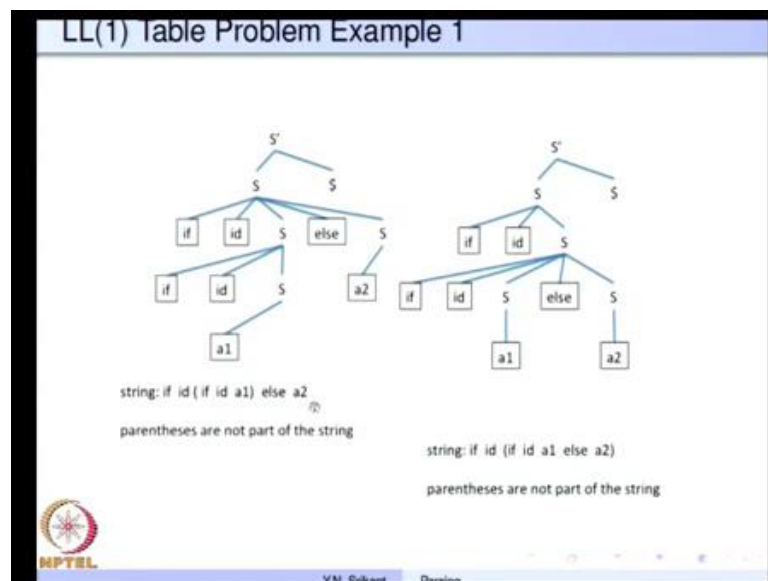
So, the LL (1) test fails let us see what happens, if we fill the table because using directions symbols to check the LL (1) property is one way building the table and then now checking the slot is a simpler option. So, let us see how it can be done, so for the

non terminal S prime, so S prime going to S dollar, so we consider the direction symbol set of S dollar, it contains two symbols i f and A, so the production S prime going to going S dollar is added for the two symbols i f and A.

For the production S going to i f, i d S, we look at the our direction symbol set of i d S and that that is just if so for the non terminal S on the symbol, if we add the production S going to if i d S, and for the second production S going to if i d else S. The direction symbol set again contains if so for the non terminal S for the symbol, if which is contained in the direction symbol set we add the production S going to if i d S else S and the last one S going to A direction symbol set of A is A.

So, for the combination of S and A, we add the production s going to A. So, now this stool you know contains two alternatives, so two productions and therefore, the grammar is not LL (1) of course, the problem with this grammar, is it is ambiguous we have seen this example before.

(Refer Slide Time: 34:19)



So, if you look at this the two parse trees which are produced by for the same string if i d, if i d A 1 else A 2, we already know this example we know that there are two parse trees possible because the grammar is ambiguous and therefore, precisely what creates the make the grammar fail the LL (1) test.

(Refer Slide Time: 34:44)

LL(1) Table Construction Example 2

Original Grammar

$$S' \rightarrow SS$$

$$S \rightarrow \text{if id } S \mid \text{if id } S \text{ else } S \mid a$$

LL(1) Parsing Table for modified grammar

	if	else	a	\$	
S'	S' → SS			S' → SS	
S	S → if id S S1		S → a		
S1		S1 → ε S1 → else S		S1 → ε	

$\text{dirsymb}(SS) = \{\text{if}, a\}$; $\text{dirsymb}(a) = \{a\}$
 $\text{dirsymb}(\text{if id } S \text{ S1}) = \{\text{if}\}$
 $\text{dirsymb}(\text{else } S) = \{\text{else}\}$
 $\text{dirsymb}(\epsilon) = \{\text{else}, \$\}$

Grammar is not LL(1)

Left-Factored Grammar

$$S' \rightarrow SS$$

$$S \rightarrow \text{if id } S \text{ S1} \mid a$$

$$S1 \rightarrow \epsilon \mid \text{else } S$$

tokens: if, id, else, a

$\text{dirsymb}(\text{if id } S \text{ S1}) \cap \text{dirsymb}(a) = \emptyset$
 $\text{dirsymb}(\epsilon) \cap \text{dirsymb}(\text{else } S) \neq \emptyset$

This is the original grammar that we just now saw suppose we apply what is known as left factoring. So, the problem is if i d S is common to both these productions, suppose we make a you know factoring out of it make the if ideas as common to both these productions and introduce a new non terminal called S 1 which goes to either Epsilon or else S. So, between these two if we simply expand S 1, we get both the productions which are here the rest of the grammar remains the same.

Now, you compute the direction symbol sets for S dollar, it is the same for A it is the same for if i d S, S 1, it is just if single one if i d A and then direction symbol set of else S is just S and for the Epsilon, so here is Epsilon. So, direction symbol set of this Epsilon will be follow of S 1 is here. So, it tells us that follow of S 1 is nothing but follow of S is now in two places this S give rise to gives rise to dollar and this S gives rise to first of S 1.

So, first of S 1 contains else, so direction symbol set of Epsilon contains else comma dollar, if you look at the intersection this intersection of course, this is this has if and that has A. So, it is non phi, but for this intersection we have non phi you know which is the previous intersection, if and A, it is phi whereas for this inter section it is not phi. So, this contains else and this contains else and dollar, if you fill the table entries, so exactly the way before S prime gets these two entries S.

Now has just one entry for if and one entry for A, but the two entries are now shifted to S one and else, so the problem has just shifted to a different set of productions from the if i d production to else S production this grammar is still ambiguous. So, if you look at the string if i d, if i d A 1 else, A 2 and we know that we can produce these two parse tree, so until this point the parse trees are identical. So, both of them are the same, but here do we produce this Epsilon or do we produce else S A 2 else.

This is one you know ambiguity and for this S, again the expansion is if i d S, S 1, so if i d S, S 1 and for this S, we produce an Epsilon or do we expand it to else a two this is the ever ambiguity. So, because of these two the two parse trees set are different and because the grammar is you know ambiguous and it fails the LL (1) test.

(Refer Slide Time: 38:15)

LL(1) Table Construction Example 3

$S' \rightarrow SS$
 $S \rightarrow aAS \mid c$
 $A \rightarrow ba \mid SB$
 $B \rightarrow bA \mid S$
 Grammar is LL(1)

LL(1) Parsing Table				
	a	b	c	\$
S'	$S' \rightarrow SS$		$S' \rightarrow SS$	
S	$S \rightarrow aAS$		$S \rightarrow c$	
A	$A \rightarrow SB$	$A \rightarrow ba$	$A \rightarrow SB$	
B	$B \rightarrow S$	$B \rightarrow bA$	$B \rightarrow S$	

$\text{first}(S) = \{a, c\}$ $\text{first}(A) = \{a, b, c\}$ $\text{first}(B) = \{a, b, c\}$	$\text{dirsymp}(aAS) \cap \text{dirsymp}(c) = \emptyset$ $\text{dirsymp}(ba) \cap \text{dirsymp}(SB) = \emptyset$ $\text{dirsymp}(bA) \cap \text{dirsymp}(S) = \emptyset$	
$\text{follow}(S) = \{a, b, c, \$\}$ $\text{follow}(A) = \{a, c\}$ $\text{follow}(B) = \{a, c\}$	$\text{dirsymp}(SS) = \{a, c\}$ $\text{dirsymp}(aAS) = \{a\}$ $\text{dirsymp}(c) = \{c\}$	$\text{dirsymp}(ba) = \{b\}$ $\text{dirsymp}(SB) = \{a, c\}$ $\text{dirsymp}(bA) = \{b\}$ $\text{dirsymp}(S) = \{a, c\}$

So, another example and in this case the grammar is indeed LL (1), so you have this good hold grammar S prime going to S dollar, S going to a A S or c, A going to b a or SB, B going to b A or S. We have already worked out the first end follow sets so I just reproduce them here, now let us see what the directions symbol sets are for S dollar, it is the first of S. So, first of S contains a c, so direction symbol set of S dollar is a c for the string a A S direction symbol, set is obviously little a, because A is not a normal character.

Similarly, direct symbol C is c m, direct symbol of b a is B direct symbol of SB is really first of S, so which is nothing but a comma c direct symbol of b A is little b, because it

starts with B and direct symbol of S is first of S which is a comma c, now these are the three pairs which we have here for a A S and c the direction symbol sets do not intercept they produce phi for b A and SB. This is b A, this is SB, so b A is b and SB is a comma c.

Again they produce A file for b A and S 1 is a little and the other one is a comma c, again it produces A phi. So, the LL (1) test is satisfied, now let us fill the table for the production S prime going to S dollar for the symbols a and c, we place this production S prime going to S dollar in the table.

For the two productions a A, S and c lets take one at a time S going to a A S, the direct symbol is A, so we place this production for the combination S and A for the production S going to c has direct symbol c equal to c. So, for the combination of S and c, we place S to c, so similarly A to b a is placed you know for the combination A to b a and A to SB has first of S equal to b A direct symbol is also b A, so direct symbol of S is a c.

So, for A and c, we place the production A going to SB and A going to SB, similarly for these two productions for b, we provide the production B going b a for the first of S, that is a comma c, we place the productions B going to S. So, that table has no conflicts that, is no slot has more than one entry and therefore, the grammar is indeed LL (1).

(Refer Slide Time: 41:22)

LL(1) Table Construction Example 4

Left-Recursive Grammar for Statement List

$S' \rightarrow SL S$
 $SL \rightarrow SL S \mid S$
 $S \rightarrow a$

$dirsyb(SL S) = \{a\}$
 $dirsyb(a) = \{a\}$
 $dirsyb(SL S) = \{a\}$
 $dirsyb(S) = \{a\}$

$dirsyb(SL S) \cap dirsyb(S) \neq \emptyset$

$dirsyb(SL S) = \{a\}$
 $dirsyb(a) = \{a\}$
 $dirsyb(S A) = \{a\}$
 $dirsyb(\epsilon) = \{S\}$

LL(1) Parsing Table for Left-Recursive Grammar

	a	\$
S'	$S' \rightarrow SL S$	
SL	$SL \rightarrow SL S$ $SL \rightarrow S$	
S	$S \rightarrow a$	

Grammar is not LL(1)


Right-Recursive Grammar for Statement List

$S' \rightarrow SL S$
 $SL \rightarrow S A$
 $A \rightarrow S A \mid \epsilon$
 $S \rightarrow a$

LL(1) Parsing Table for Right-Recursive Grammar

	a	\$
S'	$S' \rightarrow SL S$	
SL	$SL \rightarrow S A$	
A	$A \rightarrow S A$ $A \rightarrow \epsilon$	
S	$S \rightarrow a$	

$dirsyb(S A) \cap dirsyb(\epsilon) = \emptyset$



VN Sukant Demo

Another example, so this time the example tries to show that rewriting a grammar or transforming a grammar makes it LL (1), so here is a grammar for statement lists S' going to SL dollar and statement list is SL or S going to A . So, we as A is the any statement, so we produce as many A as necessary using the recursive production here. So, note that this is a left recursive grammar the theorem is that all left recursive grammars fail to be LL (1) here is a hint of why that happens.

See SL produces and S and SL also produces S , so in some sense the first of SL and this first of S , obviously will contain the same entries and therefore, direct symbol of these two will obviously, not be phi and that is why the you know LL (1) property fails. So, we can see that happening here direct symbol of SL dollar is just A , because first of SL is nothing but a first of SL is first of A and that is a direct symbol of A is a direct symbol of SL , S is also A , because that is nothing but first of SL which is first of S and that is A and direct symbol of S is A .

So, you can easily see that the interception of direct symbol SL and direct symbol S which is here is not phi and therefore, the table has entries SL going to SL dollar and SL , S for the combination SL and A , so the grammar is not LL (1). Suppose this left recursive grammar is mode into a cursive grammar, so it is actually the same language it is just that, we rewrite the grammar as a recursive grammar.

So, s' produces SL dollar that remains as it before instead of SL going to SL , S bar S , we make the production as SL going to S c rewrite it and then A going to S A or Epsilon S going to A remain as it is. So, between these two the A part produces as many instances of this treatments as necessary. So, but the grammar is not left recursive anymore it is recursive, because a appears here and it is the same symbol as the LH S here. So, this is your recursive production whereas, this is a left recursive production.

So, if we compute the direction symbol sets for this we get for SL dollar and A , we get the same sets for direct symbol of SA , we get a because direct symbol of SA happens to be first of S and that gives us A and what about direct symbol of Epsilon. So, here is epsilon this is null able, it produces an Epsilon that is so it includes the direct symbol of epsilon will include the follow of a and let us see where A occurs it occurs here, but that again tells us that it is follow of A it occurs here.

So, that tells us that the follow of SL is also required and SL occurs here and that includes a dollar the follow of SL includes the dollar, so direct symbol of Epsilon gives us dollar and now for the alternatives S, A and Epsilon, we have S, A, as a direct symbol of S, A as A and direct symbol of Epsilon as dollar. So, these two actually intersect to phi therefore, the grammar is indeed LL (1) and we can fill up the table in this fashion there are no conflicts here. So, A going to A, and A combination is a going to S, A and A and dollar combination is a going to Epsilon.

So, this is the LL (1) parsing table for the right recursive grammar, now this gives us A hint that I, we are able to convert left recursive grammars to right recursive grammars then may be some of the grammars which fail the LL (1) test can be made to succeed in the test it is indeed true.

(Refer Slide Time: 46:22)

Elimination of Useless Symbols

Now we study the *grammar transformations*, elimination of useless symbols, elimination of left recursion and left factoring

- Given a grammar $G = (N, T, P, S)$, a non-terminal X is *useful* if $S \Rightarrow^* \alpha X \beta \Rightarrow w$, where, $w \in T^*$
Otherwise, X is useless
- Two conditions have to be met to ensure that X is useful
 - $X \Rightarrow^* w, w \in T^*$ (X derives some terminal string)
 - $S \Rightarrow^* \alpha X \beta$ (X occurs in some string derivable from S)
- Example: $S \rightarrow AB \mid CA, B \rightarrow BC \mid AB, A \rightarrow a, C \rightarrow aB \mid b, D \rightarrow d$
 - $A \rightarrow a, C \rightarrow b, D \rightarrow d, S \rightarrow CA$
 - $S \rightarrow CA, A \rightarrow a, C \rightarrow b$

MPTEL
Y.N. Srikant

So, now we study some of the grammar transformations, so as I mentioned before to compute the first and follow sets we must make sure that the grammar has no useless symbols and I also mentioned at that time that the elimination of users symbols will be dealt with later. So, now we come to the algorithm which eliminates the useless symbols what exactly is a useful non terminal and what is a useless non terminal the grammar transformations that, we study are elimination of useless symbols elimination of left recursion and what is known as left factoring.

We have seen instances of this before, but let us do it formally, now so the grammar you know let it be NPTS as usual suppose a non terminal X is in is it occurs in a sentential form that is s derives $\alpha X \beta$ and this $\alpha X \beta$ finally, derives W which is a string in the language of G .

So, S derives w , so here I should have put A star here to show that there are more than one productions possible and that is the non terminal must occur in a sentential form and finally, we should be able to derive a string from the sentential form in which the non terminal occurs if both these happen then you know the non terminal X is useful. Because, it has been instrumental and producing A string in the language if it is not, so then X is useless, so now, let us be more you know you know let us pin down the conditions which are required to make x useful the first condition is $S X$ must derive W and w must be a string. So, that is X derives some terminal string the second condition is $S X$ must occur in some sentential form S derives $\alpha X \beta$.

So, there is X occurs in some string derivable from S , both these conditions must hold at the same time if we consider them individually then you know actually some useless symbol may slip into other grammar let us take an example here is S , the grammar is S going to $a b$ or $c a b$ going to $b c$ or $a b a$ going to $a c$ going to $a b$ or $b d$ going to d . So, let us see, if what these conditions yield. So, we are before that if you look at the productions it is very clear you know intuitively that this non terminal b is useless why see what happens is this non terminal b again has two production b going to $b c$ and b going to $a b$, but there are no productions which yield terminal string for b .

So, in other words if we apply the production s going to $a b$ we can keep on applying b going to $b c$ or b going to $a b$, but we will not be able to finish it off by using a terminal production because it has none. So, that the reason why b is a useless symbol and it. So, happens that once b is useless the productions containing b will also be useless and the have to be eliminated. So, s going to $a b$ must be eliminated b going to $b c$ must be eliminated b going to $a b$ of course,, must be eliminated and then c going to $a b$ must also be eliminated. So, what we really do is we systematically apply the two conditions here we collect the non terminals which produce terminal strings to begin with. So, a going to a and c going to b these are the two production which have only terminal symbols on the right hand side.

So, they indeed produce terminal strings now we take those non terminals which have only the non terminals corresponding to these productions on the right hand side it. So, happens that S going to c a is such a production and of course, D going to D is other production which has been included because it produces only terminal strings on the it has only terminal strings on the right hand side. So, S going c a has c and A which are already included here. So, the first condition now is satisfied right every non terminal produces some terminal symbol. So, those are the only productions which we have included.

But, now what about the non terminal D even though it produces A terminal string on its own does it occur in a sentential form which is derivable from S definitely not because S to c a is the only production which we have considered which contains non terminals on the right hand side and there is no D in it. So, D going D is also useless and the symbol D happens to be useless. So, finally, applying the condition to above we get s going to c A A going to a and c going to b, the this is the grammar which contains only useful symbols.

(Refer Slide Time: 52:30)

Testing for $S \Rightarrow^* \alpha X \beta$

$G' = (N', T', P', S')$ is the new grammar
 $N' = \{S\};$
Repeat {
for each production $A \rightarrow a_1 | a_2 | \dots | a_n$ with $A \in N'$ do
add all nonterminals of a_1, a_2, \dots, a_n to N' and
all terminals of a_1, a_2, \dots, a_n to T'
} until there is no change in N' and T'
 $P' = \{p \mid \text{all symbols of } p \text{ are in } N' \cup T'\}; S' = S$

NPTEL

YN Sakant

So, now let us formalize the two algorithms and see how they can be stated. So, the first condition X derives W, so G be the grammar and G prime is the new grammar. So, we iterate we start with N old equal to phi and N, new is all those non terminals X which have only terminal strings on the right hand side. So, those are included in N, new while

N_{old} not equal to N_{new} keep iterating. So, we accumulate N_{new} in N_{old} or rather retain it in N_{old} and N_{new} becomes $N_{old} \cup$ all those symbols X , such that $X \rightarrow \alpha$ is a production and α contains only those symbols which have already been included in N_{old} and of course, otherwise it is a terminal symbol. So, this gives us a few more symbols hopefully and that will be our n_{new} .

We go back now N_{new} is bigger therefore, A it is not equal to N_{old} , so again expand N_{new} by including more and more symbols, until we cannot grow it anymore and we come out we say N_{prime} is N_{new} , T_{prime} is the old T , itself S_{prime} is S and P_{prime} is all the symbols P , you know all the productions P which contain only the symbols in $N_{prime} \cup T_{prime}$ in the right hand side part of the production. So, this is exactly what I explained just now you know in this example. So, we included $A \rightarrow ac$ to $b \rightarrow d$ and $S \rightarrow ca$ in this first step.

The second algorithm checks whether the non terminal X is present in any sentential form that is whether it can be derived from S , so to do that we start with this start symbol and then try to include as many symbols as possible in the set N_{prime} consider the productions of the symbols which are present in N_{prime} . So, $A \rightarrow \alpha_1$ to α_2 α_3 α_4 etcetera any of these you know with A in N_{prime} . So, to begin with its always S going to something the all the productions of S . Now consider the you know non terminals of α_1 , α_2 etcetera add them to N_{prime} consider, the terminal symbols of all this and add them to T_{prime} .

Here I must stress that the grammar G that, we consider here is the one in which we have applied the production or rather the algorithm which is presented here that is we have removed some of the useless symbols already. So, this is repeated until there is no change in N_{prime} and T_{prime} , then we accumulate in P_{prime} , the productions all of the symbols are in $N_{prime} \cup T_{prime}$. So, this is what we did here you know once we did is start from S , we include c and a and those productions, and those are the only ones which will be included in the set of productions. So, let us stop here and in the next lecture we will consider a recursive descent parsing.

Thank you.