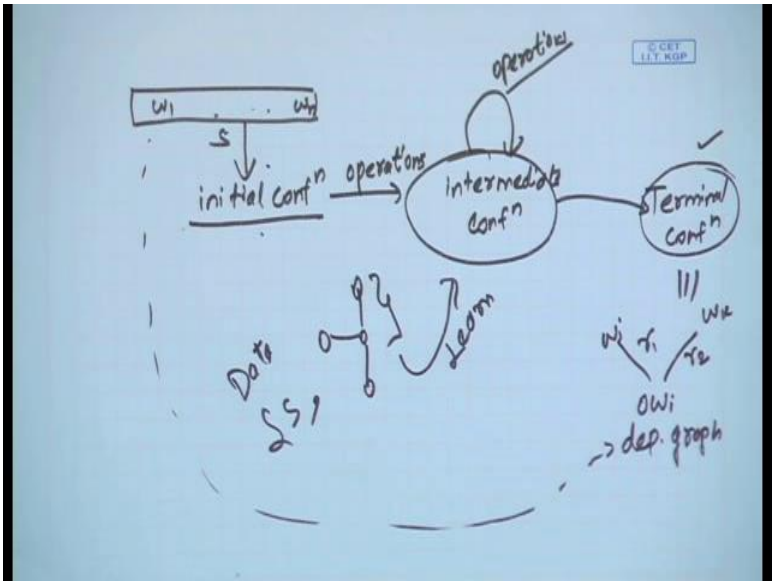
Natural Language Processing
Prof. Pawan Goyal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 27
Transition Based Parsing: Formulation

So, welcome to the second lecture of this week. So, we had started for our dependency parsing formulation and we define what is the problem and what are the very formal conditions on a dependency graph in the last lecture. So, we are given input sentence, I want to obtain a dependency graph for that sentence. And further we discussed that there are certain ways we can handle this problem. And in this course, we will mainly focus on approaches that you just that use some sort of labeled data for this task.

So, it will be some sort of machine learning task for finding out the dependency path for a new sentence. So, now so in the in the next four lectures in this week, I will cover two different methods for the same problem. But before discussing the methods in detail, let me try to give you a basic intuition on what we are trying to achieve. And this will also help you to understand that if you are given a new problem and you want to take it in a machine, you want to solve it in a machine learning based method, then how you should go about it.

(Refer Slide Time: 01:28)

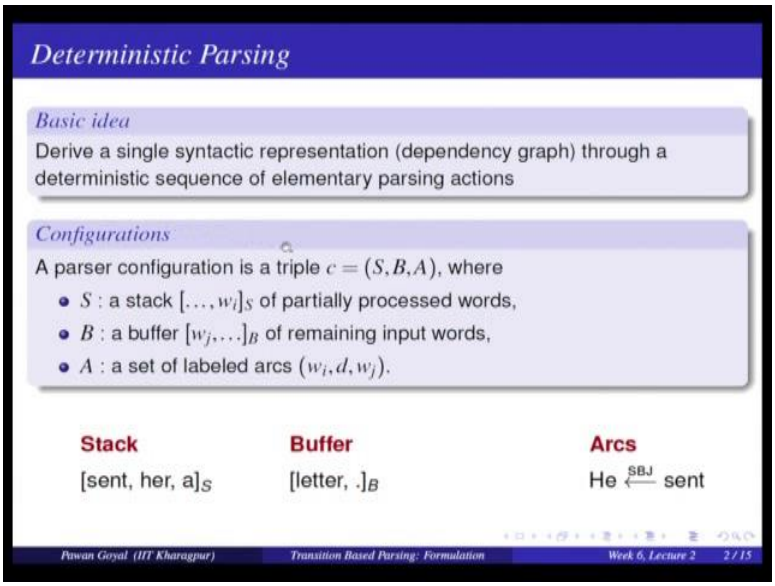


So, let me take this case. So, I have a sentence that contains some words. This is some arbitrary sentence, this is a new sentence I might not have seen it before. And my goal is to come up with the dependency graph. The nodes are different words and they have certain relations, r 1, r 2 etcetera are certain relations. This is my dependency graph. So, I want to go from the sentence to my dependency graph. Now, suppose I want to do it in a in some sort of systematic manner, by using some sort of data. So, what I will do. I try to convert that to some sort of configuration format, some initial configuration, so that means, there will be a function that will take any sentence as input and convert that would initial configuration.

Now, I will define some operations that I can make on this initial configuration and that will take me to some intermediate configuration. And I should be able to keep on doing these operations until at some point of time, I arrived at terminal configuration. And terminal configuration should be something that resembles a dependency graph. So, my whole task is now starting from the input sentence convert to a initial configuration apply some well defined operations and go to intermediate configuration until you finally, reach a terminal configuration that may be deterministic, may not be deterministic. So, this is my whole thing. This is my sentence and this is my dependency graph; and operations are something I do not know, but I need to define, I need to find out for a particular configuration, what are the operation that I should take.

Now, where does machine learning comes in here. So, this looks like very deterministic. What is the problem, the problem is they may not be a single operation. So, I should have a way to find out for the given configuration, what is the operation that I should take and that I am trying to learn from the data that I have, so that is where the machine learning comes in. So, we have data where we have a set of sentences and their corresponding dependency graphs. So, there I can find out for what configuration what was the operation I had taken. And I will use that to learn for a new sentence or a new configuration what operation I should be taking, so that given any new sentence I can actually find out its dependency graph. And this is just generic idea we will see specifically how it will be used in the two different methods that we will cover in this course.

(Refer Slide Time: 04:50)

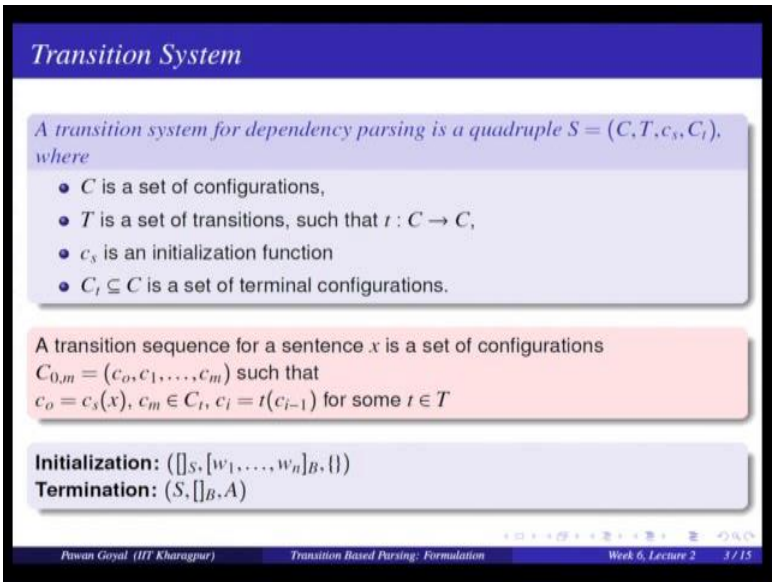


So, now the first method that we are going to cover is a transition based parsing method. So, we will see its formulation. So, what is the basic idea? So, it is similar to the intuition that I had provided earlier that is I have to derive a single syntactic representation, it is called the dependency graph via a deterministic sequence of elementary parsing actions and this is what we were saying as operations. I start with the sentence; I keep on having certain operations over that until I arrive at some dependency graph. So, now in this particular case of transition based parsing, what do I mean by a configuration, what do we mean by configuration look like.

So, a configuration, I will need a triple that is S stack, buffer and a set of arcs so that is at any point I can define a configuration. In the configuration, I will have a stack where there will be some set of words that has been partially processed. Then the buffer, it will contain the words that are remaining and whatever arcs I have already obtained via my operations, this will be in the set of arcs. So, at the bottom of this slide, you are seen one example of a configuration. So, this is for the sentence, he sent her a letter. So, this is the intermediate configuration, where you have the words sent, her, a, in the stack; so you are partially processed at this point. The word letter and punctuation are in the buffer they are remaining. And there is already an arc between the words he and sent. And he is the subject of the word sent – the verb sent. So, this is the intermediate configuration.

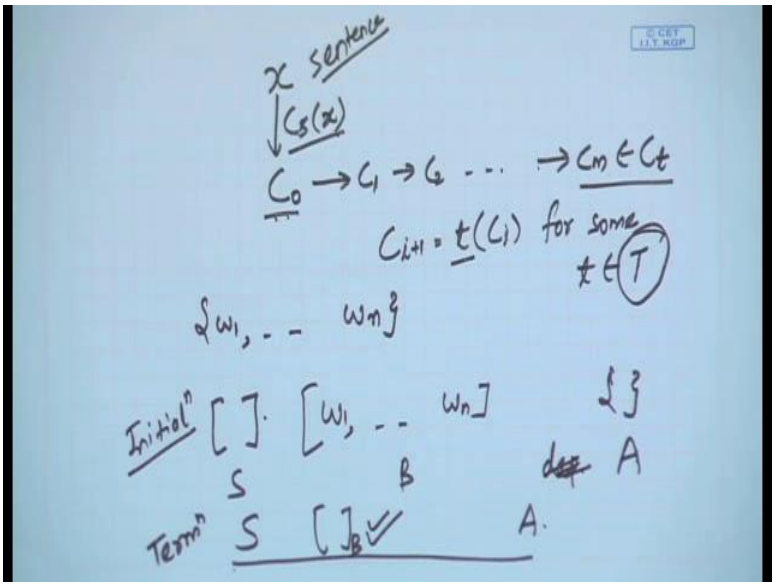
Now, how do I formally define my whole transition system? So, formally so it will be very analogous to what I started discussing. So, I need to define what are my possible configurations, how do I go from a sentence to an initial configuration and which configuration, I will call as a final configuration. And what are operations I should make so that I can transit from one configuration to another configuration and that is why name of transition system also comes in.

(Refer Slide Time: 07:04)



So, then formally we will define dependency parsing transition system as a quadruple that has C, T, c s and C t. And what are these, capital C is a set of configurations, all the configuration that are possible. Now, capital T denotes a set of transitions and each transitions takes me from one configuration to another configuration, from C to C; one configuration in the set C to another configuration in the set C. Now, c s will be an initialization function that you will give a sentence as input and you will find the initial configuration as output. And then you will have a set of terminal configurations, so; that means, whenever you reach a terminal configuration via your operations, you will stop, anyway call it as your dependency graph.

(Refer Slide Time: 08:17)



Now, whenever you are given a sentence x, what is my transition sequence. So, a sequence will be a set of configurations, it is starting from c 0 to c m and what are the conditions on those c 0 is something that you can derive from the sentence. So, suppose x is the sentence that you are giving as a input and you want to find out its dependency graph. So, now what is the idea? You have a function – initialization function, you have applied that function over x and you get a transition sorry you get a configuration c 0. Now, you keep on applying various operations and get c 1, c 2 up to c m; and c m is in the set of terminal configurations. This is my initialization function, set of terminal configurations and c 0 to c m, all are in the set of configurations.

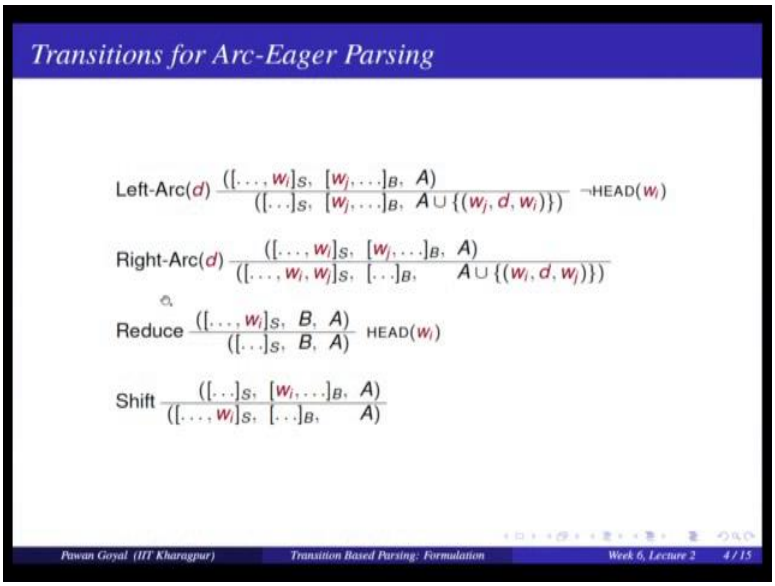
And what is the relation between any two conjugative c i and c i plus 1 relation would be I can obtained c i plus 1 by making a transition over c i, for some transition I have defined in my set capital T. So, this is my process. It starts with the sentence. Apply the initial initialization function get the initial configuration; keep on applying by a various transitions until you obtain a terminal configuration.

Now, as per the transition system that we have defined what will be the initialization. Initialization will be simple. So, I have to define what is my S stack, what is my buffer, and what are my arcs because when I am starting with the sentence w 1 to w n. Initially, my structure contains the set of partially processed words, so there is nothing that is processed right now, so my stack will be empty. My buffer should contain the set of

remaining input words, so remaining words are all the words w 1 to w n. And my arcs should contain whatever dependency relation, I have already found, so it will be empty, this will be empty set. So, this is my initialization.

Now, what will be the terminal configuration? A termination, so the idea is I should not have any words remaining in the buffer, so my buffer should be empty. So, terminal condition I can define as stack would have some may or may not some words; buffer should be empty, and I will have obtained a set of arcs. So, this is important, my buffer should be empty. A stack may or may not contain some words, and arcs will have some dependency relations already. So, now once you know what is my initial configuration, what is my termination condition, now what is remaining in this in this whole thing. We know all the possible set of configurations. So, what is remaining is what are all the possible transition that I can take for going from one configuration to another configuration. How can I move from initial configuration by a sequence of operations so that I arrive at a terminal configuration?

(Refer Slide Time: 11:28)

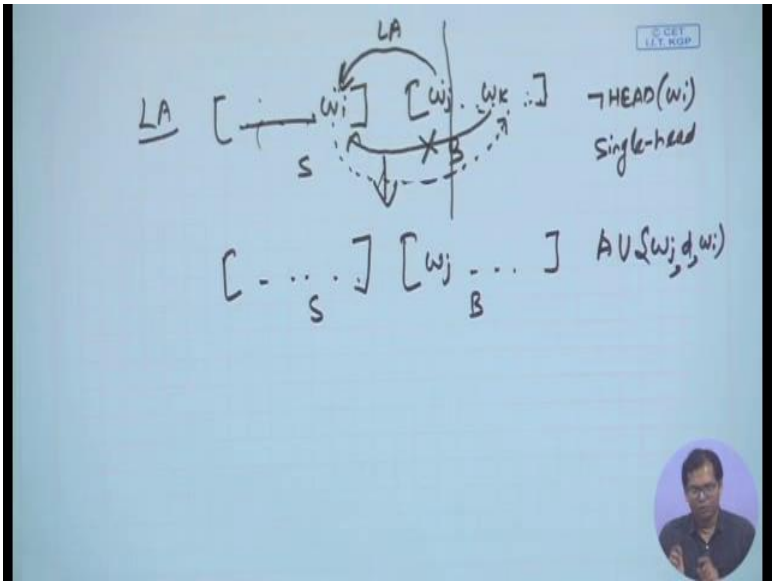


So, now in this system, there are four transitions that have been defined, so; that means, for any configuration, you can take one of these four transitions. So, in this slide, I am showing you the four transitions, different transitions that you can take, so how you should read it. So, in the left the names of the transitions, left-arc, right-arc for a dependency d, reduce and shift, these are the four transitions possible. Now, for each

transitions like left-arc, you will have what is the configuration that is the input to that of with which you are starting on top. And what is the configuration that you will end up with. A starting from this configuration, if you apply a left-arc, what configuration you will arrive at. And finally, in the right, if there is something that is a condition, what is the condition under which you can apply this transition; this is just a necessary condition. You need to have this conditions satisfied to be able to apply this transition.

So, for example, if I see the left-arc transition, what does it say, it says that I can apply it when there is stack contains some words. And I have been shown here the last word in the stack, I can also call it the top of this stack, the word is s i. Buffer contains some words w j, it is starting from w j, top of buffer is w j. And I have some set of arcs. Now, what do I mean by left-arc, left-arc means, I making a transition from w j to w i, yes, this is the left-arc, because the words are occurring in the same order that they occur in the sentence. So, there is a arc from w j to w i. So, what is the head here, w j is the head and w i is the dependent. So, now there is also a condition here, w i should not already have a head. And let us try to understand this condition.

(Refer Slide Time: 13:53)



So, what am I saying my stack contains some words and top is w i; buffer contains some words starting from w j, and I making a left-arc. So, the left-arc as the name says will be from buffer to S stack. And we are we have always operating on the top words in buffer and stack. So, I making an arc from the top word in buffer to the top word in S stack, this

is the left-arc. Now, what is the condition? Condition is w i should not already have a head.

Now, why is this condition? Just see if I am making this transition, we are saying that the w j is the head of w i. Now, w i already has a head, I cannot apply this, why? Can you see what particular condition in dependency parse, does it violate. So, dependency parse has a single head constraint, so it is a single head constraint, so that says each word can have at most one head. So, if w i already has a head, then I cannot apply this particular transition, so that is left-arc can be applied only if w i does not already have a head.

Now, suppose if I apply a left-arc, now I start with a configuration like that, what is my real thing configuration, so real thing configuration would be I will remove w i from the S stack. It will contain all the words except w i. Buffer - will remain as it is. And my arcs will be, I have got a new arc, what is that, w j dependent, w i. This is my new arc. So, I have removed w i from my S stack. Now, you might have this question, why am I removing w i from the S stack. If the word is removed, I cannot put it back. So, I can remove a word from the S stack only if I am sure that all its relations have already been captured. Now, the question here is can this word w i have anymore relations with any words. So, if w i has any relation with the words that occurs before w i, it would already have been captured at this point, because these are already partially processed words. So, the only relation w i might have is with any words after w j.

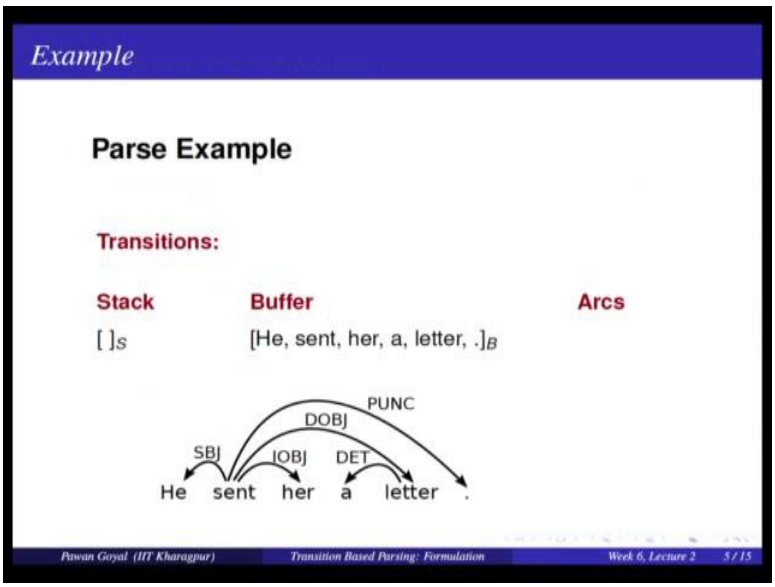
Now, let us take a scenario, take a word a w k. Now, what kind of relation w i might have with w k. It can be either an incoming arc or it can be an outgoing arc. Now, can it can have an incoming arc from w k, it cannot; yes, because it can have only one single one head, so this is not possible. Now, what is the other condition, can it have outgoing arc to some word w k. Now, what do you think about it, is that possible.

Now, if you think again about the formal conditions that we are defined, what condition is it violating, remember the condition on projectivity. If there is a relation from j to i then from i the next dependent can be only on this side of j, it cannot be on that side of j. So, this has to be either in between that is not possible or here. So, this is also not a possibility, so; that means, w i cannot have any further relations, so that is why we can remove w i from the S stack and this my s stack and this contain everything other than w i, buffer remains the same.

Now, let us look at the second transition – right-arc. So, now right-arc is, you are making a relation from w i to w j; w i is the head and w j is the dependent. There is no conditions that you have to see here. And when you apply this particular transition, what do you obtain w j goes to the S stack. So, I do not remove W i and W j goes to the S stack and I get a new arc. So, now again you can try to think here why we did not remove W i from the S stack, why we did not remove W j from the buffer and remove it all together, why we are not doing that. And try to think in terms of the constraints that we are put over our dependency parsing. We will come up with some nice intuitions that why we are defining the transitions in this particular manner. So, we have seen the justification for the previous case; now try to justification for yourself for this case. But here the difference is that now the transition is from W i to W j, and accordingly I am getting a new arc.

Then we have two more transitions reduce and shift. What is reduce, in reduce I remove the top word from the S stack and the condition is W i already has a head. If W i already has a head, I can remove that. And finally, the fourth transition is shift, what is that, I take the top word on the buffer and shift it to the top of S stack. And these are the four transitions. Now, the main thing here is when you are at a particular configuration, you might be able to take more than one of these operations or transitions. And your task could be to find out what is the transition that I should take in a particular scenario. So, I hope the four transitions are clear.

(Refer Slide Time: 20:48)

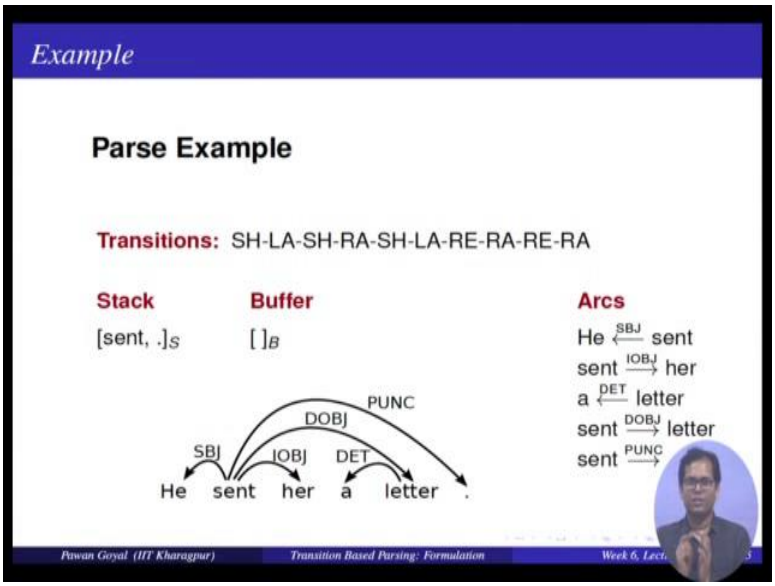


Let us see a particular example and how we do apply these transitions. So, we will take a very simple example. So, what we will have, we will have a sentence – he sent her a letter. And we also know the dependency parse of this sentence. Now, once you are given the sentence and its actual dependency parse that you want to obtain you have to find out what are the transitions that you should be taking at each intermediate or initial configuration, so that you can end up with the dependency graph. So, let us do this exercise.

So, here this is my sentence – he sent her a letter, and there is also a punctuation mark. How do we start, find out what is the initial configuration. Now, how do we convert this sentence to the initial configuration? S stack would be empty, buffer would contain all the words and arcs would be empty and that is what initial configuration; S stack is empty; buffer contains all the words; and arcs is empty. Now, your task is starts. This is the initial configuration.

Now, what is the transition that you must be taking here? So, this one is relatively easy. I cannot do left arc left arc means say arc from he to word in S stack. There is no word in S stack; they cannot be any dependency relation. Similarly, right arc is not possible because there is no word on the top of s stack; I cannot to reduce because again there is no word on the top of S stack. So, the only transition possible is shift. I can take the top word in the buffer and move that to the stack and this is what I will do. I will take the transition as shift.

(Refer Slide Time: 22:39)



So, if I do right, how do I modify my configuration? So, if I do a shift, he will go from buffer to the S stack and that is what I will obtain now. Now, this is my intermediate configuration. Now, again you defined out what is the transition that he must be taking at this point and that is where you will look at the dependency graph that you want to obtain. And you will see is there any relation you want to establish between the top of s stack and top of buffer.

So, in the parse we are seeing he is a subject for sent; that means, you can establish a relation between sent and he and will be that it should be from sent to he, so it should be a left arc relation so; that means, I can take a left arc transition at this point. And if I take a left arc, what will be the output configuration, I will remove he from the S stack, buffer will remain as it is; and arcs, I will get one more arc and that is what is my next configuration. S stack is empty; buffer contains the remaining words; and I have one arc obtained.

Now, you are at this configuration. Now, what is the transition you should take? Again whenever stack is empty, the only transition possible is shift. So, I will do a shift.

Now, I have one word in the in that stack sent, and there are some words in the buffer her, a and letter. Now, what is the transition you can take at this point? Again you will try to have a look at the dependency parse, you will see if this is relation between sent and her, yes her is a dependent and sent is the head. So, what is this transition, from sent

to her, it will be a right-arc. So, I will take a right-arc transition. And what will be the output of the right-arc, the word her should go the s stack and I will get a new relation in my arcs. So, this will be my configuration at this point. I have the words sent and her in the s stack, a and letter in buffer, and I got the arcs.

Now, am I at the point where I have some words in the s stack, I have some words in the buffer. And now I have to see what is the transition I should be taking care. So, let us have a look. Can I take a left-arc or right-arc, you see her and a are not related in my dependency parse, so I cannot take left-arc or right-arc. Now, what are the possibilities, the possibilities are I can either take reduce or shift is both are possible. Now, how do I choose between reduce and shift? And there is a rule of thumb for choosing between rule shift and reduce. And the idea is if you find the (Refer Time: 25:34) words in the stack not on the top, just they were before the top, any word before the top of the s stack that connects to that word on the top of the buffer, you do reduce otherwise shift. Of course, you can do reduce only if you have already found the head for their word.

So, in this case, should I take reduce or shift, let us take this, this rule of thumb. Is there a word in the stack below her that connects to the top of buffer, there is not so I cannot take reduce, I should take shift. So, if I do shift that is what I get. I take a shift; I now have sent her a letter. Now, once I made this configuration, what is the next transition I will take? This time it is easy, there is a relation between letter and a, this should be left-arc. Left-arc means I removing.

So, now again I am here, there is no relation between her and letter. So, now so; that means, you can choose between reduce and shift, so use the rule of thumb again. Is there a word below the top in the stack that connects to the top of buffer, and you will see, yes. The word sent in the s stack connects to the word letter in the buffer. So, here you can do reduce. And you do reduce and you have the word sent in the s stack, and letter and punctuation in the buffer. Now, is there any relation, yes, there should be a right-arc relation. So, right-arc means letter will go to the s stack, yes. Again, there is no relation between letter and dot – the punctuation. What should I take, is there any relation between any other word in s stack with the buffer, yes, sent and punctuation are related, so I will do a reduce here. Now, sent and punctuation are related by right-arc; and by right-arc, I will take the dot to the s stack. And this is my configuration.

Now, what do you see, should I go any further once I have reach this configuration and you will say no, because what is the condition terminal configuration, the condition is that my buffer should be empty. So, at this point, my buffer is already empty so; that means, I have arrived at a terminal configuration and that is where I will stop. And I have obtained all the possible set of arcs for this sentence. Now, this is an example we have seen just to understand how to take all these transitions. But does that help you to solve this problem whenever you are giving a new sentence, why a new sentence I mean you do not know dependency graph or you might also ask if I already know the dependency parse for a sentence, what is the need for doing all these transitions and that is where the concept of learning will come into picture.

So, we have seen here if we know the dependency graph, we can choose the configuration. At run time, whenever I am given a sentence my whole task is to find out what are the transitions I should be taking at different configurations. And this I will learn from my already labeled data. And this is what we will discuss in detail in the next lecture that is how do I use how do I use concept to define it as a learning problem. And for a new sentence, how do I come up with a dependency graph.

Thank you.