

# **Artificial Intelligence: The Rete Algorithm**

**Prof. Deepak Khemani**

**Department of Computer Science and Engineering**

**Indian Institute of Technology, Madras**

**Module – 03**

**Lecture - 08**

So we are looking at forward chaining and we want to look at an efficient way of doing this match process and we will look at this algorithm called the Rete algorithm which does this thing very efficiently. This was devised by a guy called Charles Forgy and he published his PhD thesis in 1979 and that was his PhD thesis. Subsequently of course he went on to start a company which is doing roaring business because forward chaining in rule based system is something which is very popular in the industry especially things like banking industry which does the you know have their managers write the rules and somebody else do the reasoning for them. And the whole idea of expert systems was basically that separate the knowledge part with the reasoning part. You have an inference engine that does the reasoning for you and the knowledge is provided by the domain experts essentially so and its very popular in many areas.

Ok so we have a knowledge base which we will call as a working memory and the data is the working memory elements. By data we mean the facts and then we have rules which are. So this whole way of looking at things came from Newell and Simon who did a lot of work in how humans solve problems. In fact, they wrote a book on human problem solving and remember that Simon was an economist by training essentially and they looked at all kinds of things. So their model of cognition if you might say was that we have a long term memory and a short term memory. The long term memory consists of rules of the kind that we know which is the association between things and that kind of stuffs. So all men are mortals is a part of our long term memory. Short term memory is concerned with the problems that you are trying to solve at hand. So it's the data concerned with the problem you are trying to solve. So in some sense the working memory corresponds to the short term memory. Anyway we are not so much interested in the psychological aspect of this but we must be aware of this fact. Whereas the rules, the correspond to long term memory.

(Refer Slide Time: 3:43)

Feb11\_2015-kr - Windows Journal

File Edit View Insert Actions Tools Help


Page Width

Forward Chaining with Rete Algorithm (Charles Forgy, 1979)

KB - Working Memory, data - Working Memory Elements (NMEs)

    ↑ STM

Rules ↔ LTM.



9/9

And the whole thing is that we employ or we exploit our long term memory to solve problems which are stated in the short term memory essentially. So if you remember the inference engine algorithm that we had seen, what we have is a working memory or the set of facts and this working memory is taken by this Match algorithm along with a set of rules. Remember that the rules correspond to the long term memory which produces, we mentioned briefly in the last class, something called conflict set. So just remember that the conflict set is just a collection of rule and matching data combos in some sense or pairs. Each rule with its matching data. This was fed to a resolve component which essentially picks a rule so we can denote it by some rule  $r_i$  and along with the data.

(Refer Slide Time: 5:21)

Feb11\_2015-kr - Windows Journal

File Edit View Insert Actions Tools Help

Page Width

Forward Chaining with Rete Algorithm (Charles Forgy, 1979)

KB - Working Memory, data - Working Memory Elements (NMEs)

    ↑ STM

Rules ↔ LTM.

WM = facts

↓

Rules → MATCH

↓


Conflict Set

↓

RESOLVE

↓

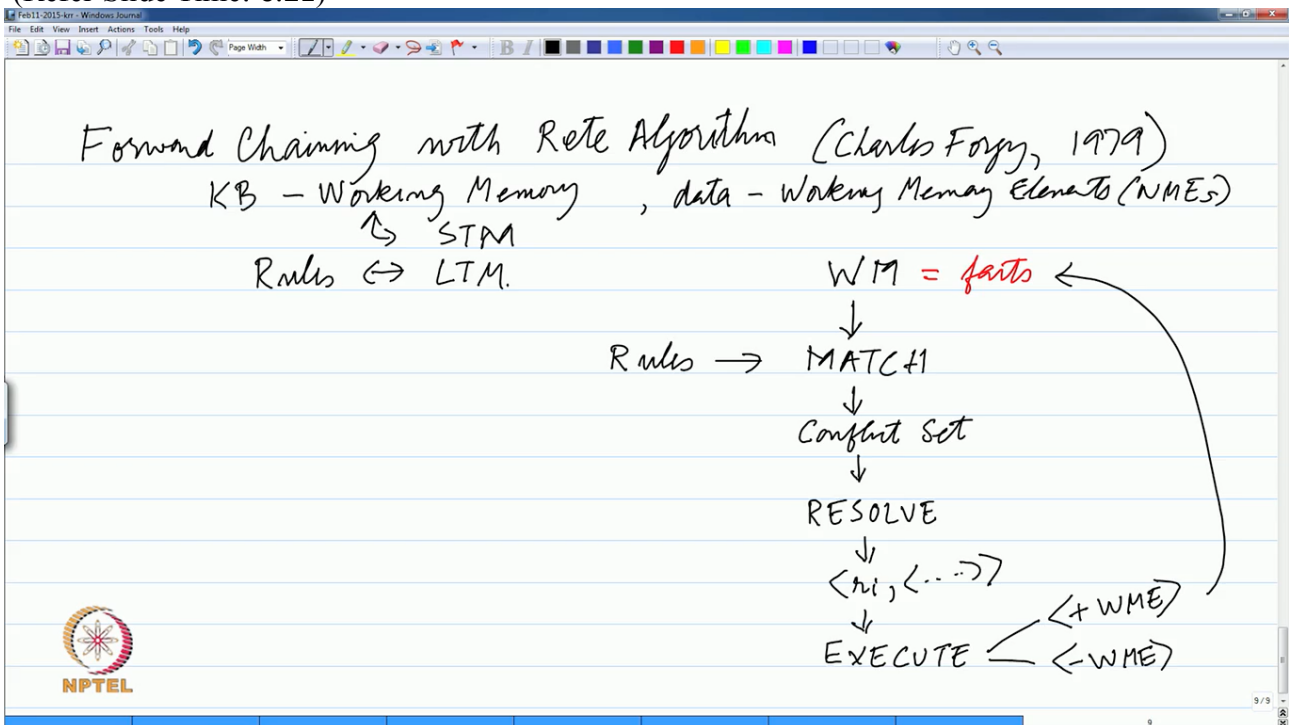
$\langle r_i, \langle \dots \rangle \rangle$



9/9

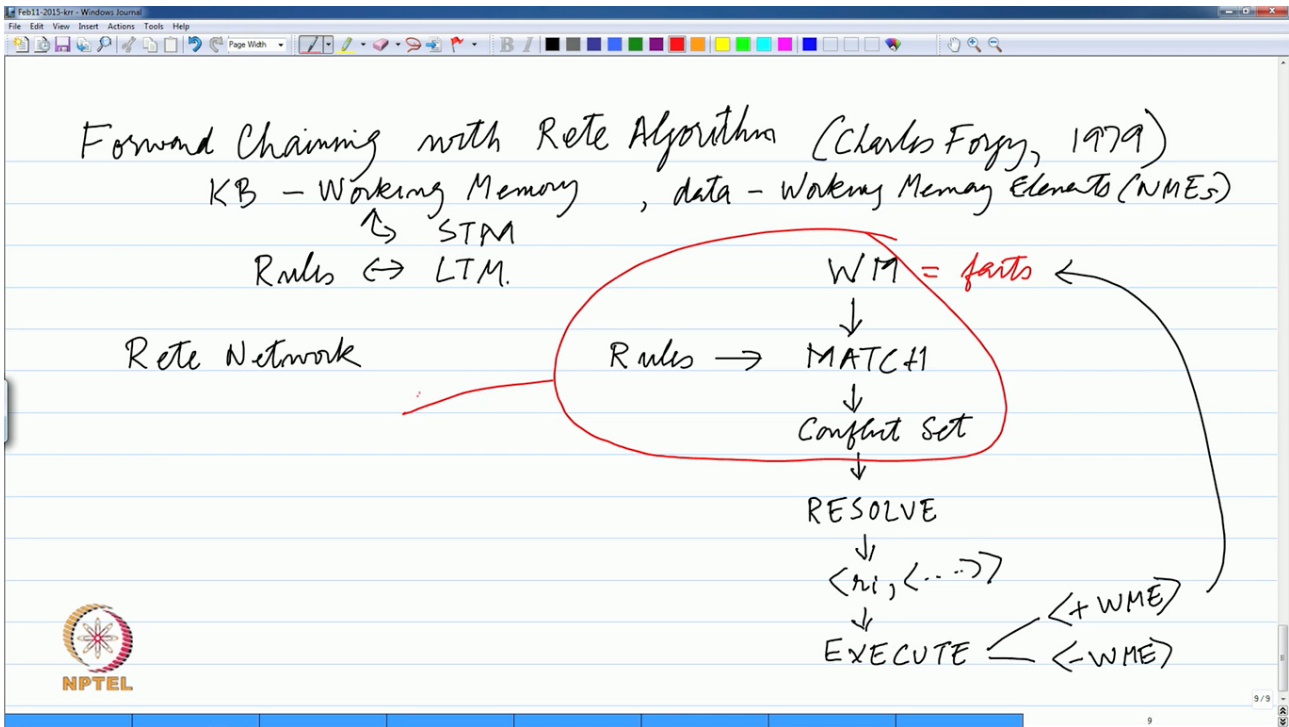
That rule is given to the execute component. Remember it's a three stage process and this produces two kinds of tokens, one is plus working memory element which means you are adding a new working memory element and we will look at the language in a little bit more detail in a later class. Today we are simply focusing on the match part essentially. Or they may produce another token which is a negative token whose meaning is that you want to delete that working memory element from your working memory essentially. So you either add new memory elements or you delete, basically it means that you are making changes into the working memory and therefore you have to go through this whole process again.

(Refer Slide Time: 6:21)



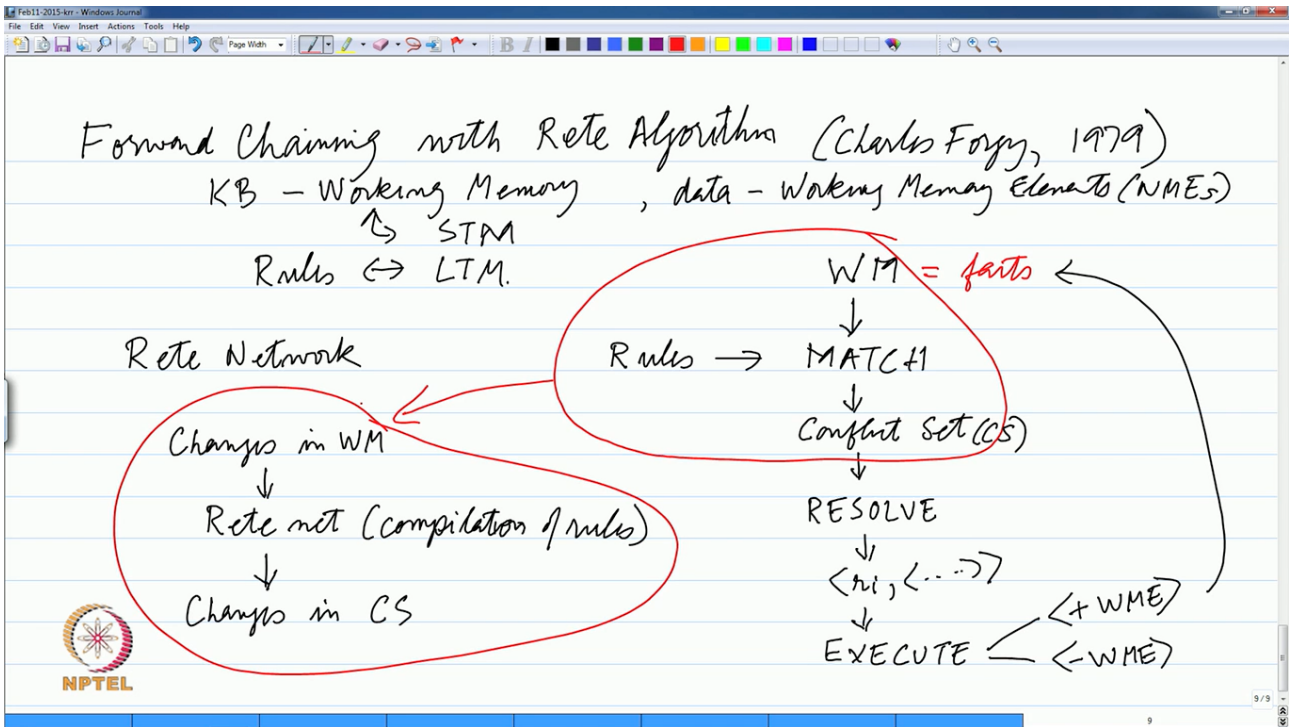
So towards the end of the last class we had said that because we are allowed to delete things, you are forced to do the match all over again essentially. But what the Rete algorithm tries to do is to carry forward, because this word Rete comes from Latin and in Latin it means a network essentially. We will see that essentially what this Rete algorithm does is to organize the rules into a network essentially and that network is called Rete network. It's a kind of redundancy because Rete itself means network but still. So what the Rete network does is that it makes a change into this part of the process.

(Refer Slide Time: 07:27)



It redoes this in a different way. What it does is that, so it's trying, so the goal is to try to carry forward all the match as we have done previously. So remember once you enter do the match process that means you have looked at every rule, try to match it with every piece of data and made a list of what matches with what. Now after the execute step some of that will get disrupted because either new elements have been added to the data which means new rules may match or some elements have been deleted from your knowledge base which means some of the rules which were matching in the previous cycle will no longer match in the next cycle. So which is why we thought we need to do the match all over again but if you are doing brute force match then you see that you have to match every pattern with every data and it's a very expensive process. What the Rete network tries to do is to carry forward as much as we can of the match done in the previous cycle and only account for the changes essentially. So we can see this part as follows. That we accept changes in working memory, the match algorithm has been defined earlier takes a working memory and tells you which rules are matching with that working memory. The Rete algorithm takes changes in the working memory and it feeds them into what we will call as Rete net which is essentially a compilation of rules. So a Rete net is just a representation of rules in a network format. And what this retenet gives us is changes in the conflict set, Remember CS stand for the conflict set. So this is a change that we are making here

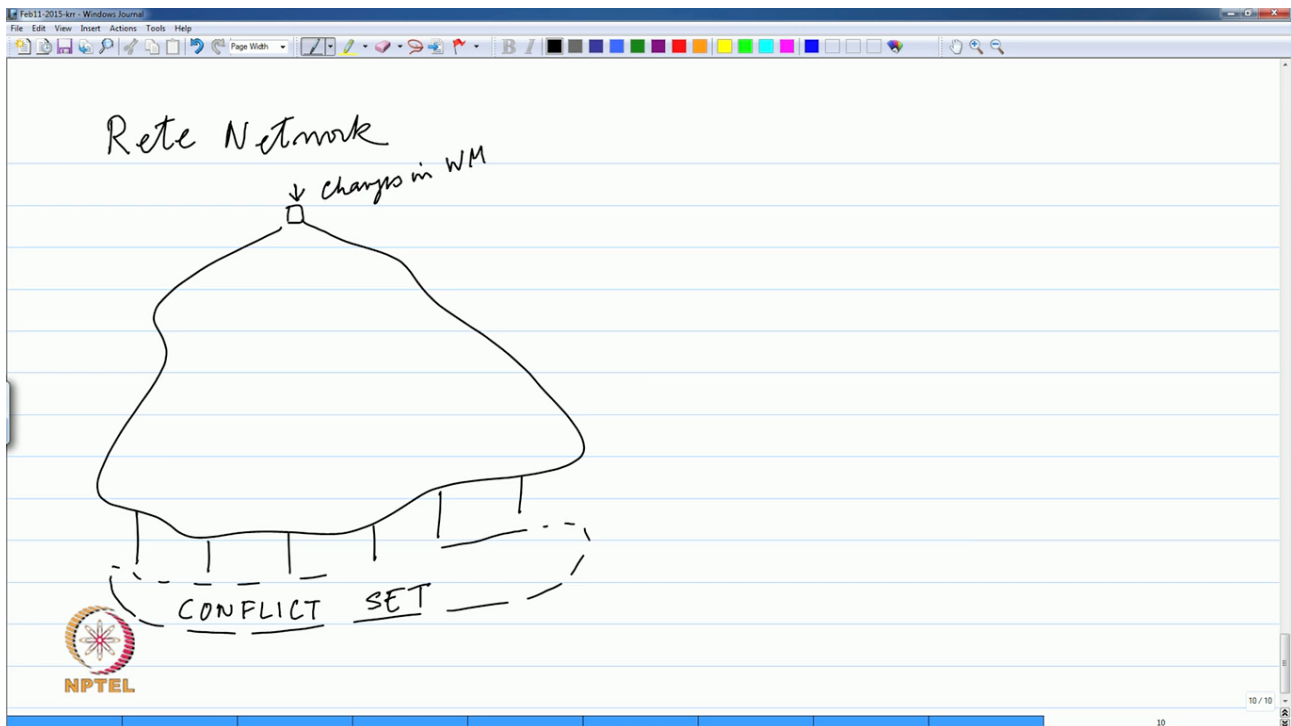
(Refer Slide Time: 9:40)



That we will only look at changes because when we look at changes and these are the changes that we being made here, either a new working memory element is being added or the old working memory element is being deleted and we essentially want to look at what is the effect of these changes onto the conflict set. So it takes as input changes to working memory and outputs changes to conflict set. So let's start by looking at the rete network today and in the next class we will see an example which will make things a little bit clearer.

So i like to think of the Rete network so it's a network with has a root just as a tree has a root and that's a place where we enter changes in working memory. And then it's a kind of a network which is what we will study and in some sense you can see things hanging from below. This is just a way of visualising things which i use. And what is hanging from here is rules essentially, the set which is the conflict set

(Refer Slide Time: 11:39)



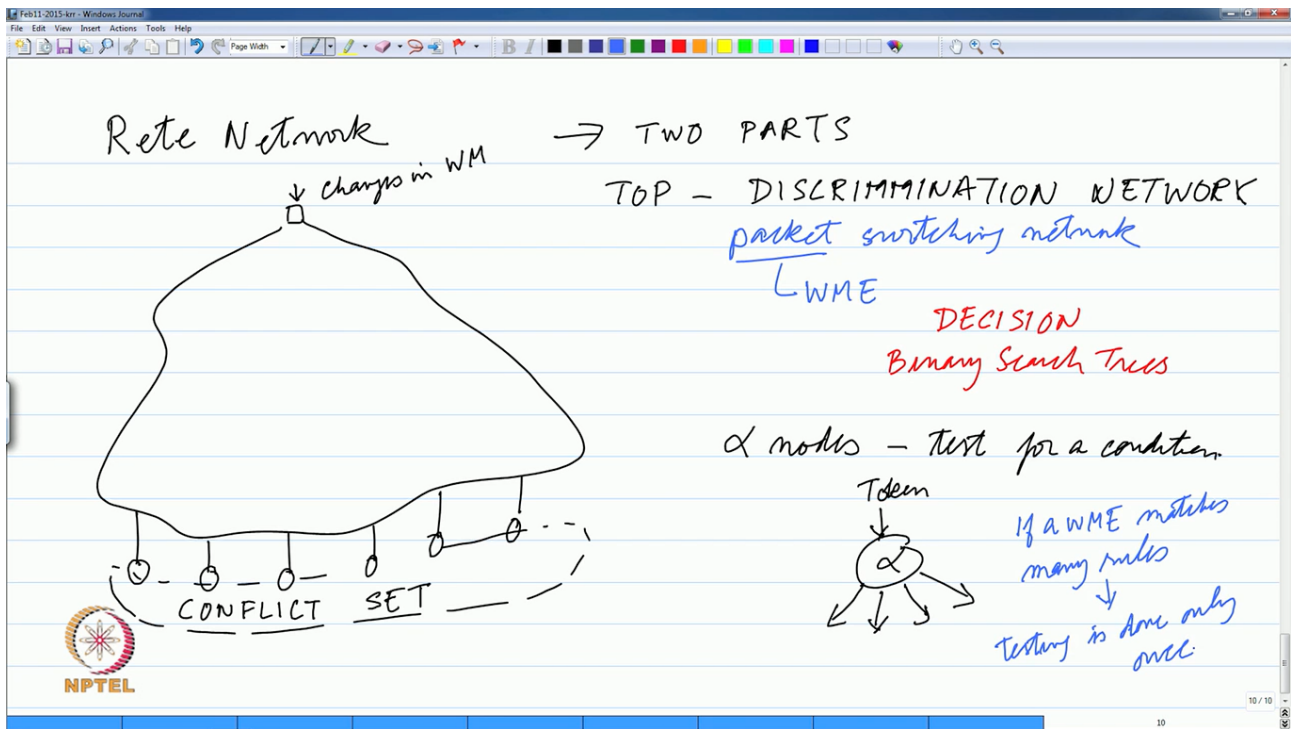
So basically what you have is some rule with matching data, another rule with matching data and so on. And we can think of the network as made up of two parts, the top part is Discrimination network. So basically what this discrimination network does is that it's like a, if you like analogy of networks, it's like a packet switching network and the packet in question is the working memory element and essentially the content of the working memory element is treated as you might say the address of a destination rule that the working memory element is destined to match. So the discrimination network is basically it kind of separates things which have to go to separate places, that's why it's known as discrimination network. Similar names that we use are, if for example you have studied decision trees the same idea, if you put something at the top, it somehow gets distributed to various destinations. The basic idea is similar to what we do in binary search trees or any search trees for that matter. But in binary search tree we have a data element or a key or a token, depending on its value of greater than some test that is being done at that node, it will either go left or right. In higher order search trees, it may have many branches. In decision trees, the search may not necessarily be numerical, it could be a test of some other kind essentially.

So for example you may test for string matching or any such thing, any kind of a test is allowed. And the discrimination network is an idea which comes from decision tree network is that you put a token on the top and what is the token in our case, the token is a working memory element. It may come with a plus sign or a minus sign. First sign will mean that it may go and trigger a rule and minus sign may mean that it may go and cancel a rule which is already matching essentially. But essentially it has to find the rule that it matches, that's the main task of the match essentially. And the top part basically separates the data according to the destination rules that they are meant for essentially.

So these are nodes, it is made up of nodes called alpha nodes. It's a terminology devised by Forgy. What they do is to test for a condition. So an alpha node will get a token, it will do some test for a condition and it may send off the token to one or more of its children's essentially. So the top part of the network can be seen as the tree where you start at the root and gradually you keep branching till you reach a destination essentially. So the destination is a rule that it is meant for. So what's an advantage that we are doing here. One advantage is that if a working memory element matches many rules the testing is done only once.

(Refer Slide Time: 17:20)





Because it is a network and sometimes you are throwing the token down from the root into the network, it will follow some path and that path will lead to more than one rule essentially because the rules are hanging in the bottom of the network essentially so that's the plus point of the Rete algorithm. The other plus point comes from the fact that we are only looking at changes in the working memory which means that whatever is hanging at the bottom of this network, will continue to hang there. So when I say hanging I am just using my own visualisation there. So it's the rules which have already received matching data. So just imagine the process that we are starting this whole process and the working memory is empty. So we only have the network and obviously there is no data so nothing matches so there are no rules in the conflict set. But as you keep putting in data, as we keep putting in the working memory elements one by one, they keep travelling down this network and eventually when enough data has collected for a rule to match then a rule is created at that point essentially. So these are rule nodes which are created when matching data is found.

So the top part, they in some sense, find roots for working memory elements, if you just use the network analogy, they tell the working memory elements where they have to go. The bottom part is assimilative. So these are beta nodes, and what they do is they collect working memory elements for a rule. So if a rule has let's say  $n$  antecedents, that means there is pattern1, pattern2, pattern3 up to pattern  $n$ , or antecedent 1, antecedent 2, up to antecedent  $n$ . It needs  $n$  pieces of data before the rule will match essentially. And what the bottom part of this network is basically it collects data so one piece of data may come from here, another piece may come from different place in the network and the beta node, it essentially collects. What does a beta node do? It checks that shared variables match the same constants. So the rule may have more than one variable and towards the end of this class we will quickly see an example and then we will work out a more detailed example in the next class. So one thing it does, it checks that a shared variable, so if a variable is coming from here let's say  $x$  and it's also coming from here say  $x$ , so it's a part of same rule essentially.

(Refer Slide Time: 21:59)

Rule  $\alpha$  nodes - find routes for WMEs

Bottom ASSIMILATIVE -  $\beta$  nodes  
 - collect WMEs needed for a rule to match

Checks that shared variables match the same constants

11 / 11

So a rule basically needs two patterns which are coming from different directions. You want to make sure that the same data is matching that essentially. I will shortly give you an example. So this may feed to another beta node along with another piece of data coming from here. So one may come from here, one may come from here.

(Refer Slide Time: 22:30)

Rule  $\alpha$  nodes - find routes for WMEs

Bottom ASSIMILATIVE -  $\beta$  nodes  
 - collect WMEs needed for a rule to match

Checks that shared variables match the same constants

11 / 11

So it checks that shared variables match the same piece of data and the JOIN if you want to use the term, is done based on the antecedents. Because you want to collect together all those tokens. So what the beta networks do is that they take the two tokens which are coming from the two parents and put them together to make a combined token. So if a rule has for example four antecedents then there would be three beta nodes which will sort of take four tokens and put them together. So here



we have two beta nodes which are collecting three tokens. If we had one more token, we will have to add one more beta node. And once you have all those tokens which a rule needs then it is simply added to the conflict set. So the top part of a network separates a token essentially according to where they should be going. The bottom part of the network assimilates the tokens that you need for a rule to match essentially.

So let me give you a small example of a rule. The notation may be slightly different from the one we have used in FOL but we will sort of gradually reconcile with that. We will see that these rule based systems are essentially versions of what we are trying to do in logical reasoning essentially. So i may have a rule which looks like this. So let's say rule and let's say it has a name and it's called total marks and it may have certain so instead of if I am saying rule essentially you must now learn to navigate between these different notations. This rule name I am using here but if you look at a book like Reckman and Levis which we are also referring, they use if essentially but it's the same thing.

So there is a rule and we are allowed to give rule name which of course in flogic we don't really talk about. But let's assume that we have. And then we have the antecedents. So let us say that this is the notation that I use. So the data is or the working memory elements are described in terms of something like a structure that we use in C. So you might say student, in some sense this is the class name. And then i will use the notation which is used in a language called OPS5. It was one of the early language which was devised for rule based systems and something which I am familiar with. So the first pattern has a title called student and it has a so in this notation we have a class name followed by attribute name and followed by value, attribute name, value and so on. This is how a working memory element is described essentially. It has a structure. So the class name is Student here, the attribute name is let's say name and I am using a variable here. So let me use a notation of OPS5 which is to put it in angular brackets. So again this is equivalent to saying a variable called n essentially. It's just a matter of notations essentially, separating variables from constants essentially. We need something. So in first order logic we said we will use a question mark. In language like OPS5 we use angular brackets to separate the variables.

Then if you say Marks and let's say we have records in this format. Subject let's say Physics it's a constant, notice that Physics is a constant, n is the variable essentially. Student, an attribute I have used the name Student here and Student is a class name, there is no conflict, they come from separate namespaces and there is no confusion here so you must learn to recognise that there is no confusion here. It's just an attribute name as far as this class called Marks is concerned. So Marks is saying in this subject this student has so many marks. So it has got three attributes and three values. The student must be the same student, so i use the same variable name. So if I use student name n then in Marks I am looking for his marks or her marks so that the variables must match exactly and this is one of the things those beta nodes do is that when they accept these two tokens, one is the token called Student and other is the token called Marks. It will make a check that if they have a shared variable then the shared variable must match the same data. So if I am talking about student called Satish then the marks must also be of Satish. Otherwise you will all start complaining about the grades that you are getting. And the third is marks so and that's the value m1 or mp you want to say for Physics.

(Refer Slide Time: 29:11)



Example

Notation - OPS5 →

(Rule total-marks

$\langle \underline{\text{student}} \text{ ^name } \langle n \rangle \rangle$

$\langle \text{Marks } \text{ ^subject Physics } \underline{\text{student}} \langle n \rangle \text{ ^mark } \langle m1 \rangle \rangle$

$\langle \text{Marks } \text{ ^subject Chemistry } \text{ ^student } \langle n \rangle \text{ ^mark } \langle m2 \rangle \rangle$

→

(Make  $\langle \text{Total } \text{ ^student } \langle n \rangle \text{ ^total-marks } \langle m1+m2 \rangle \rangle$ )

WME

(class name ^Att Value


^Att Value

?n - VARIABLE

:

:

:



NPTEL

12 / 12

so the rule has three patterns one says that I am talking of Student who we will call as n and then for physics n should have some marks let's call it m1. For chemistry that student should have some marks m2. Then i want to basically create a record which says that the total marks of student n is m1 plus m2. So it is the job of the beta nodes to collect these three working memory elements so somewhere just imagine that we have this Rete network which has other rules of other kinds for example for grading there might be rules and all kinds of things might be there. And there we produce this three pieces of data, that there is a student called Suresh, then Suresh got let's say 70 in Physics. That's one data record. Then Suresh got 65 in Chemistry that's is second data record let's say. The moment we get these three pieces of data they will flow down the network and then this rule called total marks will fire so we will have once we get this data, one element added to the conflict set which says that the rule total marks is ready to fire and it has got this three pieces of data that it is matching. And like that if there are 50 other students, there may be 50 other rules and then based on the problem solving strategy we will take one using resolve and so on and so forth.

So the basic idea of a Rete network is it's a two stage network. The top part basically discriminates between tokens, it sends them out into different directions. The bottom part if assimilative it pulls together all those tokens which are needed for a rule to match and at any given point of time in the rete network there is a conflict set existing rules which are matching with data essentially. So in the next class maybe we will look at an example to make this idea more concrete.