Artificial Intelligence: Rete Algorithm – Example

Prof. Deepak Khemani

Department of Computer Science and Engineering

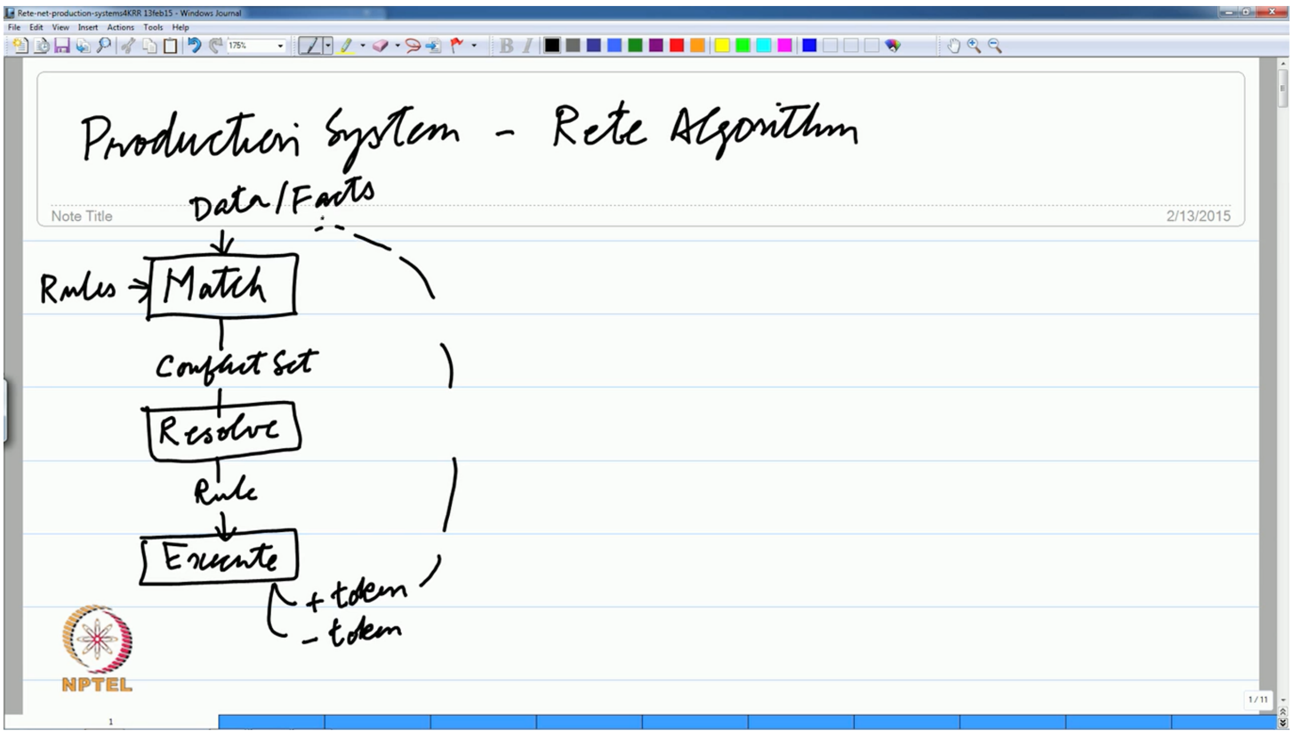
Indian Institute of Technology, Madras

Module – 03

Lecture - 09

Okay so we are looking at production systems which is a form of forward chaining or forward reasoning and in particular we are looking at an algorithm called Rete algorithm which uses the rete net as a structure which is what we were looking at. So what I am going to do is show you an example of rete network today. But just to make sure as to what we are talking about. So remember that in production systems the basic idea is that we have a three phase cycle, we have match phase. What the match phase does is takes rules as input and data or facts as input and it produces set of rules which are matching with data and we had called that set as conflict set. And this set is given to a phase called Resolve which selects a rule from the conflict set which is given to Execute. And what execute phase does is produces either a positive token or a negative token because we have said that we are allowed to delete statements from our knowledge base and then these are fed back into the network.

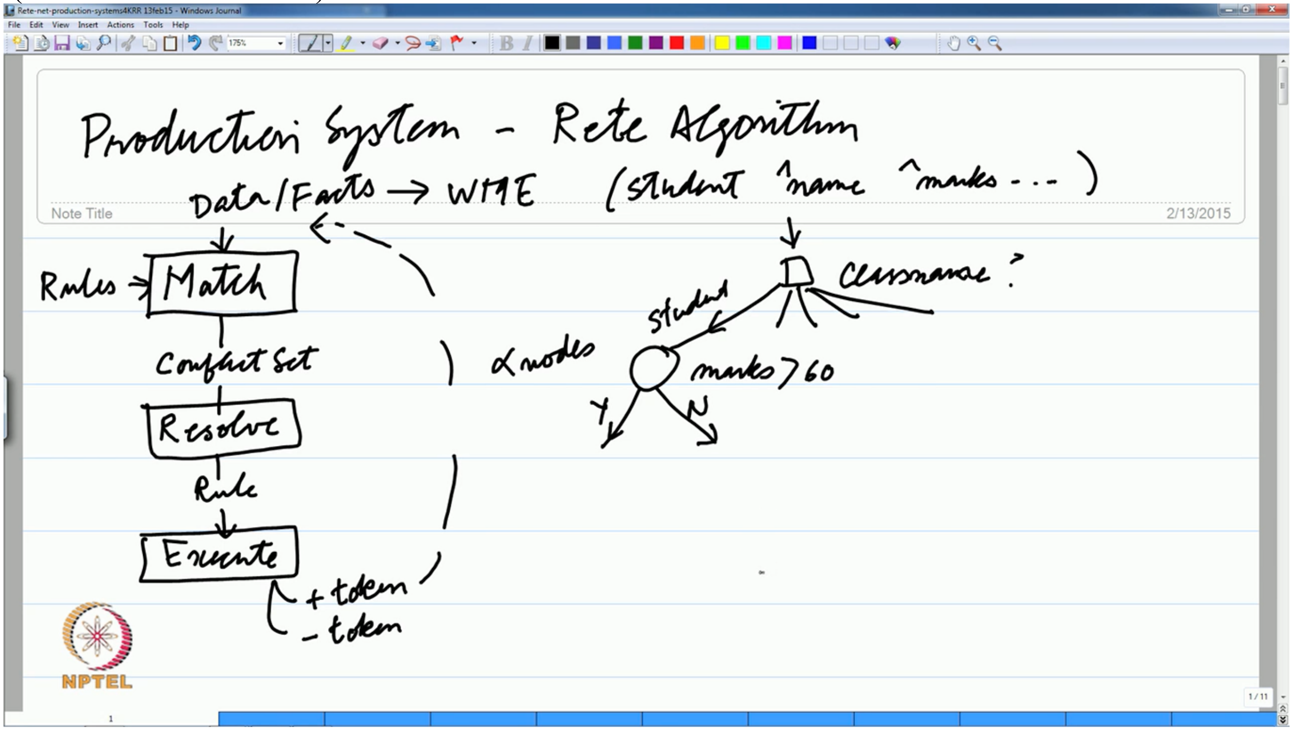
(Refer Slide Time: 02:15)



The basic idea of a rete network was that instead of doing this match all the time which we saw was expensive, we want to do it once in the start and after that any changes in the working memory or changes in the knowledge base we want to see that what changes are induced in the conflict set. So we had this notion of working memory so the facts are in working memory, these are called working memory elements and typically it may look like this. There is a student, so there is a class name, think of it as an object which comes from certain type, so the student is a certain type and then you may have attribute and values. So you have attribute called name, you may have an attribute called marks and so on. Now what the rete algorithm does is you start by entering a working memory element and then at each node you ask a question which will tell the system what is the working memory about essentially.

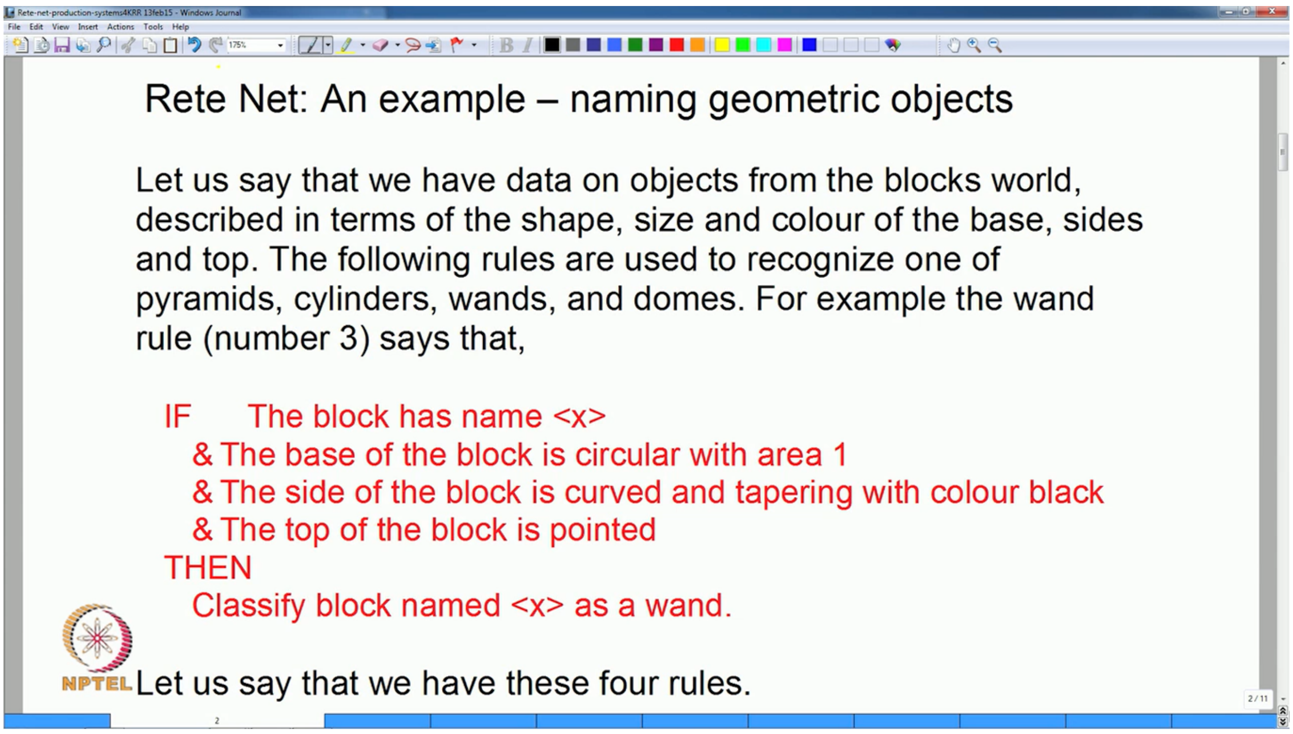
So for example at the topmost level there might be a question like class name. So for each class that we have in our knowledge base, so for example student may be here and some other kinds of data may be there. So if we see a token which is of the type student then it should come in this direction. Then we may ask some other questions essentially. So for example we may ask something like marks greater than 60. Then what happens is that whichever are the tokens with marks greater than 60 would go down the left branch and others would go down the right branch. So the top half of the network we said was a discrimination network. It sort of separates the data and directs it towards the rules that they are going to match essentially. So these kinds of nodes were alpha nodes.

(Refer Slide Time: 5:02)



and then there were beta nodes which basically pulled in data that a rule requires and essentially supplies it to a rule and says this rule has matching data, so that's what we had looked at in the last class. So today we will look at an example first and depending on how much time we have we will look at the language itself in a little bit more detail and what it allows us to do essentially. So i have sort of copied from my book some example so I won't write it all over again. So we will essentially discuss it with a pre written piece of data.

(Refer Slide Time: 5:53)



Rete Net: An example – naming geometric objects

Let us say that we have data on objects from the blocks world, described in terms of the shape, size and colour of the base, sides and top. The following rules are used to recognize one of pyramids, cylinders, wands, and domes. For example the wand rule (number 3) says that,

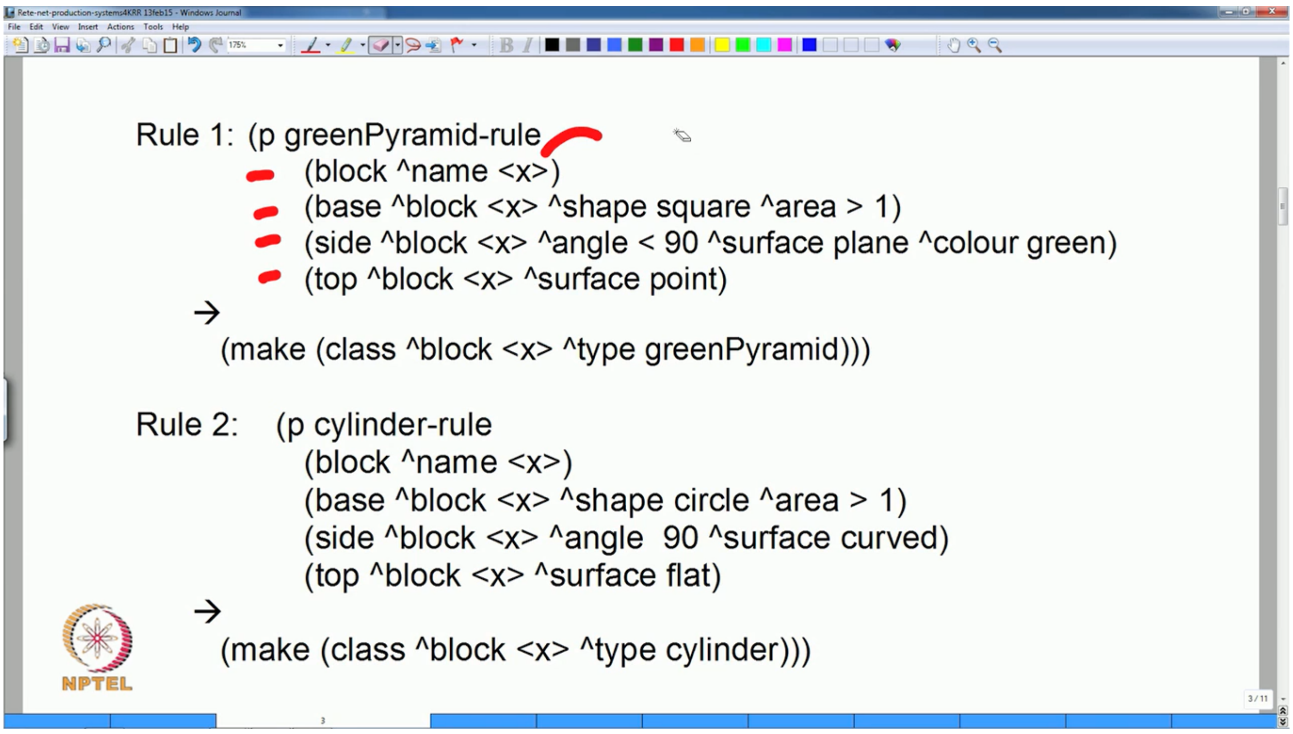
```
IF      The block has name <x>
   & The base of the block is circular with area 1
   & The side of the block is curved and tapering with colour black
   & The top of the block is pointed
THEN
   Classify block named <x> as a wand.
```

Let us say that we have these four rules.

So, let's say we want to write a small program to recognise or classify geometric shapes. So we may have things like pyramids or cylinders or wands or domes, let's say we have four kinds of language which are described using certain attributes and values and given our knowledge base we want to identify how many, which van we have, which pyramid we have, such kind of stuff essentially. So for example a rule for identifying a van as you can see in this red says that every block has a name so we want to classify a block, block a is a van, block b is a cylinder and things like that. The rule for van says that the base of the block must be circular with the area of size 1 some units. The side of the block is curved with tapering with colour black. The top of the block is pointed. If we have a block of this kind, then call it a wand.

So let us say we have these four rules. So there is rule, so here are two on the next page, so there is this rule called green pyramid rule so we are talking about green pyramid actually. Now the first, each of these things that we see here these are the four patterns that this rule is looking for, these are the antecedents of the rule essentially. And each of these patterns is describing some working memory elements which starts with the class name. So we have class names like block and base and side and top. And then the attributes and values. So remember that the thing with the hat is the attribute name and following that is a value essentially. And remember that something like this is a variable.

(Refer Slide Time: 8:20)

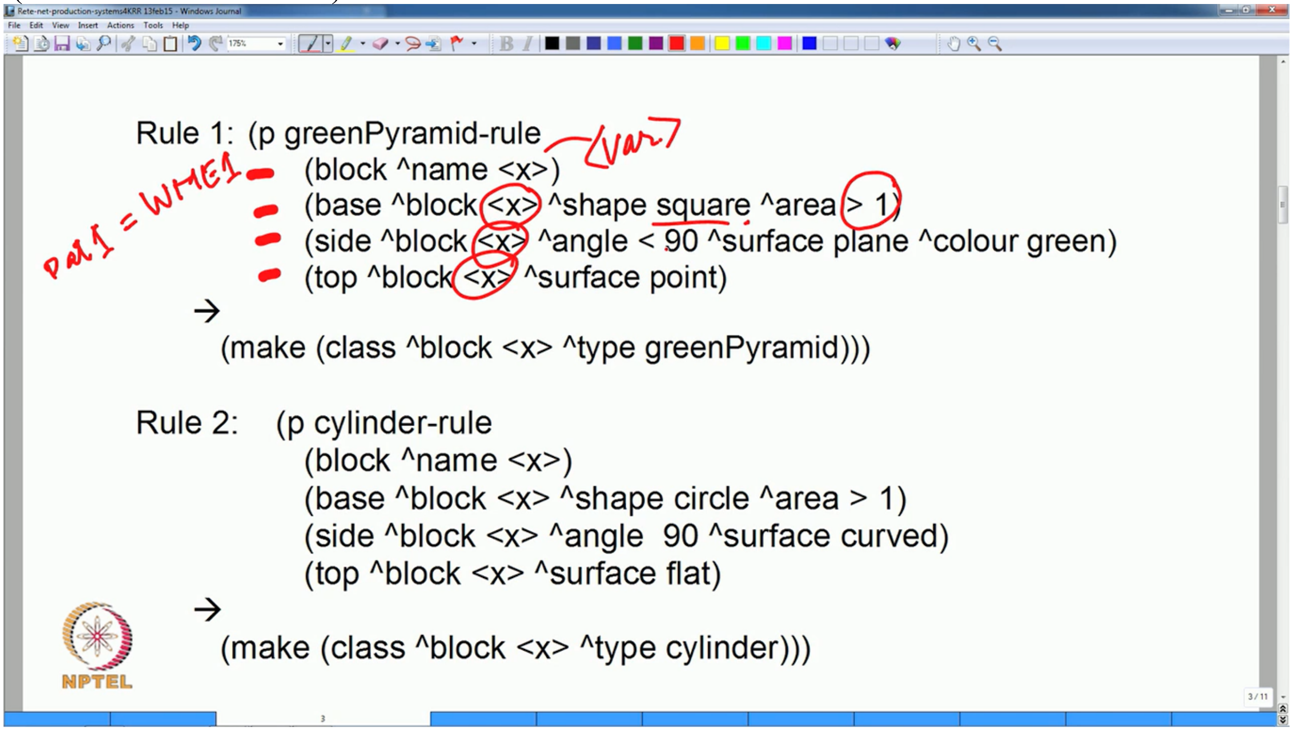


Rule 1: (p greenPyramid-rule
(block ^name <x>)
(base ^block <x> ^shape square ^area > 1)
(side ^block <x> ^angle < 90 ^surface plane ^colour green)
(top ^block <x> ^surface point)
→
(make (class ^block <x> ^type greenPyramid)))

Rule 2: (p cylinder-rule
(block ^name <x>)
(base ^block <x> ^shape circle ^area > 1)
(side ^block <x> ^angle 90 ^surface curved)
(top ^block <x> ^surface flat)
→
(make (class ^block <x> ^type cylinder)))

so this is the variable, anything in the angular brackets is a variable. So the first rule says that i must have in my knowledge base one working memory element. So these are working memory so this is working memory pattern1, and so on whose type is block and it has an attribute called name which has some value x. Then the remaining three patterns must have somewhere the same value x. So wherever we repeat this x they must have the same value. So that’s the whole idea of sharing a variable. Then the second pattern says that the type of the working memory element is base and it is we are talking about a block whose name is x, we are not talking about naming it x but the attribute name is block whose shape is square. Now this square is the constant, it’s not a variable because it does not come under angular brackets. And then we are allowed to do things like logical tests whose area is greater than 1 essentially. The side of the block has angle less than 90. the surface is plain and the colour is green

(Refer Slide Time: 10:00)



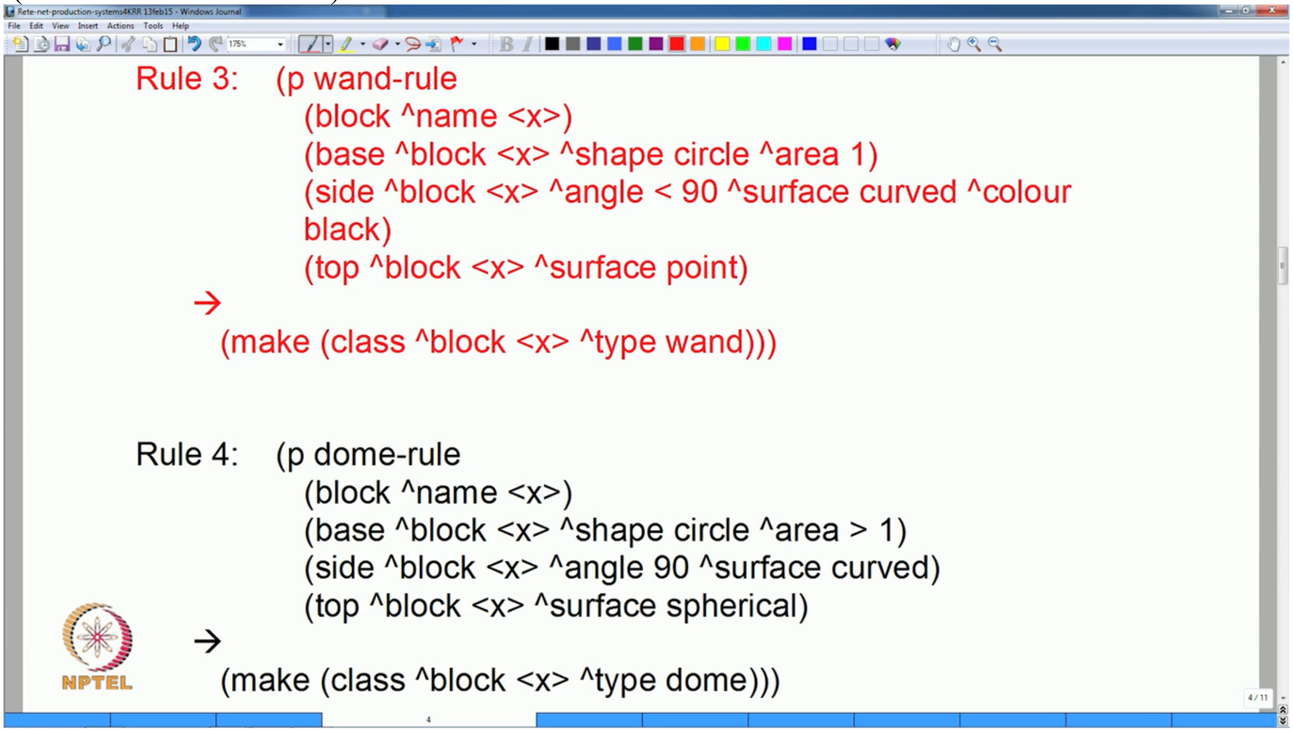
Rule 1: (p greenPyramid-rule
(block ^name <x>)
(base ^block <x> ^shape square ^area > 1)
(side ^block <x> ^angle < 90 ^surface plane ^colour green)
(top ^block <x> ^surface point)
→
(make (class ^block <x> ^type greenPyramid)))

Rule 2: (p cylinder-rule
(block ^name <x>)
(base ^block <x> ^shape circle ^area > 1)
(side ^block <x> ^angle 90 ^surface curved)
(top ^block <x> ^surface flat)
→
(make (class ^block <x> ^type cylinder)))

The top of the block is pointed essentially. So all these conditions are satisfied then we say that we have a pyramid. The right side of this is a action so we are saying we make the class block x type b essentially. So we are not really bothered about the right hand side here. So may be in next class we can see what is the kinds of actions that we can do.

So likewise the second goal says which is a cylinder goal which identifies the cylinder. So the three things, the second, third and the fourth patterns I got. Second pattern says that its base is circle. And area must be greater than one. The third pattern says that the side must be of the angle 90 degrees and the surface must be curved. And the last pattern says that the top must be, the surface must be flat as opposed to the point which was the case for the pyramic and the wand essentially. So i want to show how, i will show you two more rules and how those four rules what is the rete network that we construct. So the rete network is basically a compilation of rules, it has nothing to do with data. The moment you have decided a set of rules, then you have a rete network. And that's why Simon they called it long term memory. It is kind of the knowledge that problem solving agent has.

(Refer Slide Time: 11:35)

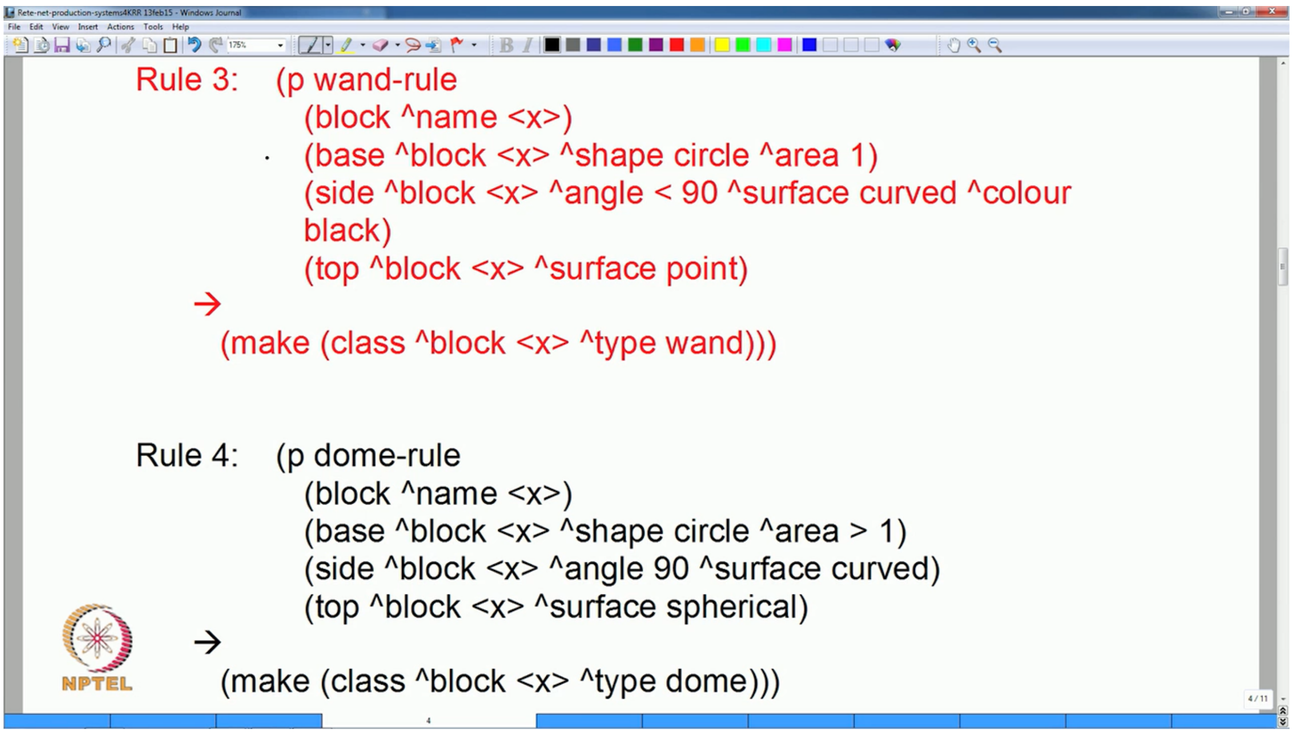


```
Rule 3:   (p wand-rule
              (block ^name <x>)
              (base ^block <x> ^shape circle ^area 1)
              (side ^block <x> ^angle < 90 ^surface curved ^colour
              black)
              (top ^block <x> ^surface point)
       →
          (make (class ^block <x> ^type wand)))


Rule 4:   (p dome-rule
              (block ^name <x>)
              (base ^block <x> ^shape circle ^area > 1)
              (side ^block <x> ^angle 90 ^surface curved)
              (top ^block <x> ^surface spherical)
       →
          (make (class ^block <x> ^type dome)))
```

so the third and the fourth rule are the wand rule and the dome rule. I have coloured the wand rule in red because when I show you the network you should look out for the fact that these four patterns that are needed for this rule to match. What are the paths they will take in the network essentially? So those paths I will show in red. So you must remember what the wand rule is and then we will see how it is chosen up in the network essentially. So the wand rule is similar. The first rule in all these four rules simply says that we have a block called some name essentially, just identifying the block.

(Refer Slide Time: 12:19)



```
Rule 3:   (p wand-rule
            (block ^name <x>)
            (base ^block <x> ^shape circle ^area 1)
            (side ^block <x> ^angle < 90 ^surface curved ^colour
            black)
            (top ^block <x> ^surface point)
        →
          (make (class ^block <x> ^type wand)))


Rule 4:   (p dome-rule
            (block ^name <x>)
            (base ^block <x> ^shape circle ^area > 1)
            (side ^block <x> ^angle 90 ^surface curved)
            (top ^block <x> ^surface spherical)
        →
          (make (class ^block <x> ^type dome)))
```

The second rule says that shape is a circle and area is 1. The third pattern says that angle is less than 90, surface is curved and the colour is black so let's say wands are always black. And the last one says that the surface is a point.

So likewise we have a dome rule. You might have seen the Sanchi Stupa and something like that. It says that the base is a circle with area greater than 1. The side starts with an angle of 90 and the surface is curved. And the top is a spherical surface essentially. So that's something we would call as a dome.

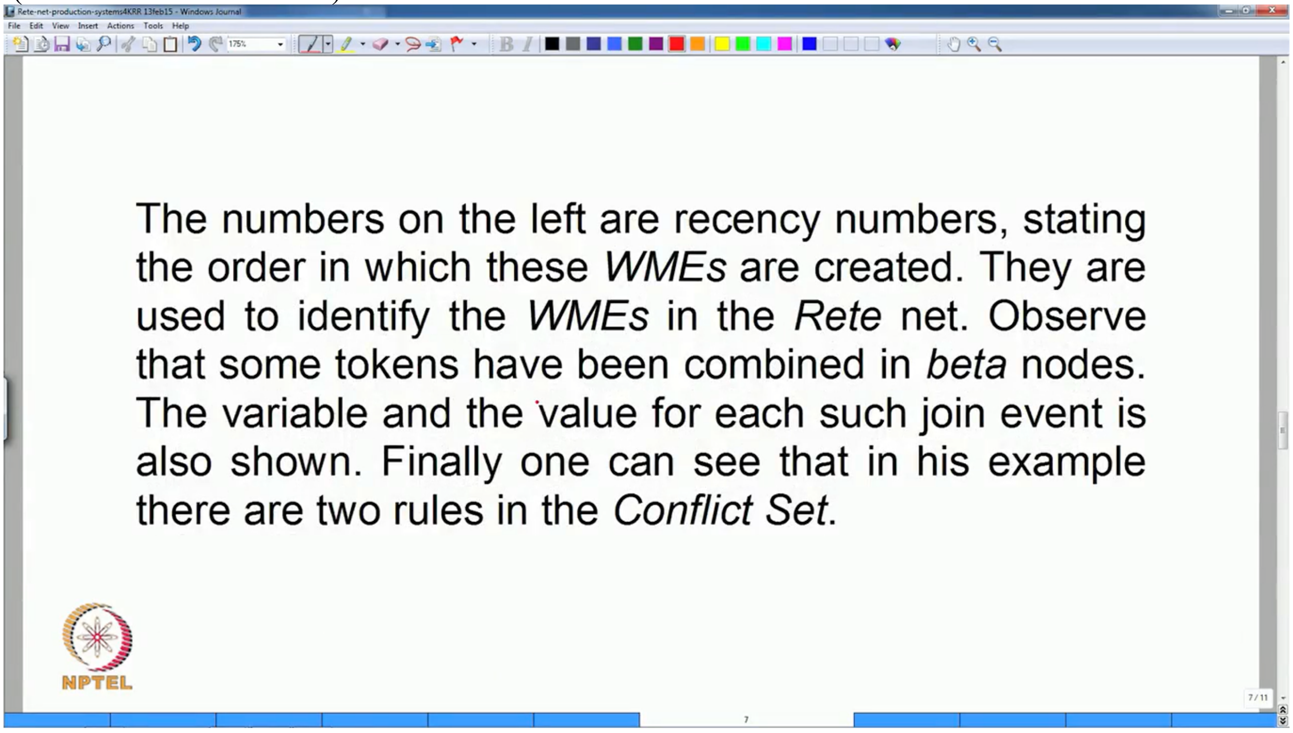
So we construct Rete net for the rules above. So this is basically, I have just copied from my book. Now the important thing is that there is no unique Rete net corresponding to a set of rules. Because essentially what you want to do is to test for various conditions and you could test in any particular order. The goal that we would have is to minimise the number of tests and to make as smaller network as possible essentially. So you could choose any test in any order and construct different networks and you could also combine beta networks in any order because you know each of the rules we saw had four patterns, you could combine 1 and 2 and then 3 and 4 and then combine patterns in any particular order.

So the objective would be to share as much as tests essentially. So if the different rules are testing for the same thing, in our case for example the first pattern is always the same so you must, so for every working memory element you must apply the test only once.

(Refer Slide Time: 14:42)

So once you construct the network we will also insert these working memory elements into the network to see how it works. So the working memory elements or the data, so each of them is like a record in the database you might say essentially. The first one says I am talking about a block such that base of block A is shape square and area is 20. Base of block B is circle and area is 20 and so on and so forth. We want to see that given this data which of the four rules match essentially. That's our main interest at this moment. So the numbers on the left that you see here, these numbers, these are timestamps and we will see later in another class that timestamps can be used in deciding what is the problem solving strategy that you are using essentially. Otherwise they are of no interest to us at this moment essentially.

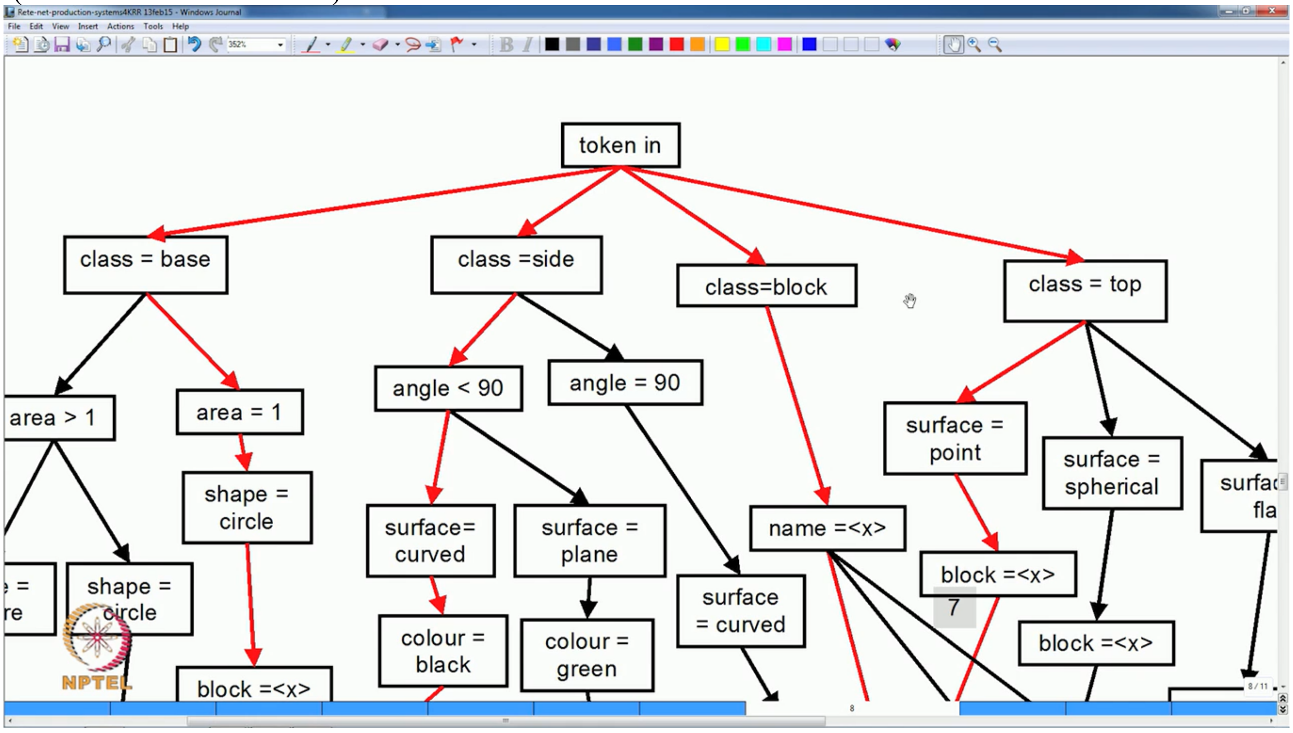
(Refer Slide Time: 15:48)



The numbers on the left are recency numbers, stating the order in which these *WMEs* are created. They are used to identify the *WMEs* in the *Rete* net. Observe that some tokens have been combined in *beta* nodes. The variable and the value for each such join event is also shown. Finally one can see that in his example there are two rules in the *Conflict Set*.

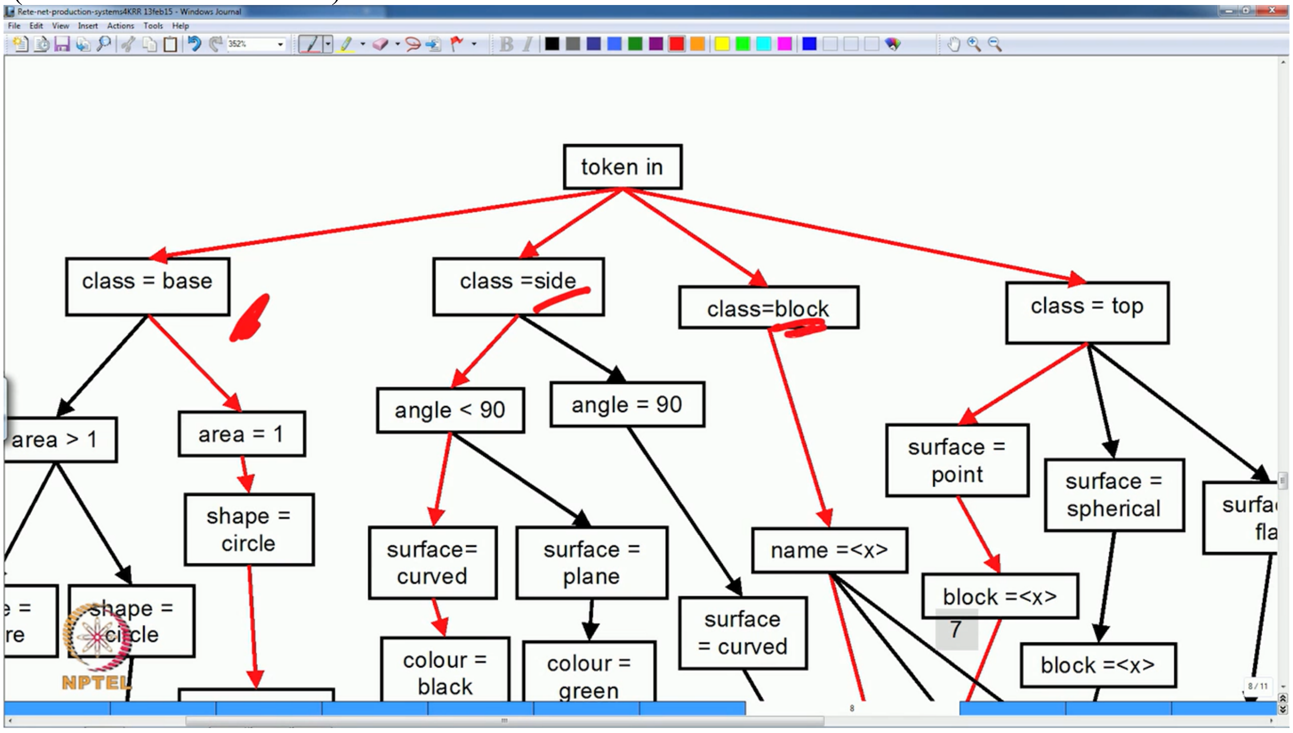
so the numbers on the left are regency numbers which is what this community calls it, stating the order in which these working memory elements were created. They are used to identify the working memory elements in the Rete net. So we will identify in the diagram that we will draw each such element by its recency or timestamp so that we don't have to write the full thing there. And we will see that some tokens will get combined in beta nodes. And then finally we will see in the network that there are two rules which are matching with this data. So here is the network.

(Refer Slide Time: 16:52)



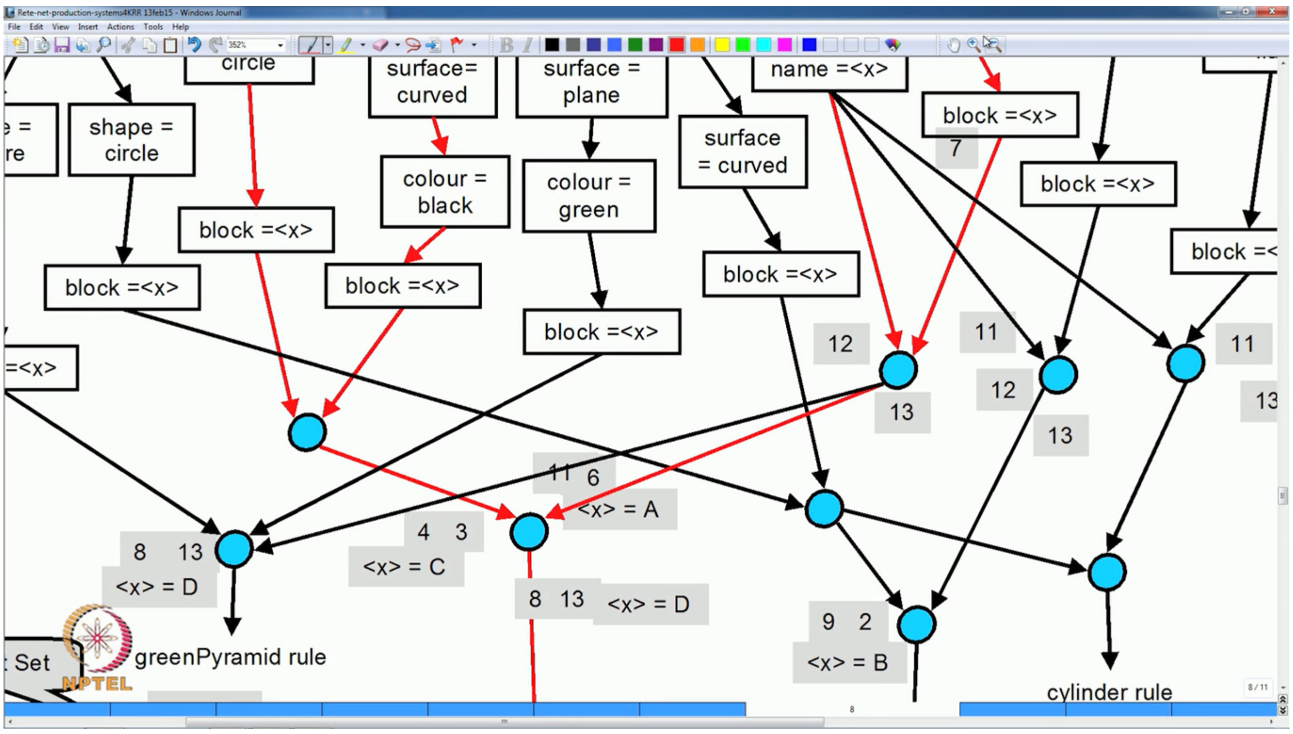
This is legible or should I enlarge it a bit more. This is fine. So you can see from the top, the root is where the token comes in and the red lines are the ones which will match, which basically show you what are the patterns that we are looking for, the wand rule. So it needs four patterns and these all four are of different types. So you can see here that the type of class is, here the class is a base, here the class is a side, here the class is a block and here the class is a top.

(Refer Slide Time: 17:34)



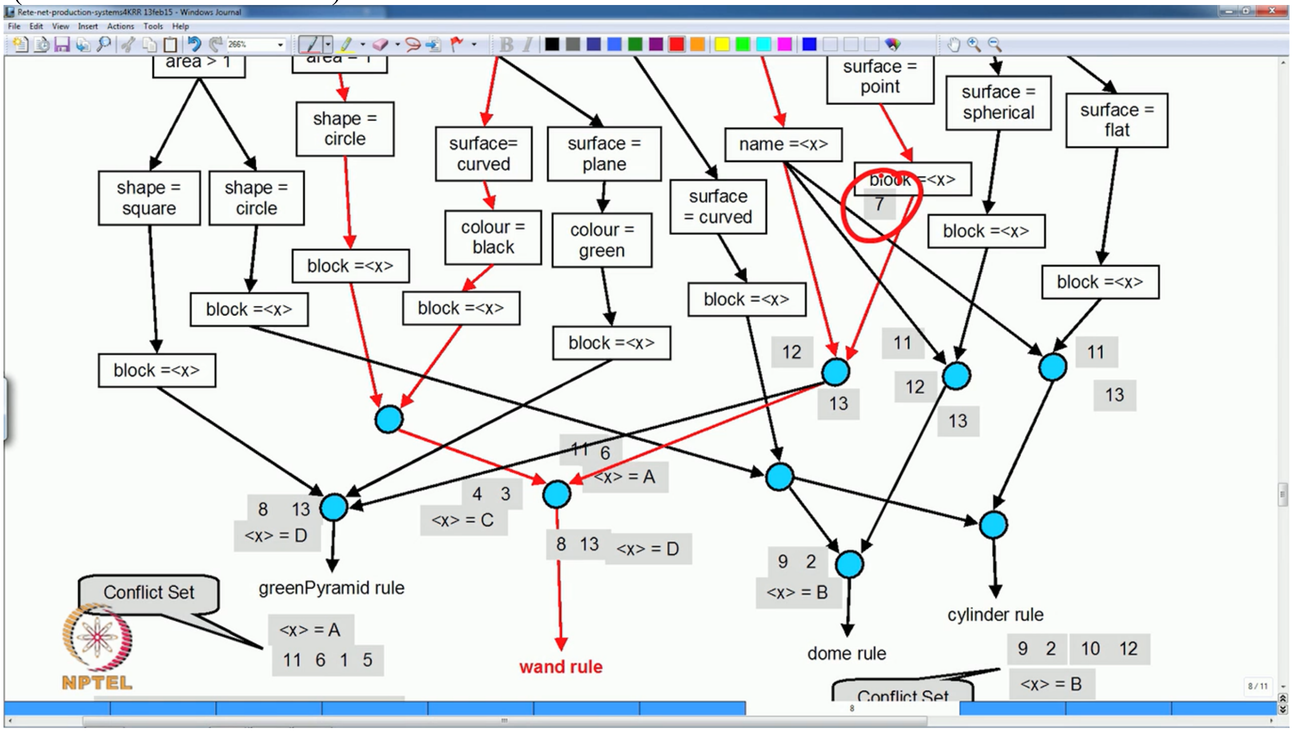
So straightaway we can see that tokens of different classes will go down different paths of the tree essentially. And then we have tests like, so if you look at this class is equal to side part here, you can see that there is a test which is angle is less than 90 and angle is equal to 90. Now how did we arrive at this test. We arrived by looking at the four rules, there were four rules, some of them wanted angle equal to 90 and some of them angle less than 90 and so we have only these two tests essentially. If there were other rules stating that for example if there was a block which said that the angle was 45 then we would have a third branch essentially. So this network is created by looking at the rules essentially. So it's a compilation of the rules in some sense. So for each thing we have, you know you go down you have another test, you go down you have another test and then the blue nodes if you see here these are the beta nodes essentially.

(Refer Slide Time: 18:45)



so they are accepting two tokens and combining back together essentially. So the number which you see in grey for example this number 7.

(Refer Slide Time: 19:06)

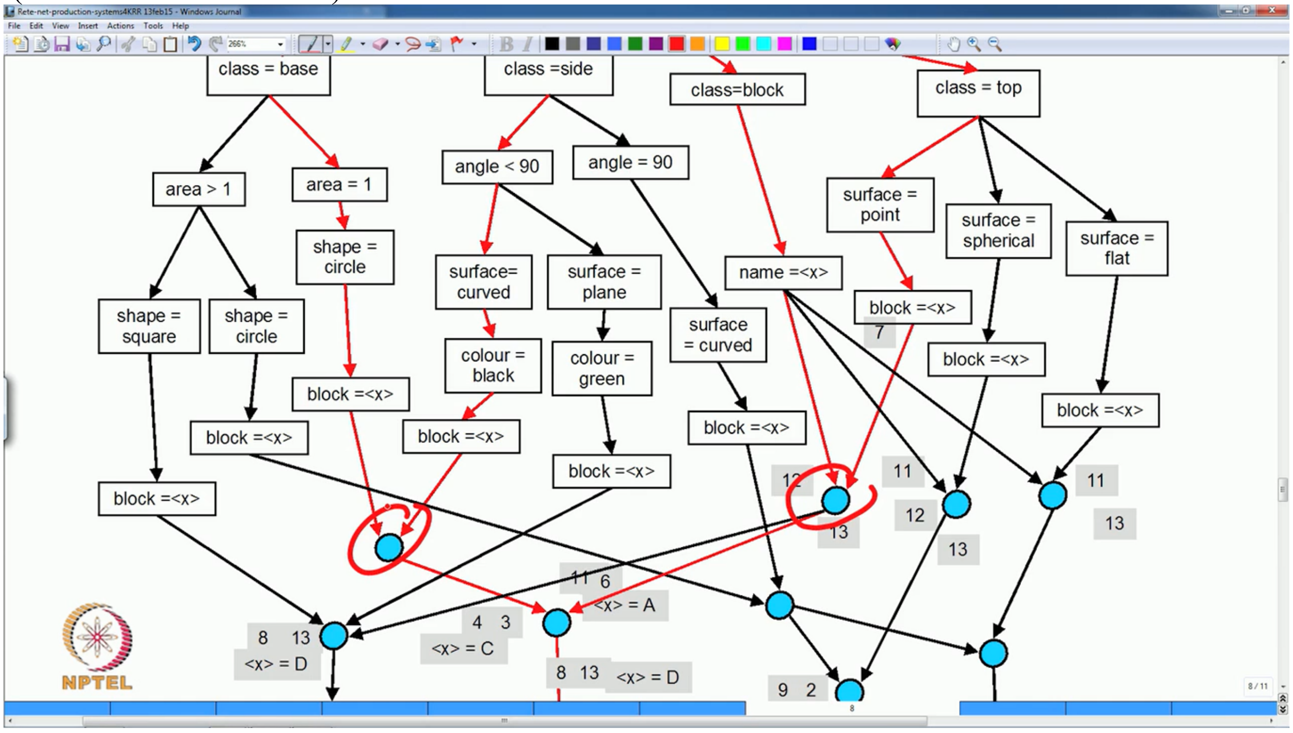


refers to the working memory element numbered 7 and it has come up to this point and it is not able to go down any further because it is probably looking for, to go down further to this beta node here it needs a matching element to come from the other side, from the other branch. And only when the matching element comes will this working memory element which is stuck here at this place go down to this circle node here and then we will have the combined pair of tokens essentially and this process will continue. So essentially when four tokens come and meet at a point because in our example each node has four patterns so once four tokens arrive then a rule would be ready to fire essentially.

So which is happening as you can see, so just look at the wand rule. I wonder whether you remember the wand rule but essentially these red lines are the wand rules. So area is 1, base the shape is circle, so the area of base is 1 and shape of base is circle so this branch that you see is talking about the base essentially and it is a block whose name is x. So this x is essentially we are using simply to say that we are talking about the same block. So if there is a block named a then it can only combine with another block named a which is coming from a different direction. That x is equal to x must match. That's what this beta node is doing, that if there are two variables which have the same name in the two branches then they must match the same value essentially. So the wand rule will have that.

Then another thing is angle of side is less than 90 the surface is curved the colour is black and of course the block has a name which must match the other. There are two more patterns which are coming, one is the class is a block whose name is x, this is the simplest pattern which is there is all the four. And the fourth pattern is that the top of the wand rule, the surface is a point essentially. So that's the only thing essentially. If you see that four red things they first meet at these two nodes.

(Refer Slide Time: 21:30)



To form pairs of two tokens and finally they meet at this node to say that the rule is matching. Now as it happens with this particular data we don't have four matching working memory elements. But we have for the green Pyramid rule as you can see here there are four elements. So x is equal to a. In all those four working memory elements if you were to look at the working memory elements 1,5, 6 and 11. their name is A, they are talking about a block which is A. And all the test that the pyramid rules says that the side is sloping and must be green and all those tests are satisfied. So the four token arrived here and now they have formed this token and this rule has now gone into the conflict set essentially.

(Refer Slide Time: 22:27)

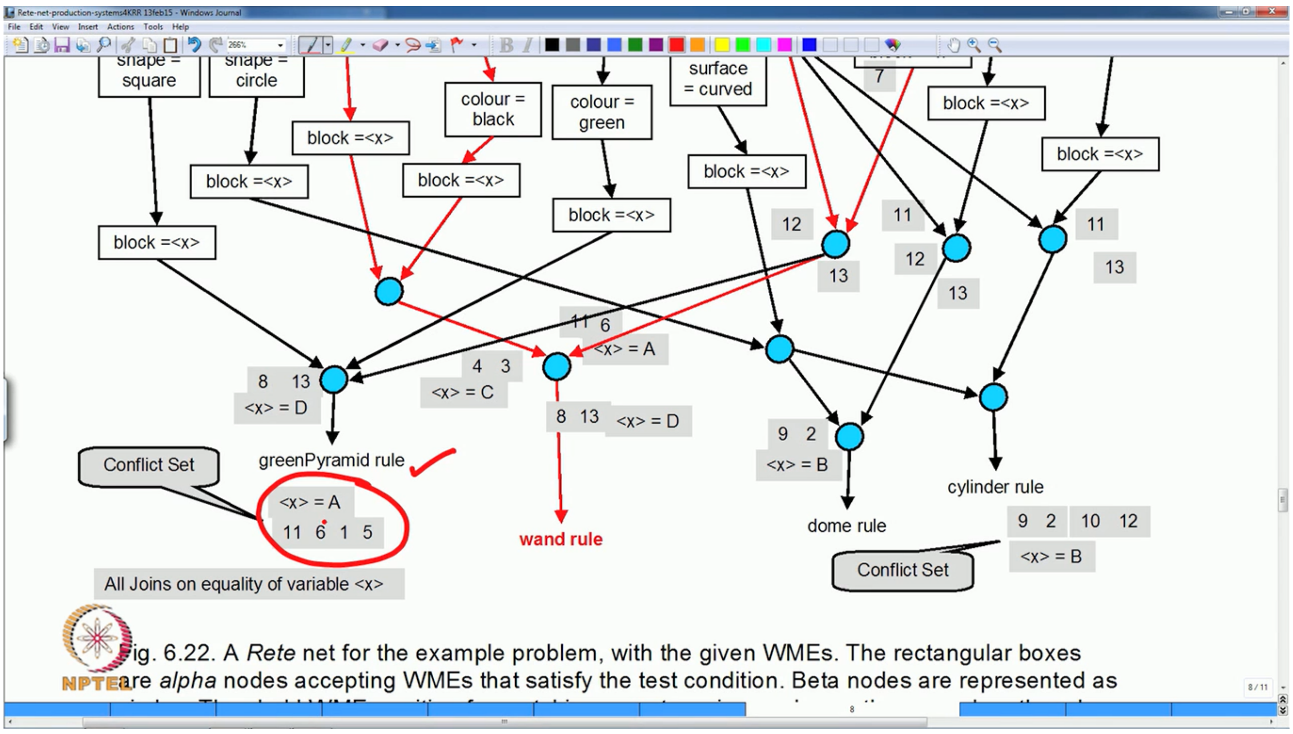


Fig. 6.22. A *Rete* net for the example problem, with the given WMEs. The rectangular boxes are *alpha* nodes accepting WMEs that satisfy the test condition. Beta nodes are represented as

this means it is ready to fire essentially. Likewise, we have another rule, cylinder rule which has got four pieces of data which is number 2, 9 10 and 12 for that x is equal to B. And all the conditions for a cylinder, the base is circular, the side is perpendicular or the angle is 90 degrees and the side surface is curved and the top is flat. All those conditions are met by those four things so we have these four things which come down to the conflict set. Now once you start the process for all the data you would produce a conflict set. What is a conflict set? A conflict set is a rule name, in this example the rule name is greenPyramid rule and the four pieces of data which is 11 16, so this pair of rule name along with matching data will go into the conflict set.

Now it may be possible in your database to be having more than one greenPyramid and there might have been more than one cylinder in which case you will have two instances of the same rule which is firing essentially. So this is the basic idea behind the Rete net is that the top half separates the data and the bottom half assimilates what you need for a rule to fire essentially.

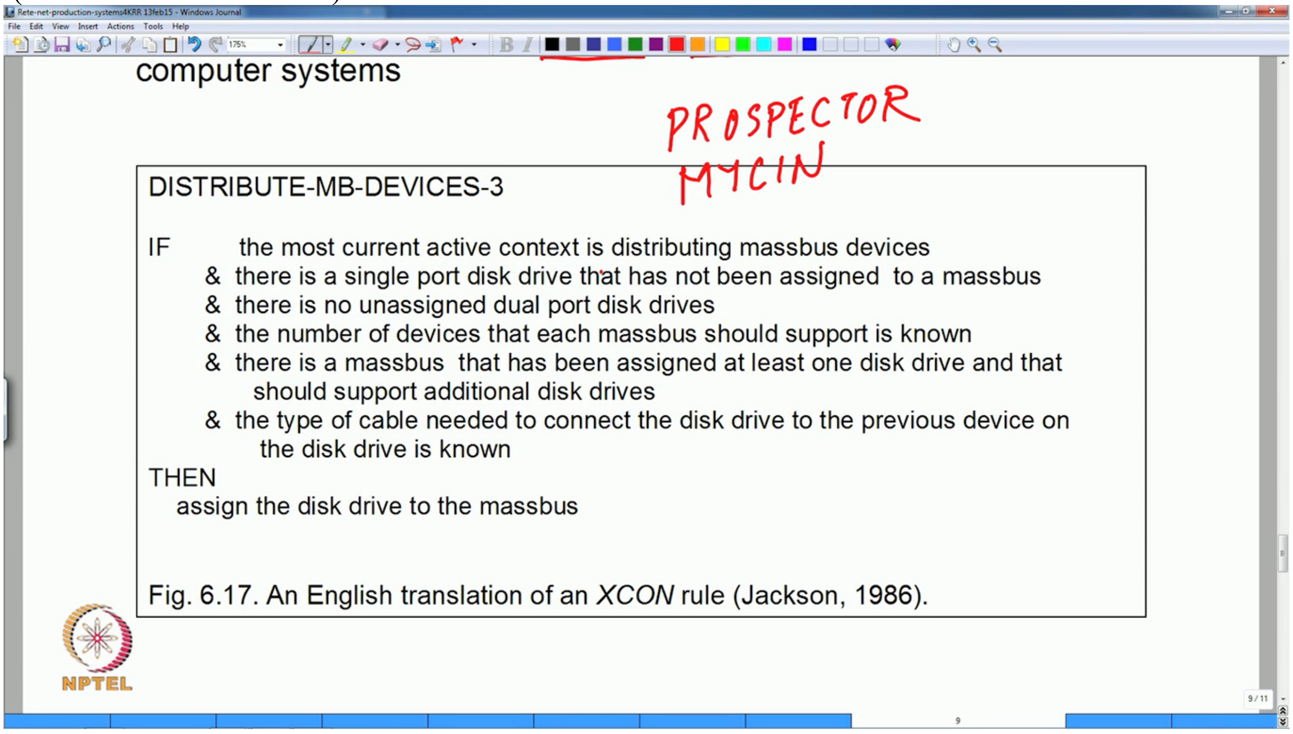
Now as i said Rule Based systems have been used extensively in what we sometimes call as or used to call in the expert system, now we just call them computer system. So one of the first such so called expert system was a system called XCON also called as R1 and it was a system which was used to configure DEC PDP systems. So essentially in those days' computers were not like PCs now a days. You cannot just go and say ok i want to buy a Macbook Pro or something like that. You had to practically configure the whole thing, you know piece by piece. It required a considerable amount of expertise and individuals ofcourse did not buy machines, it was basically organisations which bought machines.

So here is an example of a rule which is used by XCON. So this XCON was a forward chaining expert system essentially. It uses the Rete network based software to implement the rules and this rule as you can see is a english translation of an XCON rule and I have taken it from a book by Jackson called Expert Systems published in 1986. That was the time when expert systems were very popular. The idea behind expert systems was that you want to catch a domain expert and extract the domain experts knowledge and put it in the form of rules into the system essentially. And this whole process is called knowledge acquisition. And they had whole kind of protocols for how to do it. You know you would say this is how you interview an expert and so on and so forth. So this is a system to configure computer systems but there were other very popular systems called Prospective, so you must look this up on the web or something. Which was used to find places to drill for oil or minerals or something like that essentially.

Now drilling for oil is a very expensive proposition, to drill a hole costs lots of money essentially. So if you want to drill a hole then you better do it at a place where there is a good chance to get oil essentially. And Prospective was an expert system which was also build around these lines, to explore for geological explorations essentially. And the good thing about these things is that even if they sort of save you from one bad drill or point you to one good drill then they have kind of paid for themselves essentially. Developing is done by a few people over a period of time.

There was another system called Mycin this was also very famous which was used to diagnose pulmonary diseases. So essentially the idea there would be to catch a doctor who is an expert and find out okay what are the symptoms which point to this and so on and so forth. So the whole idea of rule based expert system was to get the knowledge from experts and then put it into the system and we will see in the next class how it is really operated. We have looked at the match part as of now we have not looked at the resolve part which is really another key to this whole thing essentially. So these rules say that I will just read it out if it makes sense to you.

(Refer Slide Time: 27:30)



computer systems

PROSPECTOR
MYCIN

DISTRIBUTE-MB-DEVICES-3

IF the most current active context is distributing massbus devices
& there is a single port disk drive that has not been assigned to a massbus
& there is no unassigned dual port disk drives
& the number of devices that each massbus should support is known
& there is a massbus that has been assigned at least one disk drive and that should support additional disk drives
& the type of cable needed to connect the disk drive to the previous device on the disk drive is known

THEN
assign the disk drive to the massbus

Fig. 6.17. An English translation of an *XCON* rule (Jackson, 1986).

The most current active context, so you are trying to structure your rules into a context. So for example initially you might do something so if you are building a house for example, you might say ok you want to first build the basement or the foundation, then you have to build the walls then you have to build the doors and the windows and then you have to build the roof. Each of them is a separate context and essentially you would want rules which are relevant to the context only to come into contention. And that is done by explicitly saying that it is a context. The context is building foundation or building wall or this thing and you would set the context as part of the program running. It should say ok now the context is to build the foundation then after that is finished some other rule will change the context. Now it would say change the context to building walls or something like that and all the rules which were in the context of building walls they would come into the match into the conflict set. So this is just to make the whole process a little but efficient.

Then there is a single port disc that has not been assigned to a massbus, there is no unassigned dual port disk drives and so on and so forth. There are set of conditions as you can see about 6 conditions are there and then you say that assign a disk drive to the massbus. And XCON or R1 was full of such rules.

So the high level task breakup so even that follows an algorithm at a certain level of detail and we saw that the algorithm is controlled by deciding what is the context and things like that. So the high level task breakup of XCON is as follows. You check the order, inserting any omissions and correcting any mistakes. Configure the CPU, arranging components in the CPU and the CPU expansion cabinets. Configure the unibus modules, putting boxes into expansion and this thing. So this is how XCON was designed to configure a system, so those of us that do the second step, do the third step you can say that it is very high level things like an algorithm. And it's a very generic algorithm it is not precise enough to say what is to be done.

What XCON contains is XCON fills in the low level details of the above algorithm by selecting and firing specific rules. So i have taken this example from my book and there of course I have taken one example from Jackson's book essentially.

So in the next class we will try to look at what is the language, we have not really specified the language, we have vaguely said there is working memory and there are rules. In the next class we will look at language and some of the issues which go into building such systems essentially.