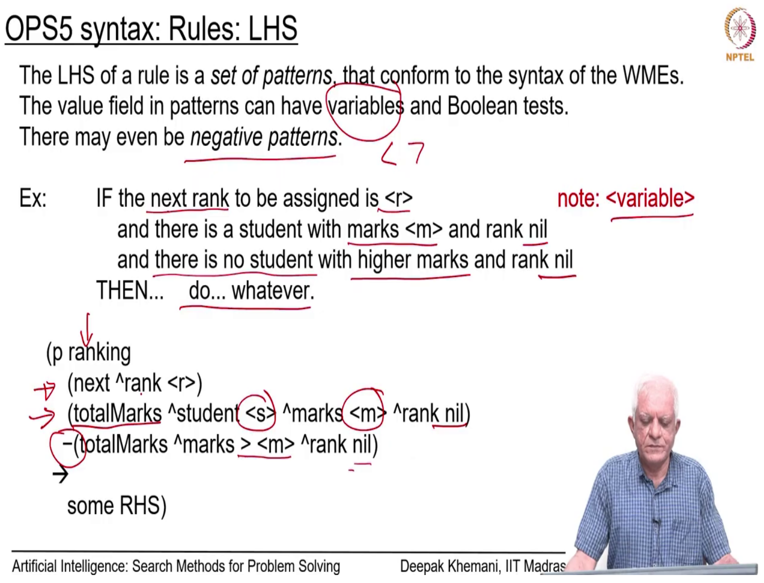
Artificial Intelligence: Search Methods for Problem Solving
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Chapter – 06
A First Course in Artificial Intelligence
Lecture – 74
Rule Based Expert Systems
The OPS5 Language

(Refer Slide Time: 00:14)



OPS5 syntax: Rules: LHS

The LHS of a rule is a *set of patterns*, that conform to the syntax of the WMEs.
The value field in patterns can have variables and Boolean tests.
There may even be *negative patterns*.

< >

Ex: IF the next rank to be assigned is <r>
and there is a student with marks <m> and rank nil
and there is no student with higher marks and rank nil
THEN... do... whatever.

note: <variable>

```
(p ranking
   (next ^rank <r>)
   (totalMarks ^student <s> ^marks <m> ^rank nil)
  -(totalMarks ^marks > <m> ^rank nil)
 →
   some RHS)
```

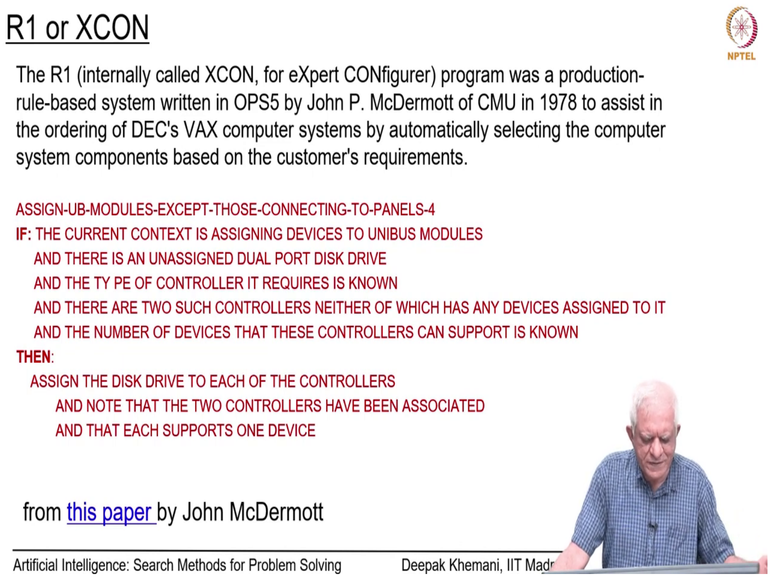
Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

So, welcome back. I hope you have finding this new way of looking at problem solving exciting. This approach in which we are looking at rule based production systems and we have seen so, far this idea of memory short term memory which is made up of working memory elements in a store which is the called working memory which is indexed with the timestamp.

And we saw of the idea of rules which have left hand sides which match those working memory elements. The fact that rules have variables, make them general which means that they will not match just one element, but they could match a whole lot of elements potentially and we have seen how this match is done.

So, in this example that we finish with we said that this rule for assigning ranks says that IF the next tank available is r and there is a student who has got marks m who does not have a rank assigned and there is no student who has got more than m marks THEN you are ready to assign rank r to student s; s is a variable which refers to the student. It could be name, it could be roll number, it does not matter here.

(Refer Slide Time: 01:51)



R1 or XCON

The R1 (internally called XCON, for eXpert CONfigurer) program was a production-rule-based system written in OPS5 by John P. McDermott of CMU in 1978 to assist in the ordering of DEC's VAX computer systems by automatically selecting the computer system components based on the customer's requirements.

```
ASSIGN-UB-MODULES-EXCEPT-THOSE-CONNECTING-TO-PANELS-4
IF: THE CURRENT CONTEXT IS ASSIGNING DEVICES TO UNIBUS MODULES
   AND THERE IS AN UNASSIGNED DUAL PORT DISK DRIVE
   AND THE TY PE OF CONTROLLER IT REQUIRES IS KNOWN
   AND THERE ARE TWO SUCH CONTROLLERS NEITHER OF WHICH HAS ANY DEVICES ASSIGNED TO IT
   AND THE NUMBER OF DEVICES THAT THESE CONTROLLERS CAN SUPPORT IS KNOWN
THEN:
  ASSIGN THE DISK DRIVE TO EACH OF THE CONTROLLERS
      AND NOTE THAT THE TWO CONTROLLERS HAVE BEEN ASSOCIATED
      AND THAT EACH SUPPORTS ONE DEVICE
```

from this paper by John McDermott

Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madr

So, let us look at the right hand side actions. Before we do that we had talked about expert systems and we had said that there was a system called R 1 also called XCON because it did configure VAX systems and it had rules essentially of the kind that we have been talking about. So, we are interested in the left hand side of the role at this moment. So, let us just focus on that.

R 1 which was within the community called XCON or eXpert CONfigurer was a production rule based system written in OPS5 by John McDermott of CMU in 1978 essentially to assist the ordering of DEC VAX computer systems as I have already mentioned by automatically selecting the computer systems component based on the customer's requirements.

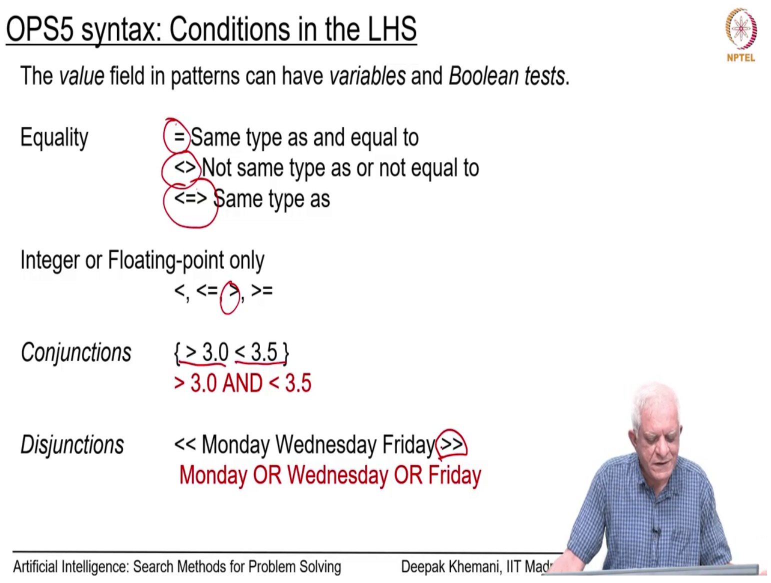
So, I will present one rule in English language from that system just to give you a flavor of the kind of systems that people actually built in this language. So, here is a rule. It says asslgn-some-UB-modules-except-those-connectlng-to-panels-4 if: the current context is assigning devices to unibus modules and so on and so forth.

There are set of conditions. There are five conditions on the left hand side. The second condition is there is an unassigned dual port disk drive. Just like we talked about an unassigned student with rank and so on.

And other conditions that we do not need to get into the fact , but we just want to understand that this is the kind of knowledge that rule based production systems wanted to capture and encapsulate and use for problem solving.

So, you can imagine that that the person who was working for deck had to only state these rules and the program are one would help the sales person who may not be an engineer configure a system essentially. So, this rule I have taken from this paper that you can get the link from here by John McDermott.

(Refer Slide Time: 04:11)



OPS5 syntax: Conditions in the LHS

The *value* field in patterns can have *variables* and *Boolean tests*.

Equality	= Same type as and equal to
	<> Not same type as or not equal to
	<=> Same type as
Integer or Floating-point only	<, <=, >, >=
Conjunctions	{ > 3.0 < 3.5 }
	> 3.0 AND < 3.5
Disjunctions	<< Monday Wednesday Friday >>
	Monday OR Wednesday OR Friday

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Mad

So, we were talking about conditions in the left hand side which are patterns we said. Let us be a little bit more specific now. As we said the value field in patterns can have variables and can have Boolean test essentially. What are the kind of tests that we can have? We can test for equality. So, these are the symbols that OPS5 gives us to check for those tests. So, this is OPS5 syntax.

The first one says that same type or equal to. Second one says that not same type or not equal to. And the third one says not third one says same type as essentially. So, these are Boolean tests essentially.

So, the value in the attribute field of the rule is same type as the value in the working memory element for the same attribute essentially. Then for integer and floating point data we can which is basically ordered data ordered sets.

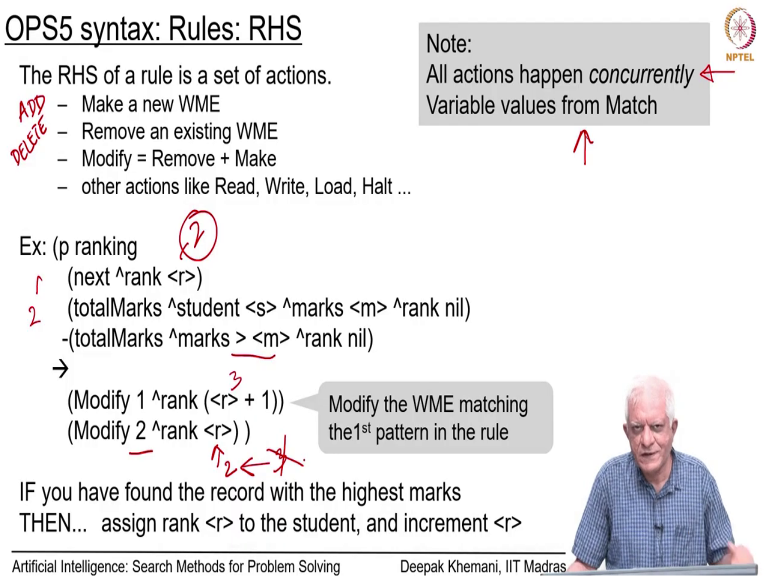
We can check whether one is less than the other or greater than the other and so on. We have already mentioned this in passing when we said that there is no student who has got marks higher than m essentially and we had used this greater then symbol there in that particular rule essentially. We can have conjunctions and we can have disjunctions. This conjunction says that conjunctions are enclosed in curly brackets.

So, there is one test which says the value must be greater than 3.0 and the second test which says it must be less than 3.5. So, it is an AND essentially. The value must be greater than 3.0 and it must be greater than 3.5. You do not have to give a name to that value. All your saying is that this value this pattern will match a value which satisfies these conditions that its a number which is greater than 3.0 and less than 3.5.

We can have disjunctions that we will have for example, an afternoon tea session in the department on Monday or Wednesday or Fridays because those are the days when people are more free in the afternoons. It can be expressed as the this thing with in two angular brackets essentially..

So, this is a disjunction either Monday, which means that if the data working memory element matches has a Monday it will match or if it has a Tuesday sorry if it has a Tuesday it will not match or if it has a Wednesday it will match and so on essentially, it is a disjunction. We can have conjunctions, disjunctions, comparison, type checking.

(Refer Slide Time: 07:09)



OPS5 syntax: Rules: RHS

The RHS of a rule is a set of actions.

ADD
- Make a new WME

DELETE
- Remove an existing WME
- Modify = Remove + Make
- other actions like Read, Write, Load, Halt ...

Ex: (p ranking
    (next ^rank <r>)
    (totalMarks ^student <s> ^marks <m> ^rank nil)
    -(totalMarks ^marks > <m> ^rank nil)
  →
    (Modify 1 ^rank (<r> + 1))
    (Modify 2 ^rank <r>))

Modify the WME matching the 1st pattern in the rule

IF you have found the record with the highest marks
THEN... assign rank <r> to the student, and increment <r>

Note:
All actions happen *concurrently*
Variable values from Match

Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Madras

Now, let us talk about the right hand side. The right hand side of a rule is a set of actions. The simplest action is to make or add. Some textbooks may use the word ADD here. OPS5 manual says make I think or you can remove a working memory element. Some books may say DELETE, it is a question of keywords.

So, you can add a new memory element or you can make a new memory element or you can remove an existing element. And modify is basically a combination that you remove that thing and add a modified one two that essentially. Or of course, you can have it is a complete programming language we said. We can do things like read, write, load a file or print a message whatever. One thing to note is that all actions on the right hand side of the rule happen concurrently.

Which basically means that the variables that we have matched they do not change inside the action, but they will change in the new element that we are adding to the system essentially. So, if you are doing something with a variable the values that they will have will be the values that were laid at match time essentially. So, variable values are from match essentially.

So, here is our first rule that we have been talking about, the ranking rule. We already seen the left hand side. It said that if the next rank available is r, if there is a student s whose marks are m, who does not have a rank and there is no student who has marks more than m and rank nil; remember this test we just spoke about and rank nil. Then what do you do? You essentially modify the two elements. Modify the working memory matching the first pattern in the rule.

What is the first pattern? It says next rank is r. What are we modifying it to? We are saying that make it r plus 1. If the next rank was 3 then the next student will get 4 and after that the next student will get 5 and so on.

So, here we are basically incrementing that value r. So, as I said you can imagine that in your knowledge base database you will have one record which says which is the next sign to be applied and we keep updating it as and when we keep assigning ranks.

And the other thing we do is we modify the second record. So, these 2 stands for the fact that this is 1 and this is 2. This is not written in the rule, but it is understood that that is the second pattern of the first pattern and give it the rank r. So, what I said here that they are applied concurrently is reflected here. We may have said that modify the rank to r plus 1, but does not mean that the modified rank will be assigned..

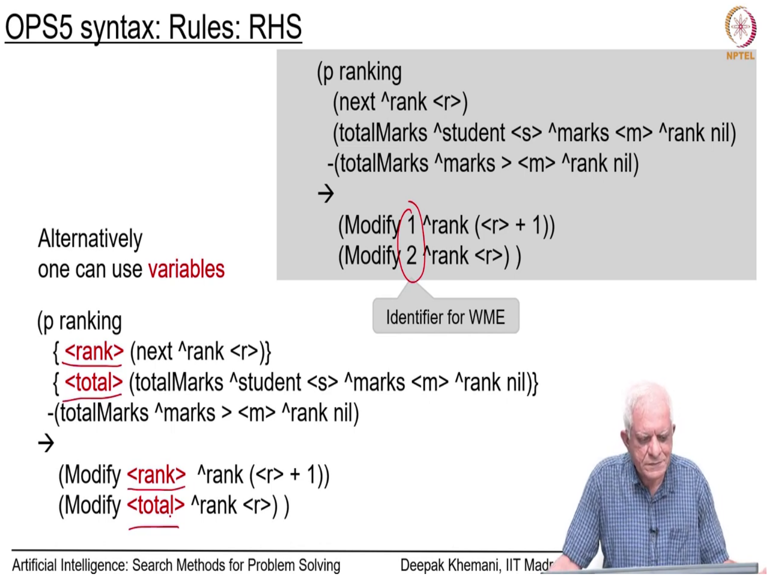
The whatever that supposing this was 2 ok, then this will still get the value 2 and this will now become 3. This will not, this will not get a value 3 that is what I meant by saying that actions are applied to data which has already been read it essentially. So, in English what does it say? If you have found the record with the highest marks then assign rank r to the student and increment r essentially. It is a very simple rule essentially.

And the nice thing about such declarative languages is that you may have a class of 200 students and you may have their total marks and you have to write, give assign them ranks. This one rule will do your all your job. We do not need to worry about loop, go from first student to 200 student and then do this and so on and so forth. This rule by repeated application will assign ranks to all the 200 students provided when you initialize the system, you say that the first rank to be assigned is 1 essentially.

It will take the student with the highest marks. Assign it rank assigned him or her rank 1 then the next highest mark. Assign the student rank 2 and so on. If there is a tie then we have not stated how it will be resolved and one of them will get rank 1, one of them will rank 2, but maybe you can think about how to how to refine the system.

So, that if there are two toppers in the class both of them get rank 1 and the third student gets rank 3 essentially. So, nobody gets rank 2 but, this system will not do that. Maybe it is a small exercise that you should work on.

(Refer Slide Time: 12:29)



OPS5 syntax: Rules: RHS

```
(p ranking
   (next ^rank <r>)
   (totalMarks ^student <s> ^marks <m> ^rank nil)
  -(totalMarks ^marks > <m> ^rank nil)
→
    (Modify 1 ^rank (<r> + 1))
    (Modify 2 ^rank <r>) )
```

Identifier for WME

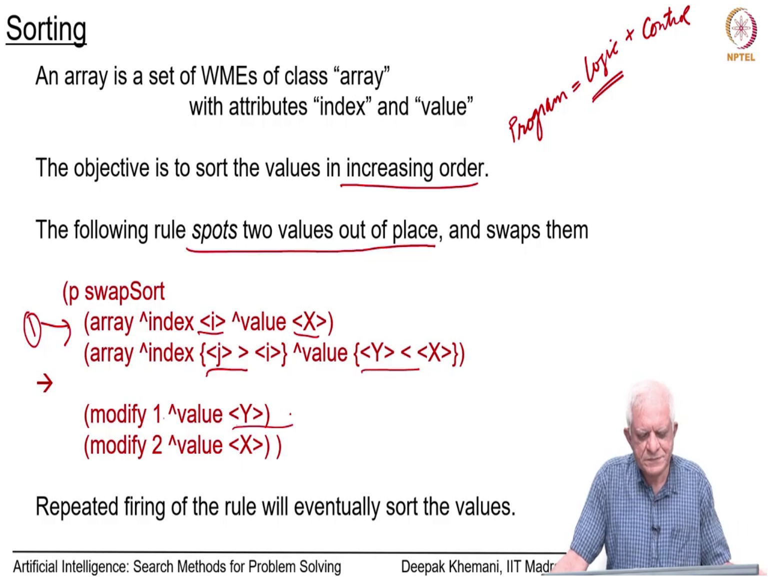
Alternatively
one can use variables

```
(p ranking
   { <rank> (next ^rank <r>)}
   { <total> (totalMarks ^student <s> ^marks <m> ^rank nil)}
  -(totalMarks ^marks > <m> ^rank nil)
→
    (Modify <rank>  ^rank (<r> + 1))
    (Modify <total> ^rank <r>) )
```

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madr

We said that we can identify which ones to modify or delete using this position in the rule. Alternatively you can even use variables. So, for example, I can call this record rank, I can call this record total and I can use those things to say that this is what I want to modify essentially. Remember that modify means I will delete the old record and I will add a new record which has made these changes that we are specified here.

(Refer Slide Time: 13:07)



Sorting

An array is a set of WMEs of class "array" with attributes "index" and "value"

The objective is to sort the values in increasing order.

The following rule *spots* two values out of place, and swaps them

```
(p swapSort
    (array ^index <i> ^value <X>)
    (array ^index {<j> > <i>} ^value {<Y> < <X>})
→
    (modify 1 ^value <Y>)
    (modify 2 ^value <X>) )
```

Repeated firing of the rule will eventually sort the values.

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madr

So, here is another small. As I have been emphasizing on the fact that this is like a programming language, so, let us look at the most popular programming task which is that of sorting essentially. It is said that that half the time computer systems are simply sorting data I think or more than half the times I mean.

So, we had introduced this notion of an array. Let us use that now and we are going to talk about sorting. So, an array is a set of working memory elements whose class name is array and which has got two attributes one is called index and the other is called value. And the objective is to sort the value say in increasing order of index essentially.

So obviously, both have to be of order types. So, indexes are typically 1, 2, 3, 4; the first element, the second element, the first row of array and so on. And the value is something which can be compared, otherwise the meaning of sorting does not make sense essentially. .

So, here is a rule. What does it do? It basically spots two values that are out of place that is all it does. Try to compare this with other programs, you may have written for sorting. You know you have to worry so much about control essentially as we will see later next week. Robert Kowalski who was one of the inventors of the language Prolog said; a program is made up of two components logic plus control essentially.

Imperative programming languages focus a lot on control, what is to be done next what is to be done next and so on and so forth. Declarative programming languages do not worry about control they were very on the logic part only ok.

So, I am just writing Kowalski statement here, but we will revisit this. So, we are going to focus only on the logic part that is the whole idea of declarative programming. State the logic. Let someone else take care of the control and that the whole thing is your program.

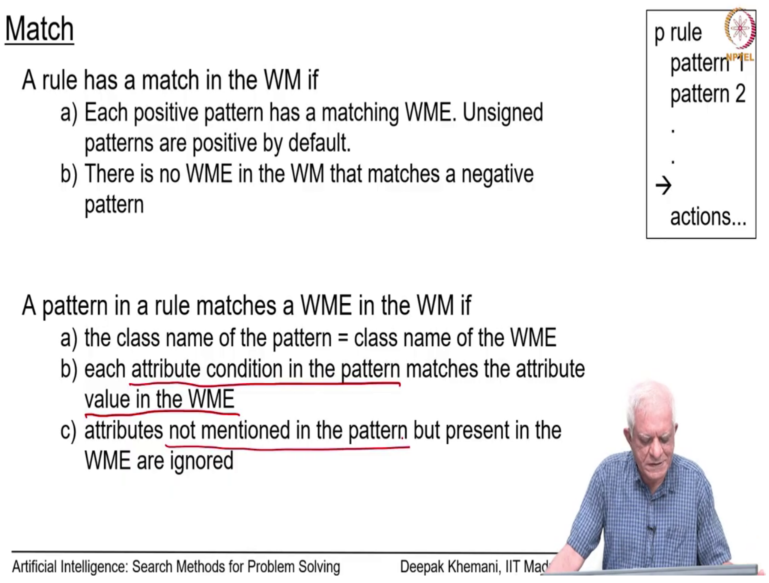
So, here is a logic for sorting. It essentially says what do you mean by sorting and it says if there are two elements out of place then exchange them essentially. How does it say that? The left hand side says that if there is an element at index value I, who has a value X and if there is another element whose index j is greater than i and whose value is Y which is smaller than X then they need to be swapped essentially because, we said that we want them in increasing order essentially. .

So, if there is a later element which has a lower value this exchange the two values essentially. So, we are not making any claims here about efficiency here. We are simply saying that declarative programming emerges out of this notion that if you understand what is the task that you are trying to do and express the kind of patterns that you are looking for and working with you can implement simple programs.

Again we are not talking about efficiency here. What do we do? We simply modify the two records, the first one and the second one and we give them the new values. Originally record i had value X, now record i has which is i is in the first; this first refers to the first pattern that we are matching here right.

This is 1. So, 1 had value X to start with, now it has a value Y essentially and 2 likewise has now got value X and we do that. So, if you just write this one rule and let our system loose or influence engine do its work, it will eventually end up sorting the elements.

(Refer Slide Time: 17:52)



Match

A rule has a match in the WM if

a) Each positive pattern has a matching WME. Unsigned patterns are positive by default.

b) There is no WME in the WM that matches a negative pattern

```
p rule
  pattern 1
  pattern 2
  .
  .
→
  actions...
```

A pattern in a rule matches a WME in the WM if

a) the class name of the pattern = class name of the WME

b) each attribute condition in the pattern matches the attribute value in the WME

c) attributes not mentioned in the pattern but present in the WME are ignored

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Mad

So, let us go back to match in a little bit more detail. So, by now it is clear to us that you know a rule has shown here is a collection of patterns followed by a collection of actions essentially and match operates on the left hand side which is on the patterns essentially. A rule has a match in the working memory if each positive pattern has a matching working memory element and unsigned patterns are by default positive.

So, as we saw in the last rule that we saw not the last rule the ranking rule that we saw the first two patterns had no signs. So, we assume that they are positive that we must find the matching working memory element for them. Its only when they were negative patterns we have put a negative sign there in front of them.

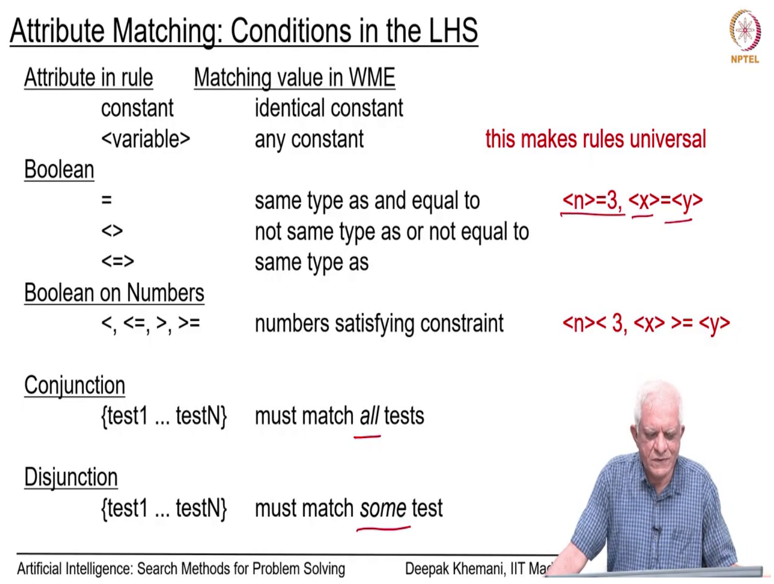
So, a rule matches of working memory element if for every positive pattern there is a matching working memory element and there is no working memory element which matches the negative pattern if there is one in the rule.

The sorting rule did not have any negative patterns. So, the second thing will not apply, but the ranking rule had. You did not you said that there should be no student with higher marks. So, this there should be no student is expressed as a negative pattern. So, a pattern in a rule matches a working memory element if the class name of the pattern is the class name of the working memory element that is the first thing we on which you will not proceed.

And each attribute condition in the pattern matches the attribute value in the working memory element. So, notice the difference here. We are talking about attribute condition in the pattern and we are talking about value in the working memory element. The working memory element does not have variables. It has only constraints these things or it has only values. The pattern may specify some condition on that value. It should be greater than 3.5, it should be less than 20 or whatever that conditions are.

Now, if there are attributes which are in the working memory elements, but which are not mentioned in the pattern are simply ignored essentially. So, we may have written this ranking program to use the name of the student and not use the roll number for example, but the record may have both name and roll number. So, it does not matter. If the pattern does not mention any attribute then you can simply ignore that attribute essentially.

(Refer Slide Time: 20:51)



Attribute Matching: Conditions in the LHS

Attribute in rule		Matching value in WME	
	constant	identical constant	
	<variable>	any constant	this makes rules universal
Boolean			
	=	same type as and equal to	<n>=3, <x>=<y>
	<>	not same type as or not equal to	
	<=>	same type as	
Boolean on Numbers			
	<, <=, >, >=	numbers satisfying constraint	<n>< 3, <x> >= <y>
Conjunction			
	{test1 ... testN}	must match *all* tests	
Disjunction			
	{test1 ... testN}	must match *some* test	

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Ma

So, let us also formalize this notion of how do you match attributes conditions in the left hand side. If the attribute in the rule is a constant, it can only match an identical constant in the working memory element. If the attribute in the rule has a variable then it can match any constant in the working memory element. That is what makes the rule universal. All data elements all working memory elements do something, ok.

So, let us say you want to increment the marks of every student, all you would say is that if there is a student whose got some marks then add 1 to that mark and put it back in the memory. So, it makes it a universal rule essentially. We will talk about rules as we go along. We will look at a logic little bit also in this thing. What we are trying to do in this course? This course is focused on search methods.

What we are trying to show is that it is not that search is in a different entirely different compartment and knowledge is in a totally different compartment. We are trying to show that there is a great intertwining of these two that to use knowledge you have to you know use search which rule to apply to what data and that kind of stuff and knowledge of course, comes in handy for solving problems. So, it has to basically work together. .

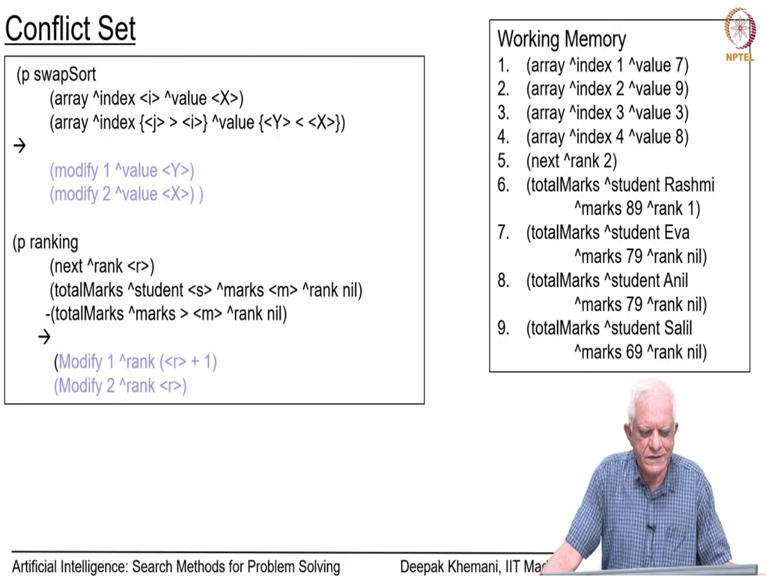
So, we talked about Boolean conditions equal to not same type as or not equal to and that we have seen already and here are some examples here. We have already seen such examples. We can say that the variable must be equal to 3 or the variable X and the variable Y. Remember these X and Y maybe in different patterns or in the same pattern on the rule as well it does not matter essentially.

So, for example, if you wanted to apply this program that I asked you to do as a exercise of if there are two toppers to give them the same rank 1 then you could use this condition somewhere there that, if there is another student who has got the same marks as the first student and you have given the first student a rank 1 also give the second student a rank 1 and increment the rank.

The rank; see after you assign the first student a rank one you have said the next rank is 2. Now, you see a student with the same marks. You say ok, give him also or give her also the rank 1 and but still I increment the rank to 3. So, in that manner you could use this condition to implement this revised program. We have talked about numbers already less than less than equal to greater than greater than equal to and we can write tests which are satisfying those constraints.

We have talked about conjunction and disjunctions. Conjunctions are a series of tests and all of them must be met. So, it must match all tests. Disjunctions are also a series of tests, but it as long as it matches only one test if you are happy essentially.

(Refer Slide Time: 24:28)



Conflict Set

```
(p swapSort
      (array ^index <i> ^value <X>)
      (array ^index {<j> > <i>} ^value {<Y> < <X>})
→
      (modify 1 ^value <Y>)
      (modify 2 ^value <X>) )

(p ranking
      (next ^rank <r>)
      (totalMarks ^student <s> ^marks <m> ^rank nil)
     -(totalMarks ^marks > <m> ^rank nil)
   →
      (Modify 1 ^rank (<r> + 1)
      (Modify 2 ^rank <r>)
```

Working Memory

```
1.  (array ^index 1 ^value 7)
2.  (array ^index 2 ^value 9)
3.  (array ^index 3 ^value 3)
4.  (array ^index 4 ^value 8)
5.  (next ^rank 2)
6.  (totalMarks ^student Rashmi
           ^marks 89 ^rank 1)
7.  (totalMarks ^student Eva
           ^marks 79 ^rank nil)
8.  (totalMarks ^student Anil
           ^marks 79 ^rank nil)
9.  (totalMarks ^student Salil
           ^marks 69 ^rank nil)
```

Artificial Intelligence: Search Methods for Problem Solving   Deepak Khemani, IIT Ma

So, this takes care of the left hand side of the rule which is matching. We will talk about conflict sets next, but let us take a short break. Time for you to take a breather and digest what we have done and come back and we will talk about how to construct the conflict set essentially, ok. So, let us take a break at this point.