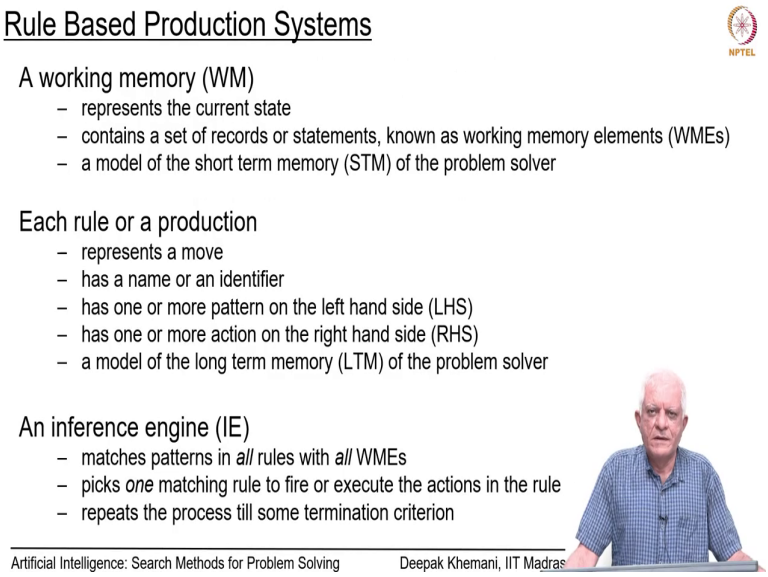
Artificial Intelligence: Search Methods for Problem Solving
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Chapter – 06
A First Course in Artificial Intelligence
Lecture – 73
Rule Based Expert Systems
The Inference Engine

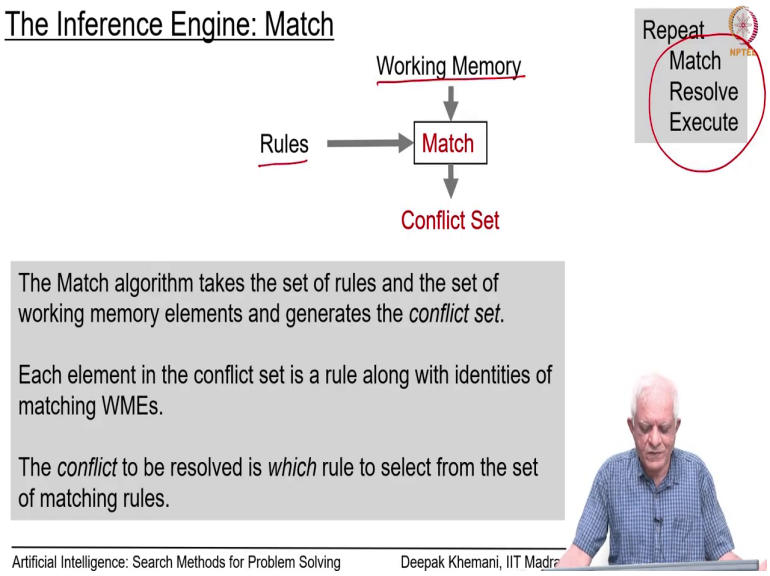
(Refer Slide Time: 00:14)



So, welcome back. We are just started looking at Rule Based Expert Systems or Rule Based Production Systems as whatever you want to call them and we have seen that such a system is made up of three components; one is the working memory, which contains the data related to the particular problem that is being solved.

The second is a set of rules which contain the problem-solving knowledge of the agent and the third is an influence engine, which has the responsibility of taking the rules or the problem-solving knowledge of the agent and applying it to the data which is there in the working memory. So, these three components are together makeup as rule based expert systems. So, let us look at that in a little bit more detail now.

(Refer Slide Time: 01:15)



The inference engine, as I said is a cycle which repeats this process. The first step is called match in which as depicted here, you take a set of rules, you take the working memory, apply the match algorithm and what you get is something called a conflict set essentially. So, that is the first of the three steps that we do and that is called match essentially and the inference engine is typically an infinite loop of match resolve execute.

So, we just keep doing that; match, resolve, execute; match, resolve, execute. One of the execute actions may be say let you know stop ok, we are done, we have solved; give the solution to the user whatever that may be, it may be part of the actions. But the inference engine itself is into this infinite loop essentially. It has now built in termination criteria.

The termination criteria comes from some rule which has decided which has looked at the situation or looked at the working memory and decided that the problem has been solved and it is time to stop and give the solution out essentially. The inference engine itself is an infinite loop, which does this match resolve and execute.

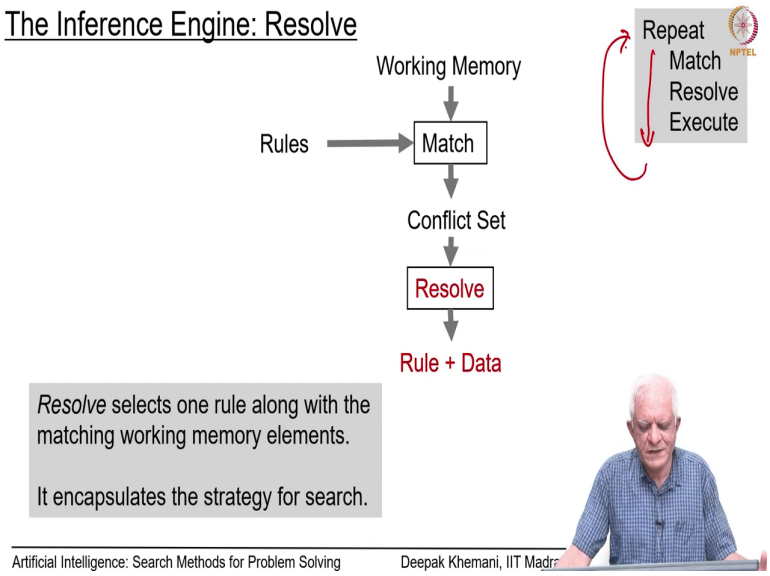
So, this is the first component in that which is match, it takes a set of rules, it takes a working memory and returns what is called as a conflict set. We will look at this in more detail as we go along. So, the match algorithm takes a set of rules the set of working memory and generates a conflict set.

Each element in the conflict set, its a set remember is a rule along with the identities of which working memory elements it matches with essentially. So, it is a rule instance with the data that it matches. What is the conflict here?

If there are more than one rules which are matching, if the conflict set has more than one element, then they are all in some sense contending to execute. There is a conflict. Each rule you can imagine is saying I want to execute, I want to execute, I am ready to execute and things like that.

It is a inference engine which has to decide which one to pick and this process is called conflict resolution as we will see shortly. So, the conflict to be resolved in the conflict set and that is why it is called a conflict set is which rule to select from the set of matching rules.

(Refer Slide Time: 04:00)



So, the next step is resolve ok. So, remember, we said that the influence engine is made up of these three things; match, resolve and go back and do this whole thing again. So, the second step is resolve.

The resolved component takes the conflict set; remember that a conflict set is a set which contains rule instances along with matching data and it picks one from that, it picks one rule along with matching data. How it does that, we will see as we go along; but that is what resolve does. It resolves a conflict between the different contending rules and picks one essentially.

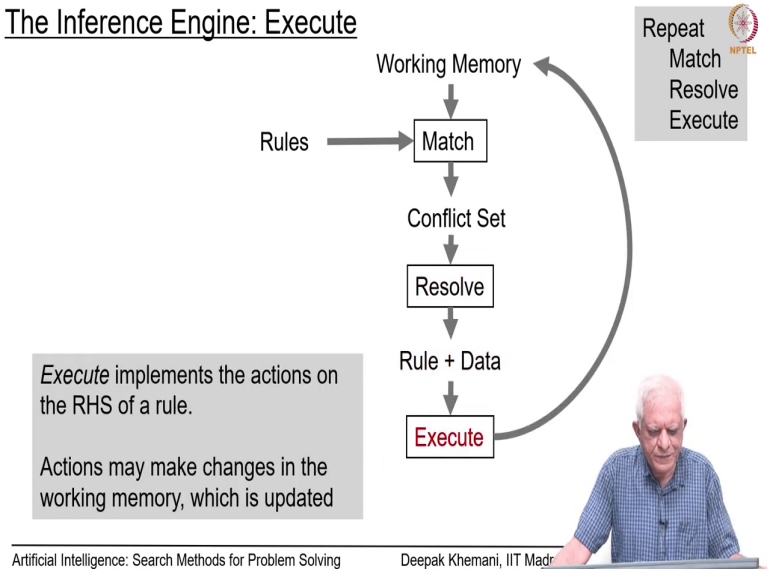
As we will see, this is a critical component in an inference engine because just like we did, when we are doing search in the state space or solution space, you generated moved in and then, it was very important as to which of those moves that you will select essentially.

Likewise, here also you are generating many rule instances which are ready to execute and deciding which one to execute next is going to be critical to solving the problem.

And we will see that, this is where lot of knowledge has to be embedded and we will look at different conflict resolution strategies essentially. So, essentially the resolved component encapsulates the strategy that you are using for search essentially. At this point, I should also emphasize that these are forward reasoning systems. Just like we had forward plan state space planning and backward state space planning, these are forward reasoning system.

In the sense, you have work from the given state, look for a pattern in the given state, do some action and then, look at the pattern again essentially. Later on, we will also look at backward reasoning with rules, but that is probably next week I think.

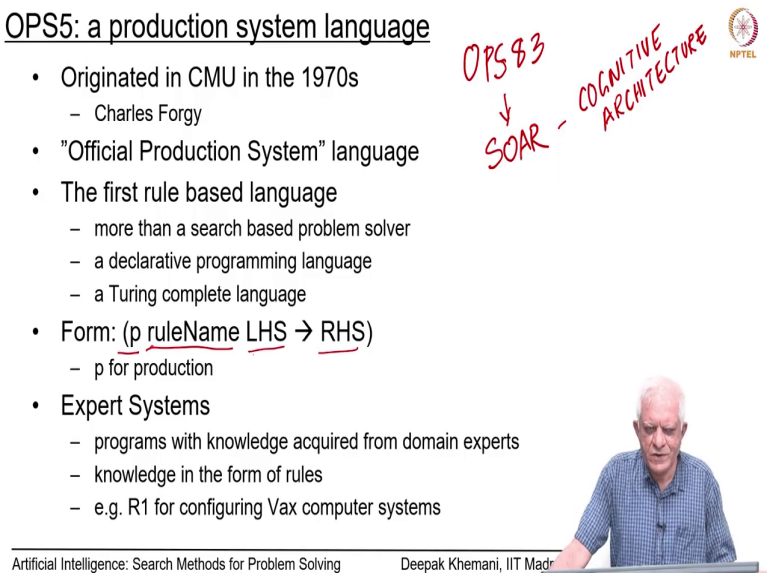
(Refer Slide Time: 05:57)



The third component is execute. Its job is relatively simple. Resolve has given it one rule instance with data, that rule in sense, we will have some actions as we will see on the right hand side. The job of execute is simply to execute those actions. Now, we will see that those actions will typically change the working memory, it may add new elements, it may delete some element from that and therefore, we have to go back all over and do the match all over again.

So, at a conceptual level at least, at a higher level at least this is the cycle that we are going through match, produce a conflict set, resolve, pick one element from the conflict set, execute. Execute the actions in the right hand side of the rule that you have picked and go back and do match all over again essentially.

(Refer Slide Time: 06:53)



So, I said that this has given rise to a language which is during complete and often, these kind of systems the inference engine that we are talking about are called expert system shells. Somebody implements this whole thing and gives it to a user and the user basically decides what is going to be in the working memory, what is what are going to be the rules, takes one of the strategies offered by the implementation and builds the system.

So, this language was called OPS5; it came out of CMU. Remember CMU was the place hotbed of AI activity started about 70 years ago by Newell and Simon and it produce a lot of exciting work in AI essentially. This was one of those things. So, as I said it was this language came from CMU, it was devised by a person called Charles Forgy.

In fact, it was part of his PhD thesis; one of the more influential PhD thesis you might say. The name itself kind of reflects the fact that it was official production system language essentially.

And they were like most languages they were versions OPS5 was the version which they started with. For some reason, it was called OPS5. In fact, when I was a student, I looked that programming in OPS5 myself and by the time, I was graduating the new version called OPS 83 had come; maybe I should mention it here, 83. And much later, it evolved into a system which was called SOAR, which we will not have time to cover in this course. But it was a kind of a cognitive architecture.

Remember that this whole thing came out of Newell and Simons work on how humans solve problems. So, they have a very cognitive flavor about how things go about. And cognitive architecture SOAR is just one of the cognitive architectures which are around nowadays, but SOAR in sense some sense implements all these different strategies, which people think that humans use for solving problems so that a user of that system has to only plug in the problem and the data and the rules and so on.

And remember that is a general flavor of AI that we are looking at; weak methods in problem solving that. Somebody devises the problem-solving system; the user just puts in the problem

and related data along with it. So, it has as has its own evolution and the latest version is called SOAR essentially.

So, OPS5 was the first rule-based language that was devised. The first forward chaining rule-based language I should say. We will see that the backward chaining languages like prolog which some of you must have been must be familiar with were devised independently of this in a different part of the world, in mostly in UK and France. It was as we have said, it was more than a search-based problem solver.

It was a complete declarative programming language with search which was built into that language ok. Again, I would emphasize the fact that declarative programming means that somebody else, meaning not the programmer or not the user, decides what are the actions to be done at what time.

Declarative programming means you giving the knowledge related to solving the problem which is expressed in the form of rules. And a search algorithm takes care of implementing that knowledge essentially, but as we have mentioned it is a turing complete language, which means anything you can do in any programming language you can do it in OPS5 essentially.

The whole idea of exploring different programming languages and currently, the world is you know awash with so many different languages. Some of them are very specialized designed to do things nicely on the web and some are designed to do work well with large amounts of data. So, there are all kinds of things essentially.

Declarative languages like OPS5 were designed to work with human knowledge and operationalizing it in some way. The form of a rule in OPS5 was this structure. You started with an opening bracket and the word p, where p stands for production. You give a name to that particular rule, you describe the left hand side; then, there is an arrow and then, you describe the right hand side.

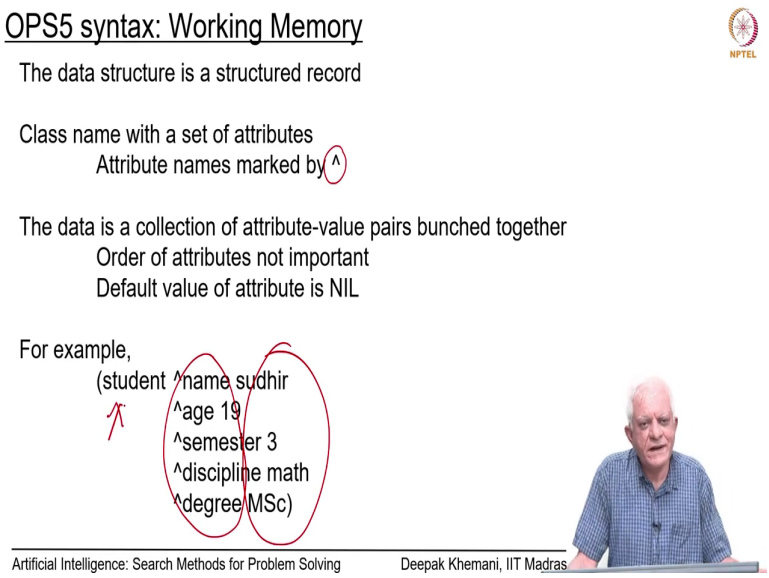
So, this was a syntax of a high-level syntax of rules in this particular language. As I said these were designed to build expert systems. Programs with knowledge acquired from domain

experts and the knowledge was acquired in the form of rules essentially. So, one of the successful systems that was built at that time was called R1 and it was used for configuring Vax computer systems.

So, in those days and we are talking about the 70s and the 80s, computer systems were huge systems, room full of system, room full of hardware and if you wanted to buy a computer, typically a university would want to buy a computer; then, they would have to carefully configure what is the thing that you will want to have. And this R1 system expert system was designed to configure DEC Vax system.

So, DEC was the company and Vax was a particular line of computers that they were producing which are many popular, very popular and that I remember in the 80s, DIFR in Bombay had a very good DEC system, where I had the opportunity to go and program during my masters work.

(Refer Slide Time: 13:49)



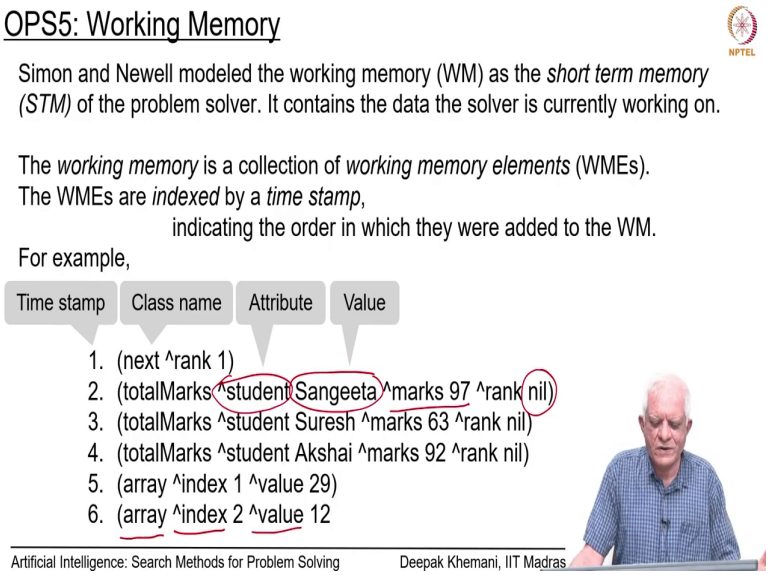
So, let us look at OPS5 syntax a little bit. The first thing is the working memory. The data structure is going to be a structured record which means every record will have a class name and a set of attributes and the attributes would be marked by a special symbol which is that hat. So, you can imagine for example, if you are creating a system for a college or a hostel or something, you would have class name as course, marks or student; even does not matter.

Anything which identifies the record right and attribute names would be things like name, roll number or subject or whatever the case may be. So, the data structure was a structure which has a class name followed by attributes and of course, the attributes had to have values which is what made it data. The data is a bunch of attribute value pairs bunched together in one class name.

It turns out that it is easy to implement systems, where the order of attributes is not important. You do not have to say the name and age for example and roll number in this particular order. You could state that in any order and the system would work. The default value for any attribute was NIL, which means if the data does not specify; then, this is what would be stored in the value field of the particular attribute.

So, here is an example. Now, we first look at an OPS5 data element. The class name here is student; the attribute name has a value Sudhir; attribute age has a value 19; attribute semester has a value 3; attribute discipline has a value math and attribute degree has a value MSc. So, you can say it is a collection of attributes and collection of values which are associated with the attributes, all of them are bunched together and put in one structure whose address by the name student essentially.

(Refer Slide Time: 16:30)



So, the working memory is a collection of such data elements. As I said Simon and Newell model, the working memory as a short term memory of the problem solver, it contains the data that the solver is currently working with essentially. It is like a state representation for us and we are familiar with this notion. It is a collection of working memory elements and each element is indexed by a time stamp essentially.

We will see that this is going to be important, especially for conflict resolution. When we have to decide as to which rule to apply, timestamp will play a role in that and the timestamp is essentially an a label which indicates the order in which they were added to the working memory. So, it is not real time in the sense that 17 hours, 15 minutes and so on.

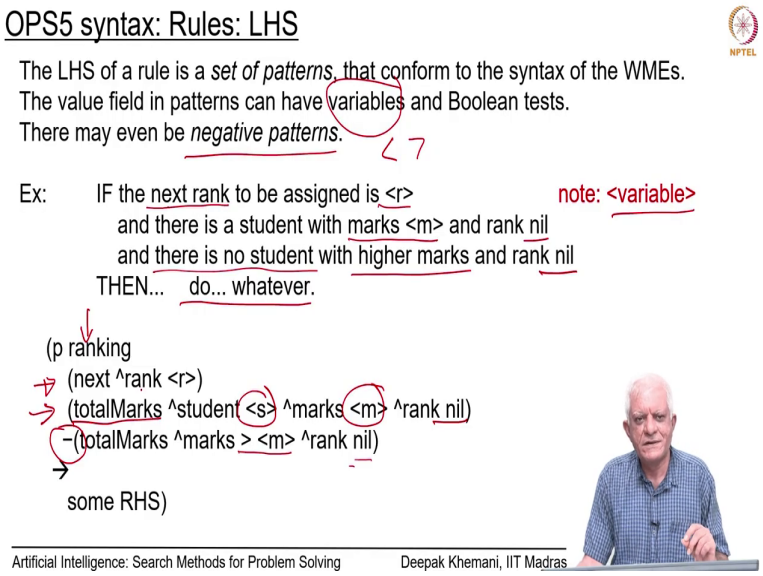
It is just the order which matters; which was the first working memory element, which was the second element and so on and so forth. So, here is the example to illustrate that idea. This is the working memory. It is got 6 elements in it. The first column shows a timestamp; the second column shows the class name, then for each record there is an attribute name and a value.

So, what we have seen here is for example, the attribute student has the name Sangeeta; the attribute marks has a value 97. Some attributes have nothing in it. So, they are just stored as NIL.

So, you can imagine that this is part of some data that you are using to when you would grade students, for their courses and you know assign ranks and things like that and so on and so forth. Here is another example, where there is a class called array. It has an attribute called index and an attribute called value essentially.

So, you can think of an array. Of course, when we talk about arrays then we say that it is a two-dimensional array or one-dimensional array and so on. So, this represents a one-dimensional array or an array which has a index and which has a value stored in it essentially. But the array is stored in a kind of a distributed fashion inside the working memory. Each row has a separate record in it essentially.

(Refer Slide Time: 19:14)



So, we have seen working memory elements and we have seen working memory. Now, let us look at rules essentially. We said that rule has the following component, it has a name, it has a left hand side and it has a right hand side essentially. So, let us look at the left hand side to begin with. The left hand side of a rule is a set of patterns that conform to the syntax of the working memory elements because eventually they are going to match working memory elements.

The difference between working memory elements and patterns in rules is that the value field in patterns can have variables and it can have tests which can be Boolean tests for example; we will see examples. Also, there may be what are called as negative patterns. We will see what is the semantics of a negative pattern as we go along. In fact, there is a example right here. So, this is an example which of a rule for assigning ranks to students essentially.

So, we have already seen the kind of data that we may be working with. But if we speak of this rule in English, we can see that the what this rule says is that you may have assigned. So, remember that you know this is problem solving in progress and you may have assigned some ranks to let us say the toppers and then, you may say there is one rank which is the next rank which is available and let us call it r.

Now, as we said patterns have variables and a variable is denoted in this angular brackets; anything inside an angular bracket is a variable. So, here for example, I have said this is a variable and r also is a variable essentially.

So, what this rule is saying is that if the next rank that is to be assigned in r. So, you can imagine that in your working memory, there would be one record which will tell you what is the next rank and this is essentially saying that look at that record and let that variable let that rank be called r essentially.

So, we are just calling it by a variable name. It is only one record in your working memory and then, you are saying and there is no and there is a student, we do not care who the student is the pattern only says that the student should have some marks m. Again remember, this m is a variable and it has not been assigned a rank as of now essentially. So, the rank must match nil essentially ok. So, r is the rank that you want to assign.

So, for example, r could be 2, 3, 4, 10 whatever the case is and we are saying that here is a student who has not been assigned the rank and the student got m marks essentially. But what is more important, why should we choose this particular student and this is captured in a negative pattern. It says there is no student with higher marks than m, I could have written m here.

I just wrote it in more English lines fashion who has not been assigned the rank still essentially. So, essentially what are you saying? You are saying that of all the students that are there in your working memory, you look at the students who do not have a rank that have

been assigned, pick a student who has got some marks m such that nobody else has got higher marks. Then of course, that student can be given rank r ok.

So, I have not stated the right hand side here, but we will do so as we go along. So, let us finish this rule and then we will take a break and hopefully it will give you some time to digest this slightly different way of looking at programming. So, this is how our left hand side of the ranking rule will look like in our language that we have decided. We have given it a name, it is called ranking. There is a pattern; the first pattern here, it says that is there a working memory element with some value of r.

So, since it is a variable, it will match anything. We will see this as we go along. Is there a record which is called total marks in which there is some student, we do not care who the student is and has got some marks we again, do not care about those marks. But we do care that the student has not been assigned the rank here and the negative pattern comes in here. It says that there is no record in which a student has marks greater than m and rank nil essentially.

Then, we are ready to assign a rank to this student essentially ok. So, so far in this rule-based system, we have looked at this notion of working memory. We started by looking at the notion of working memory elements, we saw that these are the collection of records and records of class names with attributes names and every attribute name is associated with a value.

A working memory is the collection of such records and it has apart from the data itself, it has a time stamp which tells you when the working memory element was created or added to the system and we have rules whose left hand side match some statements in the working memory elements.

So, how many statements will this particular rule match? It will match two statements; one is that there must be an element called next with whatever the rank is and there must be a student who satisfies this criteria that it has that she or he has not been assigned the rank has

got marks m and there is no one higher and that is a third pattern. The third pattern will not match anything obviously, because it says that there must be no such pattern.

But two the first two patterns must must match specific working memory elements. We know which student we are talking about and we know what rank the student should be assigned. We will take this up after a break, when we look at matching in more detail and also the actions in the right hand side.