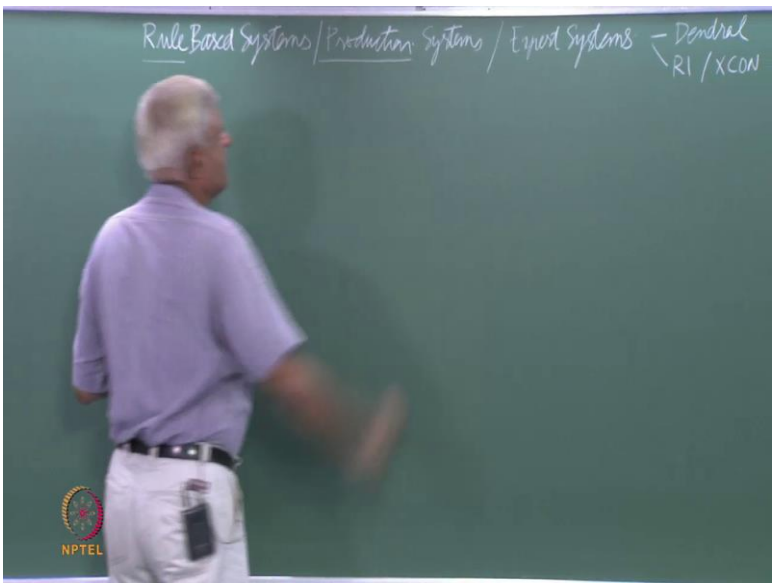
Artificial Intelligence
Prof. Deepak Khemani
Department of Computer Science & Engineering
Indian Institute of Technology, Madras

Lecture - 30
Rule Based Systems

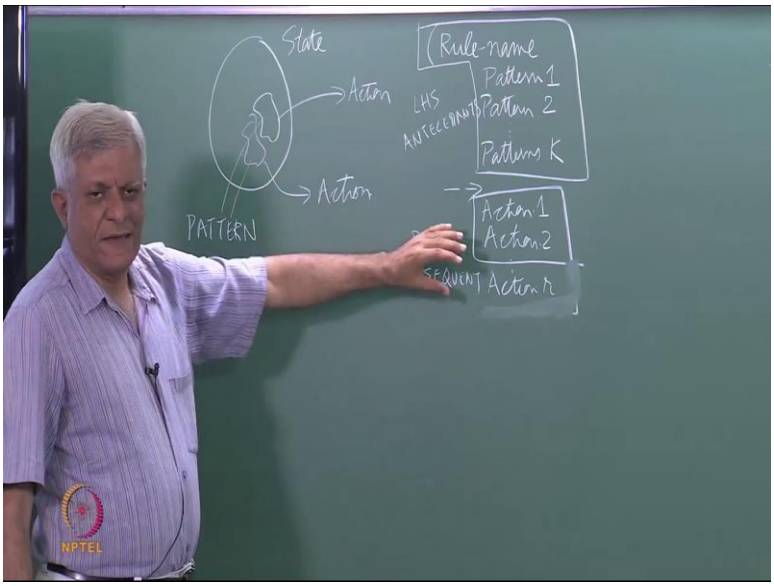
So, shall we begin? So, the last few, the last few lectures that we had on games was a bit of a diversion intended to get you ready for the assignment, which you will soon give you, possibly next week. We come back to this idea of problem decomposition.

(Refer Slide Time: 00:35)



And today we look at this thing called rule based systems, also known as production systems, also somewhat ambitiously known as expert systems. So, far in our discussions we have already mentioned 2 systems, one is Dendral and the other one is R 1 also known as X CON which were supposed to be 2 of the first so called expert systems. And by this we mean that programs which harnessed the knowledge of human experts and use that for solving problems essentially. So, this knowledge of that we get from human experts in this form that we are talking about was generally in the form of rules or productions essentially.

(Refer Slide Time: 01:58)



And the idea is that if you have a state description, then instead of devising a moves and function which gives takes one state and gives you another state. What we do is we look at some part of the state and based on this part of the state we device, let us say action one or another part of the state we look at we device another action.

So, in some sense we are not looking at the complete state, but we are looking at what we call as a pattern in the state. So, these things are patterns. Now, this pattern is typically described in the form of a production or the rule and I will use the syntax of a language called Opius 5, which I will shortly tell you. But the basic idea is that the structure is as follows.

So, you have a rule name and you have pattern 1 and pattern 2 and so on, some number of patterns and I will shortly describe what these are followed by action 1, action 2 some number of action essentially. So, this is the structure of a rule also known as a production. And the basic idea is that this is called the left hand side this is called the right hand side because of the arrow here. This is also called the antecedents and we can think of this as a consequent.

So, a rule is basically a set of antecedents followed by a set of actions or consequents and it is a very modular form of knowledge representation. We have so far not used the word knowledge representation at all in this course, we have assumed that somehow you will have a state representation and somehow you will have a moves and function which will

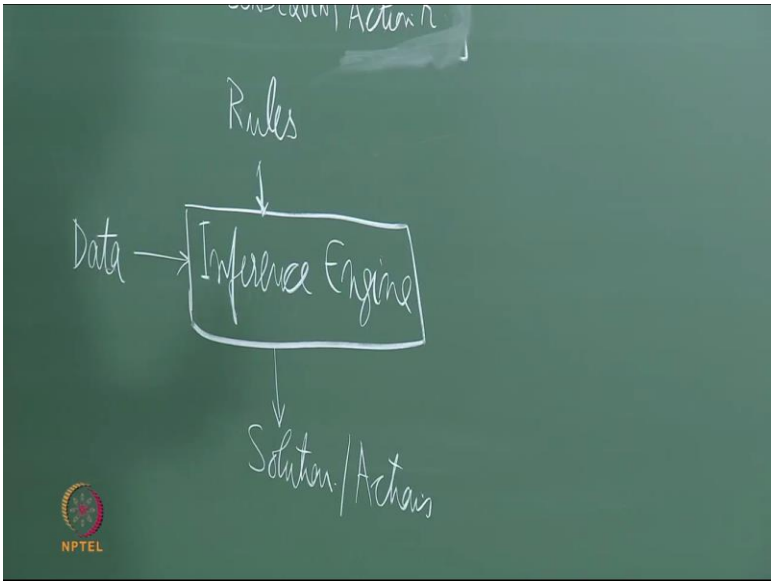
move from one state to another. Now, with this we for the first time we are talking about how do we actually represent things.

And the idea here is that knowledge is represented in the form of rules. So, here on the left hand side followed by the right hand side here. So, the left hand side that we are talking about here is essentially the pattern that we are looking at and if the pattern matches then the action can be done. So, for example, the system R 1 had rules like. So, you should look up the actual rule in X CON or maybe I can just read out a rule from here. This says, this is the rule from R 1 its rule name is distribute (Refer Time: 05:36).

So, remember R 1 was the system which was used to configure bags machines, deck vack machines which was some of the most advanced computing systems at that time which was in the late 70s. And this rule reads the following, if that is the left hand side, the most current active context is distributing mass first devices and there is a single port disk drive that has not been assigned to a mass bus and there are no unassigned dual port disk drive and so on and so forth.

And then the action is, then assigned the disk drive to the mass bus. So, it has a set of conditions on the left hand side and in this example only one action on the right hand side essentially. So, the idea is to capture knowledge of a human in this form and use this knowledge to solve problems. Now, how does that happen?

(Refer Slide Time: 06:45)



So, there is this idea of an inference engine. And this inference engine takes on the one hand data and on the other hand rules and it generates the, I will just use the loose term to say the solution or a sequence of actions. So, the idea behind rule based systems which also were for some reason called expert systems and production systems was that the problem solver or the human expert or the domain expert will only provide the rules for solving problems.

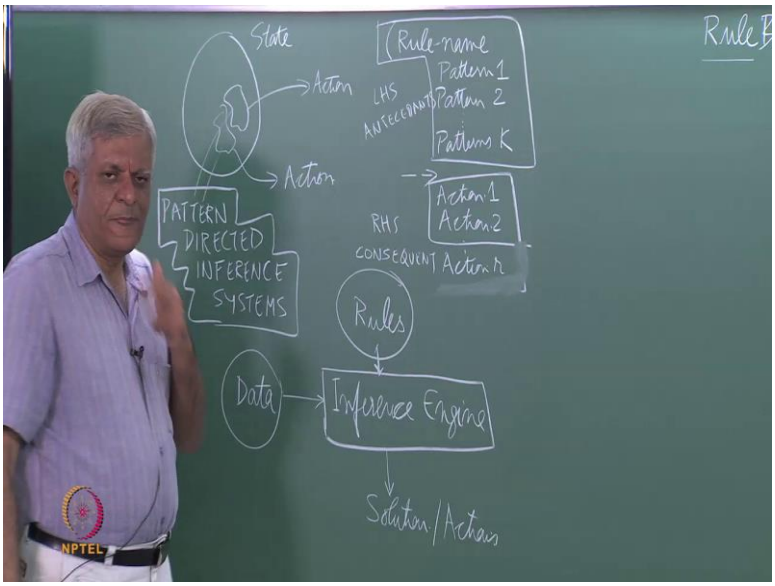
So, rules like this if some conditions are true then some action happens, if some conditions are true then some action happens and so on. So, you might say for example, you might have a rule which says is a particular customer worthy of being given a loan? Then a bank manager may have a rule of certain kind that if this persons income is so much and if he is been a customer for so many days and if he has no outstanding loans then you can give him a loan of whatever 10 lacs or something like that.

So, you can have a rule like this. Now, the thing is that if we allow the domain expert to express rules like this then they do not have to worry about program which works on this rules. They only write the rules and somebody else in this case inference engine does the work for them essentially.

So, this inference engine is the one which will actually pick rules from the collection of rules when a new problem comes beside which rules to apply and then generate the solution essentially. Alternatively you might have a rule, the railways might have a rule based which says who gets what kind of concession and so on so. And if you are a war veteran you get a certain kind of concession and if you are a sports man you get a certain kind of concession. If you are a student going home you get a certain kind of concession and that kind of stuff.

So, all these so called what which people now days call as business rules are described as a domain people. They are not worried about computer, computing and programs and things like that. There task is to only provide the rules essentially. We could have a rule for example to say, good student award which might say that if the student has done all the assignments and the student has done well in courses. And if there are no proxies in the attendance then he is a good student we can give him an award or her an award. Focus we do not do that. So, you do not have to worry about it.

(Refer Slide Time: 09:40)



So, the basic idea behind rule based systems is this that a rule is a pattern, action, combination, there is a collection of rules. So, this is a whole set which the domain expert provides. This data pertains to the current problem that you are trying to solve and inference engine. And we will look at this in some detail today. Is the one which does the job of picking rules and applying them to data and arriving at the solutions essentially. Which is one reason why some people have called this as pattern directed inference systems. In fact, there is a book by this name, it is a little old.

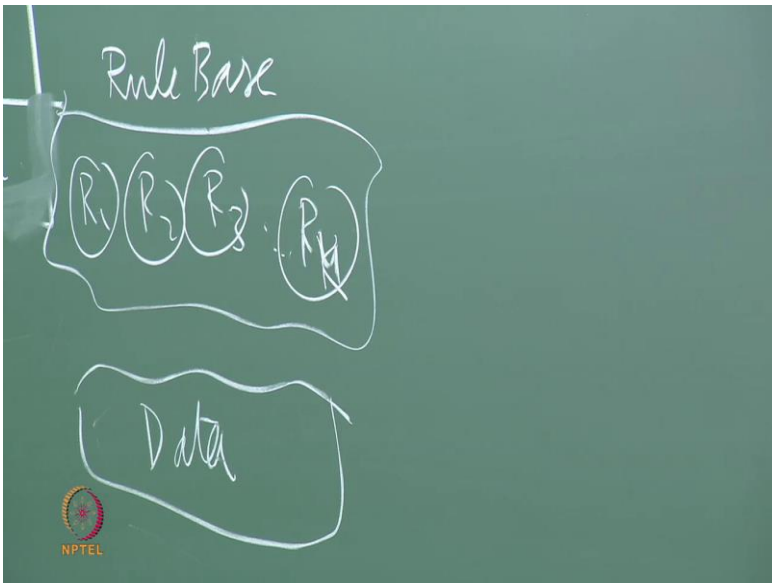
Now, the thing for you to observe and we will do this over the next, this lecture and the next is that rule based programming can be seen as a different paradigm of programming in itself essentially. Now, in imperative programming which is what most people are used to languages like c and Pascal and so on. The task of the programmer is to give a control flow, say do this action, then do this action, then do this action.

And of course, you may have more complicated things like conditional statements and branches and loops and all this kind of stuff, but it is the programmer which specifies what actions have to be done essentially. Now, you can think of a rule based system, you can in some sense port it to an imperative language and you can. So, remember that the rule based systems basically consists of a large number rules set, the domain expert rights.

You could convert it into an imperative program and you could have a sequence of if

then statements, if this then this else this then this and so on and so forth. The control flow is fixed and rigid in imperative program, in rule based systems. On the other hand you can think of this as follows that you have this data...

(Refer Slide Time: 12:00)



in some pool and there are a set of rules hanging above this. And when I say hanging above this, this has to sort of make you think about it slightly differently.

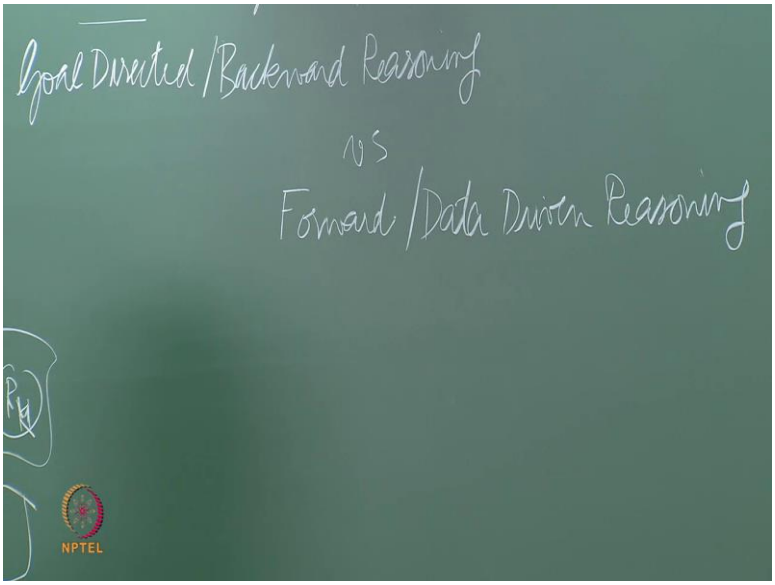
In the sense that there is no control relation between the different rules. You are not saying apply this rule first then apply this rule first and then apply this rule first and so on. You are simply saying in each rule you are giving a little bit of modular piece of knowledge we say if you see this pattern then this is the action that you have to do with it essentially. And all the rules are kind of floating around.

So, this is a what we would call is as a rule base and what the inference engine does is that it will pick some rule and apply it to the relevant data and do the relevant action. Then they may pick another rule and apply it and so on and so forth. And it will keep doing that. It is a inference engine, and we will look at this in a little more bit detail, which decides what are the actions which are actually done. As opposed to an imperative programming who programmer says do this action then you do this action and so on and so forth essentially.

So, before we get into the details, I also want to highlight the difference between 2 kinds

of reasoning and we have, we have slowly talking about this thing we have not really done so.

(Refer Slide Time: 13:40)



One is goal directed or backward reasoning and by this we mean that you reason from goals to sub goals in some sense. We have already seen an example of this when we were talking about planning an evening out when you say that evening out is planned if you have an outing plan and if you have a movie plan and if you have a dinner plan. So, those are the sub goals essentially.

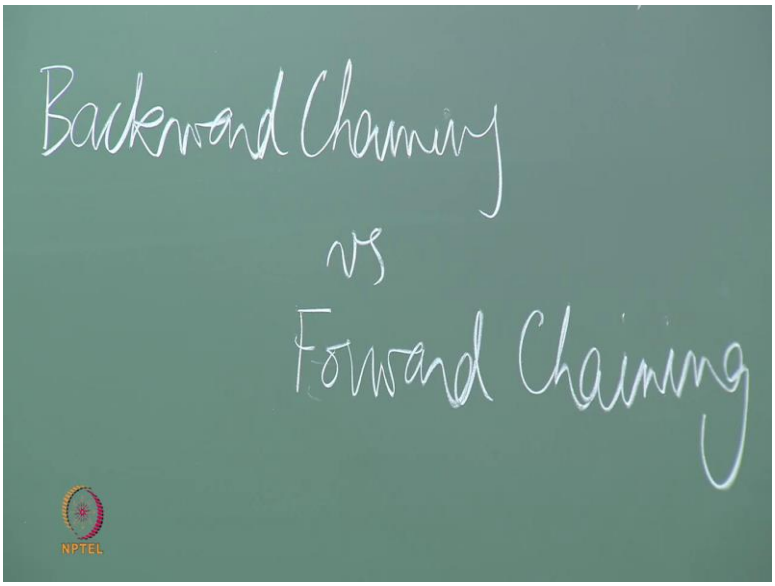
So, you start with a major goal and then you come to the sub goals essentially. And there are people, a lot of people who feel that human beings by and large are goal directed reasons. They are backward reasons, they do not reason from. So, this is goal directed versus forward or you might say data driven reasoning. So, data driven reasoning is some kind of an algorithm which looks at data and says that if I say this pattern, I will do this action and so on and so forth.

So, the decision of what to do next is driven by, is decided by the data that is available to you, which is the form that we will be looking at today. Goal directed reasoning says that if you have to achieve a certain goal, what sub goals should I achieve essentially? So, for example, you might say that if you want to get a scholarship into a good US university for doing post graduate studies then what do you need to do?

You need to write g r e, you need to do well, you need to get recommendation letters, you have to write a excellent statement of purpose all this kind of stuffs. These are sub goals and to do that to get good grades what you have to do? You have to attend classes and study hard and write exams well and so on. So, if you go from goals towards sub goals and essentially, eventually towards actions then you can decide what are the actions that you need to do essentially.

Again let us say that if I have to plan a trip from here to mandi then in goal directed reasoning I would say that first maybe I need to go to Delhi and then from Delhi I need to fly to Chandigarh or take a train to Chandigarh and then take a bus or car or something and then work out the details later essentially. How do I go from here to the airport? Now, how do I book my ticket and how do I go from? The lower details come later. In goal directed reasoning, you reason from the high level goals to the low level goals and at the lowest level you have action essentially. Now, there is a corresponding implementational comparison that you can make.

(Refer Slide Time: 16:45)



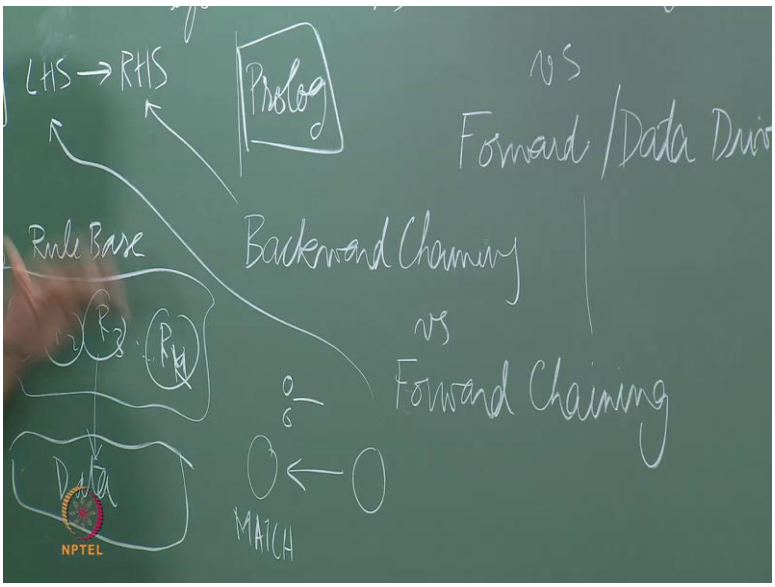
It is called backward chaining versus forward chaining. So, by backward chaining and forward chaining we essentially are talking about how the programs are implemented whereas with backward reasoning and forward reasoning we are talking about what is the kind of reasoning you are doing. So, by and large of course, backward chaining goes well with backward reasoning and forward chaining goes well with forward reasoning.

But it is possible to do forward chaining and implement backward reasoning or it is also possible to implement backward chaining and implement forward reasoning essentially. So, this combination of backward reasoning and backward chaining sorry backward reasoning and forward chaining is something or actually you should call it backward chaining.

So, this combination for those of you have written programs and prologs you would recognize that in prolog you are doing essentially backward reasoning and if you view the sign, this sign in prolog as in backward pointing arrow which we will do when we look at logic a little bit later in this course. We will look at this then you can see that you are essentially chaining from this to this. So, in backward chaining you are chaining from the right hand side to the left hand side which means you match.

So, this is actually the right hand side, why because the arrow is pointing this way I could have drawn it like that, but in prolog you write the goal first and then you write the sub goals, if you remember prolog that if you want to sort an array then you have to have a combination of array which should be sorted and so on and so forth. So, it really depends on the match which side. So, if you look at this rule format LHS and RHS then in backward chaining you are matching here with the RHS and moving to the LHS. So, you are asking that if this has to be true then can I show that this is true.

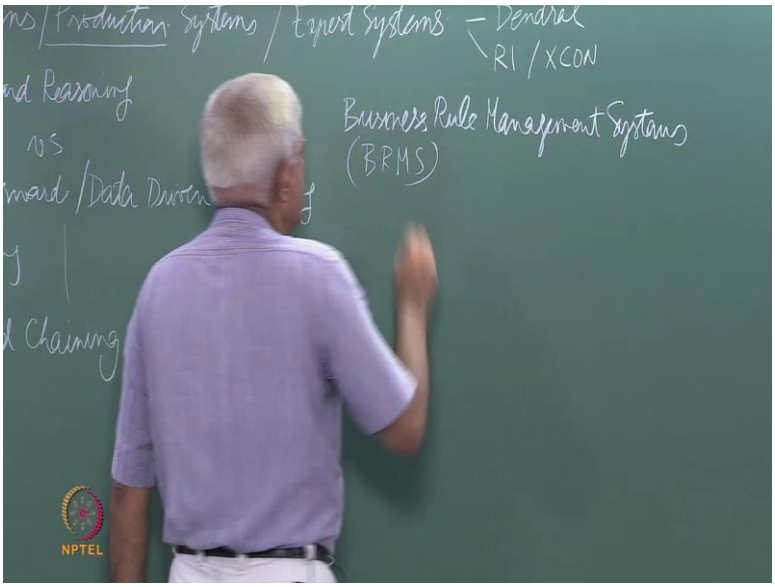
(Refer Slide Time: 19:08)



In forward chaining, I match the LHS and go to the right hand side. If I see this pattern

then I do the sections. So, we will be focusing today on forward chaining mechanism because it is a very widely used technique in the industry. So, I gave you a couple of examples about banking and giving loans and so on and so forth.

(Refer Slide Time: 19:31)

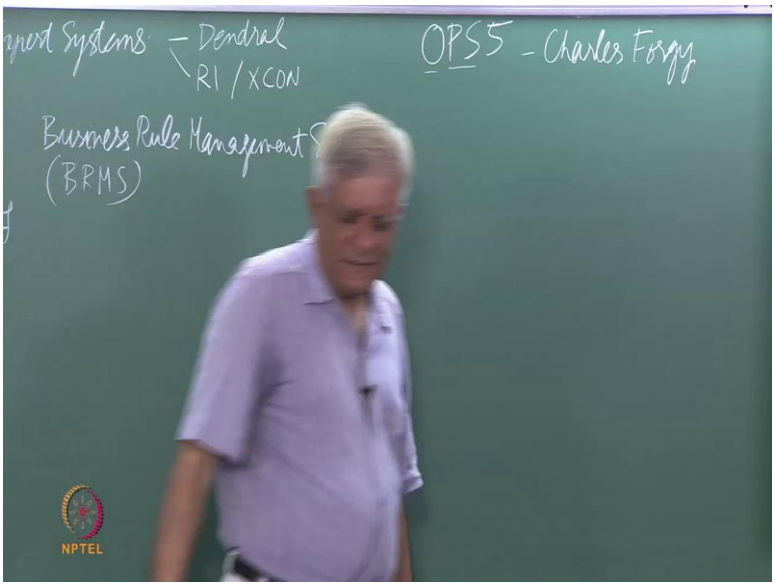


But there is this whole field called business rule management systems. So, if you just look at, look up this thing B R M S, there is a whole community out there which is developing software for people in business meaning it could be industry, manufacturing industry, banking business or anything.

People who are domain experts, to exploit computing power the basic mechanism is this that all you ask them to do is write the rules and you implement the inference engine which is the general purpose inference engine. It is like you can say a search engine is a general purpose program and in similar manner inference engine is a general purpose program which looks at their rules and their corresponding data and makes all the actions which are necessary to be done essentially.

Now, let us get down to some examples. So, the language that I am using, we do not really have to use it and in, I mean you may not have to implement it is called O P S 5. This is the language is devised, which was devised I think in the 70s,0 also at (Refer Time: 20:56) university which is one of the centers where a lot of a i related work was done in the middle of the last century.

(Refer Slide Time: 21:05)



And some people say that O stands for official. So, it is a official production system language and its version was 5 became popular. It was developed by a guy called Charles Forgy who implemented this algorithm that we are going to look at for his p h d thesis and then eventually of course, it became something which was commercial in nature. So, of course, what we need to do is to describe what is this language, what is the syntax of this language. And so I will focus we will first focus on the pattern. So, how do patterns describe in OPS5.

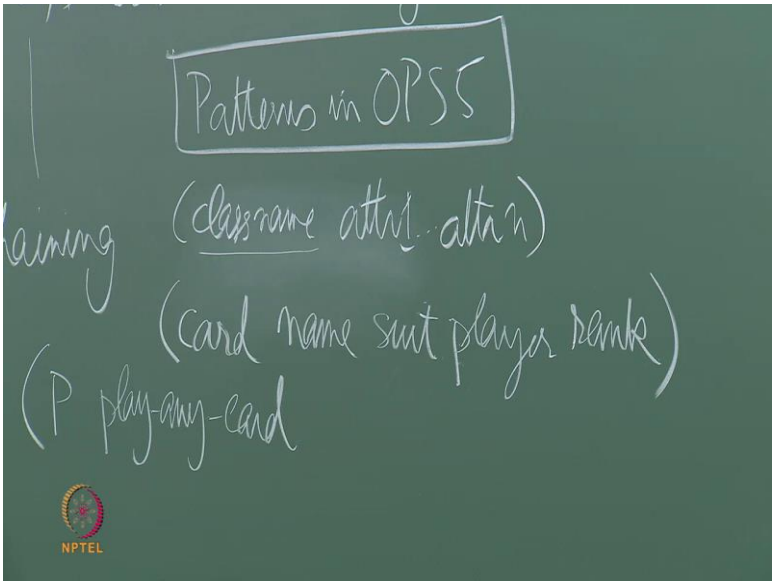
So, they follow the, something like a, like a object oriented way of representing things which are like classes. The syntax that this language uses is that it uses p to stand for production then name oh so sorry that is the language of a rule. So, we will come to rule a bit later. First we want to talk about patterns. So, there is a class name followed by attribute names eventually. So, very much like the class name in a. Remember that patterns are these things which are will which constitutes the left hand side of a rule and each pattern is made up of a class name followed by a set of attribute name.

So, this is the data structure that we are using in this language OPS5. So, for example, if I am implementing a program to play cards, I might have a class name called card then I might have a name of a card for example, ace or queen or 10 the suit of a card. So, for those of you who are familiar with cards spades and diamonds and clubs and so on. I might have the name of a player who is holding that card and I might have rank of the

card. So, by rank I mean the current rank.

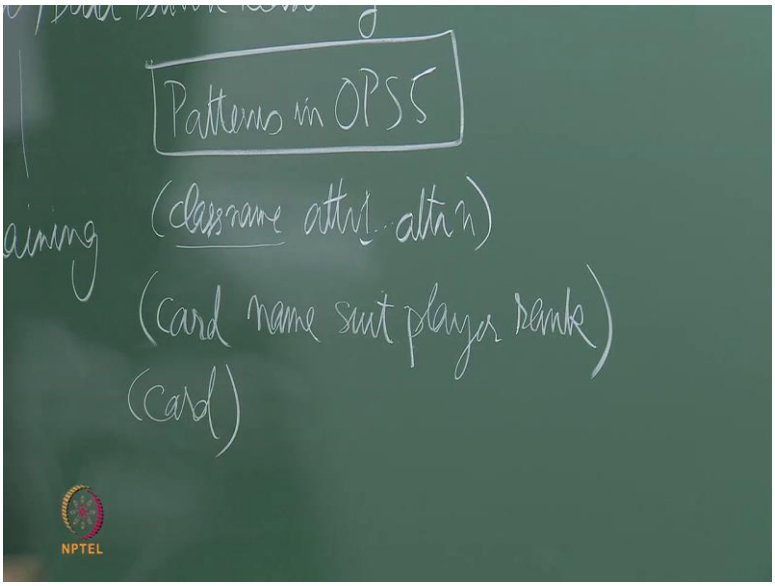
So, in most games for example, ace is rank 1 and king is rank 2 and so on. But as the game progresses these ranks might change. So, I might want to store that essentially. So, this is a, this is a class description and correspondent to this class description I may have several patterns.

(Refer Slide Time: 24:31)



So, the patterns are the left hand side of the rule and a rule is signified by this opening bracket followed by p which stands for production, and then the production name for example play any card. Let us say I just want to write a rule which says play any card that you have essentially. And in this rule I might have a pattern which says that you have a card and I might just specify this essentially. So, let us, I will come to this later. So, next let us just talk about the different kind of patterns that we can have.

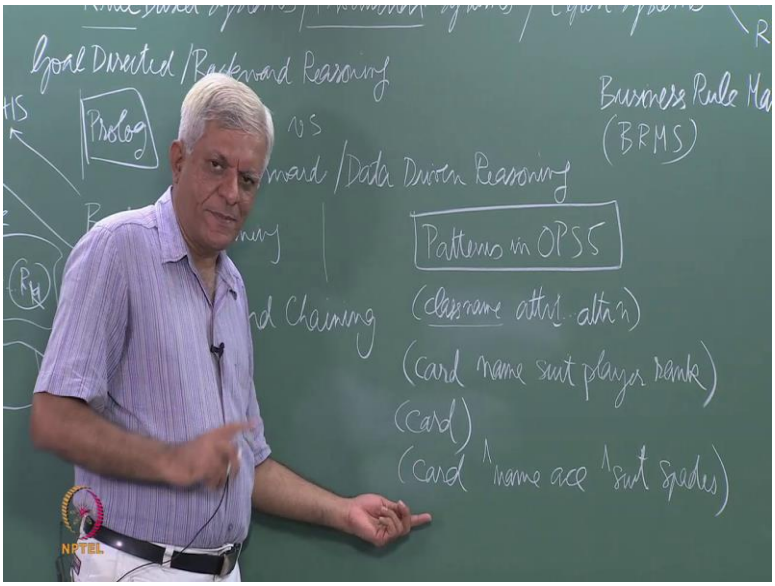
(Refer Slide Time: 25:18)



So, we can have just a name of the class as a pattern which means that if the inference engine can see one data element in your data. So, remember we have this data which has a class name card then that will match that would. Our task is to eventually going to be the match the pattern and see which patterns match. Of course, there may be more than one rule which may be matching, we will come to that later and then what action can be done.

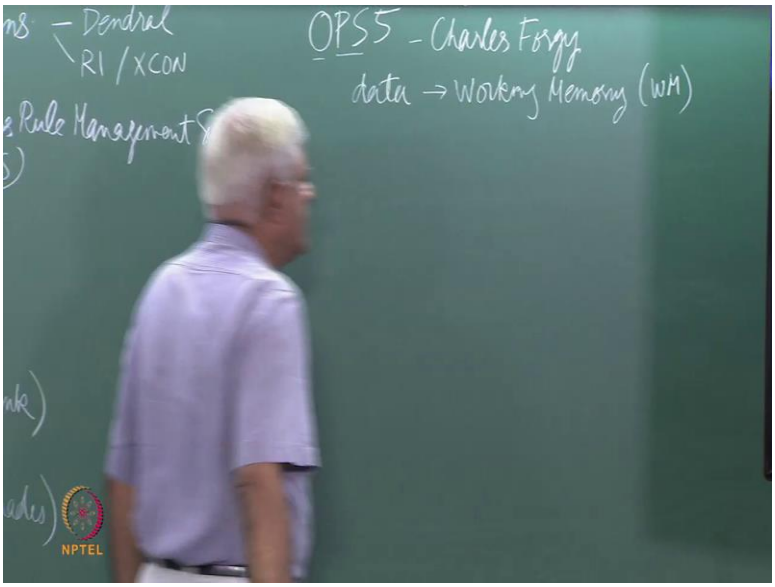
So, for matching the rule we have to match patterns and each pattern should match some class description. So, if my this class name is card and its attributes are name, suit, player and rank. Then the pattern must be an instance of that essentially.

(Refer Slide Time: 26:12)



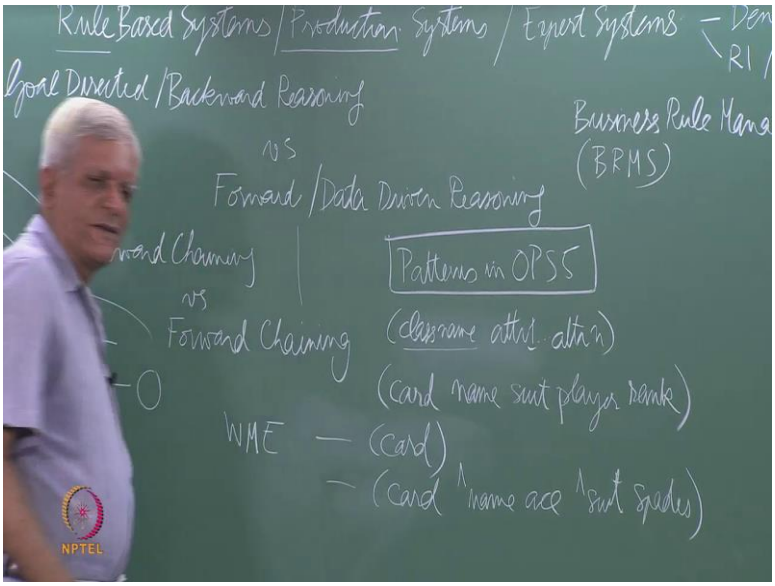
So, I can have for example, card, name, ace, suit, spades. So, here I am saying that so this is the way data is represented. This is the way the matter rate as given as to which describes the data. So, data itself is written like this.

(Refer Slide Time: 26:48)



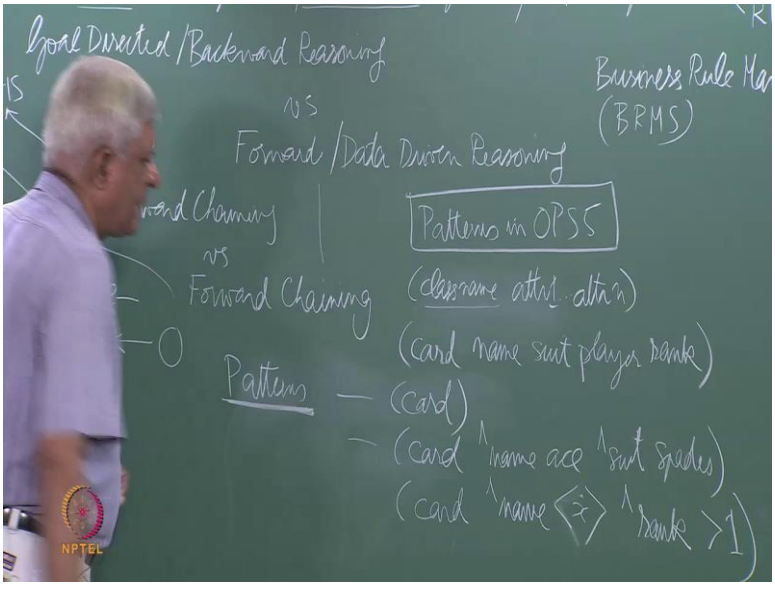
So, I have a element which OPS5 data is in the working memory which is called WM.

(Refer Slide Time: 27:06)



And each of this is called a working memory element, these is just the terminology that they use. So, this is a working memory and inside these of course, there are this working memory elements where each element is an instance of some class name which maybe not completely specified, which maybe incompletely specified.

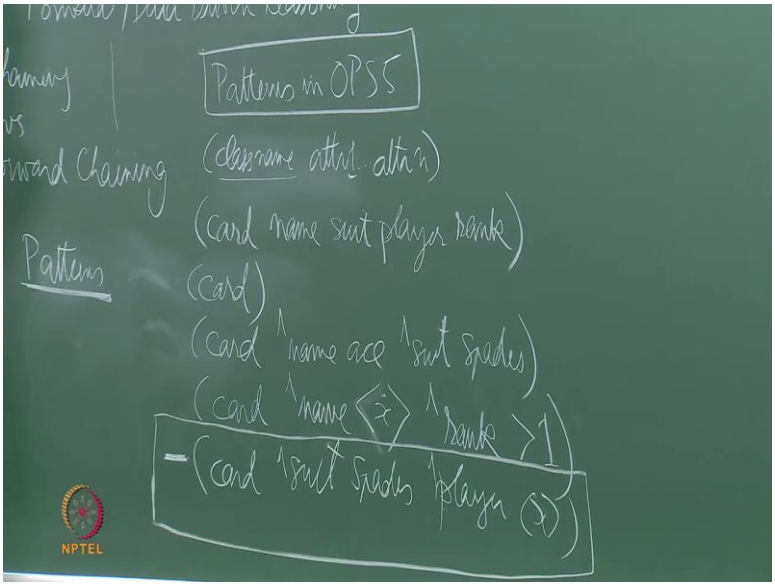
(Refer Slide Time: 27:29)



So, for example, here I have only said that there is an element which belongs to the class name card. Here I am saying there is an element which belongs to the class name card and the attribute name has value ace and the attribute suit has value spade essentially.

Of course, I can specify other things also, but any partial description can be there in the pattern. We are talking about patterns in rules essentially. I can say things like card, name. So, this angular brackets are used to distinguish variables from constants. So, ace is a constant, but anything within angular bracket is a variable and the variable is which will match anything essentially. And I can have other kinds of relation. So, for example, I can say rank greater than 1 and so on. So, these are the patterns. So, the left hand side of the rule is made up of collection of such patterns.

(Refer Slide Time: 29:06)



You can also have a pattern which says, it is a negative sign here.. So, this pattern should be interpreted differently. It should be interpreted as saying that there is no such pattern in your, in your working memory or there is no such data element in the working memory which says that player x, this is a variable x has a card of spades.

So, such a pattern would be used to describe the fact that a given player does not have any spade cards in his or hand, her hand. So, this negation sign basically is the opposite of this. This says that this must be present in the database or in the working memory. This says that it must be absent in the database and only then this rule will match essentially.

We will look at some examples. Now, actions so remember we have patterns and we have actions. So, the 2 most important actions in ops 5 is one is called make and after this you describe the working memory element. So, you can say for example, card,

spades sorry suit, name, age and so on and so forth. So, what this action does?

This makes action. It creates a working memory element and puts it into the working memory, how it otherwise it creates a data element and puts it in the database if you want to call it a database. Corresponding to this there is also a remove action. Now, a remove action takes an argument like 2 and this 2 as you will see in the example the first 2, the second pattern in that particular rule. Remember these actions are on the right hand side of a rule and this action is saying that remove the pattern, remove the data element which match the second pattern. So, this pattern number 2 matches some data.

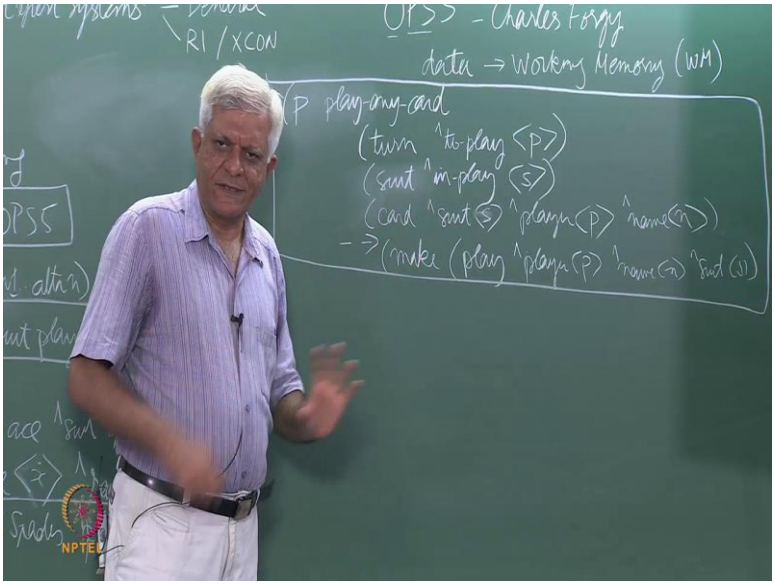
So, that will match something, some working number x let us call it. And this action is saying that, remove that from the working memory or remove that from data essentially. If you can combine this you can have a action called modify. So, for example, as the game progresses you want to change the rank of a card essentially. So, let us say somebody has played the ace of spades and now you want to say king of spades has rank 1 then you could say modify that particular working memory element and change the rand to value 1 that is all.

So, we have the so basically we have this 2 actions make and remove, but these can lead to a third action called, any modify action can be seen as a combination of make and remove, remove the old one and put in a new one, it is like modify. Apart from that you have standard actions that you may need for example, reading from a file and so on which we will not go into here because we want to focus more on how reasoning is done.

(Refer Slide Time: 32:54)

So, there are actions like read, write, print. There is even a action called halt which says that stop the system essentially and exit essentially. So, we will not look into those details here. So, let me now give you a examples of a few rules and then we will see how the system operates with these rules essentially. So, I will write these rules for a card game and I assume that everybody has some in cling of what card games are like essentially.

(Refer Slide Time: 33:35)



So, let us say we have a rule called play any card and the pattern is let us say there is a

class name called turn which I have not described. I have only described one class name here which is this. But obviously, you will have many different class names in your system.

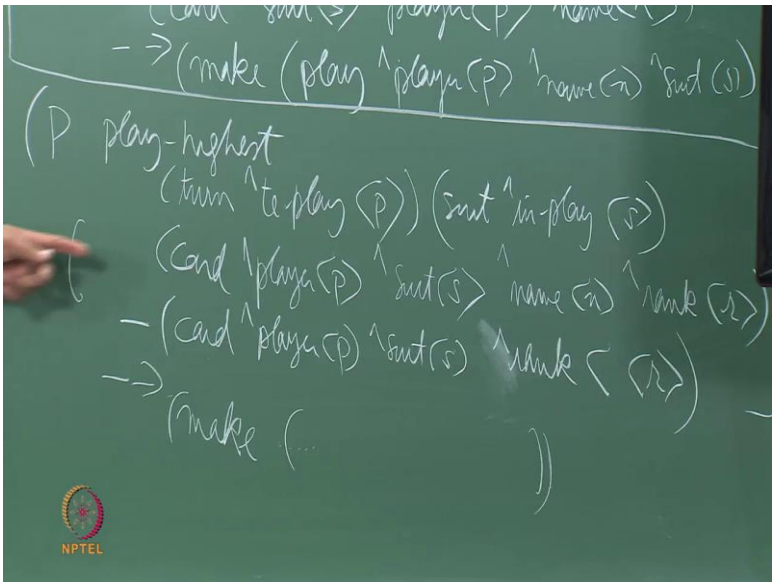
So, let us say that you are carefully keeping track of whose turn it is to play next into one data element which is belongs to this class kind type turn and it says turn to play of some variable. So, we have not specified who, some player p essentially. And let us say that already somebody has started playing a particular suit. So, in many games in which you have to play the card of the same suit at somebody else has done. So, this is saying that the suit that is in place s, some variable name and this is some player p and this player has a card of suit s.

So, notice that there is no order sanctity here essentially. I may say that this name, suit, player, rank, but that does not matter because we are segregating the attribute names with the attribute values. You can specify the attribute names in any order. You do not have to specify them in this particular order. So, here I said card with suit s, player p. So, all I am saying.

So, whatever I say here that if it is a players turn to play and the player could match anything. And if the suit in play is s and if there is a card which is of suit s and being held by player p then that player p should play that card essentially. So, let us say I use this make. So, this is one rule and it has 3 patterns on the left hand side which says that there are 3 different class names.

Of course, we are interpreting it as a card game that, if it is p's turn to play and if the suit in plays s, and if p has a card of this suit s whose name is n then make working memory element called play player p that card essentially. And then of course, maybe we will have another rule which will take it and print it onto the screen or something like that let us not bother about that.

(Refer Slide Time: 37:27)



So, let me give you another rule just to so if you want to not play any card, but you want to play the highest card essentially then what should the rule look like. The first 2 will be the same, turn. So, let me write this here. So, the first 2 patterns are still the same as this rule, but the other patterns will change.

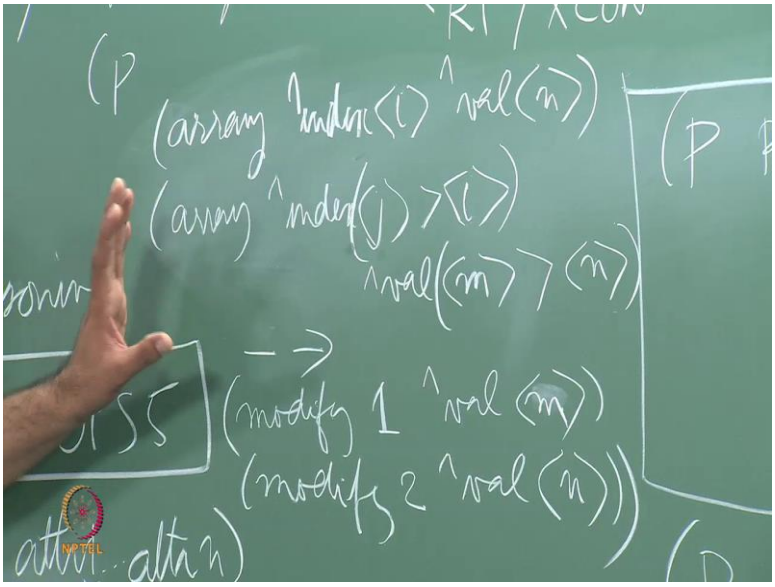
Now, you want to specify that the card that you play must be the highest card with your suit and we can do that by saying the following next card, player p, suite s. So, first of course, we are looking for a card being held by this player and in this suit. Let us say name n and rank r because we are interested in picking the highest card we should talk about the rank and we should say that there is no higher card essentially, which means we can say that there is no card the same player p, name we do not care about. So, this the that negative thing that I was telling about. So, between these 2 patterns we are looking at a set of data elements and picking the one with the highest rank which means the lowest number. So, rank 1 we want to look at or rank 2 whichever is the lowest available.

So, we are saying that card, player p, suite s, name n, rank r and there is no card with this player p of this suit, same suit s whose rank is smaller than this r. So obviously, this r and this r must match the same value then the same thing. And you can write more rules depending on you know what kind of game you are playing and what is the strategy you want to adopt and that kind of stuff, but we would not get into those details here

essentially.

But you should observe that this one. This combination of this positive pattern and the negative pattern can be used to pick the highest rank. So, for example, I could use such a rule, variation of this rule to start grading or something and say let me pick the student with the highest marks and do something and then you know that kind of stuff. So, this can be done. So, the illustrate the power of this let me show you another rule. Supposing that I have a array of numbers which is signified by index and value.

(Refer Slide Time: 41:12)

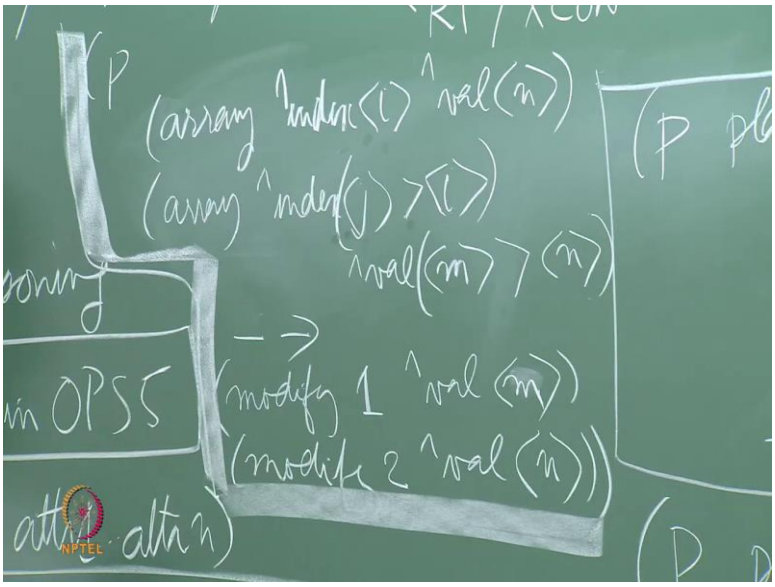


And if I have a rule of this kind or this pattern which says that array index i, value n and array index j greater than i. So, I can do like this, I can put this in brackets and say that the index value is j and this value j is greater than this value i that we are talking about here, which means if I am talking about the 5th element here I could be talking about the 7th element or something like that. Let us say value m which is greater than value n. So, I am not. So, some rule it is there.

Then I do the following, modify one value n. So, I have 2 actions and 2 patterns. The first pattern let me read it out it says that index value is i and the value is n. The second pattern says index value is j which is greater than i and the value is m which is greater than n then I am saying modify this first pattern. Now, the data matching that first pattern and change its value attribute to m, which is what I have taken from here and modify the second pattern and change its value attribute to n essentially.

So, what am I doing here? I am swapping 2 elements if they satisfy this condition right. Now, this single rule if I let it loose on the data then it will end up sorting as the same the entire array essentially. So, just think of you have written a algorithm for sorting elements. You have looked at all kinds of sorting algorithms and you wrote them.

(Refer Slide Time: 43:53)



But here is one single rule, this much. And this will do the job of sorting for you essentially. It will repeatedly apply the rule and we will, we will come to this repetition part in the next class, but it will repeatedly apply to, if it is, it is if any 2 elements out of place it will swap them essentially and it will keep doing that till this rule matches and eventually of course, everything will be in place or sorted and then this rule will no longer apply and your array would be sorted essentially.

So, this also gives you something to think about you can write a program. In fact, ops 5 is a complete programming language like any other programming language like prolog or c plus, plus or java or python or whatever. And you can write any program in any language, but in some languages some programs are simpler to write. So, this program of course, is very easy to write, but what can you say about this performance? It will depend on so many other things which we will address in the next class.

So, supposing you have many elements which are out of place. This inference engine here has to decide which instance of the rule to execute next and therein of course, lies the key to efficiency. If you exchange the correct elements first then you will do things

faster and so on and so forth. But of course, there is no way of knowing that here, but it is a very simple program to sort in this essentially.

So, the idea of rule based systems is that somebody sits and write down, writes down this rule in this thing and then we give it to an inference engine which basically does the inferencing for us. Which means basically it picks a rule, applies it then picks the next rule then applies it and keeps doing that till it either runs out of rules to apply or it runs into something like a halt statement here which we have explicitly given.

So, maybe if you have a certain goal in mind and if that goal is achieve you will say if this goal I can see the goal being achieved then halt essentially. You do not have to keep running the program essentially. So, in the next class you will look inside this inference engine and it is a very well known algorithm which is very popular and has extensive use in the industry for developing these systems.

So, basically the idea in the industry is that we will write business rules, we will say when to give a loan, when to do this, when to do that, but your program must do the computations for us. They must, your program must accept the rules and find the solutions that we expect the program to produce. So, in the next class we will look at inside this inference engine and how it works essentially.

And we are going to focus on the forward chaining inference engine, a backward chaining engine is already you are familiar with something called prolog. And we might just look at it a little bit later on in the course, but today we will focus on forward chaining essentially. So, I will stop here.