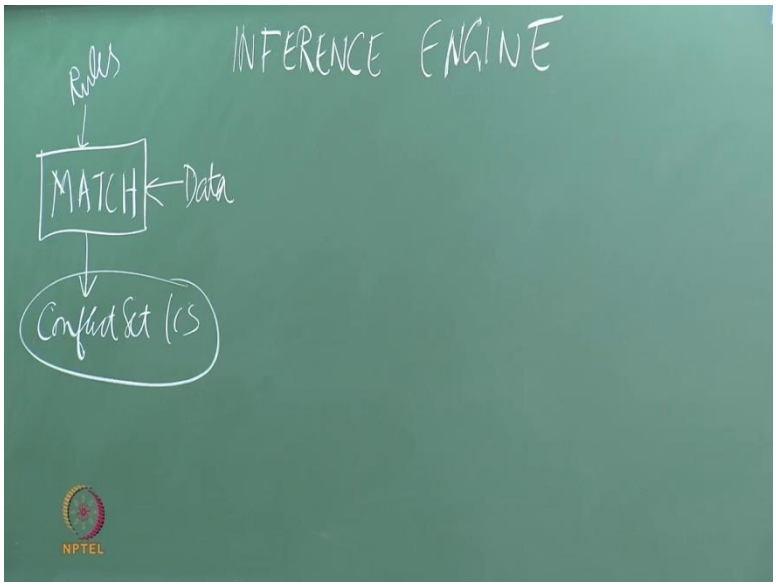
Artificial Intelligence
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 31
Inference Engines

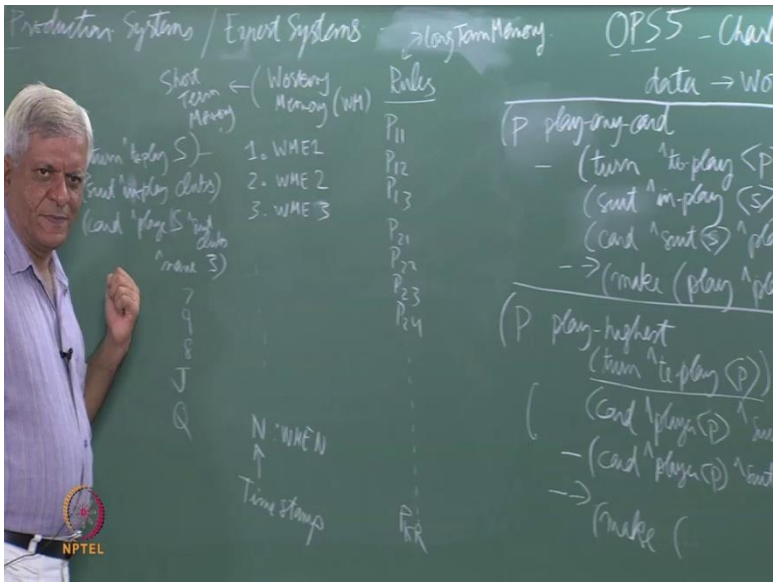
So, we are looking at rule based systems power changing systems, and now we want to look at the inference engine.

(Refer Slide Time: 00:22)



So, the inference engine is basically a program which works in three phases. The first phase is called match, and what it takes as input is a set of rules and some data and produces a set of matching rules which in their terminology is called the conflict set. It produces the data. So, let us first see what is match is doing.

(Refer Slide Time: 01:30)



So, if you have these rules. Mainly notice that we have written this two rules in the last class. This rule has three conditions 1, 2, 3. This rule has four conditions, this is 1, this is 2, and 3, and 4. One of them is in negative condition. So, essentially a rule is a collection of patterns and actions. As far as matching is concerned, we are only interested in patterns. So, we can think of this as P 11, P 12, P 13. So, that is for the first rule. Then, P 21, P 22, P 23, P 24. So, for this rule for example and then, so on.

So, P K R or whatever, depending on how many rules you have and how many patterns each rules has essentially. So, that is rule memory and when Simon annual for talking about this kind of problems. They would associates it is a long term memory. So, they, so, basically rules capture the problem solving solver knowledge essentially, which is always there with the problem solver so, you can think of it is long term memory. Whereas the actual problem that you have trying to solve sets in a memory which we call as working memory or WM which is also ordered. So, you have this data elements which we call working M E 1. So, this is the data item 1 then 2 and so on essentially.

Now, remember that the actions on the right hand side of rules, I allow to delete data which means they might delete some working memory element. So, some things might vanish from the way. But initially let us assume that we have some capital N number of working memory elements. Now, this can be thought of as a time stamp, which says

when was this working memory element created and as far as we are concerned time is just an ordering. So, this is the first element, second element, third element.

We will see that this plays a role in the control of the execution essentially. So, we can also think of these as short term memory. If you want to think of this as how we solve problems essentially. So, we have these two memories, one is the working memory which contains is working memory elements, and we saw examples of that. The other is the set of rules where each rule consists of a set of patterns. So, this rule has 3 patterns, this rules has 4 patterns and so on and so forth. And these are the examples of such rules essentially.

Now, for a rule to match, now the first task is to match which means look at all the rules, look at all the data and compute which rule matches with which data. So, if you look at this first rule for example, it says that to play any card it does not specify which card to play. Now, if you a just starting the game and let us assume that this rule here. So, let us say this is, this one of the working memory element is this here. So, let a say this one is turn to play S. So, let us say S is a name of a player. So, it is S turn to play. So, there is a piece of data, here there is a pattern which says turn to some variable name. So, this will match with this, which with this data essentially.

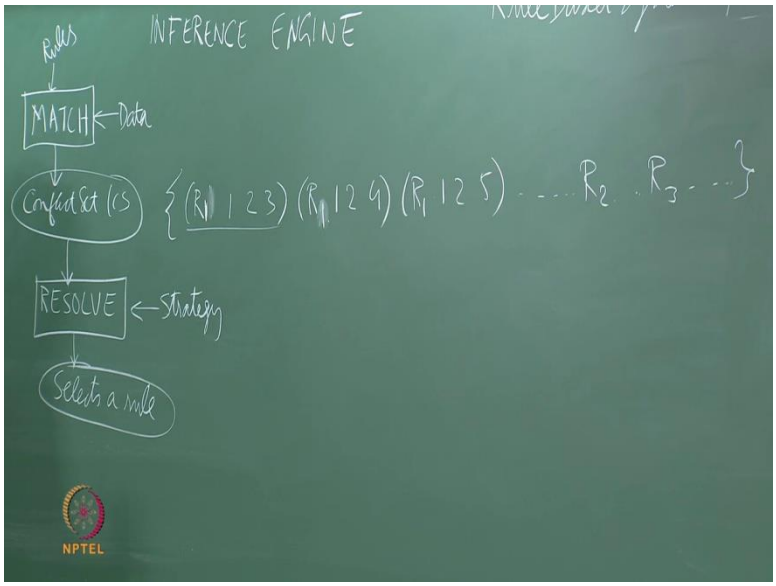
And let us say, this element is suit to play, let us say club. So, this is a working memory element which will match that pattern, second pattern that suit in play, in play. This is a constant. The data will only have constants, variables are only in the patterns and if a pattern has a variable it means it can match anything essentially. Whether this was clubs or diamonds or hearts that rule would still match essentially. And then look at this third element that player P, in this case P is S has some cards of, let us say this player has 6 cards of clubs. So, there would be 6 card, this player is S and suit is clubs.

Let us say name is 3. So, let us say he has 3 of clubs and he has a 7 of clubs and the 9 of clubs and the 8 of clubs and jack of clubs and the queen of clubs. So, let us say this player has got 1, 2, 3, 4, 5, 6 cards of clubs essentially. That is effective and 6 working memory element in which says and each memory element is saying that there is 3 of clubs it is held by S. So, let us say S stand for south essentially. Now, you can see that this rule, the first rule will have 6 instances, for each of these 6 cards 1 instance of that rule will fire will match.

So, 1 instance will says yes it is south's turn to play, the suit in play is clubs and he has got the 3 of clubs. Another instance of that rule will say yes it is south turns to play, suit in plays clubs and he has got a 7 of clubs. So, like this for each of this cards, 1 instance of the rule will fire and they may be of course many rules in the system and each will have some instances, it should ready to fire essentially.

So, the task of match this first part of the inference engine is to compute this set of instances of rules and the corresponding matching data and put it into this set which they call is a complex set. So, this is rule. So, let us say I call this rule 1 and this is rule 2 then, my conflict set will have something like.

(Refer Slide Time: 08:53)



Rule 1 matches with first, second and third working memory element. first, second, third. Let us say this is rule 1 another entry into my working memory element into my conflicts at would be rule 2 matches with 1, 2 and let us say 4. Another one would be rule sorry the rule 1, rule 1 matches with 1, 2 and 5. Let us say where 3, 4 and 5 are those these element.

So, this is the third one, the fourth one, and the fifth one. This is saying that this is an instance of rule 1 which is. So, these are the times stamps we are using from that working memory. It is matching working memory number 1 and 2 and 3. This one is matching the same rule in matching 1 and 2 and 4. This one is matching 1 and 2 and 5. And likewise for rule 2 and rule 3 and whatever rules we have this set is called the

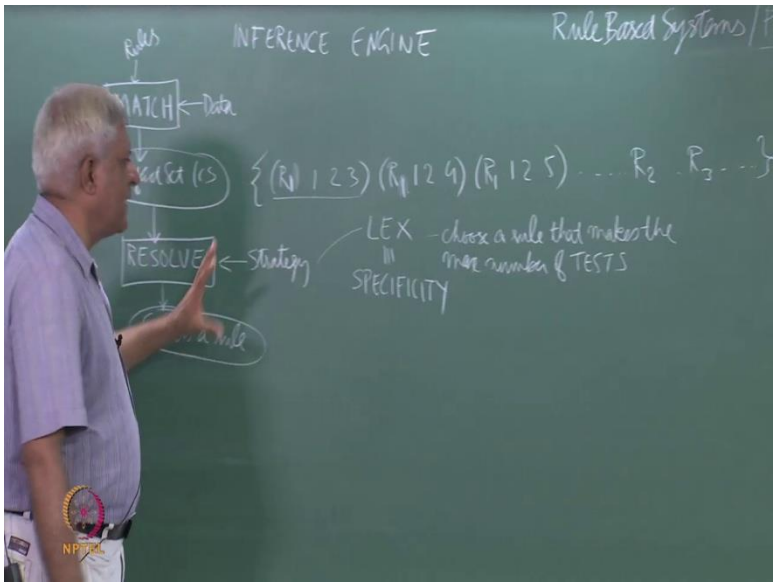
conflict set and the objective of the first phase of the algorithm which is the match phase is to compute this conflict set. Why do you call it conflict set?

It is like each rules clambering to say I will, I will execute, I will execute and so on. So, there is a conflict between all these rules. But you can execute only one rule at a time assuming that you are talking of sequential system here. So, people talk of parallel systems in which parallel rule firing takes place, but you will not get into that here. What is the complexity of these task? How difficult is it to compute this conflict set in terms of how many operations, how many comparison you have to do?

So, the brute force algorithm would take the first pattern which is P 1 1 which means the first pattern of the first rule and try matching it again solve these things. Then, it will take the second pattern of the first rule and try matching it with all these things, then the third pattern of the first rule try matching it with all of this then it will go to the second rule. I mean here it you do not have to distinguish between rules. We have just a set of patterns. So, each pattern in this it will try to match with each of this and at the end of it, it will compute the conflicts set let us keep this in mind.

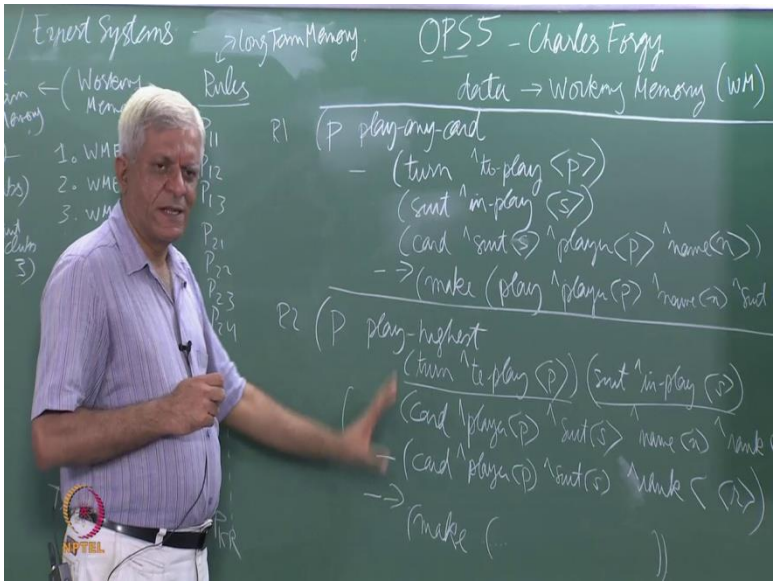
So, this conflicts set is a set of rules which is ready to execute. So, we use a term execute or fire. So, rule is ready to fire essentially. We keep this conflict set to step which is called resolve and it work it does is it select a rule. So, there is another program, another module to the program which looks at this conflict set and picks one of them to fire essentially. Now obviously, what this resolve needs is strategy. What is your problem solving strategy given that many rules of ready to execute which rule should be select to execute actually essentially.

(Refer Slide Time: 12:29)



Now, there are different kinds of strategies. So, one strategy which is called lax in ((Refer Time: 12:33) terminologies or lexicon graphic strategy. It says that choose a rule that matches or that makes a maximum number of test and what do I mean by test. By test I mean each individual these things.

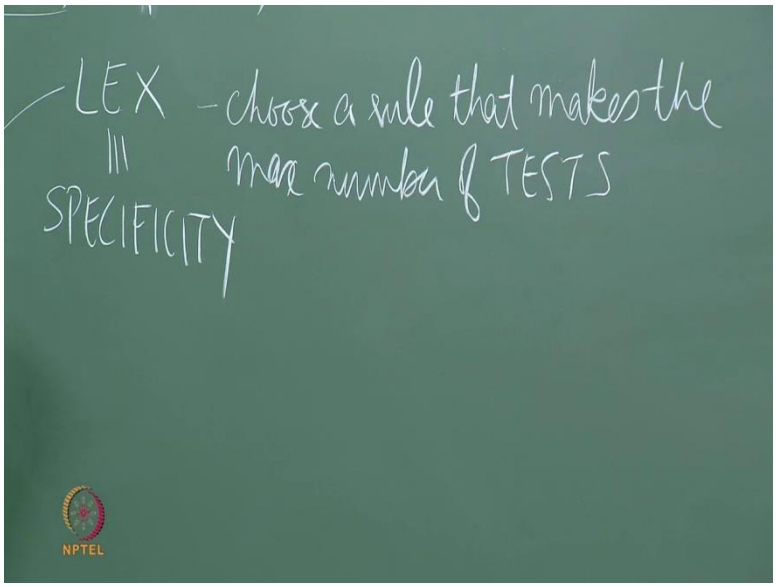
(Refer Slide Time: 13:02)



So, for example, I will match the class name with the working memory element that is one tests. Then, I will match the value of this attribute to play which the working memory element that is the second test. Third, fourth, fifth, sixth, seventh, eighth. So,

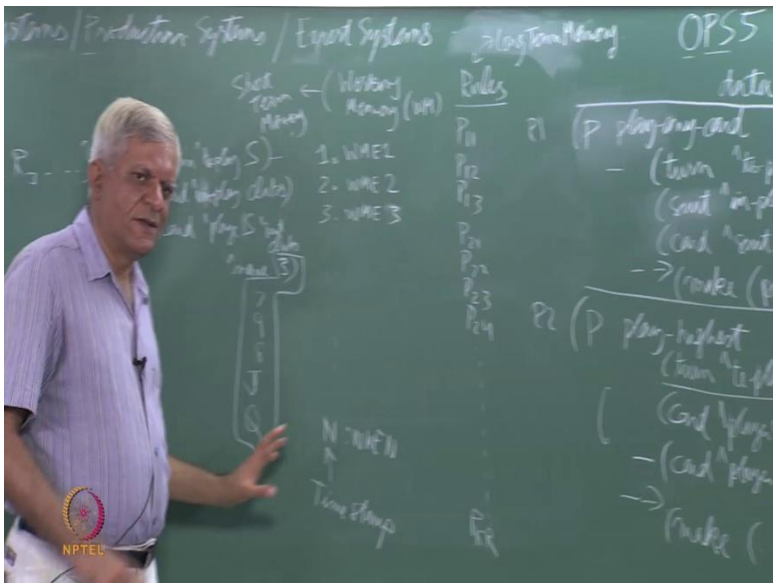
this rule has eight tests that it is doing, this rules has a little bit more tests than it is doing. So, the lax strategy says that choose a rule that makes the maximum number of tests essentially. So, what is the strategy essentially doing?

(Refer Slide Time: 13:37)



It is saying specificity. In other words, it is saying choose the most specific rule which matches. Now, if you look at this two examples that we have.

(Refer Slide Time: 13:52)



In these two rules, one rule is saying that, if you have to play a card of suit S and if you have a card of suite S play that. This rule is saying, if you have to play a card of suite S

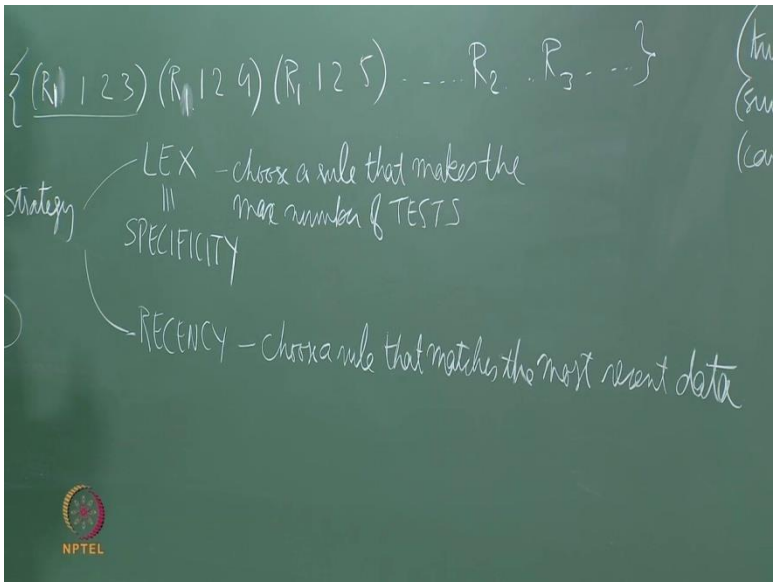
and if you have many cards of suite S, then pick the one which is the highest or highest rank card and play that card. Now, obviously given any data that I have so, for example, I said that we have these 6 cards. So, this will have rank 3, this will have rank 4, and so on and so forth. Both rules are matching, both rules 6 instances of this rule will match and 1 instance of this rule will match.

So, there are 7 rules which are trying to tell me which card to play essentially. So, 6 rules are saying any of these cards and the seventh rule is specifically saying this card queen, which is of highest rank why because we have said that the rank of this card is r then there is no card held by this player whose rank is higher or higher than r which means the the number is lower than r. So, this rule is more specific than this rule. So, this conflicts resolution strategy at as it is called which is specificity will choose the second rule essentially or if the second rule matches little.

So, you can see that this strategy allows you to implement what we sometimes call as default reasoning which is you like many rules essentially. So, it is like saying one rule says if you are hungry go to the mess and eat essentially. The other rule says if you are hungry and you have lots of money and you do not have any exam tomorrow then go out to a restaurant an eat essentially. Now, the first rule is less specific than the second rule.

The second rule requires many conditions to be true that you should have money you should be free and that kind of stuff. If both rule matches than the second rule would be selected, if only if the second rule does not match that means you do not have money, or you have an exam tomorrow then that will not match and then only the first rule will match and that will execute. So, you can see that the more specific the rule the greater we would like it to reselected and specificity say exactly that choose a rule, which is making more number of tests out of the competing set of rules essentially.

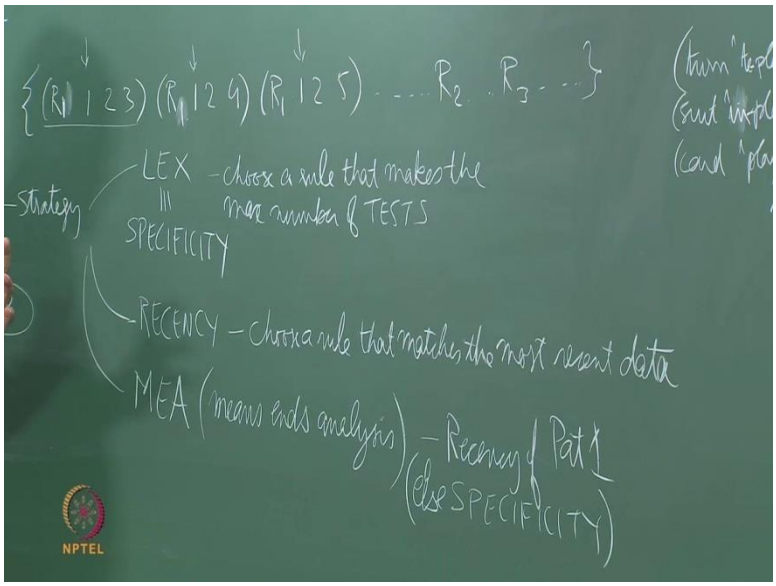
(Refer Slide Time: 16:48)



Another strategy is looking at the time stamp which is called recency. It says choose a rule that matches a most recent data and what do you mean by most recent data? Is basically this time stamp that we have said which has the highest time stamping. So, if every rule will match certain number of elements remember for example, this is my thing 1, 2 and 3, this is my thing 1, 2 and 4 and so on and so forth.

Whichever rule is matching the latest data choose that essentially. What is the intention behind this strategy? It is to kind of maintain of flow of reasoning essentially. So, just imagine you are doing theorem proving, you are proving something, you have proved some lambda 1 then, you want to use lambda 1 to prove lambda 2 and this rule will allow you to do that because lambda 1 which is whatever that lambda is could be the latest entry into a database and if a rule is matching that, that will automatic get selected. So, it sort of helps in maintain chain of reasoning.

(Refer Slide Time: 18:10)



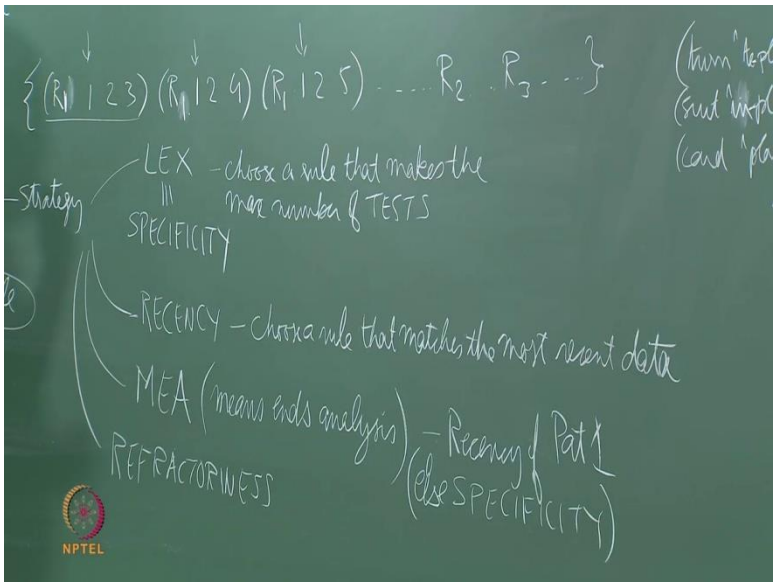
There is a first strategy which is called mea, which stands for may be at some point we will a discuss this, means ends which is due to a Simon and Noel and it essentially combines these two by saying recency of pattern one and if there is still a conflict then specificity. So, I should write else. So, by this I mean that you are only looking at the recency of the pattern one and pattern one is the first pattern in a every rule essentially. So, this is the first pattern, this is the first pattern, this is the first pattern and so on.

So, the first element which follows after the rule, look at the recency of that and as you can see all these are same, but some of the rule may have higher recency and use that essentially. If there is still conflict that is more than one rule which is in contention then use specificity to choose between them essentially. So, that is why I have written else specificity. So, what is the intention behind this? Essentially, you can partition your rule sets into, set into groups which will solve particular problems essentially. So, let us say you are making dinner essentially. Then, you might have a set of rules let say to how to make sambar and another set of rules how to make subjee and so on and so forth.

Now, if you in each of this rules, if you have the first element which says something like making sambar or making subjee or making chapati or making rice, then the moment you create a context for saying, okay now I am making rice then you will add that data element saying making rice that will become the most recent and all the rules which are concern with making rice they will get priority essentially.

And that is done by the fact that recency of the first pattern. The first pattern setting the context, making rice, making tea or whatever essentially. The moment you said this is my task I am doing only those rules which have that as the first pattern will be in contention the rest will not come into play. So, you can see it is a combination of these two things. It helps you keep focus to what you are doing, but it also tries to see which rules are best and so on.

(Refer Slide Time: 21:10)

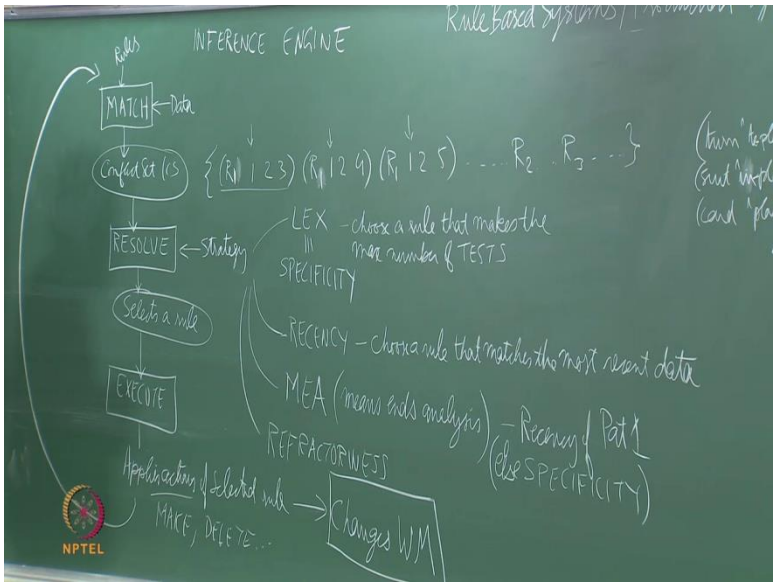


Now, one more thing is refractoriness, this simply says that a same rule cannot fire with the same data again essentially. So, this comes from some neurobiological studies which says that, you know when neurons are firing then once a neuron gets certain set of input they can only fire once with that input and then there is a period of refractoriness in which they do not, they are not active at all essentially. But from our point of view, we can see that we do not want the same rule to fire again and again and again essentially.

So, let us say you are doing a classification task or let us say now the task is to classify students into grades essentially. So, you have said if a student has got more than 95 and he has done all the assignments then give an a grade. And this let us say, this rule get selected and it fires. So, you have already done with the task of giving the a grade or classifying the student in as a student. You do not want that rule to fire again essentially with the same piece of data.

And that is what refractoriness says that every rule can only fire once with that piece of data. Which means of course, with the same time stamps in the working memory if you were to delete some data and add the same data again with a new time stamp then it could fire, that is a different story.

(Refer Slide Time: 22:54)



And then so, its selects a rule and gives it to a module call execute and what is execute does is to applies the actions of selected rule. So, remember that this resolve module resolves the conflicts between all these rules and it says this is the rule that is going to execute and this execute module basically takes a rule and takes its right hand side which is the actions and in and sort of execute to those actions.

So, remember what was, what was the actions thinks like make or delete and so on essentially. So, what is this doing? It changes the working memory. The effect of execute is to change the working memory. It may add some new data and it will deletes some old data from the working memory.

So, now you have a new working memory and now you have to go back all the way and do the match all over again. So, this is the algorithm which the, this in some sense a high level algorithm. Actually, you do implement it like this. We will see how it is done, but you can think of it as doing like this. That first is a match phase, which looks at all the rules and all the data and by that mean everything in the working memory and creates

this set or this list of rules and the corresponding times stamps of the data elements they are matching.

Then resolve looks at this set and picks one element out of that and says this is the rule that we will execute and then when execution happens, the right hand of the side of the rule may make some changes in the working memory. So, some elements are deleted, some may be added and then we go back to matching again and selecting a rule and so. So, we keep doing this till some termination criteria which could be that either the conflict sets becomes empty at some point, which means no rules are matching all we have an explicit halt statement somewhere in the program which says if you see a certain pattern that then you halt and execute.

So, let us now discuss the complexity of this whole process. What can we do to improve this essentially? Now, empirically people have found which is the hardest part of this, in terms of complexity, which one will take the most amount of computing time match or resolve or execute. which one will take the most amount of time? Look at the complexity of each of the task, look at these tasks, execute what is it doing? So, in this example the right hand side has only one action which is to add one working memory element with this particular this one. So, you can see it is a little bit of work.

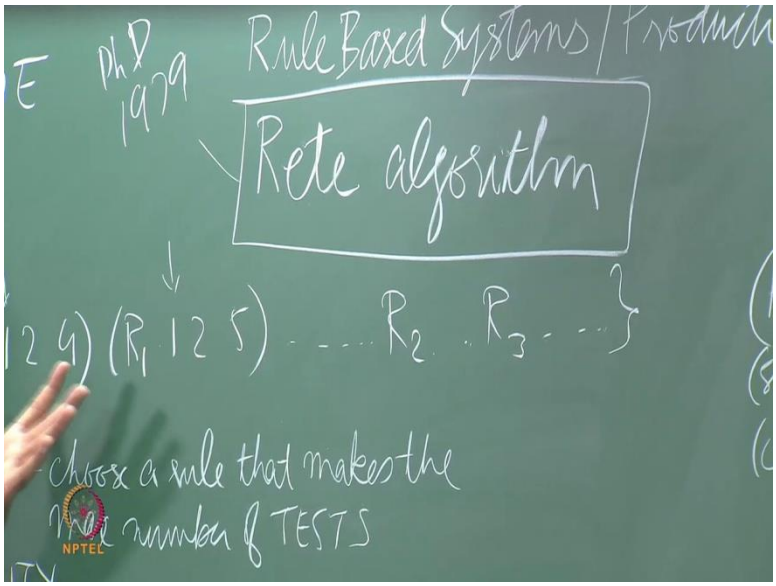
What does resolve have to do? Resolve has to look at this entire set of matching rules of the conflict set and applied a chosen strategy whichever the strategy we have chosen, to select one of these from this and what does match have to do? Match has to look at the entire set of rules or the entire set of patterns we have compare it with the entire working memory and try to find out what is the conflict set it is. So obviously, this is a largest task, the number of comparison we have to do if you have k rules and each rule as let us say p patterns. So, k into p patterns totally and each pattern may have a certain number of tests essentially and then, all these working memory elements. So, with each you have to find.

Remember that you want to find every rule that can possibly execute or every rule that matches. It is not that you are if you find one or two rules match you are done. All the rules with match, the complete set must be, this conflict set must be exhaustive. It must consist all possible rules which are matching. So, this is really the most time consuming part of the algorithm and people have observed that match takes up something like 80

percent of computation time. So, just like you know they say in an industry that most of the time of computing is spent in sorting for example, most of the time in power changing inference engine is spent in matching essentially.

You could have other strategies which I have not mentioned here. If you know how prolog works. Prolog will take the first rule with matches. So, it sort of goes down the program and the first rule which matches it applies essentially. It does not compute all possible matches and then selects one. So, that is a different strategy that is the, something like order of rules or something like that. Here we are considering rules to be floating in some space and there is no order and any rule is as good as any other rule. So, what Charles Forgy. So, if you look up some of these terms either ((Refer Time: 29:17)) Charles Forgy or this algorithm which he developed which is called the rete algorithm.

(Refer Slide Time: 29:23)



Which is what you want to study, but I think you do not have enough time today so, we will do it in the next class. Charles Forgy device the rate algorithm in 1979, his phd theses at a you and subsequently it became a commercial product. It is available in any these business rule management engine you talk about, it uses some variation of rate for matching rules because it is really the hardest part of doing that essentially. So, this is off course available to us that we have a paper written by Charles Forgy and you can find it on the net which talks about this is rete algorithm.

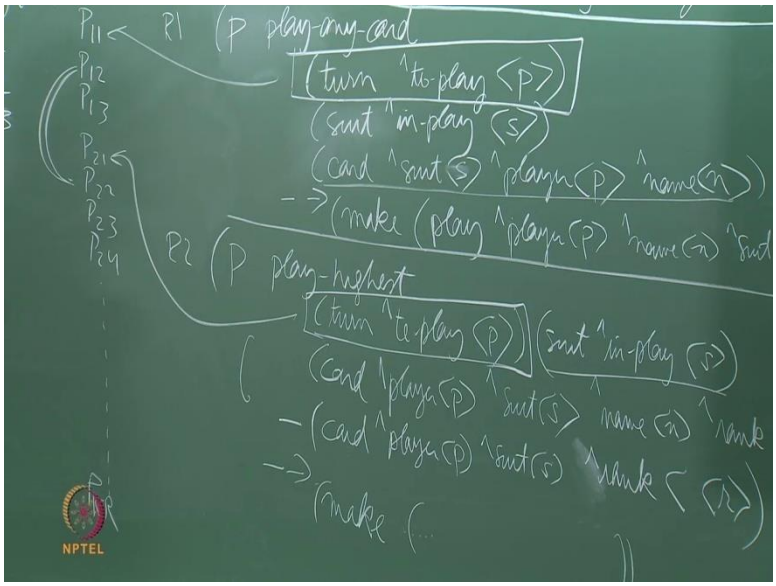
And then he went on to develop algorithm called rete 2 and from there onwards they stop disclosing their algorithm to the public because they have commercial interest. They had this company which was selling that algorithm and did not want to diverse this create secret that this say.

(Refer Slide Time: 30:38)

And eventually he wrote on algorithm call rete NT. For those of you who are familiar with windows will see the influence of windows, so just around that time windows NT came, it was a new thing, and so, this rete NT was new thing. This rete NT is suppose to be 500 times faster than these original rete algorithm, which means it takes 500 times less time to the same match essentially. And you can see that the great improvement instead of having to wait of few minutes, you might have to just wait a second for your match to be done essentially. Unfortunately we do not know the algorithm, we will only look at the rete algorithm.

Rete itself, the word rete is a latin word which means the net essentially. And what this rete algorithm does is to, it compiles the rules into a network which we will see in the next class to improve upon the efficiency of match. So, what we want to do today now is to observe where does the inefficiency come from in matching these things. There are two sources of inefficiency and can you think about then. See one is the following, that we have many rules and each rule has some number of patterns and those patterns have to be match with data. Now, if you look at that two rules that I have written.

(Refer Slide Time: 32:18)



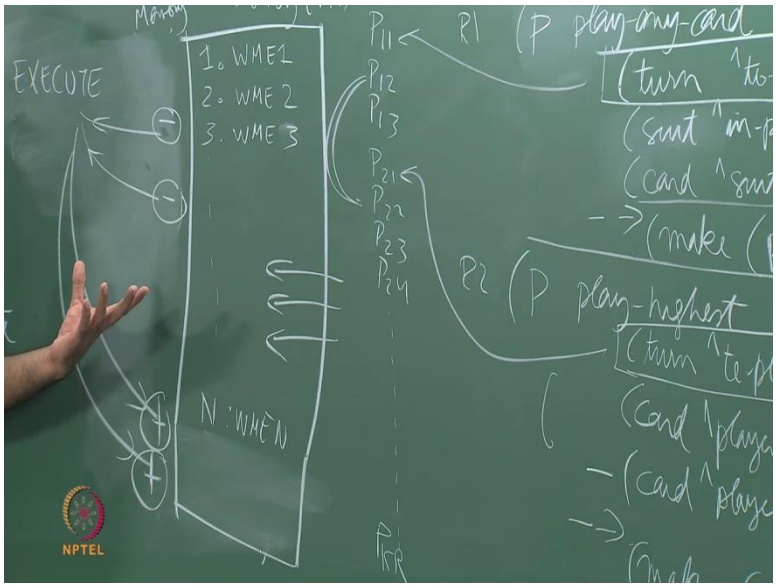
We have these pattern, we have these pattern turn to play p. So, it was match a working memory element whose class name is turn and whose to play attribute must match something, there must be something in the these thing. It could be south, north, east, west to use the terminology from which. This rule also has a same pattern so, this one could be here for example, and this one could be here. Likewise as you can see the second pattern is also the same suit in play S, suit in play S. So, this and this are the same.

Why should we spend computation time doing this match separately? Why not once and for all save as the this is match, then we can tell this rule there is matching also tell this rule that is pattern is matching. So, if you can reduce that there will say sometime essentially. Not only that in this third pattern certain things are common. So, card, suit, play these things. These also has card, player, suit, name. It has a additional thing called rank, but at least for those 3 tests or 4 tests, we can share those 4 tests.

So, the first thing that we would like to do is to share the tests the different rules are doing and different patterns are doing. So, that as far as possible each test we make only ones or each pattern will match only ones. So, if you match this pattern ones and then both the rules come to know about it, then we are saving time essentially. So, that is one. So, this is one source of inefficiency or one avenue for increasing the efficiency of your match algorithm.

But there is another which will occur to you if you look at what this execute is doing. What is this execute doing? It is making some new working memory elements or it is deleting of you working memory elements. So, in a large working memory of this kind.

(Refer Slide Time: 34:43)



Let me rub this out. So, what execute my do? It might send a signal that this is to be deleted, so I will use minus sign for deletion here. It might say this to be deleted and it might say new one to be added. So, typically a rule when it executes it will delete may be a couple of element from the working memory, at may add a couple of elements to the working memory.

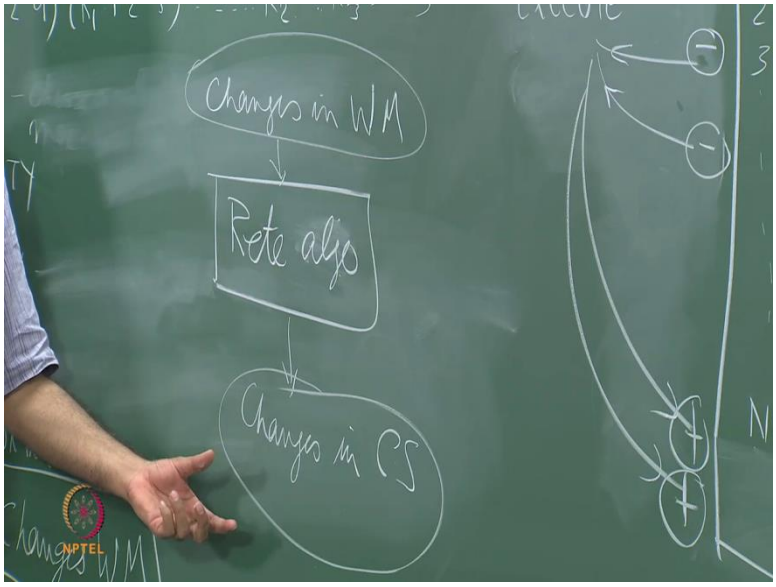
Now, what is it doing? After it executes its goes back to the match and at least if you look at it from the brute force point of view, it is going to match all the rules with all the data all over again. Does it need to do that? Now, if you look at this first rule here, which says play any card, it is adding one working memory element essentially. And let us say I had a 100 rules for playing cards and all of them I have done the match in the previous cycle and all of the above matching that say 2, 3 instances. So I have 3 or 400 instances of rules which I match in the last cycle. What I have done when I executed this rule.

Let us say this was executive for some reason or this one is executive for some reason, yeah I have added 1 working memory element to my these thing. So, why should I match those other 99 rules again. In this particular example, I am only adding 1 working memory element and assuming that those 99 rules do not have negative clauses. Even if

they have why should I do that match all over again essentially. Rule number 2, rule number 3, rule number 7 they were matching with some data which is nothing to do with this 4 data elements that we are talking about.

May be some, may be the some rule was matching a data here and data here and data here. If it was matching in the last cycle, it will match in the, this cycle as well. So, essentially what the rete net does. So, what is the match doing? Match is taking rules and data which is working memory and producing the conflicts set.

(Refer Slide Time: 37:49)



The rete net is an algorithm or the rete algorithm is a algorithm which uses a structure call the rate net or let me just a use algorithm here, takes as input changes in working memory which is what this execute action is doing. This change is in working memory, its changing, is deleting a few elements and adding a few elements. This algorithm rete algorithm takes changes in working memory and gives you changes in the conflicts set.

So, compare this with this thing this match here. What match is doing is taking the full working memory, taking all the rules and producing the complex set and it is doing this every time we are going through this cycle. Every time a rule fires it go the does match all over again or the area algorithm does is says tell me what are the changes in the working memory element and it computes that as a result what are the changes in the complex set. So obviously, as you can imagine this is must less work as compare to

doing the full match all over again essentially. We only have to see the effect of these changes into the conflict set.

May be, if I have because of adding this may be because of deleting this some rule which was firing matching earlier will not match now. So, I should able to know that essentially. Likewise if I have add this some new rules may come into the conflict set and may be some rules might even go out of the conflict set, if there was a negative clause here. I should be able to capture that essentially.

So, in the next class we will look at this rete net, which is basically a sort of a discrimination net or you can some people are use other terms like many sorted network or it is a generalization of a trice structure, but basically it is a network which discriminates between different kinds of data essentially. You might even think of it as a extension of binary search tree, which this discriminates between only one kind of data which is numbers. So, its sends you down one branch or the other.

So, same principle holds except that this is of multifaceted data and we have different kind of test which are being. So, we even have a test which says that the rank is less than r or rank not equal to r or something like, things like that essentially. So, all those tests should be will together. So, will take this up in the next class which is on Friday and complete this part of the this unit.