

So the reasoning part is concerned with algorithms and we essentially want to focus on what is the algorithm. Now what is the reasoning algorithm. We are given some knowledge base. We will use the term KB as usual for knowledge base and a goal or we can call it a query α . The question we are asking is does the knowledge base entail α and then we are translating this question to a different question. So now we have gone past the semantics to does the knowledge base derive α .

(Refer Slide Time: 03:50)

Forward Chaining - Modified Modus Ponens
uses Verification

Knowledge Representation - ad hoc predicates

Reasoning - algorithms - Given a KB
and a goal/query α
Does $KB \models \alpha$
 $KB \models \alpha$

NPTEL

5/5

So if you try to imagine what is the situation right here pictorially you might say this is the knowledge base. And it contains many statements. So as far as the abstract representation or the logical representation is concerned we are not making any statements about how these things are stored, how they are retrieved and any such things.

(Refer Slide Time: 04:13)

Feb11-2015-kr - Windows Journal

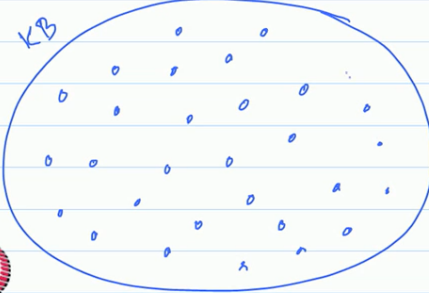
File Edit View Insert Actions Tools Help

Page Width

Forward Chaining - Modified Modus Ponens
↳ uses Verification.

Knowledge Representation - ad hoc predicates

Reasoning - algorithms - Given a KB
and a goal/query α
Does $KB \models \alpha$
 $KB \not\models \alpha$



NPTEL

5/5

So we just have a collection or set of statements and then we have a collection of rules. In our case we have one rule which is modus ponens, modified modus ponens but we could have more rules it doesn't matter and we are given a query α and we are asked whether this α is true or not essentially and we want to generate a proof essentially. What is a proof? A proof is a sequence of conclusions, at the end of it, the last conclusion is α essentially. So that's the direct proof or the forward chaining method that we are looking at essentially. So somehow from this sea of formulae we have to pick the right rule, apply it to right facts and generate some new things essentially. So for example we may pick this and we may pick this and we may add another thing to the knowledge base, then we may pick this and then we may pick this and we add another thing to the knowledge base and so on. We keep adding more things the knowledge base expands and at some point hopefully we will add this α essentially.

(Refer Slide Time: 05:26)

Feb11-2015-krr - Windows Journal

File Edit View Insert Actions Tools Help

Page Width

Forward Chaining - Modified Modus Ponens
↳ uses Verification

Knowledge Representation - ad hoc predicates

Reasoning - algorithms - Given a KB
and a goal/query α
Does $KB \models \alpha$
 $KB \not\models \alpha$

NPTEL

5/5

Now if you look at the two student communities, the logic community or the mathematics community, so a logician will simply say a proof exists. Or they might even produce a proof essentially. So they don't tell you have they arrived at the proof. They will say here is the proof, so your task is now to simply validate or verify that the proof is correct or wrong. So you check whether every inference is sound and if that's the case then you will accept the proof essentially. But as a computer scientist which I presume all of us are, our task is to find the proof.

(Refer Slide Time: 06:28)

Feb11-2015-krr - Windows Journal

File Edit View Insert Actions Tools Help

Page Width

Forward Chaining - Modified Modus Ponens
↳ uses Verification

Knowledge Representation - ad hoc predicates

Reasoning - algorithms - Given a KB
and a goal/query α
Does $KB \models \alpha$
 $KB \not\models \alpha$

NPTEL

5/5

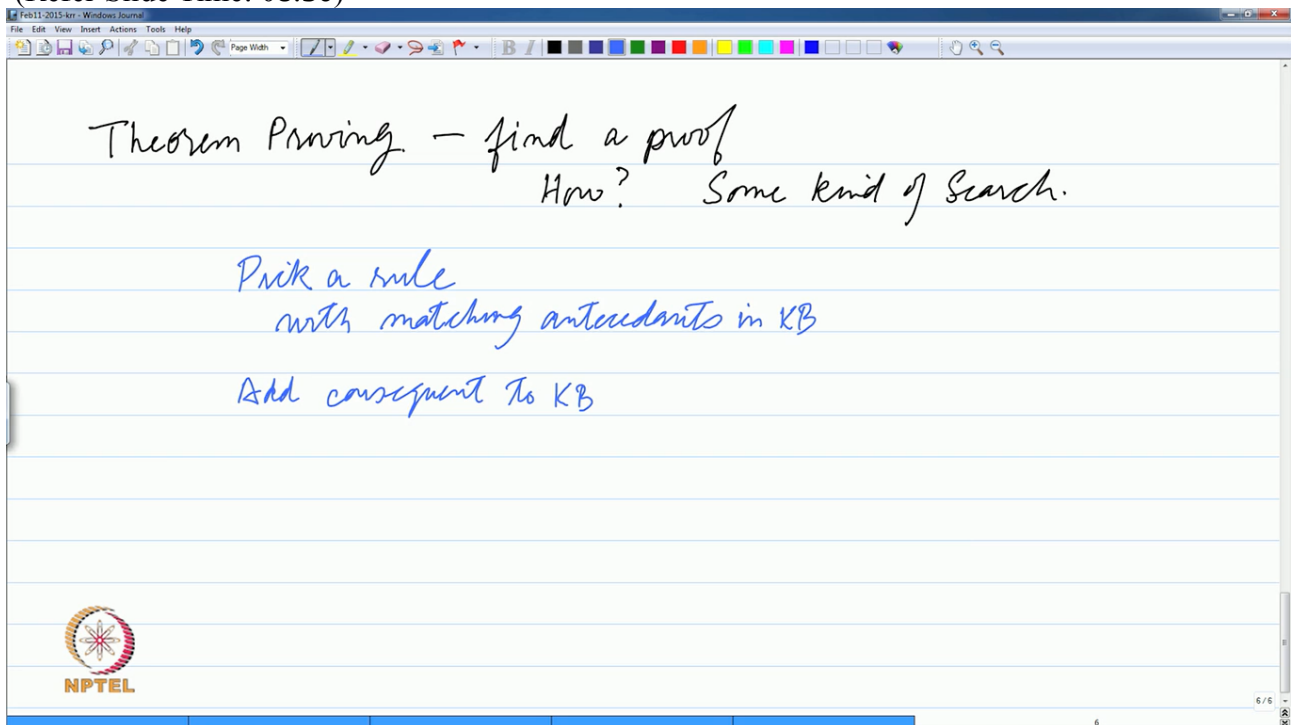
Logician - "A proof exists"
Computer Scientist → task "FIND" the proof

The logician or mathematician does not give you the process by which he or she arrived at the proof. Whereas when we are writing programs to write the proofs then that becomes our primary goal essentially. How do we find the proof essentially? So this exercise I might have mentioned is called theorem proving. So we use the term theorem to stand for any proof statements essentially because that's what mathematics says that once you prove something it becomes a theorem or lemma. And

the task of theorem proving is to find an algorithm which will find a proof essentially. It may be more than one so you can at least find one. So the question of course is how. And the answer to that is some kind of search.

So at a high level we have already said that the algorithm is as follows. Pick a rule with matching data, antecedent is left hand side of the rule, so if something, that something is the antecedent. So if you can find the rule which has those parts present in the knowledge base, then we add consequent to knowledge base. So antecedents are in the knowledge base.

(Refer Slide Time: 08:30)



The screenshot shows a Windows Journal window with the following handwritten text in blue ink:

Theorem Proving - find a proof
How? Some kind of search.

Pick a rule
with matching antecedents in KB

Add consequent to KB

The NPTEL logo is visible in the bottom left corner of the journal page.

So in this way we keep adding more and more statements to our knowledge base and reach, we will terminate when we have added the statement we are interested in, the thing we call alpha. So the question is what kind of search can one employ so what do we employ. We have a set of rules in which we have one or more rules and we tend to distinguish between two kinds of things, one which we say are facts and other are rules. This is just a distinction that that we use from the point of view of revising algorithms, both are statements in logic but you distinguish between facts and rules.

(Refer Slide Time: 09:33)

Feb11_2015-kr - Windows Journal


File Edit View Insert Actions Tools Help

Page Width

Theorem Proving - find a proof
How? Some kind of Search.

Pick a rule with matching antecedents in KB
Add consequent to KB

FOL statements
FACTS RULES



6/6

So fact for example can be Man Socrates and the rule can be like for all x Man x implies Mortal. So i will just use m here. So we distinguish between these two kinds. So essentially what modus ponens which is one inference rule which are using has access to a large collection of rules like things all men are mortal, all student are bright or all bright students are girls and things like that. So whatever statements we have we will call them rules and whatever facts we have is the actual data in the knowledge base. Socrates is a man, Shrestha is a student, Shreshta is a girl, so such kinds of statement is a fact. So modus ponens is trying to pick a rule essentially. So when we say rule here, it's a statement, that's a statement that I mean here.

(Refer Slide Time: 10:34)

Feb11_2015-kr - Windows Journal

File Edit View Insert Actions Tools Help


Page Width

Theorem Proving - find a proof
How? Some kind of Search.

Pick a rule with matching antecedents in KB
Add consequent to KB

FOL statements
FACTS RULES

Man(Socrates) $\forall x [Man(x) \supset Mortal(x)]$



6/6

These are also statements in first order logic, there is a slight confusion between rule of inference and rule here. Rule of inference is modus ponens or modified modus ponens, it's a metalevel thing.

Rules in a language is basically universally quantified statement like this statement, all x are y, things like that essentially. So the algorithm now needs to wait through all the rules and all the facts, to find something which is a matching piece of rule. A statement which is a rule and a matching statement which is a fact essentially. Now you can see that if you have a large knowledge base, then the choices that you have will be in proportion to the number of facts we have and the number of rules that we have. So it's a problem which is going to increase combinatorially in terms of complexity so that's one of the main issues that we want to attend.

(Refer Slide Time: 11:43)

Theorem Proving - find a proof
 How? Some kind of Search.

$\xrightarrow{\text{FIND}}$ Pick a rule with matching antecedents in KB
 Add consequent to KB

FOL statements
 FACTS: $\text{Man}(\text{Socrates})$
 RULES: $\forall x [\text{Man}(x) \supset \text{Mortal}(x)]$

NPTEL

so what are the kind of search algorithms that we can think of. One is, of course the simplest search is linear search or sequential search. This simply says, take the first rule, apply it to the first set of data, try to match it with the second piece of data and so on and so forth. So you take the rules one by one, then you match them with data one by one, so you keep doing that. We will come to this later essentially. This kind of an approach is used in a language Prolog.

(Refer Slide Time: 12:50)


Feb11-2015-krr - Windows Journal

Theorem Proving - find a proof
How? Some kind of Search.

Pick a rule ^{FIND} with matching antecedents in KB
Add consequent to KB

FOL statements
FACTS: Man(Socrates)
RULES: $\forall x [Man(x) \supset Mortal(x)]$

Simplest - Linear or Sequential
↓
PROLOG




And then when we come this you will see that because we are using an algorithm which is sequential search, the onus will be on the person who is writing the knowledge base or who is writing the rules, to put them in certain order such that the algorithm will find the solution quickly essentially. We will come to Prolog, which is actually a process of backward chaining as opposed to forward chaining that we are looking at, and Prolog uses sequential search so will come to that a little bit later. So today we want to see how forward chaining can be made more efficient essentially. So we want to try to look at that aspect.

So what is forward chaining essentially is the process that we are saying. Look for a rule, look for matching data, add the consequent and then keep doing this. Now let's assume that rules are given to us in a particular format which is, so when I say rules, I mean rules which are also statements in a language, statements of kind

If antecedent1 antecedent2 antecedentN, let's say there are N conditions on the left hand side, then consequent.

(Refer Slide Time: 14:50)

Rules IF antecedent 1
antecedent 2
⋮
antecedent N
→ (consequent)



So we are given a collection of such rules, so when you look at all these rules we will have a set of patterns essentially. So the patterns that we are trying to match are the left hand side of the rules. We are trying to see whether there is some element in the knowledge base which matches antecedent1, some element which matches antecedent2 and so on and so forth. And we are given a set of rules so let us say that our knowledge base looks like this, a11 so a11 I will use as a notation to stand for rule 1 and antecedent1. But I may have other things, so I may have antecedent12 and so on, a1N..so these are the N antecedents that I am talking about. Then I will have a21 and so on a2R, let's say it has R antecedents and so on and so forth. So for every rule I will have a certain number of antecedents. So i will have a list of patterns that I am trying to match essentially. And what am I trying to match.


(Refer Slide Time: 16:30)

Rules IF antecedent 1
antecedent 2
⋮
antecedent N
→ (consequent)

Patterns

Rule 1
← Antecedent 1

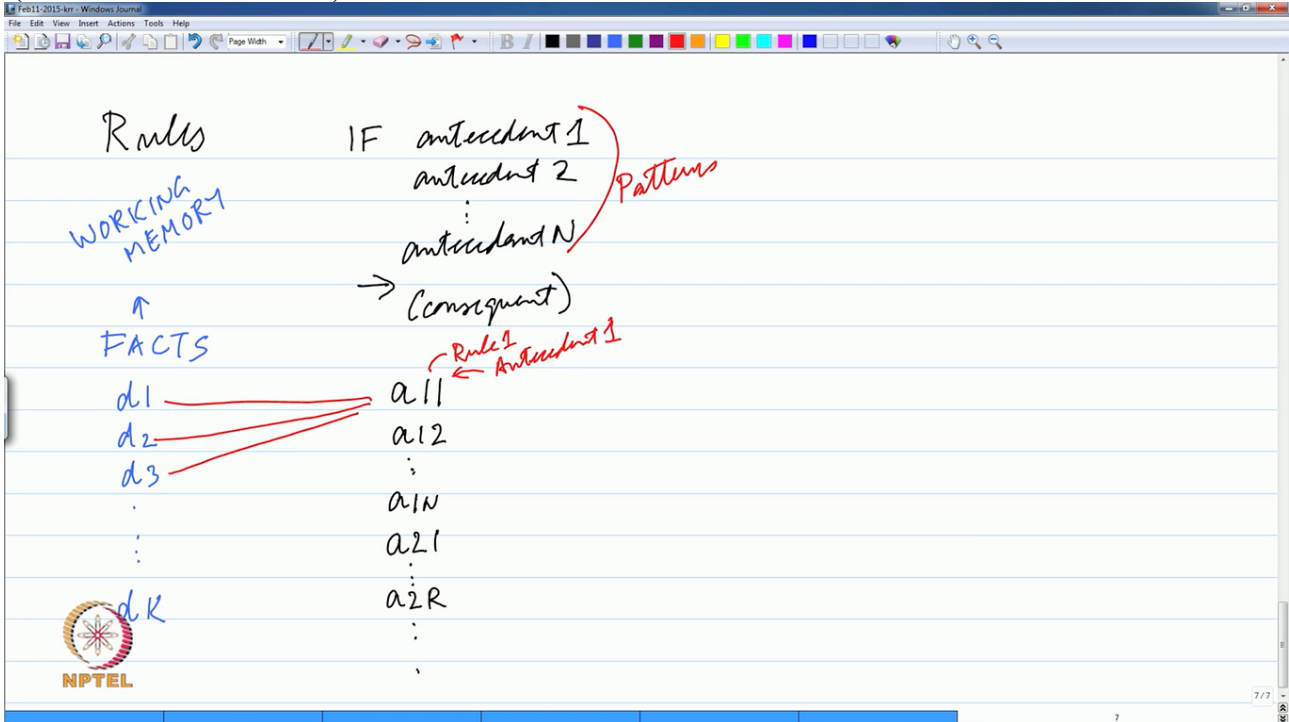
a11
a12
⋮
a1N
a21
⋮
a2R
⋮



So I am trying to match facts. So let's say simply facts are some statements, data point 1, data point

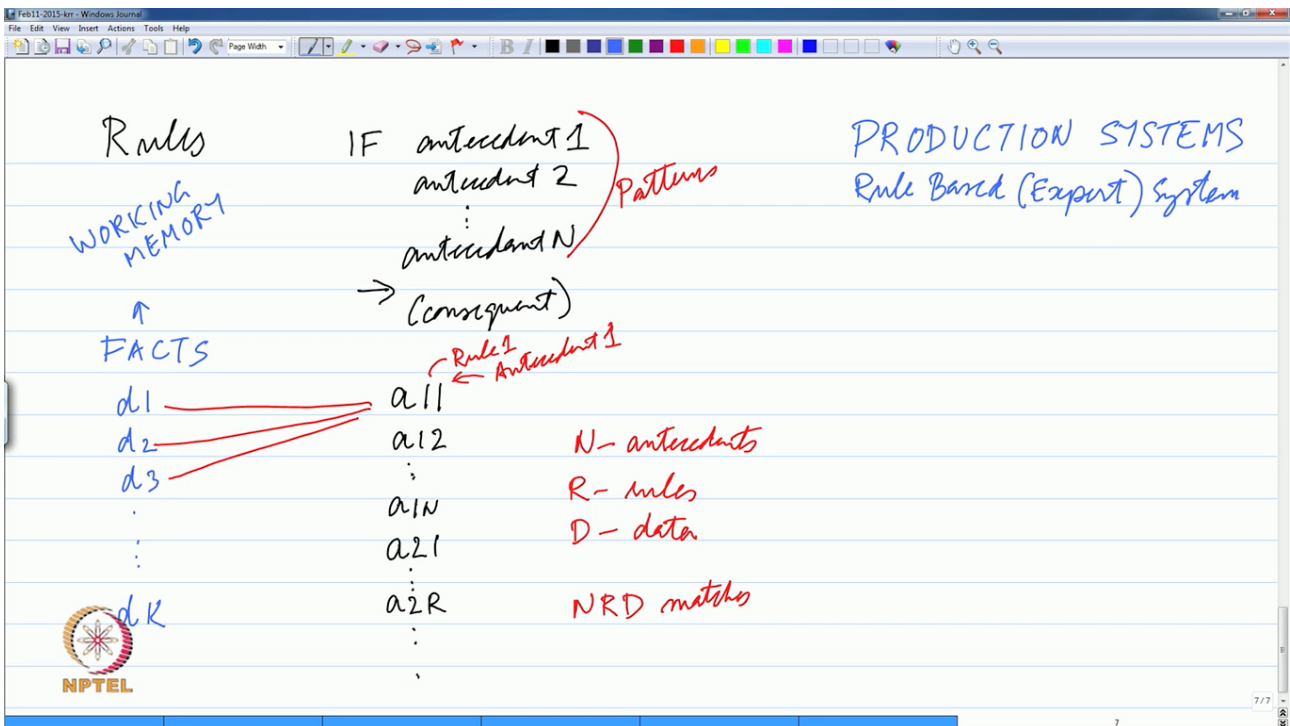
2, data point 3, some number of data points essentially. In the terminology that soon we will look at of forward chaining algorithms that you will see these facts are also called working memory elements. The facts are essentially called working memory and each of those data items is also called working memory elements. So what is the task. The task is you try to match a11 with d1, you try to match a11 with d2, a11 with d3 and so on so forth.

(Refer Slide Time: 17:49)



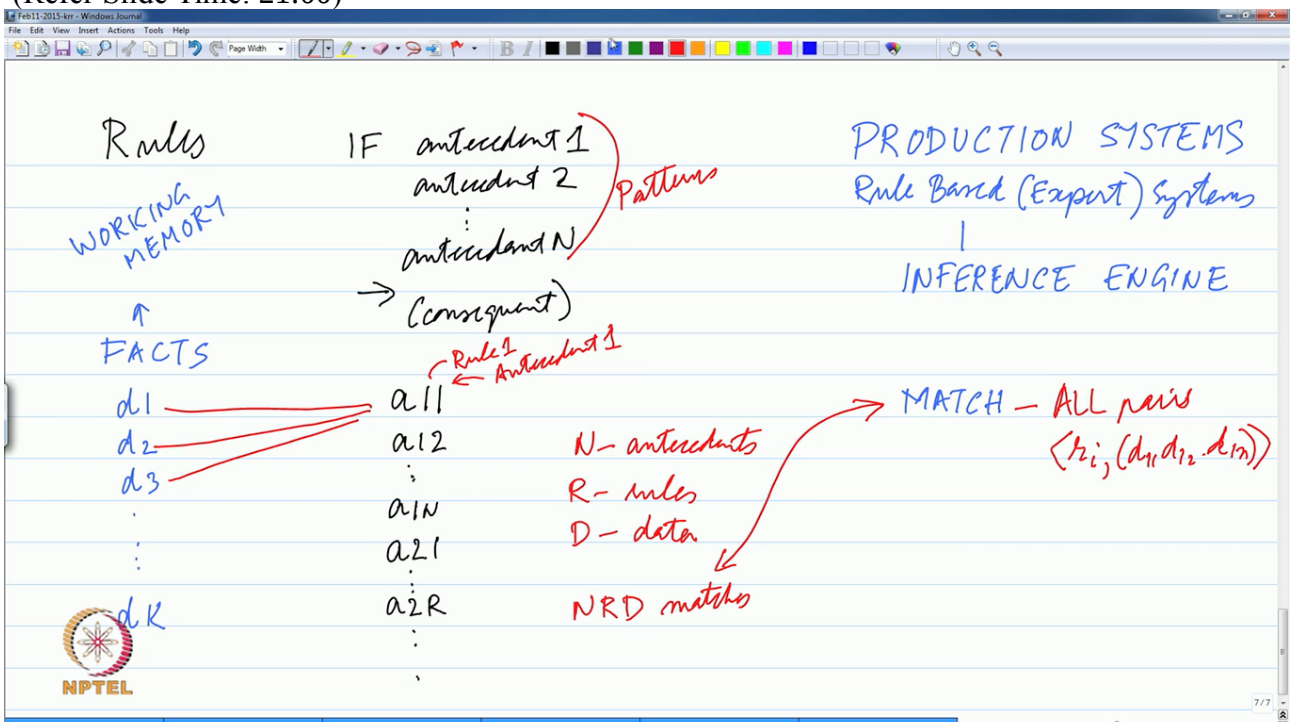
So you can say that the total number of matches that we will have is the if we have N antecedents on an average, let's say we have R rules and D data element, then we will end up matching every antecedent with every data element so N into R into D matches. So this scenario is a little bit different than what I started by saying. What I started by saying is pick a rule with matching data but now what I have moved to slightly different way of doing things which is the method that is used in what we call as production systems or rule based systems. In the 1980s they would be called expert systems.

(Refer Slide Time: 19:15)



And what a production system does is that it has something which is called an inference engine. This is a terminology with this particular community. Inference engine is simply an algorithm that we are talking about which says that add a matching rule, add the consequent and so on and so forth. But the difference is that the typical forward chaining inference engine is made up of three steps and the first step is Match and the thing about this is all pairs of the kind some rule r_i with matching data point so $d_{i1}, d_{i2}, \dots, d_{in}$ let us say. Find all such pairs let me emphasise it. So the match algorithm essentially says that given a set of facts and given a set of rules, both are the statements in first order logics, which rules are matching with which data. Produce the whole set for me and I will choose which one to select. So we are doing a pre-processing of the match essentially. And this match is what will do this NRD number of matches.

(Refer Slide Time: 21:00)

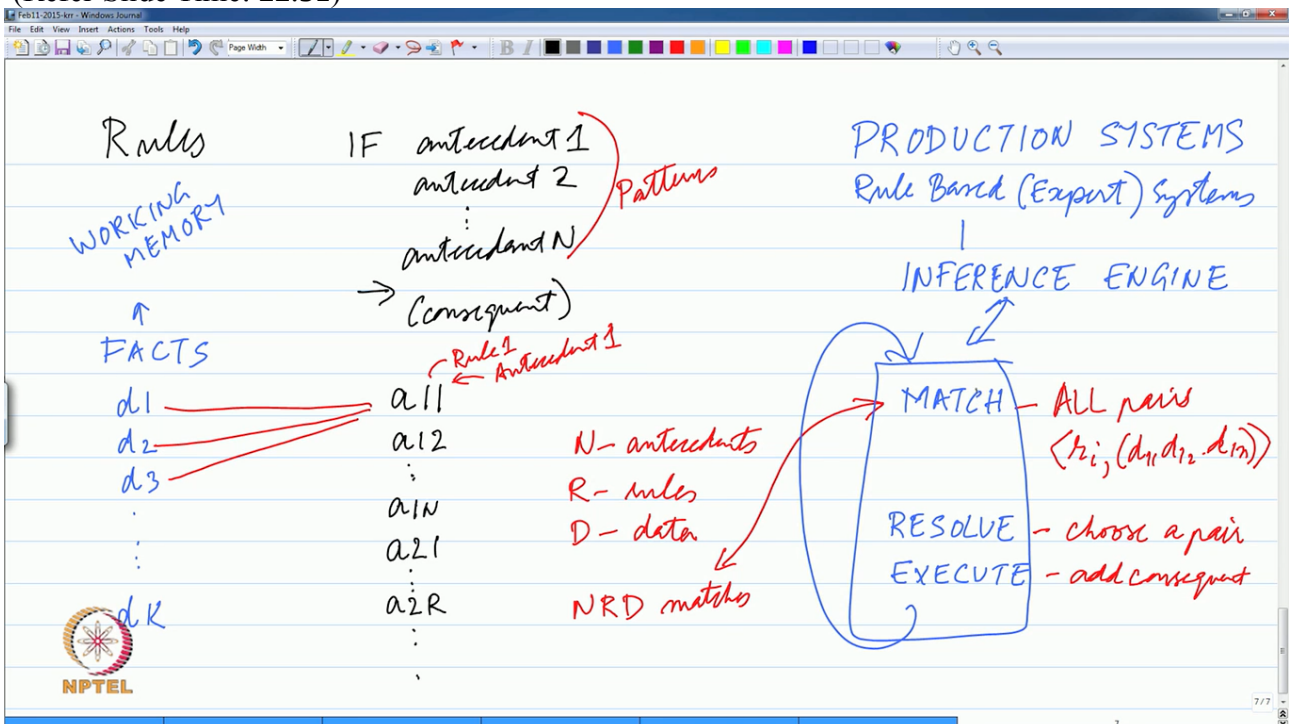


Now that we have done all the matches we have a step called RESOLVE which essentially says

choose a pair. Obviously it is very critical as to which one you are choosing. So this element of resolution. How is, what is the problem solving strategy that we are trying to use as to which rule you will apply. We will come to this a little bit later. So at the moment I want to focus more on how to make this process of inferences efficient and the algorithm that we are looking at in essentially designing to do the match more efficiently.

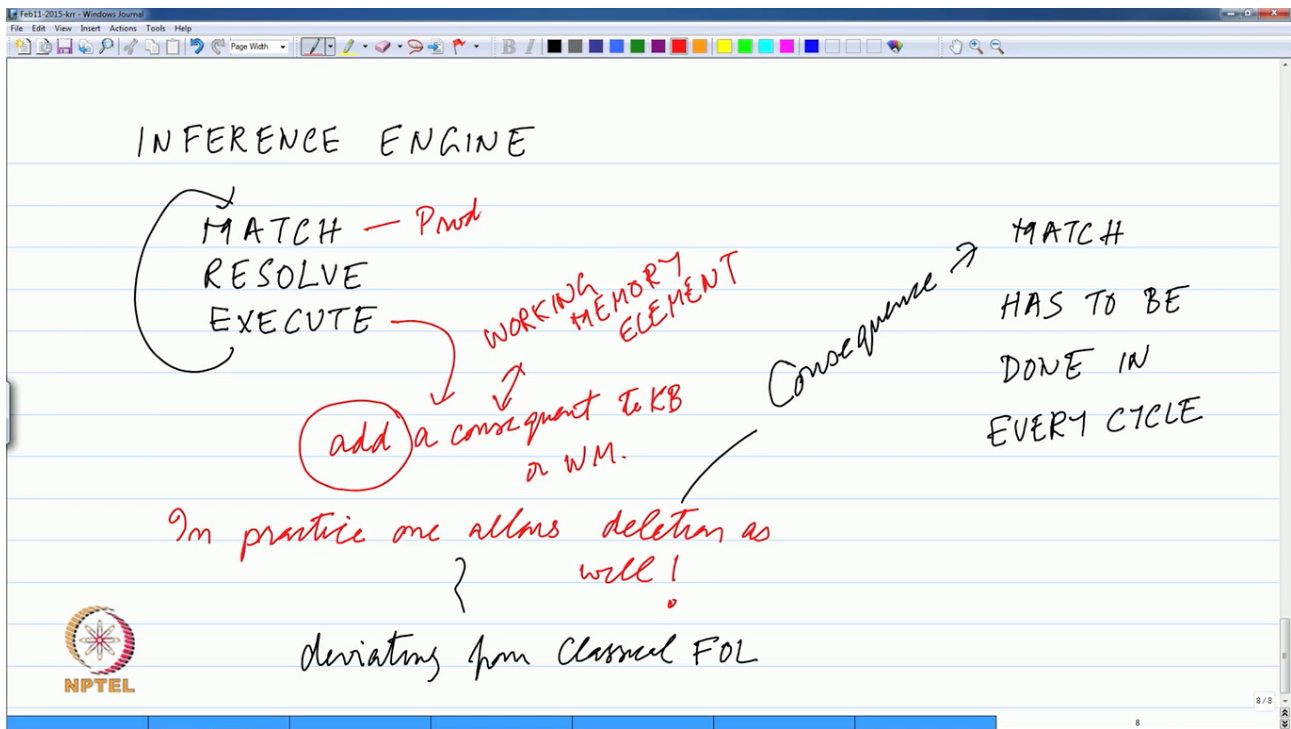
The third step in this process is execute which is basically the step that we have which is basically that you add the consequent to the knowledge base or in this terminology to add the consequent to the working memory essentially. So the same thing we are doing except that we are saying that we will do all the matchings first then we will decide which one to actually apply. So the match component will find all the matching rules with the corresponding data. The resolve component will choose one of them and execute will basically add the consequent essentially. That's the inference engine, it's a three step process, match, resolve, execute and this is put into a loop.

(Refer Slide Time: 22:51)



So what happens is that it's the match which is the most expensive part. So there is one small thing that I want to state before we move on to match. This execute part we have said it is add a consequent to the knowledge base or working memory and in terms of this notation that we are using, this consequent is also a working memory element. But in practice one allows, so not only do we add a consequent, but we are in some sense deviating from the classical logic. We are devising a language or a system in which we are allowed to delete elements essentially. So just give some thought to this, what is happening to this whole process essentially. So not only do you add new facts to the knowledge base which means new rules may be triggered so on and so forth but you are also allowed to delete facts from the knowledge base. So the consequence of this, what should be the consequence of this, that is the match has to be done in every cycle. What is the cycle we are referring to? We are referring to this cycle of match, resolve, execute. The consequence of the fact that you can delete elements means that something which may have been matching earlier will no longer be matching now essentially. If you are not allowed to delete elements then we could have a simple strategy which says that do a match, produce all the elements, all combinations of rule and matching data, pick one and execute it which means you will add some more facts. Just see as to those new facts which rules they may trigger so add that to the set that match produces.

(Refer Slide Time: 27:43)



So the set that match produces a set which looks like this Rule R followed by matching data as I said di1, di2, din. This is called conflict set, it's just a terminology and you should be aware of it. A conflict set is basically a collection of rules and matching data that match algorithm outputs. The consequence of the fact that we can delete elements is that this conflict set has to be computed every time essentially. If we were not allowed to delete working memory elements or facts from the database then once a rule and its matching data is entered the conflict set it will stay there till it is executed but the fact that we are allowed to delete elements, working memory elements of dataset means that even if a working memory element or a rule and the combination of data has got in the conflict set, because we are deleting one of the antecedents, that rule may no longer be applicable essentially. So it complicates the whole process so in principle we have to match all over again essentially. And it has been show that match consumes eighty percentage of the computational time even with optimisations that we will talk about, match part really takes up all the time in the cycle of match, resolve, executed. Resolved is the most straightforward process and execute is of course constant time essentially.

So in the next class when we meet we will look at an algorithm which is more efficient, it's called the RETE algorithm. So keep in mind that we are working in this match, resolve, execute paradigm where the execute phase may delete or add data to the knowledge base and we want to basically make the next round matching process efficient and in doing that what we really want to do is how much of the match we can carry forward and try to you know do these kind of optimisations essentially. This is what is done in the RETE algorithm and we will see that in the next class.