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 – 72
Rule Based Expert Systems

So, welcome back. Today, we take a say slightly different approach to problem solving. Remember that our goal in this course is to build autonomous agent which can do problem solving on its own and in this course in particular we have been focusing on Search Methods for Problem Solving.

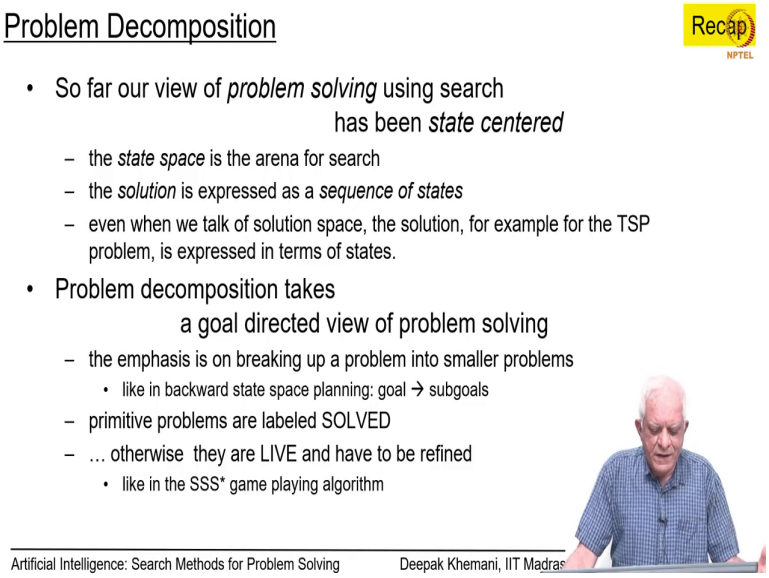
We have kind of shoved under the rug things like how do you represent the state and things like that we did see a little bit of that during planning and now, we are going to see a little bit more of that.

But, in addition to that we will also see that there was a period when a lot of AI work was done using modular chunks of knowledge which are called rules and so, that is why you can see the title of this weeks lectures is broadly speaking Rule Based Expert Systems.

Why expert systems? Because in the mid 80s people thought that the best way to build AI systems is to catch experts in some domain. So, for example, a medical doctor or a computer configuration expert or somebody who has lot of knowledge of geology and things like that and take their knowledge and express it in the form of rules.

We will see a lot of rules as we go along and the idea was that you take their rules and put them into a system and the system will somehow make use of those rules and we will see what do we mean by that in this set of lectures essentially ok. So, we are calling this rule based expert systems.

(Refer Slide Time: 02:13)



So, just a very quick recap we looked at problem decomposition and there we said that we are going to break up a problem into smaller problems and before that we looked at state space search. So, our focus of search was on states problem solving was state centered.

The state space was the arena in which we were doing search and the solution that we were looking for was either expressed as a final state if it was a configuration problem or it was expressed as a sequence of states if it was a planning problem. Even when we talked about solution space, the solution for example, for the TSP problem as we looked at was expressed in terms of the states essentially.

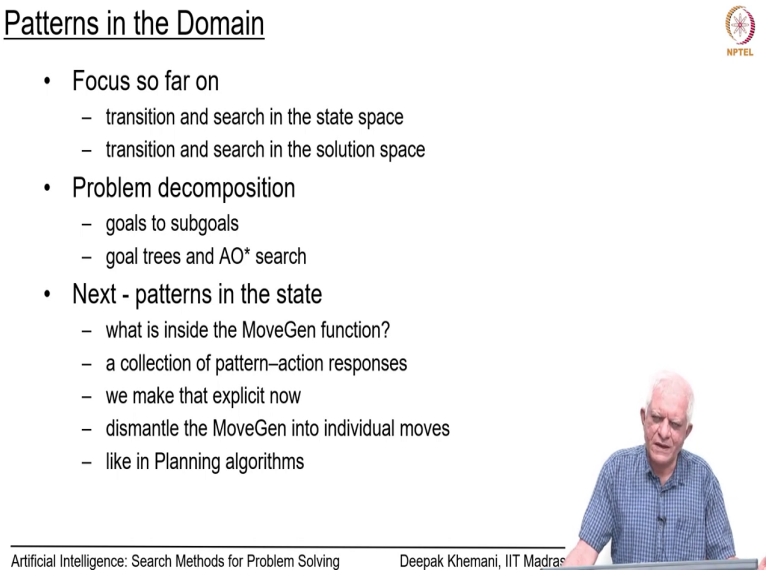
After that we looked at problem decomposition and we said that we will take a goal directed view that if we have a goal or a problem to solve we will break it up into sub goals and address each sub goal separately. And, this approach led us to this idea of goal trees where the

emphasis was is on breaking the problem into smaller problems or and or trees as we call them and we kind of moved from goals to sub goals essentially.

We also saw this when we were looking at forward state space backward state space planning in our case we had two kinds of goals either primitive goals or primitive problems if you want to call them which could be labelled SOLVED and they did not need any further breaking up. So, you might remember for example the symbolic integration things where something like integral x dx was a primitive problem and you did not need to try to solve it any further.

Otherwise they were LIVE goals as we call them and they had to be worked upon to either break them into smaller sub goals or transform them into some other problems and we saw flavour of this also in the SSS star game playing algorithm. So, this slide of course, I have taken from the problem decomposition slide and it is a recap of what we have done so far.

(Refer Slide Time: 04:35)



Now, we are looking at what some people call as pattern directed systems. So, instead of talking about states and solutions or candidate solutions we talk about patterns in the domain essentially.

So, our focus was so far on transition and search in the state space or transition and search in the solution space. So, we always were interested in moving from some element in the space whether it was a state or whether it was a candidate solution to another element in the space looking for the final state or finals final state or the final solution that was the destination essentially.

In problem decomposition we kind of moved away from that a little bit and we say that we will look till your logical point of view or a goal directed point of view and say that if we have this goal to achieve, then what are the sub goals that you want to achieve essentially.

So, for example, if you are planning a trip to Himachal let us say then you have to you know worry about travel, and you have to worry about booking, and you have to worry about local travel and you know all these kind of things which are sub goals which you can solve independently. And, that approach took us to this whole field called AND-OR trees or AND-OR graphs right and we looked at the algorithm AO star which kind of find own optimal solutions in that space.

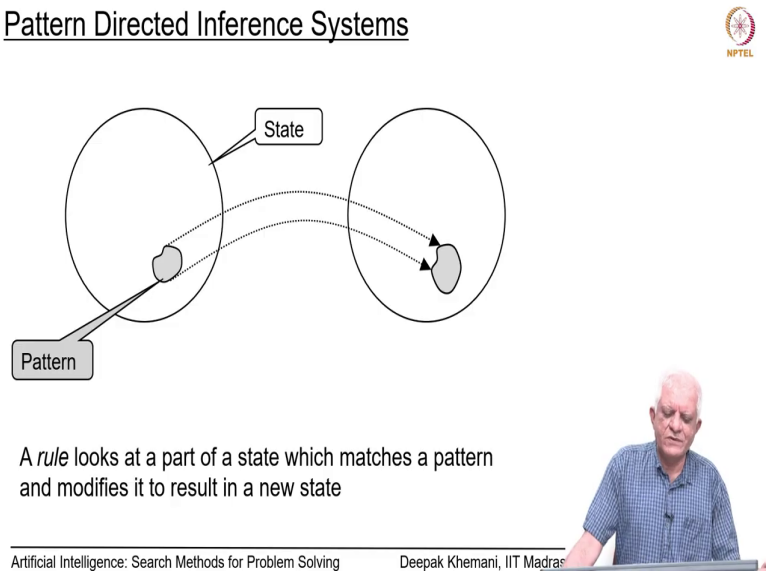
Now, we are interested in looking at patterns in the space essentially. We will start off by looking at what is inside the MoveGen function I mean after all we kind of skirted the issue and we said that the user will somehow give you a MoveGen function and our search algorithm will operate upon that essentially. Now, we will see what is really inside that. We did see that a little bit when we looked at planning algorithms. So, the flavour is not very different going to be very different essentially.

We will see that moves are made up of a collection of pattern-action responses and we will make that explicit now. So, instead of considering all considering the a bird's eye view of the state space or solution space search we will now look at it at a slightly greater level of detail

and look at specific moves or actions that we can do which are tied up to specific patterns essentially. So, and we are going to make that explicit now.

So, in some sense we are going to dismantle the move into individual moves and then we will worry about how those individual moves can be reasoned with essentially which again as I said we did a little bit of that using planning algorithms.

(Refer Slide Time: 07:39)



So, there was this area which at some point used to be called as pattern directed inference systems because essentially you looked at patterns in some representation and based on the pattern you associated rules with them or actions with them and the rules or the actions would modify something which was related only to that pattern.

So, we can see that in this diagram that I have drawn here the state may be a much larger entity and if you are thinking of doing a move from one state to another state essentially you may be modifying part of a state to something different in the final stage.

You can imagine that if you have the task of for example, painting your house and supposing you have decided to do it yourself. Then you could say that this window is not painted so, I should paint the window or the door is not painted I should paint the door or this wall is not painted and so on. So, each move that you can think of doing is really tied up to a particular pattern in the representation that you are talking about and this is what we are going to be referring to now.

So, a rule looks at part of a state which matches a pattern and modifies it to result in a new state essentially. So, rules will have patterns and patterns will match some subset of the states and if they do so, then we will say that the rule can be fired or executed essentially. So, that is going to be the rough idea behind pattern directed inference systems or rule based systems or rule based expert systems yeah.

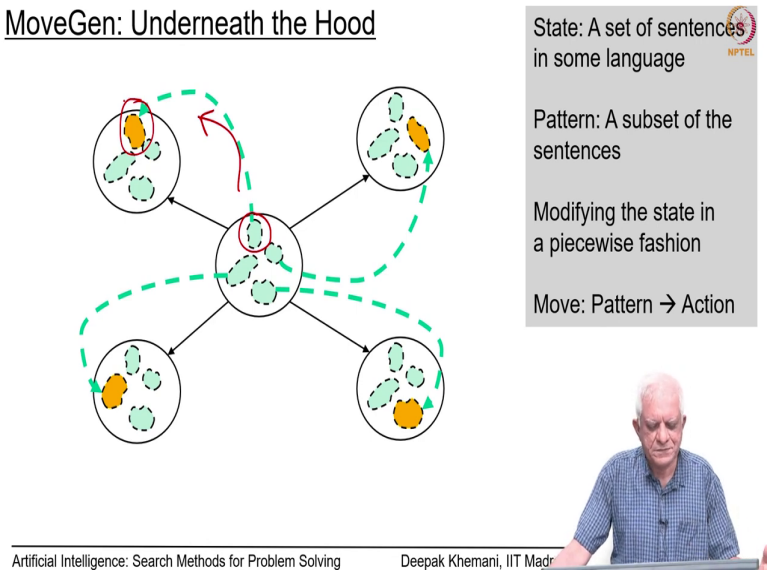
So, I was talking about expert systems also. So, the idea of why these were called as expert systems was the that they would encapsulate the knowledge of a human expert you know whatever the domain of that expert was if you could extract that knowledge and there used to be a whole protocol an exercise of how to extract this knowledge you know it was called knowledge acquisition.

And, people would say that this is how you should talk to an expert and they would record the entire conversations, and then you know convert it into rules and then show it to the expert and say that is this what you wanted to say and do that kind of a thing. So, so, there was this whole process of you know somehow extracting rules from experts and then hope that the system will perform at an expert level.

We did see or we did mentioned expert systems when we were looking at AND-OR trees also we saw the program dendral which outperformed most human chemists in doing the task of

discovering the structural formula of a chemical essentially. So, that was the general idea essentially.

(Refer Slide Time: 10:32)



This is our view of the MoveGen you are in some state and a move gen function gives you a set of neighbourhood states essentially, but we never worried too much about how you would get this set of neighbourhood states. What we are saying now is that a state is a set of sentences in some language. We already saw a little bit of this when we were talking about planning problems ah.

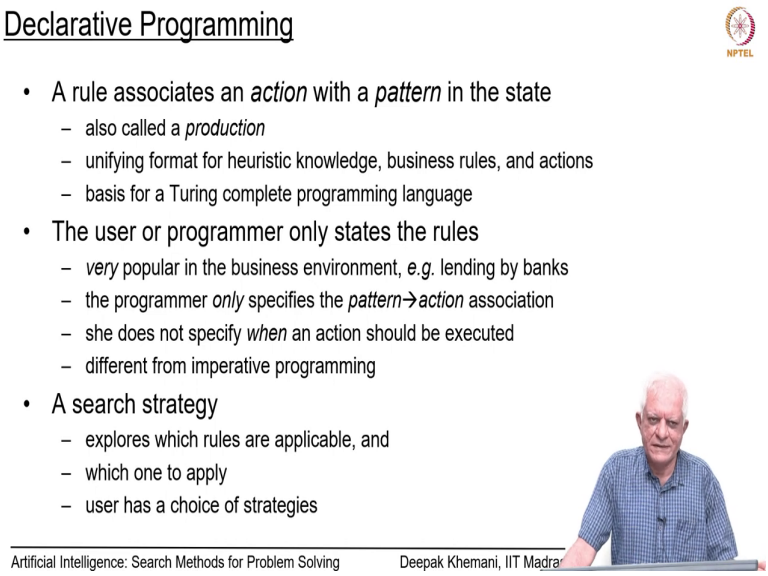
Then a pattern and that is what we are going to talk about is going to relate to a subset of the sentences in the given state. Modifying the state in a piecewise fashion that not the entire state for example, you know I just said if you are painting your house you may want to say paint your window or paint your door and so on.

So, solving part of the problem essentially, but not necessarily in a goal directed fashion as we saw in goal trees. What we are going to see now is in a forward reasoning process and essentially, the flavour of problem solving is going to be the you spot a pattern and you do something about it essentially hm. The overall goal is there of course, at the back of your mind essentially.

And, a move of course, would be a pattern associated with an action. So, if you look underneath underneath the hood of our MoveGen function what you would see is something like this where each pattern in the domain is associated with a particular action and it modifies a part of a domain essentially.

So, in this example I have shown four moves and in some sense they may operate on four different parts of the state you know it could be overlapping, but for the sake of simplicity we have drawn it like this essentially. So, that is the idea of pattern directed in inference system that you look for patterns and you look for actions which are associated with those patterns and implement some way of doing this whole thing.

(Refer Slide Time: 12:37)



Curiously this idea led to the idea of declarative programming. Now, most of the programming that most of us do is classified into what is called as imperative programming languages like C for example, where a program is a set of statements saying that you do this action, you do this action then you do this action and so on and so forth.

So, you are saying what to do and when, and we may introduce conditionals, you may introduce loops and things like that, but essentially a program is a sequence of instructions so, that you want to do. The idea of declarative programming on the other hand is saying that most users of our systems are not going to be expert programmers.

They may not be able to think in terms of algorithms and flow charts and things like that. But, they may have a lot of domain knowledge which we want to make use of and the idea of

declarative programming came from there we will see a little bit of this in this week and also a little bit in the next week essentially.

So, as we said a rule associates an action with a pattern in the state also called a production and so, such systems are also called as production systems. Now, the interesting thing is that a rule could capture many different kinds of things it could capture the heuristic knowledge that a problem solver gave to you it would say that.

For example, if you are making let us say sambar or something like that then add the sambar powder and boil the (Refer Time: 14:26) for 10 minutes or whatever the case may be. So, this is the kind of problem solving knowledge which was captured in the form of rules.

So, a rule would have a as we will see the left hand side and the right hand side and it would capture heuristic knowledge. It would also capture things like business rules essentially and we and this is in fact, the predominant use of rule based systems.

Nowadays, people have given up little bit on the idea of expert systems as of now, but in many domains they have found utility in the form of business rules and typically for example, in banking and so on and the rules also capture the actions that we wanted to do essentially.

So, in some sense everything that one does in building a complex program was expressed in the form of rules we will see some examples as we go along. So, it turned out that in the process rule based systems were like Turing complete programming languages. What is the Turing complete language? As you know is a language in which whatever you can do with a Turing machine you can do in that implement in that language.

So, a language is said to be complete if it is equivalent to a Turing machine. So, all these languages that we work with C, Java, Pascal, Haskell or OCaml or whatever we work with are Turing complete languages and we will see may be today and in the next lecture, some example of another language which is a language which is embodies a notion of declarative programming ok.

So, the idea of declarative programming is that the user or the programmer we do not want to distinguish between the two only states the rules essentially ok. It has been very popular in the business community because you know you know you may have banking experts who would say this is the kind of people that we would want to loan money to that they must satisfy these conditions and so on.

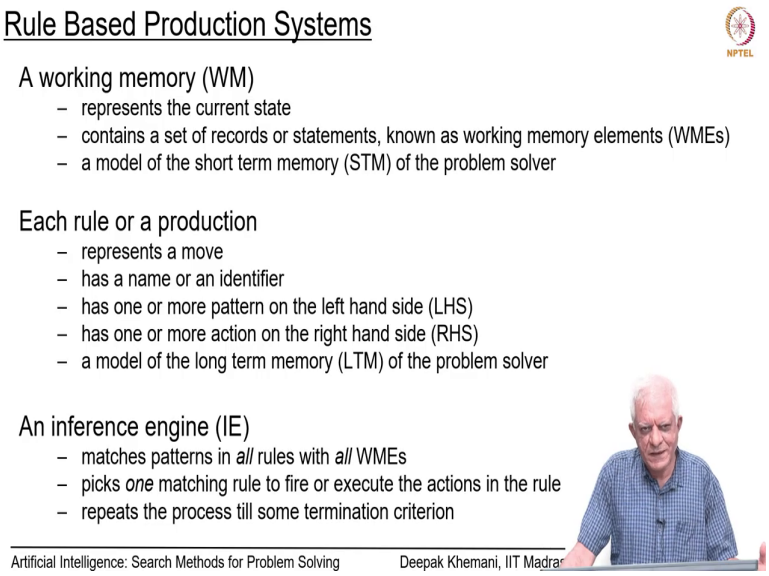
They just want to express that and then if you have to write a program which will do the decision making for you that a new application comes in do you is that applicant qualify for a loan or not is left to a program as we will say which is called an inference engine.

So, the only the programmer only specifies, the pattern action association she does not specify when an action should be said should be executed and as I said this is what makes it different from imperative programming. The user only says that how is a pattern related to an action, when that action has to be done is left to a program which takes these rules and does something with them and we will see that we call that as an inference engine.

The inference engine will embody a search strategy. Which rule to apply at what time is the problem that is going to impact how good our system is essentially. So, we will talk about such strategies which will look at the bunch of rules and a bunch of data and then try to decide which rule should be applied next and this should be an overall strategy which should be problem independent this thing and applicable to the entire thing.

We will define a few strategies and what we will offer to the user is that you can choose one of these strategies for building your rule based system or if you want to call it a rule based expert systems system ok.

(Refer Slide Time: 18:19)



So, we also call them as production systems as I said a little while ago and so, we call often use the term rule based production systems and if you search for this phrase on the web you will get plenty of articles mostly from end of the last century.

The components of a rule based production systems is the primary component is called a working memory it represents the current state essentially. It contains a set of records or statements which are called as working memory elements essentially and the working memory is a model of the short term memory of problem solvers of human problem solvers.

We will see that these ideas they came out of the out of the work of Newell and Simon who did work on human problem solving. They were the pioneers in AI and they were trying to look at how humans solve problems and trying to replicate those kind of strategies in that.

So, the short term memory is the working memory or the working memory is the short term memory. It contains data which is only relevant to the particular problem that you are trying to solve essentially.

Then there are rules. So, there is working memory which is full of working memory elements and then there are rules. So, a rule basically represents a move in some sense it has a name or an identifier. So, just like we had names in planning operators. It is very similar in nature has one or more patterns on the left hand side. We will see this in more details as we go along and has one or more actions on the right hand side.

So, we will use these acronyms LHS and RHS in our representation, but we will call them left hand side and right hand side essentially. And, rules in a system are a model of the long term memory of the problem solver, because rules they stay. Once a once a user knows or the program knows how to solve a particular class of problems, the rules will remain the same, but it is the in problem instance which will change which will be reflected in the changing working memory.

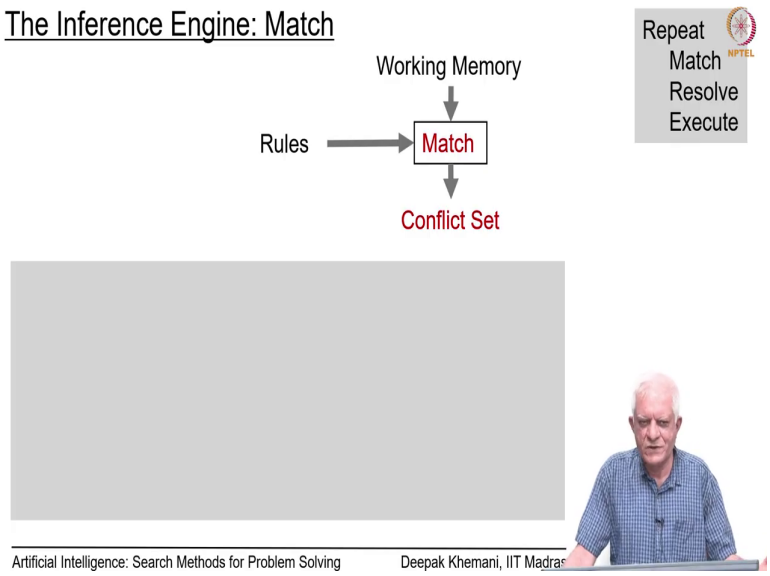
The rules themselves will remain quasi-static essentially they will I mean they hardly ever change. Of course, we learn keep learning and nowadays there is a lot of emphasis on learning and also on how to learn new rules acquire new rules, but that is the different area altogether. We will not get too much into that and the third component as I said is called an inference engine essentially.

So, an inference engine matches to begin with all the rules with all the working memory elements you do not want to miss out on any possible action. So, in some sense our system should be complete that if there is an action which is applicable it should be within the site of the rule based systems essentially.

So, first it does matching of all the rules with all the working memory elements, then it picks one instance of a matching rule to fire or execute. These are the phrases that we use that we fire rule a rule or we execute a rule, and we repeat this process till some termination criterion.

So, in a way we put this thing into infinite loop. It keeps doing that it keeps matching rules with data it keeps picking one rule one matching rule from the set which matches executes it and goes back and does the whole thing all over again and of course, we have to put in some termination criteria which will signal the fact that we have solved our problems. So, as we go along we will see this in a little bit more detail essentially.

(Refer Slide Time: 22:32)



So, let me take a break at this point and we will come back, and look at the details of a rule based system and then get along with strategies and things like that. So, see you in the next video I think.