Rule Based Systems and Expert Systems

Chapter 8

Limitations of FOL

- Consider the statement "Everyone on the enterprise who is not a human or a doctor is a vulcan" from A1
- In FOL: $\forall$x:on(X,enterprise) ^ ~(human(X) v doctor(X)) $\rightarrow$ vulcan(X)
- in Clause form: ~on(X,enterprise) v human(X) v doctor(X) v vulcan(X)
- How many ways can this be resolved?
- 4: with an on, ~human, ~doctor, or ~vulcan

Limitations of Clause Form

- Logically this is of course equivalent to the original statement, but the clause form statement can be used in many different ways, only some of which were ever explicitly expressed in the English statement
- The extras come from the power of logic
- But this isn't a good thing computationally – remember we went to clause form in the first place to use resolution to reduce the number of possible operations at each point

Rules

- The original statement expressed as a RULE is: IF X is on the Quantum-Leap-Guy's Enterprise, and is not a human or a doctor THEN X is a vulcan
- How many ways can this be used?
- take the fact that X is on the enterprise, and the fact that X is not a human and not a doctor and deduce they are a vulcan (FORWARD direction)
- Or assume they are a vulcan and prove they're on the Enterprise, not a human, and not a doctor (BACKWARD direction)

Production Systems

- That, in a nutshell, is why rules are such a popular representation mechanism in AI: they preserve the original semantics of a statement in ways that clause form doesn't, and they are easier to work with computationally
- Prolog uses clause form; most other more specialized systems for performing reasoning (shells) use more obvious rule-based mechanisms or even more powerful forms of knowledge rep.
- A rule is also called a production, and a rule-based system is also called a production system

Production Systems

- Have three components:
- A set of rules (productions)
- A working memory to store intermediate (deduced) facts about the problem at hand
- A control strategy (inference engine) to somehow cycle through the rules and produce new facts
- The first two are pretty straightforward: rules are just IF-THEN statements, with a logical condition that can be evaluated to true or false, and some action to take (deduce a new fact, cause some action to occur)

Control Strategies

- With rules we can work either forward or backward
- Let's look at an example: a trivial domain for identifying animals
- Our rules will be of the form:
  if (cond) then (conclusion)
  Where each condition involves testing attributes of objects the system knows about
- e.g. IF (?x has hair) then (?x is-a mammal)
- Where the ? before a name denotes a variable
- We will cover the details of such object-like representations later, but if you think of it as an object for now, that's enough

Forward Reasoning: naive

- Search rules, applying conditions to facts
- If all conditions in a rule are satisfied, add conclusions to rule set
- Repeat with the other rules in the rule base
- Ignore rules that conclude in facts that have already been asserted
- Repeat this process again and again until no further deductions can be made or until the particular deduction we are interested in (e.g. diagnosis) has been made
- Working in a forward direction involves a lot of potentially useless deductions, because we can't focus on the goal

Example Rule Set

R1: if (?x has hair) then (?x is a mammal)
R2: if (?x gives milk) then (?x is a mammal)
R3: if (?x has feathers) then (?x is a bird)
R4: if (?x flies)^(?x lays eggs) then (?x is a bird)
R5: if (?x is a mammal)^(?x eats meat) then (?x is a carnivore)
R6: if (?x is a mammal)^(?x has pointed teeth)^(?x has claws)
^(?x has forward-pointing-eyes) then (?x is a carnivore)
R7: if (?x is a mammal)^ (?x has hoofs) then (?x is an ungulate)
R8: if (?x is a mammal)^ (?x chews cud) then (?x is an ungulate)
R9: if (?x is a carnivore)^(?x has tawny color)^(?x has spots)
then (?x id is cheetah)
R10: if (?x is a carnivore)^(?x has tawny color)^(?x has stripes)
then (?x id is tiger)

Example Rule Set

R11: if (?x is an ungulate)^(?x has long legs)^(?x has long neck)
^(?x has tawny color)^(?x has spots)
then (?x id is giraffe)
R12: if (?x is an ungulate)^(?x has white color)^
(?x has stripes) then (?x id is zebra)
R13: if (?x is a bird)^(?x cannot fly)^(?x has long legs)
^(?x has long neck)^(?x has black and white color)
then (?x id is ostrich)
R14: if (?x is a bird)^(?x cannot fly)^(?x can swim)
(?x has black and white color)
then (?x id is penguin)

Forward Direction

- Working Memory consists of the current set of objects about which a system can reason – facts that have been deduced thus far or are given to the system to begin with
- Any consultation with the system will consist of adding a set of facts involving the current specific problem and asking the system to draw new conclusions from it
e.g. I might add the following facts in the representation we are using:

Forward Direction

- Stretch has hair
Stretch chews cud
Stretch has a long neck
Stretch has a tawny color
Stretch has dark spots
Stretch has long legs
- Stretch in our case will end up getting bound to ?x in our rules (though obviously we could have >1 variable in rules and >1 object – we would just have more deductions we could make)

Forward Direction

- Consider R1: Stretch does have hair, so we conclude stretch is a mammal
- We can ignore R2 – it concludes in a fact we already know once the variable is bound
- R3...R7 fail
- R8: stretch is a mammal, stretch chews cud, so stretch is an ungulate
  R9-R10 fail, and then R11 succeeds: stretch is a giraffe!

>1 Pass

- Getting a useful conclusion may require a number of passes through the rules, since earlier rules may rely on facts that can only be deduced by later rules
- If R1 and R11 were exchanged, for example, we would never be able to know that Stretch was a mammal until after it was too late to realize Stretch was a giraffe – a second pass would deduce this
- In general, rearranging rules doesn't change what can be deduced, but it does affect efficiency

Forward Reasoning: more realistic

- In real world settings, we do not simply make use of rules whenever we can during forward reasoning
- We look at what we *could* possibly derive, and THEN make a decision as to what to do
- Whenever a rule could be used, we say it is *Triggered*
- It doesn't mean we *are* using it, just that we *could* if we wanted to
- If we make a pass through the rules, we have a set of such triggered rules, which we call our *conflict set*

Forward Reasoning: more realistic

- Once we have the conflict set, we choose (usually) one rule from this, and cause it's conclusion to be run/added (FIRE the rule)
- Why one? And Why build a conflict set?

Non-Monotonism

- Because allowing one rule to fire might cause one of the conditions of the other rules to be retracted – firing order matters very much!
- Having the whole set at least lets us consider all the current possibilities together
- We could rebuild the conflict set and choose the best to fire over and over
- But we usually keep a conflict set and add/delete items from it based on what just changed in working memory, rather than regenerating from scratch
- We can use meta-knowledge to choose the best (most general rule? heuristic to lead us to the goal? Some other type of control knowledge?)

Conflict Resolution

- The process of choosing the best rule to fire from the conflict set is called conflict resolution
- Many schemes for this, mostly domain dependent, and often mixable
- There are domain independent strategies though, but remember they're weak
  - refraction: rules that are fired can't be re-fired on the same data
  - recency: choose rules involving recently added data
  - Specificity: choose rules with > #'s of premises

Production System Control

- An example of forward reasoning reminds us that it is convenient computationally to reason backwards if this is possible, to ensure that we are always performing reasoning relevant to our goal
- If we have a goal, we check to see if it is in WM. If it isn't we look for a rule that concludes in it. If we find one, we adopt the preconditions (antecedents) of that rule recursively as subgoals
- If those subgoals fail, we try to find a different rule concluding in our goal. If we can't find any more, our goal fails.
- Natural depth first search!

Backward reasoning example

- Stretch has hair
  Stretch chews cud
  Stretch has a long neck
  Stretch has a tawny color
  Stretch has dark spots
  Stretch has long legs
- Our goal is to prove stretch is a giraffe
- We would look at rules that conclude in this condition: R11
- R11's first condition is that stretch is an ungulate: prove that

Backward Direction

- We try R7, the first ungulate rule
- This requires us to prove stretch is a Mammal
- We adopt that goal and try R1: success
- Back to R7, we need to prove stretch has hoofs: failure
- Need other ungulate rules: find R8
- Need to prove stretch is a mammal (already done)
  Need to prove stretch chews cud: in working memory

Backward Direction

- Back to R11, and prove other conditions: every other condition is in WM and we are done
- What if we had a more general goal? Find the ID of Stretch?
- We would simply have a broader range of initial rules to select from (R9...R14)

Rules

- A very modular way of representing knowledge
- A natural way to express knowledge in many domains
- Easily understood and readable
- Enforces separation of knowledge and control
- Order of rules is irrelevant logically (that's part of the modularity), but still can affect efficiency
- Typically easier to extend a RBS to handle more complex problems (e.g. uncertainty) than a pure logic-based system

Expert Systems

- Knowledge-based systems are systems whose primary focus is on knowledge that is explicit and is represented separately from the problem-solving algorithm (i.e. declarative representation)
- Expert Systems are knowledge-based systems that perform at a level comparable to that of an expert in a very narrow domain
- Expertise is the value that is added to basic domain knowledge (e.g textbook knowledge)
- ES development is all about emulating a human expert's expertise

Advantages of Expert Systems

- Problems may be solved faster by the system than by the expert
- An expert system provides a permanent record of the expertise
- Expertise can be widely distributed
- Expert systems do not have "bad" days or miss work
- Machine learning techniques may eventually be useful in building and refining a knowledge base

Advantages of Expert Systems

- May be a significant financial advantage to solving problems using an expert system
  - increase the volume of production
  - reduce the number of mistakes
  - reduce the amount of liability
- Developing an expert system may force the experts to develop a better understanding of the problem domain
- ES technology can be combined with other technology (such as neural networks) to create very complex systems

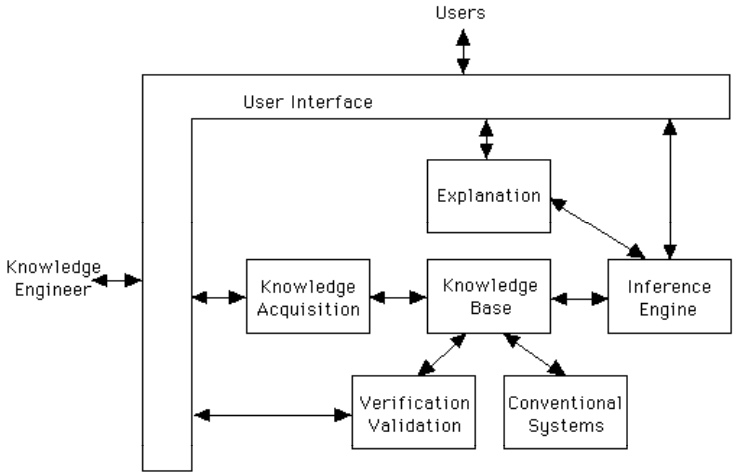
Expert Systems

- The users of the expert system must be knowledgeable about the domain in order to ensure that the advice provided by the system is reasonable
- Expert systems are rarely if ever designed to remove the need for experts; instead, they can be used to make the expertise more widely available or where it is dangerous to send the expert
  - e.g. medical specialists in remote areas, chemical or geological analysis on other planets

Expert System Components

- Expert systems have been developed in a wide variety of domains, such as medical diagnosis, the configuration of computer systems, prospecting for minerals, process control, etc.
- An expert system consists of a variety of components, including a knowledge base, an inference engine, and a user interface that controls the interactions with the users
  - inference eng. is the same term we used in production systems, but NB to remember that expert systems are not necessarily rule-based

Expert System Components



Expert System Shells

- The components of an expert system may be purchased as a package, referred to as a "shell"
- Most shells use a rule-based representation, which may include support for representing uncertainty
- The shell's inference engine may include several control strategies, such as forward search and backward search

Knowledge Engineering

- "The first principle of knowledge engineering is that the problem-solving power exhibited by an intelligent agent's performance is primarily the consequence of its knowledge base, and only secondarily a consequence of the inference method employed. Expert systems must be knowledge-rich even if they are method-poor. This is an important result and one that has only recently become well understood in AI. For a long time AI has focused its attentions almost exclusively on the development of clever inference methods: almost any inference method will do. The power resides in the knowledge."

  – Feigenbaum, in Luger, p. 207

Knowledge Engineering

- Knowledge acquisition is the most difficult problem encountered when developing an expert system
- Experts frequently are not aware of how they solve problems and find it difficult to verbalize the steps that they take
- The knowledge engineer must become a pseudo-expert in the domain in order to be able to converse with the expert

Knowledge Engineering

- Knowledge acquisition and knowledge base refinement are an iterative process
- The knowledge engineer extracts the knowledge, formalizes the knowledge, represents the knowledge, and then tests the resulting system with the expert
- This process continues until the expert is satisfied with the performance of the system

Prototyping

- The development of most expert systems begins with a prototype
- The use of a prototype ensures that the approach (representation technique and inference strategy) is effective and that the problem is solvable

Verification and Validation

- The contents of the knowledge base must be inspected carefully to ensure that the knowledge base is correct and complete
- Tools that assist in the manual V&V of a knowledge base are available
  - for example, the graphical display of the knowledge base
  - Consistency checking in rules
  - looking for omissions
- Tools are now being developed that will carry out some of the V&V tasks automatically

Explanation

- One of the features that is provided by most expert systems is an explanation of why a decision was made
- This explanation consists of a summary of the steps/rules that were followed in generating the conclusion; the summary is usually provided in an English-like format

Explanation

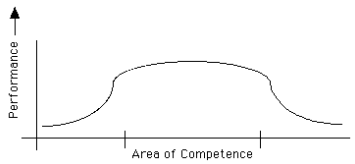
- Having an explanation available creates a higher level of confidence in the conclusion than if an explanation is not available
- Having an explanation facility also makes it easier to debug a system -- the steps that were taken are specified in the explanation
- Systems with good explanation facilities can be used as teaching tools
- It is very difficult to generate an explanation when a procedural representation is used

System Maintenance

- The maintenance of an expert system is an extremely difficult task
- It is almost impossible to specify completely any complex problem – the system will evolve over time
- Just as our knowledge about any specific domain changes over time: not just getting better at solving problems, but new developments in the field

Brittleness

- Expert systems perform very well in narrow areas
- But they rarely if ever take into account general, common sense knowledge
- Expert systems suffer from the "falling off the edge of the cliff" syndrome: performance does not degrade gracefully as problems begin to fall outside of the scope of expertise



Brittleness

- The system is said to be "brittle" at the edges of the expertise
- It is difficult for the system itself to determine when it is within its area of competence and when it has gone beyond its competence -- as a result, the system may provide bad answers to problems that are outside its competence
- Expert systems that **do** degrade gracefully are referred to as "robust" systems
- These are rare indeed – extensive self knowledge!

Surface Reasoning

- In many expert systems, the model of the domain is significantly simplified in order to make the development of the expert system easier
- The reasoning performed in such systems is referred to as "surface" reasoning because the more complex details that lie beneath the surface of the domain are not represented in the model
- Expert systems perform well in domains where this can be done: where there is little common-sense knowledge, or general breadth

Surface Reasoning

- The knowledge often consists of heuristics -- shortcuts or simplifications learned from experience

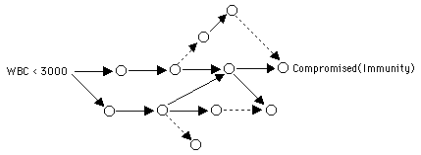
  If (WBC < 3000) Then
  Compromised(Immunity)
- This statement is normally true of medical patients but does not consider the complex chain of interactions that occurs when a patient's white-cell count is low

Surface Reasoning

- Surface reasoning is normally quite quick because the problem space is small
- Many expert systems use surface reasoning very successfully
- In some situations, the simplification (or oversimplification) of a domain may prevent the expert system from reaching the correct conclusion

Deep Reasoning

- In these situations, a more detailed form of reasoning, referred to as "deep" reasoning, is required
- Deep reasoning is often defined in terms of "first principles" or laws of the domain
- Requires a much more elaborate model of the domain



Deep Reasoning

- The search process becomes much slower than when surface reasoning is used because the problem space is much larger
  - Similar in people – that's why we use so many surface models!
- There may be multiple layers of models with increasingly complex descriptions of the interactions of the objects
- We need more sophisticated mechanisms than simple rules to even begin to handle models of this complexity!

Knowledge Representation

- Rules are the most commonly used representation for surface reasoning in ES
- As problems become more complex, the use of rules as a knowledge representation language becomes more problematic because there is no structure to the knowledge base
- For example, in a knowledge base of 10,000 rules, it is difficult to examine the knowledge base and manually determine the impact of making a change

Knowledge Representation

- In complex problems in which *deep* reasoning is used, a structure must be imposed on the knowledge in order to permit the knowledge engineer to concentrate on the significant aspects of a problem instead of having to concentrate on the representation itself
- This structuring also eventually becomes necessary simply to manage a growing base of knowledge, irrespective of deep reasoning!

Limitations of Expert Systems

- The major problem encountered when developing an expert system is building the knowledge base and ensuring that it is correct and complete
- Not all complex problems should be attacked using ES technology
- Do not use ES technology if conventional technologies can be used
- similar to ai in general: if there's a well known algorithm the advantages to using it are clear!

Limitations of Expert Systems

- Expert systems must be integrated with conventional systems (such as database management systems) if real-time response is to be provided
- The development of expert systems is expensive -- a production system will normally require several years to develop at a cost of several million dollars
- However, this cost can be recovered many times over if the system is able to function at an expert level of performance

Mycin Expert System

- Mycin was developed by the AI group and medical faculty at Stanford University to diagnose and recommend therapy for infectious diseases of the blood
- Mycin first diagnoses the disease and then selects the most appropriate antibiotic to combat the disease (or several diseases if the diagnosis can not be limited to one disease)

Mycin Expert System

- Mycin was developed using a rule-based representation language (developed in Lisp) that includes certainty factors
- In blind tests, Mycin has consistently performed as well as human experts
- Mycin has never been commercialized -- in part because the system is limited to only certain types of infectious diseases

Xcon Expert System

- Xcon was developed jointly by the AI group at Carnegie-Mellon and by staff at Digital Corp. to configure computer systems
- Each mistake in configuring a system can cost Digital a significant amount of money as the purchaser has to wait for additional parts
- Xcon is able to recognize invalid configurations (e.g. some components may be incompatible)

Xcon Expert System

- Xcon generates detailed technical descriptions of each system, including a list of all components required, a diagram of spatial relationships between components, the cable lengths, etc.
- Xcon can generate a complete specification in about 1 minute, it takes a team of technical specialists 20 minutes to perform the same task on average – sometimes hours
- Digital expects Xcon to be correct about 95% of the time -- it is realized that Xcon will never have complete knowledge

Polygram Expert System

- Polygram presses 2 million CD's per day (probably a little less now...)
- Each run presses 50 discs; flawed discs must be destroyed (even though the sound is not flawed)
- There are 75-100 potential causes of flaws in generating the images on the CD's (e.g. too much paint on the arm)

Polygram Expert System

- Workers with more than 3 years of experience average 3 runs to correct a problem with an image while workers with less than 3 years of experience average 10 runs
- A prototype ES was developed in 8 weeks and covers about 50 of the problems
- The system was developed using Level 5 Object and uses scanned images of typical printing problems on a 20" touch-screen monitor to direct the user to a solution to a problem

Polygram Expert System

- The ES does not interact with any other systems and is almost completely graphically oriented
- In initial tests with the ES, workers with over 3 years experience required 2 or 3 runs to correct problems while workers with under 3 years required 4 or 5 runs
- It is estimated that the system saves approximately $10 million per year

NSERC Summer Student Opportunities

- Work for one of us (us= your Friendly Neighbourhood Comp Sci Professors) in summer 2007
- NSERC has a program that funds good students to work for people with NSERC research grants (most of our department)
- Good for the student – work with cutting-edge research in some area, get to do something different
  - also get to know the department better, set yourself up for grad school
- Good for us (us=FNCSP) as well – our research funding goes further because NSERC pays part of the research students' salaries

NSERC Summer Student Opportunities

- Forms can be picked up in the dept general office, info about the program can be viewed on nserc's website (nserc.ca)
- Deadline is around the University close time in December.