Expert System in Prolog

Lecture Module-16
Expert Systems (ES)

- Expert systems are knowledge based programs which provide expert quality solutions to the problems in specific domain of applications.
- The core components of expert system are
  - knowledge base and
  - navigational capability (inference engine)
- Generally its knowledge is extracted from human experts in the domain of application by knowledge Engineer.
  - Often based on useful thumb rules and experience rather than absolute certainties.
- A process of gathering knowledge from domain expert and codifying it according to the formalism is called knowledge engineering.
Expert System Architecture

- Inference Engine
  - Inference & Control
- Special Interfaces
- Case History
- Knowledge Base
  - Static database
  - Dynamic database (working memory)
- Explanation Module
- User Interface
- Knowledge Acquisition & Learning Module
- Human Expert
- User

Fig: Architecture of Expert System
Knowledge Base (KB)

- KB consists of knowledge about problem domain in the form of static and dynamic databases.
- Static knowledge consists of
  - rules and facts which is compiled as a part of the system and does not change during execution of the system.
- Dynamic knowledge consists of facts related to a particular consultation of the system.
  - At the beginning of the consultation, the dynamic knowledge base often called working memory is empty.
  - As a consultation progresses, dynamic knowledge base grows and is used along with static knowledge in decision making.
- Working memory is deleted at the end of consultation of the system.
Inference Engine

- It consists of inference mechanism and control strategy.
- Inference means search through knowledge base and derive new knowledge.
- It involve formal reasoning involving matching and unification similar to the one performed by human expert to solve problems in a specific area of knowledge.
- Inference operates by using modus ponen rule.
- Control strategy determines the order in which rules are applied.
- There are mainly two types of control mechanism viz., forward chaining and backward chaining.
Knowledge Acquisition

- Knowledge acquisition module allows system to acquire knowledge about the problem domain.
- Sources of Knowledge for ES
  - text books, reports, case studies,
  - empirical data and
  - domain expert experience.
- Updation of Knowledge can be done using knowledge acquisition module of the system.
  - insertion,
  - deletion and
  - updation of existing knowledge
Case History

- Case History stores the file created by inference engine using the dynamic database created at the time of consultation.
- Useful for learning module to enrich its knowledge base.
- Different cases with solutions are stored in Case Base system.
- These cases are used for solving problem using Case Base Reasoning (CBR).
Explanation module

- Most expert systems have explanation facilities that allow the user to ask the system *why* it asked some question, and *how* it reached to conclusion.
- It contains 'How' and 'Why' modules attached to it.
  - The sub-module ‘How’ tells the user about the process through which system has reached to a particular solution
  - ‘Why' sub-module tells that why is that particular solution offered.
- It explains user about the reasoning behind any particular problem solution.
- Questions are answered by referring to the system goals, the rules being used, and any existing problem data.
<table>
<thead>
<tr>
<th><strong>User</strong></th>
<th><strong>Why module</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Do you have running nose?</td>
</tr>
<tr>
<td>User</td>
<td>Yes.</td>
</tr>
<tr>
<td>System</td>
<td>I am trying to prove that you have cold. Do you have headache?</td>
</tr>
<tr>
<td>User</td>
<td>Yes</td>
</tr>
<tr>
<td>System</td>
<td>Do you have sore throat?</td>
</tr>
<tr>
<td>User</td>
<td>No</td>
</tr>
<tr>
<td>System</td>
<td>Then you do not have cold. Do you have fever?</td>
</tr>
<tr>
<td>User</td>
<td>Yes</td>
</tr>
<tr>
<td>System</td>
<td>Now I am looking at the possibility of measles. Do you have cough?</td>
</tr>
<tr>
<td>User</td>
<td>Yes</td>
</tr>
<tr>
<td>System</td>
<td>I can infer measles using rule “If symptoms are fever, cough, running_nose, then patient has measles” measles is concluded.</td>
</tr>
<tr>
<td><strong>User</strong></td>
<td><strong>How Module</strong></td>
</tr>
<tr>
<td>System</td>
<td>Since you have fever, running_nose and cough and there is a rule “If symptoms are fever, cough, running_nose, then patient has measles”. So measles is concluded for you.</td>
</tr>
</tbody>
</table>
User Interfaces

- Allows user to communicate with system in interactive mode and helps system to create working knowledge for the problem to be solved.

<table>
<thead>
<tr>
<th>Dialogue Module (User Interface)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System</strong></td>
</tr>
<tr>
<td><strong>User</strong></td>
</tr>
<tr>
<td><strong>System</strong></td>
</tr>
<tr>
<td><strong>User</strong></td>
</tr>
<tr>
<td><strong>System</strong></td>
</tr>
<tr>
<td><strong>User</strong></td>
</tr>
<tr>
<td><strong>System</strong></td>
</tr>
<tr>
<td><strong>User</strong></td>
</tr>
<tr>
<td><strong>System</strong></td>
</tr>
<tr>
<td><strong>User</strong></td>
</tr>
</tbody>
</table>
Special interfaces

- It may be used for specialized activities such as handling uncertainty in knowledge.
- This is a major area of expert systems research that involves methods for reasoning with uncertain data and uncertain knowledge.
- Knowledge is generally incomplete and uncertain.
- To deal with uncertain knowledge, a rule may have associated with it a confidence factor or a weight.
- The set of methods for using uncertain knowledge in combination with uncertain data in the reasoning process is called reasoning with uncertainty.
Rule Based Expert Systems

- A rule based expert system is one in which knowledge base is in the form of rules and facts.
  - Knowledge in the form of rules and facts is most popular way in designing expert systems.
- It is also called production system.
- Example: Suppose doctor gives a rule for measles as follows:
  "If symptoms are fever, cough, running_nose, rash and conjunctivitis then patient probably has measles".
- Prolog is most suitable for implementing such systems.

```prolog
hypothesis(measles) :- symptom(fever), symptom(cough), symptom(running_nose), symptom(conjunctivitis), symptom(rash).
```
Simple Medical diagnostic system with dynamic databases

- The system starts with consultation predicate, that initiates dialog with user to get information about various symptoms.
- Positive and negative symptoms are recorded in dynamic database and \texttt{hypothesis(Disease)} is satisfied based on stored facts about symptoms.
- If the hypothesis goal is satisfied then the disease is displayed otherwise display 'sorry not able to diagnose'.
- Finally in both the situations, symptom database for a particular user is cleared.

Query: \texttt{-consultation}. 
consultation :- writeln('Welcome to MC System'),
               writeln('Input your name),
               readln(\Name),
               hypothesis(\Dis), !,
               writeln(\Name, ‘probably has’, \Dis),
               clear_consult_facts.

consultation :- writeln('Sorry, not able to diagnose’),
                clear_consult_facts.
hypothesis(flu) :- symptom(fever), symptom(headache), symptom(body_ache), symptom(sore_throat), symptom(cough), symptom(chills), symptom(running_nose), symptom(conjunctivitis).

hypothesis(cold) :- .................

hypothesis(measles) :- .................

hypothesis(mumps) :- .................

hypothesis(cough) :- .................

hypothesis(chicken_pox) :- .................
Cont…

\[
\begin{align*}
symptom(fever) & : - \text{positive\_symp('Do you have fever (y/n) ?', fever).} \\
symptom(cough) & : - \text{positive\_symp('Do you have cough (y/n) ?', cough).} \\
symptom(chills) & : - \text{positive\_symp('Do you have chills (y/n) ?', chills).} \\
symptom(conjunctivitis) & : - \text{-} \\
symptom(headache) & : - \text{-} \\
symptom(sore\_throat) & : - \text{-} \\
symptom(running\_nose) & : - \text{-} \\
symptom(body\_ache) & : - \text{-}
\end{align*}
\]
positive_ symp(_, X) :- positive(X), !.
positive_ symp(Q, X) :- not(negative(X)),
query(Q, X, R), R = 'y'.
query(Q, X, R) :- writeln(Q), readln(R),
store(X, R).
store(X, 'y') :- asserta(positive(X)).
store(X, 'n') :- asserta(negative(X)).
clear_consult_facts :- retractall(positive(_)).
clear_consult_facts :- retractall(negative(_)).
Forward Chaining

- Prolog uses backward chaining as a control strategy, but forward chaining can be implemented in Prolog.
- In forward chaining, the facts from static and dynamic knowledge bases are taken and are used to test the rules through the process of unification.
- The rule is said to be fired and the conclusion (head of the rule) is added to the dynamic database when a rule succeeds.
- Prolog rules are coded as facts with two arguments, first argument be left side of rule and second is the list of sub goals in the right side of the rule.
Cont…

- Represent prolog rule as a fact by `rule_fact` predicate and simple facts by `fact` predicate.
- Consider the following Prolog rules and facts with their corresponding new fact representations.

  a :- b. ⇒ rule_fact(a, [b]).
  c :- b, e, f. ⇒ rule_fact(c, [b, e, f]).
  b. ⇒ fact(b).
  e. ⇒ fact(e).
  f. ⇒ fact(f).

  – Here a, b, c, e, f are atoms (predicates with arguments, if any).
  – Newly generated facts are stored in database file ‘dfile’ which is consulted in the prolog program.
consult ('dfile').
forward :- finished, !.
forward :- fact(F), doall(rule(F)),
          assertz(used_facts(F)),
          retract(fact(F)), forward.
rule(F) :- rule_fact(L, R), rule1(F, L, R).
rule1(F, L, R) :- member(F, R), delete(F, R, NR),
                 new_rule(L, NR).
new_rule(L, [ ]) :- not(fact(L)), asserta(fact(L)).
new_rule(L, R) :- not(R = []), asserta(rule_fact(L, R)).
finished :- not(fact(X)).
doall(P) :- not(alltried(P)).
alltried(P) :- call(P), fail.