STIN2103
Knowledge engineering & expert systems

Wan Hussain Wan Ishak

SOC 2079  Ext.: 4786  Email: hussain@uum.edu.my  Url: http://www.wanhusain.com
System Design

- System architecture and conceptual design
- Interface design
- Input method
- Explanation facility
Knowledge Engineering Phases

- **Problem Assessment**
- **Knowledge Acquisition & Analysis**
- **Design & Implementation**
- **Testing**
- **Documentation**
- **Maintenance**

Flowchart shows the process of knowledge engineering, with arrows indicating the progression and feedback loops.
Involve the following tasks:

- Task 1: Select knowledge representation technique
- Task 2: Select control technique
- Task 3: Select ES development software
- Task 4: Develop the prototype
- Task 5: Develop the interface
- Task 6: Develop the product
Get some initial insight into how the system’s knowledge can best be controlled by asking the expert to work through a typical problem.

Forward chaining

- Data driven
- Is appropriate if the expert first collects information about the problem and then sees what can be concluded.
- The amount of data < than the number of solutions
Begins with initial information about the problem being asserted into working memory.
- information from a database, sensors or users.

The engine scans the rule looking for ones whose premises match the contents of the working memory.

If a match is found
- the system fires the rules
- places the rule’s conclusion in the working memory
- runs the rules again
- repeats the same process until no additional rules fire
Forward Chaining
Input data is required by the forward chaining system to get started.

This can be obtained through the use of a startup rule:

Rule 1:

IF task IS begin
THEN ASK car problem
Data-driven structures is used to control or determine when a rule can fire:

Compare:

IF lights don’t brighten
THEN tasks is test battery

with:

IF task is test battery connection
AND lights don’t brighten
THEN task is test battery
Consider example of automobile diagnostic domain

- Forward chaining is suitable for diagnostic problems with very large number of possible faults.
- One problem in automobile diagnostic domain is ‘engine does not start’
  - This could caused by faulty with cranking system, ignition system, fuel system or engine compression.
Startup rule is written to automatically fire when the system starts.

The premise of the startup rule must be asserted into the working memory, then when the rule fires, it will cause question to be asked:

what is the problem?
- Car won’t start
- Car hesitates at high speeds
- …
Engine doesn’t start

yes

no

Test cranking system

good

bad

Test ignition system

ok

not ok

Test fuel system
1) new_derived_fact(Cond1):-
   if Cond then Conc1,
   not fact(Cond1),
   composed_fact(Cond).

2) composed_fact(Cond):-
   fact(Cond),!.

3) composed_fact(Cond):-
   new_derived_fact(Cond),!.

4) composed_fact(Cond1 and Cond2):-
   !,
   composed_fact(Cond1),
   composed_fact(Cond2).

5) composed_fact(Cond1 or Cond2):-
   composed_fact(Cond1);
   !,
   composed_fact(Cond2).

6) composed_fact(Cond):-
   write("Q: "),
   write(Cond),
   write("? [yes/no]"),
   nl,
   write("ANS"),
   read(Ans),
   process(Ans,Cond).

---

<table>
<thead>
<tr>
<th>R1</th>
<th>IF</th>
<th>OR</th>
<th>THEN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IF</td>
<td>OR</td>
<td>THEN</td>
</tr>
<tr>
<td></td>
<td>physiciant knows patient has meningitis</td>
<td>we suspect meningitis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td>infection is meningitis</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R2</th>
<th>IF</th>
<th>OR</th>
<th>THEN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IF</td>
<td>OR</td>
<td>THEN</td>
</tr>
<tr>
<td></td>
<td>we suspect meningitis from test results</td>
<td>we suspect meningitis from patient symptom</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td>we suspect meningitis</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R3</th>
<th>IF</th>
<th>AND</th>
<th>THEN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IF</td>
<td>AND</td>
<td>THEN</td>
</tr>
<tr>
<td></td>
<td>tests were run</td>
<td>cultures were seen</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and</td>
<td>cultures look like meningitis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>then</td>
<td>we suspect meningitis from test results</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R4</th>
<th>IF</th>
<th>AND</th>
<th>THEN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IF</td>
<td>AND</td>
<td>THEN</td>
</tr>
<tr>
<td></td>
<td>the appearance of the culture is coccus</td>
<td>the stain of the culture is gramos</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and</td>
<td>cultures look like meningitis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>then</td>
<td>we suspect meningitis</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R5</th>
<th>IF</th>
<th>AND</th>
<th>THEN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IF</td>
<td>AND</td>
<td>THEN</td>
</tr>
<tr>
<td></td>
<td>patient is suffering persistent headaches</td>
<td>patient is suffering dizziness</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and</td>
<td>patient has been lethargic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>then</td>
<td>we suspect meningitis from patient symptom</td>
<td></td>
</tr>
</tbody>
</table>
Backward chaining

- Goal driven
- If expert first considers some conclusion or goal, then attempts to prove it by searching for support information.
- Concern with proving some hypothesis or recommendation.
- If the number of goals < than the amount of data.
The principal objective – is to prove some goal or hypothesis.

Begins by collecting a set of rules that contain the goal in their THEN part – goal rules.

Goal rules, will fire only if its premises are true.

The premises of the goal rule may themselves be supported by other rules.

Premise primitive – premise that is not supported by any of the system’s rules.
Backward Chaining
The system asks to use a question and the answer is placed in the working memory.

Will cause the firing of rules – add their conclusions to the working memory.

Process continues until all the goals and sub goals have been searched.
Begins with THEN part
(\text{IF } a \text{ AND } b \text{ THEN } c)\)

may be supported by other rules
(\text{IF } q \text{ OR } w \text{ THEN } a)\)

Premise primitive
(q)

Ask question
q?

Premise primitive
Ask question
Q: Tests were run?

R1: IF physician knows patient has meningitis OR we suspect meningitis THEN infection is meningitis

R2: IF we suspect meningitis from test results OR we suspect meningitis from patient symptom THEN we suspect meningitis

R3: IF tests were run AND cultures were seen AND cultures look like meningitis THEN we suspect meningitis from test results

R4: IF the appearance of the culture is coccus AND the stain of the culture is grampos AND cultures look like meningitis THEN we suspect meningitis

R5: IF patient is suffering persistent headaches AND patient is suffering dizziness AND patient has been lethargic THEN we suspect meningitis from patient symptom
1) is_true(Conc):-
    if Cond then Conc,!,
    is_true(Cond),!,
    write('CONC: '),
    write(Conc),nl.

2) is_true(Cond):-
    fact(Cond),!.

3) is_true(Cond1 and Cond2):-
    !,
    is_true(Cond1),
    is_true(Cond2).

4) is_true(Cond1 or Cond2):-
    !,
    is_true(Cond1);
    !,
    is_true(Cond2).

5) is_true(Cond):-
    write('Q: '),
    write(Cond),
    write('? [yes/no]'),
    nl,
    write('ANS'),
    read(Ans),
    process(Ans,Cond).

% process user input/response

6) process(no,_):-
    fail.

7) process(yes,Cond):-
    assert(fact(Cond)).
## Selecting Control Strategy

<table>
<thead>
<tr>
<th>PROBLEM TYPE</th>
<th>INFEERENCE</th>
<th>BACKWARD</th>
<th>FORWARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>Low</td>
<td></td>
<td>High</td>
</tr>
<tr>
<td>Design</td>
<td>Low</td>
<td></td>
<td>High</td>
</tr>
<tr>
<td>Diagnosis</td>
<td>High</td>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Instruction</td>
<td>High</td>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Interpretation</td>
<td>Avg.</td>
<td></td>
<td>High</td>
</tr>
<tr>
<td>Monitoring</td>
<td>Low</td>
<td></td>
<td>High</td>
</tr>
<tr>
<td>Planning</td>
<td>Low</td>
<td></td>
<td>High</td>
</tr>
<tr>
<td>Prediction</td>
<td>Avg.</td>
<td></td>
<td>High</td>
</tr>
<tr>
<td>Prescription</td>
<td>Avg.</td>
<td></td>
<td>Avg.</td>
</tr>
<tr>
<td>Selection</td>
<td>High</td>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Simulation</td>
<td>Low</td>
<td></td>
<td>High</td>
</tr>
</tbody>
</table>
Advantages of Forward Chaining

- It works well when the problem naturally begins by gathering information and then seeing what can be inferred from it.
- Can provide a considerable amount of information from only a small amount of data.

<table>
<thead>
<tr>
<th>Data</th>
<th>New Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raining</td>
<td>→ Grass is wet</td>
</tr>
<tr>
<td></td>
<td>→ Game cancelled</td>
</tr>
<tr>
<td></td>
<td>→ can’t mow grass</td>
</tr>
<tr>
<td></td>
<td>→ go to theater</td>
</tr>
</tbody>
</table>

- An excellent approach for certain types of problem solving tasks, such as planning, monitoring, control, and interpretation.
Disadvantages of Forward Chaining

- Have no means of recognizing that some evidence might be more important than others – the system will ask all possible questions, even though it may only need to ask a few questions to arrive at a conclusion.

- The system may also ask unrelated questions.
Advantages of Backward Chaining

- Works well when the problem naturally begins with hypothesis.

- Remains focused on a given goal – produces a series of questions on related topics.

- Excellent approach for certain types of problem solving tasks, such as diagnostics, prescription and debugging.
- It will continue to follow a given line of reasoning even if it should drop it and switch to a different one.
ES Software

Expert System Software

Languages
- Rule-Based
- Object-Based

Shells
- Rule-Based
- Frame-Based
- Induction
- Hybrid

Knowledge engineering
expert systems
Selecting ES Software

Considerations:

- **General**
  - Cost, hardware, license, training / support

- **Developer interface**
  - Coding knowledge, inexact reasoning, rule sets, external program access, debugging utilities

- **User interface**
  - Questions, explanations, graphics, hypertext, multimedia elements, web access
Most ES projects begin the development by building a small prototype system.

Prototype is a model of the final system.

Serve the purposes:

- Validates the ES approach
- Confirms the choice of the knowledge representation technique and control strategies.
- Provides a vehicle for knowledge acquisition.
User views the program through the system’s interface.

The acceptance of a system will depend on how well the interface accommodates the needs of the user.

Developers should pay attention to every aspect in interface design, including the introductory display, the questions asked, the screen directions, the explanation and the conclusion display.
**Introductory display**
- Is used to tell user about the overall purpose of the system
- Provide insight into what the system will do, how the system will perform the task etc.

**Questions**
- Most of the interaction takes place in a form of Q & A session, therefore the questions asked must be clear enough to user
- Most shells ask questions automatically generated from the found primitives in the rules. In fact most of them permit developers to tailor the questions to avoid ambiguity problems.
Screen directions

- Clear directions on what options available on the current screen and how to use them will be very helpful.

E.g:
- Please press ENTER to continue the session
- ... Question ... Select as many as apply
- To exit the system press F10. If you would like to restart the session press F3.
Conclusion display

■ Is used to present to the user the system’s findings
■ In addition to the final recommendation, user should also receive some reasons why the recommendation was made. Most of the system provide this ability through the use of HOW button.
Keys to effective design:
- Consistency
- Clarity
- Control

Consistency – consistent screen format
- Various types of materials are presented on screen such as title, questions, answers, explanation & control functions.
- When presenting these materials, make sure that similar material is place in the same location.
Clarity – clarity of presented materials

- Screens are used to:
  - Ask questions
  - Provide explanation on the system’s reasoning
  - Display intermediate of final results

These materials should be presented in a clear manner for two reasons:

- To ensure the user will be receptive to the system
- To enhance the reliability of exchanging information between the user and the system
Asking clear questions
- To minimize errors in the user’s response, which will lead to erroneous findings by the system

Providing clear explanation and result
- To avoid confusion
- Don’t present a lengthy discussion of a question or result
- Don’t include so many graphics to avoid screen becoming too “busy”
- Provide only the necessary material
Control – screen control

- To ensure the user always feel in control when using the system
- To convince the user that any mistake he/she might make couldn’t lead to disastrous consequences.
- Minimum requirement: easy to start, exit and access the system’s explanation
- Incorporate with the use of an interactive technique such as mouse, light pen, touch screen etc.
Control – screen control

- To ensure the user always feel in control when using the system
- To convince the user that any mistake he/she might make couldn’t lead to disastrous consequences.
- Minimum requirement: easy to start, exit and access the system’s explanation
- Incorporate with the use of an interactive technique such as mouse, light pen, touch screen etc.
A trademark of a KBS/ES is its ability to explain WHY a question was asked and HOW a recommendation was derived.

**WHY**
- The system uses a forward chaining approach when trying to explain why it asked a question.
- Is used when a user wants to know why the system asks a question.
HOW

- The system uses a backward chaining approach when trying to explain how it derived a conclusion.
- Is used when a user wants to know how the system derives a conclusion or recommendation.

Explanation for both WHY and HOW questions must be clear and concise.