1

Knowledge Based Systems

(CM0377)
Lecture 10

(Last modified 19th March 2001)

2

Expert Systems

- One very important kind of knowledge-based system is the *expert system.*
- May be defined, briefly, as:
  - Tool for dealing with problems normally requiring involvement of a human expert, professional or specialist. Often able to (partially) justify its behaviour.

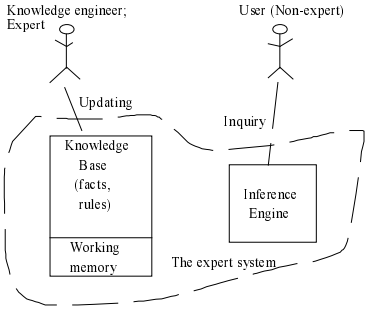
3

BCS ES specialist group definition

'An expert system is regarded as the embodiment within a computer of a knowledge-based component, from an expert skill, in such a form that the system can offer intelligent advice or take an intelligent decision about a processing function. A desirable additional characteristic, which many would consider fundamental, is the capability of the system, on demand, to justify its own line of reasoning in a manner directly intelligible to the enquirer. The style adopted to attain these characteristics is rule-based programming.'

4

(One) basic expert system structure



5

Production rules

- Rule-Based Expert Systems very common.
- A Production Rule has form:
  if <conditions>
  then <actions/deductions>
- Normally Production Rules work in conjunction with *working memory*. Thus:
  - <conditions> test the state of the working memory; and <actions/deductions> change it.

6

Forward/backward chaining

- Consider:

```
IF there's a knock on going over a bump
AND there is nothing loose in the boot
THEN the problem is a worn-out ball joint
```

- *If* the 2 conditions are satisfied, *then* deduce the conclusion.

7

Forward/backward chaining

- Can invoke this rule in *forward chaining mode,* i.e. knowing conditions are true, infer conclusion.
- Alternatively, suppose you want to prove problem is a worn out ball joint.
- *True* if both conditions true; try to establish truth of these conditions—*backward chaining.*
- **Forward chaining** *(data-driven reasoning):* rule 'fired' when conditions are true;
- **Backward chaining** *(goal-directed reasoning):* rule which can deduce what we're trying to deduce is chosen and conditions checked.

8

What is a production system?

- (Forward-chaining system)
- Defined by:
  - *Set of production rules.* Condition-action pairs.
  - *Working memory.* Description of current state of world in a reasoning process.
  - *Recognise-act cycle.*

9

The recognise-act cycle

1 Match conditions of all rules against elements in WM

2 If >1 rule could fire, decide which to apply *(conflict resolution)*

3 Apply the rule, perhaps amending WM, then repeat from (1)

10

Jess - an expert system shell

- Later we'll learn how to implement a forward chaining inference engine in Prolog, but this time we'll consider JESS ("Java Expert System Shell"):
  - Implemented in Java
  - Closely related to OPS5 and CLIPS
  - Forward chaining
  - Uses special technique (Rete algorithm) for efficiency *(also to be covered later)*
  - Can interface to and from Java if desired

11

Running Jess

Type:

java -classpath /opt/jess jess.Main <knowledge-base-name>.clp

Manuals, etc., available at:

http://herzberg.ca.sandia.gov/jess/

It's much more sophisticated than we have time to see here!

12

Example

Diagnosis of computer fault:

- If the mains light isn't on then conclude that the user should plug it in(!)
- If the machine boots but crashes then try reinstalling the operating system
- If it still crashes after reinstallation of OS, take it to a specialist service station
- If it doesn't crash after reinstallation of OS, then maybe fixed: keep an eye on it

13

The Jess program

```
(deftemplate stage
(slot value ))

(deftemplate mains_light_on
(slot value))

(deftemplate boots
(slot value))

(deftemplate crashes
(slot value))
```

(ctd.)

14

The Jess program (ctd.)

```
(defrule start
=>
(assert (stage (value 1)))
(printout t "Is the mains light on?")
(assert (mains_light_on (value (readline))))
(printout t "Does it boot?")
(assert (boots (value (readline))))
(printout t "Does it crash later?")
(assert (crashes (value (readline))))
)
```

(ctd.)

15

The Jess program (ctd.)

```
(defrule idiot_check
(stage (value 1))
(mains_light_on (value "no"))
=>
(printout t "Your problem is that your computer isn't
   plugged in!" crlf)
)
```

(ctd.)

16

The Jess program (ctd.)

```
(defrule try_reinstalling
(boots (value "yes"))
?cr <- (crashes (value "yes"))
?st <- (stage (value 1))
=>
(printout t "Try reinstalling the operating system"
   crlf)
(printout t "Now does it crash?")
(retract ?st ?cr)
(assert (crashes (value (readline))))
(assert (stage (value 2)))
)
```

- (ctd.)

17

The Jess program (ctd.)

```
(defrule dunno
(stage (value 2))
(crashes (value "yes"))
=>
(printout t "Take it to a specialist service centre" crlf))

(defrule watch_it
(stage (value 2))
(crashes (value "no"))
=>
(printout t "Let's hope you've fixed it. Keep an eye on it."
   crlf))

(reset)
(run)
```

18

Main features used

- *Templates* for *working memory elements*
- *Rules* where LHS is matched against working memory and RHS is the action to 'fire' if LHS is satisfied
- The 'assignments' are actually assigning temporary variables to the index numbers of WMEs, so that they can be retracted