Artificial Intelligence: Expert Systems Lecture 1

Forward and backward chaining

Expert Systems (ES)

- Overview
- Facts & rules
- Inference engines
- Conflict resolution

Overview

Definitions

"An Expert System is a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice"

Peter Jackson

"An Expert System is an A.I. program that uses knowledge to solve problems that would normally require a human expert"

Finlay & Dix

Overview (cont.)

Typical ES tasks

- Interpretation of data e.g. solar signals.
- Diagnosis of malfunctions e.g. electronics, humans.
- Configuration of complex objects e.g. computer systems.
- Planning action sequences e.g. robot movement.

Overview (cont.)

Characteristics of a good ES

- Deal with matters of realistic complexity which would normally require human expertise.
- High performance: competency ≥ expert.
- Good response: solution time < expert.
- Reliable: not prone to crashing.
- Understandable: capable of explaining/ justifying conclusions.
- Flexible: knowledge can be added/deleted as required.

Overview (cont.)

ES advantages

- Explanation: shows reasoning behind decisions.
- Uniform: decisions not influenced by pressure situations, moods, etc. Responses are complete.
- Fast: useful information returned quickly.
- Tutoring: can act as an intelligent tutor.

Overview (cont.)

- ES are designed to be experts in only one *problem domain* e.g. medicine, finance, science, engineering.
- An expert's knowledge about problem solving is the *knowledge domain* e.g. detecting diseases, advising on investments.
- Knowledge domain is subset of problem domain.
- An ES should be able to create new facts from existing ones: *inference*.

Facts & rules

Facts

(gives daisy milk)
(lives-in daisy pasture)
(has daisy hair)
(eats daisy grass)

Rules

(Rule 1 (has ?x hair) => (is ?x mammal))
(Rule 2 (is ?x mammal) (has ?x hoofs)
=> (is ?x ungulate))
(Rule 3 (is ?x ungulate) (chews ?x cud)
(goes ?x moo) => (is ?x cow))

Note: 1. *ungulate* means having nail, claw, or hoof
2. `=>' is only used to improve Lisp readability

Facts & rules (cont.)

Rule parts

- Implication: the `=>' in the rule
- Antecedent: the part of the rule before `=>'
- Consequent: the part of the rule after `=>'

Other aspects

- Read a rule as "antecedent implies consequent" or IF antecedent THEN consequent
- Rules can be used to represent AND, OR, and NOT implicitly or explicitly.

Inference engines

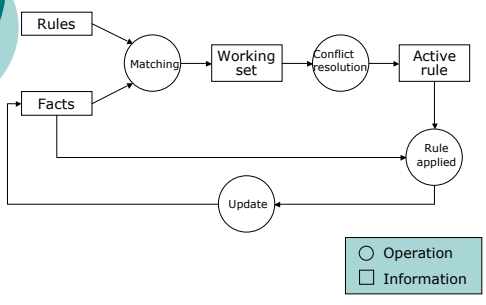
A definition

"A generic control mechanism that applies the axiomatic knowledge present in the knowledge base to the task-specific data to arrive at some conclusion"

Mechanisms

- Forward chaining
- Backward chaining

Inference engine schematic



Forward chaining

Basic principle

1. Start with some initial facts.
2. Keep using the rules to draw new conclusions.

- Forward chaining systems are data-driven.
- Apply when all initial facts known.
- Similarities with bottom-up design: grouping of lower level concepts into higher level ones.

Forward chaining basic example

Rule1: IF *hot* AND *smoky* THEN ADD *fire*

Rule2: IF *alarm_beeps* THEN ADD *smoky*

Rule3: IF *fire* THEN ADD *switch_on_sprinklers*

Fact1: *alarm_beeps*

Fact2: *hot*

Forward chaining checks to see if Fact1 holds

Fact3: *smoky*

Fact4: *fire*

Fact5: *switch_on_sprinklers*

Forward chaining algorithm

Exhaustive application of rules over facts...

Forward chain

UNTIL no change occurs DO
FOR each rule in the ruleset DO
IF all antecedents are facts
AND not all consequents are facts
THEN
add consequents to facts (avoid duplicates)
& note that a change has occurred

Forward chaining example

```
(defvar facts1
  '((big elephant) (small mouse) (small sparrow)
    (big whale) (ontop elephant mouse) ))

(defvar rules1
  '((Rule 1 (heavy ?x) (small ?y) (ontop ?x ?y) =>
            (squashed ?y) (sad ?x))
    (Rule 2 (big ?x) => (heavy ?x))
    (Rule 3 (light ?x) => (portable ?x))
    (Rule 4 (small ?x) => (light ?x)) ))
```

Forward chaining example (cont.)

```
> (fwd-chain rules1 facts1))
((portable sparrow) (portable mouse) (squashed mouse)
 (sad elephant)
 (heavy whale) (heavy elephant) (light sparrow) (light
 mouse) (big elephant)
 (small mouse) (small sparrow) (big whale) (ontop
 elephant mouse))
```

Forward chaining problems

What happens if more than 1 rule matches the set of facts?

- Which rule gets applied?
- How can you be sure of the same rule being applied in the same circumstances?

- *Think about how you might resolve these problems.*

Backward chaining

Basic principle

1. Start with some hypothesis or goal that you are trying to prove.
2. Look for rules that allow you to conclude the hypothesis or goal.

- Backward chaining systems are goal-driven.
- Apply when not all initial facts known.
- Similarities with top-down design: high level concepts broken down into lower level ones.

Backward chaining basic example

Given Goal 1, can derive goals 2..5 using rules

Rule1: IF *hot* AND *smoky* THEN ADD *fire*
Rule2: IF *alarm_beeps* THEN ADD *smoky*
Rule3: IF *fire* THEN ADD *switch_on_sprinklers*

Goal1: switch_on_sprinklers
Goal2: *fire*
Goal3: smoky
Goal4: hot
Goal5: alarm_beeps

Backward chaining algorithm

Goal directed

bwd-chain(goal, rules)
  IF goal is in facts THEN
    report success
  FOR each rule in ruleset DO
    IF goal is one of rule's consequents THEN
      call bwd-chain on each antecedent & check result
      IF all tests are proved THEN
        add consequents to facts & report success

Backward chaining example

```
(setf *rules*
  '((Rule 1 (has ?x warm-blood) => (is ?x mammal))
    (Rule 2 (is ?x mammal) (has ?x hoofs)  => (is ?x
ungulate))
    (Rule 3 (is ?x ungulate) (chews ?x cud) (goes ?x moo)
            => (is ?x cow))
    (Rule 4 (lives-in ?x pasture) (eats ?x grass)
            => (chews ?x cud) (is ?x herbivore))
    (Rule 5 (has ?x hair)     => (is ?x mammal))
    (Rule 6 (gives ?x milk)  => (is ?x mammal))
    (Rule 7 (is ?x herbivore) (is ?x ungulate) (gives ?x milk)
            => (is ?x cow))))

(setf *facts*  '((gives daisy milk) (lives-in daisy pasture)
               (has daisy hair) (eats daisy grass)
               (has daisy hoofs) ))
```

Backward chaining example (cont.)

Start with the premise *(is daisy cow)*
First problem: Rules 3 and 7.

Backward chaining example (cont.)

Practical examples

See Simon Lynch's intranet site for practical examples of the forward and backward chaining mechanisms.

Try out the examples on the site -- they provide traces to show how the final answers are achieved.

Trace influenced by conflict resolution mechanism.

Conflict Resolution

- Conflicts arise in chaining when 2 rules lead to the same conclusion (*consequent*).
- Chaining algorithm could pick either (known as *non-determinism*).
- Conflicts are bad for reproducibility.
- Need a strategy to resolve the conflict.

Possible strategies are:

- Context limiting.
- Ordering by Rule, Specificity, Data, Size, or Recency.

Conflict Resolution (cont.)

Context limiting

- Reduce the likelihood of conflict by separating the rules into groups, only some of which are active at any time.
- Groups are disjoint subsets of the rulebase.

Rule Ordering

Arrange all rules in one long prioritized list. Use the rule that has the highest priority. Ignore the others.

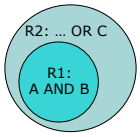
Conflict Resolution (cont.)

Specificity ordering

Whenever the conditions of one triggering rule are a superset of the conditions of another triggering rule, use the superset rule on the ground that it deals with more specific situations.

Example R1 IF A AND B THEN X
R2 IF A AND B OR C THEN X

Favour R2 over R1 as it's more specific.



Conflict Resolution (cont.)

Data ordering

Arrange all possible assertions in one long prioritized list. Use the triggered rule that has the condition pattern that matches the highest priority assertion in the list.

Size ordering

Use the triggered rule with the toughest requirements, where toughest means the longest list of conditions.

- *How is this different from specificity?*

Conflict Resolution (cont.)

Recency ordering

Use the least recently used rule.

Note:

Need to apply strategies consistently otherwise they may as well not be used.

- *What happens if strategies are mixed?*

Review

- ES designed to offer solutions to problems which would normally require a human expert.
- Derive new knowledge from available knowledge: *inference*.
- Forward chaining used when all initial facts known: *data driven*.
- Backward chaining used when final goal known but few initial facts: *goal driven*.
- *Have you come across any concept in AI which has any similarities with chaining?*
- *Can you suggest another conflict resolution strategy?*