Rules, Substrates and Cognitive Modelling

—

Winston, Chapter 8

Michael Eisenberg and Gerhard Fischer
TA: Ann Eisenberg

AI Course, Fall 1997

Explanation in Rule-Based Systems

To answer a question about the reasoning done by a rule-based deduction system,

- If the question is a **how** question, report the assertions connected to the if side of the rule that established the assertion referenced in the question.

- If the question is a **why** question, report the assertions connected to the then sides of all rules that used the assertion referenced in the question.

Certainty Factors

- in many domains: conclusions are rarely certain

- certainty-computing procedures associate a probability between 0 and 1 with each assertions
  certainty factor = 0 ---> an assertion is definitely false
  certainty factor = 1 ---> an assertion is definitely true

- used extensively in Mycin

Knowledge Acquisition

domain expert: a person with expertise in a specific area

knowledge engineer: extracts useful knowledge from domain experts for use by computers

challenge: to make domain experts articulate knowledge, we have to provide them with evocative stimulation (because human knowledge is *tacit*)

Powerful Idea: to learn from an expert,

- Ask about specific situations to learn the expert's general knowledge.
- Ask about situation pairs that look identical, but that are handled differently, so that you can learn the expert's vocabulary.

Rule-Based Systems

Strengths:

- Rule-based systems solve many problems.
- Rule-based systems answer simple questions about hoe they reach their conclusions.

Weaknesses:

- They do not reason on multiple levels.
- They do not use constraint-exposing models.
- They do not look at the problem from different perspectives.
- They do not know how and when to break their own rules.
- They do not have access to the reasoning behind their rules.

Rule-Based Systems as Models for Human Problem Solving

in human problem solving:

- if-then rules ----> production systems
- working memory -----> short-term memory
- rule base memory -----> long-term memory

protocol analysis:

- The state of knowledge is what the subject knows. Each time the subject acquires knowledge through his senses, makes a deduction, or forgets something, the state of knowledge changes.

- The problem-behavior graph is a trace of a subject moving through states of knowledge as he solves a problem.

SOAR — A Cognitive Architecture

A **preference net** is a representation that is a state space in which

- Absolute preferences are identified by Is links that connect states to acceptable, rejected, best, and worst nodes.

- Relative preferences are identified by better, worse, and indifferent links that connect states to each other.

SOAR's Automatic Preference Analyzer

To determine the preferred state using SOAR's automatic preference analyzer,

- Collect all the states that are labeled acceptable.
- Discard all the acceptable states that are also labeled rejected.
- Determine dominance relations as follows:
  - State A dominates state B if there is a better link from A to B but no better link from B to A.
  - State A dominates state B if there is a worse link from B to A but no worse link from A to B.
  - A state labeled best, and not dominated by another state, dominates another states.
  - A state labeled worst, which does not dominate any other state, is dominated by all other states.
- Discard all dominated states.
- Select the next state from among those remaining as follows:
  - If only one state remains, select it.
  - Otherwise, if no states remain, select the current state, unless it is marked rejected.
  - Otherwise, if all the remaining states are connected by indifferent links,
    - If the current state is one of the remaining states, keep it.
    - Otherwise, select a state, at random.
- Otherwise, announce an impasse.

Production Systems

- advantage: simplicity and uniformity of their structure

- productions of two kinds:
  - test on the contents of short term memory (STM); e.g.: "If your goal is to enter the house, open the door" (goal driven)
  - test on the contents of the world; e.g.: "If the door is locked, use your key" (data driven)

A Production System as Foundation for Learning from Examples

9X + 17 = 6X + 23

3X + 17 = 23 (subtract 6X from both sides)

3X = 6 (subtract 17 from both sides)

X = 2 (divide both sides by 3)

production system:

- if expression has the form <variable> = <real number>
  ---> halt
- if expression has variable term on right side
  ---> subtract variable term from both sides and simplify
- if expression has numerical term on left side
  ---> subtract numerical term from both sides and simplify
- if variable term has coefficient other than unity
  ---> divide both sides by coefficient

Critiquing

- **critiquing** = presenting a reasoned opinion about a user's product or action

- critics make the constructed artifact **"talk back"** to the users (beyond the "back-talk" provided by the materials)

- critics should be **embedded** into domain-oriented design environments

- **critiquing process:**
  - goal acquisition:
  - product analysis:
    * differential: compare system and user generated solution
    * analytical: checks products against predefined features
  - critiquing strategies:
    * intrusiveness: active versus passive (see VDDE)
    * adaptation capability (disable critics)
    * explanation and argumentation

- **classes of critics:**
  - educational and/versus performance: primary objective is learning and/versus better product
  - negative and/versus positive

Example: LISP Critic — Transform a "COND" into an "And"

```
(rule cond-to-and-1                    ;;; the name of the rule
     (cond (?condition ?action))       ;;; the original code
   ==>
     (and ?condition ?action)          ;;; the replacement
     safe (machine people))            ;;; rule category
```

Example:

```
(cond (value (eq (cadr value) 1.0)))
                  ---> (and value (eq (cadr value) 1.0))
```

Example: LISP Critic — Replace a Copying Function with a Destructive Function

```
(rule append/.1-new.cons.cells-to-nconc/.1...  ;;; the name
of the rule
  (?foo:{append append1}                       ;;;
    (restrict ?expr                            ;;; the
condition (rule can only be
              (cons-cell-generating-expr expr))  ;;; applied if
cons cells
       ?b)                                     ;;; are
generated by "?expr")
  ==>
  ((compute-it:                                ;;;
    (cdr (assq (get-binding foo)               ;;;
                '((append . nconc)             ;;;  the
replacement
                  (append1 . nconc1)))))                ;;;
       ?expr ?b)                               ;;;
 safe (machine))                               ;;;  rule
category
```

Example (see illustration by Kaestle)

(append (explode word) char) ======> (nconc (explode word) char)

Example — Janus / Kid: Domain-Oriented Design Environments with Critiquing

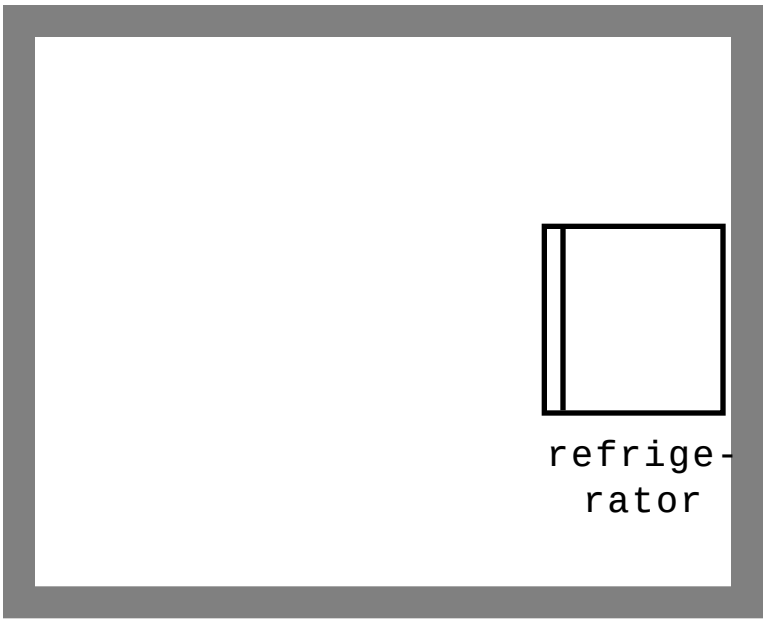
- critic rules analyze the current construction situation and display critic messages if they detect suboptimal designs

- critiquing is implemented as a rule based system; condition of a critiquing rule: spatial relation — examples:
  - away-from (stove, door)
  - next-to (dishwasher, sink)
  - in-front-of (sink-window)
  - work-triangle-small-enough (stove, refrigerator, sink)

- a critic rule that checks for blocked doors:

  name: NOT-DOOR-BLOCKED
  parameters: x y
  condition: (not (overlaps (relative-rectangle x :top-y (depth x) :depth (width x)) y))
  critique: The door of ~A is blocked by ~B.
  praise: The door of ~A is not blocked by ~B.
  documentation: A door of a design unit is blocked if another design unit is in the area in front of the first one.

Using Examples for Defining New Rules
—
Steps towards Machine Learning



Positive Example

Negative Examp

Summary

- The simplicity rule-based deduction systems enables you to build extremely useful modules on top of a basic chaining procedure.

- Explanation modules explain reasoning by using a goal tree to answer how and why questions. Acquisition modules assist knowledge engineers in knowledge transfer from a human expert to a collection of rules.

- Two key heuristics enable knowledge engineers to acquire knowledge from human experts. One is to work with specific situations; another is to ask about situation pairs that look identical, but are handled differently.

- Rule-based systems can behave like idiot savants. They do certain tasks well, but they do not reason on multiple levels, they do not use constraint-exposing models, they do not look at problems from different perspectives, they do not know how and when to break their own rules, and they do not have access to the reasoning behind their rules.

- Rule-based systems can model some human problem solving. SOAR, the most highly developed rule-based model of human problem solving, uses an automatic preference analyzer to determine what to do, instead of using a fixed conflict-resolution strategy.