Rules

Another form of KR

What we will focus on

- “Rule” as a form of knowledge representation
- How do we use them?
  - Rule-based Architectures and Mechanisms
    - Reasoning cycles
    - Control strategies
- Sections covered from Luger are Chapter 5 (5.0-5.3) and Chapter 7 (7.0-7.2)
**Rules**

- Represents knowledge in the form of rules of form: Premises $\rightarrow$ Conclusions
- Are basic KR in Expert systems (Production systems) where a production is of the form
  Conditions $\rightarrow$ Actions
  + “Ask if you don’t understand” = “$\neg$understand(you) $\rightarrow$ Ask(you)”
  - “She ate breakfast at 6am”
  - “They are siblings”
- **Limitation**: can’t explicitly represent events and relations
- **Reasoning mechanisms**: using rules will be discussed next (on Expert systems)

---

**Expert Systems at a glance**

Luger: Strong Method Problem Solving (Ch 7.0-7.2)
Outline

- Production Systems
- Rule-based architectures
- Reasoning cycles
- Control mechanisms
- Application systems vs. Shells
- Current issues

Expert Systems

Part I: Architectures and Mechanisms
Motivation

Problem
- General problem solvers are too weak for high performance systems

Observation
- Expert behavior is not weak (in areas of expertise)

Solution
- Build a “knowledge system” that uses domain knowledge extracted from experts by knowledge engineers

What is an expert system?

Expert systems
(Knowledge-based systems or Knowledge systems)

A computer program that uses large amounts of specific and expert knowledge in solving problems

Expert knowledge
- Often privates to experts (who know more than they articulate)
- Can be heuristic, experiential and uncertain
Characteristics

- Use domain-specific knowledge
- Use domain-specific method (algorithmic and heuristic)
- Perform well in the problem domain
- Flexible and Explainable

Examples of Applications

Expert systems represent broad AI applications in many tasks and domains

<table>
<thead>
<tr>
<th>Systems</th>
<th>Domain</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>MYCIN</td>
<td>medicine</td>
<td>Diagnoses/treats bacterial Infection</td>
</tr>
<tr>
<td>XCON/R1</td>
<td>engineering</td>
<td>Configure DEC VAX systems</td>
</tr>
<tr>
<td>PROSPECTOR</td>
<td>geology</td>
<td>Advice on mineral exploration</td>
</tr>
<tr>
<td>DESIGN ADVISOR</td>
<td>engineering</td>
<td>Critiques chip designs</td>
</tr>
<tr>
<td>DENDRAL</td>
<td>chemistry</td>
<td>Interpret molecular structure</td>
</tr>
</tbody>
</table>
Production systems

A principle of expert systems technology

- Introduced by Post (1943) as a computational model that is as powerful as Turing machine

- Newell & Simon:
  Human problem-solving model ≈ Production systems

Production systems (Luger, Section 5.3)

- Are basic for human-problem solving models
- Basic components:
  - Working memory (WM): data structures representing current state of the system
  - Production memory: set of production rules of form
    \[ <\text{WM pattern}> \rightarrow <\text{WM changes}> \]
  - Rule interpreter: applies production rules to WM
Production systems (contd)

Advantages
- Modularity - by separating memory and interpreter
- Uniformity - by the form of production rules
- Natural

Disadvantages
- Hard to debug and test for consistency for systems with large set of rules

Rule-based systems

Common expert system architectures based on ideas from production systems
- Basic components
  - Facts: stored in database, knowledgebase
  - Rules: of form
    - IF <premise> THEN <conclusion>
  - Inference engine: infers new facts
Knowledge Bases (KB)

- Contain Facts and Rules
  - Facts
    - Temporary or permanent
    - Specific or individualistic
    - Transient and subject to change
    
    **Example:** Allie likes her school. Today is Saturday. It is a very hot day. It might rain this afternoon.

  - Rules
    - Permanent
    
    **Example:** IF it is Saturday THEN Allie doesn’t go to school
Inference Engines (IE)

- Starting for an initial set of facts, IE infers new facts using appropriate rules in the KB
- Repeat the process until the facts obtained matched with the facts of the goal state

Knowledge Base \(\rightarrow\) Inference Engine

- Initial Facts
- New Facts
- Final Facts

≠ Goal facts \(\rightarrow\) ≠ Goal facts \(\rightarrow\) = Goal facts

Reasoning Cycles

Each Inference step = Each Execution cycle

- Start \(\rightarrow\) Recognize \(\rightarrow\) Resolve \(\rightarrow\) Act \(\rightarrow\) Halt

Recognize applicable rules \(\rightarrow\) Resolve applicable rule \(\rightarrow\) Act

- Find all applicable rules
- Select one of the applicable rules
- Execute the selected rule

Pattern Matching \(\downarrow\) Conflict Resolution
Pattern Matching

Example: in language OPS5

Rule001

\[(\text{p grant-degree}) \rightarrow (\text{make approve graduating-status for } <n>))\]

Facts:

\[(\text{student name Mary id 900-80-700 credit-hrs 30})\]

(address number 109 street 10th city BocaRaton)

New Fact: (approve graduating-status for Mary)

An agenda in IE updates list of instantiated rules

Example of execution cycles

A production system to rewrite a string

Given:

- Fact: cbaca
- Production Set:
  1. ba \rightarrow ab
  2. ca \rightarrow ac
  3. cb \rightarrow bc

<table>
<thead>
<tr>
<th>Iter.</th>
<th>WM content</th>
<th>Rules Matched</th>
<th>Rule Fired</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>cbaca</td>
<td>1, 2, 3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>cbaca</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>aclaca</td>
<td>2, 3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>aclaca</td>
<td>1, 3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>aclaca</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>aclaca</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>aclaca</td>
<td>(\phi)</td>
<td>Halt</td>
</tr>
</tbody>
</table>
Another Example of execution cycles

A rule-based system

Given:
- Start fact I, Goal G
- Production Set:
  1. $p \land q \rightarrow G$
  2. $r \land s \rightarrow p$
  3. $w \land r \rightarrow q$
  4. $t \land u \rightarrow q$
  5. $v \rightarrow s$
  6. $I \rightarrow v \land r \land q$

<table>
<thead>
<tr>
<th>Iter.</th>
<th>WM content</th>
<th>Rules Matched</th>
<th>Rule Fired</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>I</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>I, v, r, q</td>
<td>6, 5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>I, v, r, q, s</td>
<td>6, 5, 2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>I, v, r, q, s, p</td>
<td>6, 5, 2, 1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>I, v, r, q, s, p, G</td>
<td>6, 5, 2, 1</td>
<td>Halt</td>
</tr>
</tbody>
</table>

Conflict resolution strategies

- Refractoriness
  Each rule is executed exactly once from the same data
- Recency
  Prefer rules that use more recent data
- Specificity
  Prefer rules that are more specific
- OPS5 uses all of the above
- Other conflict resolution criteria: Select
  - The first activated rule found
  - The rule with highest priority
Inference Control Strategies

- Forward Chaining
  - Start from facts
  - Deduce new facts that eventually lead to goal
- Backward Chaining
  - Take goal as a hypothesis
  - Try to prove a series of subgoals
- Mixed

Example: Forward chaining

Rules
1. (Zebra x) → (Mammal x)
2. (Zebra x) → (Striped x)
3. (Zebra x) → (Medium x)
4. (Mammal x) → (Animal x)
5. (Mammal x) → (Warm x)
6. (Striped x) → (Nonsolid x)
7. (Striped x) → (Nonspotted x)
8. (Medium x) → (Nonsmall x)
9. (Medium x) → (Nonlarge x)

Fact: (Zebra Bo)  Goal: (Nonlarge Bo)

Diagram:

- (Zebra Bo) → (Mammal Bo) → (Animal Bo) → (Warm Bo)
- (Zebra Bo) → (Striped Bo) → (Nonsolid Bo) → (Nonspotted Bo)
- (Zebra Bo) → (Medium Bo) → (Nonsmall Bo) → (Nonlarge Bo)
Example: Backward chaining

**Rules**

1. \((\text{Zebra } x) \rightarrow (\text{Mammal } x)\)
2. \((\text{Zebra } x) \rightarrow (\text{Striped } x)\)
3. \((\text{Zebra } x) \rightarrow (\text{Medium } x)\)
4. \((\text{Mammal } x) \rightarrow (\text{Animal } x)\)
5. \((\text{Mammal } x) \rightarrow (\text{Warm } x)\)
6. \((\text{Striped } x) \rightarrow (\text{Nonsolid } x)\)
7. \((\text{Striped } x) \rightarrow (\text{Nonspotted } x)\)
8. \((\text{Medium } x) \rightarrow (\text{Nonsmall } x)\)
9. \((\text{Medium } x) \rightarrow (\text{Nonlarge } x)\)

**Fact:** \((\text{Zebra } \text{Bo})\)

**Goal:** \((\text{Nonlarge } \text{Bo})\)

- \((\text{Nonlarge } \text{Bo})?\)
- \((\text{Medium } \text{Bo})?\)
- \((\text{Zebra } \text{Bo})?\)  Matches with Fact. Goal is true.

**For this problem**
Backward chaining is more efficient than Forward chaining

---

Choosing control strategies

Factors that effect choice of control strategies

- 
  
  #start states and # goal states:
  Move from smaller set to larger set
  
  **Example:** Finding a route from home to a new destination
  
  Search: Easier to search for home (multiple landmarks) than to search for an unfamiliar place (single location)

Start: Home

Unfamiliar place

Goal

Backward chaining
Choosing control strategies (contd)

- Branching factors in both directions:
  Proceed in the direction with lower branching factors

  Example
  Theorem proving

  Start Axioms

  Backward chaining

Choosing control strategies (contd)

- Match thinking/reasoning direction
  This is for easy explanation and justification

  Example:
  Rules:
  1. sinus $\rightarrow$ infection
  2. fever $\rightarrow$ headache
  3. migraine $\rightarrow$ headache
  4. infection $\rightarrow$ fever
  5. cold $\rightarrow$ headache

  Symptoms causes Disease or cause

  Diagnosis Backward reasoning Explanation
  Symptom A could be caused by ....

  Forward reasoning
  Bob has disease A because it causes all his symptoms
Note

Examples of programming environments that support

• Forward Chaining – CLIPS
  (a rule-based programming language)

• Backward Chaining – Prolog
  (a logic programming language)

Review

- Name two most basic components of rule-based systems
- Name basic control strategies
- Name factors for selecting appropriate control strategies
- Consider a system that automatically calculates a symbol integration
  Start: a formula containing an integral symbol
  Goal: a formula equivalent to the initial one with no integral symbol

*What control strategy should be used? Why? I.e., which factor helps you decide?*
Why control?

- Good performance from an expert system
  - Sensitivity: responds quickly to changes of environment
  - Stability: continues the same line of reasoning

→ Good control

- Control mechanisms in rule-based systems
  - Global controls: domain independent, “hard-coded”
    E.g., Conflict resolution strategies
  - Local controls: domain dependent
    E.g., Meta-rules

Meta-rules

- Meta-knowledge
  Knowledge about reasoning process to solve a given problem. Usually indicates known limits of the knowledge.

- Meta-rules
  Rules to manage or control behavior of other rules
  Example:
  - IF situation-A THEN rules in group 1 are not useful
  - IF there are rules that don’t mention current goal in the premise THEN try rule 10 before rule 145
Efficiency in Rule-based systems

Pattern Matching Process:
- For each rule, search for facts that satisfy the rule’s conditions and place the rule on the agenda
- Maintain & update these rules after each execution cycle

Pros: simple
Cons: slow with temporal redundancy (each cycle might have only a small % of facts changed)

Efficiency in Rule-based systems (contd.)

Remedy - Avoid unnecessary re-computation by
  - Remembering what has already matched from cycle to cycle
  - Computing only the changes necessary for the new facts
→ Use the changed facts to direct the search for rules

Two steps: pattern network and join network
**Expert system shells**

Application systems:
- Systems developed for specific domain and applications

Expert system shells:
- Tools that provide mechanisms to build application systems other than knowledge

Examples:

<table>
<thead>
<tr>
<th>Application systems</th>
<th>Expert system shells</th>
</tr>
</thead>
<tbody>
<tr>
<td>MYCIN</td>
<td>EMYCIN</td>
</tr>
<tr>
<td>XCON/R1</td>
<td>OPS, OPS5</td>
</tr>
<tr>
<td>Guardian</td>
<td>BB1</td>
</tr>
</tbody>
</table>

**Two important capabilities**

Of expert systems other than performing tasks
- Explaining its reasoning
  - Reasoning process must be understandable
  - Enough meta-knowledge
- Acquiring knowledge
  - By interaction with experts
  - By program that learns expert behavior
Current Issues

- Brittleness
  - Break down when encounter unanticipated case
  - Can’t fall back on principle knowledge when needed
- Lack of Meta-knowledge
  - Do not know their own limitations
- Knowledge acquisition
  - A major bottleneck in developing expert systems
- Validation
  - Hard to quantify measurement for its performance
- Real-time expert systems