Knowledge Representation: Production Systems I

References:
Outline

- Production systems
- Knowledge representation issues
- The “recognize-act” cycle
- Inference: forward vs. backward chaining
Procedural vs declarative knowledge

- AI beginnings questioned traditional algorithmic approach to problems
  - Fixed control flow
  - Sequentiality
  - Inadequate for changing environments

- Solution
  - Computation based on available information rather than on fixed control flow
Components of a production system

1. A rule set (or rule base = RB)
2. A database management system (or working memory = WM) used to store the case-specific facts at each time
3. A rule interpreter (control strategy or inference engine) in charge of chaining the operation cycles
Components of a production system: **RB**

- **Production rules**

  Rules or productions have the form \( \textbf{IF} \ <C> \ \textbf{THEN} \ <A> \)
  
  - \(<C>\) : preconditions (LHS = left hand side)
  
  - \(<A>\) : actions (RHS = right hand side)

- **Example (Mycin 1972-1980)**

  **IF:**
  
  [stain of organism is Gram-neg] and
  [morphology of organism is rod] and
  [aerobicity of organism is aerobic]

  **THEN:**
  
  [strong evidence (0.8) that organism is Enterobac]
Components of a production system

- Production rules
  - Rules or productions of the form IF <C> THEN <A>
  - <C>: preconditions (LHS = left hand side)
  - <A>: actions (RHS = right hand side)

Example (Mycin 1972-1980)

**IF:** [stain of organism is Gram-neg] and [morphology of organism is rod] and [aerobicity of organism is aerobic]

**THEN:** [strong evidence (0.8) that organism is Enterobac]
Production rules

- Rules can express expert knowledge

IF: [stain of organism is Gram-neg] and [morphology of organism is rod] and [aerobicity of organism is aerobic]

THEN: [strong evidence (0.8) that organism is Enterobac]
Components of a production system: WM

- The working memory contains facts that are believed to be true during the search.
- Facts that are subject to change while searching are also called “fluents”.
- Facts are matched with the LHS part of a rule (to check whether it can be fired or not) and can be asserted in the RHS of a rule.
Facts

- Case-specific knowledge can be represented as facts (e.g. knowledge about ORGANISM-01)

<table>
<thead>
<tr>
<th>ORGANISM-01</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAM = Gram-neg</td>
</tr>
<tr>
<td>MORPHOLOGY = ROD</td>
</tr>
<tr>
<td>AIR = aerobic</td>
</tr>
</tbody>
</table>
Components of a prod. system: **engine**

- **An operation cycle has ...**
  - Activation phase: select a rule
  - Action phase: apply the selected rule

- **A rule is activated (or “fired”) when ...**
  - its preconditions (LHS) are true in the current state of the WM (forward reasoning), **or**
  - the effect of the rule (RHS) contains something that needs to be established (backward reasoning)

- **The action part of a rule allows to ...**
  - update the data stored in the WM, **or**
  - modify the external world (as a side effect)
Recognize – act cycle

- **Recognize-act** cycle of a rule interpreter:
  - **Activation phase:**
    - Match the premise patterns of rules against elements in the working memory
    - Conflict resolution: If there >1 rules that apply (i.e. can be fired) choose one. If none, stop
  - **Action phase:**
    - Apply the chosen rule, modifying the working memory. If termination condition is met, stop. Otherwise go to 1
  - **Termination condition**
    - defined by a goal test or a condition on the maximum number of steps (i.e. max 100)
Another example

- **Terms**
  - Umbrella
  - Wet
  - Cold

- **Predicates**
  - Raining
  - Outside
  - Have(x)
  - Use(x)
  - State(x)
  - Develop(x)

- **Semantics**
  - yes, it is raining
  - the agent is outside
  - the agent has the object “x”
  - the agent uses the object “x”
  - the agent is in state “x”
  - the agent develops “x”
Another example

- Rules (in RB)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>IF: Raining, Outside, Have(Umbrella)</td>
<td>THEN: Use(Umbrella)</td>
</tr>
<tr>
<td>R2</td>
<td>IF: Raining, Outside NOT Have(Umbrella)</td>
<td>THEN: State(Wet)</td>
</tr>
<tr>
<td>R3</td>
<td>IF: State(Wet)</td>
<td>THEN: Develop(Cold)</td>
</tr>
</tbody>
</table>

- Initial facts (in WM)
  - Raining
  - Outside
Another example: Inference

- Execution cycle
  - 1st step:
    - Only one rule (R2) matches the facts
    - Add “State(Wet)” to the KB
    - Facts after the 1st cycle: Raining, Outside, State(Wet)
  - 2nd step:
    - Now R3 matches the facts
    - Add “Develop(Cold)”
    - Facts after the 2nd cycle: Raining, Outside, State(Wet), Develop(Cold)
Matching

In general, variables in a rule are matched when the rule is applied (i.e., variables stand for arbitrary objects)

The matching process consists of finding bindings for variables so that a rule can be applied
Matching

Bindings for $x$ is-a horse:
- $[x/\text{Comet}], [x/\text{Prancer}]$
- $[x/\text{Thunder}], [x/\text{Dasher}]$

Bindings for $y$ is fast:
- $[y/\text{Prancer}], [y/\text{Thunder}]$

Bindings for $x$ and $y$ in $x$-is-a-parent-of $y$:
- $[x/\text{Comet}; y/\text{Dasher}], [x/\text{Comet}; y/\text{Prancer}]; [x/\text{Dasher}; y/\text{Thunder}]

e.g. Comet is-a horse matches $x$ is-a-horse, but
Comet is-a lion would not match $x$ is-a horse

The given rule is applicable with bindings
- $[x/\text{Comet}; y/\text{Prancer}]$ and $[x/\text{Dasher}; y/\text{Thunder}]$

New facts that can be derived are
- Valuable(Comet) and Valuable(Dasher)
8 puzzle: initial WM

- **Lists**
  
  \((v_{11}, v_{12}, v_{13}, v_{21}, \ldots) : (1,2,3,0,5,6,4,7,8)\)

- **Predicate logic**
  
  \(\text{cell}(x, y, \text{value}) : \text{cell}(1,1,1), \ldots, \text{cell}(3,1,4), \ldots\)

- **Frames**

<table>
<thead>
<tr>
<th>Cell</th>
<th>IsA:</th>
<th>Possible values/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*x:</td>
<td></td>
<td>{1,2,3}</td>
</tr>
<tr>
<td>*y:</td>
<td></td>
<td>{1,2,3}</td>
</tr>
<tr>
<td>*value:</td>
<td></td>
<td>{1,2,..9}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cell31</th>
<th>InstanceOf:</th>
<th>Cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Possible values/Value</td>
<td></td>
</tr>
<tr>
<td>*x:</td>
<td></td>
<td>\3</td>
</tr>
<tr>
<td>*y:</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>*value:</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>
8 puzzle: final WM

- **Lists**
  
  (2,0,3,1,5,6,4,7,8)

- **Predicate logic**
  
  `cell(1,1,2), cell(1,2,0),...`

- **Frames**

<table>
<thead>
<tr>
<th><strong>Cell11</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceOf: <strong>Cell</strong></td>
</tr>
<tr>
<td><strong>Attribute</strong></td>
</tr>
<tr>
<td><em>x</em>:</td>
</tr>
<tr>
<td><em>y</em>:</td>
</tr>
<tr>
<td><em>value</em>:</td>
</tr>
</tbody>
</table>

i.e., the goal
8-puzzle RB (for lists)

- **Rule Base**

  ```
  if (0,x1,x2,x3,x4,x5,x6,x7,x8,x9) then (x1,0,x2,x3,x4,x5,x6,x7,x8,x9)
  ```

  ```
  if (0,x1,x2,x3,x4,x5,x6,x7,x8,x9) then (x3,x1,x2,0,x4,x5,x6,x7,x8,x9)
  ```

- **Problem**

  - We must define all possibilities for the empty-cell location and moves
8-puzzle RB (for predicate logic)

Rule Base

<table>
<thead>
<tr>
<th>Condition</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>if cell(x, y, 0), cell(x1, y, z), x = x1 + 1</td>
<td>then cell(x1, y, 0), cell(x, y, z), ~cell(x, y, 0), ~cell(x1, y, z)</td>
</tr>
<tr>
<td>if cell(x, y, 0), cell(x1, y, z), x = x1 - 1</td>
<td>then cell(x1, y, 0), cell(x, y, z), ~cell(x, y, 0), ~cell(x1, y, z)</td>
</tr>
<tr>
<td>if cell(x, y, 0), cell(x, y1, z), y = y1 + 1</td>
<td>then cell(x, y1, 0), cell(x, y, z), ~cell(x, y, 0), ~cell(x, y1, z)</td>
</tr>
<tr>
<td>if cell(x, y, 0), cell(x1, y, z), y = y1 - 1</td>
<td>then cell(x, y1, 0), cell(x, y, z), ~cell(x, y, 0), ~cell(x, y1, z)</td>
</tr>
</tbody>
</table>
Rule Base

if ?cell ← (cell (x ?x) (y ?y) (value 0))
   ?cell1 ← (cell (x ?x1) (y ?y) (value ?v))
   (test ?x=？x1+1)
then modify(?cell, value, ?v), modify(?cell1, value, 0)

if ?cell ← (cell (x ?x) (y ?y) (value 0))
   ?cell1 ← (cell (x ?x1) (y ?y) (value ?v))
   (test ?x=？x1-1)
then modify(?cell, value, ?v), modify(?cell1, value, 0)
Problems with recognize-act cycle

- Efficiency of matching
  - Naive approach
    - to find all applicable rules, try to match all elements of the working memory against all premises in all rules
    - unfortunately, n rules with m premises on average and k elements in the working memory give rise to $nmk$ possible matches in each cycle
  - Improvement
    - RETE algorithm
Problems with recognize-act cycle

- Rule selection
  - different selections may get to different outcomes
  - decision on which rule to apply is crucial
  - no general solution but some strategies for conflict resolution
    - e.g. most recently inferred antecedents, most specific rule, domain-specific preference ordering
Properties of rule-based systems

- Rules are very expressive
- Easy handling of certainty factors (e.g. for probabilistic reasoning)
- Lack of precise semantics of the rules
- Not always very efficient
Types of inference

<table>
<thead>
<tr>
<th>Directed by LHS (forward chaining)</th>
<th>Directed by RHS (backward chaining)</th>
</tr>
</thead>
<tbody>
<tr>
<td>![A][A\rightarrow B][B]</td>
<td>![B][A\rightarrow B][A]</td>
</tr>
</tbody>
</table>

It is raining
If it rains, the street is wet
Deduce that the street is wet

*Forward chaining (FC)*
... or *data-driven inference*

The street is wet
If it rains, the street is wet
Assume that it is raining

*Backward chaining*
... or *goal-driven inference*
Forward vs. backward chaining

- Forward chaining or *data-driven* inference works “forward”, from the initial state to the goal.
- Backward chaining or *goal-driven* inference works “backwards”, from the goal to the initial state.
Forward chaining (FC)

- Idea:
  fire any rule whose preconditions are satisfied in the KB, then add its conclusion to the KB until the goal is reached (or no further rules can be activated)
FC algorithm – abstract view

- Starts from an initial state and, when the LHS of a rule allows to activate it, performs the actions in the RHS, possibly updating the WM.
- The process continues until no more rules can be applied or some cycle limit is met.
FC algorithm - implementation view

defun PL-FC-ENTAILS? ( KB, query ) returns boolean

  count --- a dictionary, indexed by clause, initially # of premises
  inferred? --- a dict., indexed by symbol, each entry initially false
  agenda --- a list of symbols, initially the symbols known as true

while not agenda.isEmpty?() do
  p ← agenda.pop()
  if inferred?[p] then continue
  for each c in count.keywords() do
    if not PRE(c).contains(p) then continue
    count[c] ← count[c] - 1
    if count[c] > 0 then continue
    if HEAD(c) = query then return true
    agenda.push(HEAD(c))
  return false

Forward chaining is sound and complete for Horn KB!
Forward chaining: an example
Forward chaining: an example

- In the given example, no more rules apply.
- The inference chain is:

```
A → D → F → Z
```

Problems with forward chaining

- Many rules could be applicable
- The whole process is not directed towards the goal
Forward chaining: another example

- Another example

\[
P \Rightarrow Q \\
L \land M \Rightarrow P \\
B \land L \Rightarrow M \\
A \land P \Rightarrow L \\
A \land B \Rightarrow L \\
A \\
B
\]
Forward chaining: another example
Forward chaining: another example
Forward chaining: another example
Forward chaining: another example
Forward chaining: another example
Forward chaining: another example
Forward chaining: another example
Forward chaining: another example
Backward chaining – abstract view

- Idea:
  to prove \( q \) check whether \( q \) is already known, or prove (by BC) all premises of some rule concluding \( q \)
Backward chaining - implem. view

- Looks in the WM to see if the goal is already there
- If not, look up for rules that will establish the goal in their RHS and set up subgoals for achieving the premises specified in their LHS
- Continue till some rule can be applied to achieve goal state
Backward chaining

- **Avoid loops**
  - check whether a subgoal is already on the goal stack

- **Avoid repeated work**
  - check whether a subgoal has already been proved true, or has already failed
Backward chaining

- **Advantages**
  - The search process is driven by the goal

- **Disadvantages**
  - The goal has to be known
Backward chaining: an example
Backward chaining: an example
Backward chaining: an example
Backward chaining: an example
Backward chaining: an example
Backward chaining: an example
Backward chaining: an example
Backward chaining: an example
Backward chaining: an example
Backward chaining: an example
Backward chaining: an example
Forward vs. backward chaining

- **Forward chaining**
  - data-driven, automatic, unconscious processing (e.g., object recognition, routine decisions)
  - may perform a lot of work actually irrelevant to the goal

- **Backward chaining**
  - goal-driven, appropriate for problem-solving (e.g., Where are my keys? How do I get into a PhD program?)
  - the complexity of BC can be much less than linear in the size of the KB
Forward or backward reasoning?

- **Four major factors:**
  - Greater number of possible start states or goal states?
  - Move from smaller set to the larger set
  - Prefer direction that corresponds more closely with the way users think
  - If program has to justify (explain) its reasoning

- **What kind of events trigger problem solving?**
  - Forward chaining is more appropriate upon a new fact arrival
  - Backward chaining is more appropriate if a query must be answered
Forward or backward reasoning?

- Add fact State(Dry) to the "raining" example
  - Initial facts: State(Dry), Raining, Outside
  - Result of inference: State(Dry), Raining, Outside, State(Wet), Develop(Cold) Inconsistent!

- How can we check that with thousands of rules and facts?
  - Use a truth maintenance system
Forward or backward reasoning?

- Which global strategy to apply? FB or BC?
- Which local strategy to apply? i.e. which rule to select in recognize-act cycle
- Complexity of matching
Summary

➢ There is no single most adequate knowledge representation algorithm for everything
  ◆ The choice should be based on what has to be represented and how the knowledge must be processed (inference)

➢ There are many more representation formalisms:
  ◆ All above mentioned are symbolic
  ◆ Non-symbolic: e.g. neural nets work on non-symbolic representations