Rule-based Systems

Big idea: Separation of Knowledge and Control

- Rules
- Forward Rule Systems
- Goal-directed Rule Systems
Rules on NEAR and Messenger: A Simple Reflex System

Applying Rules to the NEAR Spacecraft

- Sample NEAR rule
  - Symptom:
    - (Charger current > 0.8 A) for 10 sec
  - Recovery:
    - Switch to the redundant charger and disengage the primary.
Rules on Cassini: A State Centered Approach

- Deduction style

<table>
<thead>
<tr>
<th>IF</th>
<th>A1</th>
<th>Antecedent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A2</td>
<td></td>
</tr>
<tr>
<td>THEN</td>
<td>B1</td>
<td>Consequent</td>
</tr>
<tr>
<td></td>
<td>B2</td>
<td></td>
</tr>
</tbody>
</table>

- Each rule should *capture a single independent inference and should make sense by itself*

- Example:
  
  IF (is-a-parent-of ?x ?y)
  (is-a-parent-of ?y ?z)
  THEN (is-a-grand-parent-of ?x ?z)

- Variables of the same name must have the same value

Rule Based Systems

- Big idea: Separation of Knowledge and Control

- Rules
  - Forward Rule Systems
    - Forward chaining
    - Production rules
    - Goal-directed Rule Systems

Rule Based Architecture

- Programs: Knowledge is lost within the control structures

- Rule-based Architectures: Separate knowledge and control
  - Knowledge: rules & assertions
Rule Based Architecture

- Programs: Knowledge is lost within the control structures
- Rule-based Architectures: Separate knowledge and control
  - Knowledge: rules & assertions
  - Control: rule interpreter
• Programs: Knowledge is lost within the control structures

• Rule-based Architectures: Separate knowledge and control
  • Knowledge: rules & assertions
  • Control: rule interpreter
Rule Based Architecture

- Programs: Knowledge is lost within the control structures
- Rule-based Architectures: Separate knowledge and control
  - Knowledge: rules & assertions
  - Control: rule interpreter & conflict resolution strategy

Rule Based Systems

- Big idea: Separation of Knowledge and Control

- Rules
- Forward Rule Systems
  - Forward chaining
  - Production rules
- Goal-directed Rule Systems
**Rule Interpreter (forward chaining)**

- Identify triggered rules, creating rule instances

- Choose one rule instance (called conflict resolution)

- Fire the chosen rule instance

- Terminate when no triggered rules or special stop assertion

---

**Rule Interpreter (forward chaining)**

- Identify triggered rules, creating rule instances
  - triggered rule:
    - antecedents in assertions database

- rule instance: rule with variables filled in
Rule Interpreter (forward chaining)

- Identify triggered rules, creating rule instances
  - triggered rule:
    - antecedents in assertions database

- rule instance: rule with variables filled in
- Choose one rule instance (called conflict resolution)
  - Conflict resolution strategies
    e.g. first instance, random instance, etc.

Firing the chosen rule instance
- Firing a rule instance means performing the actions indicated by the rule
  - (e.g. adding a new assertion)
Rule Interpreter (forward chaining)

- Identify triggered rules, creating rule instances
  - triggered rule:
    - antecedents in assertions database
      - can change the database (e.g. add new assertion)
  - rule instance: rule with variables filled in
- Choose one rule instance (called conflict resolution)
  - Conflict resolution strategies
    e.g. first instance, random instance, etc.
- Fire the chosen rule instance
  - Firing a rule instance means performing the actions indicated by the rule
    - (e.g. adding a new assertion)
- Terminate when no triggered rules or special stop assertion

Forward chaining - strep throat
(using rule order for conflict resolution)

R1: if (signs (throat infec)) (org streptococcus) then (strep throat)
R2: if (red throat) then (signs (throat infec))
R3: if (sore throat) then (signs (throat infec))
R4: if (org stain gram-pos) (org morph coccus) (org growth chains) then (org streptococcus)

Assertions:
(sore throat)
(org stain gram-pos)
(org morph coccus)
(org growth chains)
Forward chaining - strep throat
(using rule order for conflict resolution)

R1: if (signs (throat infec)) (org streptococcus) then (strep throat)
R2: if (red throat) then (signs (throat infec))
R3: if (sore throat) then (signs (throat infec))
R4: if (org stain gram-pos) (org morph coccus) (org growth chains) then (org streptococcus)

Assertions:
(sore throat)
(org stain gram-pos)
(org morph coccus)
(org growth chains)
R3 fires: (signs (throat infec))
R4 fires: (org streptococcus)
### Forward chaining - strep throat
(Using rule order for conflict resolution)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>If (\text{signs (throat infec)}) (\text{org streptococcus}) then (\text{strep throat})</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>If (\text{red throat}) then (\text{signs (throat infec)})</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>If (\text{sore throat}) then (\text{signs (throat infec)})</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>If (\text{org stain gram-pos}) (\text{org morph coccus}) (\text{org growth chains}) then (\text{org streptococcus})</td>
<td></td>
</tr>
</tbody>
</table>

**Assertions:**
- \(\text{sore throat}\)
- \(\text{org stain gram-pos}\)
- \(\text{org morph coccus}\)
- \(\text{org growth chains}\)

**Fires:**
- \(\text{R3 fires: (signs (throat infec))}\)
- \(\text{R4 fires: (org streptococcus)}\)
- \(\text{R1 fires: (strep throat)}\)

**Done.**
Forward Chaining with variables


Forward Chaining with variables


(x:A y:B) (x:A y:C) (x:B y:D) (x:C y:E)
Forward Chaining with variables

THEN (parent A B)          (parent A C) (parent B D) (parent C E)

(p x ?y) (parent A C) (grand-parent ?x ?z)
(x:A y:B)      (parent A z:B)

(x:A y:C)

(x:B y:D)
(x:C y:E)

Forward Chaining with variables

THEN (parent A B)          (parent A C) (parent B D) (parent C E)

(p x ?y) (parent y ?z) (grand-parent ?x ?z)
(x:A y:B)      (y:A z:B) conflict in y

(x:A y:C)

(x:B y:D)
(x:C y:E)
Forward Chaining with variables

<table>
<thead>
<tr>
<th>IF</th>
<th>(parent ?x ?y)</th>
<th>( (\text{parent } A \ B) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(parent ?y ?z)</td>
<td>( (\text{parent } A \ C) )</td>
</tr>
<tr>
<td>THEN</td>
<td>(grand-parent ?x ?z)</td>
<td>( (\text{parent } B \ D) )</td>
</tr>
<tr>
<td></td>
<td>( (\text{parent } C \ E) )</td>
<td></td>
</tr>
</tbody>
</table>

\((\text{parent } ?x ?y)\) \((\text{parent } ?y ?z)\) \((\text{grand-parent } ?x ?z)\)

\((x:A \ y:B)\) \(\rightarrow\) \((y:A \ z:B)\) \(\rightarrow\) conflict in \(y\)

\((y:A \ z:C)\) conflict in \(y\)

\((x:A \ y:C)\)

\((x:B \ y:D)\)

\((x:C \ y:E)\)
Forward Chaining with variables


(x:A y:B) (y:A z:B) conflict in y (y:A z:C) conflict in y (y:B z:D)

(x:A y:C)

(x:B y:D)

(x:C y:E)
Forward Chaining with variables

IF (parent ?x ?y)
   (parent ?y ?z)
THEN (grand-parent ?x ?z)

(x:A y:B)
   (y:A z:B)
   (y:A z:C)
   (y:B z:D)
   (y:C z:E)

(x:A y:C)

(x:B y:D)

(x:C y:E)

Forward Chaining with variables

IF (parent ?x ?y)
   (parent ?y ?z)
THEN (grand-parent ?x ?z)

(x:A y:B)
   (y:A z:B)
   (y:A z:C)
   (y:B z:D)
   (y:C z:E)

(x:A y:C)

(x:B y:D)

(x:C y:E)
Forward Chaining with variables


(x:A y:B) (y:A z:B) conflict in y
(y:A z:C) conflict in y
(y:B z:D) (x:A y:B z:D)
(y:C z:E) conflict in y

(x:A y:C) (y:A z:B) conflict in y
(y:A z:C) conflict in y
(y:B z:D) conflict in y
(y:C z:E) (x:A y:C z:E)

(x:B y:D) (y:A z:B) conflict in y
(y:A z:C) conflict in y
(y:B z:D) conflict in y
(y:C z:E) conflict in y

(x:C y:E)
Forward Chaining with variables


IF (parent A B) (parent A C) THEN (parent B D) (parent C E)

Goal: Find all variable bindings that make each rule antecedent match an entry in the database:

• For every variable binding of the first antecedent,
  • find all bindings of the second antecedent,
  • check for consistency of shared variables.
    – For the third antecedent,
      – repeat for each combination of valid bindings of the first and second antecedents.
    » And so forth…
Implementing forward chaining efficiently

- grandparent example:
  - 1 rule with 2 antecedents
  - 4 assertions in knowledge base
    ➢ $4 \times 4 = 16$ matching operations.

- In general:
  - 1 rule with $K$ antecedents
  - $N$ assertions,
    ➢ worst-case matching grows as $N^K$

➢ Practical systems use the RETE algorithm.

Rule Based Systems

- Big idea: Separation of Knowledge and Control

- Rules
- Forward Rule Systems
  - Forward chaining
  - Production rules

- Goal-directed Rule Systems
Production Rules

- Deduction rules only add new assertions to the database.
- Production rules can also
  - delete assertions,
  - have side-effects (printing, executing programs, etc).

Example:

```
IF     (at ?x ?y)
   (move ?x to ?z)
ADD    (at ?x ?z)
DELETE (move ?x to ?z)
   (at ?x ?y)
```

- Initial database:
  - (at A room1) (move A room2)
- Rule Adds: (at A room2)
- Rule Deletes: (at A room1) (move A room2)

Production rules are very sensitive to order of execution, so conflict resolution strategy is crucial.

Some Conflict Resolution Strategies

- **First** - Choose an instance of the rule that occurs earliest in the list of rules.
- **Random** - Choose randomly among all the triggered instances. Gives every rule a chance to fire.
- **Least recently fired** - Choose instance of the rule that was least recently fired. Gives every rule a chance to fire.
- **Most specific** – For backward chaining …choose instance of the rule with the most antecedent conditions, since that suggests the rule is more specific to the current state of the database.
- Combinations of these are also possible, e.g. most specific and least-recently fired.
- **Beware**: Conflict resolution strategies can destroy modularity!
Conflict Resolution for Deduction Rules

• For deduction rule sets that only add assertions, conflict resolution is irrelevant
  • all triggered rule instances will eventually fire, and all be added to the database.
• But, it is possible to write deduction rules that create “infinite loops”.
  • IF ?x THEN (not (not ?x))
    If this rule is triggered once, it will keep triggering…
• A conflict resolution strategy can
  – Give other triggered rules a chance to fire
  – Terminate rule firing.

Rule Based Systems

• Big idea: Separation of Knowledge and Control

• Rules
• Forward Rule Systems
  • Goal-directed Rule Systems
Goal-directed, Backward Chaining

- Start with a possible conclusion and check whether the assertions, together with the rules, justify that conclusion.
- Start with a query, and apply rules that derive the answer to that query.

Backward Chaining
- conclusions -> assertions
- Focuses derivation, working back from one desired conclusion.
- Can ask questions when a relevant fact cannot be derived.

Forward Chaining
- assertions -> conclusions
- Derives all conclusions from the rules and assertions.
- Assertions must be known at the start of chaining.

strep throat expert: What the doctor sees

Questions:
2. Sore throat? Yes
3. Stain gram-pos? Yes
4. Morphology coccus? Yes
5. Growth chains? Yes
strep throat expert: What the “expert” decides

(strep throat)

AND

(signs (throat infec))

(sore throat)

(org streptococcus)

AND

(red throat)

(org stain gram-pos)

(org growth chains)

Rule Based Systems

• Big idea: Separation of Knowledge and Control

• Rules
  • Forward Rule Systems
  • Goal-directed Rule Systems
    • Backward Chaining
    • Unification
Backward chaining - strep throat

R1: if (signs (throat infec)) (org streptococcus) then (strep throat)
R2: if (red throat) then (signs (throat infec))
R3: if (sore throat) then (signs (throat infec))
R4: if (org stain gram-pos) (org morph coccus) (org growth chains) then (org streptococcus)

AND-OR Tree for (strep throat) goal

Questions:
Questions:
2. Sore throat? Yes
3. Stain gram-pos? Yes
4. Morphology coccus? Yes
5. Growth chains? Yes
**Backchain(G) - Rule interpreter**

If $G$ matched assertion in database, then return True

Else

If there are rules whose consequent matches $G$

   For each matching rule $R$

      For each antecedent $C$ of $R$

         If Backchain($C$) is true, proceed

         Else go to next rule.

      Return True /* when all antecedents are true */

   Return False /* when no matching rule succeeds */

Else, ask user if $G$ is true.

   If answer is "yes" return True,

   Else return False.

---

**Rule Based Systems**

- Big idea: Separation of Knowledge and Control
- Rules
- Forward Rule Systems
- Goal-directed Rule Systems
  - Backward Chaining
  - Unification
**Unification**

**Forward-chaining:**
- Assertions (no variables) match rule antecedents (patterns with variables)

\[ (\text{sore throat}) \]

R3: if (sore ?x) then (signs (?x infec))

**Backward-chaining:**
- (sub)Goals (patterns with variables) match Rule consequents (patterns with variables)

\[(\text{grandparent A ?g})\]
\[(\text{grandparent ?x ?z})\]
\[R1\]
\[(\text{parent ?x ?y}) \quad (\text{parent ?y ?z})\]

---

**Unification**

- **Problem**: Find bindings for the variables in two patterns so that the patterns become identical.
- **Example**:
  - Unify \((\text{grandparent ?x ?z}) \& (\text{grandparent A ?g})\) ➔ Binding: \{?x = A, ?z = ?g\}
  - Binding ?z = ?g
  - Indicates that both variables must ultimately be assigned the same value.
  - Value not known at time of binding.
Backward chaining with variables

R1: if (parent ?x ?y) (parent ?y ?z) then (grandparent ?x ?z)
R2: if (father ?v2 ?w2) then (parent ?v2 ?w2)
R3: if (mother ?v3 ?w3) then (parent ?v3 ?w3)

Assertions:
(father A B)
(mother B C)

Backward chaining with variables

R1: if (parent ?x ?y) (parent ?y ?z) then (grandparent ?x ?z)
R2: if (father ?v2 ?w2) then (parent ?v2 ?w2)
R3: if (mother ?v3 ?w3) then (parent ?v3 ?w3)

Assertions:
(father A B)
(mother B C)
**Issue 1: Avoiding variable name conflicts**

Consider subgoal and rule

\[
\text{foo} \ ?x \quad \text{IF} \quad (\text{bar} \ ?x \ ?y) \\
\text{THEN} \quad (\text{foo} \ ?y)
\]

- Backward chaining unifies:
  \[
  (\text{foo} \ ?x) \quad \text{with} \quad (\text{foo} \ ?y) \\
  \Rightarrow \{?y = ?x\}
  \]
- Creates new subgoal:
  \[
  (\text{bar} \ ?x \ ?x)
  \]
  - Matches assertions like \( (\text{bar} \ A \ A) \)
  - Doesn’t match assertions like \( (\text{bar} \ A \ B) \)

- **Problem:** Occurrence of \(?x\) in subgoal and rule accidental
- **Solution:** Give variables *unique* names in rules
**Issue 2: Occurs check**

Consider the unifier for:

\[
\begin{align*}
& \text{likes } ?z \ ?z) \quad \text{(likes } ?y \text{ (house-of } ?x)) \\
\Rightarrow & \{ ?z = ?y, ?z = \text{(house-of } ?x) \}, \\
& \text{hence } ?y = \text{(house-of } ?x), \text{ which is consistent.}
\end{align*}
\]

Consider the unifier for:

\[
\begin{align*}
& \text{likes } ?z \ ?z) \quad \text{which is problematic} \\
& \text{(likes } ?x \text{ (house-of } ?x)) \\
\Rightarrow & \{ ?z = ?x, ?z = \text{(house-of } ?x) \} \\
& \text{hence } ?x = \text{(house-of } ?x), \text{ which is problematic.}
\end{align*}
\]

**Solution:** In unification, a variable cannot be assigned to an expression where that variable occurs.

---

**Rule Based Architecture**

- **Assertion & Rules**: Knowledge and control components
- **Rule Interpreter**: Matches antecedents, adds consequent, picks rule
- **Conflict Resolution Strategy**: Core computation is unification

**Approaches**:
- **Forward Chaining**: Deductive and Response Systems
- **Backward Chaining**: Goal-driven and Query Systems
- **Rule-based Architectures**: Separate knowledge and control
  - Knowledge: rules & assertions
  - Control: rule interpreter & conflict resolution strategy

**Programs**: Knowledge is lost within the control structures
Rule Based Systems

• Big idea: Separation of Knowledge and Control

• Rules
  • Forward Rule Systems
    • Forward Chaining
    • Production Systems

• Goal-directed Rule Systems
  • Backward Chaining
  • Unification