Pattern-directed Inference Systems

Pattern-directed Inference Systems
Pattern-directed modules (PDMs) operate on data structures via matching and controlled by a central executive unit.

Rule-based Systems
Consist of PDMs in rule format. Matching typically with left-hand side, operations on right-hand side.

Network-based Systems
PDMs are located at nodes of a network and activated by signals received on incoming arcs.

Production Systems
Rule-based systems with central control for matching and scheduling.

Transformation Systems
Rule-based systems without integrated matching and control.

Antecedent-driven Systems
Production systems where rule selection is controlled by the antecedent part of rules.

Consequent-driven Systems
Production systems where rule selection is controlled by the consequent part of rules.

Logical Systems
Transformation systems for defining and processing grammatical structures, e.g. parsing.

Grammatical Systems
Transformation systems applied to problems in formal logics, e.g. theorem proving.

Rules for Knowledge Representation

Rule-based knowledge representation is useful for specifying inference steps in a declarative way.

Example (scene interpretation):

If (region.color = green) and (region.location = picture-bottom) then (region.type = grass)

Rules may express different types of reasoning:

- premise ➔ conclusion ➔ logical implication
- antecedence ➔ consequence ➔ inference from given preconditions
- evidence ➔ hypothesis ➔ interpretation of facts
- situation ➔ action ➔ situated behaviour
- IF ➔ THEN ➔ informal paraphrase
- left-side ➔ right-side ➔ can mean anything

Rules typically refer to a frame-based knowledge base.
### Rule-based Programming Language

**Experimental AI programming language PLANNER (Hewitt, 1972)**

Rules have format:

- **IF GIVEN** <extensional data> **THEN CONCLUDE** <intensional data>
- **IF WANTED** <intensional data> **THEN FIND** <extensional data>

<table>
<thead>
<tr>
<th>Data</th>
<th>Consequent Theorems</th>
</tr>
</thead>
<tbody>
<tr>
<td>(THASSERT (IN BIRD CAGE))</td>
<td>(THCONSE (X Y Z) (IN ?X ?Y))</td>
</tr>
<tr>
<td>(THASSERT (IN TABLE ROOM))</td>
<td>(THGOAL (IN IX ?Z) (THUSE NIL))</td>
</tr>
<tr>
<td>(THASSERT (IN CHAIR ROOM))</td>
<td>(THGOAL (IN !Z !Y))</td>
</tr>
<tr>
<td>(THASSERT (IN FLOWER VASE))</td>
<td>(THGOAL (IN IX ?Z))</td>
</tr>
<tr>
<td>(THASSERT (ON CAGE TABLE))</td>
<td>(THGOAL (IN !Z !Y))</td>
</tr>
<tr>
<td>(THASSERT (ON VASE TABLE))</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query</th>
<th>The intensional data (IN FLOWER ROOM) is derived with the help of consequent theorems</th>
</tr>
</thead>
<tbody>
<tr>
<td>(THGOAL (IN FLOWER ROOM))</td>
<td></td>
</tr>
</tbody>
</table>

### Rule-based Expert Systems

**Developed 1970 - 1985 to**

- collect and preserve expert knowledge
- replace human experts by computer programs
- to automatically derive interesting knowledge.

**Basic idea:** Represent expert knowledge in terms of IF-THEN rules

**Basic structure:**

- **INFERENCE ENGINE**
- **KNOWLEDGE BASE**
  - facts
  - rules
- **USER INTERFACE**
  - knowledge acquisition
  - explanation
  - dialogue
Recognize-and-act Cycle

Determine applicable rules by matching the antecedent part (in case of forward-chaining) or the consequent part (in case of backward chaining) with data objects.

If more than one rule is applicable, invoke conflict resolution to select rule.

Forward and Backward Chaining

Rule systems may support forward and/or backward inferencing
Processing Steps of Recognize-and act Cycle

**Forward Chaining:**
Repeat until all goals have been derived:
- Determine rules which can be applied based on available facts
- Select one of those rules
- Apply rule, establish new facts

**Backward Chaining:**
Repeat until all goals have been derived:
- Determine rules which can be used to derive a goal
- Select one of those rules
- Apply rule, establish unsatisfied conditions as new goals

Knowledge-based Diagnosis of a Car Problem

**Regel 1** Wenn (Anlasser arbeitet normal) dann (Batterie ok)
**Regel 2** Wenn (Batterie OK) und (Wert Tankuhr > 0) und (Benzinfilter sauber) dann (Problem = Zündanlage)
**Regel 3** Wenn (Batterie OK) und (Wert Tankuhr > 0) und (nicht Benzinfilter sauber) dann (Defekt = Benzinzuleitung)
**Regel 4** Wenn (nicht Scheibenwischer OK) und (nicht Licht OK) dann (Defekt = Batterie leer)
**Regel 5** Wenn (nicht Wert Tankuhr > 0) dann (Defekt = Tank leer)
**Regel 6** Wenn (Problem = Zündanlage) und (Verteilerdose OK) dann (Defekt = Zündspule)

**Fakten des aktuellen Problems**
- Anlasser arbeitet normal
- Scheibenwischer OK
- Licht OK
- Wert Tankuhr > 0
- Benzinfilter sauber
- Verteilerdose OK

**Abgeleitete Fakten**
- Batterie OK
- Problem = Zündanlage
- Defekt = Zündspule
Rule Selection

The order of execution cannot be completely controlled in a rule system. It is expected that the user abstracts from individual inference steps.

Rules are selected in a recognize-and-act cycle. If more than one rule can be applied, a "conflict resolution" process decides.

Conflict resolution strategies available in a typical rule system:

- prefer old facts (goals)  breadth-first search
- prefer new facts (goals)  depth-first search
- prefer more special rule  more special = more conditions
- prioritize rules  e.g. by memory order (PROLOG)
- use meta-rules  rules about rule selection

Conflict Resolution with Meta-rules (1)

Expert system for chemical spill treatment may have rules:
R1: If spill is sulfuric acid, apply treatment A
R2: If spill is acid, apply treatment B

Forward chaining may generate conflict set (R1, R2).

Knowledge base may contain following facts:
- treatment A is expensive, treatment B is cheap.
- treatment A is not dangerous, treatment B is dangerous
- R1 has been entered by expert Miller, R2 by novice Johnson

How can rule selection be controlled in a reasonable way?
Conflict Resolution with Meta-rules (2)

Meta-rules for conflict resolution:
R3: Prefer rules with less expensive treatment
R4: Prefer rules with less dangerous treatment
R5: Prefer rules entered by experts before rules entered by novices

Rules R4 and R5 recommend: R1 before R2
Rule 3 recommends: R2 before R1

\{R3, R4, R5\} is a meta-conflict set.

Meta-meta-rule for meta-conflict resolution:
R6: Prefer meta-rules entered by experts before meta-rules entered by novices.

In practical systems, one rarely needs more than 2 meta levels.

Conflict Resolution by Prioritizing

Total order:
R1 < R2 < ... < RN

Examples: • order by storage
• order by indexing

Partial order:
Ri < Rk, Rm < Rn, ...

Example: Rules are structured as a rule tree

```
   R11
  /   \
R21  R22
 /   /   \
R31  R32  R33
      /   \
     R34  R35
```
Conflict Resolution Based on Specialization Relations

Prefer most special rule

1. Compare non-instantiated rules
   A rule R1 is more special than R2 if
   • R1 has at least as many premises as R2
   • each premise in R2 subsumes at least one premise in R1
   • R1 and R2 are not identical

   Example:
   A, B, C, ... attributes
   a, b, c, ... constants
   X, Y, Z, ... variables

   R1: \{[A a] [B e] [C X] [D Y] \Rightarrow ...\}
   R2: \{[A X] [B e] [D Y] \Rightarrow ...\}

2. Comparison of instantiated rules
   Analogous to 1), however no subsumption test for variables required

Conflict Resolution Based on Data Seniority

Data may get time stamp from inference cycle.

• Prioritizing most recent data
  Prefer rules whose instantiation involves recently generated data
  \Rightarrow work on new facts first

• Prioritize least oldest data
  Prefer rules whose instantiation has younger elements than the oldest element of other rules
  \Rightarrow prefer rules which use the youngest facts

• Avoid rule repetition
• Avoid repeated instantiation
The Rule System OPS5

OPS5 ("Official Production System, Version 5")
• developed at CMU 1980 ...
• implementation language for successful expert systems
  (XCON, XSEL a.o.)

CLIPS
• reimplementation of OPS5 in C for NASA
• freeware

JESS
• reimplementation of OPS5 in Java
• freeware

Rules in OPS5

Syntax of a rule in OPS5:
<rule>::= [P <rule-name> <antecedent> --> <consequent>]
<antecedent>::= (<condition>)
<condition>::= <pattern> I - <pattern>
<pattern>::= [object] {[^attribute] <value>}
$consequent>::= {<action>}
<action>::= [MAKE object] {[^attribute] <value>} I
[MODIFY pattern-number] {[^attribute] <value>} I
[REMOVE pattern-number] I
[WRITE <value>]

Example: "If there are 2 disks close to each other and with equal size, make them a wheel pair"
[P find-wheel-pair [disk ^location <x1> ^size <y>]
[disk ^location |<x2> - <x1>| < 10 ^size <y>]] --- ]

• depth-first search
• limited expressiveness for constraints
RETE Algorithm in OPS5 (1)

A naive Implementation of the recognize-act cycle leads to unacceptably poor runtime performance except for small knowledge bases.

Improving efficiency:
1. **Pattern matching only for a small section of working memory (WM)**
   Rule applications usually lead to small changes of the WM and the conflict set does not change drastically.
   => Store pattern matching results and check only changed WM elements in next recognize-act cycle.
2. **Identical premises occurring in multiple rules must only be evaluated once**
   Premises of different rules often share common conditions,
   => Analyze rules for common premises and optimal order of premise evaluation.

RETE Algorithm in OPS5 (2)

Construct a net [lat. rete] from the rules which holds code for premise evaluation at its nodes.

**Input:** Changes of WM data

**Output:** New conflict set

Net consists of unary nodes coding a condition on a single WM element, and binary nodes for relations between unary nodes.

Example of unary node:

```
[cube  ^colour red  ^weight < 100]
```

A unary node receives WM elements marked for adding or deleting as input, and delivers as output elements which satisfy the conditions.
**RETE Algorithm in OPS5 (3)**

Single element conditions may be combined by binary nodes.

Binary nodes store the WM elements received via the two input lines to generate all possible combinations (cross product).

The output of the last node representing a rule consists of tuples of WM elements satisfying the rule.

**RETE Algorithm in OPS5 (4)**

Rules may be merged as long as their initial parts coincide.

Order of premises influences effectiveness of rule merging.
Example for RETE Algorithm (1)

A rule:  
\[
(P \text{ search-pyramid} \\
[\text{cube } \langle \text{name c} < \text{cube1} \rangle \langle \text{on} \text{ table} \rangle \\
[\text{brick } \langle \text{weight >} 200 \rangle \\
[\text{pyramid } \langle \text{colour <yellow white> \rangle \langle \text{on} \text{ cube1 \langle \text{weight < 200} \rangle]} \\
-> \langle \text{action part} \rangle
\]

Previous contents of WM:

<table>
<thead>
<tr>
<th>time stamp</th>
<th>class</th>
<th>name</th>
<th>colour</th>
<th>weight</th>
<th>on</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cube</td>
<td>C1</td>
<td>blue</td>
<td>250</td>
<td>table</td>
</tr>
<tr>
<td>4</td>
<td>cube</td>
<td>C2</td>
<td>red</td>
<td>100</td>
<td>table</td>
</tr>
<tr>
<td>6</td>
<td>pyramid</td>
<td>P1</td>
<td>yellow</td>
<td>120</td>
<td>table</td>
</tr>
<tr>
<td>9</td>
<td>pyramid</td>
<td>P2</td>
<td>white</td>
<td>NIL</td>
<td>C1</td>
</tr>
<tr>
<td>12</td>
<td>brick</td>
<td>B1</td>
<td>blue</td>
<td>300</td>
<td>table</td>
</tr>
</tbody>
</table>

New data entered into WM:  
[15, brick, B2, blue, 280, NIL]

How does the RETE net compute changes of the conflict set?

The Expert Configuration System

XCON

XCON has been developed in the early 80’s at CMU using OPS5. The task of XCON was to configure computer systems by Digital Equipment Company. XCON was the first commercially successful expert system.

incomplete component list based on customer wishes

XCON

complete and consistent component list, floor plan for cabinets, slot plan for components

History:

- 1982 start of operations with 1000 rules, 7 min per configuration
- XCON earned money by avoiding configuration errors and delayed customer payment
- 1988 more than 10000 rules:
  - average of 6 conditions per rule
  - average of 5 tests per condition
  - average of 4 actions per rule
Typical Rule of XCON

(paraphrase, not in OPS5 rule language)

IF:  
The most current active context is assigning a power supply  
and an SBI module of any type has been put in a cabinet  
and the position it occupies in the cabinet is known  
and there is space in the cabinet for a power supply  
and there is no available power supply  
and the voltage and frequency of the components is known

THEN:  
find a power supply of that voltage and frequency  
and add it to the order

Input of XCON

Typical component list based on customer wishes:

COMPONENTS ORDERED:
1  SV-AXMMA-LA  [packaged system]
1  FP780-AA  [floating point accelerator]
1  DW780-AA  [unibus adaptor]
1  BA11-KE  [unibus expansion cabinet box]
6  MS780-DC  [memory]
1  MS780-CA  [memory controller]
1  H9002-HA  [cpu expansion cabinet]
1  H7111-A  [clock battery backup]
1  H7112-A  [memory battery backup]
1  REP05-AA  [single port disk drive]
4  RP05-BA  [dual port disk drive]
1  TEE16-AE  [tape drive with formatter]
2  TE16-AE  [tape drive]
8  RK07-EA  [single port disk drive]
1  DR11-B  [direct memory access interface]
1  LP11-CA  [line printer]
1  DZ11-F  [multiplexer with panel]
1  DZ11-B  [multiplexer]
2  LA36-CE  [hard copy terminal]
Example of a Configuration Run (1)

Numbers correspond to rule applications, lines show context transitions

1. MAJOR-SUBTASK-TRANSITION
2. SET-UP
3. UNBUNDLE-COMPONENTS
53. NOTE-CUSTOMER-GENERATED-EXCEPTION
56. NOTE-UNSUPPORTED-COMPONENTS
57. CHECK-VOLTAGE-AND-FREQUENCY
104. CHECK-FOR-TYPE-OR-CLASS-CHANGES
110. VERIFY-SBI-AND-MB-DEVICE-ADEQUACY
111. COUNT-SBI-MODULES-AND-MB-DEVICES
126. GET-NUMBER-OF-BYTES-AND-COUNT-controllers
137. FIND-UBA-HBA-CAPACITY-AND-USE
146. VERIFY-MEMORY-ADEQUACY
148. PARTITION-MEMORY
160. ASSIGN-UB-MODULES-EXCEPT-THOSE-CONNECTING-TO-PANELS
177. VERIFY-UB-MODULES-FOR-DEVICES-CONNECTING-TO-PANELS
178. FIND-ATTRIBUTE-OF-TYPE-IN-SYSTEM
180. VERIFY-COMPONENT-OF-SYSTEM
207. NOTE-POSSIBLY-FORGOTTEN-COMPONENTS
213. CHECK-FOR-MISSING-ESSENTIAL-COMPONENTS
215. MAJOR-SUBTASK-TRANSITION
216. DELETE-UNNEEDED-ELEMENTS-FROM-WM
236. FILL-CPU-OR-CPUX-CABINET
240. ADD-UBAS
248. ASSIGN-POWER-SUPPLY

Example of a Configuration Run (2)

251. ADD-MBAS
252. DISTRIBUTE-MB-DEVICES
260. ASSIGN-SLAVES-TO-MASTERS
269. ASSIGN-POWER-SUPPLY
272. fill-MEMORY-SLOTS
278. SHIFT-BOARDS
298. add-MEMORY-MODULE-SIMULATORS
306. ASSIGN-POWER-SUPPLY
312. fill-CPU-SLOTS
318. ASSIGN-POWER-SUPPLY
322. ADD-NECESSARY-SIMULATORS
326. DELETE-TEMPLATES
340. DELETE-UNNEEDED-ELEMENTS-FROM-WM
353. fill-CPU-OR-CPUX-CABINET
356. ADD-MBAS
359. ASSIGN-POWER-SUPPLY
382. ADD-UBAS
384. fill-MEMORY-SLOTS
388. SHIFT-BOARDS
389. add-MEMORY-MODULES-SIMULATORS
398. ASSIGN-POWER-SUPPLY
399. TERMINATE-SBI
402. ADD-NECESSARY-SIMULATORS
406. DELETE-TEMPLATES
415. MAJOR-SUBTASK-TRANSITION
417. GENERATE-OPTIMAL-SEQUENCE
Example of a Configuration Run (3)

436. ASSIGN-UBAS-TO-BOXES-TO-CABINETS
438. ASSIGN-UBAS-TO-BOXES
441. ATTRIBUTE-BOXES-AMONG-CABINETS
442. SET-UP-FOR-BOX-ASSIGNMENTS
446. ASSIGN-BOXES-TO-CABINETS
452. COMPUTE-DISTANCES-FROM-UBAS-TO-BOXES
458. SET-SEQUENCING-MODE
462. FILL-BOXES
465. FILL-HALF-BOXES
468. SELECT-BOX-AND-UB-MODULE-FOR-NEXT-SU
470. ASSIGN-BACKPLANE-TO-BOX
474. GENERATE-SLOT-TEMPLATES
478. PUT-UB-MODULE
482. LEAVE-BACKPLANE
485. AUGMENT-UB-LENGTH
488. GET-UB-JUMPER
491. CHECK-NEED-FOR-UB-REPEATER
497. SELECT-BOX-AND-UB-MODULE-FOR-NEXT-SU
501. ASSIGN-BACKPLANE-TO-BOX
505. GENERATE-SLOT-TEMPLATES
510. PUT-UB-MODULE
518. ADD-SUBOPTIMAL-UB-MODULE
527. LEAVE-BACKPLANE
540. AUGMENT-UB-LENGTH
543. GET-UB-JUMPER
547. CHECK-NEED-FOR-UB-REPEATER
553. LEAVE-HALF-BOX

Example of a Configuration Run (4)

559. CHECK-FOR-UB-JUMPER-CHANGES
561. CHECK-TERMINATION-CONDITIONS
568. SELECT-BOX-AND-UB-MODULE-FOR-NEXT-SU
571. ASSIGN-BACKPLANE-TO-BOX
576. GENERATE-SLOT-TEMPLATES
580. PUT-UB-MODULE
581. ASSOCIATE-MULTIPLEXER-WITH-PANEL-SLOT
590. ASSOCIATE-MULTIPLEXER-WITH-PANEL-SLOT
598. ASSOCIATE-MULTIPLEXER-WITH-PANEL-SLOT
604. ADD-SUBOPTIMAL-UB-MODULE
608. ASSOCIATE-MULTIPLEXER-WITH-PANEL-SLOT
615. ADD-SUBOPTIMAL-UB-MODULE
617. LEAVE-BACKPLANE
626. AUGMENT-UB-LENGTH
629. GET-UB-JUMPER
633. CHECK-NEED-FOR-UB-REPEATER
643. LEAVE-HALF-BOX
644. CHECK-FOR-UB-JUMPER-CHANGES
646. CHECK-TERMINATION-CONDITIONS
657. SELECT-BOX-AND-UB-MODULE-FOR-NEXT-SU
660. ASSIGN-BACKPLANE-TO-BOX
663. GENERATE-SLOT-TEMPLATES
667. PUT-UB-MODULE
668. ASSOCIATE-MULTIPLEXER-WITH-PANEL-SLOT
677. ASSOCIATE-MULTIPLEXER-WITH-PANEL-SLOT
690. LEAVE-BACKPLANE
711. AUGMENT-UB-LENGTH
Example of a Configuration Run (5)

15.
16.
17.
18.
19.
20.
21.
22.
23.
24.
25.
26.
27.
28.
29.
30.

Example of a Configuration Run (6)

974.
1021.
1135.
1179.
1183.
1187.
1192.
1198.
1248.

The trace shows the complexity of the resulting process. The context structure has been forced onto the process against the spirit of the data-driven operations of rule-based systems.
Example of a OPS5 Rule in XSEL

XSEL has been developed 1980 - 1982 by CMU for DEC as a companion system for XCON. The task was to support salespersons acquiring customer wishes.

```plaintext
[p capacity-specified:1:adjust-requirement
 ; if memory capacity was ordered on the same line as the system [e.g. system
 ; with 4 meg of memory], then assume the user wants the requirement in total
 ; and not in addition to what is returned as part of the system, therefore,
 ; adjust the requirement
 [context ^status active ^cname capacity-specified]
 [line-item ^status input ^class memory ^name nil ^units kilobytes
  ^kilobytes <required> >0 ^token <token>]
 [line-item ^status pending ^class system ^parse-token <token>]
 [bus-node ^class memory ^name <device> ^ordered <count>]
 - [local ^information count-memory-capacity ^source <device>]
 - [component ^status reference ^name <device> ^number-of-kilobytes <kb>]
 ->
 [bind <ordered> [compute <quantity> * <kb>]]
 [bind <difference> [compute <required> - <ordered>]]
 [remove 3]
 [modify 2 ^kilobytes <difference>]
 [make local ^type temporary ^context capacity-specified
  ^information count-memory-capacity ^source <device>]]
```

Example for RETE Algorithm (2)

```plaintext
[15, brick, B2, blue, 280, NIL]
^class = cube?
^on = table?
^weight > 200 ?
^colour <<yellow white>>?

^class = cube?
^weight < 150 ?
^weight < 150 ?

^class = pyramid?
^on of token in alpha store =
^name of first token in beta store ?
```

```plaintext
1 12 9
4 15 9
```