Production Systems

Rule base Systems

(Busse book handout)

CSE 352
(Lecture Notes 4)
Professor Anita Wasilewska
Production Systems
(Rule Based Systems)

A production system consists of:

1. A **knowledge base**, also called a **rule base** containing production **rules**, or productions.

2. A **database**, contains **facts**

3. A **rule interpreter**, also called a rule application module to control the entire production system.
Production Rules
(Expert System Rules)

Production rules are the units of knowledge of the form:

**IF** conditions

**THEN** actions

Condition part of the rule is also called the

**IF** part, premise, antecedent or left side of the rule.
Production Rules
(Expert System Rules)

**Action part** is also called **THEN** part, conclusion, consequent, succeedent, or the right side of the rule.

**Actions** are **executed** when **conditions** are **true** and the **rule** is **fired**.

**Rules Format:**

\[ C_1 \& C_2 \& \ldots \& C_n \Rightarrow A \]

\[ C_1, \ldots, C_n, A \text{ are atomic formulas} \]
Production Rule
(Expert System Rule)

1. **Propositional logic conceptualization:** rules are propositional logic formulas i.e.

Rules are:

\[ C_1 \& C_2 \& \ldots \& C_n \Rightarrow A \]

where \( C_1, \ldots, C_n, A \) are atomic formulas

In this case atomic formulas are propositional variables or sometimes their negations

All our book examples use propositional logic conceptualization!
Production Rules

2. Predicate Form conceptualization (knowledge representation)

Rules are:

\[ C_1 \& C_2 \& \ldots \& C_n \Rightarrow A \]

where \( C_1, \ldots, C_n, A \) are atomic formulas

Atomic formulas now represent records in the database and are written in a triple form:

- \((x, \text{attribute, value of the attribute})\), or
- \((\text{ID, attribute, value of the attribute})\)

or in a predicate form

attribute \((x, \text{value of the attribute})\), attribute \((\text{ID, value of the attribute})\)
Production System ES

ES = (R, RI, DBF)

R - is a finite set of production rules
RI – is an inference engine called rule interpreter
DBF – is a database of facts (changing dynamically)

Rules are always

\[ C_1 \& \ldots \& C_n => A \]

For \( n >= 1 \) and

\( C_1, \ldots, C_n, A \) are atomic formulas in a Knowledge Representation we work with
Propositional Rule of Inference in ES

Rules Interpreter RI

Rule of inference of the Rule Interpreter is:

\[ C_1 \& C_2 \& \ldots \& C_n \Rightarrow A ; \ C_1, \ldots , C_n \]

for \( C_1, \ldots , C_n \) belonging to DBF

APPLICATION of the Rule of Inference means that

for a given rule of the production (expert) system ES

\[ C_1 \& \ldots \& C_n \Rightarrow A \]

the rule interpreter RI will check database of facts DBF and

if all \( C_1, \ldots , C_n \) belong to DBF, the interpreter will deduce A and

add A to the database of facts DBF.

We also say that the interpreter "Fire the rule" and add new fact A to the database of facts.
In **Predicate Form Conceptualization**

**Facts** are certain **atomic formulas**

attribute \((x, \text{value of the attribute})\)

where the **variable** \(x\) is replaced (**unified**) with record identifier **ID**

In **Propositional conceptualizations**

**Facts** are **propositional atomic formulas** i.e.

propositional variables or

(sometimes) negations of propositional variables
DBF – Database of Facts

The content of DBF (database of facts) is changed cyclically by the rules interpreter RI.

Facts may have time tags so that the time of their insertion by RI in to DBF can be determined.

Example: (propositional)
DBF = \{A, B\} and our ES has a rule:
\[
A \land B \Rightarrow C
\]

The interpreter RI matches A \land B with facts A, B and fires rule and adds C to the DBF and new get
NEW DBF = \{A, B, C\}
RI Rule Interpreter

RI works iteratively in **recognize-and-act** cycles

In a **ONE CYCLE**

1. **RI matches** the condition part of the rules against **facts** (current state of DBF)
2. **Recognizes all** applicable rules
3. **Selects one** of them and **applies it** (**fires**, **executes**)
4. **Adds the action part** of the applied rule (**fired rule**) to the current **DBF**.

**RI stops** when goal is reached (**problem solved**) or there are no more applicable rules.
## Predicate Form Conceptualization: Example

<table>
<thead>
<tr>
<th>Records</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$O_2$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>$O_3$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>a</td>
</tr>
</tbody>
</table>

**Constants:** (key attributes) $o_1$, $o_2$, $o_3$

Values of $a_1$ are: 1, 0,
Values of $a_2$ are: 2, 0, 1
Values of $a_3$ are: 0, 1, 2,
Values of $a_4$ are: 1, a, and
Values of $a_5$ are: 1, a, b

**TRIPLE PREDICATE FORM CONCEPTUALIZATION**

Some Atomic Formulas that are NOT FACTS are:

$(x, a_1, 1)$, $(x, a_1, 0)$, $(x, a_2, 2)$, $(x, a_5, a)$, where $x$ is a variable!

Some Atomic Formulas that ARE FACTS in our data table are:

$(O_1, a_2, 0)$, $(O_2, a_3, 1)$, $(O_3, a_5, a)$

Rule example:

$(x, a_1, 1) \& (x, a_5, a) \Rightarrow (x, a_3, 1)$
Different Forms of Atomic Formulas

Atomic formula that is a FACT written in a triple form:

\((o_1, a_7, 1)\)

The same formula written in predicate form is: \(a_7 (o_1, 1)\)

Atomic formula that is NOT a FACT written in a triple form is:

\((x, a_7, 1)\)

The same formula written in predicate form is: \(a_7 (x, 1)\)

In Busse Handout the form of atomic formulas is:

\((\text{Entity, Attribute, Value}), \ (\text{person, Attribute, Value})\),

where Entity represents a variable x, person represents a constant (like John):

\((x, \text{Attribute, Value}), \ (\text{John, Attribute, Value})\),

Where John is a constant and atomic formula becomes a FACT

We will use \(x\) to denote variables and we use the predicate form: \(\text{attribute}(x, \text{value})\)
Different Forms of Atomic Formulas

<table>
<thead>
<tr>
<th>Name</th>
<th>a1</th>
<th>Valuehouse</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>yes</td>
<td>100,000</td>
</tr>
</tbody>
</table>

**Atomic Formula** that is a **FACT** written in a **predicate form**:  

Valuehouse(John, 100,000)

**Atomic Formula** that is **NOT** a **FACT** written in a **predicate form**:  

Valuehouse(x, 100,00)

x is a variable

In our Data Table: **John** is the **key attribute**
Two Forms of Atomic Formulas

1. Some **atomic formulas** from our database that are **facts** written in Busse’s handout **triple form** are
   - (John, eyes, blue),  
   - (Mary, children, 0)  
   - (Mary, house, small),  
   - (Anita, yes, green)

2. Some atomic formulas that are **not facts** written in a **predicate form** are **eyes(x, blue)**, **house(x, small)**

Observe that the above formulas become **FACTS** when x becomes John or Mary. We say that **we MATCH** x in **eyes(x, blue)**, with the record John, or with the record Mary in **house(x, small)**

We write it:  
- **eyes(x, blue)**{x/John} = eyes(John, blue),  
- **house(x, small)**{x/Mary} = house(Mary, small)
Rule Interpreter RI

The RI works iteratively in Recognize-And-Act cycles. In such a cycle, RI:

1. **Matches** the condition part of the rules against the **facts** and **recognizes** all applicable rules
2. **Selects** one of the applicable rules and applies the rule i.e. **fires** or **executes it**: adds fact (action part) to the database

Rules have names, many have **time tag**.

**RI stops** when problem solved or no rules are applicable.
Pattern Matching: Unification

ES RULES with atomic formulas that are not FACTS written in a triples form:

\((\text{entity, attribute, value}),\) where \text{entity} is a variable, i.e.

atomic formulas that are NOT FACTS are:

\((x, \text{attribute, value}).\)

FACTS are represented by similar triples, with entity as a constant. i.e. they are:

\((\text{ID, attribute, value}).\)

Pattern matching – is matching the variable \(x\) in the triple \((x, \text{attribute, value})\) with a proper record in the database identified by the key attribute \text{ID}, i.e. with the fact \((\text{ID, attribute, value}).\)

We write it: \((x, \text{attribute, value}) \{x/\text{ID}\} = (\text{ID, attribute, value})\)
Example

Let's look at a **RULE** in a **predicate triple form representation**

(person, yearly income, >$15,000) &
(person, valuehouse, >$30,000) => (person, loantoget, <$3,000)

**Person:** variable x

**Rule Format is:** \( C_1(x) \& C_2(x) \rightarrow A(x) \)

(x, yearly income, >$15,000) &
(x, valuehouse, >$30,000) => (x, loantoget, <$3,000)

In “Plain English”: If somebody has an yearly income greater the $15,000 and his/hers house has a value greater the n$30,000, then bank approves any loan smaller than $3,000.
Facts

John – constant person = x - variable
F1: (John, yearly income, >$15,000)
F2: (John, valuehouse, >$30,000)

PATTERN MATCHING:
We assign (UNIFY) x/John (person/John)
Use Inference Rule (RI matching)
C₁ & C₂ → A {x/John}; F₁ & F₁
(John, loantoget, <$3,000)

RI adds new fact (John, loantoget, <$3,000) to the DBF
During a **cycle of RI**, most of the time is spent on **pattern matching = unification**

The most popular efficient pattern matching algorithm was **RETE algorithm** (Forgy 1982)

It is used in a rule-based **language OPS5**, a language used and still being used for **programming** expert systems

There are excellent **new unification techniques algorithms** and **Prolog** is based on the **predicate resolution** and uses them and is **the most natural, efficient and modern language** to use

We will cover **Predicate Resolution** as the next subject
ES Conflict Resolution

**RI recognition** – *part of the cycle* is divided into two parts

1. **Selection**: identification of applicable rules based on pattern matching and

2. **Conflict resolution**: choice of which rule to fire (apply, execute)

There are many possibilities and we decide what we want to use while designing the system
Conflict Resolution Heuristics

Here are some conflict resolution heuristics (choices)

**Most specific rule**
- **Example:** rules \( P \Rightarrow R \), \( P \) and \( Q \Rightarrow S \) both applicable,
- we **choose** \( P \& Q \Rightarrow S \) as it is **more specific** (contains more detailed information)

- **The rule using the most recent facts:** facts must have time tags

- **Highest Priority rule:** rules must have assigned priority

- **The first rule:** rules are linearly ordered

- **Principle:** **No rule** is allowed to **fire** more then once on basis of the same contents of **DBF**

- **We eliminate** firing the same rule all the time
Production Rules and Expert System Rules

Production rules are the rules in which actions are restricted exclusively to ADD FACTS to the DBF. Expert Systems might contain also different rules; like rules about rules (METARULES), DOMAIN-FREE rules, DOMAIN specific rules, or others.

Rules can have names (can be numbers, like R1, R2, ... etc)

Rules often have time tags or other indicators, depending of heuristics used by RI module.
Metarules

Metarules – are rules about rules.

Metarules may be **domain-specific**, such as:

*IF* the car does not start

**THEN** first check the set of rules about the fuel system

Metarules may be **Domain-free** (not connected with DBF) such as

*IF* the rules given by manual apply

**AND** textbook rules apply

**THEN**: check first manual rules
Advantages and Disadvantages of Rules Based Expert systems

**Advantage**: modularity. Rules are independent pieces of knowledge so may be added or deleted.

They are easy to understand (should be)

**Disadvantages**: inefficiency of big production systems with non-organized rules

**Rules based expert systems are the most popular**
Forward Chaining

Data -> Rules -> Goal
Also called DATA DRIVEN, BOTTOM UP, or ANTECEDENT chaining

During the SELECTION step of each cycle, the RI is looking for applicable rules by MATCHING (unifying) condition part of a rule with the CURRENT CONTENT of the DB;

Forward chaining is applied, i.e. the proper rule is FIRED and a new FACT (action part) is added to the DB.

Process TERMINATES when the GOAL is reached, or when all possible FACTS are already inferred from the INITIAL database.
Backward Chaining

Also called GOAL-DRIVEN consequent chaining

- The production system ESTABLISHES whether a goal is supported by a given database

Start with the goal
- Applicable RULES are found by matching ACTION parts with the GOAL

\[ C_1 \land \ldots \land C_n \rightarrow \text{GOAL} \]

Now the conditional part:
\[ C_1 \land \ldots \land C_n \] is checked against the DB.
If all are (after matching) in DB, the solution is reached.
If \( C_i \) is not in DB, we treat it as a SUBGOAL and repeat.
Backward Chaining (re-captured)

GOAL = Fact F
Selected rule (by matching action parts with F)
\[(R) \ C_1 \land \ldots \land C_n \rightarrow F\]

1. If all $C_1 \land \ldots \land C_n$ are in DB – End
2. Let $C$ be any of $C_1 \land \ldots \land C_n$
   after unification and substitution, if needed.
   If $\sim C$ is in DB, (R) can’t be used and another rule should be selected
3. Neither $C$ nor $\sim C$ is in DB, then
   $C$ is a SUBGOAL and we start over again as with F.
4. If no applicable rules exist, GOAL F is not established.
   System may need new rules.

Usually, backward chaining is executed as depth-first search.
Backward chaining is used in applications with large data.
Forward chaining might produce too much.
Usually, mixed strategies are used.
Example (Busse book)

Simple rule system, no variables.

Knowledge representation = propositional logic

RULES:

R1: IF the ignition key is on
    AND the engine won’t start
    THEN the starting system (including battery) is faulty

\[
R1 \quad A \land B \rightarrow E
\]

R2: IF E AND the headlights work
    THEN the starter is faulty

\[
R2 \quad E \land C \rightarrow G
\]

R3: IF E AND ~C
    THEN the battery is dead

\[
R3 \quad E \land \sim C \rightarrow I
\]
Example (continued)

R4: IF the voltage test on the ignition switch shows 1 to 6 volts,
    THEN the wiring between the ignition and the solenoid is OK R4

D → F

R5: IF F
    THEN replace the ignition switch

R5 F → H

FACTS in the INITIAL DATABASE:
A: The ignition key is on
B: The engine won’t start
C: The headlights work
D: The voltage test on the solenoid shows 1 to 6 volts

Syntax (in propositional logic representation): A, B, C, D
Initial DB
IDB = \{A, B, C, D\}

Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>A \land B</td>
<td>\rightarrow E</td>
</tr>
<tr>
<td>R2</td>
<td>E \land C</td>
<td>\rightarrow G</td>
</tr>
<tr>
<td>R3</td>
<td>E \land \neg C</td>
<td>\rightarrow I</td>
</tr>
<tr>
<td>R4</td>
<td>D</td>
<td>\rightarrow F</td>
</tr>
<tr>
<td>R5</td>
<td>F</td>
<td>\rightarrow H</td>
</tr>
</tbody>
</table>

GOAL:
Infer all possible facts from IDB

1. Rules are ordered by number:
   \( R_1 < R_2 < R_3 < R_4 < R_5 \)

2. And they are scanned by RI in this order and inserted into a queue

Conflict Resolution: ORDER (1) and Fire a rule from the front of the queue (and remove it)

STEP 1: Applicable: R1, R4
Queue (front to rear): R1, R4
Fire: R1 and add E to the IDB
NEWDB = \{A, B, C, D, E\}

STEP 2: (second cycle)
E: The starting system is faulty is added
- R1 is no longer applicable, since its action would add E, which is already in (new) DB (last in C.R.)
- R2 is applicable
Queue (front to rear): R4, R2
Step 2:  
R3 is not applicable; R4 is applicable (and is in queue); R5 is not applicable.  
R4 is FIRED from the FRONT of the queue, removed from the queue and new fact  
F: The wiring between the ignition and the solenoid is OK  
Is added to the DB, now DBF = \{ A, B, C, D, E, F \}

STEP 3 (third cycle)  
Queue: R2, R5  
R5 is inserted, R2 is FIRED (and removed) and new fact  
G: The starter is faulty  
Is added to the DB, now DBF = \{ A, B, C, D, E, F, G \}

STEP 4 (fourth cycle)  
Queue: R5  
No new rules are applicable, so R5 is fired and new fact  
H: Replace the ignition switch  
Is added to the DB

STEP 5  
No applicable rules (all are used!)  
DBF = \{ A, B, C, D, E, F, G, H \}

RI STOPS COMPUTATION
Search Space

Goal: All possible facts deduced
Initial DB
IDB= \{A, B, C, D\}

Rules

R1 \quad A \land B \rightarrow E;
R2 \quad E \land C \rightarrow G;
R3 \quad E \land \neg C \rightarrow I;
R4 \quad D \rightarrow F;
R5 \quad F \rightarrow H

GOAL
Use backward chaining to infer/reject \(H \land I\)

**First:** Consider H. H is not in the DB. The only rule that matches H (action) is
R5: \quad F \rightarrow H

Look at F; It is not in the IDB, so it is a SUBGOAL. Applicable:
R4 \quad D \rightarrow F, and D is in the IDB.
So, F is SUPPORTED and hence H is supported.

**Next:** Consider I. I is not in the DB, applicable rule is
R3 \quad E \land \neg C \rightarrow I

C is in the DB, hence R3 cannot be used. R3 is the ONLY rule, hence I is not supported and
GOAL \(H \land I\) is rejected.
Example 2 re-captured

Initial Database:  DBF= \{A, B, C, D\}

Rules
R1: A & B => E
R2: E & C => G
R3: E & ~ C => I
R4: D => F
R5: F => H

Backward Chaining Goal : H & I

First: Consider H.
H is not in DBF only rule that matches H (as action) is R5.
R5: F => H

Look at F; F is not in DB, so F becomes a subgoal

Applicable: R4: D => F, and D is in DBF so F is supported and hence H is supported.
Example 2 continued

Next: check I.

I is not in DBF, only applicable rule is \( \text{R3: } E \land \neg C \implies I \)

C is in DB, hence R3 can’t be used.

R3 is the only applicable rule, hence I is not supported and GOAL \( H \land I \) is rejected.
Propositional Logic Conceptualization

Example 3

R1: If you are hot, then turn thermostat down
R2: If you are not hot and window is open, then close the window
R3: If the thermostat is turned down and you are cold, then open the window

1. Conceptualize this system in propositional logic
2. Design questions the program has to ask the user to achieve the goal: “open the window” by backward chaining and conflict resolution
Example 3 Rules revisited

R1: hot => turn down thermostat
R2: ¬ hot & window open => close window
R3: thermostat down & cold => open window

GOAL: open window
The GOAL has to be reached by use of conflict resolution and rules R1, R2, R3 from a certain database of fact.
We need to build our DBF by asking user some questions.
Propositional Logic Conceptualization 1

H – you are hot      ¬ H – you are not hot
O – window open (open window)
D – Thermostat down
W - close window (closed window)
C - you are cold

R1: H => D
R2: ¬ H & O => W
R3: D & C => O

Goal: reach O by backward chaining
- You need to build your DBF by asking questions.
Example 3

In order to reach the goal we have only one rule applicable:

R3: D & C => O

We have two subgoals: D, C

We get D by R1: H => D and D becomes a subgoal.

No applicable rule, so we need ask a question about H.

Question: Are you hot (H) ?

If answer is YES: we ADD H into DBF , i.e.

DBF = {H} and we apply (fire ) R1: H => D and get D.

D is supported

We look now for C, no applicable rule, so we need ask a question about C
Example 3 continued

Question: Are you cold (C)?
If answer is **YES**, we ADD C into DBF, and **C is supported**, and the GOAL O is **SUPPORTED**.

If answer to the question: Are you hot (H)?
is **NO**, we added ¬H to DBF, i.e DBF = {¬H}.
No applicable rule, we STOP,
GOAL O IS REJECTED.
Propositional Logic Conceptualization 2

H – you are hot
WO– window open
OW – open the window
D – Thermostat down
CW- close the window
WC- window closed
C- you are cold
R1: H => D
R2: ¬ H & WO => CW
R3: D & C => OW
Goal: reach OW by backward chaining
- You need to build your DBF by asking questions.
**Propositional Logic Conceptualization 3**

- **H** – you are hot
- **NH** – you are not hot
- **WO** – window open
- **OW** – open the window
- **D** – Thermostat down
- **CW** – close the window
- **WC** – window closed
- **C** – you are cold

**Rules (R):**

1. **R1:** $H \implies D$
2. **R2:** $NH \& WO \implies CW$
3. **R3:** $D \& C \implies OW$

**Goal:** reach OW by backward chaining

- You need to build your DBF by asking questions.
OBSERVATION:

**FACTS are always true in ES Database**

For example a Fact:

(car#42, battery, weak), or battery(car#42,weak)

means that in our database we have a record

<table>
<thead>
<tr>
<th>Key</th>
<th>Other attribute</th>
<th>Other attribute</th>
<th>Battery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car#42</td>
<td></td>
<td>weak</td>
<td></td>
</tr>
</tbody>
</table>
Example 4: Predicate Conceptualization

Another way of writing the fact \((\text{car#42}, \text{Battery}, \text{weak})\) is:

\[
\text{Battery(}\text{ar#42}, \text{weak})
\]

This is called a **predicate form**

Atomic formula written in a **triple form** is:

\[
(\text{x, Battery, weak}) \text{, or } (\text{ID, Battery, weak})
\]

First is not a FACT, second is a FACT.

Atomic formula written in a **predicate form** is:

\[
\text{Battery(}x, \text{weak})
\]

Atomic formula that is a fact is

\[
\text{Battery(}c\#42, \text{weak})
\]
**Example 5: given a DB**

<table>
<thead>
<tr>
<th>Cars</th>
<th>Battery</th>
<th>Color</th>
<th>Buy</th>
<th>PutGarage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>good</td>
<td>red</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>$C_2$</td>
<td>weak</td>
<td>black</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

The **DB** represents the following **FACTS**: (in triple form)

F1. $(C_1, \text{battery}, \text{good})$
F2. $(C_1, \text{color}, \text{red})$
F3. $(C_1, \text{buy}, \text{no})$
F4. $(C_2, \text{battery}, \text{weak})$
F5. $(C_2, \text{color}, \text{black})$
F6. $(C_2, \text{buy}, \text{no})$

We want to use the expert system rules to PUT cars into proper garages, i.e. to fill missing values of the attribute **PutGarage**. We assume that we have two garages: G1, G2.

**WHAT IS WRONG WITH THIS PROBLEM??**
WHAT IS WRONG WITH THIS PROBLEM???

<table>
<thead>
<tr>
<th>Cars</th>
<th>Battery</th>
<th>Color</th>
<th>Buy</th>
<th>PutGarage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>good</td>
<td>red</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>$C_2$</td>
<td>weak</td>
<td>black</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

The **DB** represents the following **FACTS**: (in triple form)

F1. $(C_1, \text{battery, good})$
F2. $(C_1, \text{color, red})$
F3. $(C_1, \text{buy, no})$
F4. $(C_2, \text{battery, weak})$
F5. $(C_2, \text{color, black})$
F6. $(C_2, \text{buy, no})$

We want to use the expert system rules to PUT cars into proper garages, i.e. to fill missing values of the attribute **PutGarage**. We assume that we have two garages: $G_1, G_2$.

**NONE OF LISTED FACTS** $F_1, F_2, \ldots F_6$ **BELONGS** to the **DB**!!!
Example 6: CORRECTED

<table>
<thead>
<tr>
<th>Cars</th>
<th>Battery</th>
<th>Color</th>
<th>Buy</th>
<th>PutGarage</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₁</td>
<td>good</td>
<td>red</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td>weak</td>
<td>black</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

The DB represents the following FACTS: in PREDICATE Form

F1. Battery(C₁, good)
F2. Color(C₁, red)
F3. Buy(C₁, no)
F4. Battery(C₂, weak)
F5. Color(C₂, black)
F6. Buy(C₂, no)

Use the expert system rules (next slide) to PUT cars into proper garages, i.e. to fill missing values of the attribute PutGarage.
We assume that we have two garages: G1, G2.
A Predicate Rule of inference of the Rule Interpreter is:

\[ C_1(x) \land ... \land C_n(x) \Rightarrow A(x) \{x/ID\}; C_1(ID) \ldots C_n(ID) \Rightarrow A(ID) \]

APPLICATION of the Predicate Rule of Inference means that

for a given rule of the production (expert) system ES

\[ C_1 \land ... \land C_n \Rightarrow A \quad \text{i.e.} \quad C_1(x) \land ... \land C_n(x) \Rightarrow A(x) \]

the rule interpreter RI will check database (or database of facts) and match (unify) x with a proper record identifier ID (constant ID), if possible and evaluate

\[ C_1(x) \land ... \land C_n(x) \{x/ID\} = C_1(ID) \land ... \land C_n(ID) \]

if all \( C_1 \) (ID), ... \( C_n \) (ID) belong to DBF, the Interpreter RI will deduce

\[ A(x) \{x/ID\} = A(ID) \] and add \( A(ID) \) to the database of facts DBF.
Example 5

Some Rules in our ES (in a triple form) are:

R1. \((x, \text{Battery}, \text{good}) \& (x, \text{Color}, \text{red}) \Rightarrow (x, \text{PutGarage}, 2)\)

R2. \((x, \text{Battery}, \text{weak}) \& (x, \text{Buy}, \text{no}) \Rightarrow (x, \text{PutGarage}, 1)\)

• **Matching (Unification):** we unify \(x\) in the R1 with \(C1\) and we get

  \((x, \text{Battery}, \text{good}) \& (x, \text{Color}, \text{red}) \}\{x/C1\} = F1\&F2

  \((x, \text{PutGarage}, 2)\}\{x/C1\} = (C1, \text{PutGarage}, 2)\)
Example 5

Rules in our ES (in a triple form) are:

R1. \((x, \text{Battery, good}) \& (x, \text{Color, red}) \Rightarrow (x, \text{PutGarage, 2})\)

R2. \((x, \text{Battery, weak}) \& (x, \text{Buy, no}) \Rightarrow (x, \text{PutGarage, 1})\)

- Matching (Unification): we unify \(x\) in the rule R2 with \(C2\) and we get

\((x, \text{Battery, weak}) \& (x, \text{Buy, no})\)\{x/C2\} = F4&F6

\((x, \text{PutGarage, 1})\)\{x/C2\} = (C2, \text{PutGarage, 1})\)
Example 5: Extended Data Base

<table>
<thead>
<tr>
<th>Cars</th>
<th>Battery</th>
<th>Color</th>
<th>Buy</th>
<th>PutGarage</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_1</td>
<td>good</td>
<td>red</td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>C_2</td>
<td>weak</td>
<td>black</td>
<td>no</td>
<td>1</td>
</tr>
</tbody>
</table>

We used the expert system rules to PUT cars into proper garages, and
As a consequence we filled the missing values of the attribute PutGarage.

**EXERCISE:** Repeat it all writing rules in PREDICATE Form