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:

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

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 Rule
(Expert System Rule)

2. Predicate logic conceptualization (knowledge representation)

Rules are:

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

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:

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

or in a predicate form

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

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

For \( n \geq 1 \) and

\( C_1, C_n, A \) are atomic formulas
Propositional Rule of Inference in ES

Rules Interpreter RI

Rule of inference of the Rule Interpreter is:

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

\[ A \]

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

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

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.
DBF – Database of Facts

**Facts** are certain **atomic** formulas. 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
  
  \[
  A \& B \implies C
  \]

  The interpreter **RI** matches \(A \& B\) with **facts** \(A, B\) and **fires** rule \(r\) and **adds** \(C\) to the DBF.

  **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.
# 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, 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

**Some Atomic Formulas** are:

- $(x, a_1, 1)$, $(O_1, a_2, 0)$, $(O_2, a_3, 1)$, $(O_3, a_5, a)$,
- $(x, a_1, 0)$, $(x, a_2, 2)$, $(x, a_5, a)$, where $x$ is a variable!

**Rule example:**

- $(x_1, a_1, 1) \& (x_5, a_5, a) \rightarrow (x_2, a_3, 1)$

**Some Facts:** $(O_1, a_1, 1)$, $(O_3, a_5, a)$

**Observe** that atomic formula with a variable $x$, for example $(x, a_1, 1)$ is NOT a Fact
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:
\((x, a_7, 1)\)
The same formula written in predicate form is: \(a_7(x, 1)\)

In Busse Handout the form of atomic formulas is:
\((\text{Entity}, \text{Attribute}, \text{Value}), (\text{person}, \text{Attribute}, \text{Value}),\)
where Entity is a variable, or
\((\text{ID}, \text{Attribute}, \text{Value}), (\text{John}, \text{Attribute}, \text{Value}),\)
Where ID, John is a constant and atomic formula becomes a FACT.

We will use \(x\) to denote variables and use also 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></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 a key attribute
Two Forms of Atomic Formulas

<table>
<thead>
<tr>
<th>ID</th>
<th>Eyes</th>
<th>Shoe Size</th>
<th>Children</th>
<th>House</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Blue</td>
<td>10</td>
<td>2</td>
<td>Big</td>
<td>100,000</td>
</tr>
<tr>
<td>Mary</td>
<td>Green</td>
<td>9</td>
<td>0</td>
<td>Small</td>
<td>5,000</td>
</tr>
<tr>
<td>Anita</td>
<td>Green</td>
<td>9</td>
<td>1</td>
<td>Small</td>
<td>3,000</td>
</tr>
</tbody>
</table>

Atomic formulas
1. Some atomic formulas from our database that are facts written in Busse handout triple form
   (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
   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 with the record John, or with the record Mary
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

**ES RULES** are written with atomic formulas (in triples form) *(entity, attribute, value)*, where **entity** is a **variable**, i.e. atomic formulas are: *(x, attribute, value).*

**FACTS** are represented by similar triples, with entity as a **constant**. i.e. they are: *(ID, attribute, value).*

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

Let's look at a RULE:

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

Person- variable x

Rule is: $C_1 \& C_2 \rightarrow A$

(x, yearly income, >$15,000) &
(x, valuehouse, >$30,000) => (person, 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:
Assign x/John (person → John)
Use Inference Rule (RI matching)
\[ C_1 \land C_2 \rightarrow A \{ \text{person/John} \}; F_1 \land F_1 \]
(John, loantoget, <$3,000) ← add new fact to DB
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 for programming expert systems.

Now there are excellent new unification techniques- and Prolog is based on the resolution – so is the most natural language to use.
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)

3. 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 \( P \Rightarrow R, P \& Q \Rightarrow S \) 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 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** – eliminates 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**
USE HAND Written SLIDES NOW!
Example

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

Rules
R1: A & B => E  \hspace{1cm} R2: E & C => G
R3: E & \neg C => I  \hspace{1cm} 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

Next: check I.
I is not in DBF, only applicable rule is \textbf{R3: } E \& \neg C \Rightarrow I
C is in DB, hence R3 can’t be used.

R3 is the only applicable rule, hence \textbf{I is not supported} and \textbf{GOAL } H \& I \textbf{ is rejected.}
Proportional Logic Conceptualization
Example

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
Rules revisited

R1: hot ⇒ turn down
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 your DBF by asking user some questions.
Propositional Logic Conceptualization

H – you are hot
O – window open (open window)
D – Thermostat down
W - close window (closed window)
C- you are cold

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

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

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

If answer is **NO**, we added {¬ H} to DBF, i.e DBF = {¬ H}.

No applicable rule, we STOP.

We look for **C**, no applicable rule.

GOAL **O** IS REJECTED.
OBSERVATION:

**FACTS are always true in ES Database**

For example a Fact:

(car#42, battery, 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></td>
<td>weak</td>
</tr>
</tbody>
</table>
Example

<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></td>
<td>weak</td>
</tr>
</tbody>
</table>

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

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

This is called a predicate form.

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

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

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

\[\text{battery}(x, \text{weak})\]
Example: given a DB

<table>
<thead>
<tr>
<th>Cars</th>
<th>Battery</th>
<th>Color</th>
<th>Buy</th>
<th>Garage</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₁</td>
<td>Good</td>
<td>Red</td>
<td>Yes</td>
<td>2</td>
</tr>
<tr>
<td>C₂</td>
<td>Weak</td>
<td>Black</td>
<td>No</td>
<td>1</td>
</tr>
</tbody>
</table>

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

R₁. \((x, \text{battery, good}) \& (\text{color, red}) \Rightarrow (x, \text{garage, 2})\)

R₂. \((x, \text{battery, weak}) \& (x, \text{gar, 1}) \Rightarrow (x, \text{buy, no})\)

Observe that the rules R₁, R₂ are true in our DB!
Facts in our ES are:

**F1.** \((C_1, \text{battery}, \text{good})\)

**F2.** \((C_1, \text{color}, \text{red})\)

**F3.** \((C_2, \text{battery}, \text{weak})\)

**F4.** \((C_2, \text{garage}, \text{1})\)

**Question 1:** What our ES with R1 and R2 will deduce from these facts?
FACTS written in the predicate form are:

F1. battery(C₁, good)
F2. color(C₁, red)
F3. batt(C₂, weak)
F4. garage(C₂, 1)

**Question 2:** write rules R1 and R2 in predicate form and use conflict resolution to deduce all you can from facts F1-F4.