Production Systems ES (2)
(Book and Busse handout)

CSE 352
Lecture Notes (5)
Professor Anita Wasilewska
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 $\neg C$ is in DB, (R) can’t be used and another rule should be selected
3. Neither $C$ nor $\neg C$ 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: A ∧ B → E

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

R2: E ∧ C → G

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

R3: E ∧ ~C → 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

R1 \( A \land B \Rightarrow E \)
R2 \( E \land C \Rightarrow G \)
R3 \( E \land \neg C \Rightarrow I \)
R4 \( D \Rightarrow F \)
R5 \( F \Rightarrow H \)

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 R1 in this order and inserted into a queue

STEP 1  Applicable: R1, R4
Queue (front to rear): R1, R4

Conflict Resolution: ORDER (1) and
Fire a rule from the front of the queue (and remove it)
Fire: R1 and add E to the IDB

\[ \text{NEWDB} = \{A, B, C, D, E\} \]

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
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**
Goal: All possible facts deduced
EXAMPLE 2

Initial DB
IDB= \{A, B, C, D\}

Rules

R1 \ A \land B \Rightarrow E;
R2 \ E \land C \Rightarrow G;
R3 \ E \land \lnot C \Rightarrow I;
R4 \ D \Rightarrow F;
R5 \ 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: \ F \Rightarrow H

Look at F; It is not in the IDB, so it is a SUBGOAL. Applicable:
R4 \ 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 \ E \land \lnot 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:  \( \text{DBF}= \{A, B, C, D\} \)

Rules

\[
\begin{align*}
\text{R1: } A & \& B \Rightarrow E & \quad \text{R2: } E & \& C \Rightarrow G \\
\text{R3: } E & \& \neg C \Rightarrow I & \quad \text{R4: } D \Rightarrow F \\
\text{R5: } F \Rightarrow H
\end{align*}
\]

Backward Chaining Goal :  \( H \& I \)

First: Consider  \( H \).

\( H \) is not in \( \text{DBF} \) only rule that matches \( H \) ( as action) is \( \text{R5} \).

\( \text{R5: } F \Rightarrow H \)

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

Applicable:  \( \text{R4: } D \Rightarrow F \), and  \( D \) is in \( \text{DBF} \) so  \( F \) is supported  and hence  \( H \) is supported.
Example 2 continued

Next: check I.

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

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
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 continued

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

**R3**: $D \& C \Rightarrow O$

We have two subgoals: $D, C$

We get $D$ by **R1**: $H \Rightarrow D$ 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**: $H \Rightarrow D$ 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 \( \neg H \) to DBF, i.e DBF = \{\neg 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 O 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

R1: H => D
R2: NH & WO => CW
R3: D & C => OW

Goal: reach O 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

<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 \((\text{car#42, battery, weak})\) is:

\text{battery(car#42, weak)}

This is called a **predicate form**

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

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

Atomic formula that is a fact is

\text{battery(car#42, 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. $(C_1, \text{color}, \text{black})$
F6. $(C_2, \text{buy}, \text{no})$

We will use the expert system rules to PUT cars into proper garages, i.e. to fill missing values of the attribute **PutGarage**
Example 6: 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 PREDICATE Form

F1. Battery($C_1$, good)
F2. Color($C_1$, red)
F3. Buy($C_1$, no)
F4. Battery($C_2$, weak)
F5. Color($C_1$, black)
F6. Buy($C_2$, no)

We will use the expert system rules to PUT cars into proper garages, i.e. to fill missing values of the attribute PutGarage
Rule of inference of the Rule Interpreter is:

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

**APPLICATION** of the **Predicate Rule of Inference** means that for a given rule of the production (expert) system ES

\[ C_1 \land \ldots \land C_n \Rightarrow A \] i.e. \[ C_1(x) \land \ldots \land C_n(x) \Rightarrow A(x) \]

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

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

if all \( C_1(ID), \ldots C_n(ID) \) belong to DBF, the **Interpreter RI** will **deduce** \( A(x)\{x/ID\} = A(ID) \) and add \( A(ID) \) to the **database of facts DBF**.
Some Rules in our ES (in a triple form) are:

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

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

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

\((x, \text{battery, good}) \land (x, \text{color, 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 \(C_2\) and we get

\((x, \text{battery, weak }) \& (x,\text{buy, no}) \}x/C_2\} = F_4 \& F_6\)

\((x, \text{PutGarage, 1})\}x/C_2\} = (C_2, \text{PutGarage, 1})\)
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>C₁</td>
<td>good</td>
<td>red</td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>C₂</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