CE213 Artificial Intelligence – Lecture 8

Production Rule Interpreters

How to search knowledge base for problem solving?

- How many rules are relevant to given situations or hypothesis?

- How to apply (combine) these relevant rules to make decisions?

Interpreters: Forward chaining vs. Backward chaining

Core Components of a Production System

1. A set of production rules

Sometimes called the ***rule base*** *(or even knowledge base)*

2. *A production rule interpreter*

A program which selects and executes rules whose condition parts match the current situation.

3. An environment (may be embedded in rule set)

Condition parts of the rules must be propositions about some objects or entities.

Action parts of the rules must specify operations to be performed on some objects or entities.

An environment consists of a **set of objects or entities** that the system responds to and manipulates.

Production Rule Interpreter

What is a production rule interpreter?

An interpreter is a program that can read/select and apply the rules in a production system.

Such an interpreter will ***systematically*** access the rules in an attempt to establish what conclusions may be drawn.

There are several ways that such an interpreter could be constructed.

The two commonest types are called ***forward chaining interpreter*** and ***backward chaining interpreter***.

Information about Environment States

An environment consists of a set of objects or entities.

In all but the simplest systems, the states of an environment provide two types of information:

Dynamic

The information that determines which rules can fire and is modified by the rules.

In many systems dynamic information is called ***working memory,*** *which stores confirmed conditions and conclusions.*

Static

Other information used by the rules in matching conditions and determining actions.

Forward Chaining Rule Interpreter

The Basic Idea and Steps:

For a simple forward chaining production system, the interpreter goes through ***match-execute*** *cycles*:

Match Step

Scans through rules to find one whose **condition part** is matched by the current state (situation) of the environment (working memory).

Execute Step

Performs the operation specified by the **action part** of the rule that has been found to match.

This is sometimes called ***firing*** the rule.

Typically this will change the state of the environment.

Normally this match-execute cycle continues until either no rule matches or the system is explicitly stopped by the action part of a rule.

A Toy Example: Mechanic

To demonstrate the operation of forward chaining we will use a toy expert system called Mechanic.

Mechanic's task is to determine why a car doesn't start and suggest a remedy.

To keep things really simple we will:

Consider just the first 5 rules of Mechanic (21 rules in total).

And always choose the first rule we find that matches the current situation (simplest conflict resolution).

Mechanic – The First Five Rules

Rule 1:

IF Lights work AND Starter will turn
THEN CONCLUDE Battery OK

Can you identify the environment from these 5 rules?

Rule 2:

IF Lights do not work AND Starter will not turn
THEN CONCLUDE Battery flat

Rule 3:

IF Battery flat
THEN CONCLUDE Remedy is recharge battery

The remaining 16 rules are in the last 4 slides for this lecture.

Rule 4:

IF Lights work AND Starter will not turn
THEN CONCLUDE Loose connection in starter circuit

Rule 5:

IF Loose connection in starter circuit
THEN CONCLUDE Remedy is locate and repair loose connection

Mechanic: Forward Chaining

The system begins the **first match-execute cycle** by considering Rule 1.

Rule 1:

IF Lights work AND Starter will turn

THEN CONCLUDE Battery OK

To evaluate the condition part it must ask the user whether the lights work and whether the starter will turn.

Assume, as a result, it adds the following facts to its working memory:

Lights work

Starter will not turn

Rule 1 does not match, so the system considers Rule 2 next.

Mechanic: Forward Chaining (2)

Rule 2:

IF Lights do not work AND Starter will not turn

THEN CONCLUDE Battery flat

Because Lights work, Rule 2 fails to match. The system now considers Rule 3.

Rule 3:

IF Battery flat

THEN CONCLUDE Remedy is recharge battery

There is no fact in working memory stating that the battery is flat, and this is not something that can be asked, so Rule 3 also fails.

Mechanic: Forward Chaining (3)

So, the system next considers Rule 4

Rule 4:

IF Lights work AND Starter will not turn

THEN CONCLUDE Loose connection in starter circuit

The facts in working memory match the condition part of Rule 4, so this rule is selected to execute or fire.

The additional fact from the conclusion drawn by Rule 4

Loose connection in starter circuit

is added to working memory

This concludes the first match-execute cycle. Please note that in each match-execute cycle only one matched rule fires.

Mechanic: Forward Chaining (4)

The system now begins the **second match-execute cycle**.

Rules 1 to 4 proceed as before (except that it is not necessary to ask the user what has already been asked):

Rules 1 to 3 will fail again.

Rule 4 will match again and would be able to fire.

It looks as though we are stuck in an infinite loop!

To avoid this, many forward chaining interpreters do not allow the same rule to fire twice as a result of matching the same facts.

This property is called ***refractoriness***.

We will assume that our forward chaining rule interpreter has this property, so Rule 4 will not be selected for firing again.

Mechanic: Forward Chaining (5)

So, the system next considers Rule 5

Rule 5:

IF Loose connection in starter circuit

THEN CONCLUDE Remedy is locate and repair loose connection

The condition part of Rule 5 matches the fact in working memory, so this rule is selected to fire and the conclusion

Remedy is locate and repair loose connection

is added to working memory.

This completes the second match-execute cycle.

During the **third match-execute cycle**, the only rules that match the facts in working memory will be refractory.

Consequently, no rules will be selected for firing. The system will stop.

Conflict Resolution

The need for conflict resolution arises whenever two or more rules match the current environment state at the same time.

Generally it is not possible for more than one action to be carried out, because they may be mutually contradictory.

e.g., a robot control system with two matched rules that suggest “Turn left” and “Turn right” at the same time.

Procedures used to select one rule from several candidates for firing are called ***conflict resolution strategies***.

Conflict Resolution Strategies

First match found

Widely used in early production systems. Rarely used now.

Convenient for human interpreter.

System's behaviour depends on the order in which rules are written down.

Random choice

Major disadvantage - makes behaviour non-deterministic.

For some simulations this could be an advantage, but generally it is inconvenient.

Conflict Resolution Strategies (2)

Specificity

Chooses the rule which has the most restrictive condition part (… AND …).

Makes it easy to handle exceptional cases.

Widely used.

Recency

Chooses the rule whose condition part has been satisfied by information most recently added to working memory.

Tends to keep the system focused on same part of the problem until it has been resolved.

Widely used.

Assigned Priorities

In many systems, the programmer can assign a numerical priority to rules.

In cases of conflict, the one with the highest priority is chosen.

Features/Properties of Forward Chaining

- **Data Driven**

  The system has no explicit goals. What it does is determined only by the current situation.

- **Rule Selection**

  It selects rules with conditions matching the current situation.

  Conflict resolution is used to choose which rule to fire if more than one matches.

- **Iterative**

  Repeated execution of the match-execute cycle is used to choose and fire rules.

Backward Chaining Rule Interpreter

The Basic Idea and Steps:

1. Start with a conclusion hypothesis
2. Find **all** the rules whose RHSs draw conclusions about the hypothesis.
3. Determine whether the LHSs of those rules **match** the current situation. If it is not sure whether a rule matches or not, then make a subsidiary hypothesis about its LHS and go back to step 2 to determine whether the LHS is true. Otherwise go to step 4.
4. **Execute** the corresponding RHSs with matched LHSs, thus confirming or rejecting the hypothesis.

Step 3 may involve setting up subsidiary hypothesis to determine whether a LHS is true. This is the difficult part in backward chaining!

Backward chaining is ***recursive***, whilst forward chaining is ***iterative***.

Mechanic: Backward Chaining

Consider again the Mechanic expert system made of just the first 5 rules.

The car won't start – our goal is to find out the reason and a remedy.

Therefore, the initial hypothesis is *Remedy is X,* where *X* could be anything. (Step 1)

Inspecting the RHSs of all 5 rules we find two rules that match the hypothesis:
Rule 3:

IF Battery flat

THEN CONCLUDE Remedy is recharge battery

Rule 5:

IF Loose connection in starter circuit

THEN CONCLUDE Remedy is locate and repair loose connection

(Step 2)

Mechanic: Backward Chaining (2)

First, the system considers Rule 3. (Starting Step 3 …)

Its LHS is *Battery Flat*.

But, we don't know if the battery is flat at the moment, so we are not sure if the LHS of this rule is satisfied.

We therefore set up a subsidiary hypothesis *Battery Flat*

Then we look for rules whose RHSs draw conclusions about the flatness of the battery.

(*Note we do not abandon this rule yet, as what would be done by forward chaining*.)

Mechanic: Backward Chaining (3)

(Going back to Step 2 …)

Among all 5 rules only Rule 2 has the RHS matching the subsidiary hypothesis.

Rule 2:

IF Lights do not work AND Starter will not turn

THEN CONCLUDE Battery flat

(Step 3 again …)

The LHS of Rule 2 is *Lights do not work AND Starter will not turn*.

The system itself doesn't know either of these facts.

These are something the system must ask the user, similar to what is done in forward chaining.

In general, the system will know which facts can be deduced using other rules and which are data the user must supply.

Mechanic: Backward Chaining (4)

(Still in Step 3 …)

Let us suppose that the user tells the system (similar as in forward chaining):

Lights work

Starter will not turn. (add them to working memory)

The LHS of Rule 2 is *Lights do not work AND Starter will not turn.*

So, the system is unable to conclude *Battery flat.*

The system has now exhausted all the rules it found when it set out to establish the subsidiary hypothesis *Battery flat.* It was doing this in an attempt to determine whether Rule 3 could fire, but the outcome is negative.

Rule 3:

IF Battery flat

THEN CONCLUDE Remedy is recharge battery

Mechanic: Backward Chaining (5)

(Still is Step 3 …)

The system therefore abandons Rule 3 and turns to Rule 5, the other rule it found when it set up the original hypothesis Remedy is X.

Rule 5:

IF Loose connection in starter circuit

THEN CONCLUDE Remedy is locate and repair loose connection

The LHS of Rule 5 is Loose connection in starter circuit, which we are not sure based on facts in working memory.

We therefore set up a subsidiary hypothesis Loose connection in starter circuit.

(Going back to Step 2 again …)

The system finds one rule only, which draws the conclusion about this subsidiary hypothesis. That is,

Rule 4:

IF Lights work AND Starter will not turn

THEN CONCLUDE Loose connection in starter circuit

Mechanic: Backward Chaining (6)

(Step 3 again …)

The LHS of Rule 4 is matched by facts already established (Lights work AND Starter will not turn) , so the rule fires and draws the conclusion (add it to working memory):

Loose connection in starter circuit

(Step 4 …)

This deduced fact matches the LHS of Rule 5, so it can fire and draw its conclusion

Remedy is locate and repair loose connection

This matches the original hypothesis (add it to working memory).

There are no more untried rules to consider (exhaustive backward chaining).

So, the consultation is complete. The rules that fire are in the following order: Rule 4, Rule 5.

Features/Properties of Backward Chaining

- **Hypothesis Driven (good for diagnosis and classification problems)**

  At each stage the system has a goal of trying to establish the truth of some conclusion. May need ***less working memory*** than forward chaining.

- **Rule Selection**

  It selects rules with RHS matching the hypothesis, which will help determine if the hypothesis is true. All matched rules are considered.

  Conflict resolution is not necessary.

- **Recursive**

  While trying to establish the truth of a **hypothesis**, the system may find it needs to know the truth value of some other proposition.

  To do this, it will set up this other proposition as a **subsidiary hypothesis** and try to prove that it is true.

Summary

Core Components of a Production System

A set of production rules, Interpreter, Environment (working memory)

Forward Chaining Rule Interpreter

Basic idea and steps, Conflict resolution, Refractoriness, Key features

Backward Chaining Rule Interpreter

Basic idea and steps, Key features

Mechanic Expert System as an Example

Forward or backward, that is the question.

MECHANIC - A SIMPLE EXPERT SYSTEM

A simple set of informal rules that attempts to determine why a car will not start.

Rule 1:

IF Lights work AND Starter will turn

THEN CONCLUDE Battery OK

Rule 2:

IF Lights do not work AND Starter will not turn

THEN CONCLUDE Battery flat

Rule 3:

IF Battery flat

THEN CONCLUDE Remedy is recharge battery

Rule 4:

IF Lights work AND Starter will not turn

THEN CONCLUDE Loose connection in starter circuit

Rule 5:

IF Loose connection in starter circuit

THEN CONCLUDE Remedy is locate and repair loose connection

Rule 6:

IF Starter will turn AND Engine will not fire
THEN CONCLUDE Suspect ignition system fault
AND CONCLUDE Suspect fuel supply fault

Rule 7:

IF Suspect ignition system fault
AND Each spark plug fires correctly
THEN CONCLUDE Ignition system OK

Rule 8:

IF Suspect ignition system fault
AND Only one plug fails to fire
THEN CONCLUDE Plug fault

Rule 9:

IF Plug fault
THEN CONCLUDE Remedy is check HT cable from distributor and plug itself, replacing as necessary

Rule 10:

IF Suspect ignition system fault

AND All plugs fail to fire

THEN CONCLUDE Distributor fault

Rule 11:

IF Distributor fault

THEN CONCLUDE Remedy is check LT cable to distributor and distributor itself, replacing as necessary

Rule 12:

IF Suspect fuel supply fault

AND Fuel reaches cylinders

THEN CONCLUDE Fuel supply OK

Rule 13:

IF Suspect fuel supply fault

AND Fuel reaches carburettor

THEN CONCLUDE Carburettor fault

Rule 14:

IF Carburettor fault

THEN CONCLUDE Remedy is repair or replace carburettor

Rule 15:

IF Suspect fuel supply fault

AND Fuel does not reach carburettor

THEN CONCLUDE Suspect empty petrol tank

AND CONCLUDE Suspect fuel pump failure

AND CONCLUDE Suspect fuel blockage

Rule 16:

IF Suspect empty petrol tank

AND Petrol gauge indicates no fuel

THEN CONCLUDE Remedy is fill tank with petrol

Rule 17:

If Suspect empty petrol tank

AND Petrol gauge indicates fuel in tank

THEN CONCLUDE Petrol in tank

Rule 18:

IF Suspect fuel pump AND Fuel pump operates

THEN CONCLUDE Fuel pump OK

Rule 19:

IF Suspect fuel pump

AND Fuel pump does not operate

THEN CONCLUDE Remedy is repair or replace fuel pump

Rule 20:

IF Suspect fuel blockage

AND Fuel pump OK

AND Petrol in tank

THEN CONCLUDE Remedy is clear blockage in fuel supply

Rule 21:

IF Battery OK

AND Ignition system OK

AND Fuel supply OK

THEN CONCLUDE Remedy is call in a car expert!