Knowledge based

Humans use heuristics a great deal in their problem solving. Of course, if the heuristic does fail, it is necessary for the problem solver to either pick another heuristic, or know that it is appropriate to give up. The rules, found in the knowledge bases of rule-based systems, are very often heuristics.

Knowledge based expert systems

Expert system programming is distinctively different from conventional programming.

Whereas one could describe a conventional program (or at least, the part of it that produces the results, as opposed to the user interface, etc) in these terms:

Program = algorithm + data

One would have to describe an expert system in these terms:

Expert system = inference engine + knowledge base + data.

Inferencing

The inference engine uses one of several available forms of inferencing.

By inferencing means the method used in a knowledge-based system to process the stored knowledge and supplied data to produce correct conclusions.

Example

How old are you?

Subtract the year you were born in from 2014.

The answer will either be exactly right,

or

one year short.

Multiple solutions.

In planning or design tasks, a single solution will probably be enough.

In diagnostic tasks, all possible solutions are probably needed.

Reasoning with uncertainty.

Rules in the knowledge base may only express a probability that a conclusion follows from certain premises, rather than a certainty.

This is particularly true of medicine and other life sciences.

Forward chaining

Forward chaining working from the facts to a conclusion. Sometimes called the data driven approach. To chain forward, match data in working memory against 'conditions‘ of rules in the rule-base.

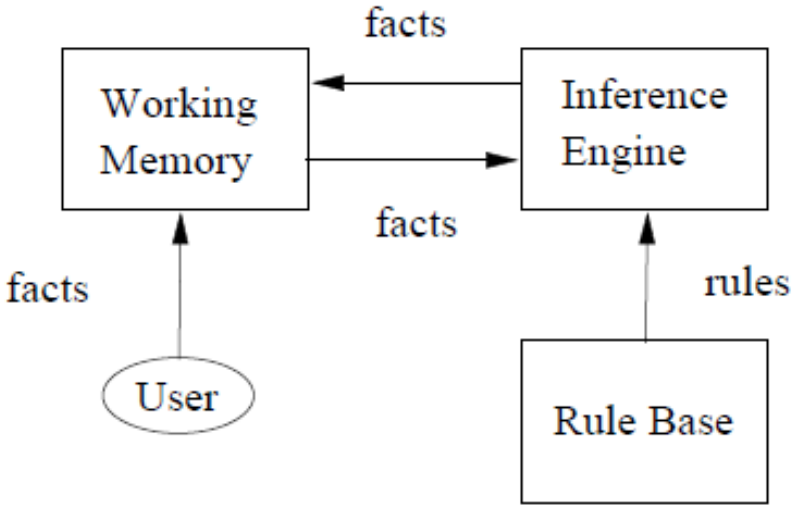
Starts with the facts, and sees what rules apply (and hence what should be done) given the facts.

How is it works

Facts are held in a working memory

Condition-action rules represent actions to take when specified facts occur in working memory.

Typically the actions involve adding or deleting facts from working memory.



Steps in FC

- To chain forward, match data in working memory against 'conditions' of rules in the rule base.
- When one of them fires, this is liable to produce more data.
- So the cycle continues up to conclusion.

Example

- Here are two rules:
- If corn is grown on poor soil, then it will get blackfly.
- If soil hasn't enough nitrogen, then it is poor soil.

Forward chaining: This soil is low in nitrogen; therefore this is poor soil; therefore corn grown on it will get blackfly.

More realistically,

“there's something wrong with this corn. So I test the soil. It turns out to be low in nitrogen. If that’s the case, corn grown on it will get blackfly. Therefore the problem is blackfly caused by low nitrogen”

Backward chaining

Backward chaining: working from the conclusion to the facts. Sometimes called the goal-driven approach.

Starts with something to find out, and looks for rules that will help in answering it goal driven.

Steps in BC

- To chain backward, match a goal in working memory against 'conclusions' of rules in the rule-base.
- When one of them fires, this is liable to produce more goals.
- So the cycle continues

Example

- Same rules:
- If corn is grown on poor soil, then it will get blackfly.
- If soil hasn't enough nitrogen, then it is poor soil.

Backward chaining: This corn has blackfly; therefore it must have been grown on poor soil; therefore the soil must be low in nitrogen.

More realistically,

> “The BC reasoning would be: there's something wrong with this corn. Perhaps it has blackfly; if so, it must have been grown on poor soil; if so, the soil must be low in nitrogen. So test for low nitrogen content in soil, and then we'll know whether the problem was blackfly.”

FC or BC

The choice of strategy depends on the nature of the problem.

Assume the problem is to get from facts to a goal (e.g. symptoms to a diagnosis)

IF BC

Backward chaining is the best choice if: The goal is given in the problem statement, or can sensibly be guessed at the beginning of the consultation;

or:

The system has been built so that it sometimes asks for pieces of data (e.g. "please now do the gram test on the patient's blood, and tell me the result"), rather than expecting all the facts to be presented to it.

Reasons

This is because (especially in the medical domain) the test may be

1. expensive,
2. or unpleasant,
3. or dangerous for the human participant

so one would want to avoid doing such a test unless there was a good reason for it.

IF FC

Forward chaining is the best choice if:

All the facts are provided with the problem statement;

or:

There are many possible goals, and a smaller number of patterns of data;

or:

There isn't any sensible way to guess what the goal is at the beginning of the consultation.

Which is better

If you have clear hypotheses, backward chaining is likely to be better.

Diagnostic problems or classification problems Medical expert systems Forward chaining may be better if you have less clear hypothesis and want to see what can be concluded from current situation.

Mixed Chaining

Some systems use mixed chaining, where some of the rules are specifically used for chaining forwards, and others for chaining backwards.

The strategy is for the system to chain in one direction, then switch to the other direction, so that: the diagnosis is found with maximum efficiency; the system's behaviour is perceived as "human".

Best for expert systems

A backwards-chaining system tends to produce a sequence of questions which seems focussed and logical to the user,

A forward-chaining system tends to produce a sequence which seems random & unconnected.

If it is important that the system should seem to behave like a human expert, backward chaining is probably the best choice.

Example

R1: IF hot AND smoky THEN fire

R2: IF alarm_beeps THEN smoky

R3: If fire THEN switch_on_sprinklers

F1: alarm_beeps [Given]

F2: hot [Given]

Example

R1: IF hot AND smoky THEN ADD fire

R2: IF alarm_beeps THEN ADD smoky

R3: If fire THEN ADD switch_on_sprinklers

F1: alarm_beeps [Given]

F2: hot [Given]

Example

R1: IF hot AND smoky THEN ADD fire
R2: IF alarm_beeps THEN ADD smoky
R3: If fire THEN ADD switch_on_sprinklers

F1: alarm_beeps [Given]
F2: hot [Given]
F3: smoky [from F1 by R2]
F4: fire [from F2, F4 by R1]
F5: switch_on_sprinklers [from F4 by R3]

For BC

Should I switch the sprinklers on?

F1: alarm_beeps [Given]