الجامعة التكنولوجية

قسم علوم الحاسوب

فرع الذكاء الاصطناعي

المرحلة الثالثة

الكورس الثاني

مادة الأنظمة الخبيرة
2025 - 2024

أ.د. حسنين سمير عبدالله

الجامعة التكنولوجية / قسم علوم الحاسوب
فرع الذكاء الاصطناعي – المرحلة الثالثة – الكورس الثاني
مادة الأنظمة الخبيرة – أ.د. حسنين سمير

1. Introduction to Expert Systems

Expert systems are computer programs that are constructed to do the kinds of activities that human experts can do such as design, compose, plan, diagnose, interpret, summarize, audit, give advice. The work such a system is concerned with is typically a task from the worlds of business or engineering/science or government.

Expert systems are usually set up to operate in a manner that will be perceived as intelligent: that is, as if there were a human expert on the other side of the video terminal. A characteristic body of programming techniques give these programs their power. Expert systems generally use automated reasoning and the so-called weak methods, such as search or heuristics, to do their work. These techniques are quite distinct from the well-articulated algorithms and crisp mathematical procedures more traditional programming.

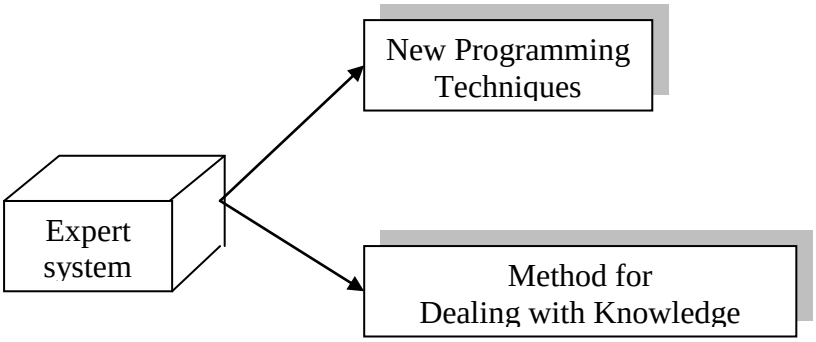


Figure (1) the vectors of expert system development

As shown in Figure(1), the development of expert systems is based on two distinct, yet complementary, vectors:

a. New programming technologies that allow us to deal with knowledge and inference with ease.

b. New design and development methodologies that allow us to effectively use these technologies to deal with complex problems.

The successful development of expert systems relies on a well-balanced approach to these two vectors.

2. Expert System Using

Here is a short nonexhaustive list of some of the things expert systems have been used for:

- To approve loan applications, evaluate insurance risks, and evaluate investment opportunities for the financial community.
- To help chemists find the proper sequence of reactions to create new molecules.
- To configure the hardware and software in a computer to match the unique arrangements specified by individual customers.
- To diagnose and locate faults in a telephone network from tests and trouble reports.
- To help geologists interpret the data from instrumentation at the drill tip during oil well drilling.
- To help physicians diagnose and treat related groups of diseases, such as infections of the blood or the different kinds of cancers.
- To help navies interpret hydrophone data from arrays of microphones on the ocean floor that are used t\u the surveillance of ships in the vicinity.
- To examine and summarize volumes of rapidly changing data that are generated too last for human scrutiny, such as telemetry data from landsat satellites.

Most of these applications could have been done in more traditional ways as well as through an expert system, but in all these cases there were advantages to casting them in the expert system mold.

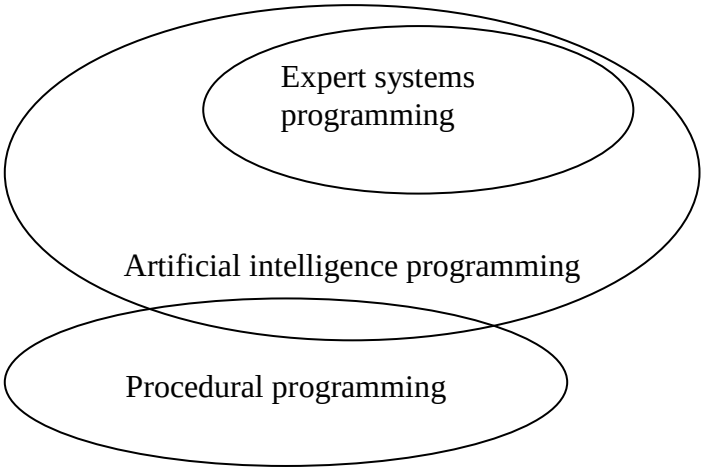
In some cases, this strategy made the program more human oriented. In others, it allowed the program to make better judgments.

In others, using an expert system made the program easier to maintain and upgrade.

3. Expert Systems are Kind of AI Programs

Expert systems occupy a narrow but very important corner of the entire programming establishment. As part of saying what they are, we need to describe their place within the surrounding framework of established programming systems.

Figure(2) shows the three programming styles that will most concern us. Expert systems are part of a larger unit we might call AI (artificial intelligence) programming. Procedural programming is what everyone learns when they first begin to use BASIC or PASCAL or FORTRAN. Procedural programming and A.I programming are quite different in what they try to do and how they try to do it.



Figure(2) three kinds of programming

In traditional programming (procedural programming), the computer has to be told in great detail exactly what to do and how to do it. This style has

been very successful for problems that are well defined. They usually are found in data processing or in engineering or scientific work.

AI programming sometimes seems to have been defined by default, as anything that goes beyond what is easy to do in traditional procedural programs, but there are common elements in most AI programs. What characterizes these kinds of programs is that they deal with complex problems that are often poorly understood, for which there is no crisp algorithmic solution, and that can benefit from some sort of symbolic reasoning.

There are substantial differences in the internal mechanisms of the computer languages used for these two sorts of problems. Procedural programming focuses on the use of the assignment statement (" = " or ":-") for moving data to and from fixed, prearranged, named locations in memory. These named locations are the program variables. It also depends on a characteristic group of control constructs that tell the computer what to do. Control gets done by using

if-then-else	goto
do-while	procedure calls
repeat-until	sequential execution (as default)

AI programs are usually written in languages like Lisp and Prolog. Program variables in these languages have an ephemeral existence on the stack of the underlying computer rather than in fixed memory locations. Data manipulation is done through pattern matching and list building. The list techniques are deceptively simple, but almost any data structure can be built upon this foundation. Many examples of list building will be seen later when we begin to use Prolog. AI programs also use a different set of control constructs. They are :

procedure calls

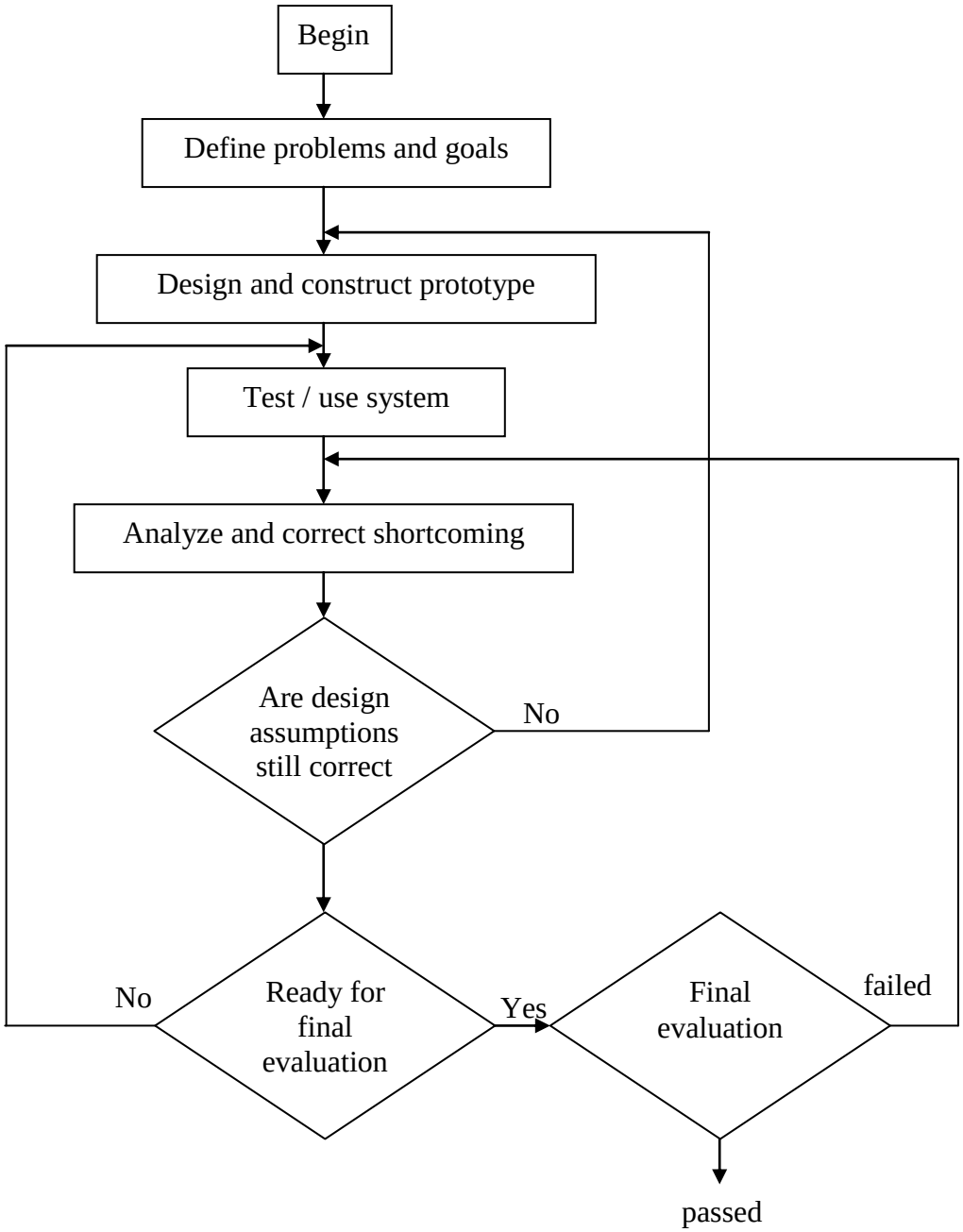
sequential execution

recursion

4. Expert System, Development Cycle

The explanation mechanism allows the program to explain its reasoning to the user, these explanations include justification for the system's conclusions, explanation of why the system needs a particular piece of data.

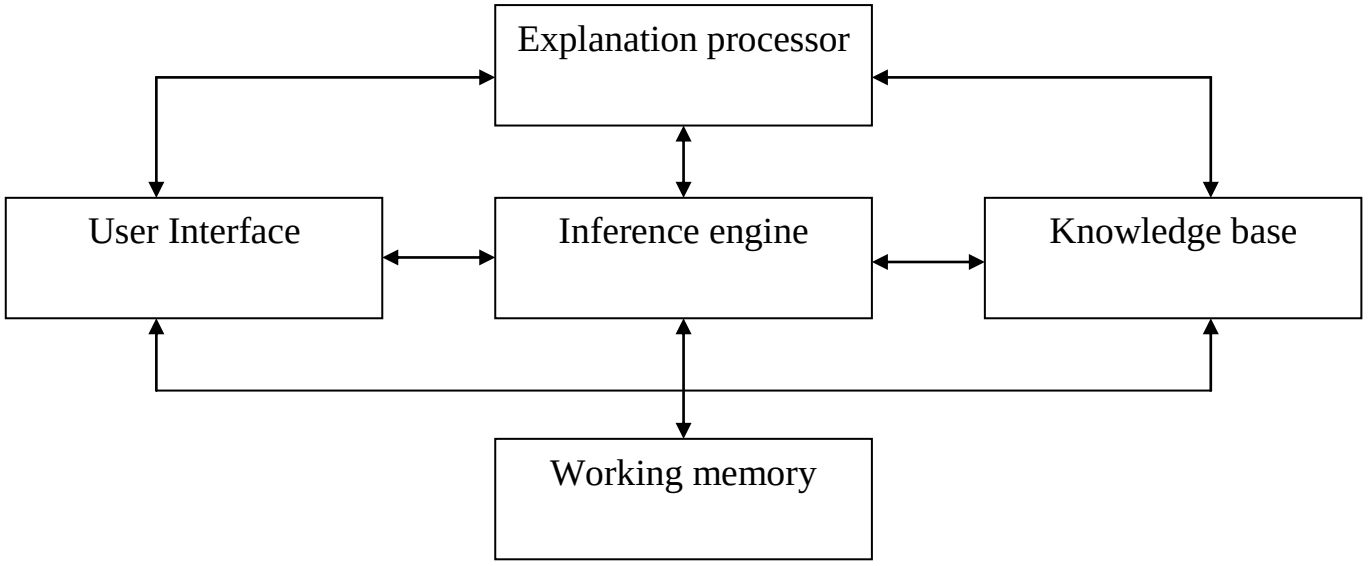
Figure (3) below shows the exploratory cycle for rule based expert system.



Figure(3) The exploratory cycle for expert system

5. Expert System Architecture and Components

The architecture of expert system consists of several components as shown in figure (4) below:



Figure(4)Expert system architecture

5.1. User Interface

The user interacts with the expert system through a user interface that make access more comfortable for the human and hides much of the system complexity. The interface styles includes questions and answers, menu-driver, natural languages, or graphics interfaces.

5.2. Explanation Processor

The explanation part allows the program to explain its reasoning to the user. These explanations include justifications for the system's conclusion (HOW queries), explanation of why the system needs a particular piece of data (WHY queries).

5.3. Knowledge Base

The heart of the expert system contains the problem solving knowledge (which defined as an original collection of processed information) of the

particular applications, this knowledge is represented in several ways such as if-then rules form.

5.4. Inference Engine

The inference engine applies the knowledge to the solution of actual problems. It s the interpreter for the knowledge base. The inference engine performs the recognize act control cycle.

The inference engine consists of the following components:-

1. Rule interpreter.
2. Scheduler
3. HOW process
4. WHY process
5. knowledge base interface.

5.5. Working Memory

It is a part of memory used for matching rules and calculation. When the work is finished this memory will be raised.

6. Systems that Explain their Actions

An interface system that can explain its behavior on demand will seem much more believable and intelligent to its users. In general, there are two things a user might want to know about what the system is doing. When the system asks for a piece of evidence, the user might want to ask,

"Why do you want it?"

When the system states a conclusion, the user will frequently want to ask,

"How did you arrive at that conclusion?"

This section explores simple mechanisms that accommodate both kinds of questioning. HOW and WHY questions are different in several rather obvious ways that affect how they can be handled in an automatic reasoning program. There are certain natural places where these questions are asked, and they are at opposite ends of the inference tree. It is appropriate to let the user ask a WHY question when the system is working with implications at the

bottom of the tree; that is: when it will be necessary to ask the user to supply data.

The system never needs to ask for additional information when it is working in the upper parts of the tree. These nodes represent conclusions that the system has figured out rather than asked for, so a WHY question is not pertinent. To be able to make the conclusions at the top of the tree, however, is the purpose for which all the reasoning is being done. The system is trying to deduce information about these conclusions. It is appropriate to ask a HOW question when the system reports the results of its reasoning about such nodes.

There is also a difference in timing of the questions. WHY questions will be asked early on and then at unpredictable points all throughout the reasoning. The system asks for information when it discovers that it needs it. The. time for the HOW questions usually comes at the end when all the reasoning is complete and the system is reporting its results.

7. Differentiation between Expert System and other Intelligent Systems

i) In the expert systems the inference engine is split from knowledge base while in the other intelligent systems they are merged (work together), this gave the expert systems flexible properties in:

- Modifications, insertion, deletions, updating easier.
- Less run time and reduced memory capacity and effort.
- Replace the current knowledge base with new or other knowledge base from the same environment (task(s)).

ii) The expert system has the ability to answer about the user's queries or questions especially the WHY and HOW questions, WHY question related with required input, and HOW question related with resulted output.

iii) The expert system in its architecture contains the explanation mechanism, with responsible for the above property (ii) which is not available in other intelligent systems.

Controlling the Reasoning Strategy (1)

The control strategy is determined as comparing the number of initial state(s) to the number of goal state(s), therefore and according to the fact that say "the search will be from less to more" we can determine the control strategy for any system (if the set of initial state(s) and goal state(s) are clear and complete) easily.

For the classification system

The number of initial state(s) : The number of goal state(s)

Many (properties) 1 (the target class)

The search will be from less to more

Thus the preferred control strategy is "backward" chaining.

Classification Program with Backward Chaining (Bird, Beast, Fish) Version1

```
database
db_confirm(symbol, symbol)
db_denied(symbol, symbol)
clauses
guess_animal :- identify(X), write("Your animal is a(n) ",X),!.

identify(giraffe) :-
                    it_is(ungulate),
                    confirm(has, long_neck),
                    confirm(has, long_legs),
                    confirm(has, dark_spots)
identify(zebra) :-
                    it_is(ungulate),
                    confirm(has, black_strips),!.
```

```
identify(cheetah) :-
                    it_is(mammal),
                    it-is(carnivorous),
                    confirm(has, tawny_color),
                    confirm(has, black_spots),!.
identify(tiger) :-
                    it_is(mammal),
                    it-is(carnivorous),
                    confirm(has, tawny_color),
                    confirm(has, black_strips),!.

identify(eagle) :-
                    it_is(bird),
                    confirm(does, fly),
                    it-is(carnivorous),
                    confirm(has, use_as_national_symbol),!.


identify(ostrich) :-
                    it_is(bird),
                    not(confirm(does, fly)),
                    confirm(has, long_neck),
                    confirm(has, long_legs),!.
identify(penguin) :-
                    it_is(bird),
                    not(confirm(does, fly)),
                    confirm(does, swim),
                    confirm(has, black_and_white_color),!.
```

```
identify(blue_whale) :-
                    it_is(mammal),
                    not(it-is(carnivorous)),
                    confirm(does, swim),
                    confirm(has, huge_size),!.
identify(octopus) :-
                    not(it_is(mammal),
                    it_is(carnivorous),
                    confirm(does, swim),
                    confirm(has, tentacles),!.
identify(sardine) :-
                    it_is(fish),
                    confirm(has, small_size),
                    confirm(has, use_in_sandwiches),!.
identify(unknown).          /* Catch-all rule if nothing else works.
*/


it-is(bird):-
            confirm(has, feathers),
            confirm(does, lay_eggs),!
it-is(fish):-
            confirm(does, swim),
            confirm(has, fins),!.
 it-is(mammal):-
            confirm(has, hair),!.
it-is(mammal):-
            confirm(does, give_milk),!.
it-is(ungulate):-
```

```
        it-is(mammal),
        confirm(has, hooves),
        confirm(does, chew_cud),!.
it-is(carnivorous):-
        confirm(has, pointed_teeth),!.
it-is(carnivorous):-
        confirm(does, eat_meat),!.

confirm(X,Y):- db_confirm(X,Y),!.
confirm(X,Y):- not(denied(X,Y)),!, check(X,Y).

denied(X,Y):- db-denied(X,Y),!.

Check(X,Y):- write(X, " it ", Y, \ "n"), readln(Reply), remember(X, Y,
Reply).

remember(X, Y, yes):- asserta(db_confirm(X, Y)).
remember(X, y, no):- assereta(db_denied(X, Y)), fail.
```

Controlling the Reasoning Strategy (2)

According to the same assumptions, we can reach to same facts that say:

For the classification system

The number of initial state(s) : The number of goal state(s)

Many (properties) 1 (the target class)

The search will be from less to more

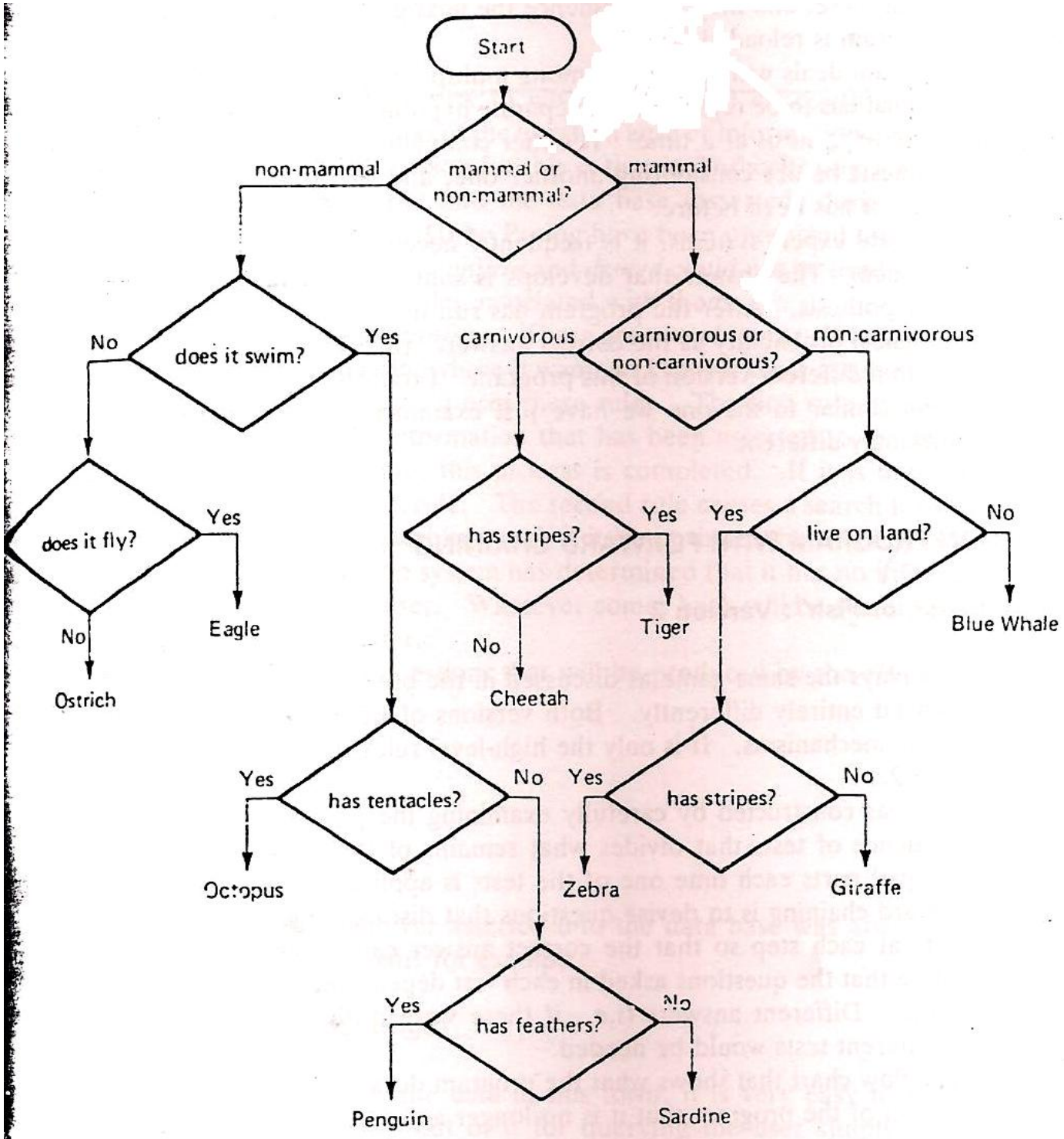
Thus the preferred control strategy is "backward" chaining, but it can be used the "forward" chaining as another strategy to solve the animal

classification system but under different conditions as they illustrated in the system requirements such as:

-Decision tree to design the problem.

-The special code for the classification system as a forward chaining.

Classification Program with Forward Chaining (Bird, Beast, Fish) Version2



BBF is a classification program. The forward chaining version makes a series of binary decisions. Each is designed to throw away one half the remaining possibilities (or as close to that as possible until only one is left.

```
database
db_confirm(symbol, symbol)
db_denied(symbol, symbol)
clauses
guess_animal :-
                    find_animal, have_found(X),
                    write("Your animal is a(n) ",X),nl,!.

find_animal:- test1(X), test2(X,Y), test3(X,Y,Z), test4(X,Y,Z,_),!.
Find_animal.

test1(m):- it_is(mammal),!.
test1(n).

test2(m,c):- it_is(carnivorous),!.
test2(m,n).
test2(n,w):- confirm(does, swim),!.
test2(n,n).

test3(m,c,s):- confirm(has, strips), asserta(have_found(tiger)),!.
test3(m,c,n):- asserta(have_found(cheetah)),!.
test3(m,n,l):- not(confirm(does, swim)),
                not(confirm(does, fly)),!.
test3(m,n,n):-  asserta(have_found(blue_whale)),!.
test3(n,n,f):- confirm(does, fly),
               asserta(have_found(eagle)),!.
test3(n,n,n):- asserta(have_found(ostrich)),!.
test3(n,w,t):- cofirm(has, tentacles),
                asserta(have_found(octopus)),!.
```

```
test3(n,w,n).

test4(m,n,l,s):- confirm(has, strips),
                 asserta(have_found(zebra)),!.
test4(m,n,l,n):- asserta(have_found(giraffe)),!.
test4(n,w,n,f):- confirm(has, feathers),
                 asserta(have_found(penguin)),!.
test4(n,w,n,n):- asserta(have_found(sardine)),!.

it-is(bird):- confirm(has, feathers),
              confirm(does, lay_eggs),!.
it-is(fish):- confirm(does, swim),
              confirm(has, fins),!.
 it-is(mammal):- confirm(has, hair),!.
it-is(mammal):- confirm(does, give_milk),!.
it-is(ungulate):- it-is(mammal),
                  confirm(has, hooves),
                  confirm(does, chew_cud),!.
it-is(carnivorous):- confirm(has, pointed_teeth),!.
it-is(carnivorous):- confirm(does, eat_meat),!.

confirm(X,Y):- db_confirm(X,Y),!.
confirm(X,Y):- not(denied(X,Y)),!, check(X,Y).
denied(X,Y):- db-denied(X,Y),!.

Check(X,Y):- write(X, " it ", Y, \ "n"), readln(Reply), remember(X, Y, Reply).

remember(X, Y, yes):- asserta(db_confirm(X, Y)).
remember(X, y, no):- assereta(db_denied(X, Y)), fail.
```

Conclusions

1. Code written for backward chaining is clearer. All the rules in version 1 of BBF have a nice declarative reading. They correspond nicely to most people's intuitive idea of how things should be described when they are part of some kind of hierarchy. The description is top down.
2. Code written for backward reasoning is also much easier to modify or expand. It is apparent without much thought what would have to be done to add another animal (class) to the structure: just define it. But it is not always clear where to attach another instance to a forward reasoning rule structure. In fact, if a number of additions have to be made, all the rules may have to be redone to accommodate the additions and at the same time to maintain the same testing efficiency as was there before.
3. Code for the backward reasoning system will be easier to develop in the first place because the built-in inference method in prolog is backward chining.

Rule-Based Expert Systems

Rule-based expert systems represent problem-solving knowledge as *if.. then...* rules. This approach lends itself to the architecture of typical expert system that was described previously, and is one of the famous techniques for representing domain knowledge in an expert system. It is also one of the most natural, and remains widely used in practical and experimental expert systems.

The Production System and Control Strategy in Problem Solving

The architecture of rule-based expert systems may be best understood in terms of the production system model for problem solving. The production system was the intellectual precursor of modern expert system architectures, where application of production rules leads to refinements of understanding of a particular problem situation. When the production system is developed, the goal was to model human performance in problem solving.

If we regard the expert system architecture as a production system, the domain-specific knowledge base is the set of production rules. In a rule-based system, these condition action pairs are represented as *if*..... *then*..... rules, with the premises of the rules, the *if* portion, corresponding to the condition, and the conclusion, the *then* portion, corresponding to the action: when the condition is satisfied, the expert system takes the action of asserting the conclusion as true. Case-specific data can be kept in the working memory. The inference engine implements the recognize-act cycle of the production system; this control may be either data-driven or goal-driven.

Many problem domains seem to lend themselves more naturally to forward search. In an interpretation problem, for example, most of the data for the problem are initially given and it is often difficult to

formulate an hypotheses or goal. This suggests a forward reasoning process in which the facts are placed in working memory and the system searches for an interpretation.

In a goal-driven expert system, the goal expression is initially placed in working memory. The system matches rule *conclusions* with the goal, selecting one rule and placing its *premises* in the working memory. This corresponds to a decomposition of the problem's goal into simpler subgoals. The process continues in the next iteration of the production system, with these premises becoming the new goals to match against rule conclusions. The system thus works back from the original goal until all the subgoals in working memory are known to be true, indicating that the hypothesis has been verified.

Thus, backward search in an expert system corresponds roughly to the process of hypothesis testing in human problem solving. In an expert system, subgoals can be solved by asking the user for information. Some expert systems allow the system designer to specify which subgoals may be solved by

asking the user. Others simply ask the user about any subgoals that fail to match rules in the knowledge base; i.e., if the program cannot infer the truth of a subgoal, it asks the user.

As an example of goal-driven problem solving with user queries, we next offer a small expert system for analysis of automotive problems. This is not a full diagnostic system, as it contains only four very simple rules. It is intended as an example to demonstrate goal driven rule chaining, the integration of new data, and the use of explanation facilities:

Rule 1: **if**
the engine is getting gas, and
the engine will turn over,
then
the problem is spark plugs.

Rule 2: **if**
the engine does not turn over, and
the lights do not come on
then
the problem is battery or cables.

Rule 3: **if**
the engine does not turn over, and
the lights do come on
then
the problem is the starter motor.

Rule 4: **if**
there is gas in the fuel tank, and
there is gas in the carburetor
then
the engine is getting gas.

To run this knowledge base under a goal-directed control regime, place the top-level goal, the problem is X, in working memory. X is a variable that can match with any phrase, for example the problem is battery or cables; it will become bound to the solution when the problem is solved.

Three rules match with this expression in working memory: rule 1, rule 2, and rule 3.

If we resolve conflicts in favor of the lowest-numbered rule, then rule 1 will fire. This causes X to be bound to the value spark plugs and the premises of rule 1 to be placed in the working memory. The system has thus chosen to explore the possible hypothesis that the spark plugs are bad.

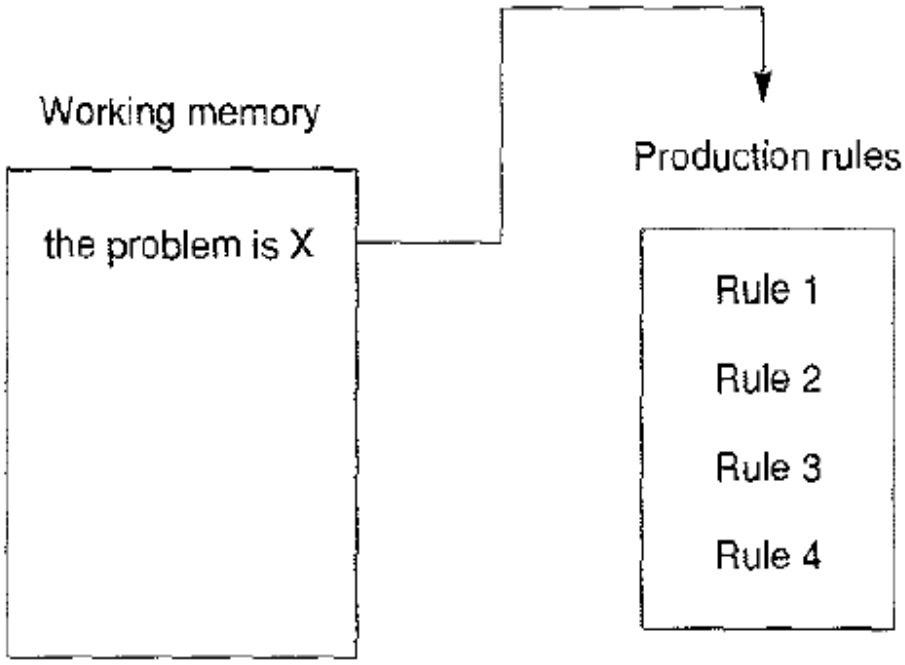


Figure (1), the production system at the start of a consultation in the car diagnostic example.

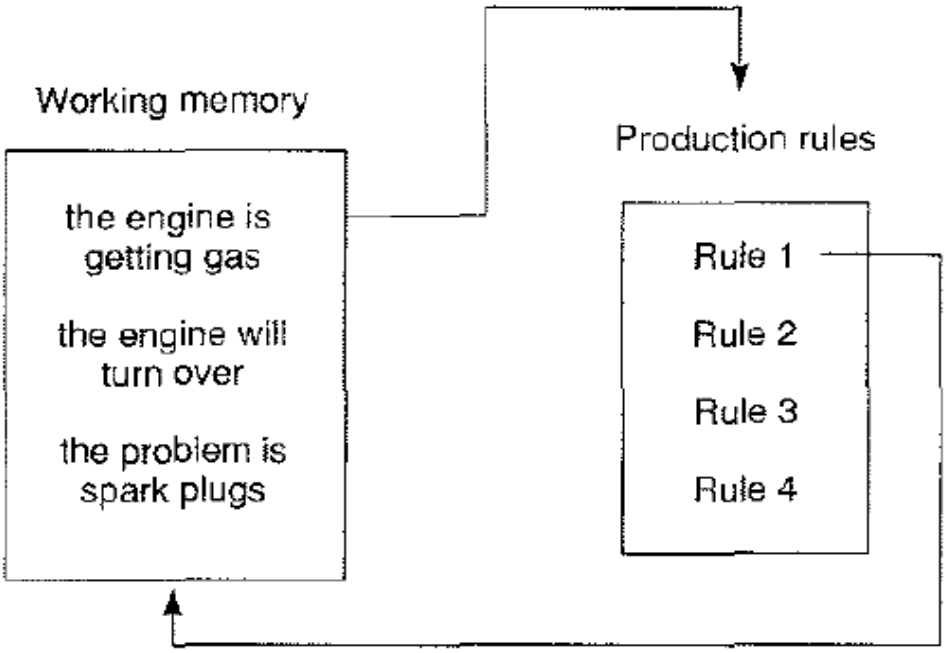


Figure (2), the production system after rule 1 has fired.

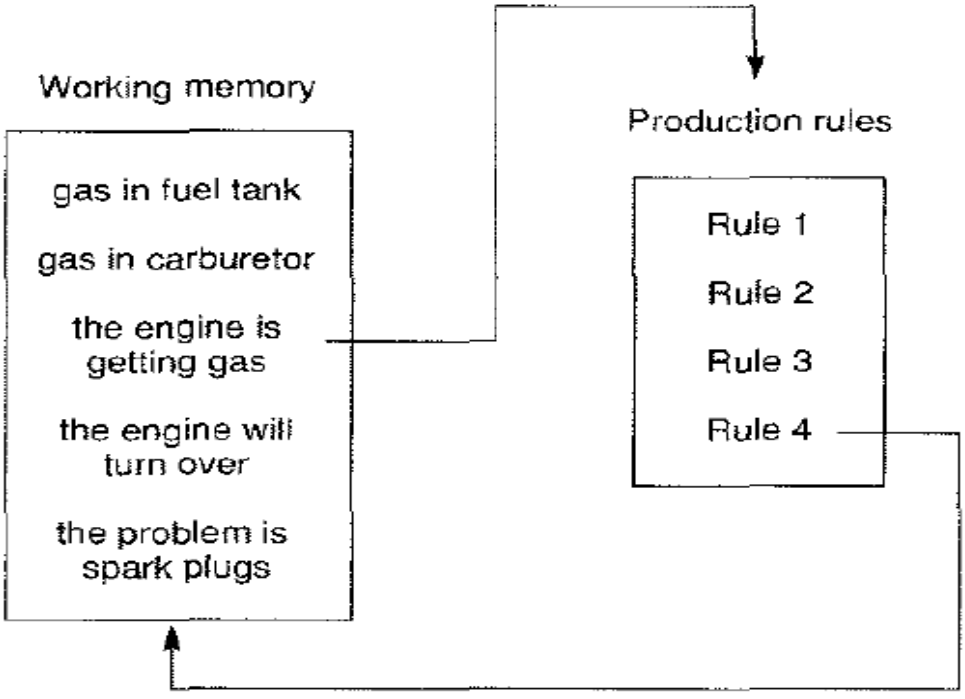


Figure (3), the production system after rule 4 has fired. Note the stack-based approach to goal reduction.

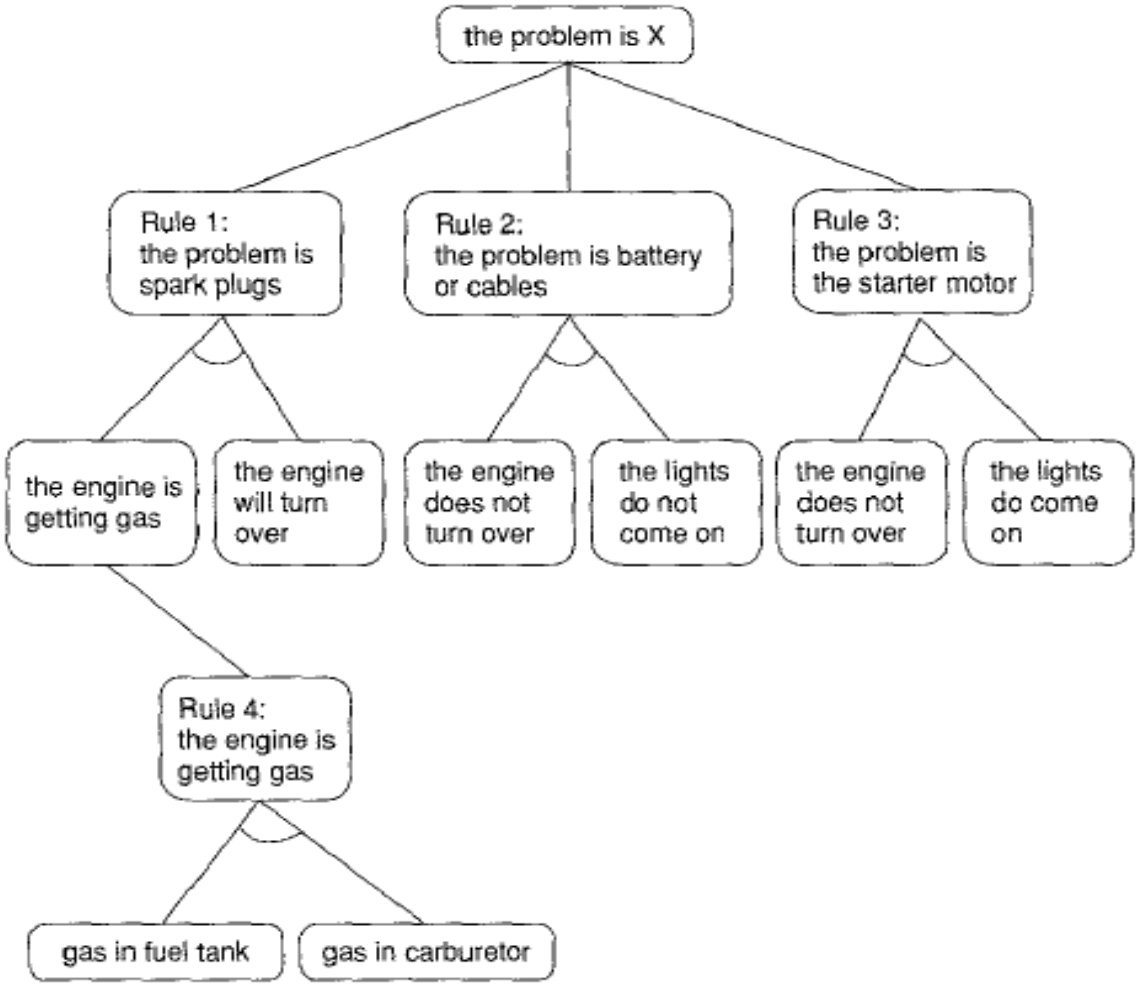
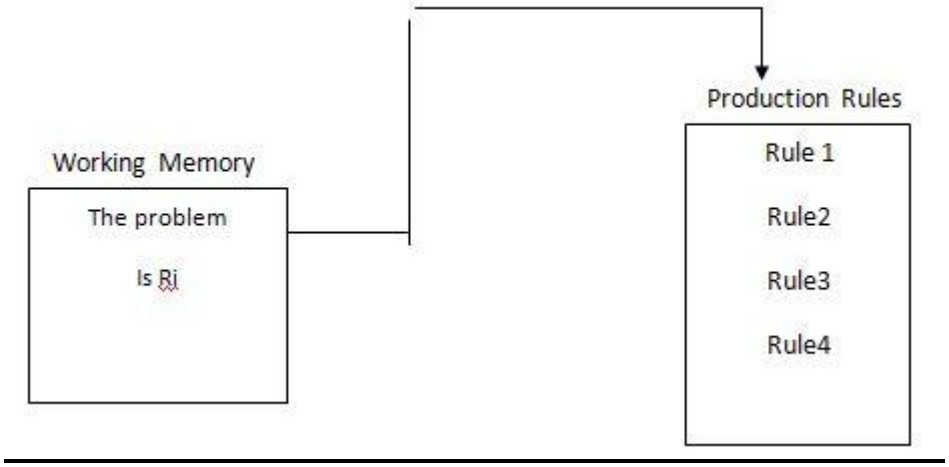


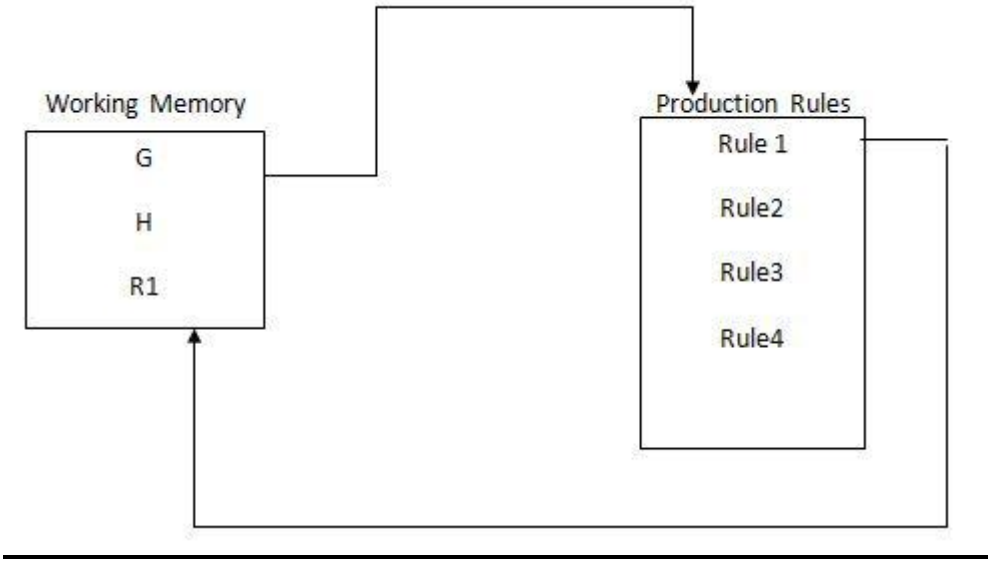
Figure (4), the and or graph searched in the car diagnosis example, with the conclusion of rule 4 matching the first premise of rule 1.

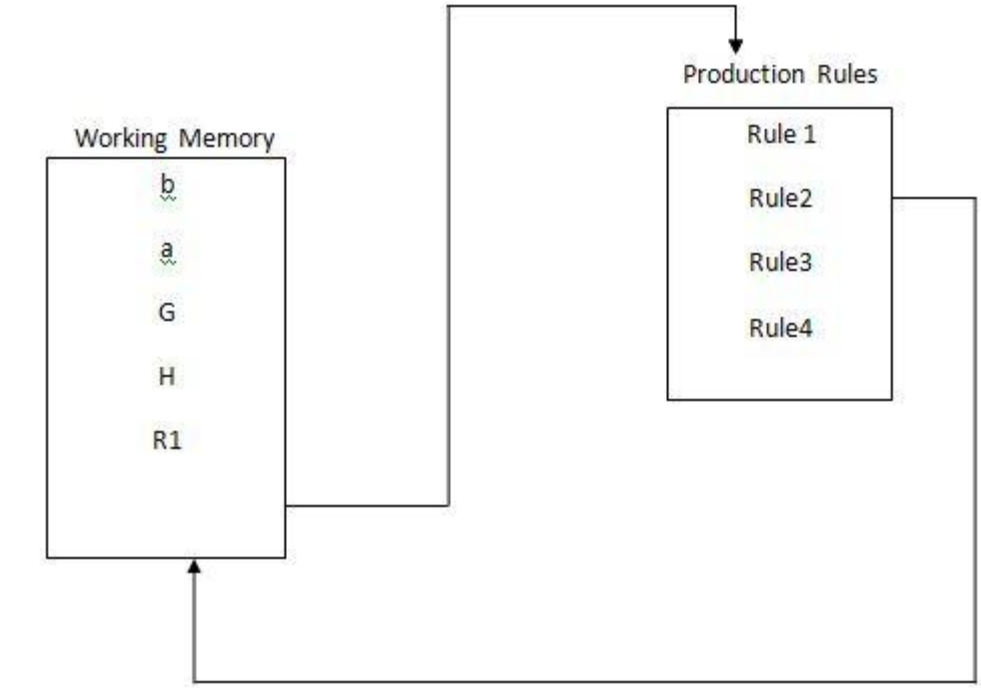
Suppose the following diagnosing problems as production rules, then attempt the tasks below:
If G and H then R1
If a and b then G
If c and d then H
If not (b) and e then R2
The facts are a, b, c, and d,

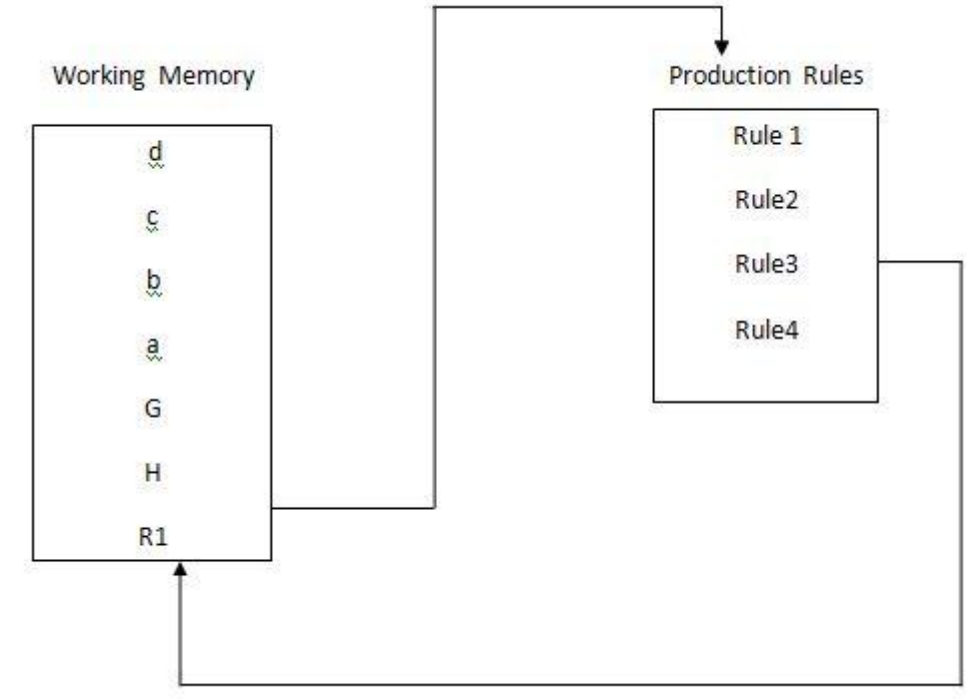
1. Through formal steps, show the contents of working memory via backward chaining.
2. Draw the AND-OR graph for the diagnosing problems.

1-

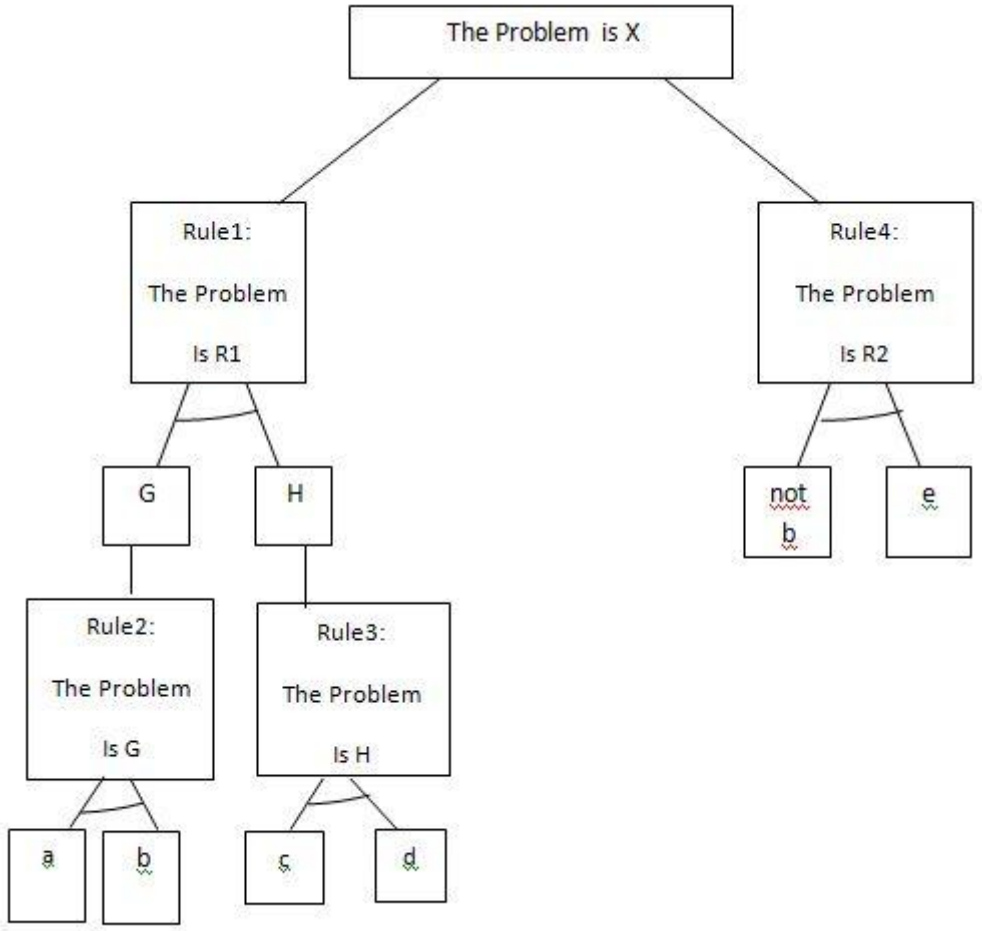





Production Rules
Rule 1
Rule2
Rule3
Rule4
Working Memory
b
a
G
H
R1


Working Memory
d
c
b
a
G
H
R1
Production Rules
Rule 1
Rule2
Rule3
Rule4

2.


The Problem is X
Rule1:
The Problem
Is R1
Rule4:
The Problem
Is R2
G
H
not
b
e
Rule2:
The Problem
Is G
Rule3:
The Problem
Is H
a
b
c
d

Rule-Based Expert Systems

(Working Memory Contents via Forward Chaining)

Consider the following production rules that describe the failure in a device, then attempt the tasks below through using these rules:

if a and b then L

if L and j then Y1

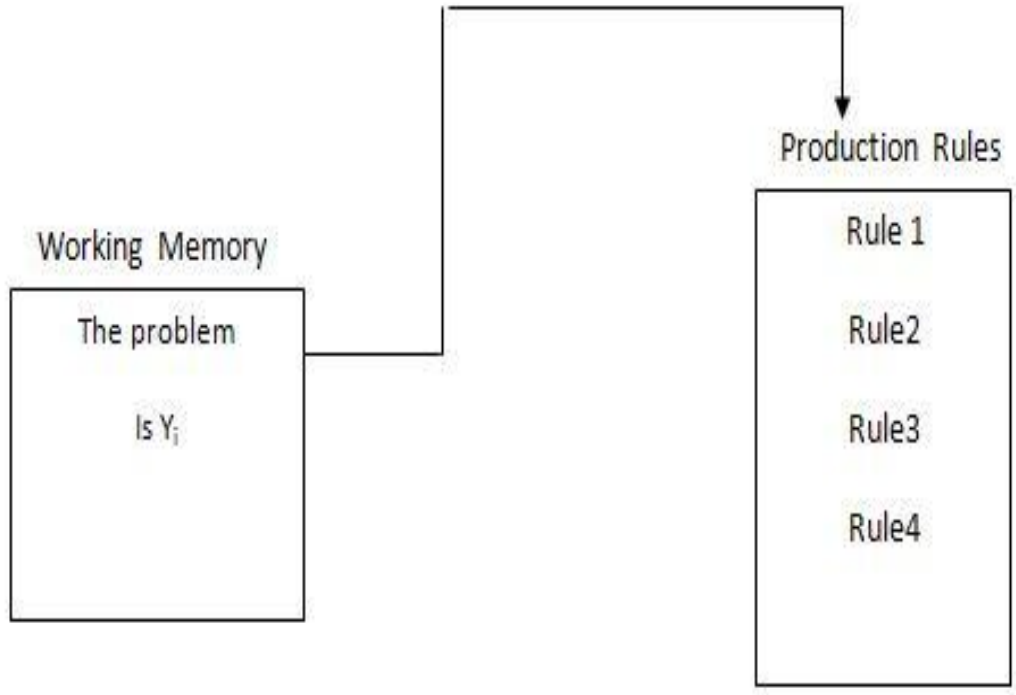
if not(j) and K then Y2

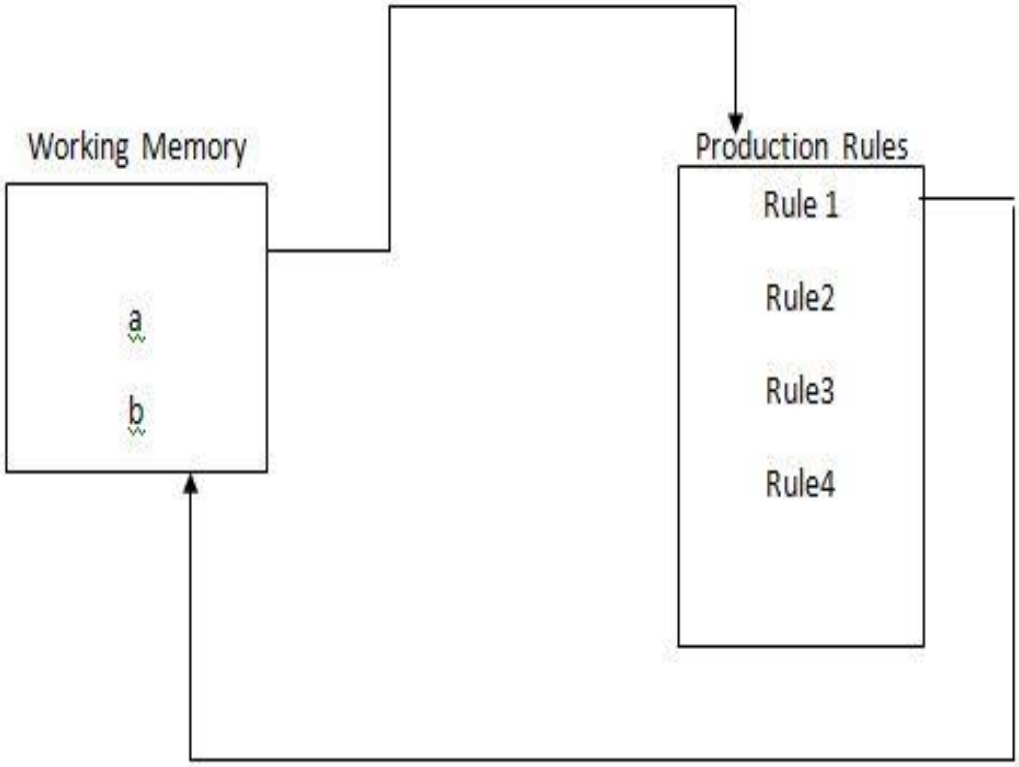
if not(j) and not(K) then Y3

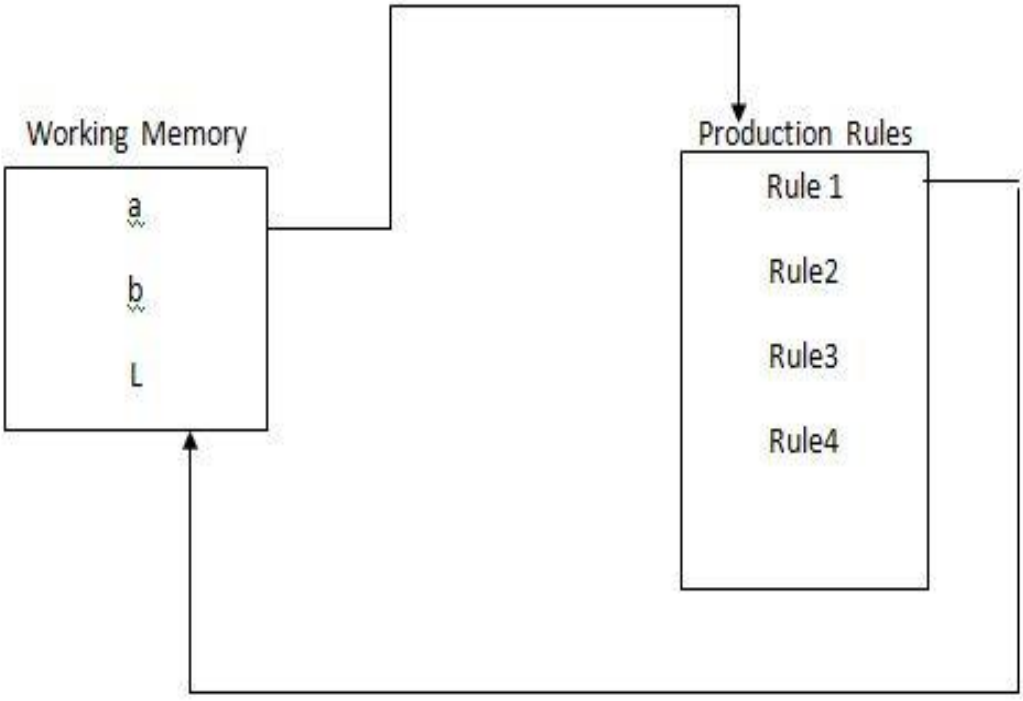
The facts are a, b, and j.

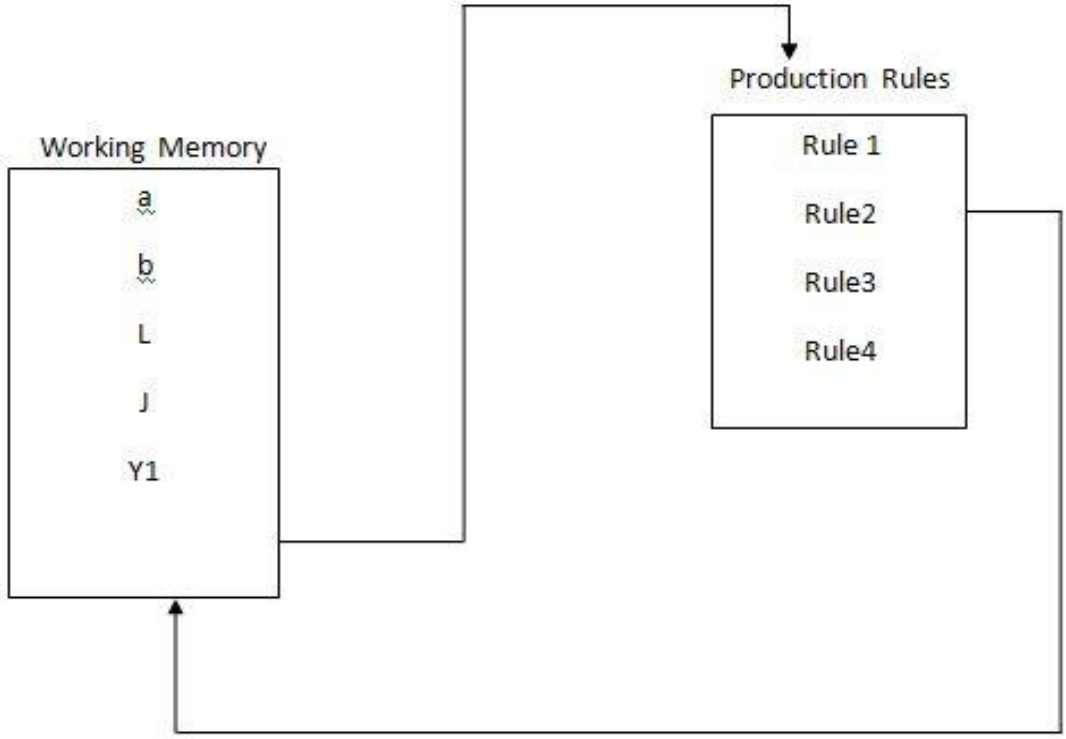
1. What are the **contents of working memory** if the system works as **forward chaining?**
2. Build the **search graph** as described by the contents of working memory for the forward chaining through any search method.

1:

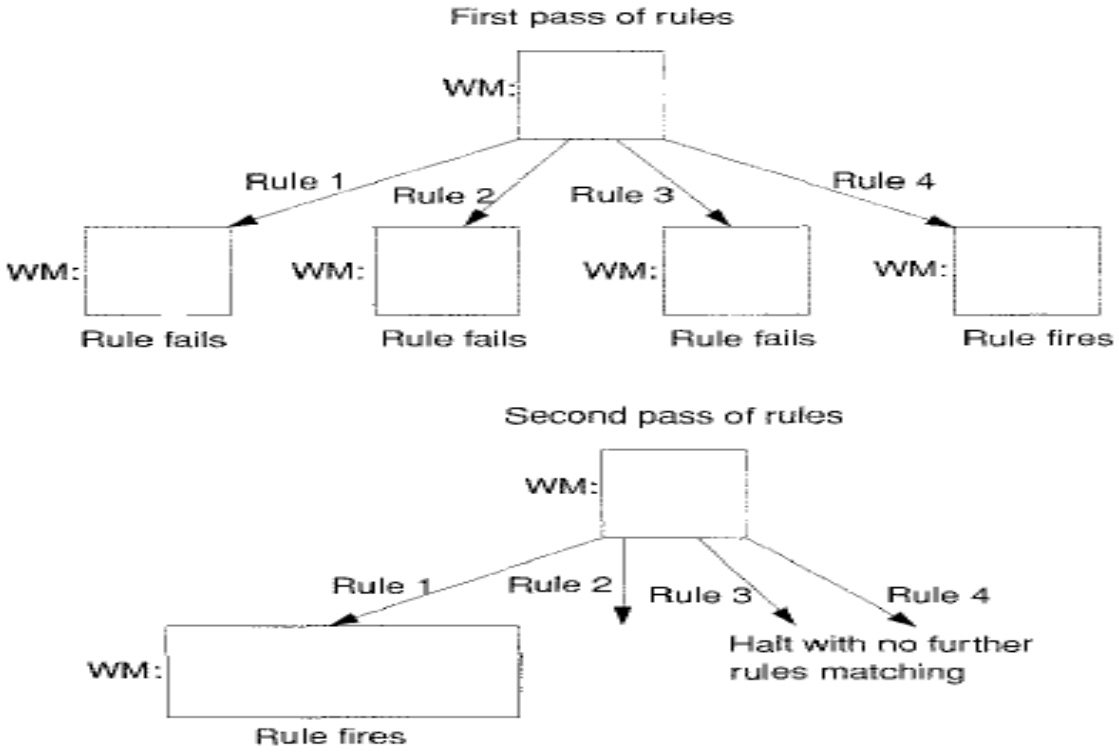



Working Memory
a
b
Production Rules
Rule 1
Rule2
Rule3
Rule4


Working Memory
a
b
L
Production Rules
Rule 1
Rule2
Rule3
Rule4


Production Rules
Rule 1
Rule2
Rule3
Rule4
Working Memory
a
b
L
J
Y1

2:


First pass of rules
WM:
Rule 1
Rule 2
Rule 3
Rule 4
WM:
WM:
WM:
WM:
Rule fails
Rule fails
Rule fails
Rule fires
Second pass of rules
WM:
Rule 1
Rule 2
Rule 3
Rule 4
Halt with no further rules matching
WM:
Rule fires

Programs that Work under Uncertainty factor Approximation Reasoning and Bipolar States

Logical Implications

- Simple Implication

ct(c) = ct(e) * ct(imp)

- AND Implication

ct(c) = min(ct(e1), ct(e2)) * ct(imp)

- OR Implication

ct(c) = max(ct(e1), ct(e2)) * ct(imp)

Bipolar Calculation Values

- If ct1(c) is +ve and ct2(c) is +ve (+ +) then

  Ct(c) = (ct1(c) + ct2(c)) - (ct1(c) * ct2(c))

- If ct1(c) is -ve and ct2(c) is -ve (- -) then

  Ct(c) = (ct1(c) + ct2(c)) + (ct1(c) * ct2(c))

- If [ct1(c) is +ve and ct2(c) is -ve (+ -)] or

  [ct1(c) is -ve and ct2(c) is +ve (- +)] then

  Ct(c) = (ct1(c) + ct2(c)) / (1-min(abs(ct1(c))), (abs(ct2(c)))

Reversible and non reversible Rules

Reversible

- If ct(c) is -ve and prefaced by not then Ct(c) is +ve
- If ct(c) is +ve and prefaced by not then Ct(c) is -ve

Non reversible

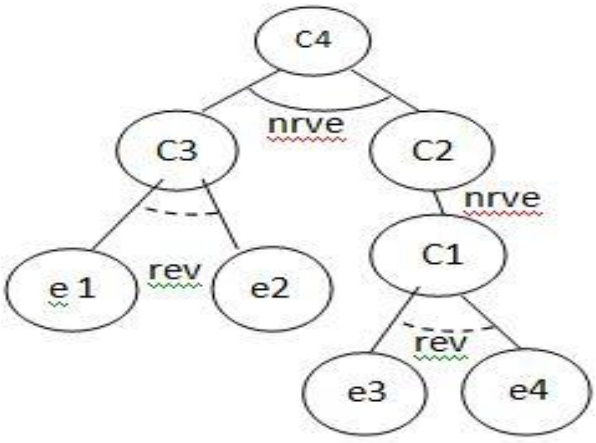
- If ct(c) is -ve and prefaced by not then Ct(c) is +ve
- If ct(c) is +ve and prefaced by not then Ct(c) = 1- (+ve)

Knowledge Base

- hypothesis_node(C).
- terminal_node(e).
- imp(logic op, rule type, conclusion name, left condition sign, left condition name, right condition sign, right condition name, imp value)

Examples:

Consider the following inference network, then answer the items below:



Certainty node (Ci) → non-terminal nodes

Evidence node (ei) →Terminal nodes

e1= 0.3 e2= 0.4 e3= 0.8 e4= 0.6

the implication value (imp) is equal to 0.5 for all rules.

1. Write the knowledge base of the given inference network.

2. Calculate the certainty factor for the node **C4**.

1.

hypothesis-node(C4).

terminal-node(e1).
terminal-node(e2).
terminal-node(e3).
terminal-node(e4).

imp(o, rev, C1, pos, e3, pos, e4, 0.5).
imp(o, rev, C3, pos, e1, pos, e2, 0.5).
imp(s, nrev, C2, pos, C1, __, __, 0.5).
imp(a, nrev, C4, pos, C3, pos, C2, 0.5).

2.

ct(C1) = max(e3, e4) * 0.5
= 0.8 * 0.5
= 0.4

ct(C3) = max(e1, e2) * 0.5
= 0.4 * 0.5
= 0.2

$$ct(C2) = C1 * 0.5$$
$$= 0.4 * 0.5$$
$$= 0.2$$

$$ct(C4) = \min(C3, C2) * 0.5$$
$$= 0.2 * 0.5$$
$$= 0.1$$

Systems that Explain their Actions
The HOW & WHY Facilities

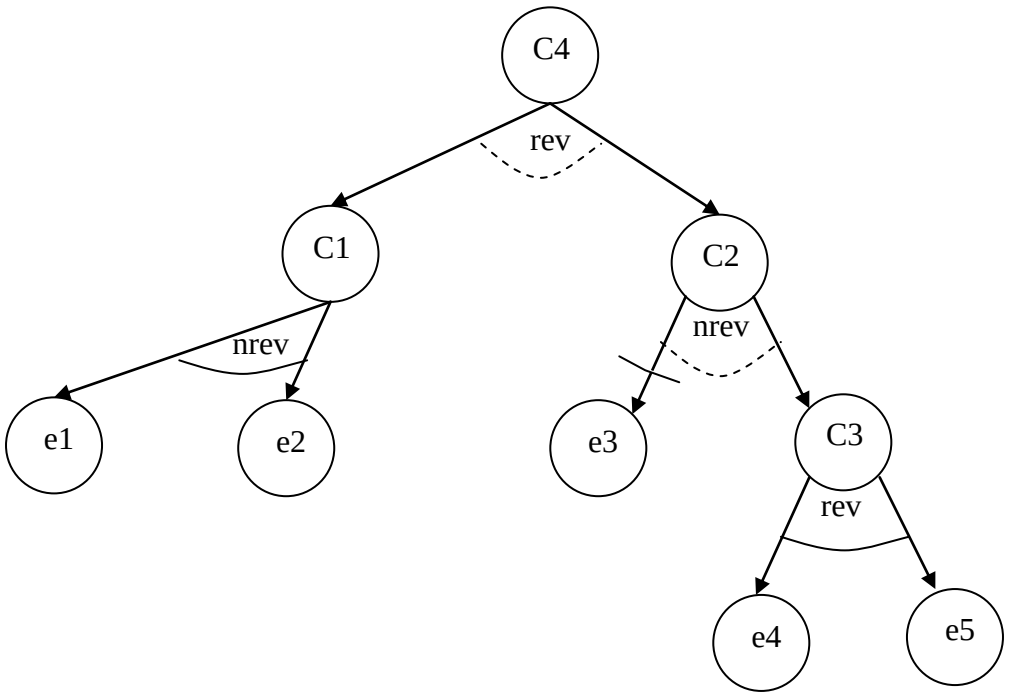
Consider the following Inference Network (fuzzy net)

If e1 and e2 then C1 (imp= 0.8) nrev

If not(e3) or C3 then C2 (imp= 0.9) nrev

If e4 and e5 then C3 (imp= 0.8) rev

If C1 or C2 thenC4 (imp= 0.8) rev



1- Answering WHY Questions

S: Type w(why) or give the certainty for node e4

U: w

S: Attempting to establish c3 via the implication

e4 and e5 → c3

Type w(why) or give the certainty for node e4

U: w

S: Attempting to establish c2 via the implication

not e3 or c3 → c2

Type w(why) or give the certainty for node e4

U: w

S: Attempting to establish c4 via the implication

c1 or c2 → c4

Type w(why) or give the certainty for node e4

U: 0.85

2- Knowledge Base

hypothesis-node (C4).

terminal-node(e1).

terminal-node(e2).

terminal-node(e3).

terminal-node(e4).

terminal-node(e5).

imp(o, rev, C4, pos, C1, pos, C2, 0.8).

imp(a, nrev, C1, pos, e1, pos, e2, 0.8).

imp(o, nrev, C2, neg, e3, pos, C3, 0.9).

imp(a, rev, C3, pos, e4, pos, e5, 0.8).

3- Why Stack Description

Step 0

The stack is empty

Step1

imp(o, r, C4, pos, C1, pos, C2, 0.8).

Step2

imp(a, n, C1, pos, e1, pos, e2, 0.8).

imp(o, r, C4, pos, C1, pos, C2, 0.8).

Step3

imp(o, r, C4, pos, C1, pos, C2, 0.8).

Step4

imp(o, n, C2, neg, e3, pos, C3, 0.9).

imp(o, r, C4, pos, C1, pos, C2, 0.8).

Step5

imp(a, r, C3, pos, e4, pos, e5, 0.8).

imp(o, n, C2, neg, e3, pos, C3, 0.9).

imp(o, r, C4, pos, C1, pos, C2, 0.8).

Step6

imp(o, n, C2, neg, e3, pos, C3, 0.9).

imp(o, r, C4, pos, C1, pos, C2, 0.8).

Step7

imp(o, r, C4, pos, C1, pos, C2, 0.8).

Step 8

The stack is empty

4- Answering HOW Question

S: Type h(how) nodename, or c (to continue).

U: h c4

S: Concluded c4 with certainty of 0.68 from

c1 or c2 → c4

The rule is reversible

Certainty of c1 is 0.6

Certainty of c2 is 0.85

The certainty of the implication is 0.8

Used alone the rule suggests a certainty of 0.68

Type h(how) nodename, or c (to continue).

U: h c1

S: Concluded c1 with certainty of 0.6 from

e1 or e2 → c1

The rule is reversible

Certainty of e1 is 0.75

Certainty of e2 is 0.65

The certainty of the implication is 0.8

Used alone the rule suggests a certainty of 0.6

Type h(how) nodename, or c (to continue).

Complete Example (Uncertainty Factor Calculations & Explanation Processor)

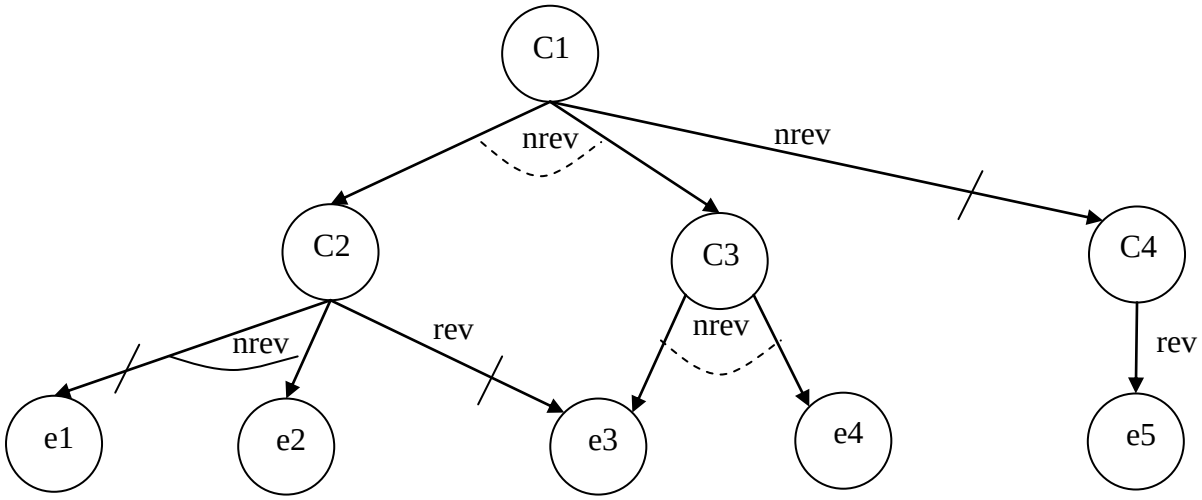
Consider the inference network bellow, then answer the following items:

1. Calculate the certainty factor for all nodes.

2. Write the knowledge base of the given inference network.

3. Show the contents of WHY stack when the system asks the user about the certainty value of node e4.

4. Describe the HOW explanation mechanism when the user asks H C3. What is the system response?



e1 = -0.2 e2 = 0.3 e3 = 0.8 e4 = -0.7 e5 = 0.4 and the implication value (imp) for each rule is equal to 0.5

The solution

1.

Ct1(C2) = min(not(e1), e2) * 0.5

Ct1(C2) = min(0.2, 0.3) * 0.5

Ct1(C2) = 0.1

Ct2(C2) = not(e3) * 0.5

Ct2(C2) = -0.8 * 0.5

Ct2(C2) = -0.4

Ct(C2) = (ct1(C2) + ct2(C2)) / (1-min(abs(ct1(C2))), (abs(ct2(C2)))

Ct(C2) = (0.1 + -0.4) / (1-min((0.1), (0.4)))

Ct(C2) = -0.3 / 0.9

Ct(C2) = -0.33

Ct(C3) = max(e3, e4) * 0.5

Ct(C3) = 0.8 * 0.5

Ct(C3) = 0.4

Ct(C4) = e5 * 0.5

Ct(C4) = 0.4 * 0.5

Ct(C4) = 0.2

Ct1(C1) = max(C2, C3) * 0.5

Ct1(C1) = 0.4 * 0.5

Ct1(C1) = 0.2

Ct2(C1) = not(C4) * 0.5

Ct2(C1) = 0.8 * 0.5

Ct2(C1) = 0.4

Ct(C1) = (ct1(C1) + ct2(C1)) - (ct1(C1) * ct2(C1))

Ct(C1) = (0.2 + 0.4) – (0.2 * 0.4)

Ct(C1) = 0.6 – 0.08

Ct(C1) = 0.52

2.

hypothesis-node (C1).

terminal-node(e1).

terminal-node(e2).

terminal-node(e3).

terminal-node(e4).

terminal-node(e5).

imp(o, nrev, C1, pos, C2, pos, C3, 0.5).

imp(o, nrev, C1, neg, C4, _, _, 0.5).

imp(a, nrev, C2, pos, e1, pos, e2, 0.5).

imp(s, rev, C2, pos, e3, _, _, 0.5).

imp(a, nrev, C3, pos, e3, pos, e4, 0.5).

imp(s, rev, C4, pos, e5, _, _, 0.5).

3.

step 0

The stack is empty

step 1

imp(o, nrev, C1, pos, C2, pos, C3, 0.5)

step2

imp(a, nrev, C2, pos, e1, pos, e2, 0.5)

imp(o, nrev, C1, pos, C2, pos, C3, 0.5)

step 3

imp(o, nrev, C1, pos, C2, pos, C3, 0.5)

step 4

imp(s, rev, C2, pos, e3, _, _, 0.5)

imp(o, nrev, C1, pos, C2, pos, C3, 0.5)

step 5

imp(o, nrev, C1, pos, C2, pos, C3, 0.5)

step 6

imp(a, nrev, C3, pos, e3, pos, e4, 0.5)

imp(o, nrev, C1, pos, C2, pos, C3, 0.5)

4.

S: Type h(how) nodename, or c (to continue).

U: h C3

S: Concluded C3 with certainty of 0.1 from

e3 and e4 → C3

The rule is non reversible

Certainty of e3 is 0.3

Certainty of e4 is 0.2

The certainty of the implication is 0.5

Used alone the rule suggests a certainty of 0.1

Type h(how) nodename, or c (to continue).

Homework / Consider the following Production rules:

if not(e3) or e4 then c1	(imp = 1.0)	nrev
if not(e1) and not(e2) then c2	(imp = 0.8)	rev
if c1 or e5 then c3	(imp = 0.7)	nrev
if not(e6) then c4	(imp = 0.9)	nrev
if e7 and e8 then c5	(imp = 0.8)	nrev
if not(e9) then c5	(imp = 0.9)	rev
if c2 then c6	(imp = 0.9)	rev
if c3 then c6	(imp = 0.9)	nrev
if c4 and c5 then c6	(imp = 0.85)	nrev

e1= 0.2 e2= -0.2 e3= -0.2 e4= 0.7 e5= -0.5 e6= -0.8 e7= 0.8

e8= 0.8 e9= -0.7

1. Calculate the certainty factor for all nodes.

2. Write the knowledge base of the drawing inference network.

3. Show the contents of WHY stack when the system asks the user about the certainty value of node e7.

4. Describe the HOW explanation mechanism when the user asks H C6. What is the system response?

Approximate Reasoning (Structure of the FUZZYNET Program)

```
driver:- hypothesis-node(X), allinfer(X, Ct),
        write("The certainty for ", X, "is", Ct), nl, fail.

allinfer(Node, Ct):- findall(C1, infer(Node, C1), Ctlist),
                     supercombine(Ctlist, Ct).

/*A simple implication */
infer(Node, Ct):-
        imp(s, Use, Node, Sign, Node2, _, _, C1),
        allinfer(Node2, C2),
        find_multiplier(Sign, Mult, dummy, 0), CS = Mult * C2,
        qualifier(Use, CS, Qmult), Ct = CS * C1 * Qmult.
/* An implication with an AND in the Premise */
 infer(Node1, Ct):-
        imp(a, Use, Node1, SignL, Node2, SignR, Node3, C1),
        allinfer(Node2, C2),
        allinfer(Node3, C3),
        find_multiplier(SignL, MultL, SignR, MultR),
        C2S = MultL * C2, C3S = MultR * C3,
        min(C2S, C3S, CX), qualifier(Use, CX, Qmult),
        Ct = CX * C1 * Qmult.

/* An implication with an OR in the Premise */
 infer(Node1, Ct):-
        imp(o, Use, Node1, SignL, Node2, SignR, Node3, C1),
        allinfer(Node2, C2),
        allinfer(Node3, C3),
```

```
        find_multiplier(SignL, MultL, SignR, MultR),
        C2S = MultL * C2, C3S = MultR * C3,
        max(C2S, C3S, CX), qualifier(Use, CX, Qmult),
        Ct = CX * C1 * Qmult.
infer(Node1, Ct):-
        terminal_node(Node1), evidence(Node1, Ct),!.
infer(Node1, Ct):-
        terminal_node(Node1)
        write("What is the certainty for node", Node1),
        nl, readreal(Ct), asserta(evidence(Node1, Ct)),!.
```

/* This is used for simple implication */

```
find_multiplier(pos, 1, dummy, 0).
find_multiplier(neg, -1, dummy, 0).
```

/* This is used for AND and OR implications */

```
find_multiplier(pos, 1, pos, 1).
find_multiplier(pos, 1, neg, -1).
find_multiplier(neg, -1, pos, 1).
find_multiplier(neg, -1, neg, -1).

supercombine([Ct], Ct):-!.
supercombine([C1, C2], Ct):- combine([C1, C2], Ct), !.
supercombine([C1, C2|T], Ct):- combine([C1, C2], C3), append([C3], T,
TL), nsupercombine(TL, Ct), !.

combine([-1, 1], 0).
combine([1, -1], 0).
```

```
Combine([C1, C2], Ct):- C1 >= 0, C2>= 0, Ct = C1 + C2 - C1 * C2.

Combine([C1, C2], Ct):- C1 < 0, C2< 0, Ct = C1 + C2 + C1 * C2.

combine([C1, C2], Ct):- C1 < 0, C2 >= 0, absvalue(C1, Z1), absvalue(C2, Z2),
                        min(Z1, Z2, Z3), Ct = (C1 + C2) / (1 – Z3).

combine([C1, C2], Ct):- C2 < 0, C1 >= 0, absvalue(C1, Z1), absvalue(C2, Z2),
                        min(Z1, Z2, Z3), Ct = (C1 + C2) / (1 – Z3).

absvalue(X, Y):- X = 0, Y = 0, !.
absvalue(X, Y):- X > 0, Y = X, !.
absvalue(X, Y):- X < 0, Y = -X, !.

qualifier(Use, C, Qmult):- Use = “r”, Qmult = 1, !.
qualifier(Use, C, Qmult):- Use = “n”, C >= 0, Qmult = 1, !.
qualifier(Use, C, Qmult):- Use = “n”, C < 0, Qmult = 0, !.
```

System that Explain their Actions

Explanation Mechanism

/* For and implication, the other in the same manner */

```
infer(Node1, Ct):-
        imp(a, Use, Node1, SignL, Node2, SignR, Node3, C1),
        assserta(dbimp(a, Use, Node1, SignL, Node2, SignR,
        Node3, C1)),
        assserta(tdbimp(a, Use, Node1, SignL, Node2, SignR,
        Node3,C1)),
        allinfer(Node2, C2),
        allinfer(Node3, C3),
        find_multiplier(SignL, MultL, SignR, MultR),
        C2S = MultL * C2, C3S = MultR * C3,
        min(C2S, C3S, CX), qualifier(Use, CX, Qmult),
        Ct = CX * C1 * Qmult,
        assertz(infer_summary(
        imp(a, Use, Node1, SignL, Node2, SignR, Node3, C1), Ct)),
        retract(dbimp(a, Use, Node1, SignL, Node2, SignR, Node3, C1)),
        retract(tdbimp(a, Use, Node1, SignL, Node2, SignR, Node3, C1)).
```

/* How Facility Sub Program */

```
Exsys_driver :- getallans, showresults,!.

    Getallans :-  not(prepare_answer).
    Prepare_answer :- answer(X, Y), fail.
```

```
answer(X, Y) :- hypothesis_node(X), allinfer(X, Y),
                assert(danswer(X, Y)).

Showresults :- not(displayall).
    displayall :- diplay_one_answer, fail.
    diplay_one_answer :- danswer(X, Y), clearwindow,
                         write("For this hypothesis:"), nl,
                         write(" ", X),nl, write("The certainty is:",
                         Y),nl, nl,
                         not(how_describer(X)).

how_describer(Node) :- repeat, nl,
                       write("Type h(how) nodename, or c(to continue),"),
                       nl, readln(Reply), nl, how_explain(Reply),!.

how_explain(Reply) :- Reply = "c".
how_explain(Reply) :- fronttoken(Reply, _, X1), fronttoken(X1, X, _),
                      infer_summary(imp(_, _, X, _, _, _, _, _), _),
                      clearwindow,!,
                      write("The rule(s) that bear upon this conclusion
                      are:"),
                      nl, nl, infer_summary(imp(A, A1, X, R, S, C, D,
                      E),F),
                      write("Concluded: ", X), nl, gettype(A, Z),
                      write("from an ", Z), nl, write(" premise 1 was:
                      ",S), nl,
                      write(" premise 2 was: ",D), nl,
                      write("The certainty from use of this rule alone
                      was: ",F),
```

```
                    nl, nl, fail.

how_explain(Reply) :- fronttoken(Reply, _, X1), fronttoken(X1, X, _),
                      terminal_node(X), evidence(X, C),
                      write("You told me that: "), nl, write(" ", X), nl,
                      write("has a certainty of: ",C), nl, fail.
```

/* Why Facility Sub Program */

```
infer(Node, Ct) :- terminal_node(Node), evidence(Node, Ct), !.
infer(Node, Ct) :- terminal_node(Node), repeat, nl,
                   write("Type w(why) or give the certainty for node ",
                   Node), nl, readln(Reply),
                   reply_to_input(Node, Reply, Ct), !.

reply_to_input(Node, Reply, Ct) :- not(isname(Reply)),
                                   adjuststack,
                                   str_real(Reply, CT),
                                   asserta(evidence(Node,Ct)),!.
reply_to_input(_, Reply, _) :- isname(Reply), Reply = "w", nl,
                               dbimp(U, V, R, S, S1, X, Y, Y1),
                               why_describer(U, V, R, S, S1, X, Y, Y1),
                               retract(dbimp(U, V, R, S, S1, X, Y, Y1)),
                               putadjustflag,
                               pauser, !, fail.

why_describer(U, U1, V, R, S, X, Y, Z) :- clearwindow, nl, U <>"s",
                                          gettype(U,UU),
        write("I am trying to use an inference rule of the type "),
        nl, write(UU), write(", to support the conclusion: "), nl,
```

```
        write("   ", V), nl, write("Premise 1 is: ",S), nl, getmode(R, RR),
        write("  This premise will be used ", RR), nl,
       write("Premise 2 is: ",Y),
        nl, getmode(X, XX), nl,
       write("  This premise will be used ", XX), nl,
        write("The certainty of the implication is: ", Z), nl, !.

why_describer("s", V1, V, R, S, X, Y, Z) :- clearwindow, nl,
        write("I am trying to use an inference rule of the type "), nl,
        write("simple implication, to support the conclusion: "), nl,
        write("   ", V), nl, write("premise 1 is: ", S), nl, getmode(R, RR),
        write("  This premise will be used ", RR), nl
        write("The certainty of the implication is: ", Z), nl, !.

gettype("a", "and implication").
gettype("o", "or implication").
gettype("s", "simple implication").

getmode("pos", "as you see it.").
getmode("neg", "prefaced by not.").
```

References:

1. Daniel H. .Marcellus,***" Expert Systems Programming in Turbo Prolog"***, prentice Hall (New Jersey).
2. George F. Luger, "***Artificial Intelligence Structures and Strategies for Complex Problem Solving",*** Pearson Education Asia (Singapore), Sixth edition.