3rd Class

Expert Systems

نظم خبيرة

أستاذ المادة : أ.م.د. حسنين سمير عبد الله

University of Technology / Computer Sciences Department
Artificial Intelligence Branch
Expert Systems
Dr. Hasanen S. Abdullah

References

1. **Daniel H. Marcellus, "Expert Systems Programming in Turbo prolog", Prentice Hall, 1994.**
2. **Goerge F. luger, "Artificial Intelligence Structures and Strategies for Complex problem Solving", Pearson Education Asia, 2009.**

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 system programs 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.

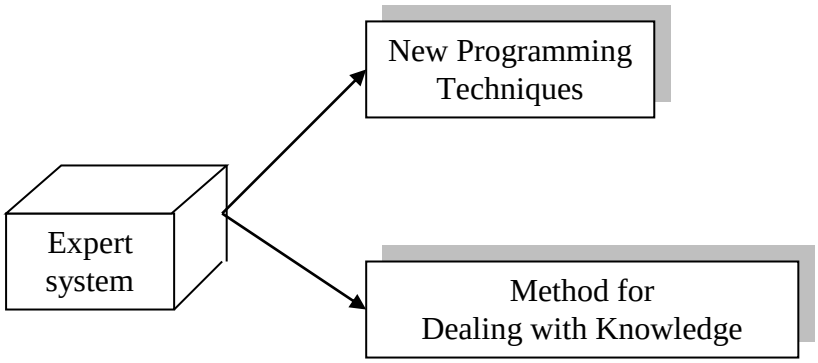


Figure (1) the vectors of expert system development

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 identify and correct malfunctions in locomotives.
- 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 figure out a chemical compound's molecular structure from experiments with mass spectral data and nuclear magnetic resonance.
- 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.

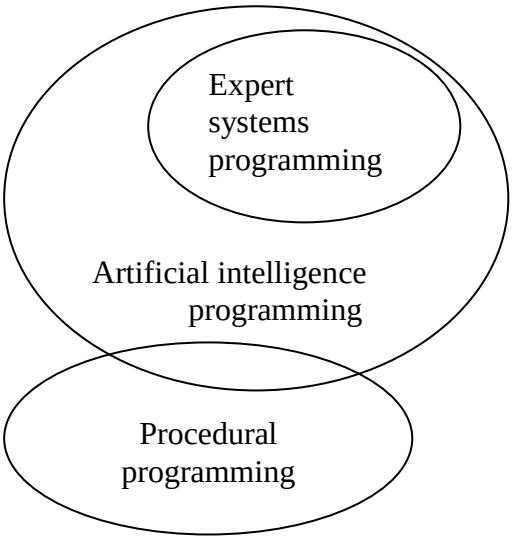
In some cases, this strategy made the program more human oriented. In others, it allowed the program to make better judgments.

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 :

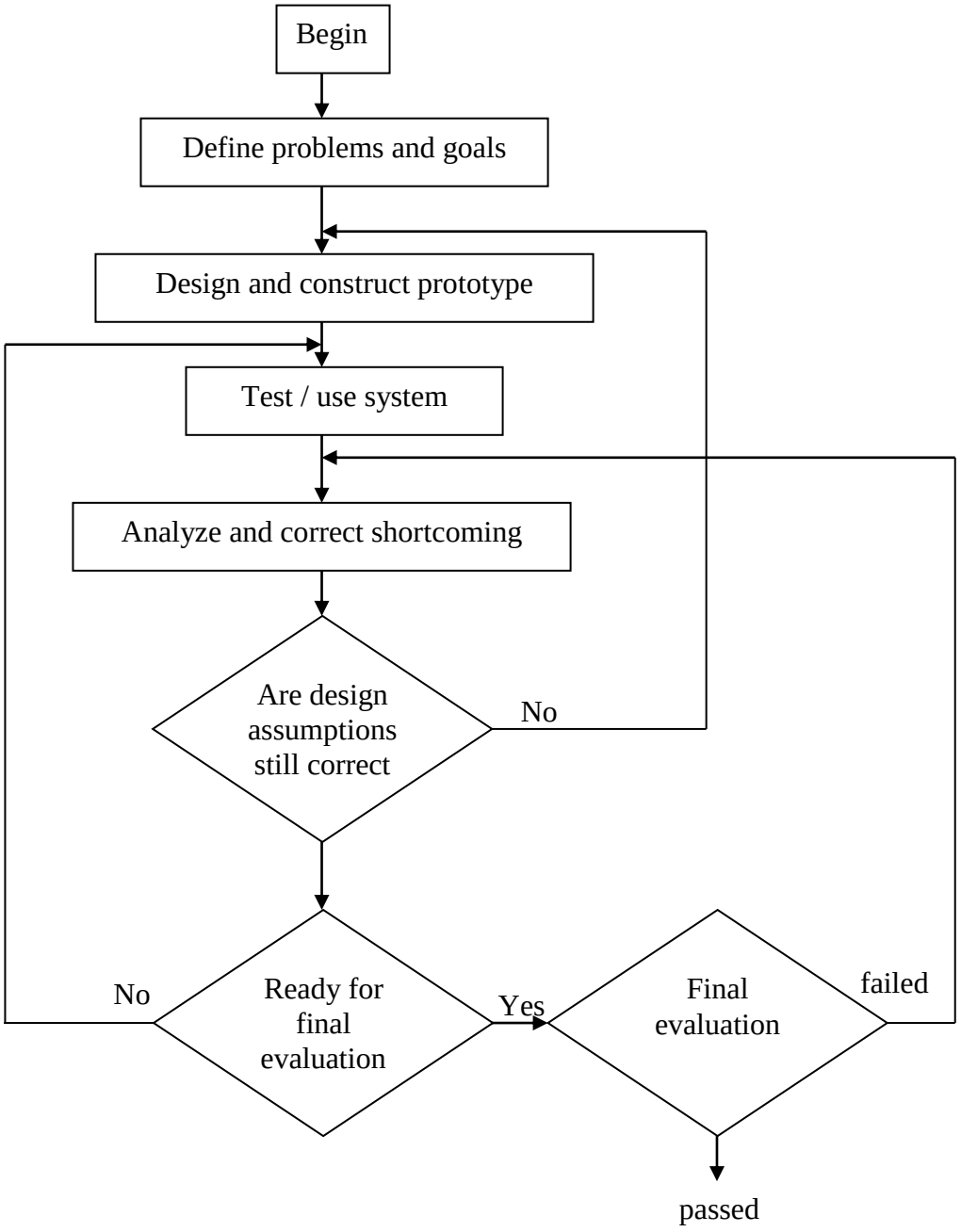
procedure calls

sequential execution

recursion

4. Expert System, Development Cycle

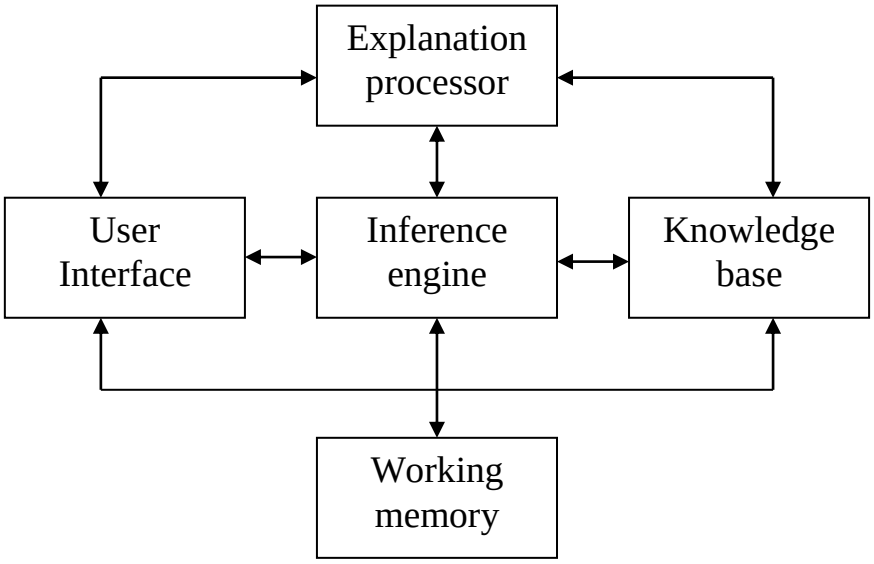
The explanation mechanism allows the program to explain its reasoning to the user, these explanations include justification for the system's conclusions, explanation of why the system needs a particular piece of data. Why questions and How questions. Figure (3) below shows the exploratory cycle for rule based expert system.



Figure(3) The exploratory cycle for expert system

5. Expert System Architecture and Components

The architecture of the 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.

Systems Based on Simple Search and Pattern Recognition

Text Recognition System

(An AI Program for Psychology Counseling)

Pattern in Sentence

From human	Machine Responds
…..I fell…..	How long have you felt that way?
…..anger…..	How are you feeling right now?
…..father…..	Tell me more about your family.
…..friendly…..	Better friends help you when you need.
…..hell…..	Please do not use word like that.
…..high performance…..	That sounds significant

```
helpme :-
        write("Speak up, what is your problem?"), nl,
        repeat,
        getclause(L),
        makeans(L),
        fail.

makeans(L) :-
                recognize(L,1), write ("How long have you felt that way?"), nl, !.
makeans(L) :-
                recognize(L,2), write ("Tell me more about your family"), nl, !.
makeans(L) :-
                recognize(L,3), write ("better friends help you when you need."), nl, !.
makeans(L) :-
                recognize(L,4), write ("Please do not use words like that."), nl, !.
```

```
makeans(L) :-
        recognize(L,5), write ("Please be more explicit."), nl, !.
makeans(L) :-
        recognize(L,6), write ("This gives you additional respecting."), nl, !.
makeans(L) :-
        recognize(L, 7), write ("How are you felling right now?"), nl, !.
makeans(L) :-
        recognize(L,8), write ("Too many mind games."), nl, !.
makeans(L) :-
        recognize(L,9), write ("Earlier you mentioned your father."), nl, !.
makeans(L) :-
        recognize(L,10), write ("Tell me more."), nl, !.


recognize(L, 1) :- contains([i, feel], L).
recognize(L, 2) :- contains([father], L), assert(father).
recognize(L, 3) :- contains([friendly], L)
recognize(L, 4) :- contains([hell], L).
recognize(L, 5) :- L= [yes]; L=[no].
recognize(L, 6) :- contains([high, performance], L).
recognize(L, 7) :- contains([sadness], L); contains([anger], L).
recognize(L, 8) :- contains([complex], L) ; contains([fixation], L).
recognize(L, 9) :- father.
recognize(_, 10).
```

Search with Heuristic Embedded in Rules

Student Advisor System

/* Set of Facts */

given_now(logic design).
given_now(Mathematics).
given_now(prolog language).
given_now(computation theory).
given_now(data structure).

required(prolog).
required(logic design).
required(artificial intelligence).
required(expert systems).
required(machine learning).

elective(computer graphics).
elective(object oriented programming).
elective(data security).
elective(web programming).
elective(operations researches).

waived(digital signal processing).
waived(image processing).
waived(information systems principles).
waived(software engineering).
waived(data hiding).

impreq(object oriented programming, c++).
impreq(prolog language, logic design).
impreq(artificial intelligence, prolog language).
impreq(expert systems, artificial intelligence).

```
impreq(computer architecture, logic design).

passed(logic design).
passed(prolog language).
passed(artificial intellgence).
passed(mathematics).
passed(data structure).

pos_req_course(X) :-
                        required(X),
                        given_now(X),
                        not(done_with(X)),
                        have_preq_for(X).

pos_elec_course(X) :-
                        elective(X),
                        given_now(X),
                        not(done_with(X)),
                        have_preq_for(X).

done_with(X) :- waived(X).
done_with(X) :- passed(X).
all_done_with(L) :- findall(X, done_with(X), L).

have_preq_for(X) :-
                        all_preq_for(X, Z),
                        all_done_with(Q),
                        subset(Z, Q).

all_preq_for(X, Z):-
                        findall(Y, preq(X, Y), Z).
preq(X,Y):- impreq(X, Y).
preq(X, Y):- impreq(X, W), preq(W, Y).
```

Systems Based on Simple Search and Pattern Recognition

Chemical Synthesis System

```
domains

rxnlist = reactions*.

reactions = rxn(symbol, ls, integer, integer).

ls = symbol*.

chemicalList= chemicalForm*.

chemicalForm= chemical(symbol, rxnList, integer, integer).

Li= integer*.

predicates

rxn(symbol, ls, integer, integer).

rawmaterial(symbol, integer, integer).

chemical(symbol, rxnlist, integer, integer).

all_chemical(symbol, chemicalList).

best_chemical(symbol, chemicalForm).

one_chemical(symbol, chemicalForm).

append(rxnlist, rxnlist, rxnlist).

min(chemicalList, chemicalForm).

run(symbol).

clauses

rxn(a, [b1, c1], 12, 60).

rxn(b1, [d1, e1], 5, 45).

rxn(c1, [f1, g1], 3, 15).


rxn(a, [b2, c2], 10, 50).

rxn(b2, [d2, e2], 2, 20).

rxn(c2, [f2, g2], 6, 30).
```

```
rawmaterial(d1, 2, 0).

rawmaterial(e1, 0, 0).

rawmaterial(f1, 2, 0).

rawmaterial(g1, 0, 0).


rawmaterial(d2, 0, 0).

rawmaterial(e2, 1, 0).

rawmaterial(f2, 1, 0).

rawmaterial(g2, 0, 0).


chemical(Y, [], Cost, Time):- rawmaterial(Y, Cost, Time).

chemical(Y, L, Ct, T):-

                rxn(Y, [X1, X2], C, T1), chemical(X1, L1, C1, T2), chemical(X2, L2, C2, T3),

                append(L1, L2, Q), Ct = C+C1+C2,

                T = T+T2+T3, append([rxn(Y, [X1, X2], C, T1)], Q, L).


best_chemical(Y, M):- all_chemical(Y, X), min(X, M).


all_ chemical(Y, X):- findall(S, one_chemical(Y, S), X).


one_chemical(Y, chemical(Y, L, Ct, T)):- chemical(Y, L, Ct, T).


append([], L, L):-!.

append([H|T], L, [H|T1]):- append(T, L, T1).


min([chemical(Y, L, Ct, T)], chemical(Y, L, Ct, T)).
```

```
min([chemical(Y, L, Ct, Time)|T], chemical(Y, L, Ct, Time)):-
                    min(T, chemical(Y1, L1, C1, Time1)), Ct <= C1.
min([chemical(Y, L, Ct, Time)|T], chemical(Y, L2, Ct2, Time2)):-
                    min(T, chemical(Y, L2, Ct2, Time2)), Ct2 <= Ct.

run(X):- write(" chemical synthesis is:"), nl, chemical(X, L, Cost, Time),
        write(L, "\n with total cost =", Cost, " Time =", Time), nl, fail.
run(X):- write("\n Best chemical synthesis:"), nl, best_chemical(X, Y), write(Y), nl.
```

Goal: run(a).

chemical synthesis:

[rxn("a", ["b1", "c1"], 12, 60), rxn("b1", ["d1", "e1"], 5, 45), rxn("c1", ["f1", "g1"], 3, 15)]

with total cost = 24 time = 120

[rxn("a", ["b2", "c2"], 10, 50), rxn("b2", ["d2", "e2"], 2, 20), rxn("c2", ["f2", "g2"], 6, 30)]

with total cost = 20 time = 100

best chemical synthesis :

chemical("a", [rxn("a", ["b2", "c2"], 10, 50) rxn("b2", ["d2", "e2"], 2, 20), rxn("c2", ["f2", "g2"], 6, 30)], 20, 100)

Controlling the Reasoning Strategy

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

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

```
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.

Systems That Depend on Reasoning under Uncertainty

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, Node1, 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),
                                supercombine(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_aoe_answer, fail.
        diplay_aoe_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.").
```

Natural Language Interfaces
Informal Method (Dictionary Building)

```
clauses

/* set of facts */


Own(John, B.Sc, 1980, Scientific).

Own(Roy, M.Sc, 1984, technique).

Own(Tomy, B.Sc, 1982, Engineer).

Own(Har, Ph.D, 1978, Scientific).


reject("HOW").

reject("GO").

reject("ALL").

reject("FIND").

reject("THE").

reject("SOME").

reject("I").

reject("HAVE").


dsyn("B.Sc", "B. of Science").

dsyn("M.Sc", "Master of Science").

dsyn("Ph.D", "Philosophy of Doctorate").


docdriver:- repeat, nl, getquery(X), findref(X, Y),
            produceans(Y), fail.
```

```
getquery(Z):- write("please ask your question."),
                    nl, readln(Y), upper_lower(Y1, Y),
                    changeform(Y1, Z).

changeform(S, [H|T]):- fronttoken(S, H, S1), !, changeform(S1, T).
changeform(_, []):-!.

findref(X, Y):- memberof(Y, X), not(reject(Y)), !.


produceans(X):- own(X, X1, Y, Z), putflag,
                     write(X, "has", X1, "since the year", Y, "in", Z),nl.
produceans(X):- syn(X1, W), own(X, W, Y, Z), putflag,
                     write(X, "has", X1, "since the year", Y, "in", Z),nl.
produceans(_):- not(flag),
                    write("we have no information on that."), nl.
produceans(_):- remflag.

putflag:- not(flag), assert(flag),!.
putflag.

remflag:- flag, retract(flag),!.
remflag.

syn(Y,X):- dsyn(X, Y).
syn(Y,X):- dsyn(Y, X).

dsyn(Y,X):- concat(X, "S", Y).
dsyn(Y,X):- concat(X, "ES", Y).
dsyn(Y,X):- concat(X, "'S", Y).
```

Natural Language Interfaces
Formal Method

The people respect clever student.

Clever students can own respecting by their good works.

1. Build the Context Free Grammar for the above sentences.
2. Write a complete prolog program that parse the above sentences using the Context Free Grammar in step 1 .

1.

S → Np, Vp, Np / Np, Vp, Np, Pp

Np → det, noun / adj, noun / noun / det, adj, noun

Vp → verb / h.verb, verb

Pp → preposition, Np

2.

```
clauses
run:- readln(S), str_to_list(S, L), parse(L).
parse(L):- append(A1, A2, A3, _, L),
           np(A1),
           vp(A2),
           np(A3).

parse(L):- append(A1, A2, A3, A4, L),
           np(A1),
           vp(A2),
           np(A3),
           pp(A4).

np(X):- append(Y1, Y2, _, _, X),
        det(Y1),
        noun(Y2).
np(X):- append(Y1, Y2, _, _, X),
        adj(Y1),
        noun(Y2).
np(X):- append(Y1, Y2, Y3, _, X),
```

```
            det(Y1),
            adj(Y2),
            noun(Y3).
np(X):- noun(X).

vp(Z):- append(Y1, Y2, _, _, X),
            h.verb(Y1),
            verb(Y2).
vp(Z):- verb(Z).

pp(M):- append(W1, W2, _, _, M),
             preposition(W1),
             np(W2).

/ * set of Facts */
det([“the”]).     det([“their”]).
noun([“people”]).          noun([“student”]).
noun([“respecting”]).     noun([“work”]).
adj([“clever”]).              adj([“good”]).
verb([“respect”]).          verb([“own”]).
h.verb([“can”]).
preposition([“by”]).
```

Study Question 1

What heuristics would you use in solving these problems?

1. You are looking for a parking space in a moderately crowded parking lot.
2. You think a particular radio show you want to hear is on now, but you do not know where it is on the dial, and you have no other guidance such as a newspaper listing.
3. You are in a large office building. You are lost, and you want to find the personal office, but you are embarrassed to ask where it is.

Think about an elevator with the following controls: buttons for three floors, buttons to open and close the door, a sensor to see if the door is obstructed, a timer to time how long to leave it open, and single call buttons on each floor. Write a production system that would cause the elevator to operate in the conventional manner if the production system were controlling the operation. Atypical production would be:

If (timer_expired and door_is_open and door_not_obstructed) then (close_door)

Consider the category scheme to classify expert system. For each of the following, discuss if the example would be an expert system at all and, if so, what type:

1. A program to forecast the local weather.
2. A program to reason about what to do when your car will not started.
3. A program for a help-line service where the person answering the phone has to give advice about poisons that someone might have taken.
4. A program to predict what courses to give and how many sections to plan for in the next three semesters in a large college department.
5. A program to determine the best route for a salesperson to take on any given day to visit all his clients and use the minimum amount of gasoline that is possible.
6. A program to produce a 3-dimensional drawing of a house, given a textual description of the arrangement and dimensions of the rooms.

Write a small expert system program to construct optimal restaurant menus that follows the pattern of Student Advisor System.

The chemical synthesis program currently works with reactions like this:

X + y --→ z ……..with cost (c)

1. How would things have to be modified so that reactions like this one could be included in the reaction data base that the program knows about? r --→ s ..…..with cost (c)
   This is anticipating the type of reaction where you treat a chemical in a certain way (heating perhaps) and it turns into something else.
2. How would things have to be modified so that reactions like this one could be included? q + r + s --→ w …….with cost (c)
3. What modification would be necessary for the program to carry along two costs with each synthesis: One might be the reaction cost and the other the length of time the reaction took to complete.
4. What modification would be necessary for the program to include a function that carries the best synthesis among many syntheses?

Study Question 2

1. Show what would be required to add these two animals to both versions of BBF:
   - The camel, an ungulate with a hump.
   - The unicorn, an ungulate with a single horn.
2. By examining the listings of the BBF programs, calculate the average number of questions that will be asked to identify an animal in the forward chaining versions and in the backward chaining version.
3. Find a set of rules that describe what to do when your computer will not started. Organize the appropriate rules into both a backward chaining and forward chaining systems (version 1 & 2).

Study Question 3

1. Try to build again the structure of the fuzzy net program (certainty Program) that accept any arbitrary inference tree.

---

2. Given the following information:

index(“book1”, “OGOFF”, [“99”]).

index(“book1”, “EDLIN”, [“41-46”, “57”]).

index(“book2”, “EDLIN”, [“100-102”]).

index(“book7”, “EDLIN”, [“100”, “110”]).

index(“book1”, “EXE”, [“14-18”]).

index(“book1”, “ECHO”, [“35”, “146”]).

index(“book7”, “BNF”, [“51”, “55-56”]).

index(“book7”, “BNF”, [“30-31”]).

index(“book8”, “BNF”, [“109”, “130-148”]).

index(“book4”, “RERURSION”, [“56-78”]).

index(“book7”, “RECURSION”, [“119-125”]).

dsyn(“LOGOUT”, “LOGOFF”).

dsyn(“LOGIN”, “LOGON”).

dsyn(“BENEFIT”, “ADVANTAGE”).

dsyn(“PROCESSING”, “MANIPULATING”).

dsyn(“INTELLIGENT”, “SMART”).

dsyn(“FACT”, “REAL”).

reject(“HOW”).

reject(“ANY”).

reject(“ABOUT”). and so on

Write a complete prolog program to index the above information by using the Dictionary (informal) Natural Language Interface technique.

---

3. Which rules (in the chemical synthesis program) is adjusted when the user asks **how** after the program implementation? Write them.

---

4. Which rules (in the B.B.F program) is adjusted when the user asks **why** when the system asks for any feature? Write them.