الجامعة التكنولوجية

قسم علوم الحاسوب / فرع الذكاء الاصطناعي

المرحلة الثالثة ـ الكورس الثاني 2024- 2025

مادة الأنظمة الخبيرة / منهاج المادة العملي

أ.د. حسنين سمير عبدالله

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)

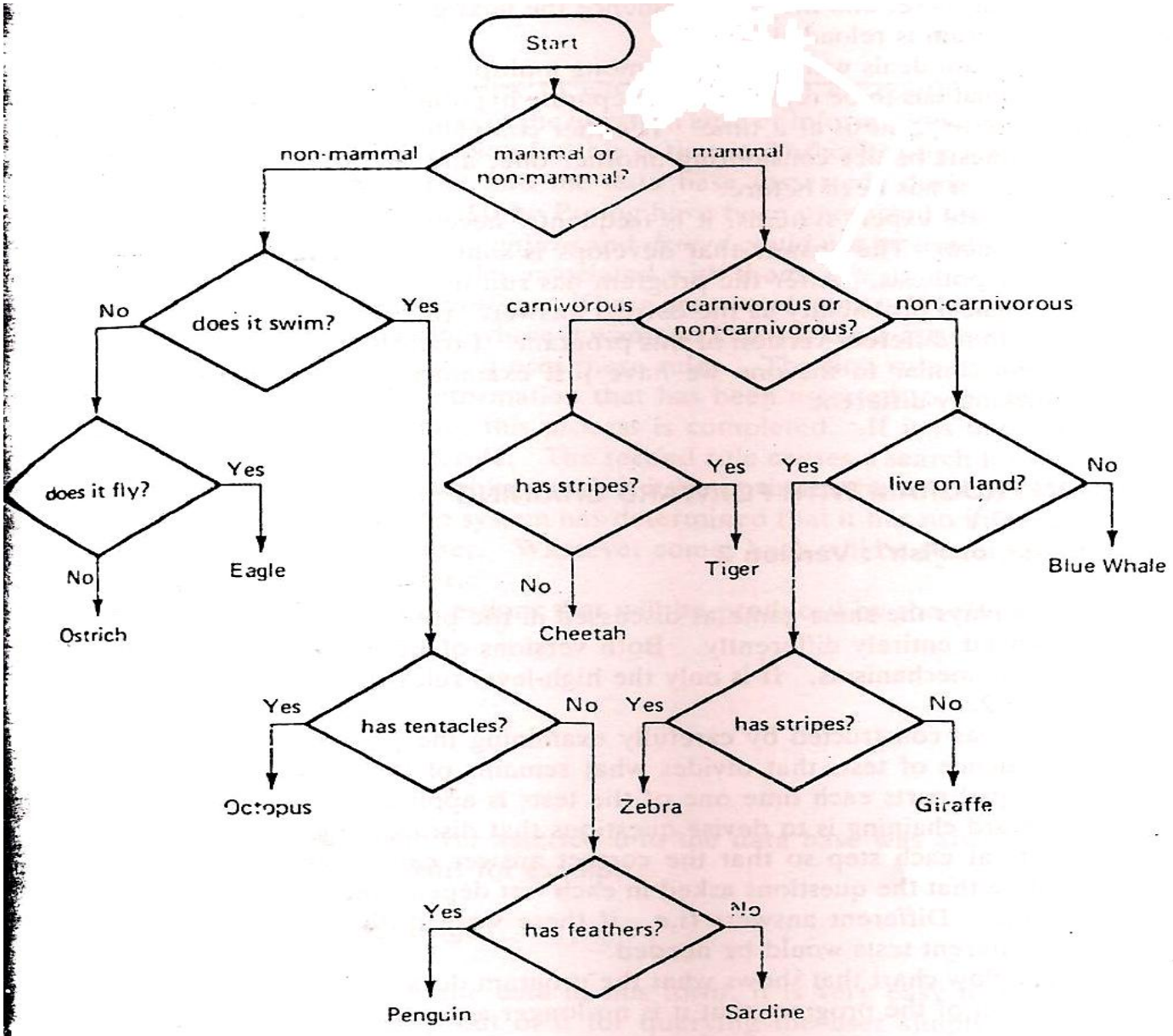
Many (properties) 1 (the target class)

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.



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.

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

```
database
have_found(symbol)
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.
```

Rule-Based Expert Systems

The Production System and Control Strategy in Problem Solving

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

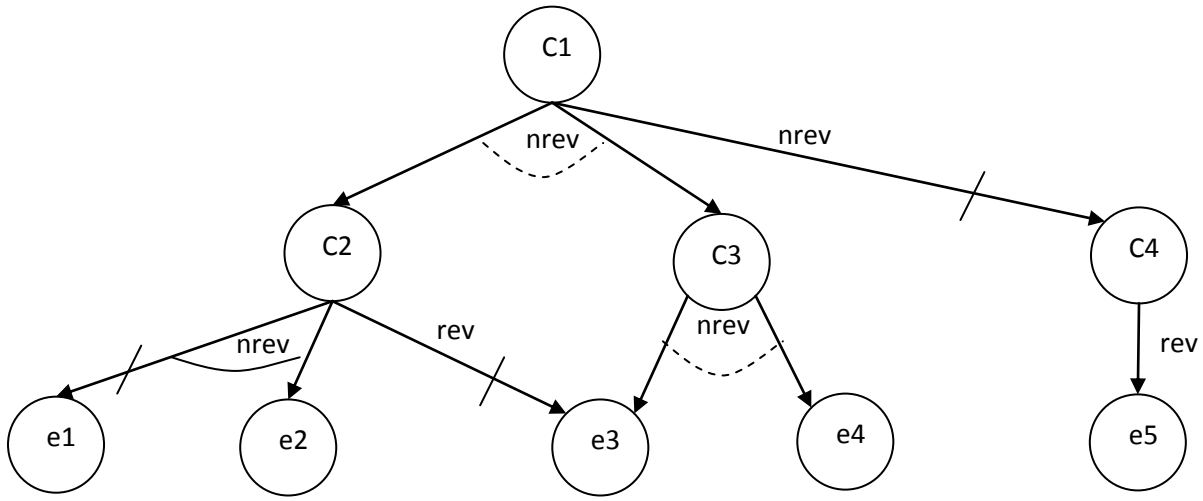
there is gas in the carburetor

then the engine is getting gas.

Programs that Work under Uncertainty factor

Approximation Reasoning and Bipolar States

Consider the inference network bellow,



/* Set of facts */

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).

/* 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.").
```