PROGRAM 17:

(PROLOG:STUDY OF PROLOG[MOTHER,FATHER])

|  |
| --- |
| parent( pam, bob). |
|  | parent( tom, bob). |
|  | parent( tom, liz). |
|  | parent( bob, ann). |
|  | parent( bob, pat). |
|  | parent( pat, jim). |
|  |  |
|  | female( pam). |
|  | female( liz). |
|  | female( ann). |
|  | female( pat). |
|  | male( tom). |
|  | male( bob). |
|  | male( jim). |
|  |  |
|  | offspring( Y, X) :- |
|  | parent( X, Y). |
|  | mother( X, Y) :- |
|  | parent( X, Y), |
|  | female( X). |
|  |  |
|  | grandparent( X, Z) :- |
|  | parent( X, Y), |
|  | parent( Y, Z). |
|  |  |
|  | sister( X, Y) :- |
|  | parent( Z, X), |
|  | parent( Z, Y), |
|  | female( X), |
|  | X \= Y. |
|  |  |
|  | predecessor( X, Z) :- % Rule pr1 |
|  | parent( X, Z). |
|  |  |
|  | predecessor( X, Z) :- % Rule pr2 |
|  | parent( X, Y), |
|  | predecessor( Y, Z). |

PROGRAM 18:

(PROLOG:DIETING SYSTEM BASED ON DISEASES)

|  |
| --- |
| domains |
|  | disease,indication,name = symbol |
|  |  |
|  | predicates |
|  | hypothesis(name,disease) |
|  | symptom(name,indication) |
|  |  |
|  | clauses |
|  | symptom(amit,fever). |
|  | symptom(amit,rash). |
|  | symptom(amit,headache). |
|  | symptom(amit,runn\_nose). |
|  |  |
|  | symptom(kaushal,chills). |
|  | symptom(kaushal,fever). |
|  | symptom(kaushal,hedache). |
|  |  |
|  | symptom(dipen,runny\_nose). |
|  | symptom(dipen,rash). |
|  | symptom(dipen,flu). |
|  |  |
|  |  |
|  | hypothesis(Patient,measels):- |
|  | symptom(Patient,fever), |
|  | symptom(Patient,cough), |
|  | symptom(Patient,conjunctivitis), |
|  | symptom(Patient,rash). |
|  |  |
|  | hypothesis(Patient,german\_measles) :- |
|  | symptom(Patient,fever), |
|  | symptom(Patient,headache), |
|  | symptom(Patient,runny\_nose), |
|  | symptom(Patient,rash). |

PROGRAM 19:

(PROLOG:MONKEY BANANA)

|  |
| --- |
| move(state(middle,onbox,middle,hasnot), |
|  | grasp, |
|  | state(middle,onbox,middle,has)). |
|  | move(state(P,onfloor,P,H), |
|  | climb, |
|  | state(P,onbox,P,H)). |
|  | move(state(P1,onfloor,P1,H), |
|  | drag(P1,P2), |
|  | state(P2,onfloor,P2,H)). |
|  | move(state(P1,onfloor,B,H), |
|  | walk(P1,P2), |
|  | state(P2,onfloor,B,H)). |
|  | canget(state(,,\_,has)). |
|  | canget(State1) :- |
|  | move(State1,\_,State2), |
|  | canget(State2). |

PROGRAM 20:

(PROLOG:FRUIT COLOURING AND BACKTRACKING)

|  |
| --- |
| colour(cherry, red). |
|  | colour(banana, yellow). |
|  | colour(apple, red). |
|  | colour(apple, green). |
|  | colour(orange, orange). |
|  | colour(X, unknown). |

PROGRAM 21:

(PROLOG:BFS)

|  |
| --- |
| %connected(+Start, +Goal, -Weight) |
|  | connected(1,7,1). |
|  | connected(1,8,1). |
|  | connected(1,3,1). |
|  | connected(7,4,1). |
|  | connected(7,20,1). |
|  | connected(7,17,1). |
|  | connected(8,6,1). |
|  | connected(3,9,1). |
|  | connected(3,12,1). |
|  | connected(9,19,1). |
|  | connected(4,42,1). |
|  | connected(20,28,1). |
|  | connected(17,10,1). |
|  |  |
|  | connected2(X,Y,D) :- connected(X,Y,D). |
|  | connected2(X,Y,D) :- connected(Y,X,D). |
|  |  |
|  | next\_node(Current, Next, Path) :- |
|  | connected2(Current, Next, \_), |
|  | not(member(Next, Path)). |
|  |  |
|  | breadth\_first(Goal, Goal, \_,[Goal]). |
|  | breadth\_first(Start, Goal, Visited, Path) :- |
|  | findall(X, |
|  | (connected2(X,Start,\_),not(member(X,Visited))), |
|  | [T|Extend]), |
|  | write(Visited), nl, |
|  | append(Visited, [T|Extend], Visited2), |
|  | append(Path, [T|Extend], [Next|Path2]), |
|  | breadth\_first(Next, Goal, Visited2, Path2). |

PROGRAM 22:

(PROLOG:MEDICAL DIAGNOSIS)

|  |
| --- |
| domains |
|  | disease,indication = symbol |
|  | Patient,name = string |
|  |  |
|  | predicates |
|  | hypothesis(string,disease) |
|  | symptom(name,indication) |
|  | response(char) |
|  | go |
|  | clauses |
|  |  |
|  | go :- |
|  | write("What is the patient's name? "), |
|  | readln(Patient), |
|  | hypothesis(Patient,Disease), |
|  | write(Patient,"probably has ",Disease,"."),nl. |
|  | go :- |
|  | write("Sorry, I don't seem to be able to"),nl, |
|  | write("diagnose the disease."),nl. |
|  |  |
|  | symptom(Patient,fever) :- |
|  | write("Does ",Patient," have a fever (y/n) ?"), |
|  | response(Reply), |
|  | Reply='y'. |
|  | symptom(Patient,rash) :- |
|  | write("Does ",Patient," have a rash (y/n) ?"), |
|  | response(Reply), |
|  | Reply='y'. |
|  | symptom(Patient,headache) :- |
|  | write("Does ",Patient," have a headache (y/n) ?"), |
|  | response(Reply), |
|  | Reply='y'. |
|  | symptom(Patient,runny\_nose) :- |
|  | write("Does ",Patient," have a runny\_nose (y/n) ?"), |
|  | response(Reply), |
|  | Reply='y'. |
|  | symptom(Patient,conjunctivitis) :- |
|  | write("Does ",Patient," have a conjunctivitis (y/n) ?"), |
|  | response(Reply), |
|  | Reply='y'. |
|  | symptom(Patient,cough) :- |
|  | write("Does ",Patient," have a cough (y/n) ?"), |
|  | response(Reply), |
|  | Reply='y'. |
|  | symptom(Patient,body\_ache) :- |
|  | write("Does ",Patient," have a body\_ache (y/n) ?"), |
|  | response(Reply), |
|  | Reply='y'. |
|  | symptom(Patient,chills) :- |
|  | write("Does ",Patient," have a chills (y/n) ?"), |
|  | response(Reply), |
|  | Reply='y'. |
|  | symptom(Patient,sore\_throat) :- |
|  | write("Does ",Patient," have a sore\_throat (y/n) ?"), |
|  | response(Reply), |
|  | Reply='y'. |
|  | symptom(Patient,sneezing) :- |
|  | write("Does ",Patient," have a sneezing (y/n) ?"), |
|  | response(Reply), |
|  | Reply='y'. |
|  | symptom(Patient,swollen\_glands) :- |
|  | write("Does ",Patient," have a swollen\_glands (y/n) ?"), |
|  | response(Reply), |
|  | Reply='y'. |
|  |  |
|  | hypothesis(Patient,measles) :- |
|  | symptom(Patient,fever), |
|  | symptom(Patient,cough), |
|  | symptom(Patient,conjunctivitis), |
|  | symptom(Patient,runny\_nose), |
|  | symptom(Patient,rash). |
|  | hypothesis(Patient,german\_measles) :- |
|  | symptom(Patient,fever), |
|  | symptom(Patient,headache), |
|  | symptom(Patient,runny\_nose), |
|  | symptom(Patient,rash). |
|  | hypothesis(Patient,flu) :- |
|  | symptom(Patient,fever), |
|  | symptom(Patient,headache), |
|  | symptom(Patient,body\_ache), |
|  | symptom(Patient,conjunctivitis), |
|  | symptom(Patient,chills), |
|  | symptom(Patient,sore\_throat), |
|  | symptom(Patient,runny\_nose), |
|  | symptom(Patient,cough). |
|  | hypothesis(Patient,common\_cold) :- |
|  | symptom(Patient,headache), |
|  | symptom(Patient,sneezing), |
|  | symptom(Patient,sore\_throat), |
|  | symptom(Patient,runny\_nose), |
|  | symptom(Patient,chills). |
|  | hypothesis(Patient,mumps) :- |
|  | symptom(Patient,fever), |
|  | symptom(Patient,swollen\_glands). |
|  | hypothesis(Patient,chicken\_pox) :- |
|  | symptom(Patient,fever), |
|  | symptom(Patient,chills), |
|  | symptom(Patient,body\_ache), |
|  | symptom(Patient,rash). |
|  | hypothesis(Patient,measles) :- |
|  | symptom(Patient,cough), |
|  | symptom(Patient,sneezing), |
|  | symptom(Patient,runny\_nose). |
|  |  |
|  | response(Reply) :- |
|  | readchar(Reply), |
|  | write(Reply),nl. |

PROGRAM 23:

(PROLOG:FORWARD CHAINING)

|  |
| --- |
| :- op(1100, xfx, if). |
|  | :- op(1000, xfy, and). % <== EDITED |
|  |  |
|  | forward(KB, Fact) :- |
|  | fixpoint(KB, nil, [true], Facts), |
|  | member(Fact, Facts). |
|  |  |
|  | fixpoint(\_, Base, Base, Base) :- !. |
|  | fixpoint(KB, \_, Base, Facts) :- |
|  | setof(Fact, derived(Fact, KB, Base), NewFacts), |
|  | ord\_union(NewFacts, Base, NewBase), |
|  | fixpoint(KB, Base, NewBase, Facts). |
|  |  |
|  | derived(Fact, KB, Base) :- |
|  | rule(KB : Fact if Condition), |
|  | satisfy(Base, Condition). |
|  |  |
|  | satisfy(Base, Condition) :- |
|  | ( Condition = (A and B) |
|  | -> member(A, Base), |
|  | satisfy(Base, B) |
|  | ; member(Condition, Base) ). |
|  |  |
|  |  |
|  | % first knowledge base |
|  |  |
|  | rule(1 : eats\_flies(fritz) if true). |
|  | rule(1 : croaks(fritz) if true). |
|  | rule(1 : sings(tweety) if true). |
|  | rule(1 : chips(tweety) if true). |
|  | rule(1 : has\_wings(tweety) if true). % <== EDITED |
|  | rule(1 : croaks(kroger) if true). |
|  | rule(1 : chips(kroger) if true). |
|  | rule(1 : frog(X) if croaks(X) and eats\_flies(X)). |
|  | rule(1 : green(X) if frog(X)). |
|  | rule(1 : yellow(X) if canary(X)). |
|  | rule(1 : canary(X) if sings(X) and chips(X) and has\_wings(X)). % <== EDITED |
|  |  |
|  | % second knowledge base (recursive example) |
|  |  |
|  | rule(2 : connected(a,b) if true). |
|  | rule(2 : connected(b,c) if true). |
|  | rule(2 : connected(c,d) if true). |
|  | rule(2 : connected(X,Z) if connected(X,Y) and connected(Y,Z)). |

PROGRAM 24:

(PROLOG:BACKWARD CHAINING)

|  |
| --- |
| %operator rules |
|  | :- op(800, fx, if). |
|  | :- op(700, xfx, then). |
|  | :- op(300, xfy, or). |
|  | :- op(200, xfy, and). |
|  |  |
|  | :- op(800, fx, fact). |
|  | :- op(800, fxf, <=). |
|  |  |
|  | % BACKWARD CHAINING INFERENCE ENGINE with proof tree |
|  | is\_true(P, P) :- |
|  | fact P. |
|  | is\_true(C, C <= ProofTreeA) :- |
|  | if A then C, is\_true(A, ProofTreeA). |
|  | is\_true(P1 and P2, ProofTree1 and ProofTree2) :- |
|  | is\_true(P1, ProofTree1), is\_true(P2, ProofTree2). |
|  | is\_true(P1 or \_, ProofTree1) :- is\_true(P1, ProofTree1). |
|  | is\_true(\_ or P2, ProofTree2) :- is\_true(P2, ProofTree2 |
|  |  |
|  | % production rules |
|  | if covering\_scales then family\_fish. |
|  | if covering\_skin then family\_mammal. |
|  | if family\_mammal and size\_large then species\_whale. |
|  | if family\_mammal and size\_small then species\_seal. |
|  | if family\_fish and size\_large then species\_tuna. |
|  | if family\_fish and size\_small then species\_sardine. |

PROGRAM 25:

(PROLOG FOR ANCHOR LINKS USING WEBLOG….)

:- module(web\_prolog,

[

flush/0,

spawn/1, % :Goal

spawn/2, % :Goal, -Pid

spawn/3, % :Goal, -Pid, +Options

send/2, % +Pid, +Message

(!)/2, % +Pid, +Message

exit/1, % +Reason

exit/2, % +Pid, +Reason

receive/1, % +ReceiveClauses

link/2, % +Parent, +Child

self/1, % -Pid

register/2, % +Alias, +Pid

unregister/1, % +Alias

whereis/2, % +Alias, -Pid

pengine\_spawn/1, % -Pid

pengine\_spawn/2, % -Pid, +Options

pengine\_ask/2, % +Pid, :Query

pengine\_ask/3, % +Pid, :Query, +Options

pengine\_next/1, % +Pid

pengine\_next/2, % +Pid, +Options

pengine\_stop/1, % +Pid

pengine\_stop/2, % +Pid, +Options

pengine\_abort/1, % +Pid

pengine\_input/2, % +Prompt, ?Answer

pengine\_respond/2, % +Pid, +Answer

pengine\_output/1, % +Term

pengine\_exit/1, % +Reason

pengine\_exit/2, % +Pid, +Reason

consult\_text/1, % +SourceCodeString

rpc/2, % +URI, :Query

rpc/3, % +URI, :Query, +Options

promise/3, % +URI, :Query, -Reference

promise/4, % +URI, :Query, -Reference, +Options

yield/2, % +Reference, ?Message

yield/3, % +Reference, ?Message, +Options

echo/1,

dump\_backtrace/2, % +Pid, +Depth

dump\_queue/2, % +Pid, -Queue

check/0,

op(1000, xfx, when),

op(800, xfx, !),

op(200, xfx, @),

node/0,

node/1

]).

:- use\_module(library(option)).

:- use\_module(actors).

:- use\_module(distribution).

:- use\_module(node).

:- use\_module(isolation).

:- use\_module(pengines2).

:- use\_module(restful\_api).

:- use\_module(rpc).

:- use\_module(io).

:- if(current\_predicate(uuid/2)).

actor\_uuid(Id) :-

uuid(Id, [version(4)]). % Version 4 is random.

:- else.

:- use\_module(library(random)).

actor\_uuid(Id) :-

Max is 1<<128,

random\_between(0, Max, Num),

atom\_number(Id, Num).

:- endif.

:- user:consult(resident).

:- multifile

actors:hook\_goal/3.

actors:hook\_goal(Goal0, isolation:with\_source(Goal0, GoalOptions), Options0) :-

option(node(localnode), Options0, localnode),

strip\_module(Goal0, SelfModule, \_),

translate\_local\_sources(Options0, Options, SelfModule),

actor\_uuid(Module),

GoalOptions = [ module(Module)

| Options

].