**Prolog programming**

**Student, Teacher and Sub-Code**

% Facts: Student, Teacher, and Subject Code relationships

student(john).

student(lisa).

student(ali).

teacher(prof\_smith).

teacher(prof\_jones).

teacher(prof\_doe).

subject\_code(math101, prof\_smith).

subject\_code(english101, prof\_jones).

subject\_code(physics101, prof\_doe).

% Rules: Relationship between students, teachers, and subject codes

teaches(Teacher, SubjectCode) :- teacher(Teacher), subject\_code(SubjectCode, Teacher).

enrolled(Student, SubjectCode) :- student(Student), subject\_code(SubjectCode, \_).

**Tower of Hanoi**

move(1,X,Y,\_) :-

write('Move top disk from '),

write(X),

write(' to '),

write(Y),

nl.

move(N,X,Y,Z) :-

N>1,

M is N-1,

move(M,X,Z,Y),

move(1,X,Y,\_),

move(M,Z,Y,X).

o/p-move(4,left,right,center).

**Bird fly**

bird(sparrow).

bird(eagle).

bird(duck).

bird(crow).

~bird(ostrich).

bird(puffin).

bird(swan).

bird(albatross).

bird(starling).

bird(owl).

bird(kingfisher).

bird(thrush).

can\_fly(X):-bird(X).

can\_fly(ostrich):-fail.

**Family tree**

% Facts: Define family relationships

male(john).

male(bob).

male(jim).

female(lisa).

female(pat).

female(ann).

parent(john, bob).

parent(john, lisa).

parent(bob, jim).

parent(pat, jim).

parent(bob, ann).

parent(pat, ann).

% Rules: Define additional family relationships

father(X, Y) :- male(X), parent(X, Y).

mother(X, Y) :- female(X), parent(X, Y).

grandparent(X, Y) :- parent(X, Z), parent(Z, Y).

% Example queries:

% Who are the children of John?

% ?- parent(john, X).

% Who is the father of Jim?

% ?- father(X, jim).

% Who are the grandchildren of John?

% ?- grandparent(john, X).

**Monkey and banana**

on(floor,monkey).

on(floor,chair).

in(room,monkey).

in(room,chair).

in(room,banana).

at(ceiling,banana).

strong(monkey).

grasp(monkey).

climb(monkey,chair).

push(monkey,chair):-

strong(monkey).

under(banana,chair):-

push(monkey,chair).

canreach(banana,monkey):-

at(floor,banana);

at(ceiling,banana),

under(banana,chair),

climb(monkey,chair).

canget(banana,monkey):-

canreach(banana,monkey),grasp(monkey).

o/p- ?- canget(banana, monkey).

**Forward chaining**

% Define facts and rules

% Fact: Birds can fly.

can\_fly(bird).

% Rule: If an animal has wings, it can fly.

can\_fly(X) :- has\_wings(X).

% Fact: Birds have wings.

has\_wings(bird).

% Rule: If an animal is a penguin, it cannot fly.

cannot\_fly(penguin) :- bird(penguin).

% Rule: If an animal is not known to have wings, it cannot fly.

cannot\_fly(X) :- animal(X), not(has\_wings(X)).

% Example query to find animals that can fly

find\_flying\_animals(X) :- can\_fly(X).

% Example query to find animals that cannot fly

find\_non\_flying\_animals(X) :- cannot\_fly(X).

% Example query to check if a specific animal (e.g., penguin) can fly

?- can\_fly(penguin).

**DOB**

born(jan, 20,3,1977).

born(jeroen, 2,2,1992).

born(joris, 17,3,1995).

born(jelle, 1,1,2004).

born(jesus, 24,12,2000).

born(joop, 30,4,1989).

born(jannecke, 17,3,1993).

born(jaap, 16,11,1995).

**Nam,age,sex**

person(frances,wilson,female,28,architect).

person(fred,jones,male,62,doctor).

person(paul,smith,male,45,plumber).

person(martin,williams,male,23,chemist).

person(mary,jones,female,24,programmer).

person(martin,johnson,male,47,solicitor).

man(A):-person(A,B,male,C,D).

**dog program**

% Define facts about animals and their sizes

dog(fido).

dog(rover).

dog(jane).

dog(tom).

dog(fred).

dog(henry).

cat(mary).

cat(harry).

cat(bill).

cat(steve).

small(henry).

medium(harry).

medium(fred).

large(fido).

large(mary).

large(tom).

large(fred).

large(steve).

large(jim).

large(mike).

% Define rules to classify animals by size

is\_small(Animal) :- dog(Animal), small(Animal).

is\_medium(Animal) :- (dog(Animal) ; cat(Animal)), medium(Animal).

is\_large(Animal) :- (dog(Animal) ; cat(Animal)), large(Animal).

% Example queries

% Find small dogs

small\_dog(X) :- is\_small(X).

% Find medium-sized animals

medium\_animal(X) :- is\_medium(X).

% Find large animals

large\_animal(X) :- is\_large(X).

?- small\_dog(X).

**Planets**

orbits(mercury, sun).

orbits(venus, sun).

orbits(earth, sun).

orbits(mars, sun).

orbits(moon, earth).

orbits(phobos, mars).

orbits(deimos, mars).

planet(P) :- orbits(P,sun).

satellite(S) :- orbits(S,P), planet(P).

**cities forward chaining**

% Define facts about rainy and cold cities

rainy(chennai).

rainy(coimbatore).

rainy(ooty).

cold(ooty).

% Define a rule to determine if it's a winter season in a city

winter\_season(City) :- rainy(City), cold(City).

% Define a rule to determine if it's a summer season in a city

summer\_season(City) :- rainy(City), not(cold(City)).

% Example queries

% Find cities where it's a winter season

winter\_cities(City) :- winter\_season(City).

% Find cities where it's a summer season

summer\_cities(City) :- summer\_season(City).

?- winter\_cities(City).

**Fruit colour**

colour(cherry, red).

colour(banana, yellow).

colour(apple, red).

colour(apple, green).

colour(orange, orange).

colour(X, unknown).

**Pattern matching**

% Define facts about first and second names

first\_name(tonyblair, tony).

first\_name(georgebush, georgedubya).

second\_name(tonyblair, blair).

second\_name(georgebush, bush).

% Define a predicate to match full names based on first and second names

match\_full\_name(FirstName, SecondName, FullName) :-

first\_name(FullName, FirstName),

second\_name(FullName, SecondName).

% Example queries

% Match the full name for a given first and second name (e.g., george and bush)

% ?- match\_full\_name(george, bush, FullName).

**Family relation**

% Define facts about individuals and their genders

female(sarah).

female(rebekah).

female(hagar\_concubine).

female(milcah).

female(bashemath).

female(mahalath).

female(first\_daughter).

female(rachel).

female(labans\_wife).

male(terah).

male(abraham).

male(nahor).

male(haran).

male(isaac).

male(ismael).

male(uz).

male(kemuel).

% Define family relations (parent-child relationships)

parent(terah, abraham).

parent(terah, nahor).

parent(terah, haran).

parent(abraham, isaac).

parent(abraham, ismael).

parent(nahor, uz).

parent(haran, kemuel).

parent(haran, milcah).

parent(haran, ismael).

parent(rebekah, jacob).

parent(rebekah, esau).

parent(labans\_wife, rachel).

parent(labans\_wife, first\_daughter).

parent(labans\_wife, leah).

% Define sibling relationship

sibling(X, Y) :-

parent(Z, X),

parent(Z, Y),

X \= Y.

% Define spouse relationship (married)

married(abraham, sarah).

married(abraham, rebekah).

married(isaac, rebekah).

married(ismael, hagar\_concubine).

% Define grandmother relationship

grandmother(X, Y) :-

female(X),

parent(X, Z),

parent(Z, Y).

% Define grandfather relationship

grandfather(X, Y) :-

male(X),

parent(X, Z),

parent(Z, Y).

% Define aunt relationship

aunt(X, Y) :-

sibling(X, Z),

parent(Z, Y),

female(X).

% Define uncle relationship

uncle(X, Y) :-

sibling(X, Z),

parent(Z, Y),

male(X).

% Define cousin relationship

cousin(X, Y) :-

parent(Z, X),

parent(W, Y),

sibling(Z, W).

% Example queries

% Find the siblings of a person (e.g., siblings of Isaac)

% ?- sibling(isaac, Sibling).

% Find the grandparents of a person (e.g., grandparents of Jacob)

% ?- grandfather(Grandparent, jacob).

% Find the aunts of a person (e.g., aunts of Jacob)

% ?- aunt(Aunt, jacob).

% Find the cousins of a person (e.g., cousins of Jacob)

% ?- cousin(Cousin, jacob).

**Location**

location(desk, office).

location(apple, kitchen).

location(flashlight, desk).

**Disease symptoms**

% Define predicates for hypothesis (disease) and symptoms (indications)

predicates

hypothesis(name, disease)

symptom(name, indication)

% Define facts about symptoms

clauses

symptom(amit, fever).

symptom(amit, rash).

% Define rules for disease hypotheses based on symptoms

hypothesis(Name, Disease) :-

symptom(Name, fever),

symptom(Name, rash),

Disease = 'Some Disease'.

% Example queries

% Predict the disease for a person (e.g., Amit)

% ?- hypothesis(amit, Disease).

**Inference engine**

% Define an inference engine

interpret(true) :- !.

interpret((GoalA, GoalB)) :- !,

interpret(GoalA),

interpret(GoalB).

% Example rules

human(socrates).

mortal(X) :- human(X).

% Example queries

% Query 1: Is Socrates mortal?

% ?- interpret(mortal(socrates)).

% This query should return "true" because Socrates is a human and humans are mortal.

% Query 2: Is X mortal?

% ?- interpret(mortal(X)).

% This query can be used to find all mortals in the knowledge base by backtracking.

% Query 3: Is X human and Y mortal?

% ?- interpret((human(X), mortal(Y))).

% This query can be used to find humans who are also mortals.

**Vowels count**

% Define a predicate to count the number of vowels in a string

count\_vowels(String, Count) :-

string\_lower(String, LowerString), % Convert the string to lowercase

atom\_chars(LowerString, Chars), % Convert the string to a list of characters

count\_vowels\_helper(Chars, Count).

% Define a helper predicate to count vowels in a list of characters

count\_vowels\_helper([], 0).

count\_vowels\_helper([Char | Rest], Count) :-

(is\_vowel(Char) -> TempCount = 1; TempCount = 0),

count\_vowels\_helper(Rest, RestCount),

Count is TempCount + RestCount.

% Define a predicate to check if a character is a vowel

is\_vowel(Char) :-

member(Char, ['a', 'e', 'i', 'o', 'u']).

% Example query

% Count the number of vowels in the given sentence

?- count\_vowels("This is my first Degree in Saveetha School of Engineering", Count).

**Mortal**

% Define facts about individuals

man(socrates).

man(einstein).

man(alexander).

% Define a predicate to determine if a person is mortal

mortal(X) :- man(X).

% Example queries

% Is Socrates mortal?

% ?- mortal(socrates).

% This query will return "true" because Socrates is a man, and men are considered mortal.

% Is Einstein mortal?

% ?- mortal(einstein).

% This query will return "true" for the same reason.

% Is Alexander mortal?

% ?- mortal(alexander).

% This query will return "true" as well.

**Happy student**

% Statements

% 1. Anyone passing history exams and winning the lottery is happy.

% 2. Anyone who studies or is lucky can pass all exams.

% 3. John did not study, but he is lucky.

% 4. Anyone who is lucky wins the lottery.

% Define clauses for resolution

% Convert the statements into clauses in CNF (conjunctive normal form)

% Clause 1: ¬happy(X) ∨ history\_exam\_passed(X) ∨ lottery\_won(X).

clause1(X) :- not(happy(X)) ; history\_exam\_passed(X) ; lottery\_won(X).

% Clause 2: ¬pass\_exam(X, Y) ∨ studies(X) ∨ lucky(X).

clause2(X, Y) :- not(pass\_exam(X, Y)) ; studies(X) ; lucky(X).

% Clause 3: ¬studies(john) ∧ lucky(john).

clause3 :- not(studies(john)), lucky(john).

% Clause 4: ¬lottery\_won(X) ∨ lucky(X).

clause4(X) :- not(lottery\_won(X)) ; lucky(X).

% Negation of the goal: We want to prove that John is happy.

goal :- not(happy(john)).

% Use resolution to prove the goal

prove\_goal :- goal, clause1(john), clause2(john, \_), clause3, clause4(john).

% Example query to prove that John is happy

% ?- prove\_goal.