prolog

September 15, 2024

[ ]:
```
% DB with Name and DOB
person(name, dob).
person('Alice', '01-Jan-1990').
person('Bob', '05-Mar-1985').

?- person(Name, '01-Jan-1990').
```

[ ]:
```
% Student-Teacher-Sub-Code
teaches(teacher, subject, student).
teaches('Mr. John', math, 'Alice').
teaches('Ms. Sarah', english, 'Bob').

?- teaches(Teacher, math, Student).
```

[ ]:
```
% Planets DB
planet(mercury, 57).
planet(venus, 108).

?- planet(Name, Distance).
```

[ ]:
```
% Towers of Hanoi
move(1, From, To, _) :-
    write('Move top disk from '), write(From), write(' to '), write(To), nl.
move(N, From, To, Aux) :-
    N > 1,
    M is N-1,
    move(M, From, Aux, To),
    move(1, From, To, _),
    move(M, Aux, To, From).

?- move(3, left, right, center).
```

[ ]:
```
% Birds that can fly

birds(eagle).
birds(sparrow).
birds(parrot).
```

```
birds(hummingbird).
% Birds that cannot fly
birdss(penguin).
birdss(ostrich).
birdss(kiwi).
birdss(dodo).
% Rule to determine if a bird can fly or not
can_fly(X) :- birds(X), write(X), write(' can fly'), !.
can_fly(X) :- birdss(X), write(X), write(' cannot fly'), !.

?- can_fly(eagle).
```

[ ]:
```
% Family Tree
parent(john, mary).
parent(mary, susan).
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).

?- grandparent(john, susan).
```

[ ]:
```
% Dieting System Based on Disease
diet(diabetes, 'Low sugar').
diet(hypertension, 'Low salt').
recommend_diet(Disease, Diet) :- diet(Disease, Diet).

?- recommend_diet(diabetes, Diet).
```

[ ]:
```
% Monkey Banana Problem
move(grab_banana).
move(climb_box).
can_get_banana :- move(grab_banana).
can_get_banana :- move(climb_box), can_get_banana.

?- can_get_banana.
```

[ ]:
```
% Fruit and Color Using Backtracking
fruit(apple, red).
fruit(banana, yellow).
find_fruit_by_color(Color, Fruit) :- fruit(Fruit, Color).

?- find_fruit_by_color(yellow, Fruit).
```

[ ]:
```
% Best First Search Algorithm
best_first_search(Start, Goal, Path) :- bfs([Start], Goal, [], Path).

bfs([Goal|T], Goal, _, [Goal|T]).
bfs([Current|Rest], Goal, Visited, Path) :-
    findall(Next, (edge(Current, Next), \+ member(Next, Visited)), Neighbors),
```

```
    append(Rest, Neighbors, NewQueue),
    bfs(NewQueue, Goal, [Current|Visited], Path).

?- best_first_search(a, g, Path).
```

[ ]:
```
% Medical Diagnosis
diagnose(fever, 'Take paracetamol').
diagnose(cough, 'Take cough syrup').
get_diagnosis(Symptom, Treatment) :- diagnose(Symptom, Treatment).

?- get_diagnosis(fever, Treatment).
```

[ ]:
```
% Forward Chaining
parent(john, mary).
parent(mary, susan).
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).

?- ancestor(john, susan).
```

[ ]:
```
% Backward Training
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
parent(john, mary).
parent(mary, susan).

?- ancestor(john, susan).
```

[ ]:
```
% Pattern Matching
match([X|_], X).

?- match([1, 2, 3], X).
```

[ ]:
```
% Vowels
vowel_count([], 0).
vowel_count([H|T], Count) :-
    member(H, [a, e, i, o, u]),
    vowel_count(T, Rest),
    Count is Rest + 1.
vowel_count([_|T], Count) :- vowel_count(T, Count).

?- vowel_count([a, b, c, e, i], Count).
```