

## SE2052 – Programming Paradigms

### Lab – 07

- What is unification in Prolog, and how is it used to match facts with queries?

*Unification refers to the process in Prolog of making two terms identical by determining the values of any variables within them, and is essentially how Prolog matches a query against facts and rules.*

- How does Prolog use backtracking to find multiple solutions to a query?

*If a fact or rule does not completely satisfy the query, then Prolog will automatically attempt a different pathway, or "backtrack" until it finds all possible solutions, or fails altogether.*

- What does the cut operator (!)

*The cut operator instructs Prolog in the form of "do not backtrack past this point." The cut operator can be used in a predicate to prevent unnecessary backtracking and therefore attempts to make our programs more efficient or deterministic.*

- Explain the Closed World Assumption and how it affects reasoning in Prolog.

*In Prolog, if something is not in the knowledge base, it is assumed to be false. If it can't prove it to be true, Prolog assumes it is false.*

- What is the difference between logical negation and negation as failure in Prolog?

**Logical negation:** would indicate true "not true" in classical logic.

**Negation as failure:** means "Prolog failed to prove Goal." They are not always logical negation.

- Why does the order of rules and facts matter in Prolog if logic is declarative?

*Though logic is declarative, Prolog executes it in top-down order. Order is important since it handles which answer is found first, and performance.*

- Compare backward chaining (used by Prolog) with forward chaining. What are the advantages of each?

**Backward chaining (Prolog):** begins with a query (goal) to find facts/rules. Efficient for particular inquiries. **Forward chaining:** begins with facts and utilizes rules to apply everything inferred. Better for broad inquiry.

- How does Prolog handle recursion in rules? Provide an example scenario.

*Prolog uses recursion to solve problems step by step.*

- What is the meaning of predicate arity (e.g., likes/2) and why is it important?

*Arity = number of arguments a predicate takes. likes/2 means likes with 2 arguments. Important because likes/1 and likes/2 are different predicates.*

- What are some limitations of Prolog in real-world applications?

*Tends to be slow for very large datasets, Not particularly adept at number crunching or heavy computation, Scaling for complicated systems in the real-world can be difficult, Debugging can be complicated since we assume backtrack we're implemented in the thing being debugged.*

## Exercise 1:



```

1 parent(john, mary).
2 parent(susan, mary).
3 parent(john, mark).
4 parent(susan, mark).
5 parent(mary, alice).
6 parent(mark, bob).
7
8 sibling(X, Y) :-
9     parent(P, X),
10    parent(P, Y),
11    X \= Y.
12
13 grandparent(X, Y) :-
14     parent(X, Z),
15     parent(Z, Y).
16
17 ancestor(X, Y) :-
18     parent(X, Y).
19 ancestor(X, Y) :-
20     parent(X, Z),
21     ancestor(Z, Y).
22

```

Outputs:




The screenshot shows a Prolog interpreter window with three separate query sessions. Each session displays the query, the result (true), and a control bar with 'Next', '10', '100', '1,000', and 'Stop' buttons.

- Session 1:** Query: `ancestor(susan, alice).` Result: `true`
- Session 2:** Query: `grandparent(john, alice).` Result: `true`
- Session 3:** Query: `sibling(mary, mark).` Result: `true`

Exercise 2:

```
1 parent(john, mary).
2 parent(susan, mary).
3 parent(john, mark).
4 parent(susan, mark).
5 parent(mary, alice).
6 parent(mark, bob).
7
8 sibling(X, Y) :-
9     parent(P, X),
10    parent(P, Y),
11    X \= Y.
12
13 grandparent(X, Y) :-
14     parent(X, Z),
15     parent(Z, Y).
16
17 ancestor(X, Y) :-
18     parent(X, Y).
19 ancestor(X, Y) :-
20     parent(X, Z),
21     ancestor(Z, Y).
22
23 likes(john, pizza).
24 likes(john, salad).
25 likes(susan, salad).
26 likes(mary, chocolate).
27 likes(mark, pizza).
28 likes(alice, chocolate).
29 likes(bob, pizza).
30
31 dislikes(john, chocolate).
32 dislikes(susan, pizza).
33 dislikes(mary, salad).
34 dislikes(mark, chocolate).
35 dislikes(alice, pizza).
36 dislikes(bob, salad).
37
38
39 can_eat(Person, Food) :-
40     likes(Person, Food),
41     \+ dislikes(Person, Food).
```



Outputs:

 <code>can_eat(bob, X).</code> <code>X = pizza</code>	 <code>can_eat(john, chocolate).</code> <code>false</code>
<code>?- can_eat(bob, X).</code>	<code>?- can_eat(john, chocolate).</code>
 <code>can_eat(john, pizza).</code> <code>true</code>	
<code>?- can_eat(john, pizza).</code>	

Exercise 3:

```
Program x +
1
2
3 grade_system(_, a, pass) :-
4     !.
5
6 grade_system(_, b, pass).
7 grade_system(_, c, pass).
8
9 grade_system(_, d, fail).
10 grade_system(_, f, fail).
11
12 student(john, a).
13 student(mary, b).
14 student(alice, c).
15 student(bob, d).
16 student(eve, f).
17
18 check_result(Student) :-
19     student(Student, Grade),
20     grade_system(Student, Grade, Result),
21     format('Student ~W with grade ~W: ~W~n', [Student, Grade, Result]).
22
```

Outputs:

 <code>check_all_students.</code>	 <code>check_result(john).</code>
Student john with grade a: pass Student mary with grade b: pass Student alice with grade c: pass Student bob with grade d: fail Student eve with grade f: fail <b>true</b>	Student john with grade a: pass <b>true</b>
?- <code>check_all_students.</code>	?- <code>check_result(john).</code>


Exercise 4:

```
SWISH  File - Edit - Examples - Help -
Program
1 parent(john, mary).
2 parent(susan, mary).
3 parent(john, mark).
4 parent(susan, mark).
5 parent(mary, alice).
6 parent(mark, bob).
7 sibling(X, Y) :-
8     parent(P, X),
9     parent(P, Y),
10    X \= Y.
11 grandparent(X, Y) :-
12     parent(X, Z),
13     parent(Z, Y).
14 ancestor(X, Y) :-
15     parent(X, Y).
16 ancestor(X, Y) :-
17     parent(X, Z),
18     ancestor(Z, Y).
19 likes(john, pizza).
20 likes(john, salad).
21 likes(john, chocolate).
22 likes(susan, salad).
23 likes(susan, chocolate).
24 likes(mary, chocolate).
25 likes(mary, pizza).
26 likes(mark, pizza).
27 likes(mark, salad).
28 likes(alice, chocolate).
29 likes(alice, fruit).
30 likes(bob, pizza).
31 likes(bob, salad).
32
33 dislikes(john, chocolate).
34 dislikes(susan, pizza).
35 dislikes(mary, salad).
36 dislikes(mark, chocolate).
37 dislikes(alice, pizza).
38 dislikes(bob, salad).
39
40
41 can_eat(Person, Food) :-
42     likes(Person, Food),
43     \+ dislikes(Person, Food).
```

Outputs:

 <code>findall(Food, can_eat(john, Food), Foods).</code> <b>Foods</b> = [pizza, salad]	 <code>likes(john, Food).</code> <b>Food</b> = pizza Next 10 100 1,000 Stop
?- <code>findall(Food, can_eat(john, Food), Foods).</code>	?- <code>likes(john, Food).</code>

Exercise 5:


**SWISH**
File Edit Examples Help



Program

```

1 parent(john, mary).
2 parent(susan, mary).
3 parent(john, mark).
4 parent(susan, mark).
5 parent(mary, alice).
6 parent(mark, bob).
7 sibling(X, Y) :-
8     parent(P, X),
9     parent(P, Y),
10    X \= Y.
11 grandparent(X, Y) :-
12     parent(X, Z),
13     parent(Z, Y).
14 ancestor(X, Y) :-
15     parent(X, Y).
16 ancestor(X, Y) :-
17     parent(X, Z),
18     ancestor(Z, Y).
19 likes(john, pizza).
20 likes(john, salad).
21 likes(john, chocolate).
22 likes(susan, salad).
23 likes(susan, chocolate).
24 likes(mary, chocolate).
25 likes(mary, pizza).
26 likes(mark, pizza).
27 likes(mark, salad).
28 likes(alice, chocolate).
29 likes(alice, fruit).
30 likes(bob, pizza).
31 likes(bob, salad).
32 dislikes(john, chocolate).
33 dislikes(susan, pizza).
34 dislikes(mary, salad).
35 dislikes(mark, chocolate).
36 dislikes(alice, pizza).
37 dislikes(bob, salad).
38 can_eat(Person, Food) :-
39     likes(Person, Food),
40     \+ dislikes(Person, Food).
41 recommend(Person, Food) :-
42     likes(Person, Food),
43     \+ dislikes(Person, Food).
44

```

Output:

 `recommend(alice, Food).`

**Food** = chocolate

Next

10

100

1,000

Stop

---

?- `recommend(alice, Food).`