TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

A
LAB REPORT
ON

PROLOG - III: FIRST ORDER PREDICATE LOGIC

SUBMITTED BY:

Diwas Adhikari **(076BEI014)**

SUBMITTED TO:

Department of Electronics & Computer Engineering
Pulchowk Campus, IOE

LAB NUMBER

3

SUBMISSION DATE

July 12, 2023

(For course completion of BEI – Artificial Intelligence)

OBJECTIVES

- To understand use of quantifiers and application of predicate logic.
- To convert generic predicates and clauses to First Order Predicate Logic (FOPL) for various problems and solve them in PROLOG.

THEORY

Artificial intelligence (AI) uses predicate logic as a method of knowledge representation. First Order Predicate Logic (FOPL) is expressive enough to accurately capture the natural language claims. It is a potent language that expresses the relationship between the items as well as how knowledge about the objects may be developed more easily.
As an illustration, take the sentence "X is a man." It consists of two parts: the variable X, which is the statement's subject; and the predicate, "is a man," which describes a possible attribute for the statement's subject. A variable becomes propositional logic and has an associated truth value once it is assigned to the propositional function man(X).

Quantifiers
A linguistic element called a quantifier produces quantification, which describes the quantity of specimens in the discourse universe. These are the symbols that allow for the determination or identification of the variable's range and scope in the logical expression.
There are two types of quantifiers:
1. <u>Universal Quantifier</u> (for all, everyone, everything) :
Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or, every instance of a particular thing. It is represented by a symbol ∀.

2. <u>Existential Quantifier</u> (for some, at least one) :
Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something. It is represented by a symbol ∃.

For example:
1. Harry enjoys all movies.
$\forall x$ Movie(x) $\Rightarrow$ Enjoys(Harry, x)
2. Kale is a dog.
Dog (Kale)
3. Grandparent is a parent of one’s parent.
$\forall x, y$ Grandparent (x, y) $\Leftrightarrow \exists z$ Parent (x, z) $\cap$ Parent (z, y)
4. Parent and child are inverse relation.
$\forall x, y$ Parent (x, y) $\Leftrightarrow$ Child (y, x)

Conversion of PROLOG into FOPL

PROLOG clauses can be directly translated into FOPL, except for a few exceptions like write, !, is, assert, retract etc.

The three simple rules for conversion are:

- “:-” corresponds to “←”.
- “,” corresponds to “&”.
- All variables are universally quantified.

For example:

grandfather(X,Y) :- father(X,Z),parent(Z,Y).

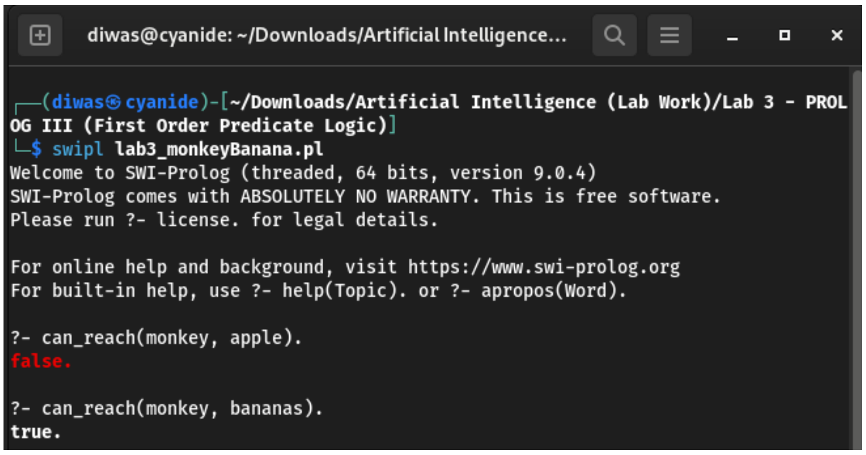
<u>In FOPL:</u>

∀x, y, z (grandfather(x,y) <- (father(x,z)&parent(z,y))).

<u>‘Monkey - Banana’ Problem</u>

Where there is a room containing a monkey, a chair, and bananas that have been hung from the center of the ceiling of the room; out of reach from monkey. If the monkey is clever enough, he can reach the bananas by placing the chair directly below the bananas and climbing on the top of the chair. Can the monkey reach the bananas ?

```
in_room(bananas).
in_room(chair).
in_room(monkey).
dexterous(monkey).
tall(chair).
can_move(monkey,chair,bananas).
can_climb(monkey,chair).
can_reach(X,Y):-
   dexterous(X),
   closeTo(X,Y).
closeTo(X,Z):-
   get_on(X,Y),
   under(Y,Z),
   tall(Y).
get_on(X,Y):-
   can_climb(X,Y).
under(Y,Z):-
   in_room(X),
   in_room(Y),
   in_room(Z),
   can_move(X,Y,Z).
```

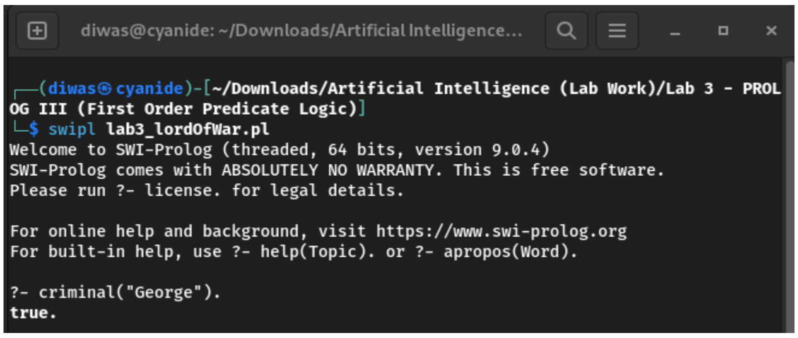


```
┌──(diwas㉿cyanide)-[~/Downloads/Artificial Intelligence (Lab Work)/Lab 3 - PROLOG III (First Order Predicate Logic)]
└─$ swipl lab3_monkeyBanana.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- can_reach(monkey, apple).
false.

?- can_reach(monkey, bananas).
true.
```

‘Lord Of War’ Problem

1. Every American who sells weapons to hostile nations is a criminal.
2. Every enemy of America is a hostile.
3. Iraque has some missiles.
4. All missiles of Iraque were sold by George.
5. George is an American.
6. Iraque is a country.
7. Iraque is the enemy of America.
8. Missiles are weapons.

```
criminal(X):-
    american(X),
    sells_missiles(X, Y),
    hostile(Y).
enemy_of_america(X) :-
    hostile(X).
    enemy_of_america("Iraq").
hostile(X):-
    country(X).
    has_missile("Iraq").
    sells_missiles("George", "Iraq").
    american("George").
    country("Iraq").
```



```
┌──(diwas㉿cyanide)-[~/Downloads/Artificial Intelligence (Lab Work)/Lab 3 - PROL
OG III (First Order Predicate Logic)]
└─$ swipl lab3_lordOfWar.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- criminal("George").
true.
```

PREMISE 1
1. Horses, cows, pigs are mammals.
2. An offspring of a horse is a horse.
3. Bluebeard is a horse.
4. Bluebeard is Charlie’s parent.
5. Offspring and parent are inverse relations.
6. Every mammal has a parent.

Q. Is Charlie a horse?

PREMISE 2
1. All people who are not poor and are smart are happy.
2. Those people who read are not stupid.
3. John can read and is wealthy.
4. Happy people have exciting lives.

Q. Can anyone be found with an exciting life?

PREMISE 3
1. All pompeians are romans.
2. All romans were either loyal to Caesar or hated him.
3. Everyone is loyal to someone.
4. People only try to assassinate rulers they are not loyal to.
5. Marcus tried to assassinate Caesar.
6. Marcus was Pompeian.

Q. Did Marcus hate Caesar?

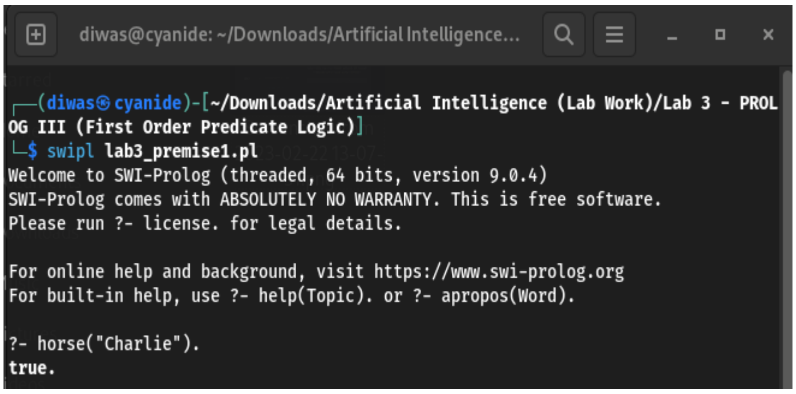
PREMISE 4
Bhogendra likes all kinds of food. Oranges are food. Chicken is food. Anything anyone eats and isn’t killed by is food. If a person likes a food means that person has eaten it. Jogendra eats peanuts and is still alive. Shailendra eats everything Bhogendra eats. Does Shailendra like chicken ?

PREMISE 5
Dave and Fred are members of a dancing club in which no member can both waltz and jive. Fred’s dad can’t waltz and Dave can do whatever fred can’t do. If a child can do something, then their parents can do it also. Prove that there is a member of the dancing club who can’t jive.

PROLOG CODE & OBSERVATION

1) **Premise 1**

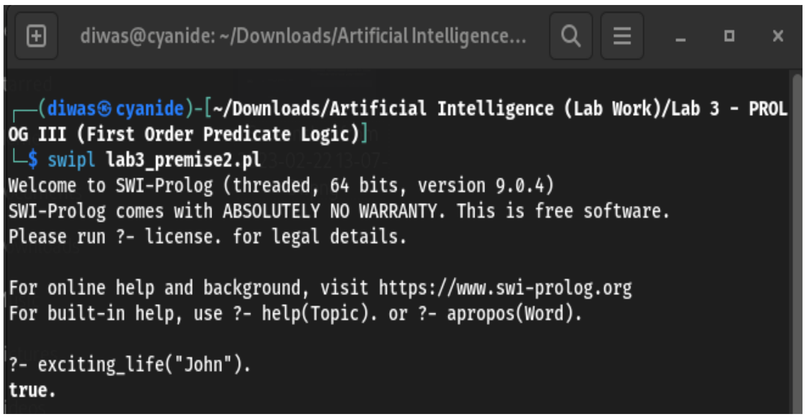
```
mammal(X):-
        horse(X),
        cow(X),
        pig(X),
        is_parent(_,X).
horse(Y):-
        is_parent(X,Y),horse(X).
        horse("Bluebeard").
is_offspring(X,Y):-
        is_parent(Y,X).
        is_parent("Bluebeard","Charlie").
        cow("?").
        pig("?").
```



```
┌──(diwas㉿cyanide)-[~/Downloads/Artificial Intelligence (Lab Work)/Lab 3 - PROLOG III (First Order Predicate Logic)]
└─$ swipl lab3_premise1.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- horse("Charlie").
true.
```

2) **Premise 2**

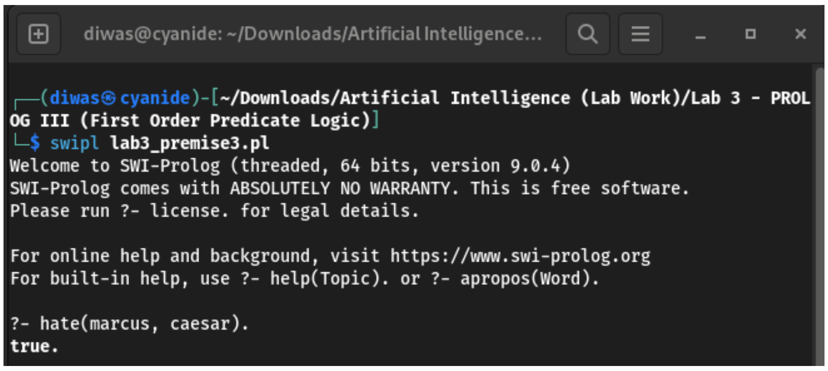
```
exciting_life(X):-
        happy(X).
happy(X):-
        not_poor(X),
        smart(X).
smart(X):-
        not_stupid(X).
not_stupid(X):-
        readWell(X).
not_poor(X):-
        wealthy(X).
        readWell("John").
        wealthy("John").
```



```
┌──(diwas㉿cyanide)-[~/Downloads/Artificial Intelligence (Lab Work)/Lab 3 - PROLOG III (First Order Predicate Logic)]
└─$ swipl lab3_premise2.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- exciting_life("John").
true.
```

3) **Premise 3**

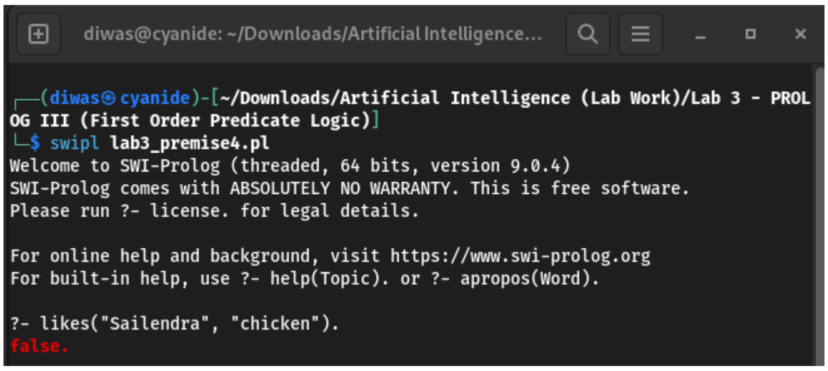
```
roman(X):-
        pompeian(X).
        assasinate(marcus,caesar).
        pompeian(marcus).
hate(X,caesar):-
        roman(X),
        not_loyal(X,caesar).
loyal(X,caesar):-
        roman(X),
        not(hate(X,caesar)).
not_loyal(X,Y):- assasinate(X,Y).
```



```
┌──(diwas㉿cyanide)-[~/Downloads/Artificial Intelligence (Lab Work)/Lab 3 - PROLOG III (First Order Predicate Logic)]
└─$ swipl lab3_premise3.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- hate(marcus, caesar).
true.
```

4) **Premise 4**

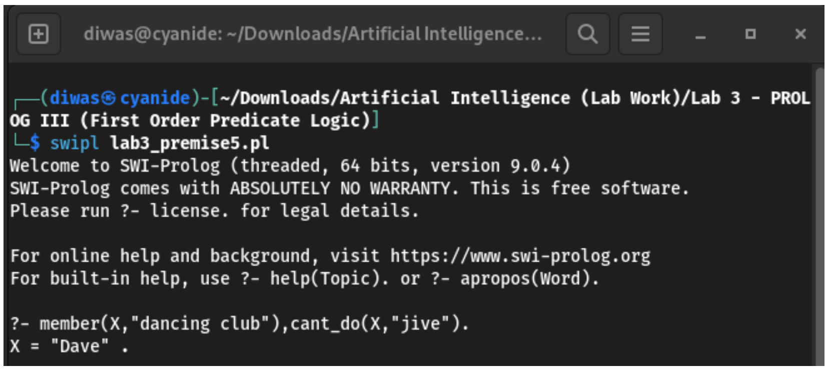
```
food("orange").
food("chicken").
food(X):-
        likes(Y,X),
        not(kills(X,Y)).
eats("Sailendra",Y):-
eats("Bhogendra",Y).
eats(X,Y):-
        likes(X,Y),
        food(Y).
likes("Bhogendra",X):-
        food(X).
        kills(_,_).
```



```
┌──(diwas㉿cyanide)-[~/Downloads/Artificial Intelligence (Lab Work)/Lab 3 - PROL
OG III (First Order Predicate Logic)]
└─$ swipl lab3_premise4.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- likes("Sailendra", "chicken").
false.
```

5) **Premise 5**

```
member("Dave","dancing club").
member("Fred","dancing club").
cant_do("Fred's dad","waltz").
cant_do(X,Y):-
        parent(Z,X),
        cant_do(Z,Y).
cant_do(X,"waltz"):-
        member(X,"dancing club"),
        can_do(X,"jive").
cant_do(X,"jive"):-
        member(X,"dancing club"),
        can_do(X,"waltz").
can_do("Dave",X):-
        cant_do("Fred",X).
        parent("Fred's dad","Fred").
```



```
┌──(diwas㉿cyanide)-[~/Downloads/Artificial Intelligence (Lab Work)/Lab 3 - PROL
OG III (First Order Predicate Logic)]
└─$ swipl lab3_premise5.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- member(X,"dancing club"),cant_do(X,"jive").
X = "Dave" .
```

DISCUSSION & CONCLUSION

In this lab session, we witnessed the use-case of existential and universal quantifiers to form predicates for various statements. We used PROLOG to solve five different predicate logic problems where we developed first order predicate logic to express natural language propositions and later verified the conclusion to be true or false.

SUBMITTED BY:

Diwas Adhikari
076BEI014 – (Roll Number: 14)
Bachelor in Electronics & Communication Engineering (BEI)
076bei014.diwas@pcampus.edu.np
lunaticoda123@gmail.com