
# NATIONAL INSTITUTE OF TECHNOLOGY, PUDUCHERRY

**(An institute of National importance under MHRD, Govt. of India)**

**KARAIKAL – 609 609**

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**April 2018**

## CS308: ARTIFICIAL INTELLIGENCE


**Name** : Shivam Kumar Giri

**Course** : B. Tech **Semester** : **VI**

**Branch** : COMPUTER SCIENCE & ENGINEERING


**Subject** : ARTIFICIAL INTELLIGENCE

**Submitted to** : Dr. B. Surendiran, Head of Department,CSE

**AN EXPERT SYSTEM IN PROLOG**

# DEVELOPING AN EXPERT SYSTEM IN PROLOG WHICH CAN DETERMINE THE COUNTRY YOU ARE THINKING AS PER YOUR RESPONSES ENCOUNTERED

## PREFACE

This documentation is along the prolog file "**base.pl**" and an output file "**output.txt**" under Artificial Intelligence covers the prolog syntax, various terminology related to the assignment and flow of control that can be implemented in SWI-Prolog. This prolog file "base.pl" is way to fetch the required country one is guessing in least span of steps using backward chaining

## INTRODUCTION

This is the short demo for creation of expert system in backward chaining by using Prolog. The main file "base.pl" has been attached along with this document and an output file named"output.txt". This document determine the country been taught by a person as per the response he provide. This entirely work on backward chaining i.e. it is a "goal driven". In this document, the short glimpse of prolog syntax has been discussed along with the flow of control of the entire project.

## CONTENT

## TERMINOLOGIES

**Prolog** is a general-purpose logic programming language associated with artificial intelligence and computational linguistics. Prolog has its roots in first-order logic, a formal logic, and unlike many other programming languages, Prolog is intended primarily as a declarative programming language: the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations.

**SWI-Prolog** offers a comprehensive free Prolog environment. Since its start in 1987, SWI-Prolog development has been driven by the needs of real world applications. SWI-Prolog is widely used in research and education as well as commercial applications.

**Expert System**
A piece of software, which uses databases of expert knowledge to offer advice or make decisions in such areas as medical diagnosis.

**Backward chaining** (or backward reasoning) is an inference method that can be described colloquially as working backward from the goal(s). It is used in automated theorem provers, inference engines, proof assistants and other artificial intelligence applications.

# COMPUTATION IN PREDICATE LOGIC

Prolog is a programming language based on predicate logic.

- A Prolog program attempts to prove a goal, such as brother(Barney,x), from a set of facts and rules.
- In the process of proving the goal to be true, using substitution and the other rules of inference, Prolog substitutes values for the variables in the goal, thereby "computing" an answer.

How does Prolog know which facts and which rules to use in the proof?

- Prolog uses *unification* to determine when two clauses can be made equivalent by a substitution of variables.
- The unification procedure is used to instantiate the variables in a goal clause based on the facts and rules in the database.

# HORN CLAUSES

To simplify the resolution process in Prolog, statements must be expressed in a simplified form, called **Horn clauses**.

- Statements are constructed from **terms**.
- Each statement (clause) has (at most) one term on the left hand side of an implication symbol ( :- ).
- Each statement has a conjunction of zero or more terms on the right hand side.

Prolog has three kinds of statements, corresponding to the structure of the Horn clause used.

- A **fact** is a clause with an empty right hand side.
- A **question** (or **goal**) is a clause with an empty left hand side.
- A **rule** is a clause with terms on both sides.

## TERMS

There are three kinds of terms in Prolog:

- A **constant** is an **atom** or a number. An atom is a quoted character string or a string of letters, digits, and underscores that starts with a lower-case letter. A number resembles the real or integer constants used in most programming languages.
- A **variable** is a string of letters, digits, and underscores that starts with an upper-case letter. There are no type declarations; types are discovered implicitly by the interpreter.
- A **structure** represents an atomic proposition of predicate calculus, and has the form "atom(parameter list)".

## FACTS AND RULES

The Prolog environment maintains a set of **facts** and **rules** in its database.

- Facts are axioms; relations between terms that are assumed to be true.
- Rules are theorems that allow new inferences to be made.

Example facts:

```
male(adam).
female(anne).
parent(adam,barney).
```

Example rules:

```
son(X,Y) :- parent(Y,X) , male(X)
daughter(X,Y) :- parent(Y,X) , female(X)
```

The first rule is read as follows: for all X and Y, X is the son of Y if there exists X and Y such that Y is the parent of X and X is male.
The second rule is read as follows: for all X and Y, X is the daughter of Y if there exists X and Y such that Y is the parent of X and X is female.

## OBSERVATIONS ABOUT PROLOG RULES

- The implication is from right to left!
- The scope of a variable is the clause in which it appears.
- Variables whose first appearance is on the left hand side of the clause have implicit universal quantifiers.
- Variables whose first appearance is on the right hand side of the clause have implicit existential quantifiers.

## demo.pl

demo.pl is a simpler subset of conditions and goals from the project “base.pl” to show the demonstration and show the line of control and order of execution. demo.pl is also attached along with this report.

```
/*begin with start  clause:- demo.pl*/
start :- checkfor(Country), nl,write('I guess that the country is: '), nl,
                write(Country),nl, write('Made with \u2764 by Shivam Kumar Giri|NIT Puducherry') .

/* Check for the following countries */
checkfor(italy) :- italy, !.
checkfor(france) :- france, !.
checkfor(united_state_of_america) :- united_state_of_america, !.
checkfor(canada) :- canada, !.
checkfor(japan) :- japan, !.
checkfor(singapore) :- singapore, !.
checkfor(australia) :- australia, !.
checkfor(unknown). /* no diagnosis */

/* Country identification rules */
italy:- europe,  verify(touches_mediterranean_sea),  verify(source_for_pasta_and_pizza).
france :-  europe,  verify(the_oe_uses_the_first_democracy),
           verify(famous_for_eiffel_tower_and_louvre_museum).
united_state_of_america :- northern_america, verify(the_country_with_most_billionare).
canada :- northern_america, verify(country_with_largest_coast_line),
           verify(country_close_to_north_pole_in_arctic_ocean).
japan :-  asia, pacific,  verify(land_of_kimono),
           verify(the_one_undergone_many_earthquake_and_fukushima_nuclear_disaster).
singapore :- island,asia,  verify(most_developed_country_in_indochina),
           verify(nation_with_negligible_unemployment).
australia :-pacific, verify(biggest_continental_landmass), verify(known_for_koala_and_kangaroos).

/* classification rules */
asia :- verify(situated_in_biggest_continent), !.
asia:-verify(belonging_to_asia).
island :- verify(an_island).
northern_america :- verify(in_northern_america).
europe :- verify(belonging_to_europe).
pacific :- island, verify(situated_in_pacific_ocean).

/* How to verify something */
verify(S) :- (yes(S) -> true ;( no(S) -> fail ;ask(S))).

/* how to ask questions */
ask(Question) :-
      write('Does the countries you thought is: '),   write(Question), write('? '),
       read(Response), nl,
       ( (Response == yes ; Response == y)
       -> assert(yes(Question)) ;  assert(no(Question)), fail).

:- dynamic yes/1,no/1.
```

## SUBSET OF "base.pl" TO SHOW ORDER OF EXECUTION AND LINE OF CONTROL

Begin

```
/*begin with start clause:-  demo.pl*/
start :- checkfor(Country), nl,write('I guess that the country is: '), nl,
                    write(Country),nl, write('Made with \u2764 by Shivam Kumar Giri|NIT Puducherry') .


/* Check for the following countries */
checkfor(italy) :- italy, !.
checkfor(france) :- france, !.
checkfor(united_state_of_america) :- united_state_of_america, !.
checkfor(canada) :- canada, !.
checkfor(japan) :- japan, !.
checkfor(singapore) :- singapore, !.
checkfor(australia) :- australia, !.
checkfor(unknown). /* no diagnosis */


/* Country identification rules */
italy:- europe,  verify(touches_mediterranean_sea),  verify(source_for_pasta_and_pizza).
france :-  europe,  verify(the_source_of_first_democracy), verify(famous_for_eiffel_tower_and_louvre_museum).
united_state_of_america :- northern_america,                verify(the_country_with_most_billionare).
canada :- northern_america, verify(country_with_largest_coast_line),
          verify(country_close_to_north_pole_in_arctic_ocean).
japan :- asia, pacific,  verify(land_of_kimono),
           verify(the_one_undergone_many_earthquake_and_fukushima_nuclear_disaster).
singapore :- island, asia,  verify(most_developed_country_in_indochina),
            verify(nation_with_negligible_unemployment).
australia :- pacific, verify(biggest_continental_landmass), verify(known_for_koala_and_kangaroos).


/* classification rules */
asia :- verify(situated_in_biggest_continent), !.
asia:-verify(belonging_to_asia).
island :- verify(an_island).
northern_america :- verify(in_northern_america).
europe :- verify(belonging_to_europe).
pacific :- island, verify(situated_in_pacific_ocean).


/* How to verify something */
verify(S) :- (yes(S) -> true ;( no(S) -> fail ;ask(S))).


/* how to ask questions */
ask(Question) :-
      write('Does the countries you thought is: '),   write(Question), write('? '),
       read(Response), nl,
       ( (Response == yes ; Response == y)
       -> assert(yes(Question)) ;  assert(no(Question)), fail).


:- dynamic yes/1,no/1.
```

Red arrow: Error Encountered.
Green arrow: check for italy
Blue arrow: The track, which successfully tracked the req. country.
Purple arrow: check for france

## OUTPUT of “demo.pl” for U.S.A.

```
?- consult(demo).
true.

?- start.
Does the countries you thought is: belonging_to_europe? n.

Does the countries you thought is: in_northern_america? |: y.

Does the countries you thought is: the_country_with_most_billionare? |: y.


I guess that the country is:
united_state_of_america
Made with ♥ by Shivam Kumar Giri|NIT Puducherry
true.
```

## BENEFITS OF USING EXPERT SYSTEM WITH PROLOG

- Easy to figure out and implement common rule.
- Rules once verified never repeat again in execution
- Easy to implement using first order logic
- Shorter and efficient compare to oops.
- Can impose various logical conditions.

## PROLOG SYNTAX

```
start :- checkfor(Country), nl .
```

Next line — prolog statements ends with full stop

begin with start clause — attempt the unification of goal by substitution checkfor(X)

```
checkfor(italy) :- italy, !.
```

Indicate to check for any further checkfor() if fails.

Structure term to check for Italy — check further verification rules of Italy.

```
italy:- europe , verify(source_for_pasta_and_pizza).
```

attempt the unification of goal by substitution verify(X)

conjuction of two condition

check further verification rules of europe. — stop the verificaion if all were checked

```
pacific :- island, verify(situated_in_pacific_ocean).
```

All, pacific countries are island, so all properties of “island of pacific”, grouped together

```
verify(S) :- (yes(S) -> true ;( no(S) -> fail ;ask(S))).
```

Check whether if it is already executed

Structure term to check for Italy attempt the unification of goal by substitution ask(x)

```
ask(Question) :-  write('Does the countries you thought is: '),  write(Question), write('? '),
```

print the question in the terminal passed via verify

```
read(Response)
```

read the response from terminal and store in “Response”

```
( (Response == yes ; Response == y)  -> assert(yes(Question)) ;  assert(no(Question)), fail).
```

check if response=”yes” or ”y”

assert question/condition=false

if response=”yes” or ”y” assert question/condition=true for future reference

```
:- dynamic yes/1,no/1.
```

Making program dynamic by making prolog verification as alteration of yes() and no()

Minimum No. of steps for reaching unknown: **13**

## RESULT

The “demo.pl” expert system is created for testing as a subset of conditions and rules from “base.pl” which executed successfully and required output is produced as per the line of control and order of execution shown above. The entire project “base.pl” is attached along with this report with the output file “output.txt”.

## DEVELOPER


| Name | Roll no. | Email |
|---|---|---|
| **Shivam Kumar Giri** | CS15B1022 | shivamgiri2015@gmail.com |