Programming in Prolog: An Introduction

Part 1

What is Prolog

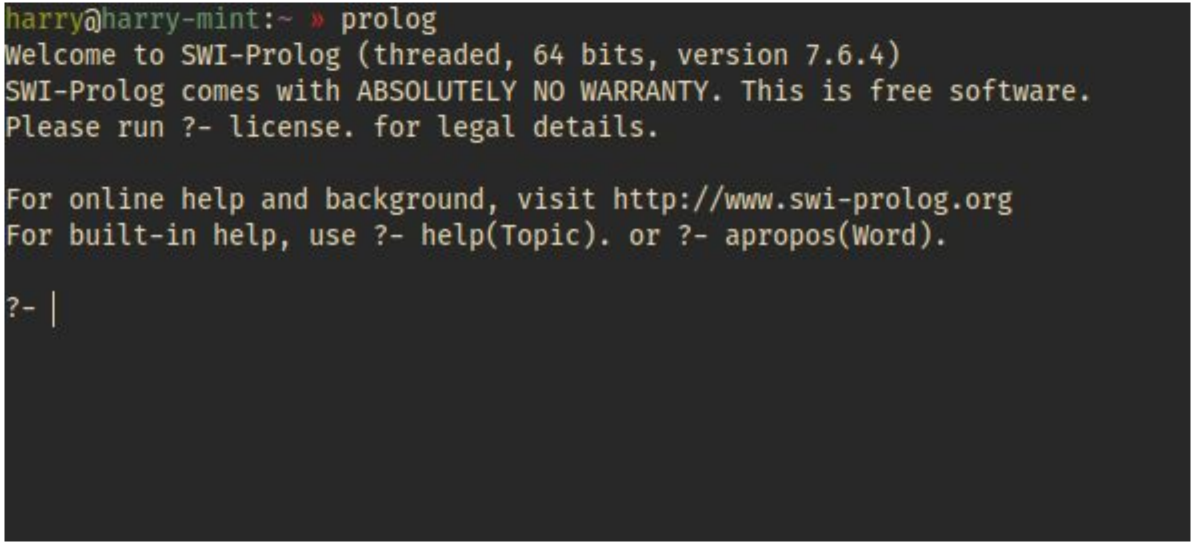
- Prolog: PROgramming in LOGic
- Every Prolog program consists of data based on facts and rules unlike computing to find a solution
- Prolog follows top-down approach
- Prolog is a declarative programming language that means we can specify what problem we want to solve rather than how to solve it
- Prolog uses backtracking strategy to search for proofs
- There can be more than one way to deduce the answer and Prolog can find more solutions for a particular problem if it exists
- Prolog is a weakly typed language with static scope rules and dynamic type checking
- Prolog applications: Problem solving, Machine learning, Robot planning, Expert systems, NLP, Automated reasoning etc.
- Prolog can be both compiled and interpreted

Installation

- To install the Prolog, run the following command
  - If you are using Debian based systems (Ubuntu, Linux Mint, MX Linux, Elementary etc.) use: ***sudo apt-get install swi-prolog***
  - If you are using Arch based systems (Manjaro, Arch Linux etc.) use: ***sudo pacman -S swi-prolog***
- After installation type ***'prolog'*** in terminal to get interactive Prolog console (just like Python interactive console)

- You can also use online versions of Prolog compilers
  - https://www.tutorialspoint.com/execute_prolog_online.php
  - https://ideone.com/l/prolog-swi

Prolog interactive console



```
harry@harry-mint:~ » prolog
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

?- |
```

“ **?-** “ is indicating that system is ready to take the input from user and every instruction should end with period (“**.**”)

You can get more help by typing the following command:

help(help). (period at the end)

Hello World !

- ***“write”*** is a built-in predicate which is used to print something onto screen
- Printing “Hello World !”:
  - Write the following line on the console

    write(“Hello World !”).

- It will print ***“Hello World !”*** onto screen
- Along with the output, ***“true.”*** will also be printed onto screen. This means the query we have given, it is successfully completed
- Exiting the interactive console:
  - To exit the Prolog console, we can type either:
    - ***halt.*** command or
    - ***Ctrl + D*** key combination

A Simple Program (family.pl)

```prolog
male(albert). %a fact stating albert is a male

male(edward).

female(alice). %a fact stating alice is a female

female(victoria).

parent(albert,edward). %a fact: albert is parent of edward

parent(victoria,edward).

father(X,Y) :- parent(X,Y), male(X). %a rule: X is father of Y if X if a male parent of Y. Here comma (,) is AND predicate

mother(X,Y) :- parent(X,Y), female(X). %a similar rule for X being mother of Y
```

Loading family.pl

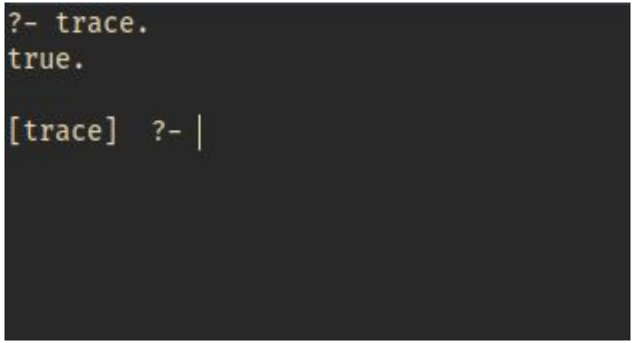
- There are two ways to load any prolog file into interactive console
  - With ***‘consult’*** in-built predicate
  - With ***[‘filename.pl’]*** statement
- To load ‘family.pl’ use either one of the following commands:
  - ***consult(‘family’).***
  - ***[‘family.pl’].***
- After loading the file, the output should be ‘true.’
- If there is any error, check your file location and it should be in same directory from which Prolog console is running
- You can check current working directory from Prolog console by running the following command:
  - ***‘pwd.’***
- We can edit the program by typing the following command:
  - ***edit(family).   or   edit(‘family.pl’).***

Querying

- After loading ‘family.pl’ we can ask following queries:
- ***male(albert).*** *% true. because we have defined this fact*
- ***male(harry).*** *% false. because we did not defined this fact*
- ***female(victoria).*** *% true.*
- ***female(albert).*** *% false. because albert is a male*
- ***female(X).*** *% here X (capital x) is a variable. We are asking for all females in our DB*
  - ***X = alice ;*** *% type semicolon (;) for more answers. Pressing enter or period(.) will terminate the backtracking*
  - ***X = victoria.***
- ***parent(X, Y).***
  - ***X = albert,***
  - ***Y = edward ;***
  - ***X = victoria,***
  - ***Y = edward .***

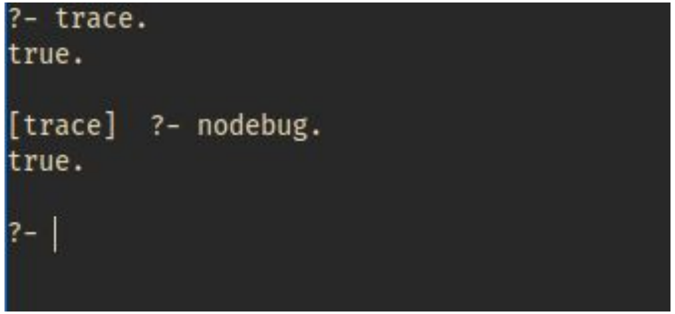
Tracing

- We can see the background execution of Prolog query by **tracing.**
- To put the console in **trace** mode, type the following command
  - ***trace.***
- To exit out of **trace** mode, type the following command
  - ***nodebug.***
- We can exit from **trace** mode by using **“*notrace.*”** command also, but this will put our console to ***“debug”*** mode
- ***“debug”*** command is used to debug the programs



```
?- trace.
true.

[trace]  ?- |
```



```
?- trace.
true.

[trace]  ?- nodebug.
true.

?- |
```

Tracing (contd..)

Tracing the query ***"father(X, Y)."***

Trace	Comment
father(X, Y).	Loading the query
Call: (8) father(_3324, _3326) ? creep	Replacing X and Y with unique variables
Call: (9) parent(_3324, _3326) ? creep	Call to *parent(X, Y)*
Exit: (9) parent(albert, edward) ? creep	Replacing X and Y with *albert* and *edward* and succeeds
Call: (9) male(albert) ? creep	Call to *male(albert)* (because X is replaced with *albert)*
Exit: (9) male(albert) ? creep	Succeeds
Exit: (8) father(albert, edward) ? creep	Succeeds
X = albert, Y = edward	Output

Prolog Syntax

- Constants:
  - Sequence of letters, digits or underscore (‘_’) that start with **lowercase letters**
  - Eg: x, alpha, 1.2 etc.
- Variables:
  - Sequence of letters, digits or underscore that start with **uppercase letters**
  - Eg: X, _x, Anna etc.
  - Underscore itself is a variable, and called as “**anonymous**” variable
- Symbols:
  - “**:-**” in Prolog represents **IF** in predicate calculus
  - “**,**” (comma) represents **AND**
  - “**;**” (semi-colon) represents **OR**
  - “**Not**” represents **NOT**

Prolog Syntax: Facts and Rules

- Generally a Prolog program consists of a collection of facts and rules
- Facts:
  - Fact is a predicate terminated by period (“**.**”)
  - Eg:
    - *wizard(harry).*   % harry is a wizard
    - *mother(lily, harry).*   % lily is mother of harry
- Rules:
  - Eg:
    - *grandparent(A, B) :- parent(A, C) , parent(C, B).*   % comma in between and period at end
  - Here *grandparent()* is called **Rule Head** and *parent()* is called **Rule Body**

Prolog Arithmetic

- In Prolog we can not declare variables and initialize them like we do with other programming languages
- For example, in Prolog the following statement will print:
  - *A = 1 + 2.* % will print “A = 1+2.” but not “3”
- To initialize the variables, we have to use an inbuilt predicate ***“is”***
- Eg:
  - *A is 1 + 2.* % will print “A = 3.”
- If we try to use already existing variable *A* it will throw an error, like this:
  - *B is A + 1.*   % Error: Arguments are not sufficiently instantiated
- This is because Prolog is a static scoped language
- But we can still use *A* to initialize *B* as following:
  - *A is 1 + 2, B is A + 1.*   % output will be A = 3 and B = 4

Factorial in Prolog

Algorithm:

Base case: factorial(0, 1).   % factorial of 0 is 1

Recursion: factorial(N, F).   % factorial of N is F (Input is N and output is F)

1. Check N > 0
2. Decrement N i.e. N_temp is N - 1.
3. Call: factorial (N_temp, F_temp).
4. F is F * N_temp

Exercise

- Write Prolog script for Half Adder (Hint: In half adder, SUM = A XOR B and CARRY = A AND B)
- Write Prolog script for Full Adder

References

1. https://www.javatpoint.com/prolog
2. https://www.cpp.edu/~jrfisher/www/prolog_tutorial/pt_framer.html
3. https://www.swi-prolog.org/pldoc/man?section=quickstart
4. https://ideone.com/l/prolog-swi
5. https://www.tutorialspoint.com/execute_prolog_online.php
6. http://www.gprolog.org/
7. https://stackoverflow.com/questions/37427094/prolog-a-compiled-or-interpreted-language-or-both
8. https://www.cs.toronto.edu/~hojjat/384w10/PrologTutorial1.pdf
9. https://www.cse.unsw.edu.au/~billw/dictionaries/prolog/tracing.html
10. https://core.ac.uk/download/pdf/82038375.pdf
11. https://www.cs.hmc.edu/~keller/courses/cs60/slides/18.28.html