cplint learning Day 1

Fani

<2024-06-17 Mon>

1 Installation

1.1 1) Install SWI prolog

`https://www.swi-prolog.org/download/stable`

1.2 2) Install cplint

It is explained here:

- `http://friguzzi.github.io/cplint/_build/html/index.html`

  I also had to make sure this runs, after installing cplint.

```
pack_rebuild(bddem).
```

2 cplint Documentation

`http://friguzzi.github.io/cplint/_build/html/index.html`

It looks like it's possible to use R with cplint:

```
Using R¶

You have to load library cplint_r (a SWI-Prolog pack) with

:- use_module(library(cplint_r)).

Then you can use predicates

bar_r/1
bar_r/2
argbar_r/1

that work as their C3.js counterpart but do not return the graph as an
argument as the graph is printed with a different mechanism.

You also have

histogram_r(+List:list,+Options:list) is det
```

3 Learning:

youtube.com: Prolog Tutorial

```
% Prolog programs are a collection of Facts, and Rules that we can
% Query.

% Prolog focuses on describing facts and relationships about problems
% rather then on creating a series of steps to solve that problem.

% These Facts and Rules are stored in a file called a Database
% or Knowledge Base

% You load a knowledge base like this [knowledge]. or this
% consult(’knowledge.pl’).
% halt. exits the prolog system
% listing. Displays the contents of the database
% All these commands are called predicates

% ---------- INTRODUCTION ----------
% write prints text between quotes to the screen
% nl stands for new line and \’s allows you to use quotes
% write(’Hello World’),nl,write(’Let\’s Program’).

% This is a fact where loves is a predicate and romeo and
% juliet are atoms (constants) and loves arguments
loves(romeo, juliet).

% This is a rule where :- (if) says if the item on the right is
% true, then so is the item on the left
loves(juliet, romeo) :- loves(romeo, juliet).

% Evaluating whether the goal was met in the terminal
% loves(juliet, romeo). = yes

% Facts and Rules are called clauses

% A Variable is an object we can’t name at the time of execution
% Variables are uppercase while atoms are lowercase
% loves(romeo, X). = X = juliet

% ---------- FACTS ----------
% Write the relationship first followed by the objects between
% parenthese followed by a dot

% albert, male, female are atom constants that must begin with a
% lowercase letter unless they are between single quotes
% An atom can contain letters, numbers, +, -, _, *, /, <, >, :, ., ˜, &
% AN ATOM CANNOT START WITH _

% The name before parenthese is called the predicate
% The names in parenthese are called arguments

% Let’s define information about the people above
```

```

male(albert).
male(bob).
male(bill).
male(carl).
male(charlie).
male(dan).
male(edward).

female(alice).
female(betsy).
female(diana).

% We can find out if alice is a woman with
% female(alice). = yes
% listing(male). = list all clauses defining the predicate male
% male(X), female(Y). = Show all combinations of male and female

% ---------- RULES ----------
% Rules are used when you want to say that a fact depends on a group of facts

% NOTE : You'll get the discontiguous predicate warning if you
% don't keep your predicates together

happy(albert).
happy(alice).
happy(bob).
happy(bill).
with_albert(alice).

% We can define the Fact that when Bob is happy he runs
% :- stands for if
runs(albert) :- happy(albert).
% runs(albert). = yes

% We can check if 2 conditions are true by putting a comma (and)
% between questions (CONJUCTIONS)
dances(alice) :-
  happy(alice),
  with_albert(alice).

% We can define predicates to keep commands brief
does_alice_dance :- dances(alice),
       write('When Alice is happy and with Albert she dances').
% Just type does_alice_dance. in the terminal

% Both rules must be true to get a yes result
swims(bob) :-
  happy(bob),
  near_water(bob).
% swims(bob). = no

% We can create 2 instances and if either comes back true the result
% will be yes
swims(bill) :-
  happy(bill).

swims(bill) :-
  near_water(bill).
% swims(bill). = yes

% ---------- VARIABLES ----------
% A variable is an object we are unable to name when writing a program.
% An instantiated variable is one that stands for an object.
% A variable begins with an uppercase letter or _ and can contain
% the same symbols as atoms.
% The same variable name used in 2 different questions represents 2
% completely different variables.

% An uninstantiated variable can be used to search for any match.

% Return all females (Type ; to cycle through them)
% female(X). X = alice X = betsy X = diana

parent(albert, bob).
parent(albert, betsy).
parent(albert, bill).

parent(alice, bob).
parent(alice, betsy).
parent(alice, bill).

parent(bob, carl).
parent(bob, charlie).

% When you are cycling through the results the no at the end signals
% that there are no more results
% parent(X, bob). X = albert, X = alice

% parent(X, bob), dances(X). X = alice

% Who is Bobs parent? Does he have parents?
% parent(Y, carl), parent(X, Y). = X = albert, Y = bob, X = alice
% Y = bob

% Find Alberts grandchildren
% Is Albert a father? Does his children have any children?
% parent(albert, X), parent(X, Y). = X = bob, Y = carl, X = bob,
% Y = charlie

% Use custom predicate for multiple results
get_grandchild :- parent(albert, X), parent(X, Y),
              write(’Alberts grandchild is ’),
              write(Y), nl.

% Do Carl and Charlie share a parent
% Who is Carls parent? Is this same X a parent of Charlie
% parent(X, carl), parent(X, charlie). = X = bob

% Use format to get the results
% ~w represents where to put each value in the list at the end
% ~n is a newline
% ~s is used to input strings
get_grandparent :- parent(X, carl),
                parent(X, charlie),
                format(’~w ~s grandparent~n’, [X, "is the"]).

% Does Carl have an Uncle?
% Who is Carls parent? Who is Carls fathers brother?
brother(bob, bill).
% parent(X, carl), brother(X, Y). = X = bob, Y = bill

% Demonstrate axioms and derived facts
% We can also use variables in the database
% If you get the singleton warning, that means you defined a variable
% that you didn’t do anything with. (This is ok sometimes)
grand_parent(X, Y) :-
  parent(Z, X),
  parent(Y, Z).
% grand_parent(carl, A). = A = albert, A = alice

% X blushes if X is human
blushes(X) :- human(X).
human(derek).

% If we say one thing is true when somehing else is true, we can also
% find that match if we only assign one thing to be true here.
% blushes(derek). = yes

% Another example on cause and effect
stabs(tybalt,mercutio,sword).
hates(romeo, X) :- stabs(X, mercutio, sword).
% hates(romeo, X). = X = tybalt

% We can use _ (anonymous variable) if we won’t use the variable
% more than once
% The value of an anonymous var is not output
% Check if any males exist in the database : male(_). = yes

% ---------- WHERE IS IF? ----------
% You can use a type of case statement instead

what_grade(5) :-
  write(’Go to kindergarten’).
what_grade(6) :-
  write(’Go to first grade’).
what_grade(Other) :-
  Grade is Other - 5,
  format(’Go to grade ~w’, [Grade]).
```

```

% ---------- COMPLEX TERMS / STRUCTURES ----------
% A Structure is an object made up from many other objects (components)
% Structures allow us to add context about what an object is to avoid
% confusion. has(albert,olive) Does Albert have a pet named Olive?
% Does Albert have the food named Olive?

% Structures have a functor followed by a list of arguments
% The number of arguments a Structure has is its arity
% female(alice). has an arity of one

% Albert owns a pet cat named Olive
% This is a recursive definition

owns(albert, pet(cat, olive)).

% owns(albert, pet(cat, X)). : X = olive

customer(tom, smith, 20.55).
customer(sally, smith, 120.55).

% An anonymous variable is used when we don’t want a value returned
% Is there a customer named sally and what is her balance
% customer(sally,_,Bal).

% tab puts the defined number of spaces on the screen
% ~2f says we want a float with 2 decimals
get_cust_bal(FName, LName) :- customer(FName, LName, Bal),
  write(FName), tab(1),
  format(’~w owes us $~2f ~n’, [LName, Bal]).

% Use a complex term to define what it means to be a vertical
% versus a horizontal line
vertical(line(point(X, Y), point(X, Y2))).
horizontal(line(point(X, Y), point(X2, Y))).

% vertical(line(point(5, 10), point(5, 20))). = yes
% horizontal(line(point(10, 20), point(30, 20))).

% We can also ask what the value of a point should be to be vertical
% vertical(line(point(5, 10), point(X, 20))). = X = 5

% We could also ask for the X and Y points
% vertical(line(point(5, 10), X)). = X = point(5,_)

% ---------- COMPARISON ----------
% alice = alice. = yes
% ’alice’ = alice. = yes (Prolog considers these to be the same)
% \+ (alice = albert). = yes (How to check for not equal)

% 3 > 15. = no
% 3 >= 15. = no
% 3 =< 15. = yes

% W = alice. = yes
% This says that we can assign the value of alice to W and not that
% W is equal to alice

% Rand1 = Rand2. = yes
% This says that any variable can be assigned anything and one of
% those things is another variable

% If variables can be matched up between 2 complex terms and the
% functors are equal then the complex terms are equal
% rich(money, X) = rich(Y, no_debt).

% ---------- TRACE ----------
% Using trace we can see how Prolog evaluates queries one at a time

warm_blooded(penguin).
warm_blooded(human).

produce_milk(penguin).
produce_milk(human).

have_feathers(penguin).
have_hair(human).

mammal(X) :-
  warm_blooded(X),
  produce_milk(X),
  have_hair(X).


% trace.
% mammal(human).
%       1    1  Call: mammal(human) ?
%       2    2  Call: warm_blooded(human) ?
%       2    2  Exit: warm_blooded(human) ?
%       3    2  Call: produce_milk(human) ?
%       3    2  Exit: produce_milk(human) ?
%       4    2  Call: have_hair(human) ?
%       4    2  Exit: have_hair(human) ?
%       1    1  Exit: mammal(human) ?
% yes

% mammal(penguin).
%       1    1  Call: mammal(penguin) ?
%       2    2  Call: warm_blooded(penguin) ?
%       2    2  Exit: warm_blooded(penguin) ?
%       3    2  Call: produce_milk(penguin) ?
%       3    2  Exit: produce_milk(penguin) ?
%       4    2  Call: have_hair(penguin) ?
%       4    2  Fail: have_hair(penguin) ?
%       1    1  Fail: mammal(penguin) ?
% no
```

```
%
% notrace. Turns off trace

% Output what ever matches the clauses
% warm_blooded(X), produce_milk(X), write(X),nl.

% ---------- RECURSION ----------

/*
parent(albert, bob).
parent(albert, betsy).
parent(albert, bill).

parent(alice, bob).
parent(alice, betsy).
parent(alice, bill).

parent(bob, carl).
parent(bob, charlie).
*/

% Works for exact matches
related(X, Y) :- parent(X, Y).
% related(albert, bob). = true

% Cycles through possible results until related returns a true
related(X, Y) :-
  parent(X, Z),
  related(Z, Y).

% related(albert,carl). = true

% 1. parent(albert, Z). = true = Z = bob, betsy, bill
% 2. related(Z, carl). = true when Z = bob

% ---------- MATH ----------
% Prolog provides 'is' to evaluate mathematical expressions
% X is 2 + 2. = X = 4

% You can use parenthese
% X is 3 + (2 * 10). =  X = 23

% You can also make comparisons
% 50 > 30. = yes
% (3*10) >= (50/2). = yes
% \+ (3 = 10). = yes (How to check for not equal)
% 5+4 =:= 4+5. = yes (Check for equality between expressions)
% 5+4 =\= 4+5. = yes (Check for non-equality between expressions)
% 5 > 10 ; 10 < 100. (Checks if 1 OR the other is true)

% X is mod(7,2). = X = 1 (Modulus)

double_digit(X,Y) :- Y is X*2.
```

```
% double_digit(4,Y). = Y = 8
% Take the 1st argument, multiply it times 2 and return it as the
% 2nd argument

% Get random value between 0 and 10
% random(0,10,X).

% Get all values between 0 and 10
% between(0,10,X).

% Add 1 and assign it to X
% succ(2,X).

% Get absolute value of -8
% X is abs(-8).

% Get largest value
% X is max(10,5).

% Get smallest value
% X is min(10,5).

% Round a value
% X is round(10.56).

% Convert float to integer
% X is truncate(10.56).

% Round down
% X is floor(10.56).

% Round up
% X is ceiling(10.56).

% 2^3
% X is 2** 3.

% Check if a number is even
% 10//2 = 5 (is 10 = 2 * 5)
is_even(X) :- Y is X//2, X =:= 2 * Y.

% sqrt, sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh,
% asinh, acosh, atanh, log, log10, exp, pi, e

% ---------- INPUT / OUTPUT ----------
% write('You saw me'), nl.

% writeq('I show quotes'), nl.

% You can read data with read
say_hi :-
  write('What is your name? '),
  read(X),
  write('Hi '),
  write(X).

% say_hi.
% What is your name 'Derek'.
% Hi Derek

fav_char :-
  write('What is your favorite character? '),

  % Receives a char and saves its ascii value to X
  get(X),
  format('The Ascii value ~w is ', [X]),

  % Outputs Ascii value as the char
  put(X),nl.

% Write to a file by defining the file, text to write, connection
% to the file (Stream)
write_to_file(File, Text) :-
  open(File, write, Stream),
  write(Stream, Text), nl,
  close(Stream).

% Read from a file
read_file(File) :-
        open(File, read, Stream),

        % Get char from the stream
        get_char(Stream, Char1),

        % Outputs the characters until end_of_file
        process_stream(Char1, Stream),
        close(Stream).

% Continue getting characters until end_of_file
% ! or cut is used to end backtracking or this execution
process_stream(end_of_file, _) :- !.

process_stream(Char, Stream) :-
        write(Char),
        get_char(Stream, Char2),
        process_stream(Char2, Stream).

% ---------- HOW TO LOOP ----------

% Use recursion to loop
count_to_10(10) :- write(10), nl.

count_to_10(X) :-
  write(X),nl,
  Y is X + 1,
  count_to_10(Y).
```

```

% Receives Low (lowest value) and High (highest value)
count_down(Low, High) :-
  % Assigns values between Low and High to Y
  between(Low, High, Y),
  % Assigns the difference to Z
  Z is High - Y,
  write(Z),nl,
  % Continue looping until Y = 10
  Y = 10.

count_up(Low, High) :-
  between(Low, High, Y),
  Z is Y + Low,
  write(Z), nl,
  Y = 10.

% Loop until they guess a number
% start is a dummy value used to start the looping
guess_num :- loop(start).

% When they guess 15 they execute this message and exit
loop(15) :- write(’You guessed it!’).

loop(X) :-
  x \= 15,
  write(’Guess Number ’),
  read(Guess),
  write(Guess),
  write(’ is not the number’), nl,
  loop(Guess).

% guess_num.
% Guess Number 12.
% 12 is not the number
% Guess Number 15.
% 15 is not the number
% You guessed it!

% ---------- CHANGING THE DATABASE ----------
% Any predicate you plan to motify should be marked as dynamic before
% this predicate is used in any way
:- dynamic(father/2).
:- dynamic(likes/2).
:- dynamic(friend/2).
:- dynamic(stabs/3).

father(lord_montague,romeo).
father(lord_capulet,juliet).

likes(mercutio,dancing).
likes(benvolio,dancing).
likes(romeo,dancing).
likes(romeo,juliet).
likes(juliet,romeo).
likes(juliet,dancing).

friend(romeo,mercutio).
friend(romeo,benvolio).
% friend(X, romeo) :- friend(romeo, X).

stabs(tybalt,mercutio,sword).
stabs(romeo,tybalt,sword).

% Add new clause to the database at the end of the list for the same
% predicate
% assertz(friend(benvolio, mercutio)).
% friend(benvolio, mercutio). = yes

% Add clause at the start of the predicate list
% asserta(friend(mercutio, benvolio)).
% friend(mercutio, benvolio). = yes

% Delete a clause
% retract(likes(mercutio,dancing)).
% likes(mercutio,dancing). = no

% Delete all clauses that match
% retractall(father(_,_)).
% father(lord_montague,romeo). = no

% Delete all matching clauses
% retractall(likes(_,dancing)).
% likes(_,dancing). = no

% ---------- LISTS ----------
% You can store atoms, complex terms, variables, numbers and other
% lists in a list
% They are used to store data that has an unknown number of elements

% We can add items to a list with the | (List Constructor)
% write([albert|[alice, bob]]), nl.

% Get the length of a list
% length([1,2,3], X).

% We can divide a list into its head and tail with |
% [H|T] = [a,b,c].

% H = a
% T = [b,c]

% We can get additional values by adding more variables to the left
% of |

%[X1, X2, X3, X4|T] = [a,b,c,d].
```

```

% We can use the anonymous variable _ when we need to reference a
% variable, but we don’t want its value
% Let’s get the second value in the list
% [_, X2, _, _|T] = [a,b,c,d].

% We can use | to access values of lists in lists
% [_, _, [X|Y], _, Z|T] = [a, b, [c, d, e], f, g, h].

% Find out if a value is in a list with member
% List1 = [a,b,c].
% member(a, List1). = yes

% We could also get all members of a list with a variable
% member(X, [a, b, c, d]).

% Reverse a list
% reverse([1,2,3,4,5], X).

% Concatenate 2 lists
% append([1,2,3], [4,5,6], X).

% Write items in list on separate line
write_list([]).

write_list([Head|Tail]) :-
  write(Head), nl,
  write_list(Tail).
% write_list([1,2,3,4,5]). = Outputs the list

% ---------- STRINGS ----------
% Convert a string into an Ascii character list
% name('A random string', X).

% Convert a Ascii character list into a string
% name(X, [65,32,114,97,110,100,111,109,32,115,116,114,105,110,103]).

% Append can join strings
join_str(Str1, Str2, Str3) :-

  % Convert strings into lists
  name(Str1, StrList1),
  name(Str2, StrList2),

  % Combine string lists into new string list
  append(StrList1, StrList2, StrList3),

  % Convert list into a string
  name(Str3, StrList3).

% join_str('Another ', 'Random String', X). = X = 'Another Random String'

% get the 1st char from a string
```

```
/*
name('Derek', List),
nth0(0, List, FChar),
put(FChar).
*/

% Get length of the string
atom_length('Derek',X).
```

4 Example

4.1 epidemic.cpl

http://github.com/friguzzi/cplint/blob/master/prolog/examples/epidemic.cpl

```
/*
Model of the development of an epidemic or a pandemic.
From
E. Bellodi and F. Riguzzi. Expectation Maximization over binary decision
diagrams for probabilistic logic programs. Intelligent Data Analysis,
17(2):343-363, 2013.
*/


epidemic : 0.6; pandemic : 0.3 :- flu(_), cold.
% if somebody has the flu and the climate is cold, there is the possibility
% that an epidemic arises with probability 0.6 and the possibility that a
% pandemic arises with probability 0.3

cold : 0.7.
% it is cold with probability 0.7

flu(david).
flu(robert).
% david and robert have the flu for sure

/** <examples>

?- epidemic.  % what is the probability that an epidemic arises?
% expected result 0.588
?- pandemic.  % what is the probability that a pandemic arises?
% expected result 0.357

*/
```

4.2 Load the library

4.2.1 Step 1

Start swipl in the same directory where epidemic.cpl lives

```
swipl
```

4.2.2 Step 2

Then type into the swipl console "[epidemic]." and press enter.

```
[epidemic].
```

This should load the epidemic.cpl program.

4.2.3 Step 3

Then to calculate the probability of an epidemic, type the following into the console and press enter:

```
prob(epidemic,P).
```

4.2.4 Output

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.2)
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).

?- [epidemic].
true.

?- prob(epidemic,P).
P = 0.42 .

?-
```

5 cplint Glossary

```
cpl
cplint
    [#prolog]
    [prolog package]

    cplint is a package for prolog that is
    used for probabilistic logic programming.

prob/2
    [#cplint]
    [predicate]

    prob is a predicate for the cplint that
    takes 2 arguments.

    Computes the probability of an atom.

        a:0.2:-
            prob(b,P),
            P > 0.2.

```

```
Read about it:
- http://friguzzi.github.io/cplint/_build/html/index.html
```