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
```

```
51
52  male(albert).
53  male(bob).
54  male(bill).
55  male(carl).
56  male(charlie).
57  male(dan).
58  male(edward).
59
60  female(alice).
61  female(betsy).
62  female(diana).
63
64  % We can find out if alice is a woman with
65  % female(alice). = yes
66  % listing(male). = list all clauses defining the predicate male
67  % male(X), female(Y). = Show all combinations of male and female
68
69  % ---------- RULES ----------
70  % Rules are used when you want to say that a fact depends on a group of facts
71
72  % NOTE : You'll get the discontiguous predicate warning if you
73  % don't keep your predicates together
74
75  happy(albert).
76  happy(alice).
77  happy(bob).
78  happy(bill).
79  with_albert(alice).
80
81  % We can define the Fact that when Bob is happy he runs
82  % :- stands for if
83  runs(albert) :- happy(albert).
84  % runs(albert). = yes
85
86  % We can check if 2 conditions are true by putting a comma (and)
87  % between questions (CONJUCTIONS)
88  dances(alice) :-
89    happy(alice),
90    with_albert(alice).
91
92  % We can define predicates to keep commands brief
93  does_alice_dance :- dances(alice),
94         write('When Alice is happy and with Albert she dances').
95  % Just type does_alice_dance. in the terminal
96
97  % Both rules must be true to get a yes result
98  swims(bob) :-
99    happy(bob),
100   near_water(bob).
101 % swims(bob). = no
102
103 % We can create 2 instances and if either comes back true the result
```

```
104 % will be yes
105 swims(bill) :-
106   happy(bill).
107
108 swims(bill) :-
109   near_water(bill).
110 % swims(bill). = yes
111
112 % ---------- VARIABLES ----------
113 % A variable is an object we are unable to name when writing a program.
114 % An instantiated variable is one that stands for an object.
115 % A variable begins with an uppercase letter or _ and can contain
116 % the same symbols as atoms.
117 % The same variable name used in 2 different questions represents 2
118 % completely different variables.
119
120 % An uninstantiated variable can be used to search for any match.
121
122 % Return all females (Type ; to cycle through them)
123 % female(X). X = alice X = betsy X = diana
124
125 parent(albert, bob).
126 parent(albert, betsy).
127 parent(albert, bill).
128
129 parent(alice, bob).
130 parent(alice, betsy).
131 parent(alice, bill).
132
133 parent(bob, carl).
134 parent(bob, charlie).
135
136 % When you are cycling through the results the no at the end signals
137 % that there are no more results
138 % parent(X, bob). X = albert, X = alice
139
140 % parent(X, bob), dances(X). X = alice
141
142 % Who is Bobs parent? Does he have parents?
143 % parent(Y, carl), parent(X, Y). = X = albert, Y = bob, X = alice
144 % Y = bob
145
146 % Find Alberts grandchildren
147 % Is Albert a father? Does his children have any children?
148 % parent(albert, X), parent(X, Y). = X = bob, Y = carl, X = bob,
149 % Y = charlie
150
151 % Use custom predicate for multiple results
152 get_grandchild :- parent(albert, X), parent(X, Y),
153               write('Alberts grandchild is '),
154               write(Y), nl.
155
156 % Do Carl and Charlie share a parent
```

```
157 % Who is Carls parent? Is this same X a parent of Charlie
158 % parent(X, carl), parent(X, charlie). = X = bob
159
160 % Use format to get the results
161 % ~w represents where to put each value in the list at the end
162 % ~n is a newline
163 % ~s is used to input strings
164 get_grandparent :- parent(X, carl),
165                    parent(X, charlie),
166                    format('~w ~s grandparent~n', [X, "is the"]).
167
168 % Does Carl have an Uncle?
169 % Who is Carls parent? Who is Carls fathers brother?
170 brother(bob, bill).
171 % parent(X, carl), brother(X, Y). = X = bob, Y = bill
172
173 % Demonstrate axioms and derived facts
174 % We can also use variables in the database
175 % If you get the singleton warning, that means you defined a variable
176 % that you didn't do anything with. (This is ok sometimes)
177 grand_parent(X, Y) :-
178   parent(Z, X),
179   parent(Y, Z).
180 % grand_parent(carl, A). = A = albert, A = alice
181
182 % X blushes if X is human
183 blushes(X) :- human(X).
184 human(derek).
185
186 % If we say one thing is true when somehing else is true, we can also
187 % find that match if we only assign one thing to be true here.
188 % blushes(derek). = yes
189
190 % Another example on cause and effect
191 stabs(tybalt,mercutio,sword).
192 hates(romeo, X) :- stabs(X, mercutio, sword).
193 % hates(romeo, X). = X = tybalt
194
195 % We can use _ (anonymous variable) if we won't use the variable
196 % more than once
197 % The value of an anonymous var is not output
198 % Check if any males exist in the database : male(_). = yes
199
200 % ---------- WHERE IS IF? ----------
201 % You can use a type of case statement instead
202
203 what_grade(5) :-
204   write('Go to kindergarten').
205 what_grade(6) :-
206   write('Go to first grade').
207 what_grade(Other) :-
208   Grade is Other - 5,
209   format('Go to grade ~w', [Grade]).
```

```
210
211 % ---------- COMPLEX TERMS / STRUCTURES ----------
212 % A Structure is an object made up from many other objects (components)
213 % Structures allow us to add context about what an object is to avoid
214 % confusion. has(albert,olive) Does Albert have a pet named Olive?
215 % Does Albert have the food named Olive?
216
217 % Structures have a functor followed by a list of arguments
218 % The number of arguments a Structure has is its arity
219 % female(alice). has an arity of one
220
221 % Albert owns a pet cat named Olive
222 % This is a recursive definition
223
224 owns(albert, pet(cat, olive)).
225
226 % owns(albert, pet(cat, X)). : X = olive
227
228 customer(tom, smith, 20.55).
229 customer(sally, smith, 120.55).
230
231 % An anonymous variable is used when we don't want a value returned
232 % Is there a customer named sally and what is her balance
233 % customer(sally,_,Bal).
234
235 % tab puts the defined number of spaces on the screen
236 % ~2f says we want a float with 2 decimals
237 get_cust_bal(FName, LName) :- customer(FName, LName, Bal),
238   write(FName), tab(1),
239   format('~w owes us $~2f ~n', [LName, Bal]).
240
241 % Use a complex term to define what it means to be a vertical
242 % versus a horizontal line
243 vertical(line(point(X, Y), point(X, Y2))).
244 horizontal(line(point(X, Y), point(X2, Y))).
245
246 % vertical(line(point(5, 10), point(5, 20))). = yes
247 % horizontal(line(point(10, 20), point(30, 20))).
248
249 % We can also ask what the value of a point should be to be vertical
250 % vertical(line(point(5, 10), point(X, 20))). = X = 5
251
252 % We could also ask for the X and Y points
253 % vertical(line(point(5, 10), X)). = X = point(5,_)
254
255 % ---------- COMPARISON ----------
256 % alice = alice. = yes
257 % 'alice' = alice. = yes (Prolog considers these to be the same)
258 % \+ (alice = albert). = yes (How to check for not equal)
259
260 % 3 > 15. = no
261 % 3 >= 15. = no
262 % 3 =< 15. = yes
```

```
263
264 % W = alice. = yes
265 % This says that we can assign the value of alice to W and not that
266 % W is equal to alice
267
268 % Rand1 = Rand2. = yes
269 % This says that any variable can be assigned anything and one of
270 % those things is another variable
271
272 % If variables can be matched up between 2 complex terms and the
273 % functors are equal then the complex terms are equal
274 % rich(money, X) = rich(Y, no_debt).
275
276 % ---------- TRACE ----------
277 % Using trace we can see how Prolog evaluates queries one at a time
278
279 warm_blooded(penguin).
280 warm_blooded(human).
281
282 produce_milk(penguin).
283 produce_milk(human).
284
285 have_feathers(penguin).
286 have_hair(human).
287
288 mammal(X) :-
289   warm_blooded(X),
290   produce_milk(X),
291   have_hair(X).
292
293
294 % trace.
295 % mammal(human).
296 %       1    1  Call: mammal(human) ?
297 %       2    2  Call: warm_blooded(human) ?
298 %       2    2  Exit: warm_blooded(human) ?
299 %       3    2  Call: produce_milk(human) ?
300 %       3    2  Exit: produce_milk(human) ?
301 %       4    2  Call: have_hair(human) ?
302 %       4    2  Exit: have_hair(human) ?
303 %       1    1  Exit: mammal(human) ?
304 % yes
305
306 % mammal(penguin).
307 %       1    1  Call: mammal(penguin) ?
308 %       2    2  Call: warm_blooded(penguin) ?
309 %       2    2  Exit: warm_blooded(penguin) ?
310 %       3    2  Call: produce_milk(penguin) ?
311 %       3    2  Exit: produce_milk(penguin) ?
312 %       4    2  Call: have_hair(penguin) ?
313 %       4    2  Fail: have_hair(penguin) ?
314 %       1    1  Fail: mammal(penguin) ?
315 % no
```

```
316 %
317 % notrace. Turns off trace
318
319 % Output what ever matches the clauses
320 % warm_blooded(X), produce_milk(X), write(X),nl.
321
322 % ---------- RECURSION ----------
323
324 /*
325 parent(albert, bob).
326 parent(albert, betsy).
327 parent(albert, bill).
328
329 parent(alice, bob).
330 parent(alice, betsy).
331 parent(alice, bill).
332
333 parent(bob, carl).
334 parent(bob, charlie).
335 */
336
337 % Works for exact matches
338 related(X, Y) :- parent(X, Y).
339 % related(albert, bob). = true
340
341 % Cycles through possible results until related returns a true
342 related(X, Y) :-
343   parent(X, Z),
344   related(Z, Y).
345
346 % related(albert,carl). = true
347
348 % 1. parent(albert, Z). = true = Z = bob, betsy, bill
349 % 2. related(Z, carl). = true when Z = bob
350
351 % ---------- MATH ----------
352 % Prolog provides 'is' to evaluate mathematical expressions
353 % X is 2 + 2. = X = 4
354
355 % You can use parenthese
356 % X is 3 + (2 * 10). =  X = 23
357
358 % You can also make comparisons
359 % 50 > 30. = yes
360 % (3*10) >= (50/2). = yes
361 % \+ (3 = 10). = yes (How to check for not equal)
362 % 5+4 =:= 4+5. = yes (Check for equality between expressions)
363 % 5+4 =\= 4+5. = yes (Check for non-equality between expressions)
364 % 5 > 10 ; 10 < 100. (Checks if 1 OR the other is true)
365
366 % X is mod(7,2). = X = 1 (Modulus)
367
368 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),
```

```
422   write('Hi '),
423   write(X).
424
425 % say_hi.
426 % What is your name 'Derek'.
427 % Hi Derek
428
429 fav_char :-
430   write('What is your favorite character? '),
431
432   % Receives a char and saves its ascii value to X
433   get(X),
434   format('The Ascii value ~w is ', [X]),
435
436   % Outputs Ascii value as the char
437   put(X),nl.
438
439 % Write to a file by defining the file, text to write, connection
440 % to the file (Stream)
441 write_to_file(File, Text) :-
442   open(File, write, Stream),
443   write(Stream, Text), nl,
444   close(Stream).
445
446 % Read from a file
447 read_file(File) :-
448         open(File, read, Stream),
449
450         % Get char from the stream
451         get_char(Stream, Char1),
452
453         % Outputs the characters until end_of_file
454         process_stream(Char1, Stream),
455         close(Stream).
456
457 % Continue getting characters until end_of_file
458 % ! or cut is used to end backtracking or this execution
459 process_stream(end_of_file, _) :- !.
460
461 process_stream(Char, Stream) :-
462         write(Char),
463         get_char(Stream, Char2),
464         process_stream(Char2, Stream).
465
466 % ---------- HOW TO LOOP ----------
467
468 % Use recursion to loop
469 count_to_10(10) :- write(10), nl.
470
471 count_to_10(X) :-
472   write(X),nl,
473   Y is X + 1,
474   count_to_10(Y).
```

```
475
476 % Receives Low (lowest value) and High (highest value)
477 count_down(Low, High) :-
478   % Assigns values between Low and High to Y
479   between(Low, High, Y),
480   % Assigns the difference to Z
481   Z is High - Y,
482   write(Z),nl,
483   % Continue looping until Y = 10
484   Y = 10.
485
486 count_up(Low, High) :-
487   between(Low, High, Y),
488   Z is Y + Low,
489   write(Z), nl,
490   Y = 10.
491
492 % Loop until they guess a number
493 % start is a dummy value used to start the looping
494 guess_num :- loop(start).
495
496 % When they guess 15 they execute this message and exit
497 loop(15) :- write('You guessed it!').
498
499 loop(X) :-
500   x \= 15,
501   write('Guess Number '),
502   read(Guess),
503   write(Guess),
504   write(' is not the number'), nl,
505   loop(Guess).
506
507 % guess_num.
508 % Guess Number 12.
509 % 12 is not the number
510 % Guess Number 15.
511 % 15 is not the number
512 % You guessed it!
513
514 % ---------- CHANGING THE DATABASE ----------
515 % Any predicate you plan to motify should be marked as dynamic before
516 % this predicate is used in any way
517 :- dynamic(father/2).
518 :- dynamic(likes/2).
519 :- dynamic(friend/2).
520 :- dynamic(stabs/3).
521
522 father(lord_montague,romeo).
523 father(lord_capulet,juliet).
524
525 likes(mercutio,dancing).
526 likes(benvolio,dancing).
527 likes(romeo,dancing).
```

```
528 likes(romeo,juliet).
529 likes(juliet,romeo).
530 likes(juliet,dancing).
531
532 friend(romeo,mercutio).
533 friend(romeo,benvolio).
534 % friend(X, romeo) :- friend(romeo, X).
535
536 stabs(tybalt,mercutio,sword).
537 stabs(romeo,tybalt,sword).
538
539 % Add new clause to the database at the end of the list for the same
540 % predicate
541 % assertz(friend(benvolio, mercutio)).
542 % friend(benvolio, mercutio). = yes
543
544 % Add clause at the start of the predicate list
545 % asserta(friend(mercutio, benvolio)).
546 % friend(mercutio, benvolio). = yes
547
548 % Delete a clause
549 % retract(likes(mercutio,dancing)).
550 % likes(mercutio,dancing). = no
551
552 % Delete all clauses that match
553 % retractall(father(_,_)).
554 % father(lord_montague,romeo). = no
555
556 % Delete all matching clauses
557 % retractall(likes(_,dancing)).
558 % likes(_,dancing). = no
559
560 % ---------- LISTS ----------
561 % You can store atoms, complex terms, variables, numbers and other
562 % lists in a list
563 % They are used to store data that has an unknown number of elements
564
565 % We can add items to a list with the | (List Constructor)
566 % write([albert|[alice, bob]]), nl.
567
568 % Get the length of a list
569 % length([1,2,3], X).
570
571 % We can divide a list into its head and tail with |
572 % [H|T] = [a,b,c].
573
574 % H = a
575 % T = [b,c]
576
577 % We can get additional values by adding more variables to the left
578 % of |
579
580 %[X1, X2, X3, X4|T] = [a,b,c,d].
```

```
581
582 % We can use the anonymous variable _ when we need to reference a
583 % variable, but we don't want its value
584 % Let's get the second value in the list
585 % [_, X2, _, _|T] = [a,b,c,d].
586
587 % We can use | to access values of lists in lists
588 % [_, _, [X|Y], _, Z|T] = [a, b, [c, d, e], f, g, h].
589
590 % Find out if a value is in a list with member
591 % List1 = [a,b,c].
592 % member(a, List1). = yes
593
594 % We could also get all members of a list with a variable
595 % member(X, [a, b, c, d]).
596
597 % Reverse a list
598 % reverse([1,2,3,4,5], X).
599
600 % Concatenate 2 lists
601 % append([1,2,3], [4,5,6], X).
602
603 % Write items in list on separate line
604 write_list([]).
605
606 write_list([Head|Tail]) :-
607   write(Head), nl,
608   write_list(Tail).
609 % write_list([1,2,3,4,5]). = Outputs the list
610
611 % ---------- STRINGS ----------
612 % Convert a string into an Ascii character list
613 % name('A random string', X).
614
615 % Convert a Ascii character list into a string
616 % name(X, [65,32,114,97,110,100,111,109,32,115,116,114,105,110,103]).
617
618 % Append can join strings
619 join_str(Str1, Str2, Str3) :-
620
621   % Convert strings into lists
622   name(Str1, StrList1),
623   name(Str2, StrList2),
624
625   % Combine string lists into new string list
626   append(StrList1, StrList2, StrList3),
627
628   % Convert list into a string
629   name(Str3, StrList3).
630
631 % join_str('Another ', 'Random String', X). = X = 'Another Random String'
632
633 % get the 1st char from a string
```

```
634  /*
635  name('Derek', List),
636  nth0(0, List, FChar),
637  put(FChar).
638  */
639
640  % Get length of the string
641  atom_length('Derek',X).
```

4 Example

4.1 epidemic.cpl

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

```
 1  /*
 2  Model of the development of an epidemic or a pandemic.
 3  From
 4  E. Bellodi and F. Riguzzi. Expectation Maximization over binary decision
 5  diagrams for probabilistic logic programs. Intelligent Data Analysis,
 6  17(2):343-363, 2013.
 7  */
 8
 9
10  epidemic : 0.6; pandemic : 0.3 :- flu(_), cold.
11  % if somebody has the flu and the climate is cold, there is the possibility
12  % that an epidemic arises with probability 0.6 and the possibility that a
13  % pandemic arises with probability 0.3
14
15  cold : 0.7.
16  % it is cold with probability 0.7
17
18  flu(david).
19  flu(robert).
20  % david and robert have the flu for sure
21
22  /** <examples>
23
24  ?- epidemic.  % what is the probability that an epidemic arises?
25  % expected result 0.588
26  ?- pandemic.  % what is the probability that a pandemic arises?
27  % expected result 0.357
28
29  */
```

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.

```

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