CS 131 - Week 6

TA : Tanmay Sardesai

How to find these slides

Piazza -> CS 131 -> Resources -> Discussion 1B

Announcements

- Email tanmays@cs.ucla.edu
- Office Hours Thursday 1:30 pm - 3:30 pm. Bolter Hall 3256S-A
- HW4 will be due Feb 22nd (next Friday) at 11:55 pm

Topics covered today

- Homework 4 Motivation
- Prolog Introduction
- Homework 4 Details
- Hands on

Homework 4 Motivation

- https://web.cs.ucla.edu/classes/winter19/cs131/hw/hw4.html
- https://www.chiark.greenend.org.uk/~sgtatham/puzzles/js/towers.html#5:2/3/2/1/4/3/1/3/3/2/4/1/2/5/2/2/4/2/1/2

Prolog Introduction

Running Prolog Code

- We will be using gprolog to check your homework code
  - http://www.gprolog.org/
  - http://www.gprolog.org/#download
  - http://www.gprolog.org/manual/gprolog.html
- There are many implementation of prolog but we won’t support them for this assignment
  - http://www.swi-prolog.org/
  - http://www.dcc.fc.up.pt/~vsc/yap/

Prolog Introduction

- Load a file - \`**consult(‘file.pl’).**\` or \`**[‘file.pl’].**\`

- How to exit interpreter: Control + D

What is a Prolog Program?

- Technically you don’t create a prolog program.

- You create a prolog database.

- In a database you have 2 types of clauses.
    - Facts
    - Rules

Sooooo what really is prolog???

- Prolog is a logic programming language
- Basic idea
    - Declare a set of facts
        - Single piece of information
        - Sky is blue
    - Declare a set of rules
        - Ways to generate new information based facts, other rules or even themselves
        - A is grandparent of Z if A if parent of B and B is parent of Z
    - Run a query to ask if things are true and get solutions based on rules

Prolog Facts

- Prolog constants
  - Uninterpreted constants
  - *alice* is a constant
  - Always use lowercase
- Predicate
  - A function that returns boolean.
  - *person* is predicate
- Variable
  - Always UPPERCASE

```
person(alice).

person(bob).

blue(sky).

mammal(rabbit).
```

Prolog Rules

- Rules describe a general relationship between facts
- Syntax
  - head :- body
  - head and body are both clauses that usually use variables.

```
grandparent(X,Z) :- parent(X,Y) , parent(Y,Z).

ancestor(X,Z) :- parent(Z,Y) , ancestor(X,Y).
```

Prolog Rules

- Rules describe a general relationship between facts
- Syntax
  - head :- body
  - head and body are both clauses that usually use variables

```
grandparent(X,Z) :- parent(X,Y) , parent(Y,Z).

ancestor(X,Z) :- parent(Z,Y) , ancestor(X,Y).
```

- We don’t explain how to use the rule, we just give prolog a database and it determines how and when to use the rules

Prolog Query

- Used to ask a question
- Query is like a rule without the body
- Prolog takes the head and tries to determine if true or false
- If we pass variables then prolog tries to determine all the variables

```
parent(alice,bob).

ancestor(bob,doug).

parent(bob, X).
```

What happens after running a Query?

- Three options
  - ;   This will compute next solution
  - a   This will compute all the remaining solutions
  - <return>   This will stop the execution of the query

Minimum “function” in prolog

```
minimum(X,Y,X) :- X<Y.
minimum(X,Y,Y) :- X>=Y.
```

Examples - family.pl

Prolog Visualizer

http://www.cdglabs.org/prolog/#/

https://www.slideshare.net/ZhixuanLai/prolog-visualizer

Tracing the logic (debugging)

- Sometimes your programs may return wrong answer because of wrong, incorrect or incomplete rules
- Use trace to figure out why

Arithmetic in Prolog

$x < y$	X < Y.
$x \leq y$	X =< Y.
$x > y$	X > Y.
$x \geq y$	X >= Y.
$x == y$	X =:= Y
$x \neq y$	X =\= Y

‘is’

```
?- 8 is 6 + 2.
yes

?- 2 is 4 - 3.
no

?- 1 is sin(pi/2).
no

?- 1.0 is sin(pi/2).
yes
```

```
?- X is 6 + 2.
X = 8
yes

?- X is sin(pi/4)
X = 0.70710678118654746
yes
```

Defining Predicate with is

```
subtract_4_and_double(X,Y) :- Y is (X-4) * 2.
```

Example:

- subtract_4_and_double(5,1). -> no
- subtract_4_and_double(5,Y). -> Y=2 yes
- subtract_4_and_double(X,1). -> Error… why?

‘is’ Usage

- Arithmetic statements must be on the right with variables that have an assigned value.
  - X is 6 + 2 allowed
  - 6 + 2 is X not allowed
  - X is Y + 2 will only work is Y has some value assigned to it
- http://www.gprolog.org/manual/gprolog.html#sec98
  - Documentation describe how to use “is”

Equality operator (==) vs Equality Unification (=)

- The goal term1==term2 succeeds only if term1 is identical to term2.
  - likes(X,prolog)==likes(john,Y).   -> no
  - likes(X,prolog)==likes(X,prolog).   -> yes
- The goal term1=term2 succeeds only if term1 and term2 unify, i.e. there is some way of binding variables to values which would make the terms identical.
  - likes(X,prolog)=likes(john,Y).   ->X=john Y=prolog yes

More example on == and =

- likes(X,prolog)==likes(Y,prolog). -> no
- likes(X,prolog)=likes(Y,prolog). -> Y=X yes

- likes(X,prolog)==likes(Y,Z). -> no
- likes(X,prolog)=likes(Y,Z). -> Y=X Z=prolog yes

- likes(X,prolog)==likes(Y,java). -> no
- likes(X,prolog)=likes(Y,java). -> no

List in Prolog

- Pattern Matching
  - ?- [a, b] = [a, X]. -> X = b yes
  - ?- [a,b] = X. -> X = [a,b] yes
  - ?- [a,b] = [X]. -> no
- Head and Tail
  - ?- [a,b,c,d] = [H|T]. -> H = a T = [b,c,d] yes
  - ?- [a] = [H|T]. -> H = a T = []

List processing

- append(List1, List2, List12)
- member(Term, List1)
- reverse(List1, List2)
- prefix(Prefix, List)
- suffix(Prefix, List)
- last(Term,List)
- length(List,Integer)
- More here http://www.gprolog.org/manual/html_node/gprolog044.html

Functions using List

- first(List,Term)

- last_in_array(List,Term)

- compress(List1,List2)

Functions using List

-
  ```
  first([H|T],H).
  ```

-
  ```
  last_in_array([A],A).
  last_in_array([A|L],E) :- last_in_array(L,E).
  ```

-
  ```
  compress([],[])
  compress([X],[X])
  compress([X,X|Xs],Zs) :- compress([X|Xs],Zs).
  compress([X,Y|Ys],[X|Zs]) :- compress([Y|Ys], Zs), X =\= Y.
  ```

Generating a list with constraints

Generate a list of length N where each element is a unique integer between 1..N

```
all_unique([]).
all_unique([H|T]) :- member(H, T), !, fail.
all_unique([H|T]) :- all_unique(T).

elements_between([], _, _).
elements_between([H|T], Min, Max) :-    between(Min,Max,H),
                                        elements_between(T, Min, Max).

unique_list(List, N) :-   length(List, N),
                          elements_between(List, 1, N),
                          all_unique(List).
```

Generating a List with constraints

This is inefficient as we have N^N lists to iterate.

Same problem using Finite Domain solver

- Finds assignments to variables that fulfill constraints
- Variable values are limited to a finite domain (e.g. integers between 0 and 10)
- Less code, optimized solution

```
unique_list2(List, N) :- length(List,N),                    % constraint on length
                         fd_domain(List, 1, N),             % constraint on value
                         fd_all_different(List),            % constraint on uniqueness
                         fd_labeling(List).     % compute List based on constraints
```

Sudoku using FD

Solve a 4 x 4 sudoku using FD.

Predicate definition -> sudoku(L) :- ....

Use:

fd_domain(List,Min,Max)

fd_all_different(List)

4	2	1	3
1	3	4	2
2	4	3	1
3	1	2	4

```
sudoku4_fd(L):-  L = [X11,X12,X13,X14,X21,X22,X23,X24,X31,X32,X33,X34,X41,X42,X43,X44],
                 fd_domain(L, 1, 4),
                 fd_all_different([X11,X12,X13,X14]) , fd_all_different([X21,X22,X23,X24]),
                 fd_all_different([X31,X32,X33,X34]) , fd_all_different([X41,X42,X43,X44]),
                 fd_all_different([X11,X21,X31,X41]) , fd_all_different([X14,X24,X34,X44]),
                 fd_all_different([X12,X22,X32,X42]) , fd_all_different([X13,X23,X33,X43]),
                 fd_all_different([X11,X12,X21,X22]) , fd_all_different([X13,X14,X23,X24]),
                 fd_all_different([X31,X32,X41,X42]) , fd_all_different([X33,X34,X43,X44]),
                 fd_labeling(L).
```

The above code defines the List L of all the elements in the sudoku. We restrict the values in L between 1 and 4. For all possible row, column and box we make sure everything is unique.

```
?- sudoku4_fd([1,2,3,4,X21,X22,X23,X24,X31,X32,X33,X34,X41,X42,X43,X44]).
```

If we run the sudoku function like above then the program returns -->

```
X21 = 3, X22 = 4, X23 = 1, X24 = 2, X31 = 2, X32 = 1, X33 = 4, X34 = 3, X41 = 4, X42 = 3, X43 = 2, X44 = 1
```

Puzzle - wolf, goat, cabbage

http://jeux.lulu.pagesperso-orange.fr/html/anglais/loupChe/loupChe1.htm

Constraints:
boat fits you plus one
can't leave the wolf with the goat
can't leave the goat with the cabbage

What's a state of the world:
a list of four elements
start: [west,west,west,west]
in general: [Person, Wolf, Goat, Cabbage]

What are the actions in the world?
wolf, goat, cabbage, nothing

Link to sample code (test code, some mistakes)

https://drive.google.com/drive/folders/14UA7vdrwJE1YENPuUPCSqV40i8S9BrK0?usp=sharing

Open using UCLA email

Prolog References

- https://www.cslab.pepperdine.edu/warford/cosc450/cosc-450-Setup-for-Prolog.pdf
- http://www.cs.utexas.edu/~cannata/cs345/Class%20Notes/12%20prolog_intro.pdf
- http://mgencer.com/files/PrologTutorial.html
- http://www.gprolog.org/manual/gprolog.html
- http://www.cs.toronto.edu/~sheila/384/w11/Prolog/prolog-tutorial-part1.pdf
- http://www.cs.toronto.edu/~sheila/384/w11/Prolog/prolog-tutorial-part2.pdf
- http://www.cs.toronto.edu/~sheila/384/w11/simple-prolog-examples.html