

## Concepts of Programming Languages, CSCI 305, Fall 2021

### Lab 7, Prolog, Nov. 12

```
/******  
* ancestry.pl  
* father(x,y) means that x is the father of y.  
* mother(x,y) means that x is the mother of y.  
*****/  
father(ed, celia).  
father(ed, david).  
father(stephen, ed).  
father(jeff, michele).  
father(wally, karen).  
mother(peggy, celia).  
mother(peggy, david).  
mother(peggy, jim).  
mother(peggy, barb).  
mother(peggy, mike).  
mother(karen, jeff).  
mother(celeia, michele).  
mother(clara, karen).
```

Using this database of facts (later you will add rules to it), write Prolog queries to return the following:

1. All mothers in the database. (Mother names will be repeated. That is ok.)

`mother(X,_).`

2. All children in the database that have both parents.

`mother(_,X), father(_,X).`

3. All children that have at least one parent.

`mother(_,X); father(_,X).`

4. All children that only have a mother.

`mother(_,X),\+ (father(_,X)).`

5. All children that only have one parent.

`(mother(_,X), \+ (father(_,X))); (father(_,X), \+ (mother(_,X))).`

Within each set of parenthesis, put the negated proposition last. Negation is “not provable”, so the engine needs to try to prove the clause and if it can’t, return true. Trying to prove the clause can take a long time (maybe infinite time). Also, once the engine tries to prove the proposition and fails, it doesn’t seem to start back at the top for the second conjunction.

6. Add a rule to the database which defines `parent(X,Y)` to mean that X is a parent of Y and display all of the parent, child combinations.

Add the rule:

```
parent(X,Y):- mother(X,Y);father(X,Y).
```

At the prompt:

```
parent(X,Y).
```

7. Add a rule to the database which defines `ancestor(X,Y)` to mean that X is an ancestor Y and display all of the ancestors of michele.

Add the rules:

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

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

At the prompt:

```
ancestor(X, michele).
```

8. Sections 12.2.5 & 11.2.6, pages 600-609, present a Prolog program which plays Tic-Tac-Toe. In that program the following built-in functions are used:
- asserta – place a fact temporarily into the db
  - repeat – continues searching (provided ‘;’ is entered)
  - retractall – to remove facts which you have asserted (using asserta).

Also the following Prolog construct is used:

Cut (!) – a zero-argument predicate which always succeeds and causes searching to stop.

Walk through the following code so that you understand what it does.

```

/*****
*   Tic-Tac-Toe
*
*   Chapter 12, Logic Languages
*   Section 12.2.5, page 609, modified so that either the player
*   or computer can go first.
*
*   To execute:
*   At ?- prompt enter consult(ticTacToe).
*   At ?- prompt enter play.
*****/

% Specify all win situations.
ordered_line(1, 2, 3).
ordered_line(4, 5, 6).
ordered_line(7, 8, 9).
ordered_line(1, 4, 7).
ordered_line(2, 5, 8).
ordered_line(3, 6, 9).
ordered_line(1, 5, 9).
ordered_line(3, 5, 7).

% Allow for wins listed in different orders.
line(A, B, C) :- ordered_line(A, B, C).
line(A, B, C) :- ordered_line(A, C, B).
line(A, B, C) :- ordered_line(B, A, C).
line(A, B, C) :- ordered_line(B, C, A).
line(A, B, C) :- ordered_line(C, A, B).
line(A, B, C) :- ordered_line(C, B, A).

% Square contains X or O
full(A) :- x(A).
full(A) :- o(A).
empty(A) :- \+(full(A)).

same(A,A).
different(A,B) :- \+(same(A,B)).

move(A) :- good(A), empty(A), !.

```

```

% Strategy
good(A) :- win(A).
good(A) :- block_win(A).
good(A) :- split(A).
good(A) :- strong_build(A).
good(A) :- weak_build(A).
good(5).
good(1).
good(3).
good(7).
good(9).
good(2).
good(4).
good(6).
good(8).

win(A) :- x(B), x(C), line(A,B,C).

block_win(A) :- o(B), o(C), line(A,B,C).

split(A) :- x(B), x(C), different(B, C),
           line(A,B,D), line(A,C,E), empty(D), empty(E).
strong_build(A) :- x(B), line(A,B,C), empty(C), \+(risky(C)).
weak_build(A) :- x(B), line(A,B,C), empty(C), \+(double_risky(C)).
risky(C) :- o(D), line(C,D,E), empty(E).
double_risky(C) :- o(D), o(E), different(D, E),
                  line(C,D,F), line(C,E,G), empty(F), empty(G).

all_full :- full(1), full(2), full(3), full(4),
            full(5), full(6), full(7), full(8), full(9).

done :- line(A,B,C), x(A), x(B), x(C), write('I won'), nl.

done :- all_full, write('Draw'), nl.

% User's move
getmove :- repeat, write('Please enter a move: '), read(X), empty(X),
           asserta(o(X)).
makemove :- move(X), !, asserta(x(X)).
makemove :- all_full.

% Computer's move
respond :- line(A,B,C), o(A), o(B), o(C),
           printboard, write('You won. '), nl.           %Shouldn't ever happen!
respond :- makemove, printboard, done.

% Print the board
printsquare(N) :- o(N), write(' o ').
printsquare(N) :- x(N), write(' x ').
printsquare(N) :- empty(N), write(' ').
printboard :- write('Board:'), nl,
              printsquare(1), printsquare(2), printsquare(3), nl,
              printsquare(4), printsquare(5), printsquare(6), nl,
              printsquare(7), printsquare(8), printsquare(9), nl.

% Clear everything to start a new game.
clear :- retractall(x(_)), retractall(o(_)),

```

```

write('Board:'), nl,
write(' 1 '), write(' 2 '),write(' 3 '), nl,
write(' 4 '), write(' 5 '),write(' 6 '), nl,
write(' 7 '), write(' 8 '),write(' 9 '), nl,
write('You will be o, computer will be x. '), nl,
write('Enter moves by giving a number (1-9) followed by a period. '),nl,
nl.

% main goal. Play the game.
play :- clear,
       write('Want to go first (y for yes, anything else for no)? (Remember the
period.) '),
       read(X), nl, first(X).

first(y) :- taketurns.
first(_) :- makemove, printboard, taketurns.

taketurns :- repeat, getmove, respond.

```

9. Get a copy of the TicTacToe program, bring it into the environment, and run it by entering:

?- play.

10. Knowing the strategy which the computer will use, tell the values to be entered to win the game.

Choose to go first and mark 8, 3, 9 and 7.

11. Add the **split** rule to the computer's strategy.

```

split(A) :- x(B), x(C), different(B, C),
           line(A, B, D), line(A, C, E),
           empty(D), empty(E).

```

A space meets the **split** rule if it is in two different lines, where the computer has a cell in both lines and the third cell in the two lines are empty.

12. Add other rules to improve the strategy of the code.