Madhuri Pyreddy
31 October 2019

CSCI 117 Lab 10

Lab 10 Part 1-Grammars:

```
determiner(S,P1,P2,[every|X],X,all(S,imply(P1,P2))).
determiner(S,P1,P2,[a|X],X,exists(S,and(P1,P2))).

noun(N,[man|X],X, man(N)).
noun(N,[woman|X],X,woman(N)).
noun(N,[child|X],X,child(N)).

name([john|X],X,john).
name([mary|X],X,mary).
name([jackie|X],X,jackie).
name([quentin|X],X,quentin).

transverb(S,O,[loves|X],X,loves(S,O)).
transverb(S,O,[knows|X],X,knows(S,O)).

intransverb(S,[lives|X],X,lives(S)).
intransverb(S,[runs|X],X,runs(S)).

sentence(X0,X,P):-
nounphrase(N,P1,X0,X1,P),
verbphrase(N,X1,X,P1).
sentence([john,loves,jackie],[],O).
sentence([quentin,knows,mary],[],O).
sentence([a,woman,lives],[],O).
sentence([a,man,runs],[],O).
sentence([a,child,runs],[],O).
sentence([a,child,who,loves,john,lives],[],O).
sentence([every, woman, who, john, loves, lives],[],O).
sentence([every, woman, who, loves, quentin, lives],[],O).
sentence([every, woman, who, quentin, loves, lives],[],O).
sentence([every, woman, who, knows, a, man, who, loves, mary, lives],[],O).
sentence([every, man, who, loves, a, woman, who, knows, quentin, runs], [], O).
sentence([every, child, who, knows, a, woman, who, knows, jackie, lives], [], O).
sentence([every, woman, who, knows, a, child, who, loves, john, runs], [], O).
```

```
nounphrase(N,P1,X0,X,P):-
determiner(N,P2,P1,X0,X1,P),
noun(N,X1,X2,P3),
relclause(N,P3,X2,X,P2).
nounphrase(N,P1,X0,X,P1):- name(X0,X,N).


verbphrase(S,X0,X,P):-
        transverb(S,O,X0,X1,P1),
        nounphrase(O,P1,X1,X,P).
verbphrase(S,X0,X,P):-
  nounphrase(O,_,X0,X1,_),
  transverb(O,S,X1,X,P).
verbphrase(S,X0,X,P):-
intransverb(S,X0,X,P).
relclause(S,P1,[who|X1],X,and(P1,P2)):-
verbphrase(S,X1,X,P2).


relclause(_,P1,X,X,P1).
```

Lab 10 Part 2-Puzzles:

1. /* There are 5 ships in a port.

   1.The Greek ship leaves at six and carries coffee.
   2.The ship in the middle has a black chimney.
   3.The English ship leaves at nine.
   4.The French ship with a blue chimney is to the left of a ship that carries coffee.
   5.To the right of the ship carrying cocoa is a ship going to Marseille.
   6.The Brazilian ship is heading for Manila.
   7.Next to the ship carrying rice is a ship with a green chimney.
   8.A ship going to Genoa leaves at five.
   9.The Spanish ship leaves at seven and is to the right of the ship going to Marseille.
   10.The ship with a red chimney goes to Hamburg.
   11.Next to the ship leaving at seven is a ship with a white chimney.
   12.The ship on the border carries corn.
   13.The ship with a black chimney leaves at eight.
   14.The ship carrying corn is anchored next to the ship carrying rice.
   15.The ship to Hamburg leaves at six.
   Which ship goes to Port Said? Which ship carries tea?

   Who owns the zebra and who drinks water?
   */

```
% Render the ship term as a nice table.
:- use_rendering(table,
             [header(s('Ship', 'Leaves at', 'Carries', 'Chimney', 'Goes to'))]).

goes_PortSaid(Goes) :-
      ships(S),
      member(s(Goes,_,_,_,portSaid), S).

carries_tea(Carries) :-
      ships(S),
      member(s(Carries,_,tea,_,_), S).

  ships(S) :-
  length(S,5),
  member(s(greek,6,coffee,_,_),S),    %1
  S = [_,_,s(_,_,_,black,_),_,_],     %2
  member(s(english,9,_,_,_),S),                      %3
 left(s(french,_,_,blue,_),s(_,_,coffee,_,_),S),     %4
 next(s(_,_,_,_,marseille),s(_,_,cocoa,_,_),S),   %5
 member(s(brazilian,_,_,_,manila),S),                  %6
 next(s(_,_,_,green,_), s(_,_,rice,_,_),S),        %7
  member(s(_,5,_,_,genoa),S),                         %8
 next(s(spanish,7,_,_,_),s(_,_,_,_,marseille),S),  %9
 member(s(_,_,_,red,hamburg),S),                      %10
 next(s(_,_,_,white,_),s(_,7,_,_,_),S),             %11
 border(s(_,_,corn,_,_),S),                         %12
 member(s(_,8,_,black,_),S),                       %13
 next(s(_,_,corn,_,_), s(_,_,rice,_,_),S),        %14
 member(s(_,6,_,_,hamburg),S),                        %15




  member(s(_,_,_,_,portSaid),S),
  member(s(_,_,tea,_,_),S).

next(A, B, Ls) :- append(_, [A,B|_], Ls).
next(A, B, Ls) :- append(_, [B,A|_], Ls).
left(A, B, Ls) :- append(_, [A,B|_], Ls).
border(A, Ls) :- append(_,[A], Ls).
border(A, Ls) :- append([A|_],_,Ls).

/** <examples>
```

```
?- goes_PortSaid(Goes).

?- carries_tea(Carries).

?- port(Ports).

*/
```

Spanish ship goes to Port Said. The French carry the tea.

2.

```
% N is size of rows/columns, T is tour taken (as list of positions m(P1,P2))
knightTour(N,M,T) :- N2 is N*M, % N2 is the number of positions on the board (N^2)
  kT(N,M,N2,[m(0,0)],T). % [m(0,0)] is starting position of the knight

% kT(N,N2,[m(P1,P2)|Pt],T)
% N is the row/column size
% N2 is the number of positions on the board
% [m(P1,P2)|Pt] is the accumulator for the tour
%   m(P1,P2) is the current position of the knight
%   Pt (partial tour) is the list of previous positions of the knight
% T is the full tour

% Finished: Length of tour is equal to size of board
% return accumulator as tour
kT(N,M,N2,T,T) :- length(T,N2).

kT(N,M,N2,[m(P1,P2)|Pt],T) :-
    moves(m(P1,P2),m(D1,D2)), % get next position from current position
    D1>=0,D2>=0,D1<N,D2<M,    % verify next position is within board dimensions
    \+ member(m(D1,D2),Pt),   % next position has not already been covered in tour
    kT(N,M,N2,[m(D1,D2),m(P1,P2)|Pt],T). % append next position to front of accumulator

% 8 possible moves for a knight
% P1,P2 is knight's position, D1,D2 is knight's destination after one move
% Iterated list solution
moves(m(X,Y), m(U,V)) :-
  member(m(A,B), [m(1,2),m(1,-2),m(-1,2),m(-1,-2),m(2,1),m(2,-1),m(-2,1),m(-2,-1)]),
  U is X + A,
  V is Y + B.

% "Number Hacking" solution
```

```
offsets(N1,N2) :-
    member(N,[1,2]),member(B1,[1,-1]),member(B2,[1,-1]),
    N1 is N*B1, N2 is (3-N)*B2.
% moves(m(P1,P2),m(D1,D2)) :-
%    offsets(A,B), D1 is P1+A, D2 is P2+B.
```

Testing:

```
?-knightTour(3,4,T)
T = [m(2, 3), m(1, 1), m(0, 3), m(2, 2), m(1, 0), m(0, 2), m(2, 1), m(1, 3), m(0, 1), m(2, 0),
m(1, 2), m(0, 0)]
```