# CSCI 235, Programming Languages, Prolog
# Exercise 2

Deadline: 28.10.2018 at 23.59

Solutions should be entered into Moodle before the deadline.

1. Write a predicate `cartesian( X, Y, Z )` that is true if `X,Y,Z` are lists, and `Z` is the Cartesian product of `X` and `Y`. The predicate must be able to compute `Z` when `X` and `Y` are given:

   ```
   cartesian( [], [1], Z )  ==> Z = [].
   cartesian( [a], [], Z )  ==> Z = [] .

   cartesian( [a,b], [1,2], Z )
      ==> Z = [pair(a, 1), pair(a, 2), pair(b, 1), pair(b, 2)].
   ```

2. Since Prolog is untyped, it is possible to mix various kinds of elements in a list, also elements of different levels of nesting, like for example `[ 1, a, [ b ] , [ c, [ d ] ] ]`. Write a predicate `deepmember( X, Y )` that succeeds if `Y` is a possibly nested list, that contains `X`.

   ```
   deepmember( 1, [ [ 2, 1 ], 3 ] ).
      ==> true
   deepmember( a, [ [ b, c ], d ] ).
      ==> false
   deepmember( X, [ [ 2, 1 ], 3 ] ).
      ==> enumerates all sublists.
   ```

3. Write a predicate `notcontains( X, L )` that succeeds if `L` does not contain `X`. Inequality is expressed by `\=`.

4. Write a predicate `addunique( X, L1, L2 )` that that inserts `X` to `L1` if it does not occur, and otherwise does nothing.

   This is actually a very bad description of a predicate. In Prolog, we write *predicates*, not *procedures*. Hence, the way that we describe a Prolog predicate, must reflect the fact that it is a predicate.

   Second attempt: Write a predicate `setinsertion( X, S1, S2 )` that succeeds if

(a) either X occurs in S1 and S1 equals S2, or

(b) X does not occur in S1, and S2 contains exactly the same elements as S1, but with X added.

Use \= for non-equality. You can write `setinsertion` directly (3 clauses), or use `notcontains` (two clauses). Your predicate must work when X and S1 are instantiated and S2 is not instantiated. It is sufficient when your predicate generates one solution for S2.

5. Since Prolog is useful for search, we should have a nice task involving search. Let us try to find Hamiltonian circuits in a graph. A graph can be represented by the set of its edges, collected in a list. For example:

```
graph1( [ [ 1, 2 ], [ 1, 3 ], [ 2, 3 ], [ 3, 4 ], [ 4, 1 ] ] ).
graph2( [ [ 1, 2 ], [ 2, 3 ], [ 2, 4 ], [ 3, 4 ],
          [ 4, 3 ], [ 3, 1 ], [ 4, 1 ] ] ).
graph3( [ [ 1, 2 ], [ 1, 3 ], [ 2, 3 ], [ 3, 2 ],
          [ 3, 4 ], [ 2, 4 ], [ 4, 6 ], [ 4, 5 ],
          [ 5, 6 ], [ 6, 5 ], [ 6, 7 ], [ 5, 7 ],
          [ 7, 1 ] ] ).
```

(a) Write a predicate `allvertices( G, L )` that succeeds if L contains all vertices of G. Use `setinsertion`. For example

```
?- graph2(G), allvertices(G,L).
      G = [[1, 2], [2, 3], [2, 4], [3, 4], [4, 3], [3, 1], [4, 1]],
      L = [4, 1, 3, 2]
```

(b) Write a predicate `connected(V0,V1,G)` that succeeds if G contains the edge $(V_0, V_1)$. The predicate must be able to enumerate edges. For example

```
?- graph1(G), connected( V0,V1,G).
      G = [[1, 2], [1, 3], [2, 3], [3, 4], [4, 1]],
      V0 = 1,
      V1 = 2 ;
      G = [[1, 2], [1, 3], [2, 3], [3, 4], [4, 1]],
      V0 = 1,
      V1 = 3 ;    /* etc. etc. */
```

(c) Next we can write a predicate `path( G, Vbegin, N, Forbidden, Path, Vend )`. This predicate succeeds if Path is a list of length N, starting with Vbegin and ending with Vend, such no element of Path (with the exception of the first element), occurs in Forbidden, no element occurs twice in Path, and each element in Path is connected to the next element by an edge of G.

For example:

```
?- graph1( G), path( G, 1, 1, [1], P, Last ).
      G = [[1, 2], [1, 3], [2, 3], [3, 4], [4, 1]],
      P = [1],
      Last = 1 ;

?- graph2( G), path( G, 3, 2, [3], P, Last ).
      G = [[1, 2], [2, 3], [2, 4], [3, 4], [4, 3], [3, 1], [4, 1]],
      P = [3, 4],
      Last = 4 ;
      G = [[1, 2], [2, 3], [2, 4], [3, 4], [4, 3], [3, 1], [4, 1]],
      P = [3, 1],
      Last = 1 ;
```

If you are writing your predicate, and you want to see what is going on, you can use `write(X)`. You can use `nl` to print a newline. You can also write things like `write( 'value of N = ' ), write(N), nl,` The solution is not long. My solutions contains two clauses, and 6 lines of code. Use `connected` to find nodes `Next` that are connected to `Vbegin`, check that `Next` is not forbidden, and after that, recursively find the rest of the path, by starting in `Next`, with `Next` added to `Forbidden`.

(d) If everything went well in the previous tasks, you can now add the predicate

```
hamiltoniancircuit( G, C ) :-
allvertices( G, Vert ),
Vert = [ V0 | _ ],
length( Vert, N ),
path( G, V0, N, [ V0 ], C, LastV ),
connected( LastV, V0, G ).
```

Now you can type

```
?- graph2( G), hamiltoniancircuit(G,C).
      G = [[1, 2], [2, 3], [2, 4], [3, 4], [4, 3], [3, 1], [4, 1]],
      C = [4, 3, 1, 2] ;
      G = [[1, 2], [2, 3], [2, 4], [3, 4], [4, 3], [3, 1], [4, 1]],
      C = [4, 1, 2, 3] ;
```

at the command line.