COMP2411 Tutorial 13

1. Consider the following program:

nationality(X, Y) :- naturalized(X, Y), not(allows_dual_citizenship(Y)), !.
nationality(X, Y) :- naturalized(X, Y).
nationality(X, Y) :- parents(X, XM, XF), same_nationality(XM, XF, Z), !, same_nationality(XM, XF, Y).
nationality(X, Y) :- born(X, Y), country(Y).

same_nationality(X, Y, Z) :- nationality(X, Z), nationality(Y, Z).

stateless(X) :- not(nationality(X, Y)).

country(australia).
country(usa).
country(france).
country(italy).
country(israel).
country(japan).

allows_dual_nationality(usa).
allows_dual_nationality(australia).

parents(joe, sally, fred).
parents(yoko, kimiko, phillipe).
parents(yoram, yael, stephan).
parents(monique, sophie, shigeki)

born(joe, usa).
born(sally, australia).
born(fred, usa).

born(yoko, japan).
born(kimiko, japan).
born(phillipe, france).

born(yoram, qantas004_over_the_pacific).
born(yael, israel).
born(stephan, france).

born(monique, japan).
born(sophie, france).
born(shigeki, japan).

naturalized(yael, usa).
naturalized(sally, usa).
naturalized(kimiko, france).
What is the result of the query `nationality(x,Y)` for each person `x`? Is `stateless(x)` true for any person `x`? What is the result of the query `stateless(X)`?

2. Write a logic program for the predicate `strict_subset(X,Y)` that determines whether the set of elements of a given list `X` is a strict subset of the the set of elements of a given list `Y`. For example, your program should run as follows:

```prolog
: strict_subset([2,3,2,7],[7,3,2,3])?
** no
```

```prolog
: strict_subset([2,7],[7,2,3])?
** yes
```

Can your solution be used to generate all the strict subsets `X` of a given list `Y`? (You may assume that the list `Y` has no repeated elements for this part, and that a list `X` returned as an answer should also be repeat free.) Can you find a solution that does allow this? Can you find one that generates each strict subset exactly once?

3. Write a logic program for the predicate `path(Graph,From,To,Length)` such that when `Graph` is a list of terms of the form `edge(X,Y)` that represents a graph, `From` and `To` are vertices of this graph, and `Length` is a number, determines whether there exists a path from `From` to `To` of length at most `Length`. Using this predicate, write a logic program for the predicate `distance(Graph,From,To,Dist)` that computes the length `Dist` of the shortest path from `From` to `To` in `Graph`. If there is no path, your program should return `Dist = infinity`. (Do not assume that the graph is acyclic.) For example:

```prolog
: path([edge(a,b),edge(b,c)],a,c,3)?
** yes
```

```prolog
: distance([edge(a,b),edge(b,c)],a,c,X)?
X = 2
```

```prolog
: distance([edge(a,b),edge(b,c)],c,a,X)?
X = infinity
```

4. Huth and Ryan, p. 128, problems 1(a) 1(b), 3, 4(c), 7 (a) 7(b), 12.

5. The final Lewis Carrol problem in lecture 24.