Lab 10: Logic Programming and uC

Learning objectives:

- Gain experience solving problems in Prolog
- Gain experience reading the uC specification

This labs aims to prepare you for homework 3, which covers logic programming in Prolog. It also will help you understand how uC code is structured, which will be useful for projects 4 and 5.

Use the following commands to download and unpack the distribution code:

```
$ wget https://eecs390.github.io/lab/lab10/starter-files.tar.gz
$ tar xzf starter-files.tar.gz
```

1. *Logic puzzle.* Write a Prolog program that finds a solution to this logic puzzle:

   There are four different fathers: Smith, Baker, Carpenter, and Tailor. Each of them also has a son: Smithson, Bakerson, Carpenterson, and Tailorson.

   Assume you know the following:

   1. Each person works as either a smith, baker, carpenter, or tailor.
   2. No two of the parents share the same profession.
   3. No two of the sons share the same profession.
   4. No individual has a name that reflects their profession (e.g., Smith is not a smith, Bakerson is not a baker).
   5. No son has the same profession as his father.
   6. Baker has the same profession as Carpenter's son.
   7. Carpenter has the same profession as Tailor's son.
   8. Smith's son is a baker.

   What is each person's profession?

   Complete the following Prolog program to solve this puzzle:

```
% professions(Professions).
%
% True if Professions is the list [smith, baker, carpenter, tailor].
professions([smith, baker, carpenter, tailor]).

% solve_puzzle(Smith, Baker, Carpenter, Tailor,
%              Smithson, Bakerson, Carpenterson, Tailorson).
%
% Solves the puzzle described above.
%
% All parameters are output parameters.
solve_puzzle(Smith, Baker, Carpenter, Tailor,
             Smithson, Bakerson, Carpenterson, Tailorson) :-
    Fathers = [Smith, Baker, Carpenter, Tailor],
    Sons = [Smithson, Bakerson, Carpenterson, Tailorson],
    % rule 1
    professions(Professions),
    fail. % replace with your solution
```

   Hint: Use the built-in `permutation` predicate to generate permutations of the professions.

2. *Logic queries.* For each of the following Prolog predicates and set of queries, do the following:

   - Predict whether the query succeeds, fails, or results in a runtime error, **without running the code**.
   - Run the code and queries. How do the results compare to your predictions?

a)
```
sum([], 0).

sum([First|Rest], Total) :-
    sum(Rest, RestTotal),
    Total is First + RestTotal.
```

Query	Succeeds	Fails	Error
`sum([1, 2], 3).`	◯	◯	◯
`sum([1, 2], 4).`	◯	◯	◯
`sum([1, A], 3).`	◯	◯	◯
`sum([1 | 2], Total).`	◯	◯	◯
`sum([1, 2], Total).`	◯	◯	◯

b)
```
contains([Item], Number) :-
    Item =:= Number.

contains([_|Rest], Number) :-
    contains(Rest, Number).
```

Query	Succeeds	Fails	Error
`contains([], 3).`	◯	◯	◯
`contains([3], 3).`	◯	◯	◯
`contains([-1], 3).`	◯	◯	◯
`contains([A], 3).`	◯	◯	◯
`contains([-1, 3, 7], 3).`	◯	◯	◯

3. *uC errors.* Consider the following code samples from the uC language in Project 4. Consult the uC spec and the Project 4 spec to determine the appropriate errors that should be output by your semantic analyzer (if any) as well as which phase the errors are caught in.

uC Code	Error(s) Reported	Phase

`int foo(int a)(boolean a) {`
`  return;`
`}`		

`void main(string[] args)(int c, int b) {`
`  c = b = 7;`
`  5 = 7;`
`}`		

`  void int_to_long()() {}`		

`  struct foo(boolean b);`
 	
`void main(string[] args)() {`
`  new foo(null);`
`  new foo(true, false);`
`  new foo { 3 };`
`}`		

... continued on next page

uC Code	Error(s) Reported	Phase

`void main(string[] args)(foo f) {`
`  f = new foo(args[0]);`
`  bar(f);`
`}`
 	
`void bar(foo f)() {`
`  println(f.s);`
`}`
 	
`struct foo(string s);`		