ECS 140A
Homework Assignment #4
Due no later than 9:00pm Tuesday, December 3, 2019

If you take a look at some Prolog programming texts you'll find solutions to most, if not all, of these problems. A little web surfing will locate solutions as well. However, you really want to do as much as you can without those aids, because you won't have those resources during your final exam. Also, this is not to be a team effort.

You may need to write additional procedures other than the ones listed here to make things work – be sure to include those when you submit your solutions. Submit your solutions via Canvas as a single file named "hw4.pl".

Grading for problems 1 through 4 will be on a 3-point scale for each solution (4 problems x 3 points maximum per solution = 12 points maximum). Problem 5 will be graded differently.

Problem 1: Write a procedure

```
shuffle(L1, L2, L3)
```

which returns true if list L3 is the result combining lists L1 and L2 such that the first element of L3 is the first element of L1, the second element of L3 is the first element of L2, the third element of L3 is the second element of L1, and so on. For example,

```
shuffle([a,b,c],[d,e,f],[a,d,b,e,c,f])
```

returns true. You may assume that L1 and L2 have the same number of elements. This should also work:

```
?- shuffle(X,Y,[1,2,3,4,5,6]).

X = [1, 3, 5],
Y = [2, 4, 6].
```

Problem 2: Write a procedure

```
double(L1, L2)
```

which returns true if every element in list L1 appears twice in L2, according to the pattern in the following example:

```
?- double([a,b,c],[a,a,b,b,c,c]).
true.
```

Note that the elements that are duplicated are also adjacent to each other.

These cases should also work:

```
?- double(X,[a,a,b,b,c,c]).
X = [a, b, c].

?- double([a,b,c],X).
X = [a, a, b, b, c, c].
```

This case should not work:

```
?- double([a,b,c],[a,b,c,a,b,c]).
false.
```

Problem 3: Write a procedure

```
no_duplicates(L1, L2)
```

which returns true if list L2 is the result of removing all duplicate elements from list L1. For example,

```
no_duplicates([a,b,c,b,d,b],[a,c,d,b])
```

returns true (note that the last duplicate 'b' is the one that remains). This should also work:

```
?- no_duplicates([a,b,c,b,d,b],X).
X = [a, c, d, b] ;
X = [a, c, d, b] ;
false.
```

Problem 4: Write a procedure

```
same_elements(L1, L2)
```

which returns true if lists L1 and L2 contain exactly the same elements, although possibly in different order. For example,

```
same_elements([a,b,c],[b,c,a])
same_elements([a,b,c],[a,c,b])
same_elements([a,b,c],[c,b,a])
```

all return true. This should also work:

```
?- same_elements([a,b,c],X).
X = [a, b, c] ;
X = [b, a, c] ;
X = [b, c, a] ;
X = [a, c, b] ;
X = [c, a, b] ;
X = [c, b, a] ;
false.
```

Problem 5: Using the minidoku program that we explored in class as inspiration, create a sudoku problem solver that works with any 9x9 grid as described in the slide that defines the sudoku problem. Your solution should be fast when few spaces in the grid are unfilled, but it can be slower (even astronomically slower) when more spaces are unfilled, just like our demonstration in class. You will find the minidoku code in your lecture slides. The skeleton code (i.e., you fill in the missing parts) for the 9x9 solution can be found in sudokusolver.pl.

Add your code to the existing skeleton and submit your result via Canvas as a separate file called “sudoku.pl” (not “hw4.pl”).