Assignment 5

Due: Friday, 10.06.2016, 15:59 via Git

For help, contact alp-staff@lists.iai.uni-bonn.de (staff only) or alp-course@lists.iai.uni-bonn.de (staff and participants).

Start working on the exercises early enough so that you can contact your tutor in time if you have problems. Don't expect your tutor to be available during weekends!

Submit your implemented predicates as a file named "assignment05/solutions.pl" in the Git repository of your group.

Add a file "assignment05/testRuns.txt" showing the console output of a session in which you test that **each** solution works for the provided input data and some queries that represent sensible test cases.

If no input data is provided in the text of the task, create some sensible input data.
If input data is represented as facts, include them into the "solutions.pl" file and add suitable comments.

Task 1. *Shifting lists* (2 Points)

Implement a predicate "shift(List1, List2)" so that List2 is List1 'shifted rotationally' by one element to the left, which means that the element that is shifted out on the left-hand-side comes in again on the right-hand-side. For instance,

```
?- L1 = [1,2,3,4,5], shift(L1, L2), shift(L2, L3).
```

should produce

```
L1 = [1,2,3,4,5]
L2 = [2,3,4,5,1]
L3 = [3,4,5,1,2]
```

Task 2. *Mapping list elements* (4 Points)

Implement a predicate "translate(DigitList, WordList)" that succeeds if DigitList is a list of digits and each element in WordList is the English word for the element at the same position of DigitList. For instance,

```
?- translate( [1,2,3], L2).
```

should produce

```
L2 = [one, two, three]
```

and

```
?- translate( L1, [one, two, three]).
```

should produce

```
L1 = [1,2,3]
```

Tip: Start by considering how to represent the mapping of digits to words and vice versa. To check whether a term is a digit use the following helper predicate, which uses built-in predicates that we haven't discussed yet – see the SWI-Prolog online documentation:

```
digit(X) :- number(X), X>=0, X<10.
```

Task 3. *Linear list* (4 Points)

Implement a predicate "linear(+NestedList, ?LinearList)" that succeeds whenever NestedList is a list whose elements may be arbitrarily deeply nested lists and LinearList contains all elements of NestedList in the same order but without any nesting. For instance,

```
?-linear ( [1,[2,3],[a,[b],c]], [1,2,3,a,b,c]).
```

should succeed but

```
?-linear ( [1,[2,3],[a,[b],c]], [1,2,3,a,c,b]).
```

should fail (because the order of c and b is inversed).

Tip: You may use the predefined predicate `append(A,B,AB)` to implement your version of linear/2. Except for that, your solution must be self-contained.

Task 4. *Grouping consecutive list elements* (6 Points)

Write a predicate that groups consecutive repeated elements of a list into sublists. If a list contains non-consecutive repeated elements (such as the different occurrences of 1 in the example below) they should be placed in separate sublists. Example:

```
?- group([1,1,1,1,2,c,c,1,1,d,e,e,e,e],X).
X = [[1,1,1,1],[2],[c,c],[1,1],[d],[e,e,e,e]]
```

Task 5. *Grouping consecutive list elements* (2 Points)

Modify your predicate from Task 4 so that it does not put an element into a sublist if there is no consecutive repeated element. Example:

```
?- group([1,1,1,1,2,c,c,1,1,d,e,e,e,e],X).

X = [[1,1,1,1],2,[c,c],[1,1],d,[e,e,e,e]].
```

Tip: In both cases (Task 4 and 5) the essential question is "How can your predicate remember whether it had seen the same element in the previous step?"