Assignment 3

Due: Friday, 26.05.2017, 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 at midnight or during weekends!

Submit results into the folder "assignment03/" of the git repository of your group.

For each task, submit your implemented predicate as a file named "taskN.pl". At the bottom of the file add a comment containing console output that shows all results of a successful test run of your predicate.

Task 1. *Classification and Negation (3 Points)*

Assume we have a database of results of tennis games played by members of a club. The results are represented as facts for the predicate beat/2, meaning that the player mentioned in the first argument has beaten the player in the second argument:

```
beat( tom, jim ).      % tom has beaten jim
beat( ann, tom ).      % ann has beaten tom
beat( pat, jim ).      % pat has beaten jim
```

Your task is to define a predicate "category(Player,Category)" that classifies the players into three categories:

1) ***winner:*** A player who won all his or her games.
2) ***fighter:*** A player who won some games and lost some.
3) ***loser:*** A player who lost all his or her games.

For instance, "?- category(tom, fighter)." should succeed.

Task 2. *Double negation (2 Points)*

Given the following program discuss and argue whether r/1 and s/1 are equivalent or not.
Provide a simple definition of p/2 on which you can demonstrate your arguments.

Tip: Consider the possible invocation modes of r/1 and s/1.

```
r(X) :- p(a,X).

s(X) :- not(not(p(a,X))).
```

Task 3. *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 4. *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 5. *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 in argument 1 and argument 2 differs).

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.