Declaratieve Talen

Prolog 4

1 One pass translation

The goal of this exercise is to define a predicate `translate/2` that converts a sequence (a list) of `def/1` and `use/1` terms to a sequence of `asgn/2`, `use/1`.

For instance, for an input list:

```
[def(a), use(a), use(b), use(c), def(c), def(b)]
```

the output should be:

```
?- translate([def(a), use(a), use(b), use(c), def(c), def(b)], L).
L = [asgn(a, 1), use(1), use(3), use(2), asgn(c, 2), asgn(b, 3)].
```

The symbols in the `use` terms are mapped to integers, starting from 1. In the translated list, `def(sa)` terms are replaced by `asgn(sa,nb)`. Similarly `use(sa)` is replaced by `use(nb)` where `nb` is the number that is assigned to the symbol `sa`.

The order of the `def` statements determines the number that is assigned to each symbol, e.g. if `def(a)` occurs first in the list, `a` will be assigned number 1, if `def(b)` occurs second in the list `b` will assigned 2, etc. In general the symbol corresponding to the N-th `def` term will be assigned the number N.

However, `use` statements can occur before their corresponding `def` statements. At first glance, this necessitates a two-pass algorithm: in the first pass all `def` statements are checked, and each symbol is assigned a number. In the second pass, symbols are replaced by their corresponding number in the `use` and `def` statements.

Use the fact that the substitution computed by unification is applied globally to the entire execution to implement `translate/2` *in a single pass*. Global application means that free variables that are introduced earlier in the execution tree can be retroactively filled in by later parts of the execution.

Hint: Use a symbol table to store assignments from symbols to numbers.

2 Holiday lights

The European government has decided to festively light major European highways during the holiday season. There will be differently colored lights, however

each highway will have exactly one color. During the holiday season you plan some family visits, and you like some variety while driving. Therefore you decide to drive each highway exactly once, but two consecutive highways you drive can not be lit by the same color.

A more abstract representation of this problem is as follows. Assume M bidirectional connections that each connect two places, and are lit by a specific color. Assume N places, numbered from 1 through N. You start at place 1. Now you have to plan a trip such that each connection is used exactly once, no two consecutive connections have the same color, and your trip should end back at place 1.

Consider the following example. There are three places. There is a yellow highway between places 1 and 2, a blue highway between 2 and 3, and a yellow highway between 1 and 3. You can drive a yellow highway from 1 to 2, a blue highway from 2 to 3, and another yellow highway from 3 back to 1.

Such a graph can be represented by Prolog facts. For our example we can write:

```
highway(1,2,yellow).
highway(2,3,blue).
highway(1,3,yellow).
```

Before you start planning your trip, you need to check the following two conditions:

1. The nodes (i.e. places) have an even number of connections,
2. If a node K, such that $1 < K \leq N$, has X connections lit by the same color, then the node K should have at least X connections with other colors.

Write the predicate `check` which will check these two conditions for a network of highways given by `highway/3` facts. This predicate succeeds if both conditions are met, and fails otherwise. Some examples:

```
highway(1,2,yellow).
highway(2,3,yellow).
highway(1,3,blue).

?- check.
fail.

highway(1,2,yellow).
highway(2,3,blue).
highway(1,3,yellow).

?-check.
true.
```

Write the predicate `tour(T)` which will first check the conditions listed above for a network of highways given by `highway/3` facts, and then calculates a trip. Our example has two possible trips: `T = [2-yellow, 3-blue, 1-yellow]` and `T = [3-yellow, 2-blue, 1-yellow]`. In such a case you only return the smallest trip: `T = [2-yellow, 3-blue, 1-yellow]`. We will use the order defined by `@</2`. Notice that this query `?- [2-yellow] @< [3-yellow]` succeeds.

The solution `T = [2-yellow, 3-blue, 1-yellow]` tells us we will drive a yellow highway from 1 to 2, then a blue highway from 2 to 3, and finally a yellow highway from 3 to 1.

The predicate `tour` will fail if `check` fails. Some examples:

```
highway(1,2,yellow).
highway(2,3,blue).
highway(1,3,yellow).

?- tour(T).
T = [2-yellow, 3-blue, 1-yellow]


highway(1,2,c).
highway(2,3,a).
highway(1,3,b).
highway(3,5,a).
highway(3,4,c).
highway(5,4,d).

?- tour(T).
T = [2-c, 3-a, 4-c, 5-d, 3-a, 1-b] ;
```