Logical reasoning and programming, lab session VI

(November 6, 2017)

VI.1 An interesting built-in predicate is `repeat/0`. It always succeeds and when reached by backtracking it creates a new choice point. How would you write this predicate in Prolog?

It is useful for failure driven loops like

```
printfile(File):-
    see(File),
    repeat,
    read(Term),
    ( Term == end_of_file
      -> !
      ;
      print(Term),nl,
      fail
    ),
    seen.
```

VI.2 Compare the output of `?-findall(_,fail,L).` and `?-bagof(_,fail,L).`

VI.3 Use the standard definition of Fibonacci numbers, $F_1 = 1$, $F_2 = 1$, and $F_n = F_{n-1} + F_{n-2}$, for $n > 2$, and write a straightforward (and highly inefficient) recursive program that computes them, e.g., use a predicate `fib(+N,-FN)`.

One trick how to improve the program is the caching of results, try that using `asserta/2`. Do not forget to add something like

```
:-dynamic fib/2.         % your predicate has to be dynamic
:-retractall(fib(_,_)).  % remove fib/2 from the database
```

at the beginning of your program. Check also the behavior of your program after several runs without using `retractall/2`. Moreover, check your database using `?-listing(fib).`

However, there is still a much better way how to compute Fibonacci numbers using accumulators. Do that and do not hesitate to use as many new variables as needed.

VI.4 Try to use our vanilla meta-interpreter

```
prove(true):-!.
prove((A, B)):-!,
    prove(A),
    prove(B).
prove(A):-
    clause(A,B),
    prove(B).
```

on some programs in pure Prolog. What happens if we use it on itself? Try for example `?-prove(prove(true))`.

The problem is caused by the use of built-in predicates like `!` and `clause/2`. Propose a workaround to these problems.

VI.5 Our definition of a difference list, defined as a pair of lists, says nothing about the meaning of `[a,b]-[c,d]`. However, if your try to use append defined using difference lists

```
append_dl(XPlus-XMinus, XMinus-YMinus, XPlus-YMinus).
```

you still get something. Try for example `?-append_dl(X, Y, [a,b]-[c,d]).` and `?-append_dl([a,b]-[c,d], Y, Z).`

VI.6 First, write predicate `reverse(L1, L2)`, where the list `L2` is obtained from the list `L1` by reversing the order of elements, using an accumulator. Hence you will obtain something like `reverse_acc/3`.

Once you have it, try to obtain a binary predicate `reverse_dl`, where you use a difference list instead of an accumulator[1].

(Hint: Removing a head from `L1` can be similarly described by adding it at the beginning of the minus part of `L2`.)

VI.7 Write a grammar (DCG) that accepts the language $L = \{a^n b^n : n \geq 0\}$. The start symbol is, e.g., `s`. Hence `?-phrase(s, L)` should produce all possible words in L.

VI.8 Write a grammar (DCG) that accepts the language $L = \{a^n b^n c^n : n \geq 0\}$. The start symbol is, e.g., `s`. Hence `?-phrase(s, L)` should produce all possible words in L.

Change your grammar in such a way that you can get the pairs n and $a^n b^n c^n$:

```
?- phrase(s(N),L).
N = 0,
L = [] ;
N = 1,
L = [a, b, c] ;
N = 2,
L = [a, a, b, b, c, c]
```

(You can also recall your knowledge of the pumping lemma for context-free languages and show that L is not a context-free language.)

[1]Note that a difference list is represented by two lists and hence there are still three lists involved.