**Kiel University**
Department of Computer Science
Programming Languages and Compiler Construction

Priv.-Doz. Dr. Frank Huch, Sandra Dylus, B.Sc. Jan-Hendrik Matthes, Rajib Chandra Das

Institut für Informatik

# 11. Exercise Sheet on „Advanced Programming (pre-masters)“

Logic Programming 1

Submit by Thursday, 27. June 2019 - 10:00

---

**Note:** In the first weeks we will use a modified version of the Prolog-interpreter. After installing SWI-Prolog you need to download one of the following files – depending on your OS.

- *Unix/Linux:* Copy this bash-script anywhere you like and add it to your `PATH`-variable. It is recommended to copy it to a dedicated `bin`-directory of your OS. You can then start the modified interpreter using the command `advprog-swipl`. Alternatively, you can copy the bash-script to the directory with (all) your Prolog files and use `./advprog-swipl` to start the interpreter.
- *Others:* Copy this Prolog program into the directoy with (all) your Prolog files and start the interpreter using the command `swipl -q -s advprog`.

If you start the interpreter as described above, you can load a Prolog file named `myprog.pl` using `[myprog].` Note that you cannot use negation in your Prolog programs with the modified interpreter.

## Exercise 1 - Who is on board? 2 Points

A Quidditch club would like to put together a three-person board consisting of a chairperson, a treasurer and a secretary. The candidates are *Potter*, *Weasley*, *Malfoy* and *Granger*. There are however different preferences of the individual members, which limit the possibilities for the formation of a board.

1. *Potter* and *Malfoy* do not want to join the board together.
2. *Malfoy* is only available for the board if *Granger* becomes chairperson.
3. *Weasley* will only join the board if *Potter* also belongs to it.
4. *Potter* does not want to join the board if *Granger* becomes the secretary.
5. *Granger* does not join the board if *Weasley* becomes chairperson.

Write a Prolog program that calculates all possible board member constellations.

**Note:** Make sure to avoid duplicate answers!

## Exercise 2 - Predicates 2 Points

1. Add the following predicates to the family example from the lecture.
   - `son(S, P)` is fulfilled if `S` is a son of `P`.
   - `cousin(C, P)` is fulfilled if `C` is a cousin (male or female) of `P`.
   - `brother_in_law(B, P)` is fulfilled if `B` is a brother in law of `P`.
   - `ancestor(A, P)` is fulfilled if `A` is an ancestor of `P`.

2. Implement the following predicates using the `append` predicate from the lecture.

   - `lookup(K, KVs, V)` is fulfilled if the tuple `(K, V)` is included in the list `KVs`.
   - `member2(E, L)` is fulfilled if `E` is included at least twice in the list `L`.
   - `reverse(Xs, Ys)` is fulfilled if the inversion of list `Xs` corresponds to list `Ys`. Implement both the naive and the efficient variant (accumulator technique using an additional argument).
   - `sublist(Xs, Ys)` is fulfilled if `Xs` occurs as a sublist in `Ys`.

### Exercise 3 - Binary Numbers

2 Points

In the lecture we presented natural numbers as Peano numbers. However, this representation is very inefficient for large numbers. We therefore want to present natural numbers as binary numbers.

- We distinguish the number 0, represented as `o`, and positive numbers $> 0$ represented as `pos(N)`.
- Positive numbers are represented as a bit sequence, where the most significant bit is always 1, so there are no leading zeros. Positive numbers consist of the following functors.
  - A term of the form `o(N)` represents the number $2 \cdot N$,
  - a term of the form `i(N)` represents the number $2 \cdot N + 1$ and
  - the atom `i` represents the number 1 (most significant bit).

  The number 0 is therefore called `o`, 1 as `pos(i)` and 4 as `pos(o(o(i))`. Hence, the outermost functor represents the least significant bit.

1. Define the following predicates based on this number representation.

   - Addition of two natural numbers `add(X, Y, Res)`.
   - Subtraction of two natural numbers `sub(X, Y, Res)`.

   **Note:** Use auxiliary predicates such as `addP`, which are defined on positive numbers. Furthermore, use the following equations and develop equations for further combinations if necessary.

$$\begin{aligned} 1 + 1 &= 2 \\ 1 + (2 \cdot N) &= 2 \cdot N + 1 \\ 1 + (2 \cdot N + 1) &= 2 \cdot (N + 1) \end{aligned}$$

2. When used in the opposite direction, e.g. with a query like `add(X, Y, pos(o(i)))`, Prolog computes all solutions, but, unfortunately, does not terminate. The reason is, that the Prolog program does not know that the sum of two positive numbers is greater than each summand, and thus increasing summands are tried.

   Implement for positive numbers $> 0$ a predicate `lessP(X, Y)` that is satisfied, when `X` is less than `Y`. Then use this predicate to improve the previous implementation of `add`.

### Exercise 4 - Boolean Operations

2 Points

Implement Boolean functions `and`, `or`, and `not` as predicates working on the values `true` and `false`. Make sure that negation is also possible. Therefore, you must provide an explicit result as an additional parameter of your predicate.

Also implement the following three Boolean expressions as Prolog predicates `ex1(X, Y, Z, Res)`, `ex2(X, Y, Z, Res)`, and `ex3(X, Y, Z, Res)`.

- $(x \wedge y) \vee z$
- $(x \wedge y) \vee ((y \wedge z) \wedge z)$
- $(x \wedge (\neg y) \wedge z) \vee ((z \wedge y) \vee z)$

Specify queries that can be used to answer the following questions. Give the answer of the Prolog system as well as an additional explanation.

- Which results do you get for the values `X = true`, `Y = false` and `Z = true`?
- Which assignments yield `true` as result?
- Is the third equation dependent on $x$ or $z$?