Prof. Dr. Jürgen Giesl

Carsten Otto

Notes:

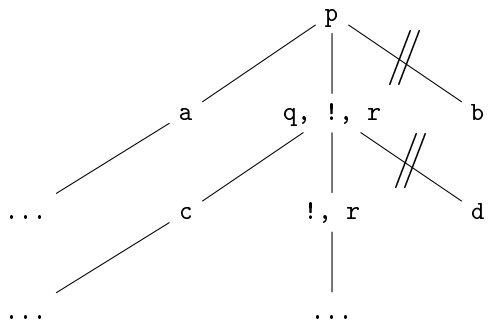
- To solve the programming exercises you can use the Prolog interpreter **SWI-Prolog**, available for free at `http://www.swi-prolog.org`. For Debian and Ubuntu it suffices to install the `swi-prolog` package. You can use the command "`swipl`" to start it and use "`[exercise9].`" to load the clauses from file **exercise9.pl** in the current directory.
- Solve these exercises in **groups of three**! For other group sizes **less points** are given!
- The solutions must be handed in **directly before (very latest: at the beginning of)** the exercise course on Wednesday, 03.07.2013, in lecture hall **AH 2**. Alternatively you can drop your solutions into a box which is located right next to Prof. Giesl's office (this box is emptied **a few minutes before** the exercise course starts).
- Please write the **names** and **immatriculation numbers** of all (three) students on your solution. Also please staple the individual sheets!

Exercise 1 (Cut): (4+6=10 points)

Consider the following Prolog program:

```
isPrime(2).
isPrime(X) :- X > 2, numbersFromTo(2, X, R), 0 is X mod R, !, fail.
isPrime(X) :- X > 2.
numbersFromTo(2, _, 2).
numbersFromTo(LOW, UP, RES) :- LOW+1 < UP, RES is LOW + 1.
numbersFromTo(LOW, UP, RES) :- LOW+1 < UP, TEMP is LOW + 1, numbersFromTo(TEMP, UP, RES).
```

In the next exercise parts you need to give graphical representations of SLD trees. For every part of a tree that is cut off by evaluating `!`, please indicate the cut (as shown in the graphics). For the cut-off parts only indicate the first cut-off goal, but do not evaluate further (i.e., do not continue below `b` or `d`).



a) Please give a graphical representation of the SLD tree for the query `?- isPrime(3).`

b) Please give a graphical representation of the SLD tree for the query `?- isPrime(9).`

Exercise 2 (Meta-Variables): (2 points)

Important: In addition to handing in the solution on paper, please also mail your the solutions for this exercise to **lp13-hiwis@i2.informatik.rwth-aachen.de**. Indicate your immatriculation numbers in the subject of the mail and inside the Prolog file.

In the lecture the binary predicate `or` (`;`) was presented which makes use of meta-variables. In this exercise we want to extend this idea to the n-ary predicates *or, nor, and, nand.*

- The function $or(a_1, ..., a_n)$ is true iff a_i is true for at least one $1 \leq i \leq n$. For $n = 0$, *or* is false.
- The function $nor(a_1, ..., a_n)$ is true iff no a_i $(1 \leq i \leq n)$ is true. For $n = 0$, *nor* is true.
- The function $and(a_1, ..., a_n)$ is true iff all a_i $(1 \leq i \leq n)$ are true. For $n = 0$, *and* is true.
- The function $nand(a_1, ..., a_n)$ is true iff at least one a_i $(1 \leq i \leq n)$ is false. For $n = 0$, *nand* is false.

Please implement these four predicates in Prolog where the (only) argument should be a list (using the predefined data structure for lists in Prolog). You may not use `;` in your solutions! However, you may use cuts (`!`) and negation (`\+`).

Exercise 3 (Operators): (2+2=4 points)

Important: In addition to handing in the solution on paper, please also mail your the solutions for this exercise to **lp13-hiwis@i2.informatik.rwth-aachen.de**. Indicate your immatriculation numbers in the subject of the mail and inside the Prolog file.

a) For the past exercises you often had to find out if a number X is a divisor of a number Y. Now we want to use a new operator `#` so that `X # Y` can be written in Prolog programs.

Define `#` to be an infix operator and also give clauses so that `#` has the desired semantics. Define the precedence of `#` so that `1 + 3 # 2 * 6` is true.

b) Over natural numbers, the *monus* function is defined as standard subtraction, but gives 0 instead of negative results:

$$monus(x, y) = \begin{cases} x - y & \text{if } x \geq y \\ 0 & \text{otherwise} \end{cases}$$

Please implement the monus function in Prolog by defining `--` (two dashes) as an infix operator and by adding clauses so that `--` has the desired semantics. For `--`, use the same precedence and the same type as for `-` (standard subtraction). On negative numbers your implementation of `--` may have arbitrary results.

For example, the query "`X is 5 -- 12`" gives the answer substitution `X = 0`, while "`X is 5 -- 3`" gives `X = 2`.

Hint: In order to make `--` behave like an arithmetic function (so that it can be used on the right-hand side of `is`), you need to write `:- arithmetic_function('--'/2).` at the top of your program after the `op`-directive.