If you like logic or functional programming, then we recommend our lecture and seminar on term rewriting systems in the winter semester 2015/16. It is suitable for students of Bachelor and Master Informatik, Master SSE, and Master Mathematik. For more information, please see: http://verify.rwth-aachen.de/tes15/

Important: This sheet is only relevant for students of the V3M version of this lecture.

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 “[exercise1].” to load the facts from the file exercise1.pl in the current directory.

- Please solve these exercises in groups of two or three!

- The solutions must be handed in directly before (very latest: at the beginning of) the exercise course on 10.07.2015, in lecture hall AH 1. 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 students on your solution. Also please staple the individual sheets!

Exercise 1 (Procedural Semantics):  (5 (only V3M) points)

Consider the following constraint logic program $P$ for the computation of the Fibonacci numbers together with the constraint theory $CT_{FD}$ from Example 6.1.4 of the lecture (i.e., $CT_{FD}$ consists of all true formulas on integers):

\[
\begin{align*}
\text{fib}(0,0). \\
\text{fib}(1,1). \\
\text{fib}(N,F) &: N > 1, N_1 = N - 1, N_2 = N - 2, \text{fib}(N_1,F_1), \text{fib}(N_2,F_2), F = F_1 + F_2.
\end{align*}
\]

Please write down both successful computations of the query $G = \{\neg \text{fib}(X,1)\}$ w.r.t. $P$ and $CT_{FD}$, i.e., a sequence of configurations of the form $(\text{fib}(X,1), \text{true}) \vdash_P \cdots \vdash_P (\emptyset, CO)$. As in Example 6.1.15 from the lecture, you may leave out the negations in the queries and simplify your constraints after every computation step. Moreover, please give the computed answer $P[P, CT_{FD}, G]$.

Exercise 2 (SLD Trees With CLP):  (5 (only V3M) points)

Consider the following Prolog program for the well-known Ackermann function with constraints from $CT_{FD}$:

\[
\text{:- use_module(library(clpfd)).} \\
\text{ack}(0,M,R) :- !, R = M + 1. \\
\text{ack}(NP,0,R) :- !, N = NP - 1, \text{ack}(N,1,R). \\
\text{ack}(NP,NP,R) :- N = NP - 1, M = MP - 1, \text{ack}(NP,M,RP), \text{ack}(N,RP,R).
\]

Please give a graphical representation of the full SLD tree for the query ?- $\text{ack}(1,1,2)$. You can restrict your representation to those parts of the tree where the conjunction of constraints is satisfiable (i.e., do not show the parts where the edges contain unsatisfiable constraints). Please also give the answer substitution.
Exercise 3 (Programming in CLP): (3+4 (only V3M) points)

**Important:** In addition to handing in the solution on paper, please also mail your solutions for this exercise to lp15-hiwis@i2.informatik.rwth-aachen.de. Indicate your immatriculation numbers in the subject of the mail and inside the Prolog file.

a) Write a Prolog query to find all triples \((a, b, c) \in \mathbb{N}^3\) such that \(a^2 + b^2 = c^2\) and \(a + b + c = 1000\). Do not define any new predicates. You may assume that the clpfd module was loaded before (e.g., by entering `use_module(library(clpfd)).` in the Prolog shell).

b) The mathematical puzzle *Graph-Sudoku* is a generalization of the popular riddle Sudoku. We are given a Graph \(G = (V, E)\) with vertices \(V\) and edges \(E\) and we want to find an assignment \(V \mapsto \{1, \ldots, n\}\) such that no two neighboring nodes in the graph have the same number. If one replaces the vertices by these numbers, we call it a valid Graph-Sudoku.

Your task is to implement a CLP predicate `gsudoku/2` in the constraint theory \(CT_{FD}\). Let a graph be given by a list `Edges` of edges `e(X,Y)`, let \(N\) be the maximal number to be used, and let `V` be a list of pairwise different variables, representing the vertices. The query `?- gsudoke(Edges,N), label(V).` should be provable if the variables used in `Edges` can be instantiated to a valid Graph-Sudoku in our representation. In addition, if `gsudoke(Edges,N), label(V).` is provable, then variable entries in the individual edges should be instantiated by the answer substitution.

For example, the list `[e(X,Y), e(Y,Z), e(Z,X)]` represents the following graph:

```
   X
  / \  \
 /   \ 
Y-----Z
```

The query `?- gsudoke([e(X,Y), e(Y,Z), e(Z,X)], 2), label([X,Y,Z]).` should fail (there is no valid assignment from \([X, Y, Z]\) to \([1, 2]\)). However, the query `?- gsudoke([e(X,Y), e(Y,Z), e(Z,X)], 3), label([X,Y,Z]).` should have the result \(X = 1\), \(Y = 2\), \(Z = 3\) (or any other permutation).