Exercise 3

Foundations of Constraint Programming

Prof. Steffen Hölldobler, Sebastian Haufe

International Master Program in Computational Logic — winter term 2010/2011

Date of exercise: 19.11.2010

Exercise 3.1

Apply the Domain Reduction Rules from Slide 3/5 to the following CSPs until you reach a successful, failed or stabilising CSP. At each step give the rule you have used.

a) $\langle x = y, y = z, x \neq w, w \neq z \; ; \; x \in \{a, b, c\}, y \in \{a, c, d\}, z \in \{c, d, e\}, w = c \rangle$

b) $\langle x \neq w, w < y, w < z, y < z \; ; \; x \in [4..8], y \in [2..6], z \in [3..6], w \in [4..9] \rangle$

Exercise 3.2

Give an infinite derivation in the Unif Proof System (cf. Slides 3/16 and 3/17) such that each rule application is relevant.

Hint: This is possible only with non-global applications of the Substitution rule.

Exercise 3.3

Take the following set of linear equations:

$$\begin{array}{rcrcrcl} a & + & b & + & c & = & 0 \\ 4a & + & 2b & + & c & = & 1 \\ 9a & + & 3b & + & c & = & 3 \end{array}$$

a) Apply Gauss-Jordan Elimination to compute an mgu for this set of equations.

b) Apply Gaussian Elimination to compute an mgu for this set of equations.

Exercise 3.4

A magic square of size n is an $n \times n$ matrix containing the numbers from 1 to n^2 (each number exactly once) such that the sums of each row, each column and the two main diagonals are equal.
Write a program in Eclipse-Prolog which generates magic squares of size n using the constraint solving library `ic` (load it with `:- lib(ic).`). Solve this task via the following subtasks:

a) Use a list of lists data structure. For example, $[[2,7,6],[9,5,1],[4,3,8]]$ represents a magic square of size $n = 3$ where the first row is 2 7 6 , the first column is 2 9 4 and the upper left to lower right diagonal is 2 5 8 .

   Define a predicate `assignM(Matrix,N,Nsq)` which instantiates a fresh variable *Matrix* to a list of N rows, where each row itself is a list containing N variables with domain $[1..Nsq]$.

b) Define a predicate `rowC(Matrix,Sum)` (`colC(Matrix,Sum)`, `diagC(Matrix,Sum)`, resp.) which takes a list of lists *Matrix* as initialized in a) and adds constraints such that the sum of each row (column, diagonal, resp.) equals *Sum*.

   Hint: Use the predefined predicate `sumlist/2` from the library `ic_global` which can be loaded with `:- import sumlist/2 from ic_global.`

c) Define a predicate `square(N)` which computes a magic square of size N.