Constraint Programming
Prof. Dr. Thom Frühwirth

Assignment 1

Execise 1. Make yourself familiar with SICStus Prolog, which is installed in the Linux pool:

- Go to `www.sics.se/sicstus/` for information regarding SICStus Prolog. Read the chapter "How to Run Prolog".
- Install Emacs-Support for SICStus: Add the line `(load "/opt/sicstus4.0.1/lib/sicstus-4.0.1/emacs/sicstus_emacs_init")` to the file `~/.emacs` (where `~` is your home directory).
- Write a "Hello world!" program in SICStus Prolog, e.g.,
  `start :- write('Hello world!').`
  Compile the source in Emacs and call it with `start.`

Execise 2 (CLP – "generate and test" vs. "constrain and generate").
Use the *permutation sort* algorithm for sorting a list of integers. Analyse the run time complexity of each implementation with respect to the length of the list.

a) To implement the *generate and test version*, use three Prolog predicates `permsort(List,Sorted)`, `permutation(List,Sorted)`, and `sorted(Sorted)` (all arguments are lists).
   A list R is the sorted version of the list L if R is a permutation of R and the elements or R are sorted in increasing order.

b) For the *constrain and generate version*, use the constraint solver given in the modul `clpq` and refer to the SICStus manual – chapter 33 – for details.
   Hint: Your source code must include `:- use_module(library(clpq)).`
   For example, the constraint solver simplifies the following two constraints (which must be enclosed in braces) `{A=<3, A=<5}` to `{A=<3}`.
   Convert, both the explicit as well as the implicit compare operations (i.e., the operations that compare list elements) from part (a), into `clpq` constraints by attaching braces.
   Note that in the definition of `permsort`, the predicate `sorted` should be called before `permutation` in order to constrain first.
   Testing should include lists with variables (e.g. `permsort([1,A,3],Sorted)`) and already sorted (output)lists (e.g. `permsort(X,[1,3,7])`).
   If you run into an endless loop, explain why and fix the problem.

Execise 3.
Implement a Prolog program to help scheduling routes for the freight forwarding business. The program uses the following knowledge base, giving distances between major cities in Germany.

```
Hamburg,Bremen,80       Hamburg,Hannover,110
Hamburg,Berlin,230      Bremen,Hannover,100
Bremen,Dortmund,200     Hannover,Kassel,140
Hannover,Nuernberg,380  Dortmund,Kassel,130
Dortmund,Koeln,80       Kassel,Wuerzburg,180
Kassel,Frankfurt,180    Frankfurt,Wuerzburg,110
Nuernberg,Muenchen,160
```

a) Represent the distances with a Prolog predicate `dist(city0,city1,km)`.
   Test your implementation with queries like `?- dist(dortmund,bremen,Km)`,
   `?- dist(From,To,180)`, and `?- dist(From,To,Km)`.

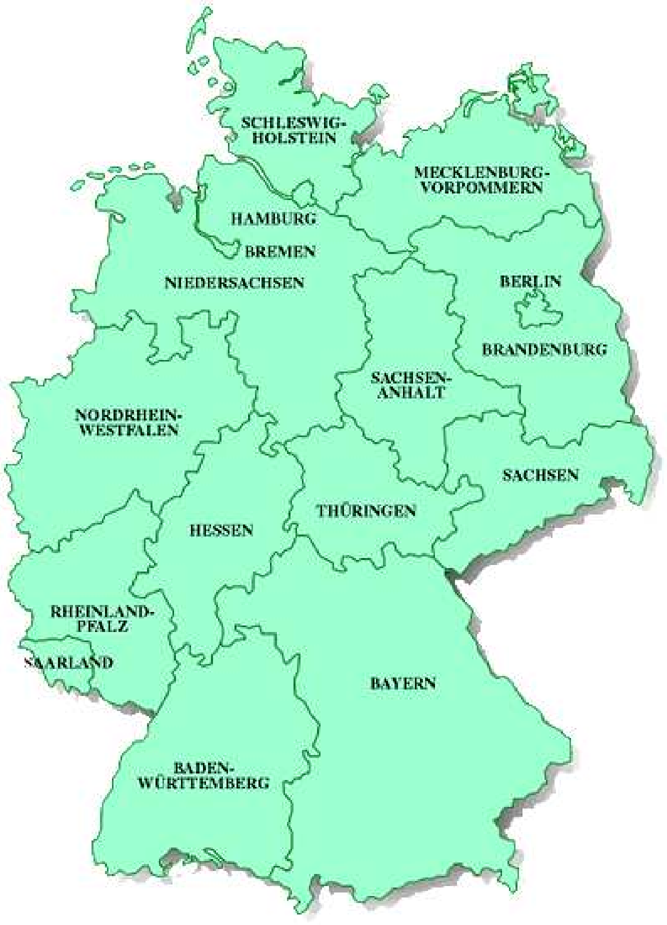
b) The Prolog predicate `route(From,To)` should succeed, iff there is a route between `From` and `To`.
   Tests should include `route(muenchen,berlin)`, `route(ulm,halle)`, and finally `route(frankfurt,To)`.

c) Define a ternary predicate `route(From,To,Route)` which has the same functionality as `route/2` with respect to the first and second parameters `From` and `To`. Additionally, the third parameter, `Route`, contains the cities already visited in order to avoid cycles (i.e. visiting a city more than once).
Thorough testing should include ground and non-ground queries, e.g.,
`?- route(dortmund,X,[kassel]).`

d) The Prolog predicate `shortestpath(From,To,Route,Km)` should succeed, iff the minimal distance between the cities `From` and `To` is `Km` along `Route`.
Tests should include, e.g., `shortestpath(kassel,bremen,[hannover],X)`.

Execise 4 (Map Coloring)**.**

The Prolog predicate `color(Map,Colors)` should succeed if each state could be assigned a color from `Colors`, such that no two bordering states have the same color. The element of the list `Map` are tuples (`Land,Color`).
Compute a valid assignment using four colors.



Execise 5. Decompose the following equations according to Clark's Equality Theory. Determine whether the equations are consistent and give the resulting variable bindings.

(a)	$p(a, X, c, Y, Z)$	$=$	$p(Y, X, c, a, W)$
(b)	$p(Y, g(Y, f(Y)), g(X, a))$	$=$	$p(f(U), V, g(h(U, V, W), U))$
(c)	$p(g(X), h(a, X), h(Y, Y))$	$=$	$p(U, h(a, g(V)), h(V, g(U)))$
(d)	$p(h(f(X), a), X, h(f(f(X)), f(f(X))))$	$=$	$p(h(V, a), U, h(W, f(V)))$