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 "`[exercise10].`" to load the clauses from file **exercise10.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, 10.07.2013, 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 students on your solution. Also please staple the individual sheets!

Important: This sheet is only relevant for students attending the **V3B** version of the lecture.

Exercise 1 (Unification): (6 points)

Please implement the binary predicate `unify` in Prolog. This predicate should have the same semantics as `unify_with_occurs_check`, but you may not use `unify_with_occurs_check` in your solution.

Hints:

- Use the predicate `==` to check terms for syntactical equality.
- Also use the predicates `var` and `=..`

Exercise 2 (Prolog interpreter): (4 points)

Based on the meta-interpreter presented in the lecture[1], please implement a meta-interpreter for pure logic programs that, for each found solution, prints the instantiated rules and facts that were used to find the solution (in the order they were used).
As an example, for the `path` program from exercise sheet 7, using the query `?- prove(path(a, a, s(s(0)))` should give the following output after pressing ';' twice, where the output of `prove` is shown in *italics*.

```
?- prove(path(a,a,s(s(0)))).
path(a,a,s(s(0))) :- true.
true ;
path(a,a,s(s(0))) :- edge(a,b),path(b,a,s(0)).
edge(a,b) :- true.
path(b,a,s(0)) :- edge(b,a),path(a,a,0).
edge(b,a) :- true.
path(a,a,0) :- true.
true ;
false.
```

Hints:

[1] Please also have a look at the end of this exercise.

- Take care to only print clauses that lead to a solution!
- Yoy may use the predefined predicate `append/3` to append lists.

The following program contains the `path` program and the basic meta-interpreter from the lecture. This file also can be downloaded from the lecture homepage.

```
path(X, X, Y).
path(X, Y, s(Z)) :- edge(X, A), path(A, Y, Z).
path(X, Y, Z) :- eps(X, A), path(A, Y, Z).

edge(a, b).
edge(b, a).
edge(c, d).
edge(d, b).

eps(b, c).

%prove(true) :- !.
%prove((Goal1, Goal2)) :- !, prove(Goal1), prove(Goal2).
%prove(Goal) :- clause(Goal, Body), prove(Body).
```

Exercise 3 (IO): (8 points)

Please write a predicate `tc/2` that reads a graph structure from a file, computes the transitive closure of all edges and writes back the result to another file. The first argument of `tc` should be the input file, the second argument is the output file.
As an example, if the input file contains

```
edge(a, b).
edge(b, c).
edge(b, d).
edge(c, e).
```

the output file must contain these edges and, in addition, the following edges which result from transitivity:

```
edge(a, c).
edge(a, e).
edge(a, d).
edge(b, e).
```

Hints:

- First collect all edges from the input file in a list.
- Based on the list of known edges, find combinations of edges that are not yet contained in the list. For example, `edge(a, b)` and `edge(b, d)` may be combined to `edge(a, d)`, which is not in the initial list.
- Extend the list of known/computed edges until no further edge can be found. Only then output the edges.