Prof. Dr. Jürgen Giesl

Carsten Fuhs, Carsten Otto, Thomas Ströder

Master Exam Version V4

First Name: ______________________________

Last Name: ______________________________

Immatriculation Number: ______________________________

Course of Studies (please mark exactly one):

○ **SSE Master** ○ **Other:** ______________________

	Maximal Points	Achieved Points
Exercise 1	10	
Exercise 2	9	
Exercise 3	6	
Exercise 4	10	
Exercise 5	5	
Exercise 6	10	
Exercise 7	10	
Total	60	
Grade	-	

Instructions:

- On every sheet please give your **first name**, **last name**, and **immatriculation number**.
- You must solve the exam **without** consulting any **extra documents** (e.g., course notes).
- Make sure your answers are readable. Do not use **red or green pens or pencils**.
- Please answer the exercises on the **exercise sheets**. If needed, also use the back sides of the exercise sheets.
- Answers on extra sheets can only be accepted if they are clearly marked with your name, your immatriculation number, and the **exercise number**.
- **Cross out** text that should not be considered in the evaluation.
- Students that try to cheat **do not pass** the exam.
- At the end of the exam, please return **all sheets together with the exercise sheets**.

Exercise 1 (Theoretical Foundations): **(3 + 3 + 4 = 10 points)**

Let $\varphi = \mathtt{q}(0, \mathtt{s}(0)) \wedge \forall X, Y \, (\mathtt{q}(X, Y) \rightarrow \mathtt{q}(\mathtt{s}(X), \mathtt{s}(Y)))$ and $\psi = \exists Z \, \mathtt{q}(\mathtt{s}(Z), \mathtt{s}(\mathtt{s}(Z)))$ be formulas over the signature (Σ, Δ) with $\Sigma = \Sigma_0 \cup \Sigma_1, \Sigma_0 = \{0\}, \Sigma_1 = \{\mathtt{s}\}$, and $\Delta = \Delta_2 = \{\mathtt{q}\}$.

a) Prove that $\varphi \models \psi$ by means of resolution.

Hint: First transform the formula $\varphi \wedge \neg\psi$ into an equivalent clause set.

b) Explicitly give a Herbrand model of the formula φ (i.e., specify a carrier and a meaning for all function and predicate symbols). You do not have to provide a proof for your answer.

c) Prove or disprove that input resolution is complete for arbitrary clause sets.

Exercise 2 (SLD tree): (9 points)

Consider the following Prolog program $\mathcal{P}$ which can be used to sort a list of numbers using the *bubblesort* algorithm:

```
bubble(L, R) :- swap(L, N), !, bubble(N, R).
bubble(L, L).
swap([A,B|L]), [B,A|L]) :- B < A.
swap([A|L], [A|N]) :- swap(L, N).
```

Please give a graphical representation of the SLD tree for the query `?- bubble([2, 1], X).` in the program $\mathcal{P}$.
Hint: As usual, you should treat $<$ as if it were defined by the infinitely many facts

```
0 < 1.
1 < 2.
0 < 2.
...
```

Exercise 3 (Fixpoint Semantics): **(3 + 3 = 6 points)**

Consider the following logic program $\mathcal{P}$ over the signature (Σ, Δ) with $\Sigma = \{\mathtt{0}, \mathtt{s}\}$ and $\Delta = \{\mathtt{gt}\}$.

```
gt(s(X), 0).
gt(s(X), s(Y)) :- gt(X, Y).
```

a) For each $n \in \mathbb{N}$ explicitly give $\underline{\text{trans}}_{\mathcal{P}}^{n}(\varnothing)$ in closed form, i.e., using a non-recursive definition.

b) Compute the set $\text{lfp}(\underline{\text{trans}}_{\mathcal{P}})$.

Exercise 4 (Definite Logic Programming): (10 points)

Implement the predicate `solve/1` in Prolog. This predicate can be used as a primitive SAT-solver for clause sets represented as lists of lists of literals. More precisely, a clause set is a list t of the form
$[[l_1^1, l_2^1, \dots, l_{k_1}^1], [l_1^2, l_2^2, \dots, l_{k_2}^2], \dots, [l_1^n, l_2^n, \dots, l_{k_n}^n]]$
where all l_i^j are of the form `pos(X)` or `neg(X)` for some Prolog variables `X`. The list t represents a set of clauses where `pos(X)` stands for the propositional variable X while `neg(X)` stands for its negation. A call `solve(`t`)` succeeds with a substitution satisfying the represented clause set t (by setting the variables to 1 or 0) if this set is satisfiable or fails if this set is unsatisfiable. If t does not represent a clause set as described above, then `solve(`t`)` may behave arbitrarily. You **must not use** any built-in predicates in this exercise. The following example calls to `solve/1` illustrate its definition:

- `?- solve([[pos(A),pos(B)],[neg(A),neg(B)]]).` has the two answer substitutions `A = 1, B = 0` and `A = 0, B = 1` (the order of the solutions is up to your implementation)
- `?- solve([[pos(A)],[neg(A)]]).` fails

Hint: In this representation, a clause is satisfied if it contains at least one literal of the form `pos(1)` *or* `neg(0)`*. Moreover, a clause set is satisfied if all its clauses are satisfied. It might be useful to implement this predicate in a way that the following example calls work as described below, although this is not mandatory.*

- `?- solve([[pos(1),pos(B)],[neg(1),neg(B)]]).` succeeds with the answer substitution `B = 0`
- `?- solve([[pos(1),pos(0)],[neg(1),neg(0)]]).` succeeds with the empty answer substitution

Exercise 5 (Arithmetic): (5 points)

Implement the predicate `binomial/3` in Prolog. A call of `binomial(`t_1`,`t_2`,`t_3`)` works as follows. If t_1 and t_2 are integers with $t_1 < t_2$ or at least one of t_1 or t_2 is negative, then it fails. If t_1 and t_2 are non-negative integers with $t_1 \geq t_2$, then t_3 is unified with the integer resulting from $\binom{t_1}{t_2}$. If t_1 or t_2 is no integer, `binomial/3` may behave arbitrarily.

Remember that the binomial coefficient $\binom{n}{k}$ for non-negative integers n and k with $n \geq k$ is defined as $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ with $0! = 1$.

The following example calls to `binomial/3` illustrate its definition:

- `?- binomial(-3,2,X).` fails
- `?- binomial(2,3,X).` fails
- `?- binomial(3,2,X).` succeeds with the answer substitution `X = 3`
- `?- binomial(3,2,1).` fails

Exercise 6 (Meta-Programming): (10 points)

Implement the predicate `map/2` in Prolog. A call of `map(`t_1`,`t_2`)` works as follows. If t_1 is a constant $f \in \Sigma_0$ and t_2 has the form $[a_1, \ldots, a_n]$, then the calls $f(a_1), \ldots, f(a_n)$ are executed. That means we assume that there is also a predicate symbol $f \in \Delta_1$ (with the same name as $f \in \Sigma_0$). Thus, `map(`f`,`$[a_1, \ldots, a_n]$`)` succeeds iff the query $f(a_1), \ldots, f(a_n)$ succeeds. If t_1 or t_2 are not of the form described above, `map/2` may behave arbitrarily.
For example, the query `?- map(foo,[a,b,c]).` is evaluated by executing the three calls `foo(a)`, `foo(b)` and `foo(c)`, while the query `?- map(foo,[]).` succeeds immediately.

Hint: You may use the built-in predicate `=../2`.

Exercise 7 (Constraint Logic Programming): (10 points)

A *magic square* is a matrix of dimension $n \times n$ containing all numbers from 1 to n^2 such that the sum of each row and of each column is $\frac{n(n^2+1)}{2}$. For instance, consider the following magic square of dimension 3×3:

$$\begin{pmatrix} 1 & 8 & 6 \\ 9 & 4 & 2 \\ 5 & 3 & 7 \end{pmatrix}$$

We represent such a square as a list of concatenated rows. For example, the above square would be represented as follows:

```
[1, 8, 6, 9, 4, 2, 5, 3, 7]
```

Implement a Prolog predicate `magic/1` such that the query `?- magic(L).` has exactly those lists `L` as answers that represent a magic square of dimension 3×3. Thus, for a correct implementation we get the following answers to the query (the order of the solutions depends on your implementation):

```
?- magic(L).
L = [1, 5, 9, 6, 7, 2, 8, 3, 4] ;
L = [1, 5, 9, 8, 3, 4, 6, 7, 2] ;
L = [1, 6, 8, 5, 7, 3, 9, 2, 4] ;

   ⋮
```

Hint: The query `?- magic(L).` *has more than 70 solutions.*
Hint: You may use constraint logic programming for your implementation, but you are not required to do so. Recall that the CLP library `clpfd` *contains predicates like* `all_different/1`, `label/1`, *the infix predicate* `ins/2`, ...

The following line is already given:

```
:- use_module(library(clpfd)).
```