Master Exam Version V4

First Name: ____________________________________________

Last Name: ____________________________________________

Immatriculation Number: __________________________________

Course of Studies (please mark exactly one):

○ SSE Master    ○ Other: ____________________________

<table>
<thead>
<tr>
<th>Exercise</th>
<th>Maximal Points</th>
<th>Achieved Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exercise 1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Exercise 2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Exercise 3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Exercise 4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Exercise 5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Exercise 6</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Exercise 7</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

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):  

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

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

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

b) Explicitly give a Herbrand model of the formula $\varphi$ (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):  

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

\[
\begin{align*}
\text{bubble}(L, R) & :\text{= swap}(L, N), !, \text{bubble}(N, R). \\
\text{bubble}(L, L). \\
\text{swap}([A,B|L]), [B,A|L]) & : B < A. \\
\text{swap}([A|L], [A|N]) & : \text{swap}(L, N).
\end{align*}
\]

Please give a graphical representation of the SLD tree for the query $?-\text{bubble}([2, 1], X)$ in the program $\mathcal{P}$.

*Hint:* As usual, you should treat $<$ as if it were defined by the infinitely many facts

\[
\begin{align*}
0 & < 1. \\
1 & < 2. \\
0 & < 2. \\
& \ldots
\end{align*}
\]
Exercise 3 (Fixpoint Semantics): (3 + 3 = 6 points)

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

$\text{gt}(s(X), 0)$.
$\text{gt}(s(X), s(Y)) :\neg \text{gt}(X, Y)$.

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

b) Compute the set lfp($\text{trans}_P$).
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

\[
[[l_1^1, l_2^1, \ldots, l_k^1], [l_1^2, l_2^2, \ldots, l_k^2], \ldots, [l_1^n, l_2^n, \ldots, l_k^n]]
\]

where all \( l_j^i \) are of the form \( \text{pos}(X) \) or \( \text{neg}(X) \) for some Prolog variables \( X \). The list \( t \) represents a set of clauses where \( \text{pos}(X) \) stands for the propositional variable \( X \) while \( \text{neg}(X) \) 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([[\text{pos}(A), \text{pos}(B)], [\text{neg}(A), \text{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([[\text{pos}(A)],[\text{neg}(A)]]).` fails

*Hint: In this representation, a clause is satisfied if it contains at least one literal of the form \( \text{pos}(1) \) or \( \text{neg}(0) \). 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([[\text{pos}(1), \text{pos}(B)], [\text{neg}(1), \text{neg}(B)]]).` succeeds with the answer substitution \( B = 0 \)
- `?- solve([[\text{pos}(1), \text{pos}(0)],[\text{neg}(1), \text{neg}(0)]]).` succeeds with the empty answer substitution
Exercise 5 (Arithmetic): (5 points)

Implement the predicate binomial/3 in Prolog. A call of binomial(t₁, t₂, t₃) works as follows. If t₁ and t₂ are integers with t₁ < t₂ or at least one of t₁ or t₂ is negative, then it fails. If t₁ and t₂ are non-negative integers with t₁ ≥ t₂, then t₃ is unified with the integer resulting from \( \binom{t₁}{t₂} \). If t₁ or t₂ 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 ≥ 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(t₁, t₂)` works as follows. If `t₁` is a constant `f ∈ Σ₀` and `t₂` has the form `[a₁, ..., aₙ]`, then the calls `f(a₁), ..., f(aₙ)` are executed. That means we assume that there is also a predicate symbol `f ∈ Δ₁` (with the same name as `f ∈ Σ₀`). Thus, `map(f, [a₁, ..., aₙ])` succeeds iff the query `f(a₁), ..., f(aₙ)` succeeds. If `t₁` or `t₂` 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 $3 \times 3$:

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

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 $3 \times 3$. 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)).`