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): (6 + 4 = 10 points)

Let

$$\varphi \;=\; \mathtt{p}(0) \;\wedge\; \exists X\, \neg\mathtt{p}(X) \;\wedge\; \forall Y\, (\, \mathtt{p}(Y) \rightarrow \mathtt{p}(\mathtt{s}(\mathtt{s}(Y)))\,) \quad \text{and}$$

$$\psi \;=\; \exists Z\, \forall U\, (\, \mathtt{p}(U) \rightarrow \mathtt{p}(\mathtt{s}(Z))\,)$$

be formulas over the signature (Σ, Δ) with $\Sigma = \Sigma_0 \cup \Sigma_1$, $\Sigma_0 = \{0\}$, $\Sigma_1 = \{\mathtt{s}\}$, and $\Delta = \Delta_1 = \{\mathtt{p}\}$.

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

Hint: First transform the formula $\varphi \wedge \neg\psi$ into a satisfiability-equivalent Skolem normal form, and then convert this Skolem normal form to 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.

Solution:

a)

$$\begin{array}{cl} & \varphi \wedge \neg\psi \\ \Leftrightarrow & \mathtt{p}(0) \;\wedge\; \exists X\, \neg\mathtt{p}(X) \;\wedge\; \forall Y\, (\, \mathtt{p}(Y) \rightarrow \mathtt{p}(\mathtt{s}(\mathtt{s}(Y)))\,) \;\wedge\; \neg\exists Z\, \forall U\, (\, \mathtt{p}(U) \rightarrow \mathtt{p}(\mathtt{s}(Z))\,) \\ \Leftrightarrow & \mathtt{p}(0) \;\wedge\; \exists X\, \neg\mathtt{p}(X) \;\wedge\; \forall Y\, (\, \neg\mathtt{p}(Y) \vee \mathtt{p}(\mathtt{s}(\mathtt{s}(Y)))\,) \;\wedge\; \forall Z\, \exists U\, \neg(\, \neg\mathtt{p}(U) \vee \mathtt{p}(\mathtt{s}(Z))\,) \\ \Leftrightarrow & \mathtt{p}(0) \;\wedge\; \exists X\, \neg\mathtt{p}(X) \;\wedge\; \forall Y\, (\, \neg\mathtt{p}(Y) \vee \mathtt{p}(\mathtt{s}(\mathtt{s}(Y)))\,) \;\wedge\; \forall Z\, \exists U\, (\, \mathtt{p}(U) \wedge \neg\mathtt{p}(\mathtt{s}(Z))\,) \\ \Leftrightarrow & \exists X\, \forall Y\, \forall Z\, \exists U\, (\, \mathtt{p}(0) \;\wedge\; \neg\mathtt{p}(X) \;\wedge\; (\, \neg\mathtt{p}(Y) \vee \mathtt{p}(\mathtt{s}(\mathtt{s}(Y)))\,) \;\wedge\; \mathtt{p}(U) \wedge \neg\mathtt{p}(\mathtt{s}(Z))\,) \end{array}$$

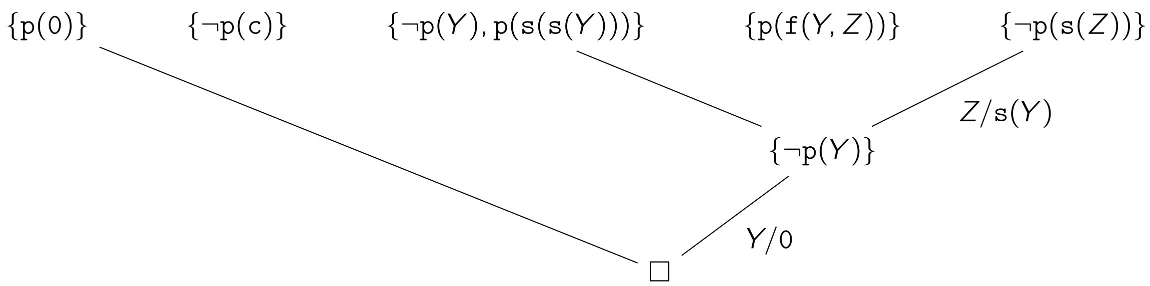
We obtain the following formula θ in Skolem normal form, which is satisfiability-equivalent to $\varphi \wedge \neg\psi$ (by dropping existential quantification, replacing X by $\mathtt{c}$ and replacing U by $\mathtt{f}(Y, Z)$).

$$\theta = \forall Y\, \forall Z\, (\, \mathtt{p}(0) \;\wedge\; \neg\mathtt{p}(\mathtt{c}) \;\wedge\; (\, \neg\mathtt{p}(Y) \vee \mathtt{p}(\mathtt{s}(\mathtt{s}(Y)))\,) \;\wedge\; \mathtt{p}(\mathtt{f}(Y, Z)) \wedge \neg\mathtt{p}(\mathtt{s}(Z))\,)$$

An equivalent clause set for the Skolem normal form θ of $\varphi \wedge \neg\psi$ is:

$$\{ \quad \{\mathtt{p}(0)\}, \{\neg\mathtt{p}(\mathtt{c})\}, \{\neg\mathtt{p}(Y), \mathtt{p}(\mathtt{s}(\mathtt{s}(Y)))\}, \{\mathtt{p}(\mathtt{f}(Y, Z))\}, \{\neg\mathtt{p}(\mathtt{s}(Z))\} \quad \}$$

We perform resolution on this clause set to show unsatisfiability of θ and hence also of $\varphi \wedge \neg\psi$.



Hence, we have proven $\varphi \models \psi$. □

b) We have $S \models \varphi$ for the Herbrand structure $S = (\mathcal{T}(\Sigma), \alpha)$ with $\alpha_0 = 0$, $\alpha_{\mathtt{s}}(t) = \mathtt{s}(t)$, and $\alpha_{\mathtt{p}} = \{\mathtt{s}^{2i}(0) \mid i \in \mathbb{N}\}$.

Exercise 2 (SLD tree): (9 points)

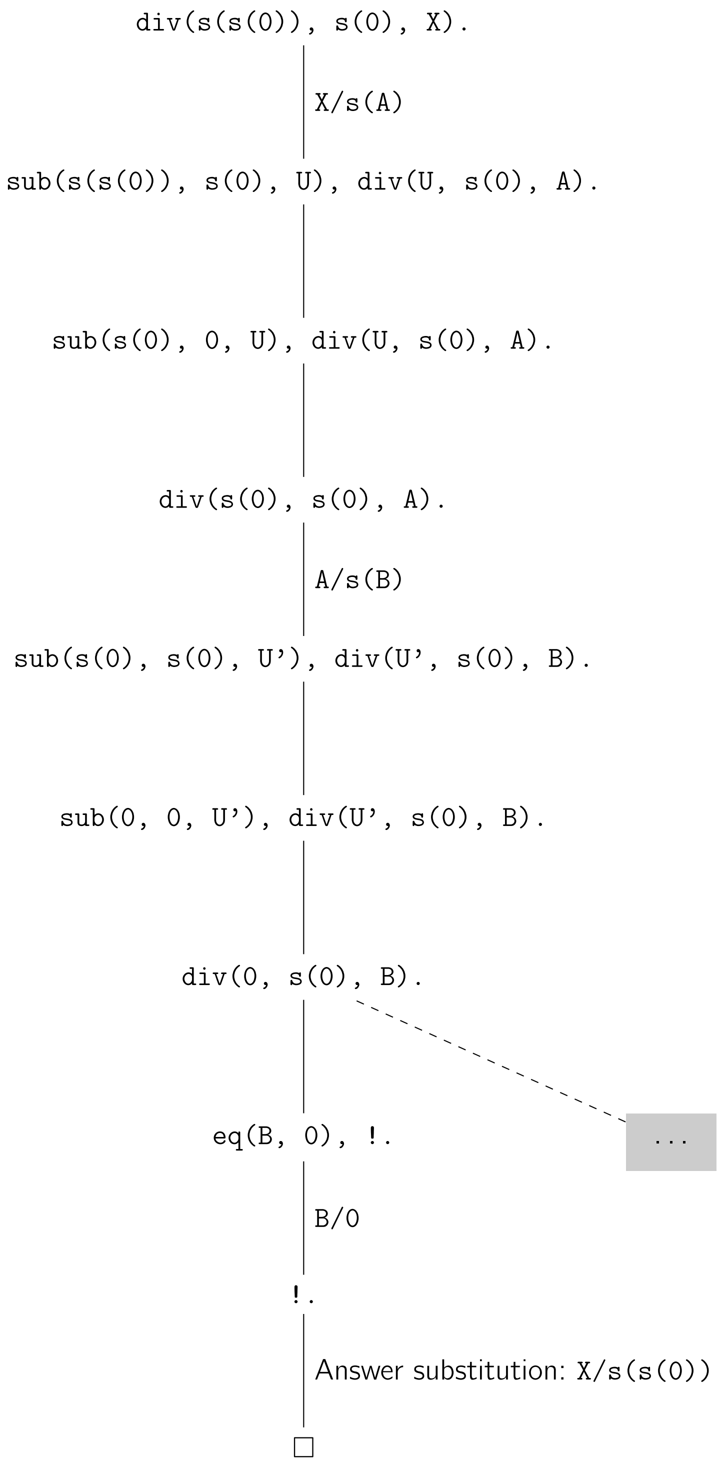
Consider the following Prolog program $\mathcal{P}$ which can be used to divide two numbers represented by terms built from 0 $\in \Sigma_0$ and s $\in \Sigma_1$ (i.e., $0 \mathrel{\hat{=}} 0, 1 \mathrel{\hat{=}} s(0), 2 \mathrel{\hat{=}} s(s(0)), \ldots$):

```
div(X, 0, Z) :- !, fail.
div(0, s(Y), Z) :- eq(Z, 0), !.
div(X, Y, s(Z)) :- sub(X, Y, U), div(U, Y, Z).
sub(0, Y, 0).
sub(s(X), 0, s(X)).
sub(s(X), s(Y), Z) :- sub(X, Y, Z).
eq(X, X).
```

Please give a graphical representation of the SLD tree for the query `?- div(s(s(0)), s(0), X).` in the program $\mathcal{P}$. Also give all answer substitutions explicitly.

Solution:

In this representation, the nodes and edges deleted by the cut are shown with a gray background and dashed edges, respectively.


div(s(s(0)), s(0), X).
X/s(A)
sub(s(s(0)), s(0), U), div(U, s(0), A).
sub(s(0), 0, U), div(U, s(0), A).
div(s(0), s(0), A).
A/s(B)
sub(s(0), s(0), U'), div(U', s(0), B).
sub(0, 0, U'), div(U', s(0), B).
div(0, s(0), B).
eq(B, 0), !.
...
B/0
!.
Answer substitution: X/s(s(0))
□

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

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

```
plus(X, 0, X).
plus(0, X, X).
plus(s(X), s(Y), s(s(Z))) :- plus(X, Y, Z).
```

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 $\text{lfp}(\underline{\text{trans}}_{\mathcal{P}})$.

Solution:

Let G be the set of all ground terms, i.e., $G = \{\mathtt{s}^i(0) \mid i \in \mathbb{N}\} = \mathcal{T}(\Sigma)$.

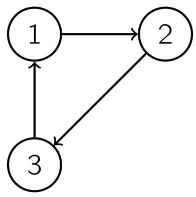
a)

$$
\begin{aligned}
\underline{\text{trans}}_{\mathcal{P}}^{0}(\varnothing) &= \varnothing \\
\underline{\text{trans}}_{\mathcal{P}}^{1}(\varnothing) &= \{\mathtt{plus}(t, 0, t), \mathtt{plus}(0, t, t) \mid t \in G\} \\
\underline{\text{trans}}_{\mathcal{P}}^{2}(\varnothing) &= \{\mathtt{plus}(t, 0, t), \mathtt{plus}(0, t, t), \mathtt{plus}(t, \mathtt{s}(0), \mathtt{s}(t)), \mathtt{plus}(\mathtt{s}(0), t, \mathtt{s}(t)) \mid t \in G\} \\
&\vdots \\
\underline{\text{trans}}_{\mathcal{P}}^{n}(\varnothing) &= \{\mathtt{plus}(t, \mathtt{s}^i(0), \mathtt{s}^i(t)), \mathtt{plus}(\mathtt{s}^i(0), t, \mathtt{s}^i(t)) \mid t \in G, 0 \le i < n\} \\
&= \{\mathtt{plus}(\mathtt{s}^i(t), \mathtt{s}^i(0), \mathtt{s}^{2i}(t)), \mathtt{plus}(\mathtt{s}^i(0), \mathtt{s}^i(t), \mathtt{s}^{2i}(t)) \mid t \in G, 0 \le i < n\}
\end{aligned}
$$

b) $\text{lfp}(\underline{\text{trans}}_{\mathcal{P}}) = \{\mathtt{plus}(\mathtt{s}^i(0), \mathtt{s}^j(0), \mathtt{s}^{i+j}(0)) \mid i, j \in \mathbb{N}\}$

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

A directed graph is a pair (V, E) where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. We represent such a graph by two lists where the first list contains the names of the nodes and the second list contains an element `e(`n_1`,`n_2`)` iff there is an edge from n_1 to n_2 in the represented graph. Both lists do not contain duplicates. For example, consider the following graph $(\{1, 2, 3\}, \{(1, 2), (3, 1), (2, 3)\})$:



This graph is represented by the two lists `[1,2,3]` and `[e(1,2),e(3,1),e(2,3)]`.

Implement the predicate `hamiltonian/2` in Prolog. A call `hamiltonian(`t_1`,`t_2`)` works as follows. If both t_1 and t_2 are finite lists representing a directed graph as described above, the call succeeds iff there is a hamiltonian cycle in the represented graph. If t_1 and t_2 are not of the form described above, then `hamiltonian(`t_1`,`t_2`)` may behave arbitrarily. You **must not use** any built-in predicates in this exercise. Note that meta-programming (e.g., using variables without a surrounding predicate) also uses built-in predicates (implicitly) and is, thus, not allowed in this exercise.

A hamiltonian cycle exists in a directed graph iff there is a path through the graph starting and ending in the same node (thus, visiting this node exactly twice) and visiting all other nodes exactly once. The empty graph has no hamiltonian cycle. The following example calls to `hamiltonian/2` illustrate its definition:

- `?- hamiltonian([1,2,3],[e(1,2),e(3,1),e(2,3)]).` succeeds with the empty answer substitution
- `?- hamiltonian([1,2,3],[e(1,2),e(2,1),e(1,3),e(3,1)]).` fails

Hint: Use a helper predicate `tour/4` *with the following four arguments: (1) the node where you currently are, (2) the list of nodes yet to visit, (3) the node you have to reach in the end, and (4) the list of edges in the graph. Moreover, it can be helpful to define a predicate* `choice/3` *for the non-deterministic choice of the next node to visit. Here,* `choice(`t_1, t_2, t_3`)` *is true if* t_1 *is a list containing* t_2 *and* t_3 *is the list* t_1 *where the element* t_2 *was deleted.*

Solution:

```
hamiltonian([N|NS],E) :- tour(N,NS,N,E).

tour(L,[],T,E) :- edge(L,T,E).
tour(L,[X|XS],T,E) :- choice([X|XS],Y,YS),
                      edge(L,Y,E),
                      tour(Y,YS,T,E).

choice([X|XS],X,XS).
choice([X|XS],Y,[X|YS]) :- choice(XS,Y,YS).

edge(L,T,[e(L,T)|_]).
edge(L,T,[_|E]) :- edge(L,T,E).
```

Exercise 5 (Arithmetic): (5 points)

Implement the predicate `mean/2` in Prolog. A call of `mean(`t_1`,`t_2`)` works as follows. If t_1 is a finite non-empty list of integers, then t_2 is unified with the *rounded mean value* of all numbers in t_1. If t_1 is the empty list, `mean/2` fails. If t_1 is not a finite list of integers, `mean/2` may behave arbitrarily.

We define the *rounded mean value* of $n > 0$ integers $a_1, \ldots, a_n$ as $\left\lfloor \frac{\sum_{i=1}^n a_i}{n} \right\rfloor$ if $\frac{\sum_{i=1}^n a_i}{n}$ is positive and $\left\lceil \frac{\sum_{i=1}^n a_i}{n} \right\rceil$ if $\frac{\sum_{i=1}^n a_i}{n}$ is negative.

The following example calls to `mean/2` illustrate its definition:

- `?- mean([],X).` fails
- `?- mean([1,2,3],1).` fails
- `?- mean([1,2,3],X).` succeeds with the answer substitution `X = 2`
- `?- mean([1,2],1).` succeeds with the empty answer substitution
- `?- mean([1,-2],X).` succeeds with the answer substitution `X = 0`

Hint: In Prolog, the term `X // Y` *can be used to compute* $\left\lfloor \frac{X}{Y} \right\rfloor$ *if* $\frac{X}{Y}$ *is positive and* $\left\lceil \frac{X}{Y} \right\rceil$ *if* $\frac{X}{Y}$ *is negative.*

Solution:

```
mean(List,Mean) :- sumMean(0,0,List,Mean).

sumMean(Length,Sum,[],Mean) :- Length > 0,
                               Mean is Sum // Length.
sumMean(Length,Sum,[X|XS],Mean) :- NextLength is Length + 1,
                                   NextSum is Sum + X,
                                   sumMean(NextLength,NextSum,XS,Mean).
```

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

Implement the predicate `fold/4` in Prolog. A call of `fold(`t_1`,`t_2`,`t_3`,`t_4`)` works as follows. If t_1 is a constant $f \in \Sigma_0$ and t_3 has the form $[a_1, \ldots, a_n]$ with $n > 0$ (i.e. t_3 is a finite non-empty list), then the calls $f(t_2, a_1, X_1), f(X_1, a_2, X_2), \ldots, f(X_{n-2}, a_{n-1}, X_{n-1}), f(X_{n-1}, a_n, t_4)$ with fresh Prolog variables $X_1, \ldots, X_{n-1}$ are executed. That means we assume that there is also a predicate symbol $f \in \Delta_3$ (with the same name as $f \in \Sigma_0$). Thus, `fold(`f`,`t_2`,`$[a_1, \ldots, a_n]$`,`t_4`)` succeeds iff the query $f(t_2, a_1, X_1), f(X_1, a_2, X_2) \ldots, f(X_{n-2}, a_{n-1}, X_{n-1}), f(X_{n-1}, a_n, t_4)$ succeeds. If t_1 or t_3 are not of the form described above, `fold/4` may behave arbitrarily.
For example, the query `?- fold(foo,d,[a,b,c],X).` is evaluated by executing the three calls `foo(d,a,X1)`, `foo(X1,b,X2)` and `foo(X2,c,X)`. The query `?- fold(foo,b,[a],r).` is evaluated by the call `foo(b,a,r)`.

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

Solution:

```
fold(F,A,[E],Z) :- !,
                   C =.. [F,A,E,Z],
                   call(C).
fold(F,A,[E|ES],Z) :- C =.. [F,A,E,X],
                      call(C),
                      fold(F,X,ES,Z).
```

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

A *heterosquare* is a matrix of dimension $n \times n$ containing all numbers from 1 to n^2 such that the respective sums of each row, of each column, and of each of the two diagonals are pairwise different. For instance, consider the following matrix of dimension 3×3:

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

Here the first row sums up to 7, the second row sums up to 18, the third row sums up to 20, the first column sums up to 9, the second column sums up to 15, the third column sums up to 21, the top-left to bottom-right diagonal sums up to 17, and the bottom-left to top-right diagonal sums up to 16. Since all these eight values are pairwise different, this matrix indeed is a heterosquare.

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

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

Implement a Prolog predicate `hsquare/1` such that the query `?- hsquare(L).` has exactly those lists `L` as answers that represent a heterosquare 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):

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

   ⋮
```

Hint: The query `?- hsquare(L).` *has more than 20,000 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)).
```

Solution:

```
hsquare(Sq) :-
  Sq = [E1, E2, E3, E4, E5, E6, E7, E8, E9],
  Sq ins 1..9,
  all_different(Sq),
  Row1 #= E1 + E2 + E3,
  Row2 #= E4 + E5 + E6,
  Row3 #= E7 + E8 + E9,
  Col1 #= E1 + E4 + E7,
  Col2 #= E2 + E5 + E8,
  Col3 #= E3 + E6 + E9,
  Diag1 #= E1 + E5 + E9,
  Diag2 #= E3 + E5 + E7,
  all_different([Row1, Row2, Row3, Col1, Col2, Col3, Diag1, Diag2]),
  label(Sq).
```