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.

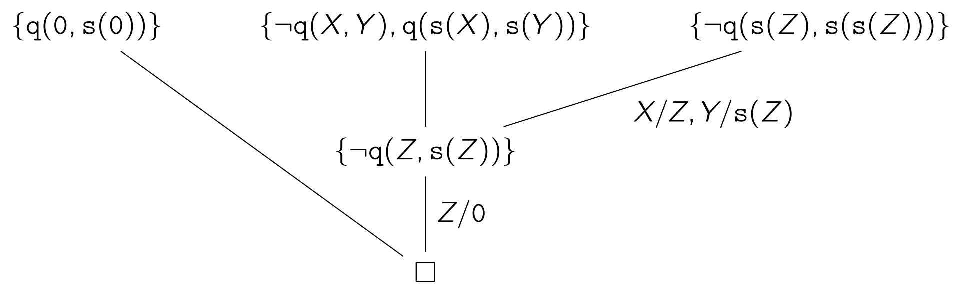
c) Prove or disprove that input resolution is complete for arbitrary clause sets.

Solution:

a)

$$\begin{aligned}
\varphi \wedge \neg\psi &= \mathtt{q}(0, \mathtt{s}(0)) \wedge \forall X, Y\, (\mathtt{q}(X, Y) \rightarrow \mathtt{q}(\mathtt{s}(X), \mathtt{s}(Y))) \wedge \neg\exists Z\, \mathtt{q}(\mathtt{s}(Z), \mathtt{s}(\mathtt{s}(Z)))\\
&= \mathtt{q}(0, \mathtt{s}(0)) \wedge \forall X, Y\, (\neg\mathtt{q}(X, Y) \vee \mathtt{q}(\mathtt{s}(X), \mathtt{s}(Y))) \wedge \neg\exists Z\, \mathtt{q}(\mathtt{s}(Z), \mathtt{s}(\mathtt{s}(Z)))\\
&= \mathtt{q}(0, \mathtt{s}(0)) \wedge \forall X, Y\, (\neg\mathtt{q}(X, Y) \vee \mathtt{q}(\mathtt{s}(X), \mathtt{s}(Y))) \wedge \forall Z\, \neg\mathtt{q}(\mathtt{s}(Z), \mathtt{s}(\mathtt{s}(Z)))\\
&= \forall X, Y, Z\, (\mathtt{q}(0, \mathtt{s}(0)) \wedge (\neg\mathtt{q}(X, Y) \vee \mathtt{q}(\mathtt{s}(X), \mathtt{s}(Y))) \wedge \neg\mathtt{q}(\mathtt{s}(Z), \mathtt{s}(\mathtt{s}(Z))))
\end{aligned}$$

Thus, the equivalent clause set for $\varphi \wedge \neg\psi$ is
$\{\{\mathtt{q}(0, \mathtt{s}(0))\}, \{\neg\mathtt{q}(X, Y), \mathtt{q}(\mathtt{s}(X), \mathtt{s}(Y))\}, \{\neg\mathtt{q}(\mathtt{s}(Z), \mathtt{s}(\mathtt{s}(Z)))\}\}$.
We perform resolution on this clause set to show $\varphi \models \psi$.

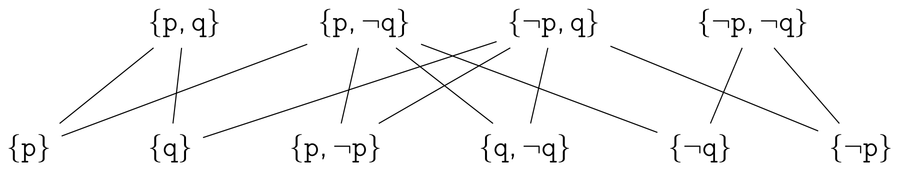


Hence, we have proven $\varphi \models \psi$.

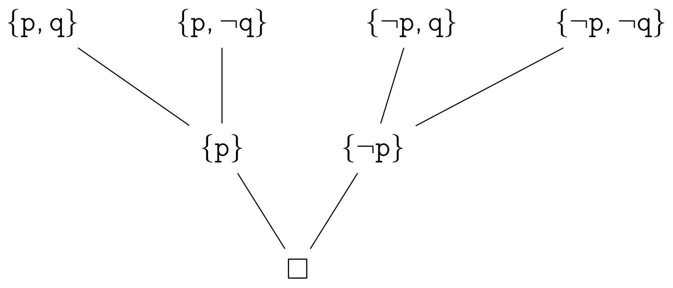
$\square$

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 $\alpha_{\mathtt{q}} = \{(\mathtt{s}^i(0), \mathtt{s}^{i+1}(0)) \mid i \geq 0\}$.

c) Consider the clause set $\{\{\mathtt{p}, \mathtt{q}\}, \{\mathtt{p}, \neg\mathtt{q}\}, \{\neg\mathtt{p}, \mathtt{q}\}, \{\neg\mathtt{p}, \neg\mathtt{q}\}\}$. Using input resolution, we obtain the following resolution proof.



All further input resolution steps lead to already existing clauses and the empty clause cannot be derived. However, using full resolution, we obtain the following derivation of the empty clause.



Hence, input resolution is incomplete.

□

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:

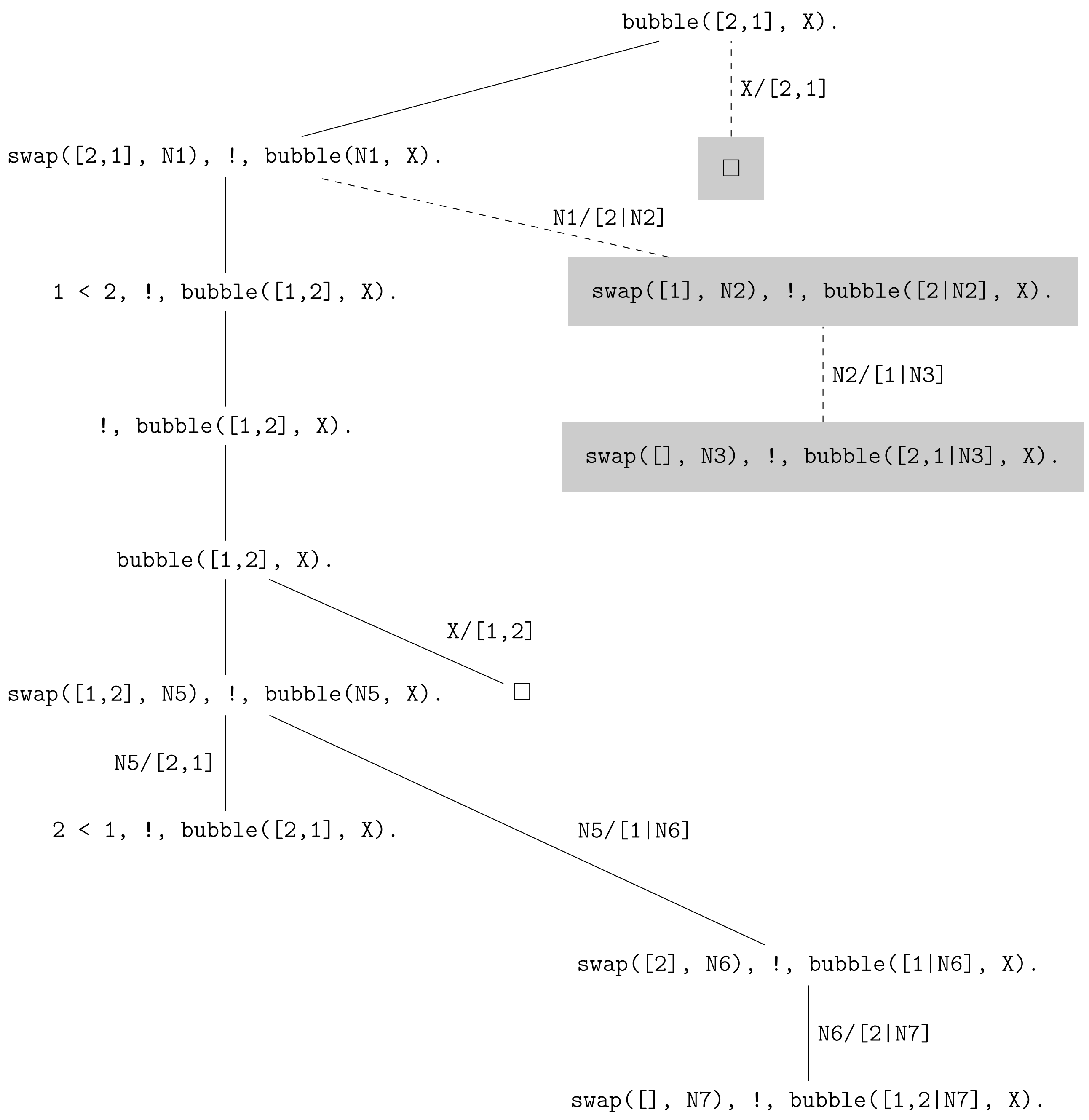
```
bubble(L, R) :- swap(L, N), !, bubble(N, R).
bubble(L, L).
swap([A,B|L]), [B,A|L]) :- B < A.
swap([A|L], [A|N]) :- swap(L, N).
```

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.
...
```

Solution:

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


bubble([2,1], X).
X/[2,1]
□
swap([2,1], N1), !, bubble(N1, X).
N1/[2|N2]
1 < 2, !, bubble([1,2], X).
swap([1], N2), !, bubble([2|N2], X).
N2/[1|N3]
!, bubble([1,2], X).
swap([], N3), !, bubble([2,1|N3], X).
bubble([1,2], X).
X/[1,2]
□
swap([1,2], N5), !, bubble(N5, X).
N5/[2,1]
2 < 1, !, bubble([2,1], X).
N5/[1|N6]
swap([2], N6), !, bubble([1|N6], X).
N6/[2|N7]
swap([], N7), !, bubble([1,2|N7], X).

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 $\underline{\text{trans}}^n_{\mathcal{P}}(\varnothing)$ 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., $G = \{\mathtt{s}^i(\mathtt{0}) \mid i \in \mathbb{N}\}$.

a)

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

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

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
$[[l^1_1, l^1_2, \ldots, l^1_{k_1}], [l^2_1, l^2_2, \ldots, l^2_{k_2}], \ldots, [l^n_1, l^n_2, \ldots, l^n_{k_n}]]$
where all l^j_i 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

Solution:

```
solve([]).
solve([C|CS]) :- solveClause(C), solve(CS).

solveClause([pos(1)|_]).
solveClause([neg(0)|_]).
solveClause([_|XS]) :- solveClause(XS).
```

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

Solution:

```
binomial(X,Y,Z) :- Y >= 0,
                   X >= Y,
                   factorial(X,XF),
                   factorial(Y,YF),
                   D is X - Y,
                   factorial(D,DF),
                   Z is (XF // (DF * YF)).

factorial(0,1) :- !.
factorial(N,F) :- N1 is N - 1,
                  factorial(N1,F1),
                  F is F1 * N.
```

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 $[a_1, \dots, a_n]$, then the calls $f(a_1), \dots, f(a_n)$ 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`,`$[a_1, \dots, a_n]$`)` succeeds iff the query $f(a_1), \dots, f(a_n)$ 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`.

Solution:

```
map(_,[]).
map(P,[X|XS]) :- G =.. [P,X],
                 call(G),
                 map(P,XS).
```

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)).
```

Solution:

```
magic(Sq) :-
  Sq = [E1, E2, E3, E4, E5, E6, E7, E8, E9],
  Sq ins 1..9,
  all_different(Sq),
  E1 + E2 + E3 #= 15,
  E4 + E5 + E6 #= 15,
  E7 + E8 + E9 #= 15,
  E1 + E4 + E7 #= 15,
  E2 + E5 + E8 #= 15,
  E3 + E6 + E9 #= 15,
  label(Sq).
```