**Kiel University**
Department of Computer Science
Programming Languages and Compiler Construction

Priv.-Doz. Dr. Frank Huch, Sandra Dylus, B.Sc. Jan-Hendrik Matthes, Rajib Chandra Das

Institut für Informatik

# 13. Exercise Sheet on „Advanced Programming (pre-masters)“

Logic Programming 3

Submit by Monday, 08. July 2019 - 14:00

---

## Exercise 1 - SLD Resolution 1

0 Points

Consider the following definition of the predicate `nth(N, L, E)`, which is fulfilled if the element `E` is at the index position `N` of the list `L`. Indexing is done using Peano numbers and starts at `o`.

```
1 nth(o,    [E|_], E).
2 nth(s(N), [_|L], E) :- nth(N, L, E).
```

Prove the following request using SLD resolution and specify the SLD tree and all intermediate substitutions and solutions.

```
1 ?- nth(X, [1, 2, 1], 1).
```

## Exercise 2 - Resolution with Cuts

0 Points

Consider the following predicate `rev` for reversing a list.

```
1 rev([],     []).
2 rev([X|Xs], Zs) :- app(Ys, [X], Zs), rev(Xs, Ys), !.
3
4 app([],     Ys, Ys).
5 app([X|Xs], Ys, [X|Zs]) :- app(Xs, Ys, Zs).
```

1. For the query

   ```
   1 ?- rev(Xs, [2, 1]).
   ```

   specify the corresponding SLD tree and prove the query by means of the evaluation strategy of Prolog. Also include failed derivations.

2. What would happen if

   - the cut would be omitted?
   - additionally, the second `rev`-rule would be modified as follows.

     ```
     1 rev([X|Xs], Zs) :- rev(Xs, Ys), app(Ys, [X], Zs).
     ```

   Explain the behaviour for both cases.

## Exercise 3 - SLD Resolution 2

2 Points

1. Consider the following predicate for concatenating two lists

```
append([],    L, L).
append([E|R], L, [E|RL]) :- append(R, L, RL).
```

   and the query

```
?- append(X, Y, [1, 2]).
```

   For the query, specify the SLD tree that is created by the SLD resolution from the lecture.

2. Consider the following predicate for being an element of a list

```
member(E, [X|_])  :- E = X.
member(E, [_|Xs]) :- member(E, Xs).
```

   and the query

```
?- member(E, [1, 2]).
```

   For the query, specify the SLD tree that is created by the SLD resolution from the lecture.

## Exercise 4 - Resolution

2 Points

1. Consider the following predicate for deleting an element from a list.

```
delete(_, [],     []).
delete(X, [Y|Xs], Xs)     :- X = Y.
delete(X, [Y|Xs], [Y|Ys]) :- delete(X, Xs, Ys).
```

   Prove the query

```
?- delete(1, [1, 0+1], Xs).
```

   by means of Prolog's evaluation strategy and specify all intermediate results and solutions.

   You can either display your solution as an SLD tree and specify the search order used by Prolog or note the search sequentially (as in the lecture).

2. Modify the above program so that

   - only the first occurrence of the first parameter is removed from the list.
   - all occurrences of the first parameter are removed from the list.

## Exercise 5 - Negation as Failure

2 Points

Define the following predicates for lists by using negation as failure.

- `nodup(Xs)` checks whether a list contains duplicates.
- `neq(X, Y)` holds if `X` and `Y` are not unifiable.
- `remove(X, Xs, Ys)` holds if removing all occurrences of `X` from the list `Xs` is equal to the list `Ys`.
- `nub(Xs, Ys)` holds if `Ys` is the list `Xs` without duplicates.

**Exercise 6 - Arithmetic** 2 Points

1. Define predicates `toPeano` and `fromPeano` to convert natural numbers into Peano numbers and vice versa.
2. We again consider the binary representation of numbers from the first logic programming exercise sheet.
   - Define a predicate `intToBin`, which can be used to convert a natural number into a binary number.
   - Define a predicate `binToInt`, which can be used to convert a binary number into a natural number.
3. The predicate `is` evaluates a given expression. In case of the evaluation of a division by zero, the predicate `is` yields a runtime error. Define a predicate `eval`, which avoids this error, but results in a failing computation (`false`).

   Start your program with the following rule.

```
1 eval(E, R) :- R is E.
```

   Add further rules for the predicate `eval`, such that the case of devision by zero is detected and the computation fails, instead of crashing with an error. For the solution it is necessary to use cuts and inequality (`\=`). Check your implementation with the following goals.

```
1 ?- eval(3/0, R).
2 ?- eval(3/0+5, R).
3 ?- eval(3/(2-2), R).
4 ?- eval(3/(4-4)+5, R).
```