Exercise Session Informatik III

5. Prolog Tutorial

Equality Operators

```
| ?- X is 2 + 3, X = 5.
X = 5 ? ;
no
```

- **X** is instantiated to **`2 + 3`** by the first statement. In the second statement the operator **=** tries to unify **X**. As **X** is already instantiated this is the same as an equality check.

```
| ?- X is 2 + 3, X == 5.
X = 5 ? ;
no
```

- **X** is instantiated to **`2 + 3`** by the first statement. In the second statement the operator **==** checks for equality of its arguments.

Equality Operators

```
| ?- X = fred, X = 6.
no
```

- In the first statement **X** is unified to **`fred`**. The second statement tries to unify X again, which fails of course.

```
| ?- X = john, Y = X.
X = john,
Y = john ? ;
no
```

- The first statement unifies **X** to **`john`**. The second statement tries to unify Y. This succeeds if **X** is **`john`** and **Y** is **`john`**.

Trace

```
append([], Ys, Ys).
append([X | Xs], Ys, [X | Zs]) :- append(Xs, Ys, Zs).

| ?- append([1, 2], [3], X).
      1  1  Call: append([1,2],[3],_214) ?
      2  2  Call: append([2],[3],_725) ?
      3  3  Call: append([],[3],_1129) ?
      3  3  Exit: append([],[3],[3]) ?
      2  2  Exit: append([2],[3],[2,3]) ?
      1  1  Exit: append([1,2],[3],[1,2,3]) ?
X = [1,2,3] ?
yes
```

append/3

```
| ?- append(X, Y, [1, 2, 3]).
      1  1  Call: append(_158,_178,[1,2,3]) ?
?     1  1  Exit: append([],[1,2,3],[1,2,3]) ?

X = [],
Y = [1,2,3] ? ;

      1  1  Redo: append([],[1,2,3],[1,2,3]) ?
      2  2  Call: append(_744,_178,[2,3]) ?
?     2  2  Exit: append([],[2,3],[2,3]) ?
?     1  1  Exit: append([1],[2,3],[1,2,3]) ?

X = [1],
Y = [2,3] ? ;
```

append/3

```
      1  1  Redo: append([1],[2,3],[1,2,3]) ?
      2  2  Redo: append([],[2,3],[2,3]) ?
      3  3  Call: append(_1148,_307,[3]) ?
?     3  3  Exit: append([],[3],[3]) ?
?     2  2  Exit: append([2],[3],[2,3]) ?
?     1  1  Exit: append([1,2],[3],[1,2,3]) ?

X = [1,2],
Y = [3] ? ;
```

append/3

```
      1  1  Redo: append([1,2],[3],[1,2,3]) ?
      2  2  Redo: append([2],[3],[2,3]) ?
      3  3  Redo: append([],[3],[3]) ?
      4  4  Call: append(_1551,_178,[]) ?
?     4  4  Exit: append([],[],[]) ?
?     3  3  Exit: append([3],[],[3]) ?
?     2  2  Exit: append([2,3],[],[2,3]) ?
?     1  1  Exit: append([1,2,3],[],[1,2,3]) ?

X = [1,2,3],
Y = [] ? ;
```

append/3

```
      1  1  Redo: append([1,2,3],[],[1,2,3]) ?
      2  2  Redo: append([2,3],[],[2,3]) ?
      3  3  Redo: append([3],[],[3]) ?
      4  4  Redo: append([],[],[]) ?
      4  4  Fail: append(_1551,_178,[]) ?
      3  3  Fail: append(_1148,_178,[3]) ?
      2  2  Fail: append(_744,_178,[2,3]) ?
      1  1  Fail: append(_158,_178,[1,2,3]) ?

no
```

count/2 and reverse/2

```
count([], 0).
count([X | Xs], N) :- count(Xs, Res), N is Res + 1.

reverse([], []).
reverse([X | Xs], Zs) :-
   reverse(Xs, Ys), append(Ys, [X], Zs).
```

listLength/0

```
listLength :-
   read(ListIn), count(ListIn, Len), write('Length = '),
   write(Len), nl.

| ?- listLength.
|: [1, 2, 3].
Length = 3

yes
| ?-  listLength.
|: 1 2 3.
{SYNTAX ERROR: read(_67) - in line 213 (within 213-214}}
** operator expected after expression **
1
** here **
2 3 .
```

sumAndMin/3

```
min(nil, Y, Y).
min(X, nil, X).
min(X, Y, X) :- X =< Y.
min(X, Y, Y) :- X > Y.

sumAndMin([], 0, nil).
sumAndMin([X | Xs], Sum, Min) :-
   sumAndMin(Xs, CurSum, CurMin), Sum is CurSum + X,
   min(X, CurMin, Min).
```

That's all folks!

Bis nächste Woche:
- Weiter mit Prolog
- SML Testat Übung Abgabe