CPL Exercises
PROLOG I

Prof. Dr. Ulrike Pado

1: Append in Prolog I

In Prolog, one list (written as [a,b,c] or [] for the empty list) is appended to another by the following code:

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

Analyse the code by answering the following questions. You may use `trace`.

What is the base case for recursion?
`acc_append([], Ys, Ys).`

Which of the variables accumulates the result?
Zs

What value should this variable therefore have for the initial call?
An unbound variable.

What happens when the non-base case rule is applied? Where is the new call with the smaller argument that allows recursion to terminate?
When the second rule is applied, the head-rest notation for the list arguments makes the head and tail of the first list available for processing.
A crucial part of the appending work already happens on the left-hand side while matching the arguments of acc_append – note how X, the head of the first list argument, is also used as the head of the third list argument in the head-rest notation. Unification will accumulate all these list heads from the different calls to the second clause into the result list (third argument) once the base case is reached and Ys is unified to Zs (the tail of the result list)
The argument reduction happens on the right-hand side of the rule, where the recursive call is made with the tail of the first list argument, and Ys and Zs unchanged. The fact that the first list argument is reduced by one element means that the base case will be reached eventually (namely, calling acc_append with an empty list as first element).