Hochschule **für Technik Stuttgart**

## 1: Append in Prolog II

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).
```

What happens if you call acc_append „in reverse“ – giving variables for the two input lists and a result list for the accumulator/result variable? Why? Trace the Prolog query and show the search tree for acc_append(X, Y, [1,2]).

## 2: List-reverse in Prolog

Here is the LISP code for list reverse using an accumulator. Re-write it in Prolog.

```
(define list-reverse-aux (lambda (L A)
              (if (null? L) A
                  (list-reverse-aux (cdr L) (cons (car L) A)))))

(define list-reverse (lambda (L) (list-reverse-aux L '())))
```

Hint 1: There is no cons predicate; use the [Head|Rest] notation to create a new list.

Hint 2: If you want to see the result of the list reversal, your query will have  to contain an unbound variable that takes on the value of A eventually (in addition to the actual accumulator).

## 3: CFGs and DCGs

Expand the sample DCG to cover the following grammar (from Lecture "Syntax and Semantics", slide 5):

```
• Expr ==> Term | Term '+' Expr
  Term ==> Factor | Factor '*' Term
  Factor ==> '0' | ... | '9' | '(' Expr ')
```