Letzte/Diese Vorlesung

...	Letzte Woche

Problemklassen	Diese Woche
Meta-Evaluator	

Problemklassen

Wir haben bis jetzt 2 Problemklassen gesehen:

Sequenzen Diese werden mit *einfacher* Rekursion oder iteration bearbeitet.

Baumstrukturen Diese werden mit *mehrfacher* Rekursion bearbeitet.

Beispiele:

Sequenzen `!, reverse, append, mapcar, ...`

Baumstrukturen `pre-order, post-order, traverse-ttree...`

Problemklassen - sequenzen

Beispiele:

Konstruiere eine neue Liste wo alle Zahlen einer Liste mit 1 erhöht sind:

```
(defun add-1-to-integers (l)
  (cond ((null l) nil)
        (t (cons (cond ((numberp (first l)) (1+ (first l)))
                       (t (first l)))
                 (add-1-to-integers (rest l))))))
```

Drehe eine Liste um

```
(defun rev (l)
  (cond ((null l) nil)
        (t (nconc (rev (rest l))
                  (list (first l))))))
```

Problemklassen - baumstrukturen

Man kann Lisplisten als binäre Bäume sehen. Man kann dabei diese Bäume traversieren, und dabei die Blätter von rechts nach links (pre-order) oder von links nach rechts (post-order) ausgeben lassen

```
(defun pre-order (bb)
  (cond ((null bb) nil)
        ((atom bb) (print bb))
        (t (pre-order (car bb))
           (pre-order (cdr bb)))))
(defun post-order (bb)
  (cond ((null bb) nil)
        ((atom bb) (print bb))
        (t (pre-order (cdr bb))
           (pre-order (car bb)))))
```

Problemklassen - *-order

Nun, die beide Funktionen sind fast „equal". Können wir das „besser" machen?

```
(defun *-order (bb f1 f2 action)
  (cond ((null bb) nil)
        ((atom bb) (funcall action bb))
        (t (*-order (funcall f1 bb) f1 f2 action)
           (*-order (funcall f2 bb) f1 f2 action))))

(defun pre-order  (bb) (*-order bb #'car #'cdr #'print))
(defun post-order (bb) (*-order bb #'cdr #'car #'print))
;; alternativ
(defmacro pre-order  (bb) `(*-order ,bb #'car #'cdr #'print))
(defmacro post-order (bb) `(*-order ,bb #'cdr #'car #'print))
```