Intelligente Suchverfahren

mit Scheme

KI mit Scheme

- In diesem Kurs werden auch Bereiche von *Algorithmen und Datenstrukturen* behandelt.
- Die Konzeption für den Kurs des erweiterten Anforderungsbereichs setzt auf die Anwendung des *funktionalen Paradigmas*.
- Dazu passend die Wahl einer funktionalen Programmiersprache wie Scheme (Version Racket: *racket-lang.org*) bzw Haskell.

KI mit Scheme

Listen sind die grundlegende Datenstruktur bei Scheme, runde Klammern das Strukturelement

- Listen:

```
(define eineListe '(1 2 3 4 5) )
```

- Unterschied zu Funktionsdefinition und Funktionsaufruf:

```
(define (quadrat x)
    (* x x) )

(quadrat 5)
25
```

Rekursion

- Viele Probleme –gerade bei Suchverfahren– lassen sich rekursiv leichter beschreiben als iterativ.
- Ein einfaches Beispiel:

```
(define (entferne element liste)
 (cond
  ((null? liste) liste)
  ((equal? element (first liste)) (rest liste))
  (else
   (cons (first liste)
         (entferne element (rest liste))))))
```

Rekursion

- Die Standard-Wiederholungsstruktur bei Scheme ist die Rekursion.
- Sie ist (*wie üblich*) in der Rekursionstiefe begrenzt -> „stack overflow“
- Scheme erkennt aber Endrekursion und legt dann keine Aufrufe auf dem Stack ab.

call by value

- Funktionsaufrufe erfolgen bei Scheme grundsätzlich mit call by value.
- Funktionen kapseln grundsätzlich, sie bekommen Daten (Objekte, Funktionen ...) übergeben und geben nach der Auswertung ihr Ergebnis zurück.

Übertragbarkeit

- Grundsätzlich gilt jedoch, dass funktionale Lösungen von Algorithmen bei anderen Sprachen (wie z.B. Python) sehr ähnlich den Scheme-Lösungen sind.
- Insbesondere sollte die Modularisierung und Beachtung der Kohärenz bei beiden Sprachen zu vergleichbaren Lösungen führen.