Struktur und Interpretation von Computerprogrammen

# Konstruktion von Abstraktionen mit Prozeduren

### 1.1.1 Ausdrücke

Eingabe eines *Ausdrucks*

Ausgabe einer *Auswertung (Evaluation)*

*Beispiel:*

5  
*5*

(+ 137 249)  
486

Eine *Kombination* ist eine *Liste von Ausdrücken* (z.B. (+ 2.7 10),  
wobei das '+' dabei der *Operator* ist, die anderen Elemente die *Operanden*.

Dabei benutzt Scheme die Präfix-Notation (Operator steht am Anfang: Präfix). Dies bietet den Vorteil, dass Ausdrücke mit Hilfe einer Baumstruktur abgebildet werden. Es gibt außerdem keine Doppeldeutigkeiten von Ausdrücken.

### 1.1.2 Namen und Umgebungen

Definition von Variablen:

(define groesse 2)

Der Speicher, in dem diese Variablen gespeichert werden, heißt *globale Umgebung.*

### 1.1.3 Auswertungen von Kombinationen

Kombinationen werden mit Hilfe eines Baumes ausgewertet (zur Veranschaulichung).

Kombinationen werden solange ausgewertet, bis nur noch Elementarfälle übrig bleiben.

Ausnahmen bestätigen die Regel:

(define x 5) ist KEINE Kombination

### 1.1.4 Zusammengesetzte Prozeduren

Definitionen, die Namen mit Werten verknüpfen, sind ein begrenztes Mittel zur Abstraktion--> Prozeduren

Definition von Prozeduren:  
(define (quadrat x) (\* x x))

Dabei ist x eine lokale Variable; sie kann an anderer Stelle wiederverwendet werden (nicht in der globalen Umgebung)

### 1.1.5 Das Substitutionsmodell für Prozeduranwendungen

Der Interpreter wertet die Elemente der Kombination aus und wendet die Prozedur auf die Argumente an.

Das Substitutionsmodell beschreibt, dass Werte der Reihe nach ausgewertet und angewandt werden. Dabei gibt es zwei 'Reihenfolgen':  
  
Bei der normaler Reihenfolge wird voll erweitert und dann reduziert.  
Bei applikativer Reihenfolge (Interpreter) werden die Argumente ausgewertet und dann angewandt.   
Der Vorteil der applikativen Reihenfolge ist, dass Ausdrücke teilweise nicht öfters ausgewertet werden müssen. Außerdem ist die Auswertung der normalen Reihenfolge sehr viel komplizierter.

!TODO: Beispiel

### 1.1.6 Bedingte Ausdrücke und Prädikate

*Fallunterscheidung - Condition:*

(define (abs x)  
 (cond ((> x 0) x)  
 ((= x 0) 0)  
 ((< x 0) -x)  
 ))

(cond (*<p1> <a1>)*

(*<p2> <a2>)…*

Auswertung: Erst wird p1 ausgewertet. Ist dies false, wird das nächste Prädikat p2 ausgewertet.   
Dies erfolgt solange, bis ein Prädikat mit true gefunden wurde. Dann liefert der Interpreter den Wert des entsprechenden Folgeausdrucks a zurück.  
Sollte kein Prädikat wahr sein, ist der Wert des *cond* nicht definiert.

Prädikat wird für alle Prozeduren verwendet, die true/false zurückgeben bzw Ausdrücke, die true/false als Wert bei der Auswertung annehmen.

*Fallunterscheidung - Cond-else:*

(define (abs x)  
 (cond ((< x 0) (- x))  
 (else x)))

*Fallunterscheidung - If:*

Besondere Art der Fallunterscheidung, da nur zwei Fälle angenommen werden können:

(define (abs x)  
 (if (< x 0)  
 (- x)  
 x))

Allgemein:

(if *<prädikat> <folge> <alternative>*)

### 1.1.7 Beispiel: Berechnung der Quadratwurzel nach dem Newtonschen Iterationsverfahren

Prozeduren in der Informatik muss ausgedrückt werden, WIE etwas geschehen soll. Es hilft oft wenig, zu wissen WAS das Ergebnis ist (WAS IST) - Deklaratives Wissen/Beschreibung in der Mathematik; Imperatives Wissen/Beschreibung in der Informatik.

Beispiel: Quadratwurzel

Mathematik: WAS IST die Wurzel von...

Informatik: WIE berechnet man die Wurzel - Newtonisches Iterationsverfahren.