## 5 Rekursive Definitionen

## 6 Mathematische Grundlagen: Relationen, Ordnungen, Induktive Definitionen

# 7 Fallstudie: Tower of Hanoi

## 7.1 Spielbeschreibung

Der Spielzustand besteht aus drei Säulen 1, 2, 3, auf denen insgesamt $n$ Scheiben unterschiedlicher Größe aufgestapelt sind.

Ein Spielzug besteht darin, die oberste Scheibe von einer Säule auf eine andere zu bewegen. Dabei darf niemals eine Scheibe auf einer kleineren Scheibe zu liegen kommen.

Zu Beginn befinden sich alle $n$ Scheiben auf Säule 1.

**Aufgabe:** finde eine Zugfolge, die alle Scheiben von Säule 1 auf Säule 3 transportiert.

## 7.2 Wiederholte Berechnungen: Der let-Ausdruck

```
(define square-sum
  (lambda (x y)
    (* (+ x y) (+ x y))))
```

wiederholt die Auswertung von `(+ x y)`.
Verbesserung durch benanntes Zwischenergebnis:

```
(let ((sum (+ x y))) (* sum sum))
```

Format: `(let ((`$\langle$*variable*$\rangle$ $\langle$*expression*$\rangle_1$`))` $\langle$*expression*$\rangle_2$`)`

- erster Ausdruck $\langle$*expression*$\rangle_1$ wird ausgewertet
- Wert des ersten Ausdrucks wird für Variable im zweiten Ausdruck eingesetzt
- zweiter Ausdruck $\langle$*expression*$\rangle_2$ wird ausgewertet und liefert Wert des gesamten Ausdrucks

$\Rightarrow$ $\langle$*variable*$\rangle$ ist *lokale Variable*, die nur in $\langle$*expression*$\rangle_2$ bekannt ist!

## 7.3 Wiederholte Berechnungen vermeiden

```
(define square-sum
  (lambda (x y)
    (let ((sum (+ x y)))
      (* sum sum))))

   (square-sum 4 3)
=> ((lambda (x y) (let ((sum (+ x y))) (* sum sum))) 4 3)
=> (let ((sum (+ 4 3))) (* sum sum))
=> (let ((sum 7)) (* sum sum))
=> (* 7 7)
=> 49
```

## 7.4 Verwendung von Let-Ausdrücken

- Vermeiden von wiederholten Berechnungen
- Definition von benannten Zwischenergebnissen

### Let-Ausdrücke sind syntaktischer Zucker

- erleichtern das Programmieren (Lesbarkeit)
- können durch Kombination anderer Ausdrücke beschrieben werden

### Alternative Definition von let

`(let ((x` $\langle$*expression*$\rangle_1$`))` $\langle$*expression*$\rangle_2$`)`

$\equiv$

`((lambda (x)` $\langle$*expression*$\rangle_2$`)` $\langle$*expression*$\rangle_1$`)`

## 7.5 Let*-Ausdrücke

Ein geschachtelter let-Ausdruck

```
(let ((x (- a b)))
  (let ((y (- b c)))
    (let ((z (- c d)))
      (+ (* x y) (* y z) (* z x)))))
```

kann zur besseren Lesbarkeit als let*-Ausdruck geschrieben werden:

```
(let* ((x (- a b))
       (y (- b c))
       (z (- c d)))
  (+ (* x y) (* y z) (* z x)))
```


**Mantra #9 (Wunschdenken, Top-Down-Design)**

Verschiebe Probleme, die noch nicht sofort lösbar sind, in noch zu schreibende Prozeduren. Lege für diese Prozeduren Beschreibung und Vertrag fest, aber schreibe sie später.

**Mantra #10 (lokale Variable)**

Benenne Zwischenergebnisse mit lokalen Variablen.