

Informatik I

7. Der let-Ausdruck und eine graphische Anwendung

Jan-Georg Smaus

Albert-Ludwigs-Universität Freiburg

18. November 2010

Informatik I

18. November 2010 — 7. Der let-Ausdruck und eine graphische Anwendung

7.1 Der let-Ausdruck

7.2 Das Graphik-Paket `image.ss`

7.3 Fraktale Bilder

Der let-Ausdruck

7.1 Der let-Ausdruck

Der let-Ausdruck

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 ((v e)) b)`

- ▶ Erster Ausdruck *e* wird ausgewertet;
- ▶ Wert von *e* wird für *v* im zweiten Ausdruck *b* eingesetzt;
- ▶ zweiter Ausdruck *b* wird ausgewertet und liefert Wert des gesamten Ausdrucks

⇒ *v* ist **lokale Variable**, die nur in *b* bekannt ist.

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
```

Verwendung von let-Ausdrücken

- ▶ Vermeiden von wiederholten Berechnungen
- ▶ Definition von benannten Zwischenergebnissen

let-Ausdrücke sind **syntaktischer Zucker**:

- ▶ Sie erleichtern das Programmieren (Lesbarkeit).
- ▶ Sie können durch Kombination anderer Ausdrücke beschrieben werden. Der Ausdruck

```
(let ((v e)) b)
```

kann durch

```
((lambda (v) b) e)
```

ersetzt werden.

Mehrere lokale Variablen

Ein let-Ausdruck kann auch dazu verwendet werden, mehrere lokale Variablen gleichzeitig einzuführen: Der Ausdruck

```
(let ((v1 e1) ... (vn en)) b)
```

steht für

```
((lambda (v1 ... vn) b) e1 ... en)
```

Dies erklärt auch die scheinbar unnötigen zusätzlichen Klammern im Fall nur einer lokalen Variablen.

Gleichzeitige Bindung

Ein let mit mehreren Variablen bindet alle Variablen gleichzeitig. Demnach ist folgendes ein Fehler:

```
(let ((a 1)
      (b (+ a 1)))
  b)
```

=> **reference to an identifier before its definition: a**

bzw., a müsste zusätzlich außerhalb des let-Ausdrucks definiert sein:

```
(define a 5)
(let ((a 1)
      (b (+ a 1)))
  b)
=> 6
```

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)))
```

Unterschied zwischen let und let*

Bei let* erfolgt die Bindung sequentiell:

```
(let* ((a 1)
      (b (+ a 1)))
      b)
=> 2
```

bzw.

```
(define a 5)
(let* ((a 1)
      (b (+ a 1)))
      b)
=> 2
```

MANTRA

Mantra #9 folgt später ...

MANTRA #10 (lokale Variablen)

Benenne Zwischenergebnisse mit lokalen Variablen.

7.2 Das Graphik-Paket image.ss

Prozeduren zur Erzeugung von Bildern

Das Teachpack image.ss stellt Prozeduren zur Erzeugung von Bildern zur Verfügung. Alle haben als Resultatsorte image. Ein Wert der Sorte image wird direkt in der REPL angezeigt.

- ▶ (`(: rectangle (natural natural mode image-color -> image)`)
 - ▶ Breite und Höhe des Rechtecks
 - ▶ mode ist (one-of "solid" "outline")
 - ▶ color ist Name einer Farbe, z.B. "red", "blue", "yellow", "black", "white" oder "gray"
- ▶ (`(: circle (natural mode image-color -> image)`)
 - ▶ Radius des Kreises

Weitere Bilderzeuger

- ▶ (`(: ellipse (natural natural mode image-color -> image)`)
 - ▶ Breite und Höhe der Ellipse
- ▶ (`(: triangle (natural mode image-color -> image)`)
 - ▶ gleichseitiges Dreieck
- ▶ (`(: line (natural natural number number number number image-color -> image)`)
 - Aufruf (`(line w h x1 y1 x2 y2 c)`) liefert
 - ▶ Bild der Größe $w \times h$
 - ▶ Darin eine Linie von $(x1, y1)$ nach $(x2, y2)$
 - ▶ Koordinatenursprung $(0,0)$ ist oben links
 - ▶ Farbe c
- ▶ (`(: text (string natural image-color -> image)`)
 - ▶ Aufruf (`(text s f c)`) erzeugt ein Bild mit Text s in Farbe c , wobei die Buchstaben die Größe f haben

Kombination von Bildern

- (`(: overlay (image image h-place v-place -> image)`)
 - ▶ legt das zweite Bild auf das erste
 - ▶ h-place ist die Signatur für horizontale Positionsangaben


```
(define h-place
  (signature
    (mixed integer ; Abstand vom linken Rand
      (one-of "left" "right" "center"))))
```
 - ▶ v-place ist die Signatur für vertikale Positionsangaben


```
(define v-place
  (signature
    (mixed integer ; Abstand vom oberen Rand
      (one-of "top" "bottom" "center"))))
```
 - ▶ Das Ergebnisbild umfasst beide Argumentbilder.

Weitere Kombinationen von Bildern

- ▶ (`(: above (image image h-mode -> image)`)
- ▶ (`(: beside (image image v-mode -> image)`)
- ▶ (`(: image-width (image -> natural)`)
- ▶ (`(: image-height (image -> natural)`)

Und andere mehr, siehe [?].

Anwendungen von image.ss finden sich in den Folien von Professor Thiemann, siehe

<http://proglang.informatik.uni-freiburg.de/teaching/info1/2009/folien/07-event.pdf>

7.3 Fraktale Bilder

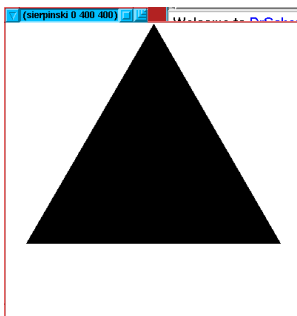
Fraktale Bilder

Konstruktion eines Bildes aus

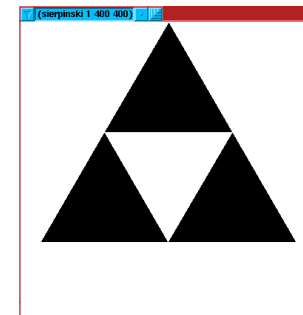
- ▶ wenigen Grundbausteinen
- ▶ einem Kombinationsverfahren, das
 - ▶ Grundbausteine transformiert (dreht, spiegelt, verkleinert, projiziert) und
 - ▶ Teile der Grundbausteine durch Grundbausteine ersetzt
 - ▶ Grundbausteine zusammenfügt

Beispiel: Sierpinski-Dreieck

Ein **Sierpinski-Dreieck 0. Ordnung** ist ein gleichseitiges Dreieck:



Sierpinski-Dreieck 1. Ordnung

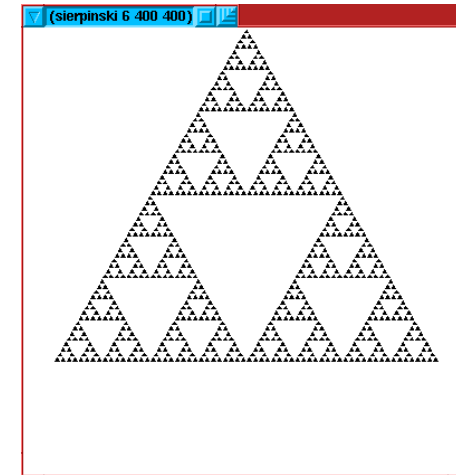


- ▶ Drei Kopien des Sierpinski-Dreiecks 0. Ordnung
- ▶ mit halber Höhe
 - ▶ die unteren beiden zentriert nebeneinander
 - ▶ das obere zentriert darüber

Sierpinski-Dreieck $(n + 1)$ -ter Ordnung

- ▶ Drei Kopien des Sierpinski-Dreiecks n -ter Ordnung
- ▶ mit halber Höhe
 - ▶ die unteren beiden zentriert nebeneinander
 - ▶ das obere zentriert darüber

Sierpinski-Dreieck 6. Ordnung



Programmierung des Sierpinski-Dreiecks n -ter Ordnung

```
; erzeuge Sierpinski-Dreieck der Ordnung n
; mit gegebener Höhe
(: sierpinski (natural real -> image))
(define sierpinski
  (lambda (n h)
    (if (zero? n)
        (triangle h "solid" "black")
        (let ((sn-1 (sierpinski (- n 1) (/ h 2))))
          (above
            sn-1
            (beside sn-1 sn-1 "center")
            "center")))))
```

Eigenschaften des Sierpinski-Dreiecks

- ▶ Das Sierpinski-Dreieck ist der Grenzwert $n \rightarrow \infty$ der Dreiecke der Ordnung n
- ▶ Es ist nicht leer, aber seine Fläche ist 0.
- ▶ Beispiel für ein **Fraktal**.
- ▶ Andere Beispiele in den Folien von Professor Thiemann, siehe <http://proglang.informatik.uni-freiburg.de/teaching/info1/2009/fohlen/14-fraktal.pdf>.

Literatur I