

Informatik I

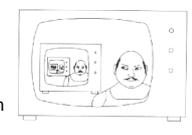
Rekursion



G. Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de

Motivation

- Neue Denkweise
- Leistungsfähiges Algorithmenschema
 - Divide-and-conquer
- Viele Berechnungen und Datenstrukturen sind von Haus aus selbst-bezüglich:
 - Ein Verzeichnis enthält Daten und andere Verzeichnisse
 - Euklid's ggT Algorithmus
 - Quicksort
 - Verkettete Datenstrukturen




Drawing Hands
M. C. Escher, 1948

G. Zachmann Informatik 1 - WS 05/06 Rekursion 2

Beispiel: Größter Gemeinsamer Teiler

- Finde größte ganze Zahl d , die p und q teilt

$$ggT(p, q) = \begin{cases} p & \text{falls } q = 0 \\ ggT(q, p \% q) & \text{sonst} \end{cases}$$

← Basisfall (base case)
← Reduktionsschritt (reduction step, converges to base case)

$$ggT(4032, 1272) = ggT(1272, 216)$$

$$= ggT(216, 192)$$

$$= ggT(192, 24)$$

$$= ggT(24, 0)$$

$$= 24.$$

$$4032 = 2^6 \times 3^2 \times 7^1$$

$$1272 = 2^3 \times 3^1 \times 53^1$$

$$ggT = 2^3 \times 3^1 = 24$$

- Anwendungen.
 - RSA Verschlüsselung
 - Geschichte von Algorithmen



Euklid, 300 v. Chr.

G. Zachmann Informatik 1 - WS 05/06 Rekursion 3

Beweis für Formel:

$$ggT(p, q) = \begin{cases} p & \text{falls } q = 0 \\ ggT(q, p \% q) & \text{sonst} \end{cases}$$

p									
q			q				p % q		
x	x	x	x	x	x	x	x	x	x

↑ ggT

$$p = kx$$

$$q = lx$$

$$p \% q = r = p - nx = kx - nlx = (k - nl)x$$

G. Zachmann Informatik 1 - WS 05/06 Rekursion 4

Python Implementierung

```
def gcd(p, q):
    if q == 0:
        return p
    else:
        return gcd(q, p%q)
```

← base case
← reduction step

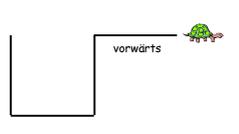
$$ggT(p, q) = \begin{cases} p & \text{falls } q = 0 \\ ggT(q, p \% q) & \text{sonst} \end{cases}$$

← Basisfall (base case)
← Reduktionsschritt (reduction step, converges to base case)

G. Zachmann Informatik 1 - WS 05/06 Rekursion 5

Exkurs: Turtle-Grafik

- Einfache Beschreibung von 2D-Grafiken
- Idee: Schildkröte auf Blatt Papier zieht Stift hinter sich her
- Schildkröte kann nur wenige Befehle ausführen:
 - Stift hoch / runter
 - Stiftfarbe wechseln
 - x mm nach vorne krabbeln (und Stift dabei hinter sich herziehen)
 - sich um n Grad nach links/rechts drehen
- Beispiel:
 - F: 1 cm vorwärts (Stift unten)
 - L: Linksdrehung um 90°
 - R: Rechtsdrehung um 90°
 - Zeichensequenz: F L F L F R F



G. Zachmann Informatik 1 - WS 05/06 Rekursion 6

Koch-Schneeflocke

- Rekursiver Algo für Koch-Kurve der Ordnung n:
 - Zeichne die Kurve der Ordnung n-1
 - Drehe das Blatt um 60°
 - Zeichne die Kurve der Ordnung n-1
 - Drehe um -120°
 - Zeichne die Kurve der Ordnung n-1
 - Drehe um 60°
 - Zeichne die Kurve der Ordnung n-1

```

def koch( n, size):
    if n == 0:
        Turtle.move(size)
    else:
        koch(n-1, size)
        Turtle.rotate(60)
        koch(n-1, size)
        Turtle.rotate(-120)
        koch(n-1, size)
        Turtle.rotate(60)
        koch(n-1, size)
  
```

G. Zachmann Informatik 1 - WS 05/06 Rekursion 7

Rekursions-Baum

G. Zachmann Informatik 1 - WS 05/06 Rekursion 8

Koch-Schneeflocke

- Rekursiver Algo für "Koch-Schneeflocke der Ordnung n":
 - Zeichne Koch-Kurve der Ordnung n
 - Drehe um -120°
 - Zeichne Koch-Kurve der Ordnung n
 - Drehe um -120°
 - Zeichne Koch-Kurve der Ordnung n

G. Zachmann Informatik 1 - WS 05/06 Rekursion 9

Koch-Schneeflocke in Python

```

def snowflake( N, width ):
    height = int( width * 2.0 / math.sqrt(3.0) )
    size = width / math.pow( 3.0, N )

    // create and init turtle
    ...

    // draw snowflake
    x0, y0 = 0, width * math.sqrt(3.0) / 2
    turtle.fly( x0, y0 )
    koch( N, size )
    turtle.rotate(-120)
    koch( N, size )
    turtle.rotate(-120)
    koch( N, size )
  
```

G. Zachmann Informatik 1 - WS 05/06 Rekursion 10

Türme von Hanoi

- Verschiebe alle Scheiben von linkem Stab zu rechtem Stab
 - Jeweils darf nur eine Scheibe verschoben werden
 - Eine Scheibe darf entweder auf leeren Stab oder auf eine größere Scheibe gesetzt werden, nicht auf kleinere Scheibe

Start Ende

Towers of Hanoi demo
<http://www.mazeworks.com/download/index.htm>

Edouard Lucas (1833)

G. Zachmann Informatik 1 - WS 05/06 Rekursion 11

Rekursive Lösung

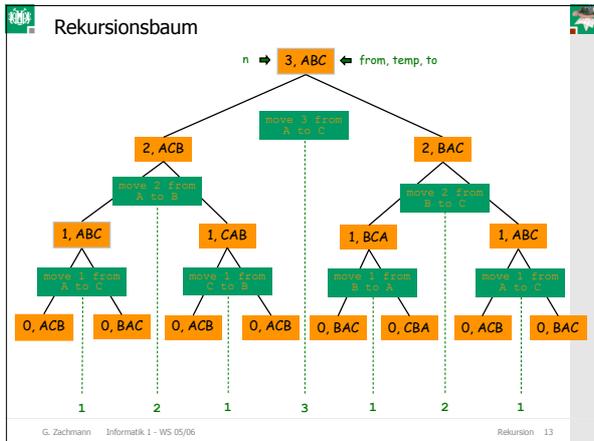
A B C

Verschieben n-1 kleinste Scheiben auf Stab B

Verschiebe größte Scheibe auf Stab C

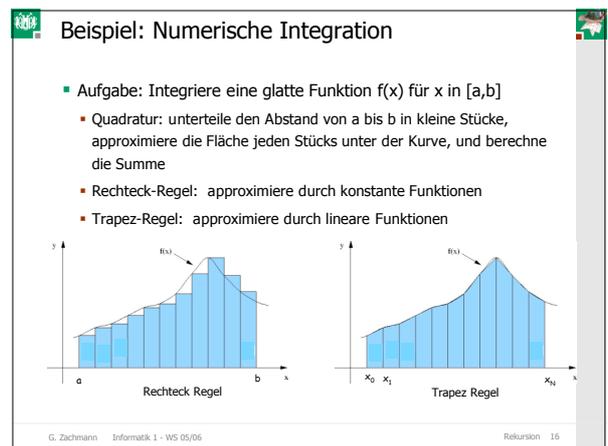
Verschieben n-1 kleinste Scheiben auf Stab C

G. Zachmann Informatik 1 - WS 05/06 Rekursion 12



- ### Eigenschaften der Lösung der Türme von Hanoi
- $2^N - 1$ Scheibenbewegungen, um N Scheiben-Problem zu lösen
 - Kleinste Scheibe bewegt sich immer in die gleiche Richtung (bei "kreisförmiger" Anordnung der Stäbe)
 - Es gibt auch nicht-rekursive (iterative) Lösungen
- G. Zachmann Informatik 1 - WS 05/06 Rekursion 14

- ### Divide-and-Conquer
- Eine **Algorithmentechnik** (Algorithmenparadigma)
 - Annahme: wir hätten schon eine Lösung für Probleme der Größe N-1
 - 1. Teile Problem der Größe N auf in zwei oder mehr kleinere Probleme mit ähnlicher Struktur
 - 2. Löse die kleineren Probleme (rekursiv mit der gleichen Methode)
 - 3. Kombiniere die Lösungen zur Lösung des originalen Problems
 - Historischer Ursprung
 - Julius Caesar (100 v.Chr. - 44 v.Chr.)
 - "Divide et impera."
- 
- Für viele Probleme gibt es elegante *divide-and-conquer* Lösung
- G. Zachmann Informatik 1 - WS 05/06 Rekursion 15



Python-Code für Trapez-Regel

```

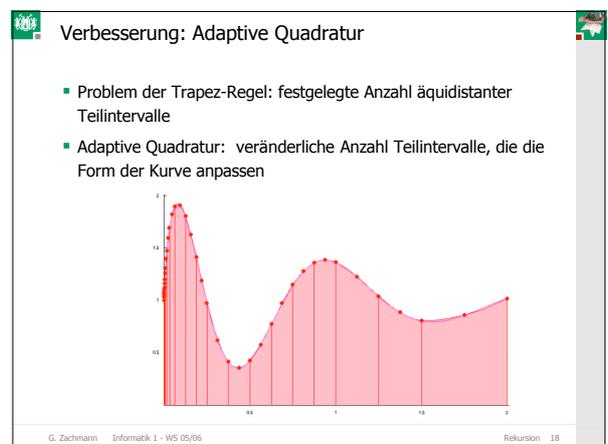
def f(x):
    # zu integrierende Funktion
    return math.exp(-x*x/3) / math.sqrt(2*math.pi)

def trapezoid(a, b, N):
    # N = Anzahl der
    # Unterteilungen des Intervalls
    h = (b - a) / N
    sum = 0.5 * h * (f(a) + f(b))
    for k in range(1, N):
        sum += h * f(a + h*k)
    return sum

print trapezoid(-3.0, 3.0, 1000)

```

G. Zachmann Informatik 1 - WS 05/06 Rekursion 17



Rekursive Lösung

- Rekursion:
 - Unterteile das Intervall in zwei gleiche Stücke
 - Berechne die Fläche jedes Stückes rekursiv
 - Liefere die Summe zurück
- Abbruchkriterium (Basisfall):
 - Durch zwei versch. Quadraturverfahren die geg. Fläche approximieren
 - Wenn fast gleich, diese Fläche liefern; sonst: Rekursion weitermachen

```
def adaptive(a, b, eps) :
    area = trapezoid(a, b, 10)
    check = trapezoid(a, b, 5)
    if math.abs(area - check) > eps:
        m = (a + b) / 2
        area = adaptive(a, m) + adaptive(m, b)
    return area
```

G. Zachmann Informatik 1 - WS 05/06 Rekursion 19

Eine schlechte rekursive Funktion

- Hier: kein Basis-Fall!
- Analog zu unendlichen Schleifen mit for- and while- Schleifen.

```
def ulam(x):
    print x
    if x % 2 == 0:
        return ulam(x / 2)
    else:
        return ulam(3 * x + 1)
```

```
#!/ulam.py
5
16
8
4
2
1
4
2
1
...
```

- Nicht bekannt, ob für beliebige Startwerte terminiert

G. Zachmann Informatik 1 - WS 05/06 Rekursion 20

Eine zweifelhafte rekursive Funktion

- Hier: es gibt einen "Anti-Reduktions-Schritt"
 - konvergiert die Folge der Parameter immer zum Basis-Fall??

```
def ulam(x):
    if x <= 1:
        return
    else if x % 2 == 0:
        return ulam(x / 2)
    else:
        return ulam(3 * x + 1) ← Anti-Reduktions-Schritt
```

- Später: in der Berechenbarkeitstheorie wird genauer untersucht, ob wir immer feststellen können, ob eine Funktion terminiert
 - Halte-Problem

G. Zachmann Informatik 1 - WS 05/06 Rekursion 21