

Zentralübung zur Informatik I

Dr. Markus Schneider
Technische Universität München

Wintersemester 2000/2001
11. Dezember 2000

Überblick

Rekursionsarten

- Lineare Rekursion
- Linear repetitive Rekursion, Technik der Einbettung (Prog–Aufgabe 25)
- Kaskadenartige Rekursion
- Verschränkte Rekursion

Terminierung

- Technik der Abstiegsfunktion
- Beispiele

Induktion über die natürlichen Zahlen

- Theoretische Grundlagen
- Beispiele

Rekursionsart: Lineare Rekursion

- Definition: Eine Rekursion heißt **linear**, wenn in jedem Zweig des bedingten Ausdrucks höchstens ein Selbstaufruf auftritt.

- Beispiele:

```
int fakultaet (int n) {  
    return n==0 ? 1 : fakultaet(n-1) * n; }
```

```
int zaehle_a (String s) {  
    return isEmpty(s) ? 0 : first(s) == 'a' ? zaehle_a(rest(s)) + 1  
        : zaehle_a(rest(s))  
};
```

```
int ggT (int, a, int b) {  
    return a==b ? a  
        : a > b ? ggT(a-b, b)  
        : ggT(a, b-a); }
```

Unterscheidung:

Nichtrepetitiv

```
int fakultaet (int n) {  
    return n==0 ? 1 : fakultaet(n-1) * n;  
}
```

Gesucht:

```
fakultaet(4)  
    (fakultaet(3) * 4)  
        ((fakultaet(2) * 3) * 4)  
            (((fakultaet(1) * 2) * 3) * 4)  
                (((((fakultaet(0) * 1) * 2) * 3) * 4)  
                    (((1 * 1) * 2) * 3) * 4)  
                        (((1 * 2) * 3) * 4)  
                            ((2 * 3) * 4)  
                                (6 * 4)  
                                    24
```

Repetitiv

```
int ggT (int, a, int b) {  
    return a==b ? a : a > b ? ggT(a-b, b)  
    : ggT(a, b-a); }
```

Gesucht:

```
ggT(374, 102)  
ggT(272, 102)  
ggT(170, 102)  
ggT(68, 102)  
ggT(68, 34)  
ggT(34, 34)  
34
```

Umwandlung: Nichtrepetitiv in Repetitiv durch Einbettung

Beispiel: Einbettung der Fakultätsfunktion

Gesucht: fakultaet(5)

Idee: Einführung eines Funktionsparameters, der das Produkt der bisher bearbeiteten Faktoren speichert.

```
int produkt (int n, int vorlaeufigesProdukt) {  
    return n != 0  
        ? produkt(n-1, vorlaeufigesProdukt * n)  
        : vorlaeufigesProdukt ; }  
  
int fakultaet(int n) {  
    return produkt(n, 1);  
}
```

```
    produkt(5, 1)  
    produkt(4, 1 * 5)  
    produkt(4, 5)  
    produkt(3, 5 * 4)  
    produkt(3, 20)  
    produkt(2, 20 * 3)  
    produkt(2, 60)  
    produkt(1, 60 * 2)  
  
    produkt(1, 120)  
    produkt(0, 120)  
    120
```

Vorteile repetitiver Rekursion:

Die Rekursion ist mit dem letzten rekursiven Aufruf beendet.

Kein Speichern von Elementen vorheriger Funktionsaufrufe nötig

Die Lösung des allgemeineren Problems lässt sich unter Umständen an anderer Stelle verwenden (Wiederverwendbarkeit !)

Kaskadenartige Rekursion (*nichtlineare Rekursion*)

- **Definition:** Eine rekursive Funktion, die nicht linear rekursiv ist, heißt kaskadenartig rekursiv
- **Beispiel:** Fibonacci-Zahlen
Gegeben sei ein neu geborenes Kaninchenpaar. Jedes Kaninchenweibchen, das mindestens zwei Monate alt ist, bringt jeden Monat ein weiteres Paar zur Welt. Kaninchen leben ewig.
Wieviele Kaninchenpaare gibt es nach n Monaten?

Lösung:

```
int fib (int n) {  
    return n==0 ? 0  
           : n==1 ? 1  
           : fib(n-2) + fib(n-1); }  
:
```

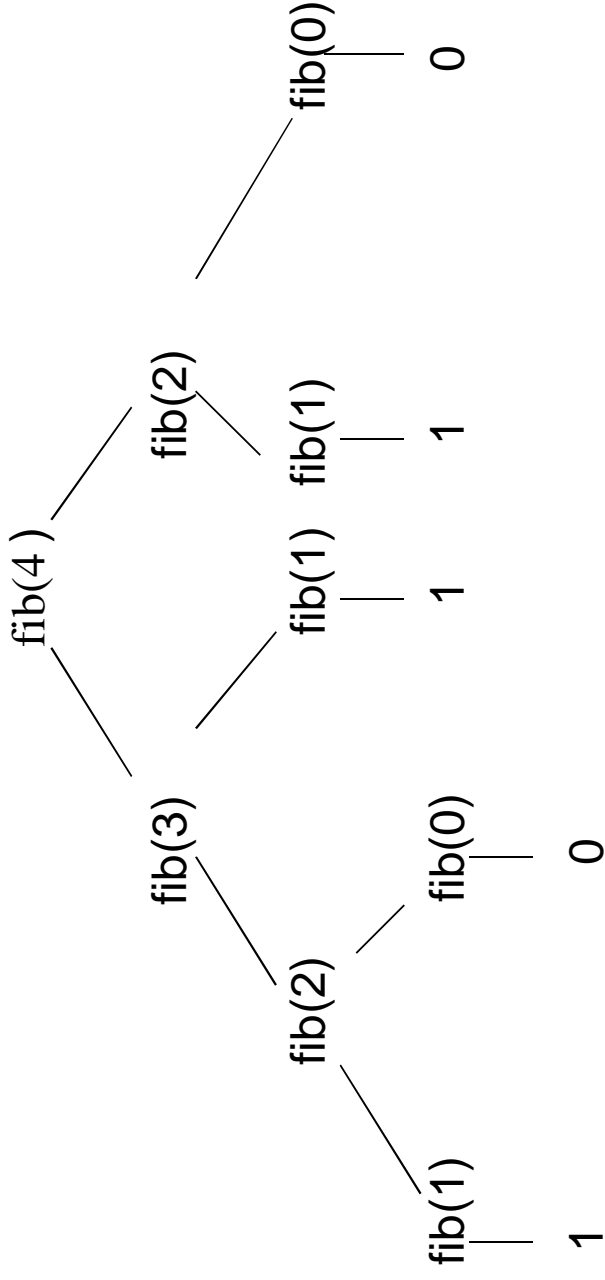
- Aufrufstruktur:

```
fib(4) = fib(3) + fib(2)  
       = fib(2) + fib(1) + fib(1) + fib(0)  
       = fib(1) + fib(0) + 1 + 1 + 0  
       = 1 + 0 + 1 + 1 + 0  
       = 4
```

Auswertung des Fibonacci–Aufrufbaumes

$$\begin{array}{l} * \\ \text{fib}(2) + \text{fib}(3) + \text{fib}(4) \\ 1 + 1 + 1 \\ \text{fib}(1) + \text{fib}(2) + \text{fib}(1) + \text{fib}(0) \\ 1 + 1 + 1 + 0 \\ * \\ \text{fib}(3) + \text{fib}(4) + \text{fib}(2) \\ 2 + 1 + 1 \\ \text{fib}(2) + \text{fib}(3) + \text{fib}(1) + \text{fib}(0) \\ 1 + 1 + 1 + 0 \\ * \\ \text{fib}(4) + \text{fib}(3) + \text{fib}(2) \\ 3 + 2 + 1 \\ \text{fib}(3) + \text{fib}(4) + \text{fib}(2) + \text{fib}(1) + \text{fib}(0) \\ 2 + 1 + 1 + 1 + 0 \end{array}$$

- **Beispiel 1 (kaskadenartige Rekursion):** Gesucht sei die Anzahl der rekursiven Aufrufe bei Aufruf der Fibonacci–Funktion $\text{Fib}(n)$:



- **Lösung:**

```
int aufrufe (int n) {
```

```
    return (n==0 || n==1) ? 1
```

```
        : 1 + aufrufe(n-1) + aufrufe(n-2); }
```

- **Behauptungen:**

- Die Funktion aufrufe(n) terminiert für alle $n \geq 0$
- Es gilt: $\text{aufrufe}(n) = 2 * \text{fib}(n+1) - 1$

Beispiel: Verschränkte Rekursion

- Problem: Gegeben sei ein Wort aus dem Zeichenvorrat $T = \{L, O\}$;
Beispiel: LLOL
- Gesucht: Rekursive Funktion, die ermittelt, ob die Anzahl der Zeichen L gerade oder ungerade ist.

```
boolean number_odd(String s) {  
    return isEmpty(s) ? false  
        : last(s)==O ? number_odd(lrest(s))  
        : number_even(lrest(s)); }  
  
boolean number_even(String s) {  
    return isEmpty(s) ? true  
        : last(s)==O ? number_even(lrest(s))  
        : number_odd( lrest(s)); }  
  
• Beispiel: number_even(LLOL)  
    number_even(LLOL)  $\Rightarrow$  number_odd(LLO)  
     $\Rightarrow$  number_odd(LL)  
     $\Rightarrow$  number_even(L)  
     $\Rightarrow$  number_odd(` `)  
     $\Rightarrow$  false
```

Nachweis der Terminierung

- Gegeben sei die rekursive Funktionsdeklaration
$$T f(T_1 \ x_1, T_2 \ x_2, \dots, T_n \ x_n,) \{ \text{return } A; \}$$
wobei der Funktionsrumpf A rekursive Funktionsaufrufe der Form $f(A_1, A_2, \dots, A_n)$ enthalte.
 - **Beispiel: int fakultaet (int n) {return n==0 ? 1: fakultaet(n-1) * n;}**
- Seien T'_1, T'_2, \dots, T'_n Teilmengen von T_1, T_2, \dots, T_n derart, dass sich die Funktionsparameter eines jeden rekursiven Aufrufes zu Werten s_1, s_2, \dots, s_n aus T'_1, T'_2, \dots, T'_n berechnen lassen, falls die Parameter des ursprünglichen Aufrufes t_1, t_2, \dots, t_n ebenfalls Werte dieser Teilmengen sind.
 - **Beispiel:** Bei der Fakultät wäre T' (die Menge der natürlichen Zahlen einschließlich der Null) eine Teilmenge der Menge T der ganzen Zahlen.
 - Dann wäre $s = n-1$ stets aus T' , wenn n aus T'
- Existiert eine Funktion $h : T'_1 \times T'_2 \times \dots \times T'_n \rightarrow \text{IN}$ mit der Eigenschaft $h(s_1, s_2, \dots, s_n) < h(t_1, t_2, \dots, t_n)$ dann terminiert die Funktion f für alle Parameterwerte aus T'_1, T'_2, \dots, T'_n .

1. Terminierungsbeispiel: Fakultätsfunktion

```
int fakultaet (int n) {  
    return n==0 ? 1: fakultaet(n-1) * n;}  
}
```

- T_1 sind hier die ganzen Zahlen: int
- T'_1 sind hier die natürlichen Zahlen einschließlich der Null
- Für x_1 und A_1 gilt hier: $x_1 = n$, $A_1 = n - 1$
- t_1 sind die Werte, die sich beim Auswerten von x_1 ergeben.
- s_1 sind die Werte, die sich beim Auswerten von A_1 ergeben.
- Als Abstiegsfunktion wählen wir:
 - $h: T_1 \rightarrow \mathbb{N}; h(x) = x$
- Nun gilt:
 - Da der rekursive Aufruf $\text{fakultaet}(n-1)$ nur im Fall $n > 0$ ausgewertet wird, gilt stets: $s_1 \geq 0$. d.h. s_1 und t_1 sind stets Elemente von T'_1 .
 - Außerdem gilt:
 - $h(s_1) = h(n-1) = n-1 < n = h(n) = h(t_1)$
- Somit terminiert die Funktion für alle Werte aus T'_1 , d.h. für alle nichtnegativen Zahlen.

2. Beispiel: Funktion der Anzahl der rekursiven Aufrufe von $\text{fib}(n)$

```
int aufrufe (int n) {  
    return (n==0 || n==1) ? 1  
        : 1 + aufrufe(n-1) + aufrufe(n-2); }  
: 1 + aufrufe(n-1) + aufrufe(n-2); }
```

- T_1 : natürliche Zahlen einschließlich Null
- s_1 ist stets Element T_1 , da nur für $n > 1$ rekursive Aufrufe erfolgen
- Wahl der Abstiegsfunktion: $h(x) = n$
 $h(n-1) = n-1 < n = h(n)$;
 $h(n-2) = n-2 < n = h(n)$
- Somit terminiert die Funktion.

Weitere rekursive Funktionen und ihre Abstiegsfunktionen

- **Größter gemeinsamer Teiler:** Es sei $a > b$ und $b > 0$;

```
int ggT(a, b) {
```

```
    return mod(a, b) == 0 ? b : ggT(b, mod(a, b)); }
```

- T_1, T_2 : natürliche Zahlen ausschließlich Null (Division durch Null!)
- s_1, s_2 : Aufgrund der Bedingung $\text{mod}(a, b) \neq 0$ erfolgt ein rekursiver Aufruf nur mit Parameterwerten $0 < \text{mod}(a, b) < b$ (Die Bedingung verhindert auch eine eventuelle Division durch Null)

- Nun gilt: $\text{mod}(a, b) < b$. Mit der Abstiegsfunktion

$$h(x, y) = x + y$$

folgt:

$$h(b, \text{mod}(a, b)) = b + \text{mod}(a, b) < b + b$$

- Da nach obiger Voraussetzung $a > b$ ist, folgt:

$$h(b, \text{mod}(a, b)) < b + b < a + b = h(a, b)$$

- Die Terminierungsbedingung ist somit erfüllt.

Induktion über natürliche Zahlen

- Sei $A(n)$ eine Aussage über eine natürliche Zahl n , dann bedeutet das Prinzip der vollständigen Induktion:

1. Variante:

Gilt für ein $n_0 \in \mathbb{N}$ die Aussage $A(n_0)$ (**Induktionsanfang**)

und folgt aus der Gültigkeit von $A(n)$ die Gültigkeit von $A(n+1)$ (**Induktionsschritt**)

dann gilt $A(n)$ für alle $n \in \mathbb{N}$ mit $n \geq n_0$

Beispiel: Gegeben sei die rekursive Funktionsvereinbarung

$$\text{summe}(n) = \sum_{i=1}^n i^3 = \begin{cases} 1, & n = 1 \\ \text{summe}(n-1) + n^3, & n > 1 \end{cases}$$

Zu zeigen: $\text{summe}(n) = \left[\frac{n(n+1)}{2} \right]^2$

Beweis:

Induktionsanfang: $n_0 = 1 \Rightarrow \text{summe}(1) = 1 \cdot (1+1) / 2 = 1$

\Rightarrow Behauptung für $n_0 = 1$ richtig

Induktionsschritt: Es gelte nun die Behauptung

$\text{summe}(n) = (n(n+1)/2)^2$ für alle $n_0 < n' \leq n$.

Also gilt gemäß obiger Funktionsvorschrift:

$$\text{summe}(n+1) = \text{summe}(n) + (n+1)^3$$

$$\text{summe}(n+1) = (n(n+1)/2)^2 + (n+1)^3$$

$$\text{summe}(n+1) = n^2(n+1)^2/4 + (n+1)^3$$

$$\text{summe}(n+1) = 1/4[n^2(n+1)^2 + 4(n+1)^3]$$

$$\text{summe}(n+1) = 1/4[(n+1)^2(n^2 + 4n + 4)]$$

$$\text{summe}(n+1) = ((n+1)(n+2)/2)^2$$

Fortsetzung: Induktion

- **2. Variante** (Zwei Spezialfälle als Induktionsanfang): Sei $A(n)$ wie zuvor eine Aussage über eine natürliche Zahl n .
Induktionsanfang: Gelten $A(n_0)$ und $A(n_0 + 1)$
Induktionsschritt: Folgt aus der Gültigkeit von $A(n-1)$ und $A(n)$ die Gültigkeit von $A(n+1)$
dann gilt $A(n)$ für alle $n \in \mathbb{IN}$ mit $n \geq n_0$
(analoges gilt natürlich auch für drei und mehr Spezialfälle)

Beispiel: Für die Funktion zur Berechnung der Anzahl der rekursiven Aufrufe der Fibonacci-Zahlen $\text{fib}(n)$

```
int aufrufe (int n) {  
    return (n==0 || n==1) ? 1  
        : 1 + aufrufe(n-1) + aufrufe(n-2); }  
soll gelten:
```

$$\text{aufrufe}(n) = 2 * \text{fib}(n+1) - 1$$

Beweis: Zu beweisen ist die Aussage:

$$\text{aufrufe}(n) = 2 * \text{fib}(n+1) - 1$$

Induktionsanfang:

$$\text{aufrufe}(n_0 = 0) = 2 * \text{fib}(1) - 1 = 1$$

$$\text{aufrufe}(n_1 = 1) = 2 * \text{fib}(2) - 1 = 1$$

Somit gilt $A(n_0)$ und $A(n_1)$.

Induktionsschritt:

Laut Definition gilt für $n > 1$

$$\text{aufrufe}(n) = 1 + \text{aufrufe}(n-1) + \text{aufrufe}(n-2)$$

Somit folgt:

$$\text{aufrufe}(n+1) = 1 + \text{aufrufe}(n) + \text{aufrufe}(n-1)$$

$$\text{aufrufe}(n+1) = 1 + 2 * \text{fib}(n+1) - 1 + 2 * \text{fib}(n) - 1$$

Nach Umformung folgt:

$$\text{aufrufe}(n+1) = 2 * (\text{fib}(n+1) + \text{fib}(n)) - 1$$

$$\text{aufrufe}(n+1) = 2 * \text{fib}(n+2) - 1$$