Informatik im Maschinenbau

Rekursion

© 2015 Martin v. Löwis

Rekursion

- Rekursive Definition: Inhalt der Definition bezieht sich auf sich selbst
- Beispiel: "Die n-te Zweierpotenz ist das Doppelte der (n-1)-ten Zweierpotenz. Die 0-te Zweierpotenz ist 1"
  - $x_0 = 1$ (Rekursionsabbruch)
    $x_n = 2*x_{n-1}$
- Alternative Definition durch Fallunterscheidung:

$$potenz(x,n) = \begin{cases} 1, & falls\ n = 0 \\ x * potenz(x, n-1), & sonst \end{cases}$$

© 2015 Martin v. Löwis

Rekursion: Struktur

- Rekursive Funktion hat wenigstens einen Parameter
  - wird im rekursiven Aufruf verändert
- Rekursionsabbruch: Parameter hat Ausgangswert (meist 0 oder 1)
- Rekursionsschritt: Formel für Funktionsergebnis enthält Funktionsaufruf mit geändertem Parameter
  - üblicherweise verkleinert

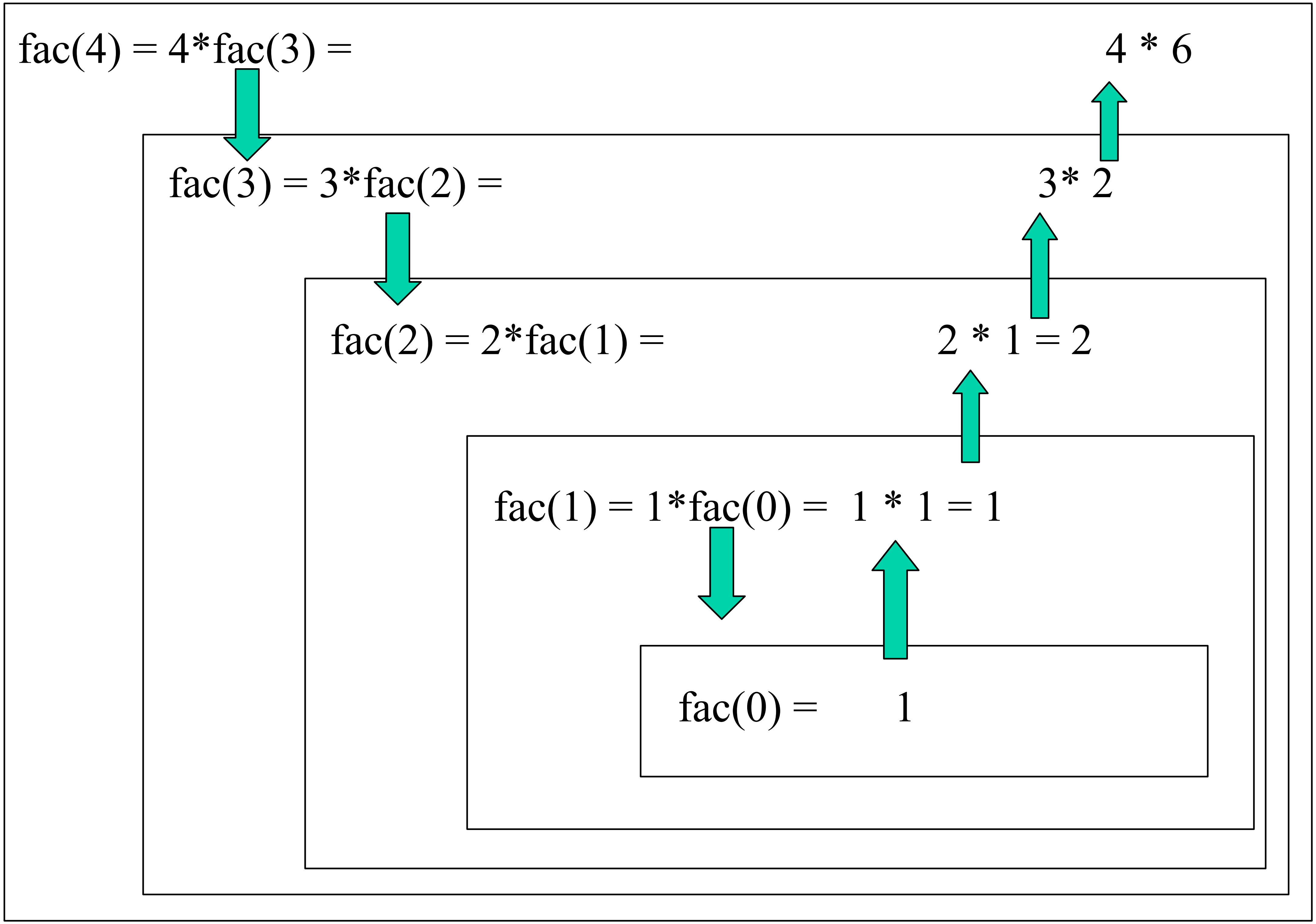
© 2015 Martin v. Löwis

Beispiel: Potenz

- ```
  double potenz(double x, int n)
  {
      if (n == 0) {
          return 1;
      } else {
          return x * potenz(x, n-1);
      }
  }
  ```

© 2015 Martin v. Löwis
  ```

Beispiel: Fakultät

$$fak(n) = \begin{cases} 1, & falls\ n = 0 \\ n * fak(n-1), & sonst \end{cases}$$

- ```
  int fak(int n)
  {
      if (n == 0) {
          return 1;
      } else {
          return n * fak(n-1);
      }
  }
  ```

© 2015 Martin v. Löwis
  ```

Verarbeitung der Rekursion



© 2015 Martin v. Löwis

Formulierung von Rekursion

- Zerlegung des Problems in Teilprobleme:
  - ein Teilproblem ist „gleichartig“ dem Originalproblem
    - Beispiel: fac(n-1) ist „im Prinzip“ genauso wie fac(n)
    - Teilproblem muss „leichter“ lösbar sein als Originalproblem
  - anderes Teilproblem kombiniert die Lösung des ersten Teilproblems mit weiterem Rechenschritt zur Gesamtlösung:
    - Beispiel: fac(n) = n*fac(n-1) (Multiplikation der Teillösung mit n)
- Rekursionsabbruch: „einfachstes“ Teilproblem wird nicht weiter zerlegt; Lösung wird „direkt“ bestimmt
- Teile-und-herrsche-Prinzip
  - Vereinfachung des Problems durch Zerlegung
  - engl.: divide-and-conquer
  - lat.: divide-et-impera
    - Historisch falsch: Teile werden nicht gegeneinander ausgespielt

© 2015 Martin v. Löwis

Die Türme von Hanoi

- Auf einem Stapel liegen N Scheiben verschiedener Durchmesser; der Durchmesser nimmt von unten nach oben schrittweise ab.
- Der Turm steht auf einem Platz A und soll zu einem Platz C bewegt werden, wobei ein Platz B als Zwischenlager benutzt werden kann.
- Dabei müssen 2 Regeln eingehalten werden:
  – Es darf immer nur eine Scheibe bewegt werden
  – Es darf nie eine größere auf einer kleineren Scheibe liegen

© 2015 Martin v. Löwis

Die Türme von Hanoi (2)

- Lösungsstrategie: induktive Lösung
  - Verschieben einer Scheibe: Scheibe von Platz 1 (z.B. A) auf Platz 2 (z.B. C)
  - Verschieben von K Scheiben: Verschiebe K-1 Scheiben von Platz 1 auf Hilfsplatz H (etwa: B), verschiebe K-te Scheibe von Platz 1 auf Platz 2, verschiebe K-1 Scheiben von H auf Platz 2
- Rekursive Sicht:
  - Angenommen, wir können bereits K-1 Scheiben verschieben, dann wissen wir auch, wie wir K Scheiben verschieben
  - Wir wissen, wie wir eine Scheibe verschieben (Rekursionsende)
  - Problem: Keine feste Zuordnung von symbolischen Plätzen (1, 2, H) zu tatsächlichen Plätzen (A, B, C)
    - Lösung: symbolische Plätze sind Variablen/Parameter, tatsächliche Plätze die Werte von dieser Variablen

© 2015 Martin v. Löwis

Die Türme von Hanoi (3)

```
void ziehe_scheibe(int nummer, string von, string nach) {
    cout << "Scheibe " << nummer << " wird von " << von <<
         " nach " << nach << " verschoben " << endl;
}
void hanoi(int N, string platz1, string hilfsplatz, string platz2)
{
    if (N == 1) {
        ziehe_scheibe(N, platz1, platz2);
    } else {
        hanoi(N-1, platz1, platz2, hilfsplatz);
        ziehe_scheibe(N, platz1, platz2);
        hanoi(N-1, hilfsplatz, platz1, platz2);
    }
}
int main() {
    hanoi(4, "A", "B", "C");
}
```

© 2015 Martin v. Löwis