Fachrichtung 6.2 — Informatik
Universität des Saarlandes
Tutorenteam der Vorlesung Programmierung 1

Programmierung 1 (Wintersemester 2012/13)

Erklärung 3

(Rekursion vs. Iteration)

Hinweis: Dieses Blatt enthält eine zusätzliche Erklärung erstellt von den Tutoren. Für die Richtigkeit besteht daher keine Gewähr.

Die Erklärung sowie ihr Thema sind weder für die Klausur relevant noch irrelevant.
Die ursprüngliche Erklärung stammt aus dem Wintersemester 11/12.

1 Rekursiver Denkansatz

- Schauen wir uns zuerst sehr einfache Fälle (Basisfälle) an, und überlege, wie wir das Problem für sie lösen könnten. Die Lösung für diese Fälle muss so offensichtlich sein, dass man sie direkt angeben kann (vor allem ohne(!) erneute Rekursion). Gute Kandidaten für Basisfälle sind oft 0 oder 1.

  Beispiele sind:

  (a) Die 0-te Potenz von x:
  $x^0 = 1$

  ```
  fun power x n =
    if n = 0
      then 1
      ...
  ```

  (b) Die Multiplikation einer natürlichen Zahl mit 1:
  $1 * b = b$

  ```
  fun mul a b =
    if a = 1
      then b
      ...
  ```

- Rekursive Lösungen erhalten wir, indem wir fest daran glauben, dass die gerade zu entwickelnde Prozedur bereits für alle einfacheren Fälle funktioniert (selbst wenn man noch nichts aufgeschrieben hat), und sie uns für diese Fälle immer eine Lösung liefert. Dann versucht man, mit Hilfe dieser von der Prozedur gelieferten Lösung eine Lösung für einen komplexeren Fall zu finden. Und genau dies ist dann die Prozedur selbst!

  (a) Nehmen wir an, dass wir bereits x^{n-1} kennen (unser Wunder der Rekursion), dann erhalten wir x^n, indem wir $x \cdot x^{n-1}$ bilden.
  $x^n = x \cdot x^{n-1}$

  ```
  ...
  else x * power(n-1)
  ```

  (b) Kennen wir schon das Ergebnis der Multiplikation einer kleineren Zahl als a mit b, also $(a-1)\cdot b$, so können wir $a \cdot b$ berechnen mit:
  $a \cdot b = (a-1) \cdot b + b$ (dies gilt wegen des Distributivgesetzes)

  ```
  ...
  else mul (a-1) b + b
  ```

2 Iterativer Denkansatz

Vielen Menschen fällt es leichter zuerst iterativ zu denken, weil es eine Betrachtungsweise ist, die wir aus dem echten Leben gewohnt sind. Betrachten wir z.B. einen Turmbau.
Um einen Turm iterativ (durch Wiederholungen) aus 10 Bausteinen zu bauen, nehmen wir einen Baustein und stellen ihn auf den Boden. Dann nehmen wir einen weiteren Baustein und stellen ihn auf den ersten Baustein. Dann nehmen wir einen weiteren Baustein, und stellen ihn auf die beiden anderen Bausteine. Dies tun wir so lange, bis wir einen Turm mit 10 Bausteinen gebaut haben.
Bei der Rekursion denken wir andersherum. Wir glauben fest daran, dass wir bereits einen Turm mit 9 Bausteinen besitzen (genau diesen Turm liefert uns nämlich das Wunder der Rekursion). Und dann nehmen wir diesen Turm, und setzen (wie zuvor) einfach einen Baustein oben drauf. Und schon haben wir einen Turm aus 10 Bausteinen. Damit unser Glaube in das Wunder der Rekursion aber auch Hand und Fuß hat, müssen wir auch hier den Basisfall bedenken: Wenn wir einen Turm aus nur einem Baustein bauen wollen, dann müssen wir diesen Baustein irgendwo hinstellen. Unser Basisfall ist also der leere Turm. Wollen wir einen Turm der Höhe 0, so müssen wir keine Steine auf den Boden legen (und können hierbei keine Rekursion anwenden!).

Stellen wir Bausteine in SML nun durch 1 dar und unseren Turm durch eine Summe von Steinen, so sieht der Turm der Höhe 4 wie folgt aus:

```
1 + 1 + 1 + 1 + 0
```

Eine iterative Variante mittels *iter* könnte dann folgendermaßen aussehen:

```
fun bauen n =
  iter n 0 (fn s => 1 + s)
```

Die rekursive Variante kann so implementiert werden:

```
fun bauen n =
  if n = 0
    then 0
    else 1 + bauen(n-1)
```

Damit der Unterschied klar wird, empfiehlt es sich verkürzte Ausführungsprotokolle zu betrachten. Wir lassen die Addition explizit stehen, um den Unterschied besser zur verdeutlichen.

Iteratives *bauen***:**
$bauen\ 4 = iter\ 4\ nil\ (fn\ s => 1 + s)$
$= iter\ 3\ (1 + 0)\ (fn\ xs => 1 + s)$
$= iter\ 2\ (1 + 1 + 0)\ (fn\ xs => 1 + s)$
$= iter\ 1\ (1 + 1 + 1 + 0)\ (fn\ xs => 1 + s)$
$= iter\ 0\ (1 + 1 + 1 + 1 + 0)\ (fn\ xs => 1 + s)$
$= 1 + 1 + 1 + 1 + 0$
Man sieht: Der Turm baut sich ausgehend von 0 nach oben auf.

Rekursives *bauen***:**
$bauen\ 4 = 1 + bauen\ 3$
$= 1 + 1 + bauen\ 2$
$= 1 + 1 + 1 + bauen\ 1$
$= 1 + 1 + 1 + 1 + bauen\ 0$
$= 1 + 1 + 1 + 1 + 0$
Vergleicht man dieses Protokoll mit dem iterativen Protokoll, so sieht man, dass hier der Turm von oben nach unten aufgebaut wird. Man könnte intutitiv auch sagen, der Turm ist schon da, und wird nur Schritt für Schritt enthüllt.

Interation/Rekursion nochmal anders: Diskutieren Sie:

(a) Iteration löst Probleme durch Aufbauen (Konstruktion)

(b) Rekursion löst Probleme durch Zerteilen (Dekonstruktion)