Verarbeitung von geschachtelten Listen und von Bäumen in Scheme

Dietmar Lippold
dietmar.lippold@informatik.uni-stuttgart.de

19. Januar 2004

1 Einleitung

Bei der Verarbeitung geschachtelten Listen interessiert man sich teilweise nicht für deren Struktur, z.B. wenn man zu einer geschachtelten Liste eine flache Liste aller ihrer Atome erzeugen möchte. Dann reicht es, sich klar zu machen, daß die Elemente einer geschachtelten Liste entweder wieder Listen oder Atome sind. Das Ergebnis kann man dann durch folgende Prozedur erhalten:

```
; Die Prozedur liefert eine nicht-geschachtelte Liste aller Atome der
; uebergebenen geschachtelten Liste
(define (atomliste geschachtelte-liste)
  (cond ((null? geschachtelte-liste) '())
        ((list? (car geschachtelte-liste))
         (append (atomliste (car geschachtelte-liste))
                 (atomliste (cdr geschachtelte-liste)) ))
        (else (cons (car geschachtelte-liste)   ; (car geschachtelte-liste)
                                                ; ist ein Atom
                    (atomliste (cdr geschachtelte-liste)) ))))
```

Teilweise ist es aber erforderlich oder sinnvoll, eine geschachtelte Liste als Baum aufzufassen, z.B. wenn die Liste einen Funktions-Term aus Funktionssymbolen und Variablen oder Konstanten darstellt, den man als Baum betrachtet, oder wenn man z.B. deren Schachtelungstiefe (die Höhe des Baums) ermitteln möchte. Dann muß man klären, wie eine geschachtelte Liste als Baum darzustellen ist, welche Arten von Bäumen es gibt und wie man diese verarbeiten kann. Dies wird in diesem Text erläutert.

2 Definition von Bäumen

Beweise und weitere Erläuterungen zum folgenden Abschnitt können der Literatur, die im Abschnitt *Referenzen* am Ende dieses Textes angegebenen ist, entnommen werden. Die definierten Begriffe bzw. die Definitionen unterscheiden sich aber teilweise.

Def.: Ein **Graph** G ist ein geordnetes Paar (V, E) aus einer endlichen Menge V, Knoten (englisch *vertices*) genannt, und einer endlichen Menge E von ungeordneten oder geordneten Paaren von Elementen aus V, Kanten (englisch *edges*) genannt.

Def.: Sind die Paare aus E ungeordnet, so heißt der Graph **ungerichtet**, sind die Paare geordnet, heißt der Graph **gerichtet**.

Die folgenden Definitionen gelten für einen gerichteten Graphen (V, E):

Def.: Ein Knoten v' heißt **Nachfolger** eines Knotens v genau dann, wenn gilt: $(v, v') \in E$.

Def.: Ein Knoten v heißt **Vorgänger** eines Knotens v' genau dann, wenn gilt: $(v, v') \in E$.

Def.: Ein Knoten heißt genau dann **Quelle**, wenn er keinen Vorgänger hat.

Def.: Ein Knoten heißt genau dann **Senke**, wenn er keinen Nachfolger hat.

Def.: Ein Knoten heißt genau dann **innerer Knoten**, wenn er weder eine Senke noch eine Quelle ist.

Die folgende Definitionen gelten für beliebige Graphen:

Def.: Ein **Pfad** zu einem Graphen G ist ein nicht-leeres Tupel $< v_1, ..., v_n >$, dessen Komponenten durch Kanten verbundene Knoten sind. Genauer:

1. Für jeden Knoten v ist das Tupel $< v >$ ein Pfad. Er heißt **trivial**.
2. Für $n \geq 2$ Knoten $v_1, ..., v_n$ mit $(v_i, v_{i+1}) \in E$ für alle $i \in \{1, ..., n-1\}$ ist das Tupel $< v_1, ..., v_n >$ ein Pfad.

Dabei heißt $n - 1$ die **Länge** (ist also die Anzahl der Kanten), v_1 der **Anfangsknoten** und v_n der **Endknoten** des Pfades. Ein trivialer Pfad hat die Länge Null.

Def.: Ein **u-v-Pfad** ist ein Pfad mit dem Anfangsknoten u und dem Endknoten v.

Def.: Zwei Knoten u und v heißen **verbunden**, wenn es einen u-v-Pfad oder einen v-u-Pfad gibt.

Def.: Ein Graph heißt **zusammenhängend**, wenn je zwei seiner Knoten verbunden sind.

Def.: Für zusammenhängende Graphen bezeichnet **dist(u, v)** den Abstand (die Distanz) zweier Knoten u und v aus V im Graphen G, wobei $dist(u, v)$ definiert ist als die Länge eines kürzesten u-v-Pfades in G.

Def.: Ein Pfad $<v_1, ..., v_n>$ heißt **elementar**, wenn seine Kanten paarweise verschieden sind, d.h. wenn für beliebige $i, j \in \{1, ..., n-1\}$ mit $i \neq j$ gilt: $(v_i, v_{i+1}) \neq (v_j, v_{j+1})$

Def.: Ein Pfad heißt **Zyklus**, wenn seine Länge mindestens Eins ist und der Anfangsknoten gleich dem Endknoten ist (d.h. wenn für $<v_1, ..., v_n>$ gilt: $n >= 2$ und $v_1 = v_n$).

Def.: Wenn es zu einem Graphen G keinen Zyklus gibt, heißt G **azyklisch**.

Def.: Ein **Baum** ist ein zusammenhängender azyklischer Graph.

Für einen Graphen $G = (V, E)$ sind folgende Aussagen äquivalent:

1. G ist ein Baum.
2. G ist zusammenhängend und es gilt: $|E| = |V| - 1$
3. Je zwei Knoten aus V sind durch genau einen elementaren Pfad verbunden.

Ein Baum mit einem ausgezeichneten Knoten wird **Wurzelbaum** genannt, der ausgezeichnete Knoten heißt **Wurzel**.

Zu einem Wurzelbaum (V, E) mit Wurzel w kann auf folgende Weise der eindeutig definierte gerichtete isomorphe Wurzelbaum (V, E') erzeugt werden (bei jeder anderen Richtung der Kanten entsteht ein gerichteter Graph, der kein Baum mehr ist): Sei (v_1, v_2) ein Element aus E. Ist $dist(w, v_1) < dist(w, v_2)$, dann wird (v_1, v_2) als geordnetes Paar in E' aufgenommen, anderenfalls wird (v_2, v_1) als geordnetes Paar in E' aufgenommen.

Für jede Kante (v_1, v_2) eines so gerichteten Wurzelbaums mit der Wurzel w gilt dann: $dist(w, v_1) + 1 = dist(w, v_2)$

Bei einem so gerichteten Wurzelbaum ist für jede Kante der Vorgänger und der Nachfolger definiert und es übertragen sich entsprechend die Begriffe *Quelle* und *Senke*. Die Wurzel ist dann die einzige Quelle. Eine Senke wird **Blatt** genannt. Außer der Wurzel hat jeder Knoten genau einen Vorgänger.

Ein **geordneter Baum** ist ein Baum, bei dem die Nachfolger eines Knotens eine festgelegte Reihenfolge haben. Bei einem geordneten Baum kann man jedem Knoten eine bestimmte Position im Baum zuordnen.

Da bei der Repräsentation eines Baums durch eine Liste automatisch eine Wurzel vorhanden und die Reihenfolge der Nachfolger eines Knotens festgelegt ist, ist im folgenden immer ein wie beschrieben gerichteter, geordneter Wurzelbaum gemeint, wenn von *Baum* gesprochen wird. Die graphische Darstellung eines Baums erfolgt im weiteren von oben nach unten und von links nach rechts.

Jeder Knoten eines Baums kann eine Markierung tragen. Die Unterscheidung, ob es sich dabei um ein Symbol oder eine Zahl handelt, ist nur für die Verwendung bzw. Verarbeitung des Baums bzw. der Liste, die den Baum repräsentiert, interessant. Sie wird daher in den folgenden Abschnitten bis auf die beiden letzten Abschnitte mit Beispielprogrammen nicht vorgenommen und es wird nur von Markierungen und Symbolen gesprochen.

Als **echter Baum** wird im folgenden ein Baum bezeichnet, der kein Blatt ist, als **echter Knoten** ein Knoten (in einem Baum), der kein Blatt ist (also ein innerer Knoten oder die Wurzel ist).

3 Darstellung geschachtelter Listen

Eine allgemeine Anforderung an die Darstellung von geschachtelten Listen als Bäume besteht darin, daß jedes Atom der Liste durch genau einen Knoten des Baums, also durch dessen Position im Baum, eindeutig repräsentiert wird, jeder Knoten also insbesondere maximal ein einzelnes Atom repräsentiert.

Eine wichtige Art geschachtelter Listen sind Ausdrücke von Prozeduren erster Ordnung, die äquivalent sind zu Funktions-Termen, wie z.B. $f(g(a, b), c)$ mit den Funktionssymbolen f und g. In Scheme entspricht diesem Term der Ausdruck (f (g a b) c). Bei ihnen ist das erste Element jeder Liste und Teilliste ein Symbol. Diese Listen lassen sich als Baum darstellen, indem das jeweils erste Element einer Liste (das ein Symbol ist) als (echter) Knoten dargestellt wird und die nachfolgenden Elemente als Unterbäume dieses Knotens dargestellt werden. Das erste Symbol der äußersten Liste wird zur Wurzel des Baums. Für den obigen

Ausdruck ergibt sich also folgender Baum:

```
    f
   / \
  g   c
 / \
a   b
```

Abbildung 1: Baum eines Funktions-Terms

Natürlich gibt es aber auch geschachtelte Listen, bei denen nicht jedes erste Element einer Liste und Unterliste ein Symbol ist, z.B. folgende Liste: ((h a) b (f c)). Sie entsprechen Ausdrücken von Prozeduren höherer Ordnung (im Beispiel kann h als eine solche Prozedur aufgefaßt werden). Sie lassen sich nicht in der oben angegebenen Weise als Baum darstellen, da sonst an einem (echten) Knoten eine Liste stehen würde und die Atome dieser Liste durch keinen anderen Knoten identifiziert wären. Solche Listen lassen sich aber durch einen Baum darstellen, indem die gesamte Liste die Wurzel des Baums darstellt und jedes Element der Liste einen Unterbaum, wobei die Unterbäume rekursiv auf die gleiche Weise dargestellt werden. Für die angegebene Liste ergibt sich dann folgender Baum:

```
((h a) b (f c))
    /   |    \
(h a)   b   (f c)
 / \         / \
h   a       f   c
```

Abbildung 2: Baum eines Prozedur-Ausdruck höherer Ordnung mit Listen

In dem so entstehenden Baum können ohne Informationsverlust alle Listen an echten Knoten weggelassen werden, da ihre Elemente auch noch einmal in den Unterlisten vorhanden sind. Zum angegebenen Beispiel ergibt sich dann folgender Baum (wobei der Stern * ein Platzhalter für eine nicht vorhandene Markierung an einem Knotens ist):

```
     *
   / | \
  *  b  *
 / \   / \
h a    f c
```

Abbildung 3: Baum eines Prozedur-Ausdruck höherer Ordnung ohne Listen

Es fällt auf, daß die Symbole ausschließlich an den Blättern des Baumes stehen und daß die echten Knoten keine Symbole tragen.

Diese Art der Darstellung einer geschachtelten Liste ist aber natürlich nicht auf Listen beschränkt, die äquivalent zu Ausdrücken von Prozeduren höherer Ordnung sind. Auch Ausdrücke von Prozeduren erster Ordnung lassen sich so darstellen. Zum ersten Beispiel ergibt folgende Darstellung:

```
  (f  (g  a  b)  c)
  /       |       \
f    (g  a  b)   c
       /   |   \
     g     a     b
```

Abbildung 4: Baum eines Prozedur-Ausdrucks erster Ordnung mit Listen

Und ohne Listen an echten Knoten ergibt sich:

```
     *
   / | \
  f  *  c
    /|\
   g a b
```

Abbildung 5: Baum eines Prozedur-Ausdrucks erster Ordnung ohne Listen

Aus dieser Darstellung läßt sich die Darstellung von Abbildung 4 erzeugen, indem an jeden echten Knoten als Markierung eine Liste geschrieben wird, die zuvor aus den rekursiv erzeugten Markierungen der Nachfolger erzeugt wurde, wobei eine schon vorhandene Markierung eines Blattes unverändert übernommen wird.

Die Darstellung von Abbildung 1 läßt sich daraus erzeugen, indem jeder Stern durch die Markierung (das Symbol) des ersten Unterbaums ersetzt wird und der Unterbaum zusammen mit dessen Kante gelöscht wird. Dadurch ist klar, daß sich die beiden Darstellungen einer solchen Liste (in Beispiel Abbildung 1 und Abbildung 5) nur in der Anzahl der Nachfolgeknoten eines echten Knotens unterscheiden, und zwar genau um Eins. Insbesondere haben beide Bäume die gleiche Höhe, soweit keine nullstelligen Funktionssymbole vorkommen.

4 Repräsentation von Bäumen als Listen

Im vorigen Abschnitt wurde beschrieben, wie eine gegebene geschachtelte Liste als Baum dargestellt werden kann. Andererseits kann aber auch ein Baum gegeben sein, den man als Liste repräsentieren möchte.

Teilweise kann dies geschehen, indem das oben beschriebene Verfahren der Darstellung einer Liste einfach umgekehrt wird: Aus jedem echten Knoten wird eine Liste, aus den Nachfolgern werden die Elemente der Liste. Wenn der echte Knoten ein Symbol trägt, wird dieses vor die Nachfolger in die Liste aufgenommen. Aus dem Blättern werden Atome.

Dieses Verfahren ist aber nur dann eineindeutig, d.h. eine entstehende Liste repräsentiert nur dann eindeutig einen Baum, wenn einheitlich an allen echten Knoten entweder ein Symbol steht oder kein Symbol steht. Ansonsten kann die Liste (f a) die folgenden beiden Bäume repräsentieren:

```
  f       *
 /       / \
a       f   a
```

Abbildung 6: Mögliche Bäume einer Liste

Um diese Mehrdeutigkeit zu vermeiden und zu ermöglichen, daß an einigen echten Knoten Symbole stehen, an anderen nicht, kann man diese Bäume als Listen repräsentieren, bei denen Blätter in der Art von echten Knoten ohne Nachfolger repräsentiert werden. Ein Blatt wird dann also als eine Liste repräsentiert, die nur das Symbol des Blattes enthält. Symbole dürfen dann ausschließlich als erstes Element einer Liste auftreten. Hat die Liste weitere Elemente, handelt es sich um einen echten Knoten und bei den weiteren Elementen um die Nachfolger des Knotens. Steht an der ersten Stelle einer Liste eine Unterliste, handelt es sich ebenfalls um einen echten Knoten, der jedoch kein Symbol trägt. Der erste der beiden Bäume aus Abbildung 6 wird dementsprechend durch die Liste (f (a)), der zweite durch die Liste ((f) (a)) repräsentiert.

Schließlich ist noch zu klären, wie man geordnete Bäume, bei denen Knoten fehlen können (z.B. Suchbäume), repräsentiert, wie also die fehlenden Knoten zu repräsentieren sind. Dies kann in Übereinstimmung mit dem im vorhergehenden Absatz beschriebenen Verfahren dadurch geschehen, daß diese in der Art von echten Knoten ohne Nachfolger und ohne Symbol, also als leere Listen, repräsentiert werden.

5 Zusammenfassung: Arten von Listen und Bäumen

Im folgenden werden die drei verschiedenen Arten von geschachtelten Listen mit den zugehörigen Bäumen noch einmal zusammenfassend dargestellt.

1. Die erste Art von Listen dient der Repräsentation von Funktions-Termen, die also äquivalent sind zu Ausdrücken von Scheme-Prozeduren erster Stufe. Eine solche Liste wird hier **Term-Liste** genannt, der zugehörige Baum **Term-Baum**. Es handelt sich um spezielle Listen zur Repräsentation spezieller Bäume. In Term-Bäumen trägt jeder echte Knoten ein Symbol. Dieses wird durch das erste Element einer Liste repräsentiert. Daher muß bei Term-Listen das erste Element einer Liste immer ein Symbol sein (Symbole können aber auch unter den weiteren Elementen auftreten).

2. Bei der zweiten Art von Listen handelt es sich um beliebige Listen, die äquivalent sind zu Ausdrücken von Scheme-Prozeduren höherer Stufe. Eine solche Liste wird hier **Allgemein-Liste** genannt, der zugehörige Baum **Listen-Baum**. Bei diesen speziellen Bäumen trägt kein echter Knoten ein Symbol und die Blätter repräsentieren genau die in der Liste enthaltenen Symbole.

3. Die dritte Art von Listen dient der Repräsentation beliebiger Bäume. Eine solche Liste wird daher hier **Baum-Liste** genannt, der zugehörige Baum **Allgemein-Baum**. Es handelt sich um spezielle Listen, bei denen ein Symbol nur als erstes Element einer Liste auftreten darf (das erste Element kann aber auch eine Unterliste sein). Wenn ein Symbol vorhanden ist, markiert es den entsprechenden Knoten. Ansonsten trägt der Knoten kein Symbol.

Die drei Arten von Listen unterscheiden sich also speziell dahingehend, ob an der ersten Position einer Liste ein Symbol stehen muß (bei Term-Listen) oder nur dort stehen kann (bei Baum-Listen) und ob es, wenn es dort steht, durch ein Blatt (bei den Allgemein-Listen), durch einen echten Knoten (bei den Term-Listen) oder durch einen beliebigen Knoten (bei den Baum-Listen) repräsentiert wird.

6 Verarbeitung von Bäumen

Bei der Verarbeitung von Bäumen muß man mehrere mögliche, semantisch unterschiedliche Argumente unterscheiden, die bei den drei Arten von Listen teilweise unterschiedlich repräsentiert werden:

- Ein einzelner Baum. Dabei kann es sich handeln um
  - ein Blatt. Dies wird bei einer Term-Liste durch ein Symbol dargestellt, das nicht an erster Stelle einer Liste steht, bei einer Allgemein-Liste

durch ein Symbol an einer beliebigen Stelle und bei einer Baum-Liste durch eine Liste mit genau einem Symbol.

  – einen echten Baum bzw. einen echten Knoten. Der wird bei einer Term-Liste durch eine Liste repräsentiert, an deren erster Stelle ein Symbol steht, bei einer Allgemein-Liste durch eine beliebige Liste und bei einer Baum-Liste durch eine Liste, die nicht nur aus einem Symbol besteht (also entweder mehr als ein Element oder eine Teilliste enthält).

- Eine Liste von Bäumen. Diese wird bei allen Arten von Listen als eine Liste einzelner Bäume dargestellt.

Eine Prozedur zur Verarbeitung eines Baums muß in der Regel sowohl ein Blatt wie einen echten Baum verarbeiten können. Sie muß aber zusätzlich zu einem Baum auch auf irgendeine Weise die Liste der Nachfolger (Unterbäume) eines Knotens verarbeiten. Und während es (wie oben beschrieben) möglich ist, bei einem einzelnen Baum syntaktisch zwischen einem Blatt und einem echten Baum zu unterscheiden, ist eine syntaktische Unterscheidung zwischen einem Baum und einer Liste von Bäumen in der Regel nicht möglich. Zur Verarbeitung dieser beiden Arten von Werten (eines Baums und einer Liste von Bäumen) gibt es folgende Möglichkeiten:

1. Definition einer polymorphen, rekursiven Prozedur zur Verarbeitung beider Wertarten
2. Definition einer (evtl. lokalen) Prozedur zur Verarbeitung einer Liste von Bäumen und Aufruf dieser Prozedur mit einer Liste, die nur den übergebenen Baum enthält
3. Definition zweier eigenständiger (evtl. lokaler) rekursiver Prozeduren zur getrennten Verarbeitung beider Wertarten
4. Definition nur einer Prozedur und zwar für einen einzelnen Baum und dabei
   (a) Verarbeitung der Liste der Unterbäume durch einen `map`-Ausdruck, wobei die Elemente der entstehenden Ergebnisliste evtl. noch durch `apply` verknüpft werden
   (b) Verarbeitung der Liste der Unterbäume in einer `do`-Schleife

Es ist für die Korrektheit, Verständlichkeit und Weiterverwendbarkeit einer Prozedur von besonderer Wichtigkeit, daß man sich überlegt und dokumentiert, welche Art von Liste verarbeitet werden soll, auf welche der angegebenen Arten dies geschehen soll und welche Art von Werten dem Parameter oder den Parametern der Prozedur übergeben werden.

7 Beispiel: Tiefe eines Listen-Baums

Im folgenden wird für die Ermittlung der Tiefe eines *Listen-Baums*, der als Allgemein-Liste repräsentiert ist, nach jeder der im vorhergehenden Abschnitt genannten Arten eine Prozedur angegeben.

Die Tiefe eines Blatts ist dabei als Eins definiert. Die Tiefe eines echten Baums ist als das Maximum der Tiefe der Unterbäume plus Eins definiert.

Um die Prozeduren sowohl für Bäume (bzw. Listen) aus Symbolen wie für solche aus Zahlen anwenden zu können, wird eine Prozedur `blatt?` erstellt. Wenn im Baum (bzw. in der Liste) nur Symbole auftreten, kann anstatt dessen in den Prozeduren auch die Prozedur `symbol?` benutzt werden.

```
(define (blatt? baum)
  (not (list? baum)) )
```

1. Definition einer polymorphen, rekursiven Prozedur zur Verarbeitung beider Wertarten

   Die Tiefe einer Liste von Bäumen wird als Maximum der Tiefe der Bäume plus Eins festgelegt, so, als würde es sich bei der Liste um einen eigenen Baum mit den Elementen der Liste als dessen Unterbäume handeln.

   ```
   (define (tiefe baum-oder-baumliste)
     (cond ((null? baum-oder-baumliste) 0)                    ; leere Baumliste
           ((blatt? baum-oder-baumliste) 1)                   ; Blatt
           (#t (max (+ 1 (tiefe (car baum-oder-baumliste)))   ; Verarbeitung vom
                                                              ; Unterbaum
                    (tiefe (cdr baum-oder-baumliste))))))     ; Verarbeitung von
                                                              ; Restliste
   ```

2. Definition einer lokalen Prozedur zur Verarbeitung einer Liste von Bäumen und Aufruf dieser Prozedur mit einer Liste, die nur den übergebenen Baum enthält

   Die Tiefe einer Liste von Bäumen wird als Maximum der Tiefe der Bäume festgelegt. Die Tiefe eines Baums kann dadurch zum einen (von außerhalb der lokalen Prozedur aufgerufen) als Tiefe einer Liste, die nur diesen Baum enthält, bestimmt werden. Zum anderen (von innerhalb der lokalen Prozedur aufgerufen) ist es (wie in der Definition angegeben) die um Eins erhöhte Tiefe der Unterbäume (wobei der Baum als Liste von Unterbäumen aufgefaßt werden kann).

```
(define (tiefe baum)
  (define (listen-tiefe baumliste)
    (cond ((null? baumliste) 0)
          ((blatt? (car baumliste))
           (max 1 (listen-tiefe (cdr baumliste))) )  ; ein Blatt hat Tiefe 1
          (else (+ 1 (max (listen-tiefe (car baumliste))  ; Tiefe von Liste
                                                          ; der Unterbäume
                          (listen-tiefe (cdr baumliste)) )))))  ; Tiefe von
                                                                ; Restliste
  (listen-tiefe (list baum)) )
```

3. Definition zweier eigenständiger lokaler rekursiver Prozeduren zur getrennten Verarbeitung beider Wertarten

   Die Tiefe einer Liste von Bäumen wird als Maximum der Tiefe der Bäume festgelegt. Die Tiefe eines einzelnen Baums ist die um Eins erhöhte Tiefe der Unterbäume (wobei der Baum als Liste von Unterbäumen aufgefaßt werden kann).

```
(define (tiefe baum)

  (define (baum-tiefe einzelbaum)           ; Tiefe eines einzelnen Baums
    (if (blatt? einzelbaum)
        1                                   ; ein Blatt hat Tiefe 1
        (+ 1 (listen-tiefe einzelbaum)) ))  ; Baum als Liste der Unterbäume
                                            ; aufgefaßt

  (define (listen-tiefe baumliste)          ; Tiefe einer Liste von Bäumen
    (if (null? baumliste)
        0
        (max (baum-tiefe (car baumliste))
             (listen-tiefe (cdr baumliste)) )))

  (baum-tiefe baum) )
```

4. (a) Definition nur einer Prozedur und zwar für einen einzelnen Baum und dabei Verarbeitung der Liste der Unterbäume durch einen map-Ausdruck mit Anwendung von apply

   Es gilt das Gleiche wie bei Version 3.

```
(define (tiefe baum)
  (if (blatt? baum)
      1                  ; ein Blatt hat Tiefe 1
      (+ 1 (apply max    ; Verarbeitung der Liste der Ergebnisse
                  (map tiefe baum) ))))   ; Verarbeitung der Liste der
                                          ; Unterbäume
```

   (b) Definition nur einer Prozedur und zwar für einen einzelnen Baum und dabei Verarbeitung der Liste der Unterbäume in einer do-Schleife

   Es gilt das Gleiche wie bei Version 3.

```
(define (tiefe baum)
  (if (blatt? baum)
      1              ; wenn baum ein Blatt ist, ist die Tiefe 1
                     ; sonst: Tiefe von baum als Baumliste berechnen
      (do ((restliste baum (cdr restliste))
           (listen-tiefe -1 (max listen-tiefe
                                 (tiefe (car restliste)) )))
          ((null? restliste) (+ 1 listen-tiefe)) )))
```

8 Beispiel: Liste der Blätter eines Term-Baums

Im folgenden wird für die Ermittlung der Blätter eines *Term-Baums*, der als Term-Liste repräsentiert ist, nach jeder der oben genannten Arten eine Prozedur angegeben.

Ein Blatt eines Term-Baums ist ein Element einer Term-Liste, das nicht an der ersten Stelle der Liste auftritt (dort steht das Symbol des echten Knotens) und selbst keine Liste ist (Listen sind Unterbäume).

Damit die folgenden Prozeduren Term-Bäume verarbeiten können, die sowohl Symbole wie Zahlen enthalten, wird vorweg eine Prozedur `blatt?` definiert. Wenn im Term-Baum (bzw. in der Term-Liste) nur Symbole auftreten, kann anstatt dessen in den Prozeduren auch die Prozedur `symbol?` benutzt werden.

```
(define (blatt? baum)
  (not (list? baum)) )
```

1. Definition einer polymorphen, rekursiven Prozedur zur Verarbeitung beider Wertarten

   Eine solche Prozedur läßt sich nicht ohne eine (lokale) Hilfsprozedur realisieren, da die Prozedur zwischen einem Baum und einer Liste von Bäumen unterscheiden muß, da bei einem als Term-Liste repräsentierten Baum das erste Symbol einen echten Knoten bezeichnet und daher weggelassen werden muß. In einer Termliste kann aber auch an erster Stelle ein Symbol stehen, das dann ein Blatt repräsentiert, das in die Ergebnisliste aufgenommen werden muß.

   Die lokale Prozedur kann nun entweder wie bei der 3. Version speziell dazu dienen, eine Baumliste zu verarbeiten. Oder sie kann dazu dienen, die Blätter einer Baumliste zumindest am Anfang der Liste vor dem rekursiven Aufruf der Prozedur zu entfernen. Eine Termliste, die bei einem rekursiven

Aufruf übergeben wird, beginnt dann also niemals mit einem Blatt (und damit auch nicht mit einem Symbol). Dieses Verfahren führt aber zu einer recht langen Prozedur, die im Vergleich mit den anderen Versionen nicht sinnvoll erscheint.

2. Definition einer lokalen Prozedur zur Verarbeitung einer Liste von Bäumen und Aufruf dieser Prozedur mit einer Liste, die nur den übergebenen Baum enthält

```
(define (termblaetter baum)

  (define (listen-blaetter baumliste)
    (cond ((null? baumliste) '())

          ; Testen, ob das erste Element von baumliste ein Blatt ist
          ((blatt? (car baumliste))
           (cons (car baumliste) (listen-blaetter (cdr baumliste))) )

          ; Wenn das erste Element eine Liste und d.h. ein Baum ist, wird
          ; die Prozedur auf die Liste seiner Unterbäume angewandt, d.h. auf
          ; die Liste ohne das Symbol des Knotens, und daran wird die Liste
          ; der Blaetter der restlichen Baeume der Liste angehaengt
          (else (append (listen-blaetter (cdar baumliste))
                        (listen-blaetter (cdr baumliste)) ))))

  (listen-blaetter (list baum)) )
```

3. Definition zweier eigenständiger lokaler rekursiver Prozeduren zur getrennten Verarbeitung beider Wertarten

```
(define (termblaetter baum)

  (define (baum-blaetter einzelbaum)
    (if (blatt? einzelbaum)
        (list einzelbaum)
        (listen-blaetter (cdr einzelbaum)) ))

  (define (listen-blaetter baumliste)
    (if (null? baumliste)
        '()
        (append (baum-blaetter (car baumliste))
                (listen-blaetter (cdr baumliste)) )))

  (baum-blaetter baum) )
```

4. (a) Definition nur einer Prozedur und zwar für einen einzelnen Baum und dabei Verarbeitung der Liste der Unterbäume durch einen map-Ausdruck mit Anwendung von apply

```
(define (termblaetter baum)
  (if (blatt? baum)
      (list baum)
      (apply append   ; Aneinanderhaengen der Teilergebnisse
             (map termblaetter (cdr baum))) ))   ; Erzeugen der Ergebnisse
                                                 ; fuer die Unterbaeume
```

(b) Definition nur einer Prozedur und zwar für einen einzelnen Baum und dabei Verarbeitung der Liste der Unterbäume in einer do-Schleife

```
(define (termblaetter baum)
  (if (blatt? baum)
      (list baum)
      (do ((restliste (cdr baum) (cdr restliste))   ; restliste ist die
                                                    ; Liste der Unterbäume
           (ergebnisliste '() (append ergebnisliste
                                      (termblaetter (car restliste)) )))
          ((null? restliste) ergebnisliste) )))
```

9 Referenzen

Einführung in Graphen und Algorithmen, Kapitel 1: *Einführung.* Stefan Hougardy, Bernd Kreuter, Hans Jürgen Prömel, Angelika Steger. 1996.
`http://www.informatik.hu-berlin.de/Institut/struktur/algorithmen/ga/`

Skript: Programmierung, Kapitel 5: *Graphen und Bäume.* Gert Smolka, Wintersemester 2000.
`http://www.ps.uni-sb.de/courses/prog-ws00/skript/5.ps`

Skript: Modellierung, Kapitel 4: *Modellieren mit Graphen.* U. Kastens, Wintersemester 2001/2002.
`http://www.upb.de/cs/ag-kastens/model/folien/inhalt.html`
`http://www.upb.de/cs/ag-kastens/model/folien/comment401-415.2.pdf`