Technische Universität München
Fakultät für Informatik
Dr. H. Ehler, Dr. M. Pizka,
W. Prenninger, Dr. M. Schneider

WS 2001
Lösungsvorschläge zu Blatt 9
13. Dezember 2001

Übungen zu Einführung in die Informatik I

Aufgabe 36 Hilfsfunktionen zum Bearbeiten von Textsequenzen (Lösungsvorschlag)

a) Die Funktion cutBlank entfernt alle Leerzeichen am Beginn einer Zeichensequenz. z.B. cutBlank(<␣␣␣ab␣c>) = <ab␣c>.

```
fct cutBlank = (seq char str) seq char:
    if str ?= empty then empty
    elif not(first(str) ?= ␣) then str
                              else cutBlank(rest(str))
    fi
```

b) Die Funktion firstWord liefert das erste Wort eines Textes zurück. z.B. firstWord(<ab␣␣cdef␣g>) = <ab>.

```
fct firstWord = (seq char str) seq char:
    if str ?= empty then empty
    elif first(str) ?= ␣ then empty
                         else conc(make(first(str)), firstWord(rest(str))
    fi
```

c) Die Funktion restString liefert von einem Text den Text ohne dem ersten Wort. z.B. restString(<ab␣␣cdef␣g>) = <cdef␣g>.

```
fct restString = (seq char str) seq char:
    if str ?= empty then empty
    elif first(str) ?= ␣ then cutBlank(rest(str))
                         else restString(rest(str))
    fi
```

Aufgabe 37 Hilfsfunktionen zum Bearbeiten von Sequenzen von Zeichensequenzen (Lösungsvorschlag)

Zunächst definieren wir die Konstante *lineLength* als nullstellige Funktion:

```
fct lineLength = nat:
    lineLength = 32
```

Weiterhin benötigen wir die Hilfsfunktionen sizeOf und connect, die die Anzahl der Elemente einer Sequenz bestimmt bzw. die Elemente einer Sequenz von Sequenzen zu einer Sequenz verbindet. z.B. sizeOf(<ab␣␣cdef␣g>) = 10, connect(<<ab><cdef><g>>) = <abcdefg>. (Bemerkung: sizeOf und connect entsprechen den Goferfunktionen `length` bzw. `concat`.)

```
fct sizeOf = (seq m seq)nat:
    if seq ?= empty then 0
                    else 1 + sizeOf(rest(seq))
    fi

fct connect = (seq (seq m) seq)seq m:
    if seq ?= empty then empty
                    else conc( first(seq), connect(rest(seq)))
    fi
```

Im Folgenden bezeichnet **string** eine Abkürzung für die Sorte **seq char**.

a) Die Funktion string2word extrahiert alle Wörter aus einem Text und liefert diese als Sequenz von Wörtern. z.B. string2word(<ab␣␣cdef␣g>) = <<ab><cdef><g>>.

```
fct string2word = (string str) seq string:
    if str ?= empty then empty
    else conc(make(firstWord(str)), string2word (restString(str)))
    fi
```

b) Die Funktion createLine ermittelt zu einer Sequenz von Wörtern in Abhängigkeit der Zeilenlänge die ersten n Wörter als Sequenz, die (inklusive Leerzeichen) in die Zeile passen.

```
fct createLine = (seq string seq) seq string:
    build( empty, seq)

fct build = (seq string seq1, seq string seq2 ) seq string:
    if seq2 ?= empty then seq1
    elsif (sizeOf( connect(seq1)) + sizeOf(seq1) + sizeOf(first(seq2))) > lineLength then seq1
    else build(conc(seq1, make(first(seq2))), rest(seq2))
    fi
```

Eine alternative Implementierung von createLine ohne Einbettung mit build:

```
fct createLine = (seq string seq) seq string:
    if (sizeOf(connect(seq)) + sizeOf(seq) - 1) ≤ lineLength then seq
    else createLine(lrest(seq))
    fi
```

c) Die Funktion delNwords entfernt aus einer Sequenz von Wörtern die ersten n Wörter.

```
fct delNwords = (nat n, seq string seq) seq string:
    if n ?= 0 then seq
    else delNwords( (n-1), rest(seq))
    fi
```

Aufgabe 38 Formatierung von Textsequenzen (Lösungsvorschlag)

a) Wir teilen die zu implementierende Funktion `flushleft` in folgende Hilfsfunktionen auf:

- `flushleft`: Formatiert den Text linksbündig. Argument der Funktion ist die zu formatierende Textsequenz, zurückgegeben wird die formatierte Textsequenz.

  Lösungsidee: Lösche führende Leerzeichen, transformiere die Textsequenz in eine Sequenz von Wörtern, wende die Funktion `left` auf das Resultat dieser Operation an und wandle die sich ergebende Sequenz von Wörtern in eine Sequenz von Zeichen um.

- `left`: Fügt am Ende jeder Zeile (mit Ausnahme der letzten) den String für den Zeilenumbruch `"\n"` ein. Hierbei wird der Funktion eine Sequenz von Zeichensequenzen übergeben, die die Worte des zu formatierenden Textes enthält. Zurückgegeben wird eine Sequenz von Zeichensequenzen, bei der nach der Zeichensequenz, die das letzte Wort einer Zeile bildet, das Element `"\n"` eingefügt wird.

  Lösungsidee: Ist die Sequenz leer, so wird die Sequenz unverändert zurückgegeben. Ansonsten werden die Worte der ersten Zeile bestimmt und diese Zeile mit Hilfe von `leftLine` linksbündig formatiert; sodann konkateniert man diese Sequenz mit der einelementigen Sequenz make(`"\n"`) und der Sequenz, die man erhält, wenn `left` auf den noch nicht formatierten Anteil angewendet wird.

- `leftLine`: Fügt zwischen den Wörtern, die in einer Zeile enthalten sind, jeweils ein Leerzeichen ein, formatiert also eine Zeile. Die Wörter werden dabei als Sequenz von Zeichensequenzen übergeben und eine Leerzeichen am Ende jedes Elementes dieser Sequenz eingefügt.

  Lösungsidee: Bei einer leeren oder einelementigen Liste wird diese unverändert zurückgegeben. Andernfalls fügt man am Ende des ersten Wortes ein Leerzeichen ein und wendet das Verfahren auf den Rest der Liste an.

Die Implementierung der Funktionen in Pseudocode hat folgende Gestalt:

- **fct** flushleft = (**string** seq) **string**:
  connect (left (string2word(cutBlank (seq))))

- **fct** left = (**seq string** seq) **seq string**:
  if (seq $\stackrel{?}{=}$ empty) **then** empty
  else conc (conc(leftLine (createLine(seq)), make(`"\n"`)),
  left (delNwords ((sizeOf (createLine(seq))), seq))
)
  fi

- **fct** leftLine = (**seq string** seq) **seq string**:
  if (seq $\stackrel{?}{=}$ empty) **then** empty
  elif sizeOf (seq) $\stackrel{?}{=}$ 1 **then** seq
  else conc (make(conc(first(seq), make(␣))), leftLine(rest(seq)))
  fi

b) Die Funktionen zur Formatierung des Textes in Blocksatz `blocksatz`, `block` und `blockLine` entsprechen jeweils den Funktionen zur linksbündigen Formatierung `flushleft`, `left` und `leftLine`. In Pseudocode lauten diese Funktionen wie folgt:

- **fct** blocksatz = (**string** seq) **string**:
  connect (block (string2word(cutBlank(seq))))

- **fct** block = (**seq string** seq) **seq string**:

  if (seq $\stackrel{?}{=}$ empty) **then** empty
  else conc(conc(blockLine (createLine (seq)), make(“\n”)),
  block (delNwords ((sizeOf (createLine (seq))), seq))
)
  fi

- **fct** blockLine = (**seq string** seq) **seq string**:

  if (sizeOf (connect seq) $\stackrel{?}{=}$ lineLength) **then** seq
  else blockLine (addSpace (seq, lineLength))
  fi

- Die Funktion `addSpace` ist eine Hilfsfunktion von `blockline`. Sie fügt in einer Zeile am Ende jedes Wortes (außer dem letzten) eine Leerzeichen ein, sofern die Zeilenlänge noch nicht überschritten ist.

  fct addSpace = (**seq string** seq, **int** z) **seq string**:

  if (seq $\stackrel{?}{=}$ empty) **then** empty
  elif rest (seq) $\stackrel{?}{=}$ empty **then** seq
  elif sizeOf (connect seq) $\stackrel{?}{=}$ z **then** seq
  else conc(conc (first (seq), make(␣)),
  addSpace (rest (seq), (z - (sizeOf (first seq) +1)))
)
  fi

Aufgabe 39 Implementierung der Funktionen und Erweiterung (Lösungsvorschlag)

a) Eine Implementierung der bisher erarbeiteten Funktionen besitzt folgende Gestalt:

```
-- zeilenlaenge

lineLength :: Int
lineLength = 32

-- Aufgabe 36

firstWord :: String -> String
firstWord [] = []
firstWord (x:xs) | (x == ' ') = []
                 | otherwise = x : (firstWord xs)

cutBlank :: String -> String
cutBlank [] = []
cutBlank (x:xs) | (x /= ' ')  = (x:xs)
                | otherwise  = cutBlank xs

restString :: String -> String
restString [] = []
restString (x:xs) | (x == ' ')  = cutBlank xs
                  | otherwise  = restString xs

-- Aufgabe 37

string2word :: String -> [String]
string2word [] = []
string2word xs = (firstWord xs) : (string2word (restString xs))

connect :: [[a]] -> [a]
connect [] = []
connect (x:xs) = x ++ (connect xs)

sizeOf :: [a] -> Int
sizeOf [] = 0
sizeOf (x:xs) = 1 + sizeOf xs

createLine :: [String] -> [String]
createLine xs = build [] xs

build :: [String] -> [String] -> [String]
build xs ys | (ys == []) = xs
            | (lol > lineLength) = xs
            | (lol <= lineLength) = build (xs ++ [head ys]) (tail ys)
              where lol = sizeOf (connect (xs)) + sizeOf (xs)
                          + sizeOf(head ys)
```

```
{- --ineffiziente Loesung von createLine ohne Einbettung
createLine :: [String] -> [String]
createLine xs | (sizeOf (connect (xs)) + sizeOf (xs) - 1 <= lineLength) = xs
              | otherwise = createLine (init xs)
-}

delNwords :: Int -> [String] -> [String]
delNwords n xs | n == 0 = xs
               | otherwise = delNwords (n-1) (tail xs)

-- Aufgabe 38

-- linksbuendiges Formatieren

flushleft :: String -> String
flushleft xs = connect (left (string2word (cutBlank xs)))

left :: [String] -> [String]
left xs | (xs == []) = []
        | otherwise = (leftLine line) ++ ["\n"]
                          ++ left (delNwords (sizeOf line) xs)
          where line = createLine xs

leftLine :: [String] -> [String]
leftLine [] = []
leftLine (x:xs) | (sizeOf (x:xs) == 1) = (x:xs)
                | otherwise = (x++" ") : (leftLine xs)

-- Blocksatzformatierung

blocksatz :: String -> String
blocksatz xs = connect (block (string2word (cutBlank xs)))

block :: [String] -> [String]
block xs  | (xs == []) = []
          | otherwise = (blockline line) ++ ["\n"]
                           ++ block (delNwords (sizeOf line) xs)
            where line = createLine xs

blockline :: [String] -> [String]
blockline xs | (sizeOf( connect xs ) == lineLength) = xs
             | otherwise = blockline (addSpace xs lineLength)

addSpace :: [String] -> Int -> [String]
addSpace [] z = []
addSpace (x:xs) z | (xs == []) = (x:xs)
           | (sizeOf (connect (x:xs)) == z) = (x:xs)
           | otherwise = (x++" ") : (addSpace xs (z - (sizeOf x + 1)))
```

b) Eine zentrierte Formatierung ist unter Verwendung folgender Funktionen möglich:

```
centered :: String -> String
centered xs = connect (center (string2word (cutBlank xs)))

center :: [String] -> [String]
center xs | (xs == []) = []
          | otherwise = (centerline line) ++ ["\n"]
                        ++ center (delNwords (sizeOf line) xs)
            where line = createLine xs

centerline :: [String] -> [String]
centerline xs = addFront (leftLine xs)

addFront :: [String] -> [String]
addFront [] = []
addFront (x:xs) | (sizeOf (connect (x:xs)) == lineLength) = (x:xs)
                | otherwise = addBack ((' ':x) : xs)

addBack :: [String] -> [String]
addBack [] = []
addBack xs | (sizeOf (connect xs) == lineLength) = xs
           | otherwise = addFront ( (init xs)++[(last xs) ++ " "] )
```

Die Funktion zur Auswahl einer der drei Formatierungsmethoden lautet:

```
formatiere :: ([String] -> [String]) -> String -> String
formatiere satz xs = connect (satz (string2word (cutBlank xs)))
```

Aufgabe 40 Induktion über den Termaufbau von Sequenzen (Lösungsvorschlag)

Wir definieren als spezielles Prädikat

$$P(t) \stackrel{\text{def}}{=} (f(f(t)) = t)$$

Induktionsbeginn (IB):
Zu zeigen sind die Teile (a) und (b) aus dem Induktionsprinzip für unser spezielles P.

zu (a): $f(f(empty)) = f(empty) = empty$ wegen (1).

zu (b): $f(f(make(s))) = f(make(s)) = make(s)$ wegen (2) für beliebige Terme s der Sorte **m**.

Induktionsschritt (IS):
Zu zeigen ist Teil (c) des Induktionsprinzips für unser spezielles P.
Als Induktionshypothese (IH) gelte $P(t_1), P(t_2)$ für Terme t_1, t_2 der Sorte **seq m**, d.h. $f(f(t_1)) = t_1$ und $f(f(t_2)) = t_2$.
Es gilt $f(f(conc(t_1,t_2))) = f(conc(f(t_2), f(t_1))) = conc(f(f(t_1)), f(f(t_2))) = conc(t_1,t_2)$ wegen (3) und der IH.

Somit gilt $P(t)$ für alle Sequenzen t nach dem gegebenen Induktionsprinzip. f revertiert Sequenzen, z.B. $f(< sualokin >) =< nikolaus >$.