

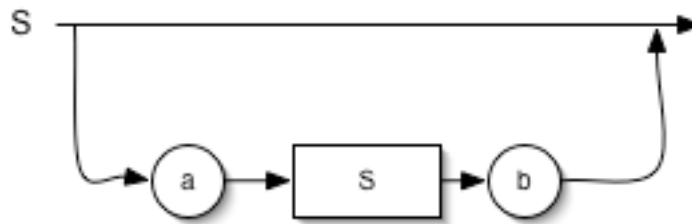
Algorithmen und Programmierung

4. Aufgabenblatt

Aufgabe 4.1

a) Durch die Regel $S \rightarrow aSb$ können beliebig viele a's und b's konkateniert werden, wobei die Anzahl der a's und b's übereinstimmt. Die erzeugte Sprache ist somit: $a^n b^n$ mit $n \geq 0, n \in \mathbb{N}$.

b) Syntaxdiagramm der Sprache $a^n b^n$



b) Eine Grammatik der Sprache $a^n b^m$:

$S \rightarrow AB$

$A \rightarrow a \mid aA$

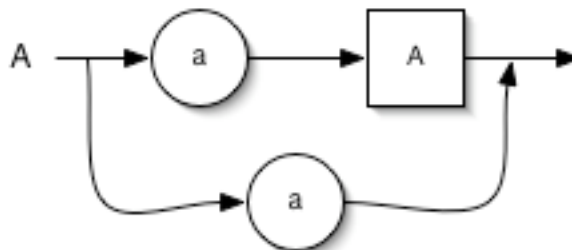
$B \rightarrow b \mid bB$

Syntaxdiagramm der Sprache $a^n b^n$

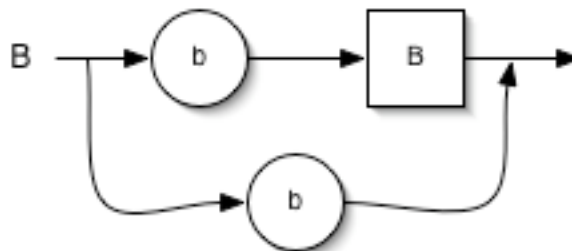
$S \rightarrow AB$



$A \rightarrow aA \mid a$



$B \rightarrow bB \mid b$



Aufgabe 4.2

Implementierung

```
type Point = (Float, Float)
type Line = (Point, Point)

-- Gerade heißt akzeptabel, wenn  $x_1 \leq x_2$ 
akzeptabel :: Line -> Bool
akzeptabel ((x1,y1),(x2,y2)) = x1<=x2

-- Gerade ist senkrecht zur x-Achse, wenn
--  $x_1 == x_2$  und Gerade kein Punkt ist ( $y_1 \neq y_2$ )
vertikal :: Line -> Bool
vertikal ((x1,y1),(x2,y2)) = x1==x2 && y1/=y2

-- Die Gleichung  $(y-y_1) / (x-x_1) = (y_2-y_1) / (x_2-x_1)$ 
-- nach y umgestellt
gibYzuX :: Float -> Line -> Float
gibYzuX x ((x1,y1),(x2,y2))
  | not (akzeptabel ((x1,y1),(x2,y2)))
    = error "keine Gerade"
  | (vertikal ((x1,y1),(x2,y2))) = error "Gerade vertikal"
  | otherwise = (y2-y1)*(x-x1)/(x2-x1)+y1
```

Testläufe

```
*Programme> akzeptabel ((10,1),(1,1))
False
*Programme> akzeptabel ((1,1),(10,1))
True

*Programme> vertikal ((10,1),(10,2))
True
*Programme> vertikal ((1,1),(10,2))
False

*Programme> gibYzuX 2 ((1,1),(3,3))
2.0
*Programme> gibYzuX 2 ((3,1),(3,3))
*** Exception: Gerade vertikal
*Programme> gibYzuX 2 ((4,2),(3,3))
*** Exception: keine Gerade
```

Aufgabe 4.3

Implementierung eines Volladdierers mit Hilfe von 2 Volladdierern

```
data Bit = 0 | 1 deriving Show

-- Boolesches OR
bOR 0 0 = 0
bOR 0 1 = 1
```

```

bOR I O = I
bOR I I = I

-- Boolesches AND
bAND O O = O
bAND O I = O
bAND I O = O
bAND I I = I

-- Boolesches XOR
bXOR O O = O
bXOR O I = I
bXOR I O = I
bXOR I I = O

-- Halbaddierer mit Ausgabe (Summe, Übertrag)
bHalfAdder x y = (bXOR x y, bAND x y)

-- Volladdierer mit Ausgabe (Summe, Übertrag)
bVollAdder x y cin = (s2,cout)
                    where
                        (s,c) = bHalfAdder x y
                        (s2,c2) = bHalfAdder s cin
                        cout = bOR c2 c

```

Testläufe

```

*Programme> bHalfAdder O I
(I,O)
*Programme> bHalfAdder I I
(O,I)
*Programme> bVollAdder O O I
(I,O)
*Programme> bVollAdder I I I
(I,I)

```

Aufgabe 4.4

a) Haskell Code

```

-- Anzahl der Elemente der Argumentenliste
elemN :: [a] -> Int
elemN [] = 0
elemN (x:xs) = 1 + elemN xs

-- alle Duplikate entfernen mit Hilfe von
-- elem aus der Prelude
unique :: Eq a => [a] -> [a]
unique [] = []
unique (x:xs) | x `elem` xs = unique xs
              | otherwise = x : unique xs

-- Variante von unique
unique2 [] = []
unique2 (x:xs) = x:(unique[y|y<-xs,y/=x])

```

```
-- Anzahl verschiedener Elemente
-- delem stützt sich auf unique ab
delem xs = length (unique xs)
```

Testläufe

```
*Programme> elemN [1..10]
10
*Programme> unique ([1..10]++[1..10])
[1,2,3,4,5,6,7,8,9,10]
*Programme> delem2 ([1..10]++[1..10])
10
```

b) Insertion Sort in Haskell

```
-- Füge i in die sortierte(!) Ergebnisliste ein
insert :: Ord a => a -> [a] -> [a]
insert i [] = [i]
insert i (x:xs)
    | i <= x      = i : (x : xs)
    | otherwise   = x : (insert i xs)

-- Entnimm ein Element der Argumentenliste und
-- füge es in die Ergebnisliste ein
insSort :: Ord a => [a] -> [a]
insSort [] = []
insSort (x:xs) = insert x (insSort xs)
```

Testläufe

```
*Programme> iSort [10,9,8,7,6,5,4,3,2,1]
[1,2,3,4,5,6,7,8,9,10]
```

b) Auswertung von **innen nach außen** und von **außen nach innen**

```
xs = [1,2, (3*4), 2]
length [] = 0; length(x:xs) = 1 + length xs
```

Auswertung: von außen nach innen

```
length [1,2,(3*4),2]
= 1 + length [2,(3*4),2]
= 1 + (1+length [(3*4),2] )
= 1 + ( 1 + ( 1 + length [2] ) )
= 1 + ( 1 + ( 1 + ( 1+length [])))
= 1 + ( 1 + ( 1 + ( 1+ 0 ) ) )
= 1 + ( 1 + ( 1 + 1 ) )
= 1 + ( 1 + 2 )
= 1 + 3
= 4
```

Von innen nach außen:

```
length [1,2,(3*4),2]
= length [1,2,12,2]
= 1 + length [2,12,2]
= 1 + (1+length [(12,2) ] )
= 1 + ( 1 + ( 1 + length [2] ) )
= 1 + ( 1 + ( 1+( 1+length [])))
= 1 + ( 1 + ( 1 + ( 1+ 0 ) ) )
= 1 + ( 1 + ( 1 + 1 ) )
= 1 + ( 1 + 2 )
= 1 + 3
= 4
```