

# Algorithmen und Programmierung I

WS 2004 / 2005

## Übung 4 (25 Punkte)

Ausgabe: Mo. 15.11.2004

Abgabe: Mi. 24.11.2004, 10.00

### Verspätet abgegebene Lösungen werden nicht akzeptiert.

Die Abgabe erfolgt in Zweiergruppen. Die Gruppe darf nur mit ausdrücklicher Genehmigung des Tutors gewechselt werden.

Abgegeben werden müssen

- Alle Lösungen einmal ausgedruckt. **Handschriftliche Lösungen werden nicht akzeptiert.**
- Programme, Testdaten, Testergebnisse in elektronischer Form in Absprache mit den Tutorinnen und Tutoren.

### Aufgabe 4.1 (5 Punkte)

a) Gegeben die Formale Sprache  $L = (A, \Sigma, R, \sigma)$  mit:

$A = \{a, b\}$ ,  $\Sigma = \{\sigma\}$ , Startsymbol  $\sigma$ ,  $R = \{\sigma \rightarrow a\sigma b, \sigma \rightarrow \varepsilon\}$ ,  $\varepsilon =$  leeres Wort

Welche Sprache wird von dieser Grammatik erzeugt?

b) Schreiben Sie die Grammatik als Syntaxdiagramm

c) Gesucht sind eine Grammatik und ein Syntaxdiagramm für die Sprache, die die Wörter  $a^n b^m$  enthält, wobei  $n, m > 0$ .

### Aufgabe 4.2 (5 Punkte)

Punkte der Ebene werden durch einen `type Point = (Float, Float)`, Geraden durch zwei Punkte `type Line = (Point, Point)` dargestellt. Eine Gerade  $((x_1, y_1), (x_2, y_2))$  soll *akzeptabel* heißen, wenn  $x_1 \leq x_2$ .

a) Schreiben Sie eine Funktion, die testet, ob eine Gerade akzeptabel ist.

b) Schreiben Sie eine Funktion, die prüft, ob eine Gerade senkrecht zur x-Achse (vertikal) verläuft.

c) Eine durch  $((x_1, y_1), (x_2, y_2))$  gegebene nicht vertikale Gerade wird durch die Gleichung  $(y - y_1) / (x - x_1) = (y_2 - y_1) / (x_2 - x_1)$  beschrieben. Definieren Sie eine Funktion, die für eine gegebene Gerade zu einem x-Wert den zugehörigen y-Wert berechnet. (Signaturen angeben!)

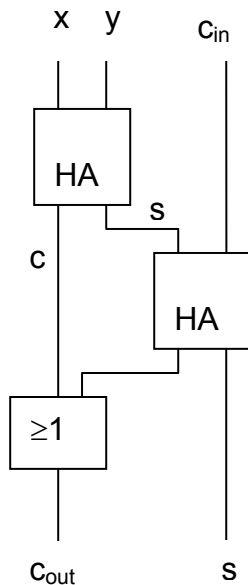
### Aufgabe 4.3 (5 Punkte)

Es soll ein 1-Bit Volladdierer implementiert werden. Das ist eine Schaltfunktion mit 3 Booleschen Eingängen  $x$ ,  $y$  und  $c$  und zwei Ausgängen  $s$  und  $c'$ . Zwei binäre Zahlen  $b_n \dots b_0$  und  $b'_n \dots b'_0$  werden stellenweise addiert. Volladdierer werden  $m$  einfachsten aus zwei Halbaddierern zusammengesetzt, die zwei Eingänge  $x$  und  $y$  und zwei Ausgänge  $s$  (Summe) und  $c$  (Übertrag) gemäß folgender Schaltfunktionen:

$$s = x \otimes y \quad (\text{xor})$$

$$c = x \wedge y$$

Beim Volladdierer wird der Übertrag einer Stelle an die nächste Stelle weitergegeben. Das erklärt, warum ein Volladdierer drei Eingänge hat. Das Schaltbild eines aus zwei Halbaddierern zusammengesetzten 1-Bit-Volladdierer sieht man in der folgenden Abbildung.



Bits sind die Konstanten `I` und `O`. Damit sie nicht als Buchstaben interpretiert werden und ausgegeben werden können, muss Ihr Programm die Datendefinition `data Bit = I | O deriving Show` enthalten. Ferner müssen die elementaren Operationen `or`, `and`, `xor` implementiert werden, z.B.

```
bOR :: Bit -> Bit -> Bit
bOR O O = O
bOR O I = I
bOR I O = I
bOR I I = I
```

#### Aufgabe 4.4 (5 Punkte)

a) Gesucht sind die folgenden Haskell-Funktionen:

```
elemN :: [a] -> Int -- Anzahl Elemente der Argumentliste
```

```
unique :: Eq a => [a] -> [a] -- alle Duplikate entfernen
```

Spezifikation:  $x \in' (\text{unique } xs) \Leftrightarrow x \in' xs \wedge \forall y \in' (\text{unique } xs) x \neq y$   
 $\in'$  bezeichnet das Enthaltensein in einer Liste

```
dElem :: Eq a => [a] -> Int -- Anzahl verschiedener Elemente
```

Spezifikation: ... wie man sich denken kann

b) Schreiben Sie ein Haskell-Programm, das den folgenden Sortieralgorithmus implementiert: Der Argumentliste wird jeweils ein Element entnommen und an die richtige Position (gemäß Sortierreihenfolge) der Ergebnisliste eingefügt.

```
isort :: Ord a => [a] -> [a] -- insertion sort
```

c) *Auswertung von Ausdrücken*: Reduzieren Sie den Ausdruck `length [1,2, (3*4), 2]` mit der Definition: `length [] = 0; length (x:xs) = 1 + length xs`. Reduzieren Sie von links nach rechts und von außen nach innen. Welchen Unterschied ergibt die Reduktion von innen nach außen (ebenfalls von links nach rechts)?

#### Aufgabe 4.5 (5 Punkte)

a) Definieren Sie die einfachen 'Verschlüsselungsfunktionen' `caesar1` und `unCaesar` die alle Zeichen einer Zeichenkette um einen festen Wert `n` in der Folge der ASCII-Zeichen verschieben. `n` ist eine natürliche Zahl und soll in beiden Funktionen der erste Parameter sein. Es soll gelten:

```
caesar n . unCaesar n = id -- id : Identität.
```

b) Nichtdruckbare Zeichen sollen ignoriert werden. Es werden nur Zeichen zwischen den Positionen 32 (Leerzeichen) und 255 des ASCII-Zeichensatzes verschlüsselt. Andere Zeichen (z.B. Zeilenumbruch) bleiben unverändert.

Die Funktionen `fromEnum :: Char -> Int` und `chr :: Int -> Char` werden verwendet, um Zeichen in Zahlen und umgekehrt zu verwandeln. So hat etwa `chr 32` den Wert `' '` (Leerzeichen). Man muss jedes Zeichen in eine Zahl umwandeln, `n` addieren und mit `chr` zurück in ein Zeichen verwandeln. Die Verschiebung erfolgt natürlich zyklisch, in a) folgt dem ASCII-Zeichen mit Position 255 das mit der Position 0, im b) das mit Position 32.

Hinweis: die ``mod``-Operation eignet sich gut für den zyklischen Abschluss, da immer gilt  $0 \leq a \bmod b < b$ . Die ``mod``-Operation ist auch für negative Zahlen definiert. So gilt etwa:

```
(-255) `mod` 256 == 1
```

<sup>1</sup> Angeblich hat Caesar dieses einfache Verschlüsselungsverfahren benutzt.