Christoph Lüth
Marcus Ermler
Christian Maeder
Martin Ring
Diedrich Wolter

2. Übungsblatt

Ausgabe: 31.10.12
Abgabe: 09.11.12

In dieser Aufgabe wollen wir die Grundlagen der binären Arithmetik in Haskell implementieren.

2.1 *Back to Basics I: Bits and Bobs* — *6 Punkte*

Zuerst benötigen wir einen Datentyp `Bit`, der aus zwei disjunkten Werte besteht, mit zwei Funktionen

```
dec1 :: Bit → Int
enc1 :: Int → Bit
```

die ein einzelnes Bit in eine ganze Zahl $i \in \{0,1\}$ verwandeln.

Ein Bitliste ist eine Sequenz einzelner Bits, und ist entweder leer, oder besteht aus einem Bit, vorne an eine Bitliste angehängt.

Implementieren Sie den Datentyp `Bitlist`, und drei Funktionen

```
encode  :: Int → Bitlist
decode  :: Bitlist → Int
display :: Bitlist → String
```

welche eine ganze Zahl in eine Bitliste kodieren, eine Bitliste in eine ganze Zahl dekodieren, sowie eine Bitliste als Binärzahl anzeigen. Beispiel:

```
*> display (encode 1238923)
"100101110011110001011"
```

2.2 *Back to Basics II: It all adds up!* — *7 Punkte*

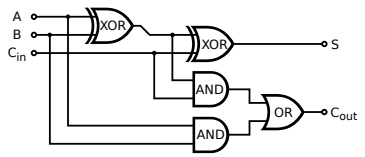
Jetzt wollen wir mit Bitlisten rechnen. Dazu benötigen wir zuerst grundlegenden boolschen Operatoren. Implementieren Sie dazu Funktionen

```
band, bor, bxor :: Bit → Bit → Bit
```

welche Konjunktion, Disjunktion und ausschließende Disjunktion auf einzelnen Bits implementieren.

Um die Addition zweier als Bitlisten repräsentierter Zahlen zu realiseren, konstruieren wir zuerst einen Volladdierer (*full adder*), der drei Bits (zwei Argumente und einen Übertrag) zu zwei Bits (einem Ergebnis und ein Übertrag) addiert. Wir implementieren den Volladdierer durch zwei Funktionen, welche die beiden Ausgänge *S* (Summe) und *C* (Übertrag) darstellen:

```
fullAdderS :: Bit → Bit → Bit → Bit
fullAdderC :: Bit → Bit → Bit → Bit
```



Quelle: `http://en.wikipedia.org/wiki/Adder_(electronics)`

Abbildung 1: Ein Volladdierer

Ein Schaltbild eines Volladdiererers, aus dem sich die Definition der Funktionen ablesen läßt, findet sich in Abb. 1.

Um zwei durch Bitlisten repräsentierte Binärzahlen zu addieren, werden rekursiv die jeweiligen Bits mit dem Übertrag der letzten Addition in einem Volladdierer addiert. Wenn beide Bitlisten leer sind, muss ein eventuell noch ausstehender Übertrag vorne angehängt werden. *Wir gehen von keinen festen Wortlängen (Länge der Bitlisten) aus!* Eine Möglichkeit wäre es, die zu addierenden Bitlisten durch vorne angefügte `0` Bits auf die gleiche Länge zu bringen, um in der rekursiven Definition von gleich langen Bitlisten ausgehen zu können.

2.3 *Back to Basics III: Many Times* — *7 Punkte*

Mithilfe der Addition können wir multiplizieren. Die Multiplikation geht von der Beobachtung aus, dass Verdoppelung und Halbierung durch einfaches Hinzufügen eine 0 als LSB[1] resp. Entfernen des LSB implementiert werden kann (*shift left* und *shift right*). Ferner können wir die Multiplikation rekursiv wie folgt formulieren:

$$0 \cdot b = 0 \quad (1)$$
$$(2 \cdot a) \cdot b = 2 \cdot (a \cdot b) \quad (2)$$
$$(2 \cdot a + 1) \cdot b = 2 \cdot (a \cdot b) + b \quad (3)$$

Wenn wir Gleichungen (2) und (3) etwas umformulieren, erhalten wir eine rekursive Definition (wobei $x \div y$ die ganzzahlige Division ist):

$$a \cdot b = \begin{cases} 2 \cdot ((a \div 2) \cdot b) & \text{für } a \text{ gerade} \\ 2 \cdot ((a \div 2) \cdot b) + b & \text{für } a \text{ ungerade} \end{cases}$$

Implementieren Sie damit eine Funktion

```
mult :: Bitlist→ Bitlist→ Bitlist
```

welche rekursiv durch Shift-Operationen und Addition die Multiplikation zweier durch Bitlisten repräsentierter Binärzahlen implementiert. Auch hier können Sie nicht von einer festen Wortlänge ausgehen.

? *Verständnisfragen*

1. Welche zusätzliche Mächtigkeit wird durch Rekursion bei algebraischen Datentypen in der Modellierung erreicht? Was läßt sich mit rekursiven Datentypen modellieren, was sich nicht durch nicht-rekursive Datentypen erreichen läßt?
2. Was ist der Unterschied zwischen Bäumen und Graphen, in Haskell modelliert?
3. Was sind die wesentlichen Gemeinsamkeiten, und was sind die wesentlichen Unterschiede zwischen algebraischen Datentypen in Haskell, und Objekten in Java?

[1] Das LSB (least significant bit) ist das geringstwertige Bit, also die letzte binäre Ziffer.