
Informatik II

Blatt 1

Abgabe: 18.5.2000

1. [5 Punkte] Betrachte folgende Prozedur:

```
(define foo
  (lambda (x n)
    (if (= n 0)
        1
        (+ (expt x n) (foo x (- n 1))))))
```

Beweise mit vollständiger Induktion, daß $(\text{foo } x \ n)$ für alle Zahlen $x \neq 1$ und natürlichen Zahlen $n \geq 0$ den Wert

$$\frac{x^{n+1} - 1}{x - 1}$$

liefert. ($(\text{expt } b \ m)$ liefert b^m .)

2. [13 Punkte] Programmiere eine Bibliothek für den Umgang mit Binärzahlen mit fester Stellenzahl und Zweierkomplement!

Anleitung:

- Definiere eine globale Variable `*n-digits*`, welche die Stellenzahl festlegt.
- Führe mit den Typabstraktionen des letzten Semesters einen neuen Typ für Binärzahlen ein. (Der Code für die Abstraktionen befinden sich bei <http://www-pu.informatik.uni-tuebingen.de/info-ii-y2k/material/types.scm>.)

Repräsentiere Binärzahlen durch Listen von Ziffern 0 oder 1. Überlege Dir, welche Reihenfolge für die Ziffern am besten geeignet ist.

- Schreibe Prozeduren `number->binary` und `binary->number`, die eine normale Zahl in eine Binärzahl und umgekehrt umwandeln. `number->binary` soll eine Fehlermeldung liefern, falls die umzuwandelnde Zahl zu groß ist. In jedem Fall sollte gelten:

```
> (binary->number (number->binary 27))
27
> (binary->number (number->binary -23))
-23
```

- Schreibe Prozeduren `and-binary`, `or-binary` und `xor-binary`, welche die entsprechenden logischen Operationen auf Binärzahlen realisieren

```
> (binary->number (or-binary (number->binary 23) (number->binary 42)))
63
> (binary->number (and-binary (number->binary 23) (number->binary 42)))
2
> (binary->number (xor-binary (number->binary 23) (number->binary 42)))
61
```

- Schreibe eine Prozedur `negate`, welche für eine Binärzahl das Vorzeichen umdreht, indem sie ihr Zweierkomplement berechnet.

```
> (binary->number (negate (number->binary 23)))
-23
```

Dabei soll `negate` eine Fehlermeldung liefern, falls die Negation nicht möglich ist.

- Schreibe Prozeduren `binary+` und `binary-`, welche zwei Binärzahlen addieren bzw. subtrahieren.

```
> (binary->number (binary+ (number->binary 23) (number->binary 42)))
65
```

Diese Prozeduren sollen direkt auf der Binärdarstellung arbeiten, also nicht auf den zurückkonvertierten „internen“ Scheme-Zahlen. Bei Überlauf sollen die Prozeduren eine Fehlermeldung signalisieren.

Entwirf und schreibe außerdem Varianten dieser Prozeduren, welche auch Übertragungsinformationen zurückgeben.

- Überlege Dir, wie Prozeduren `binary*` und `binary/` arbeiten sollten. Wie könnte sinnvoll mit dem Problem umgegangen werden, daß Multiplikation zweier Zahlen fester Länge (potentiell) eine Zahl doppelter Länge ergibt? Umgekehrt, wie kann vermieden werden, daß bei der Division Stellen „verschwendet“ werden?

3. [7 Punkte] In Scheme gibt es einen separaten Datentyp für Zeichen. Lies zunächst Abschnitt 6.3.4 im R⁵RS zu diesem Thema!

Die Prozeduren `char->integer` und `integer->char` wandeln Zeichen in Zahlen und umgekehrt um. In Verbindung mit der vorherigen Aufgabe lassen sich also auch Zeichen als Binärzahlen codieren. Beschreibe, wie `binary-xor` für eine rudimentäre Textverschlüsselung eingesetzt werden kann. Die Prozedur `encrypt` soll dabei unter Verwendung eines *Schlüssels*, also einer Liste von Binärzahlen, eine verschlüsselte Nachricht in Form einer Liste von Binärzahlen zurückgeben. Die Prozedur `decrypt` soll den Vorgang umkehren.

```
> (define key (map number->binary '(45 23 14 17)))
> (define message
  (encrypt
   key
   '(#\M #\i #\k #\e #\space #\i #\s #\t #\space #\d #\o #\o #\f)))
> message
<eine zufällig aussehende Liste von Binärzahlen>
> (decrypt key message)
(#\M #\i #\k #\e #\space #\i #\s #\t #\space #\d #\o #\o #\f)
```

Beweise dafür, daß die Verkettung von Verschlüsselung und Entschlüsselung immer den ursprünglichen Text ergibt.

Wie sicher ist diese Verschlüsselungsmethode? Wie variiert die Sicherheit mit Länge und Art des Schlüssels?

4. [5 Punkte] Der Intel Pentium besitzt eine Instruktion `daa` für „decimal adjust after addition“. Diese Instruktion dient dem Rechnen mit BCD-Zahlen. Dabei werden die BCD-Darstellungen zweier Zahlen *regulär* binär addiert (also die Tatsache ignorierend, daß es sich um BCD-Zahlen handelt). Die `daa`-Instruktion kann aber am Ergebnis der binären Addition fast ablesen, wie das BCD-Ergebnis lautet. Sie benötigt dabei lediglich Information darüber, ob ein Übertrag zwischen den Nibbles zweier nebeneinanderliegender Ziffern erfolgt ist. Beschreibe, wie `daa` arbeiten könnte!