Aufgabe 6-1

In PVS ist der Datentyp der linearen Listen über einem beliebigen Elementtyp `T` wie folgt vordefiniert (siehe auch `M-x view-prelude-file`):

```
list [T: TYPE] : DATATYPE
BEGIN
  null : null?
  cons(car:T, cdr:list) : cons?
END list
```

Wir betrachten in dieser Aufgabe lineare Listen über natürlichen Zahlen:

```
NList : TYPE = list[nat]
```

a) Definieren Sie eine rekursive Funktion `length(l:NList) : RECURSIVE nat`, die die Länge der Liste `l` berechnet.

b) Definieren Sie eine rekursive Funktion `app(l,k:NList) : RECURSIVE NList`, die zwei Listen `l` und `k` aneinanderhängt.

Zeigen Sie, dass die Länge einer mit `app` zusammengesetzten Liste der Summe der Längen der beiden Teillisten entspricht.

c) Definieren Sie eine rekursive Funktion `map(f:[nat -> nat],l:NList) : RECURSIVE NList`, die eine neue Liste erzeugt und dabei die Funktion `f` auf jedes Element der Liste `l` anwendet.

Zeigen Sie, dass die Länge einer Liste bei der Anwendung von `map` erhalten bleibt.

d) Definieren Sie eine rekursive Funktion `sum(l:NList) : RECURSIVE nat`, die die Summe der Elemente der Liste `l` errechnet.

Definieren Sie ferner eine Funktion `double(l:NList) : NList`, die aus der Liste `l` eine neue Liste erzeugt, in der der Wert jedes Elements verdoppelt ist.

Zeigen Sie, dass die Summe der Elemente der so „verdoppelten“ Liste gleich der zweifachen Summe der Elemente der ursprünglichen Liste ist.

e) Definieren Sie eine rekursive Funktion `prod(l:NList) : RECURSIVE nat`, die das Produkt der Elemente der Liste `l` errechnet.

Zeigen Sie:

```
prod_double : PROPOSITION
  prod(double(l)) = 2^length(l) * prod(l)
```

Beweisen Sie jeweils auch die entstehenden TCCs!

Aufgabe 6-2

Definieren Sie die in der Vorlesung vorgestellte Funktion `rev` zur Listenumkehr in PVS. Sie können sich dabei auf den in PVS vordefinierten Listentyp `list[T]` beziehen.

a) Zeigen Sie:

```
rev_rev : PROPOSITION
  FORALL (l:list[T]): rev(rev(l)) = l
```

b) Definieren Sie wie in der Vorlesung die „iterative“ Variante $\texttt{rev}_2$. Zeigen Sie die Gleichheit der beiden Funktionen.

Aufgabe 6-3

Gegeben sei der folgende Datentyp eines Binärbaums, in dessen Knoten natürliche Zahlen gespeichert sind:

```
BinTree : DATATYPE
BEGIN
  empty : empty?
  node(key:nat, left:BinTree, right:BinTree) : node?
END BinTree
```

a) Definieren Sie Funktionen, die die minimale bzw. maximale Höhe eines Baums berechnen (`minheight, maxheight`). Die minimale (maximale) Höhe sei dabei die Länge des kürzesten (längsten) Pfads von der Wurzel bis zu einem Blatt.

Beweisen Sie, dass die minimale Höhe eines Baums immer kleiner oder gleich der maximalen Höhe ist.

b) Definieren Sie für einen *nicht-leeren* Binärbaum die Funktionen `minval, maxval`, die den kleinsten bzw. größten Wert zurückgegeben, sowie die Funktionen `leftmost, rightmost`, die den Wert im am weitesten links bzw. rechts liegenden Knoten eines Baums liefern.

c) Definieren Sie ein Prädikat `sorted?` über Binärbäumen, das genau dann erfüllt ist, wenn für jeden Knoten eines Baumes gilt, dass die Werte im linken Teilbaum kleiner oder gleich dem Knotenwert und die im rechten Teilbaum größer oder gleich diesem Wert sind.

Zeigen Sie: in einem nicht-leeren, sortierten Baum ist der kleinste Wert immer im am weitesten links und der größte im am weitesten rechts liegenden Knoten.

Vergessen Sie nicht, die entstehenden Typkorrektheitsbedingungen zu zeigen!