

Aufgabe 9-1

- a) Definieren Sie in PVS den Typ `even` der geraden natürlichen Zahlen als Subtyp von `nat`.
- b) Definieren Sie eine rekursive Funktion `half`, die eine gerade natürliche Zahl halbiert.
- c) Überprüfen Sie mit `M-x tc` ihre Theorie auf Typkorrektheit. Lassen Sie sich mit `M-x show-tccs` die entstandenen Typkorrektheitsbedingungen anzeigen. Es treten unterschiedliche Arten von TCCs auf: welche? Erklären Sie, warum diese TCCs generiert wurden. Beweisen Sie die TCCs mit `M-x pr`.
- d) Zeigen Sie:

```
half_halves : THEOREM
  FORALL (n:nat): half(2*n) = n
```

Aufgabe 9-2

Die folgende PVS-Deklaration definiert natürliche Zahlen als einen abstrakten Datentyp `Nat` mit Konstruktoren `zero` und `succ`:

```
Nat : DATATYPE
BEGIN
  zero      : zero?
  succ(prd:Nat) : succ?
END Nat
```

- a) Definieren Sie auf diesem Datentyp die Addition als rekursive Funktion. Benutzen Sie dazu folgendes Grundgerüst:¹

```
; +(n,m:Nat) : RECURSIVE Nat =
  CASES m OF
    zero      : <... Resultat für n + zero ...> ,
    succ(y)   : <... Resultat für n + succ(y) ...>
  ENDCASES
  MEASURE m BY <<
```

¹Das Semikolon am Zeilenanfang der Deklaration von `+` trennt die Deklaration syntaktisch von der vorigen ab. Dies muss an dieser Stelle (leider) gemacht werden, da der PVS-Parser sonst `+` als Infix-Operator interpretiert und so durcheinander gerät.

b) Beweisen Sie in PVS die Assoziativität der so definierten Addition:

```
% x,y,z : VAR Nat
add_associative : THEOREM x + (y + z) = (x + y) + z
```

c) Beweisen Sie in PVS die Kommutativität der Addition:

```
add_commutative : THEOREM x + y = y + x
```

Hinweis: Sie werden in diesem Beweis vermutlich die Hilfsgleichungen $\text{zero} + y = y$ und $\text{succ}(x) + y = \text{succ}(x + y)$ benötigen. Beweisen Sie dafür geeignete Lemmata!

d) Geben Sie eine analoge Definition der Multiplikation über `Nat` an.

e) Beweisen Sie in PVS die Distributivität der Multiplikation bzgl. der Addition:

```
mult_distributes_add : THEOREM x * (y + z) = (x * y) + (x * z)
```

f) Beweisen Sie in PVS die Assoziativität der Multiplikation:

```
mult_associative : THEOREM x * (y * z) = (x * y) * z
```

Aufgabe 9-3

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 $\text{app}(l, k: \text{NList}) : \text{RECURSIVE NList}$, 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 $\text{map}(f: [\text{nat} \rightarrow \text{nat}], l: \text{NList}) : \text{RECURSIVE NList}$, 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 $\text{sum}(l: \text{NList}) : \text{RECURSIVE nat}$, die die Summe der Elemente der Liste l errechnet.
Definieren Sie ferner eine Funktion $\text{double}(l: \text{NList}) : \text{NList}$, 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 $\text{prod}(l: \text{NList}) : \text{RECURSIVE nat}$, die das Produkt der Elemente der Liste l errechnet.

Zeigen Sie:

<pre>prod_double : PROPOSITION prod(double(l)) = 2^length(l) * prod(l)</pre>
--

Beweisen Sie jeweils auch die entstehenden TCCs!

Aufgabe 9-4

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

- a)** Zeigen Sie:

<pre>rev_rev : PROPOSITION FORALL (l: list[T]): rev(rev(l)) = l</pre>

- b)** Definieren Sie wie in der Vorlesung die „iterative“ Variante rev_2 . Zeigen Sie die Gleichheit der beiden Funktionen.