4. Übungsblatt zu Programmierung (WS 1999/2000)

Andreas Rossberg, Prof. Gert Smolka

Abgabe: 19. November 1999 vor der Vorlesung

Aufgabe 4.1: Polymorphie (6 Punkte)

Geben Sie jeweils eine Deklaration an, die dem Bezeichner `foo` das folgende allgemeinste Typschema zuordnet:

(a) `'a list list`

(b) `'a * 'b * 'c -> 'a * 'c list`

(c) `'a list * 'a -> 'a list`

(d) `('a -> 'b -> 'c) -> ('b -> 'a -> 'c)`

(e) `('a -> 'b -> 'c) -> ('a * 'b -> 'c)`

(f) `''a * ''a -> ''a`

Aufgabe 4.2: Transparenzbedingung (4 Punkte)

Folgende Deklarationen verletzen die Transparenzbedingung. Modifizieren Sie die Ausdrücke so, dass die Transparenzbedingung erfüllt wird, ohne die möglichen Typen einzuschränken.

(a) `val f = map (fn x => (x,x))`

(b) `val g = foldl op@`

(c) `val h = ListPair.map (fn (x,y) => (x,x,y))`

Hinweis: Die vordefinierte Prozedur `ListPair.map` hat den Typ

```
('a * 'b -> 'c) -> 'a list * 'b list -> 'c list
```

Sie erzeugt analog zu `map` eine Liste, indem sie die Argumentprozedur auf korrespondierende Elementpaare der Argumentlisten anwendet.

Aufgabe 4.3: Listenerzeugung und Faltung (10 Punkte)

Schreiben Sie die folgenden Prozeduren:

(a) `enum: int -> int -> int list`, so dass `enum` $m\ n$ = `[`m`,`$m+1$`,...,`n`]`. Für $m > n$ sei das Ergebnis die leere Liste.

(b) `filter: ('a -> bool) -> 'a list -> 'a list`, so dass `filter p xs` die Liste der Elemente von `xs` liefert, für die `p` den Wert `true` ergibt.

(c) `partition: ('a -> bool) -> 'a list -> 'a list * 'a list`, so dass der Ausdruck `partition p xs` ein Paar von Listen (ys, zs) liefert, bei dem ys alle Elemente von `xs` enthält, für die `p` den Wert `true` liefert, und zs alle anderen. Tip: Sie benötigen `let`, um das Zwischenergebnis der Rekursion zu zerlegen.

(d) `filter` und `partition` wie oben, aber jeweils mit Hilfe von `foldr`.

Aufgabe 4.4: Quicksort (10 Punkte)

Quicksort ist ein klassischer Sortieralgorithmus, der 1962 von Tony Hoare publiziert wurde. Die Idee hinter Quicksort lässt sich wie folgt beschreiben.

Seien *xs*, *ys* und *zs* drei Listen, so dass *xs* bis auf die Reihenfolge der Elemente mit *ys*@*zs* übereinstimmt. Weiter sei *x* ein Wert mit der Eigenschaft, dass alle Elemente von *ys* kleiner gleich und alle Elemente von *zs* größer gleich *x* sind. Dann gilt für jede Sortierprozedur *sort*:

$$sort(x::xs) = sort(ys)\ @\ [x]\ @\ sort(zs)$$

Schreiben Sie basierend auf dieser Idee eine Prozedur

```
sort : ('a * 'a -> bool) -> 'a list -> 'a list
```

zum Sortieren von Listen. Verwenden Sie `partition` aus Aufgabe 4.3, um die Argumentliste aufzuteilen.

Hinweis: Beginnen Sie mit einer Prozedur

```
sortInt : int list -> int list
```

zum Sortieren von ganzzahligen Listen.

Aufgabe 4.5: Primzahlen (10 Punkte)

Eine Primzahl ist eine natürliche Zahl $p > 1$, deren einzige Teiler 1 und p selbst sind. Entwickeln Sie eine Prozedur zur Berechnung von Primzahlen wie folgt:

(a) Schreiben Sie eine Prozedur `divisible: int * int -> bool`, die für zwei positive Zahlen x,y testet, ob x durch y teilbar ist.

(b) Schreiben Sie eine Prozedur `divisibleBySome: int * int list -> bool`, die testet, ob eine positive Zahl durch mindestens ein Element einer Liste positiver Zahlen teilbar ist.

(c) Sei $x > 1$ und *ps* eine Liste, die genau die Primzahlen enthält, die kleiner x sind. Dann ist x eine Primzahl genau dann, wenn `divisibleBySome(`x`,`ps`) = false`. Schreiben Sie unter Verwendung dieser Äquivalenz eine Prozedur

```
nextPrime: int * int list -> int
```

welche zu einer Zahl $x > 1$ und einer Liste aller Primzahlen kleiner x die kleinste Primzahl $p \geq x$ liefert.

(d) Schreiben Sie nun eine Prozedur `primes: int -> int list`, die für eine natürliche Zahl n eine Liste der ersten n Primzahlen berechnet (die Reihenfolge der Listenelemente ist beliebig).