Klausur „Programmierparadigmen (PGP)“ (Sommer 2007)

Aufgabe 1 [Gesamtpunktzahl: 10]

In dieser Aufgabe geht es darum, auf unterschiedliche Weise zu einer endlichen Liste `list` die Liste ihrer „Listenreste“ zu bestimmen bzw. mit dieser zu arbeiten. Die Liste der „Listenreste“ enthält die ursprüngliche Liste und alle – jeweils durch wiederholtes Weglassen des ersten Elements gebildeten – Restlisten von `list`.

Beispiel: Eine polymorphe **Haskell-Funktion** `restlists` sollte als Wert von `restlists [1 .. 3]` liefern `[[1,2,3],[2,3],[3],[]]`

a) Definieren Sie die **Haskell-Funktion** `restlists` *durch eine Listenkomprehension.* [Punkte: 1]

b) Definieren Sie die **Haskell-Funktion** `restlists` *direkt durch Rekursion.* [Punkte: 2]

c) Definieren Sie eine **Haskell-Funktion** `filterRestlists`, die ein Prädikat und eine Liste als Argumente nimmt und die Liste derjenigen Restlisten der als Argument übergebenen Liste liefert, für die das Prädikat zutrifft. [Punkte: 2]
Beispiel: Als Wert von `filterRestlists (\l -> even (length l)) [1,2,3]` sollte sich ergeben `[[2,3],[]]`

d) Definieren Sie direkt durch Rekursion ein **Prolog-Prädikat** `restlists(List,Restlists)`, das für beliebige *endliche* Listen zutrifft, wenn `Restlists` die Liste der „Restlisten“ von `List` ist. [Punkte: 2]

e) Durch welche Query in **Prolog** mit dem eingebauten Prädikat `append` könnten Sie sich nacheinander die Restlisten zu einer gegebenen *endlichen* Liste liefern lassen? [Punkte: 1]

f) Nutzen Sie e) zu einer alternativen Definition des **Prolog-Prädikats** `restlists(List,Restlists)`. [Punkte: 2]

Beispiel (zu d) und f)): Die Query `restlists([1,2,3], Restlists).` soll ergeben
`Restlists = [[1, 2, 3], [2, 3], [3], []]`

Aufgabe 2 [Gesamtpunktzahl: 10]

Listen ohne wiederkehrende Elemente können als Darstellungen endlicher Mengen verwendet werden.
a) Definieren Sie eine **Scheme-Funktion** `powerset`, die einer (als Liste dargestellten) endlichen Menge ihre (als Liste dargestellte) Potenzmenge, also die Menge ihrer Teilmengen, zuordnet. [Punkte: 4,5]

Beispiel:
Der Aufruf
```
(powerset '(a b c))
```
soll z.B. liefern

```
((a b c) (a b) (a c) (a)
 (b c) (b) (c) ())
```

Bemerkung: Auch eine andere Reihenfolge der Elemente der Potenzmenge ist zulässig.
b) Definieren Sie ein **Prolog-Prädikat** `powerset(Set, Powerset)` – sowie benötigte und nicht bereits in Prolog vordefinierte Hilfsprädikate –, das für beliebige *endliche*, als Listen dargestellte Mengen `Set` zutrifft, wenn `Powerset` ihre zugehörige Potenzmenge darstellt. [Punkte: 5,5]

Beispiel:
Die Query
```
powerset([1,2,3],PS).
```
soll z.B. ergeben
```
PS = [[1, 2, 3], [1, 2], [1, 3], [1], [2, 3], [2], [3], []]
```

Bemerkung: Auch hier ist eine andere Reihenfolge der Elemente der Potenzmenge zulässig.

Aufgabe 3 [Gesamtpunktzahl: 8]

Gegeben seien die folgenden Definitionen der **Haskell-Funktionen** `take`, `drop` und `splitAt`.

```
take                :: Int -> [a] -> [a]
take 0 _            = []                              (take.1a)
take _ []           = []                              (take.1b)
take n (x:xs)       = x : take (n-1) xs               (take.2)

drop                :: Int -> [a] -> [a]
drop 0 xs           = xs                              (drop.1a)
drop _ []           = []                              (drop.1b)
drop n (_:xs)       = drop (n-1) xs                   (drop.2)

splitAt               :: Int -> [a] -> ([a], [a])
splitAt 0 xs        = ([],xs)                       (splitAt.1a)
splitAt _ []        = ([],[])                       (splitAt.1b)
splitAt n (x:xs)    = (x:xs',xs'')                  (splitAt.2a)
                       where
                       (xs',xs'') = splitAt (n-1) xs  (splitAt.2b)
```

Beweisen Sie, indem Sie strukturelle Induktion über die Listenlänge und Induktion über `n` geeignet verbinden, dass **für alle natürlichen Zahlen** `n` (einschliesslich Null) und **für alle endlichen Listen** `xs` gilt:

```
splitAt n xs = (take n xs, drop n xs)
```

Wichtiger Hinweis: Geben Sie bitte bei allen Umformungen in Beweisschritten jeweils die verwendeten Gleichungen an.

Aufgabe 4 [Gesamtpunktzahl: 12]

Sogenanntes *perfektes Mischen* eines Kartenstapels mit einer *geraden* Zahl von Karten funktioniert wie folgt: Der Kartenstapel wird in der Mitte geteilt, dann werden die beiden Hälften so ineinander gemischt, dass die Karten aus den beiden Hälften genau abwechselnd im gemischten Stapel vorkommen.
Beispiel: Der ursprüngliche Stapel enthalte *Zehn, Bube, Dame, König, As, Joker*. Die zwei Hälften enthalten dann einerseits *Zehn, Bube, Dame* und andererseits *König, As, Joker*. Nach dem Zusammenfügen ergibt sich die gemischte Reihenfolge *Zehn, König, Bube, As, Dame, Joker*.
Perfektes Mischen in diesem Sinne soll nun zunächst in **Haskell** und in Anlehnung daran dann in **Prolog** realisiert werden. Der Einfachheit halber repräsentieren wir den Kartenstapel dabei als Liste von natürlichen Zahlen.
a) Schreiben Sie zunächst eine **Haskell-Funktion** `interleave`, die zwei als gleich lang vorausgesetzte Listen `l1` und `l2` akzeptiert (muss nicht überprüft werden) und eine Liste zurückgibt, in der die Elemente aus `l1` und `l2` abwechselnd vorkommen, wobei mit dem ersten Element von `l1` begonnen wird. [Punkte: 1,5]
Beispiel: `interleave [1 .. 4][5 .. 8]` ergibt `[1, 5, 2, 6, 3, 7, 4, 8]`
b) Schreiben Sie damit eine **Haskell-Funktion** `shuffle`, die perfektes Mischen für Listen mit einer geraden Anzahl von Elementen realisiert. [Punkte: 2]
Beispiel: `shuffle [1 .. 8]` ergibt `[1, 5, 2, 6, 3, 7, 4, 8]`
c) Wird ein Kartenspiel oft genug perfekt gemischt, kehrt es in seine ursprüngliche Reihenfolge zurück. Schreiben Sie eine **Haskell-Funktion** `shuffleNumber`, die eine beliebige gerade Zahl grösser Null als Grösse eines Kartenstapels akzeptiert und die zurückgibt, wie oft ein Kartenstapel dieser Grösse *höchstens* perfekt gemischt werden muss, damit er seine ursprüngliche Reihenfolge wieder annimmt (wobei mindestens einmal gemischt werden muss). [Punkte: 2,5]
Beispiel: `shuffleNumber 52` ergibt `8`
d) Definieren Sie ein **Prolog-Prädikat** `interleave(L1,L2,List)`, das zutrifft, wenn `List` eine Liste ist, in der die Elemente aus `L1` und `L2` abwechselnd vorkommen, wobei mit dem ersten Element von `L1` begonnen wird. [Punkte: 1,5]
Beispiel: Die Query `interleave([1,2,3],[4,5,6],Res).`
soll ergeben `Res = [1, 4, 2, 5, 3, 6]`
e) Definieren Sie ein **Prolog-Prädikat** `splitAt(N,List,Anf,Rest)`, welches - analog zur Haskell-Funktion `splitAt` in Aufgabe 3 - zutrifft, wenn `Anf` und `Rest` diejenigen Listen sind, die sich beim "Zerlegen" von `List` nach Position `N` ergeben. [Punkte: 2,5]
Beispiel: Die Query `splitAt(3, [1,2,3,4,5],Anf,Rest).`
soll ergeben `Anf = [1, 2, 3]`, `Rest = [4, 5]`
f) Definieren Sie mit Hilfe von d) und e) ein **Prolog-Prädikat** `shuffle(List, Shuffled)`, das zutrifft, wenn `Shuffled` die durch perfektes Mischen aus den Elementen von `List` gebildete Liste ist. [Punkte: 2]
Beispiel: Die Query `shuffle([1,2,3,4,5,6,7,8], Sh).`
soll ergeben `Sh = [1, 5, 2, 6, 3, 7, 4, 8]`