Programmierung und Modellierung – Klausur vom 16. Juli 2011

Notenschlüssel:

1.0	54-49
1.3	48-46
1.7	45-43
2.0	42-40
2.3	39-37
2.7	36-33
3.0	33-31
3.3	30-28
3.7	27-25
4.0	24-22
5.0	21

Aufgabe 1: Multiple Choice

Jede rekursive Funktion, die bei jeder Anwendung terminiert ist endrekursiv.	☐ ***ja***	☐ ***nein***
Eine durch Pattern Matching definierte Funktion muss für jede Eingabe, die bezgl. des Typs der Funktion korrekt ist, ein passendes Muster bereitstellen.	☐ ***ja***	☐ ***nein***
In SML gibt es drei boolesche Werte: true, false und undefiniert.	☐ ***ja***	☐ ***nein***
Currying macht einstellige Funktionen aus zweistelligen Funktionen.	☐ ***ja***	☐ ***nein***
Ein Funktor in SML liefert Strukturen als Ergebnis.	☐ ***ja***	☐ ***nein***
Im Gegensatz zu SML hat Haskell verzögerte Auswertung.	☐ ***ja***	☐ ***nein***
Bei der lexikalischen Analyse wird geprüft, ob eine Folge von Token einen syntaktisch korrekten Satz bildet.	☐ ***ja***	☐ ***nein***
Die denotationale Semantik weist jedem Ausdruck einen abstrakten mathematischen Wert zu.	☐ ***ja***	☐ ***nein***

Aufgabe 2: Abstiegsfunktion

a) Sei f: A -> B eine rekursiv definierte Funktion und sei m: A -> N eine Abstiegsfunktion für f. Zeigen Sie, dass die Funktion m' : A -> ℕ mit $m'(x) = 3 \cdot m(x) + 5$ dann ebenfalls eine Abstiegsfunktion für f ist.

b) Gegeben ist folgende Relation R auf ℕ x ℕ:

$(x_1,x_2)R(y_1,y_2)$ gilt genau dann, wenn $x_1 < y_1$ oder $x_2 < y_2$ gilt

Zeigen Sie, dass *R* keine wohlfundierte Relation ist!

c) Sei `f: int list -> int` definiert durch

```
fun f  []        = 0
|   f (h::t)    = if h<=0 then 0 else h + f(t);
```

Geben Sie eine Abstiegsfunktion zu `f` an!

Aufgabe 3: Induktion

Gegeben sei eine Funktion `app : 'a list -> 'a list -> 'a list` definiert durch

```
fun app x y = foldr (fn (a,b) => a::b) y x;
```

Beweisen Sie mit Hilfe des Substitutionsmodells durch Induktion über l_1, dass für alle Listen l_1 `: 'a list` und l_2 `: 'a list` gilt:

$$\texttt{app}\ l_1\ l_2 = l_1\ @\ l_2.$$

Hinweise:
(i) Die infix notierte Funktion `@` hängt zwei Listen zusammen. Beispielsweise wertet `[1,2] @ [3,4,5]` zu `[1,2,3,4,5]` aus.
(ii) Sie können einfache Eigenschaften von `@` ohne Beweis verwenden, wie zum Beispiel

$$[\,]\ @\ l_2 = l_2$$
$$(x::l_1)\ @\ l_2 = x::(l_1@l_2)$$

für alle Listen l_1 und l_2 vom Typ `'a list` und alle Werte x vom Typ `'a`.
(iii) Die Funktion `foldr : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b` ist definiert durch

```
fun foldr f z []          = z
|         f z (x::xs)     = f (x, foldr f z xs);
```

Aufgabe 4: Bäume

Gegeben sei folgender Datentyp eines Baums dessen Blätter ganze Zahlen tragen.

```
datatype Tree = Node of Tree * Tree | Leaf of int
```

a) Definieren Sie eine Funktion `sumTree : Tree -> int`, die für einen Baum die Summe der Werte aus dessen Blätter berechnet.

b) Schreiben Sie eine endrekursive Funktion `numLeaves` vom Typ `Tree -> int`, die für einen Baum die Anzahl seiner Blätter zurück gibt.
Hinweis: Verwenden Sie neben einem Akkumulator für die Anzahl der Blätter auch eine Liste der noch zu betrachtenden Teilbäume. Sie dürfen Hilfsfunktionen deklarieren.

Aufgabe 5: Bags

Ein bag ist eine Sammlung von Werten, wobei Werte doppelt vorkommen können. Die erste Repräsentation von bags, die wir hier betrachten, sind Listen, welche aufsteigend sortiert sind, z.B.

```
["paper","paper","paper","pen","pencil","pencil","pin"]
```

Eine zweite Repräsentation sind sortierte Listen aus Paaren aus einem Wert und seiner Anzahl. Obige Sammlung sähe dann so aus:

```
[("paper",3),("pen",1),("pencil",2),("pin",1)]
```

a) Schreiben Sie eine Funktion `bag1ToBag2 : ''a list -> (''a * int) list`, die ein bag `b` aus der ersten in die zweite Repräsentation konvertiert. Achten Sie darauf, dass das Ergebnis eine *sortierte Liste* sein muss. Eine Funktion zum Sortieren sollten Sie aber nicht benötigen.

b) Schreiben Sie die Umkehrfunktion `bag2ToBag1 : (''a * int) list -> ''a list`, die ein bag `b` aus der zweiten in die erste Repräsentation überführt. Achten Sie darauf, dass das Ergebnis eine *sortierte Liste* sein muss. Eine Funktion zum Sortieren sollten Sie aber nicht benötigen.
Als Hilfe sei die Funktion `replicate : int -> 'a -> 'a list` vorgegeben, welche einen Wert vervielfacht. Beispiel: `replicate 3 "a" = ["a","a","a"]` oder `replicate 1 2.0 = [2.0]`.

Aufgabe 6: Prinzipale Typisierung

Geben Sie für jeden der folgenden Ausdrücke entweder den prinzipalen Typ an, oder eine kurze Begründung, warum der Ausdruck keinen gültigen Typ besitzt. Eine formale Herleitung mit dem Typinferenzalgorithmus ist nicht erforderlich. Die Antwort zur ersten Teilaufgabe a ist als Beispiel vorgegeben.

a) `{a = true, b = true, c = true}`
Beispiellösung: `{a:bool, b:bool, c:bool}`

b) `[true,true,true]`

c) `(true,true,true)`

d) `fun doubleprice s p = {price = 2.0 * p, name = s}`

e) `fun sum(h::t) = if not h then 0 else h + sum(t)`

f) `fun sum(k) = sum(k-1)+k`

g) `fun flip {a=x,b=y} = {d=y,e=x}`

h)
```
fun true_elements(     [])        = []
  | true_elements((true,y)::t)   = y :: true_elements(t)
  | true_elements(   _    ::t)   = true_elements(t)
```

i) `fun f h = h(f h)`

Aufgabe 7: Typregeln

In SML kann man Werte verschiedenen Typs mit Klammern und Kommas zu Tupeln zusammenfassen. Zum Beispiel ein Paar aus einem Wahrheitswert und einem Integer: `(true,42)`. Dieses Paar hat in SML den Typ `bool * int`.

Überlegen Sie sich analog zu den in der Vorlesung behandelten Typregeln eine sinnvolle Typregel für die Bildung von Paaren. Ergänzen Sie die fehlenden Prämissen!

$$\frac{\qquad\qquad\qquad\qquad\qquad\qquad}{\Gamma \vdash (e_1, e_2) : A * B}$$

Überlegen Sie sich dazu, welche Bedingung an die veränderlichen Terme e_1 und e_2 gestellt werden müssen.

Aufgabe 8: Strukturen

Gegeben ist folgende SML-Struktur Definition für `Rational`:

```
structure Rational : ARITH =
struct
     type t = int * int;
     val null : t = (0,1);
     val eins : t = (1,1);
     fun plus((z,n) : t, (z',n') : t): t = (z*n' + n*z', n*n');
     fun mal((z,n) : t, (z',n') : t) : t = (z*z', n*n');
end
```

a) Geben Sie die Signatur `ARITH` an, welche alle Deklarationen von Rational enthält. Die Signatur soll aber lediglich spezifizieren, dass es einen Typ t gibt, aber nicht, wie die Werte dieses Typs zu repräsentieren sind.

```
signature ARITH =
     sig





     end
```

b) Die in der Signatur `ARITH` spezifizierten Operationen sollen für reelle Zahlen zur Verfügung gestellt werden. Implementieren Sie dazu erneut eine Struktur zur Signatur `ARITH`. Sie dürfen dabei den SML-Typ `real` verwenden.

```
structure Real : ARITH =
     struct









     end
```