Ludwig-Maximilians-Universität München
Department „Institut für Informatik“
Prof. Dr. Christian Böhm
Elke Achtert, Arthur Zimek

München, 19.01.2007

Informatik 1
WS 2006/07

Übungsblatt 12: Auswertung und Ausnahmen, Strukturelle Induktion, Pattern Matching

Besprechung: 29.01.–02.02.2007

Abgabe aller mit **Hausaufgabe** markierten Aufgaben bis Freitag, 26.01.2007, 18:00 Uhr

Aufgabe 12-1 *Auswertung und Ausnahmen, Ausnahmebehandlung* **(Hausaufgabe)**

```
datatype formula = constant of int
                 | variable of string
                 | sum      of formula * formula
                 | product  of formula * formula;
```

Die Datei `12-1.sml` enthält die Definition dieses Datentyps, den Sie von Übungsblatt 11 bereits kennen.

(a) Eine Umgebung (englisch *environment*) sei eine Liste von Paaren wie zum Beispiel:

```
[(variable "y",2), (variable "x",4), (variable "z",3)]
[(variable "x",1), (variable "y",2), (variable "x",4), (variable "z",3)]
```

Die erste dieser beiden Umgebungen repräsentiert, dass die Variable x an den Wert 4 gebunden ist, die Variable y an den Wert 2 und die Variable z an den Wert 3. Die zweite Umgebung repräsentiert, dass die Variable x an den Wert 1 gebunden ist, wodurch die Bindung von x an den Wert 4 überschattet wird. Die Bindungen von y und z sind wie in der ersten Umgebung.

Definieren Sie in der Datei `12-1.sml` eine Funktion `value` vom Typ
`formula * (formula * 'a) list -> 'a`,
die den Wert einer Variablen in einer Umgebung liefert, wie in den folgenden Beispielen:

```
- value( variable "x", [(variable "y",2), (variable "x",4),
                        (variable "z",3)] );
val it = 4 : int
- value( variable "x", [(variable "x",1), (variable "y",2),
                        (variable "x",4), (variable "z",3)] );
val it = 1 : int
- value( variable "y", [(variable "x",1), (variable "y",2),
                        (variable "x",4), (variable "z",3)] );
val it = 2 : int
```

Wenn der Wert einer Variablen durch die Umgebung nicht definiert ist, soll eine Ausnahme
`variable_undefined_in_environment`
mit einem einstelligen Konstruktor vom Typ `string` geworfen werden, die mit dem Namen der unbekannten Variablen konstruiert wird.

```
- value( variable "w", [(variable "y",2), (variable "x",4),
                        (variable "z",3)] );
uncaught exception variable_undefined_in_environment
```


(b) Definieren Sie in der Datei `12-1.sml` eine Funktion `eval` vom Typ
`formula * (formula * int) list -> int`
zur Auswertung einer arithmetischen Formel in einer Umgebung. Beispiele:

```
- eval( variable "x",
        [(variable "x",1), (variable "y",2),
         (variable "x",4), (variable "z",3)] );
val it = 1 : int
- eval( constant 7,
        [(variable "x",1), (variable "y",2),
         (variable "x",4), (variable "z",3)] );
val it = 7 : int
- eval( product(sum(variable "x",constant 7), variable "y"),
        [(variable "x",1), (variable "y",2),
         (variable "x",4), (variable "z",3)] );
val it = 16 : int
```

(c) Definieren Sie in der Datei `12-1.sml` eine Funktion `evalToString` vom Typ
`formula * (formula * int) list -> string`,
die das Ergebnis der Auswertung einer arithmetischen Formel in einer Umgebung als `string` zurückgibt. Diese Funktion soll die Ausnahme `variable_undefined_in_environment` behandeln, in dem mit einer kurzen Erklärung der Name der Variablen zurückgegeben wird, die für das Auftreten der Exception verantwortlich ist. Beispiel:

```
- evalToString( product(sum(variable "x",constant 7), variable "w"),
                [(variable "x",1), (variable "y",2),
                 (variable "x",4), (variable "z",3)] );
val it = "undefined variable: w" : string
```

Aufgabe 12-2 *Ausnahmen, Substitutionsmodell* **(Hausaufgabe)**

```
exception d0 of int
fun durch(z,n)  =  if n=0 then raise d0(z) else z div n
fun positiv(x)  =  (x > 0)
```

Beschreiben Sie in Datei `12-2.txt` mit dem Substitutionsmodell, wie nach diesen Deklarationen der folgende Ausdruck ausgewertet wird. Gehen Sie von der Auswertungsreihenfolge aus, die SML benutzt, und geben Sie sämtliche Zwischenschritte vollständig an (das ergibt 9 weitere Zeilen).

```
positiv(1 + durch(~5,0)) handle d0(z) => positiv(z)
```

Aufgabe 12-3 *Strukturelle Induktion* **(Hausaufgabe)**

Betrachten Sie die Menge $N \in \mathbb{N} \times \mathbb{N}$, die wie folgt *induktiv* definiert ist:

- $(0,0) \in N$.
- $(1,1) \in N$.
- $(m,n) \in N \Rightarrow (m+2,n) \in N \wedge (m,n+2) \in N$.

Beweisen Sie durch strukturelle Induktion, dass für alle $(x,y) \in N$ gilt: $x+y$ ist gerade.

Aufgabe 12-4 *Muster vs. if-then-else*

Bartl Bastscho braucht eine Funktion, die ihm sagt, wie lang die Jahreszeiten auf der Nordhalbkugel sind. Er überlegt, ob er sie mit der Muster-Technik oder mit der if-then-else-Technik formulieren soll:

```
datatype jahreszeit = fruehling | sommer | herbst | winter;
```

```
fun laenge fruehling = 93.0
  | laenge sommer    = 93.0
  | laenge herbst    = 89.5
  | laenge winter    = 89.5
```

```
fun laenge j =
    if      j=fruehling then 93.0
    else if j=sommer    then 93.0
    else if j=herbst    then 89.5
    else (* j=winter *)      89.5
```

Er fragt Vroni Frogstmi, was ihr besser gefällt. Unvorsichtigerweise lässt er dabei die Bemerkung fallen, dass die Entscheidung zwischen Muster und if-then-else wohl nur von rein stilistischen Gesichtspunkten abhängt. Vroni erwidert, dann würde sie gern mal sehen, wie er die folgenden Funktionen mit beiden Techniken definieren kann:

- Für den Datentyp `int` sei `betrag` die Funktion, die jede positive ganze Zahl auf sich selbst abbildet und jede nicht-positive auf das (-1)-fache.
- Für **`datatype`** `koordinaten = koord` **`of`** `int * int` seien `x_wert` und `y_wert` die Funktionen, die die jeweilige Koordinate liefern.

Untersuchen Sie, welche dieser Funktionen mit welcher der beiden Techniken definiert werden kann. Leiten Sie daraus allgemeinere Antworten zu folgenden Fragen ab:

- In welchen Fällen ist eine Definition mit if-then-else möglich, aber nicht mit Mustern?
- In welchen Fällen ist eine Definition mit Mustern möglich, aber nicht mit if-then-else?

Aufgabe 12-5 *Schmankerl zum Abschluss: Polymorpher Baumtyp als Operatorbaum*

Als letzte Übungsaufgabe betrachten Sie das Programm aus Datei `12-5`. Versuchen Sie die gegebenen Definitionen zu verstehen.

(a) Definieren Sie in der selben Datei eine Funktion `intOpBaum`, die aus einer (gültigen) Liste von Strings einen Operatorbaum vom Typ `(int,int -> int,int * int -> int) baum` erzeugt. In der Liste sind Konstanten und Operationen (in Präfix-Notation) als Strings angegeben, wie sie von den Funktionen `istKonstante`, `konstante`, `istUnaer`, `unaer`, `istBinaer` und `binaer` vorgesehen sind. Wenn es Ihnen Spaß macht (?), können Sie diese Funktionen gerne erweitern. Der Operatorbaum repräsentiert dann einen arithmetischen Ausdruck.

Anwendungsbeispiel:

```
- val l1 = ["quad","+","2","-","7","+","~","1","4"];
val l1 = ["quad","+","2","-","7","+","~","1","4"] : string list
- val b1 = intOpBaum l1;
val b1 = Knt1 (fn,Knt2 (Blt #,fn,Knt2 #))
  : (int,int -> int,int * int -> int) baum
```

(b) Definieren Sie nun zusätzlich eine Funktion `eval` vom Typ
`('a,'a -> 'a,'a * 'a -> 'a) baum -> 'a`,
die einen Operatorbaum wie in der vorherigen Teilaufgabe auswertet, also den Wert des repräsentierten arithmetischen Ausdrucks berechnet.

Anwendungsbeispiel:

```
- eval b1;
val it = 36 : int
```