Praktische Informatik 3 WS 2013/2014

2. Übungsblatt

Ausgabe: 29. 10. 2013
Abgabe: 8.11.2013

Berthold Hoffmann <hof>
Christian Maeder <maeder>
Daniel Müller <dmueller>
Sören Schulze <sschulze>
Tatjana Steckel <steckel>
Jasper van de Ven <jasper>

In diesem Blatt geht es um Funktionen auf Ausdrücken. Die von mir in der Vorlesung vorgestellten Funk-Ausdrücke wurden von den Turor_innen als *geistig gefährdend* für Haskell-Neulinge angesehen. Deshalb sind sie stark vereinfacht so definiert:

```
data Exp = Var  X
         | Con  Integer
         | App  X [Exp]
         | Let  Defs Exp deriving (Eq,Show)
```

Hierbei sind X und *Defs* definiert als

```
type X = String; type Defs = [(X, Exp)]
```

(Es nützt aber alles nichts: als alte Haskell-Hasen werden Sie sich doch noch mit den vollständigen Ausdrücken aus Abschnitt 3.6 des Skripts auseinandersetzen müssen!)

2.1 Freie Variablen und Substitution (8 Punkte)

Folgende Hilfsfunktionen brauchen Sie für die Auswertung:

1. Die Funktion *free* :: $Exp \to [X]$ soll alle Variablen bestimmen, die in einem Ausdruck frei auftreten:
   - Eine Variable *Var* x ist frei.
   - In einem Konstruktor *Con* n gibt es keine freien Variablen.
   - In einer Applikation *App* $f\ e_1 \ldots e_k$ ist f *nicht* frei, wohl aber alle freien Variablen von e_1 bis e_k.
   - In einer lokalen Definition *Let* $[(x_1, e_1), ..., (x_k, e_k)]\ e$ sind alle freien Variablen von e, e_1 bis e_k frei, bis auf die Variablen $x_1, \ldots, x_k$, die hier *gebunden* werden.
2. Die Funktion *subst* $(x,d)\ e$ soll die alle freien Auftreten der Variablen x in e durch den definierenden Ausdruck d ersetzen. Ihr Typ ist*subst* :: $(X,Exp) \to Exp \to Exp$.
   - Der Ausdruck *Var* x wird durch d ersetzt; ein Ausdruck *Var* y mit $y \neq x$ beliebt unverändert.
   - Ein Konstruktor bleibt unverändert.
   - In einer Appplikation *App* $f\ e_1 \ldots e_k$ werden alle freien Auftreten von x in e_1 bis e_k ersetzt.
   - In lokalen Definitionen *Let* $[(x_1, e_1), ..., (x_k, e_k)]\ e$ werden alle freien Auftreten von x in e_1 bis e_k ersetzt; wenn keine der $x_1, \ldots, x_k$ gleich x sind, wird x auch in e ersetzt.

2.2 Auswertung (12 Punkte)

Implementieren sie eine Funktion *evaluate* :: *Exp* →*Integer*, die Ausdrücke rekursiv auswertet. Während desse werden Ausdrücke der Form *Let ds e* eleminiert, indem die lokale Definitionen *ds* in *e* eingesetzt werden.

In Applikationen *App* $f[a_1, \ldots, a_k]$ wird eine fest eingebaute Funktion auf die Liste $[n_1, \ldots, n_k]$ angewendet, wobei $n_i = eval\ a_i$ für $1 \leq i \leq k$. Wenn $f =$"$(+)$" und $k = 2$, sollte $n_1 + n_2$ berechnet werden; analog für $f \in \{$"$(-)$", "$(*)$"$\}$ (Weitere dürfen nach Belieben hinzugefügt werden).

Tipps:

1. Verwenden Sie *lookup* :: *Eq* $\alpha \Rightarrow \alpha \rightarrow [(\alpha, \beta)] \rightarrow$ *Maybe* β zur Suche in Definitionen!
2. Benutzen Sie die Funktionen *union* (eingetippt als `union`) und *nub* aus *Data.List* für die Vereinigung von Listen bzw. für das Entfernen von Doubletten aus Listen.

Dies ist Fassung 1 vom 29. Oktober 2013.