Philipps Universität Marburg

Übungen zu „Konzepte von Programmiersprachen“, WS 2010/11

Prof. Dr. R. Loogen · Fachbereich Mathematik und Informatik · Marburg

Nr. 3, Abgabe: Dienstag, 9. November 2010 vor der Vorlesung

6. Ähnlichkeit von Worten / Listenabstraktionen — 6 Punkte

Definieren Sie unter Verwendung von *Listenabstraktionen* Funktionen `similarX :: String -> String -> Bool` mit $\texttt{X} \in \{\texttt{A}, \texttt{B}, \texttt{C}\}$, die prüfen, ob zwei Worte (Typ `String`) *ähnlich* sind. Betrachten Sie als „Ähnlichkeit“ folgende unterschiedlichen Definitionen:

(a) Zwei Worte heißen ähnlich, wenn sie sich nur in einem Buchstaben unterscheiden. / 2

(b) Zwei Worte heißen ähnlich, wenn sie durch Auslassen bzw. Hinzufügen genau eines Buchstabens ineinander übergehen. / 2

(c) Zwei Worte heißen ähnlich, wenn sie durch Vertauschung zweier benachbarter Buchstaben ineinander übergehen. / 1,5

Wie ändert sich die Funktion, wenn man alle drei Möglichkeiten als Definition von Ähnlichkeit akzeptiert? / 0,5

Hinweis: Die vordefinierte Funktion `elem :: Eq a => a -> [a] -> Bool` testet, ob ein Element in einer Liste vorkommt.

7. Sparschwein / Nachweis von Programmeigenschaften — 6 Punkte

(a) Implementieren Sie auf zwei verschiedene Weisen einen abstrakten Datentyp Sparschwein mit den Operationen:

```
emptyPig :: PiggyBank                        -- leeres Sparschwein
add      :: Coin -> PiggyBank -> PiggyBank   -- Einwerfen einer Muenze
shake    :: PiggyBank -> (PiggyBank, Coin)   -- Herausschütteln einer Muenze
isEmpty  :: PiggyBank -> Bool                -- Test auf leeres Sparschwein
break    :: PiggyBank -> Money               -- Aufbrechen des Sparschweins
```

Dabei seien die Datentypen `Coin` und `Money` wie folgt definiert:

```
data Coin  = OneCent | TwoCents | FiveCents | TenCents | FiftyCents
           | OneEuro | TwoEuros
type Money = Int -- Geldwert in Cents
```

Verwenden Sie für die beiden Implementierungen die folgenden Typfestlegungen:

i. `type PiggyBank1 = (Int, Int, Int, Int, Int, Int, Int)` / 1,5

ii. `type PiggyBank2 = [Coin]` / 1

Welche der beiden Implementierungen ist effizienter? / 0,5

(b) Beweisen Sie, dass Ihre Implementierungen jeweils den folgenden Gleichungen genügen:

i. Für beliebige `x :: Coin` gilt: / 1

`shake (add x emptyPig) = (emptyPig, x).`

ii. Für beliebige `x :: Coin` und `p :: PiggyBank` gilt: / 2

`break p + break (add x emptyPig) = break (add x p).`