Praktische Informatik 3: Funktionale Programmierung

Vorlesung 11 (10.01.2023): Übungen

Christoph Lüth

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Universität Bremen

Wintersemester 2022/23

Food for Thought

Übung 11.1: Verkapselung

Warum **müssen** wir den Datentyp `State` σ α in einen Datentyp verkapseln, und wie sieht dessen Signatur aus?

Food for Thought

Übung 11.1: Verkapselung

Warum **müssen** wir den Datentyp `State` σ α in einen Datentyp verkapseln, und wie sieht dessen Signatur aus?

Lösung: Wenn wir den Zustand explizit durch die Gegend reichen, können wir ihn beliebig kopieren — das ist sicherlich nicht beabsichtigt, es sollte immer nur genau eine Kopie des Zustands geben.
Die Signatur besteht aus `comp`, `lift`, `map`, `get` und `set` — siehe nächsten Abschnitt.

Jetzt seit ihr dran.

Übung 11.2: Komposition in der Listenmonade

Betrachten wir noch mal die Komposition in der Listenmonade:

```
a : as >>= g = g a ++ (as >>= g)
[] >>= g = []
```

Welche uns (hoffentlich) wohlbekannte Funktion versteckt sich dahinter?

Jetzt seit ihr dran.

Übung 11.2: Komposition in der Listenmonade

Betrachten wir noch mal die Komposition in der Listenmonade:

```
a : as >>= g = g a ++ (as >>= g)
[] >>= g = []
```

Welche uns (hoffentlich) wohlbekannte Funktion versteckt sich dahinter?

Lösung: Das ist dasselbe wie `concatMap`, nur mit umgedrehten Argumenten:

```
concatMap :: (α → [β]) → [α] → [β]
concatMap f = concat ∘ map f

(>>=) == flip concatMap
```

Kurz Nachgedacht.

Übung 11.3: `IO` als Zustandsmonade

Können wir den Zustand des Dateisystems als expliziten Zustandsparameter modellieren, in etwa

```
data FS = Map Int String
```

wobei jede Zeichenkette den Inhalt einer Datei darstellt.

Kurz Nachgedacht.

Übung 11.3: `IO` als Zustandsmonade

Können wir den Zustand des Dateisystems als expliziten Zustandsparameter modellieren, in etwa

```
data FS = Map Int String
```

wobei jede Zeichenkette den Inhalt einer Datei darstellt.

Lösung:

- Mathematisch nein — der Zustand des Dateisystems kann sich zwischenzeitlich ändern (es ist weder referentiell transparent, noch gelten die Gleichungen des ADT `Map`).
- Pragmatisch ja — solange wir nicht versuchen, den Zustand zu duplizieren (an einen Namen zu binden), die `Map`-Gleichungen zur Optimierung zur verwenden, und die Reihenfolge der Operationen zu verändern.

Kombiniere!

Übung 11.4: Andere Kombinationen sind möglich:

i. `data Res` $\sigma\ \alpha =$ `Res (`$\sigma\rightarrow$ `Exn [`α`])`

ii. `data Res` $\sigma\ \alpha =$ `Res (Exn [`$\sigma\rightarrow\ \alpha$`])`

iii. `data Res` $\sigma\ \alpha =$ `Res ([`$\sigma\rightarrow$ `Exn` α`])`

Was für eine Art Berechnung modellieren diese, und was ist hier der Unterschied?

Kombiniere!

Übung 11.4: Andere Kombinationen sind möglich:

(i) `data Res` $\sigma\ \alpha =$ `Res (`$\sigma\rightarrow$ `Exn [`α`])`

(ii) `data Res` $\sigma\ \alpha =$ `Res (Exn [`$\sigma\rightarrow\ \alpha$`])`

(iii) `data Res` $\sigma\ \alpha =$ `Res ([`$\sigma\rightarrow$ `Exn` α`])`

Was für eine Art Berechnung modellieren diese, und was ist hier der Unterschied?

Lösung:

(i) Berechnungen sind von einem Zustand abhängig, und geben entweder einen Fehler oder eine Liste von Ergebnissen;

(ii) Berechnung sind entweder fehlerhaft, oder eine Liste von Funktionen, die zu jedem Zustand ein Ergebnis liefern;

(iii) Berechnungen sind eine Liste von Funktionen, die zu jedem Zustand entweder ein Fehler oder ein Ergebnis liefern können.

Unterschied zwischen (i) und (ii)/(iii): für (i) kann es für einen Zustand mehrere Ergebnisse geben, bei (ii)/(iii) für einen Zustand nur ein Ergebnis/Fehler.

Nachtisch

Übung 11.5: Bonusfrage

Wir hatten also als Kombinationen

(i) `data Res` $\sigma\ \alpha =$ `Res` $(\sigma\to$ `[Exn` α`])`

(ii) `data Res` $\sigma\ \alpha =$ `Res` $(\sigma\to$ `Exn [`α`])`

(iii) `data Res` $\sigma\ \alpha =$ `Res (Exn [`$\sigma\to\ \alpha$`])`

(iv) `data Res` $\sigma\ \alpha =$ `Res [`$\sigma\to$ `Exn` α`]`

Sind das alle, fehlen noch welche, und wenn ja wieviele?

Nachtisch

Übung 11.5: Bonusfrage

Wir hatten also als Kombinationen

(i) `data Res σ α = Res (σ→ [Exn α])`

(ii) `data Res σ α = Res (σ→ Exn [α])`

(iii) `data Res σ α = Res (Exn [σ→ α])`

(iv) `data Res σ α = Res [σ→ Exn α]`

Sind das alle, fehlen noch welche, und wenn ja wieviele?

Lösung: Es fehlen noch

(v) `data Res σ α = Res [Exn (σ→ α)]`

(vi) `data Res σ α = Res (Exn (σ→ [α]))`