Christoph Lüth
Marcus Ermler
Christian Maeder
Martin Ring
Diedrich Wolter

4. Übungsblatt

Ausgabe: 14.11.12
Abgabe: 23.11.12

Im Sommer diesen Jahres konnten Kunden der RBS-Gruppe (*Royal Bank of Scotland*) in Großbritannien teilweise über eine Woche nicht auf ihre Konten zugreifen, weil die nach Indien ausgelagerte IT der Bank ein Software-Update spektakulär gegen die Wand fuhr.[1] Dieses Desaster wurde im deutschen Bankensektor nicht ohne Sorge beobachtet, und deshalb ist eine namhafte deutsche Großbank, angelockt durch den Glanz der Exzellenzinitiative, an die Uni Bremen herangetreten mit der Bitte, ihr Backofficesystem „nach den neuesten Erkenntnissen der Softwareentwicklung und Programmierung“ (mit anderen Worten, in Haskell) von Grund auf neu zu gestalten.

4.1 *The Bank of Haskell* — *10 Punkte*

In diesem Übungsblatt implementieren wir deshalb das Backofficesystem für die *Bank of Haskell*. Die Bank hat Kunden und Konten, die zusammen den Zustand der Bank bilden. Der Zustand wird durch einen algebraischen Datentyp `Bank` repräsentiert, und enthält:

- das *Eigenkapital* der Bank (das hier vereinfachend konstant bleibt[2]),
- die Stammdatensätze der *Kunden* der Bank, und
- die geführten *Konten*.

Ein einzelner Kunde wird durch eine Kundennummer `KdNr` identifiziert und durch einen algebraischen Datentyp `Kunde` repräsentiert, der folgende Daten enthält:

- die Kundennummer (`KdNr`),
- den Name,
- die Adresse,
- das Kontoüberziehungslimit, und
- die Summe der ausstehenden Kredite.

Das Kontoüberziehungslimit wird beim Anlegen des Kunden vom zuständigen Sachbearbeiter bestimmt, und bleibt der Einfachheit halber konstant. Ein Konto wird durch eine Kontonummer `KtNr` identifiziert und durch einen algebraischen Datentyp `Konto` repräsentiert, der folgende Daten enthält:

- die Kontonummer (`KtNr`),
- die Kundennummer des Kontoinhabers (ein Kunde kann also mehrere Konten führen),
- das Saldo des Kontos, und
- eine Liste der *Kontobewegungen*, jeweils bestehend aus dem Betrag (kann positiv oder negativ sein), einem Datum, und einer Anmerkung (beispielsweise `"Ueberweisung␣von␣Konto␣4834000213"`).

Die verfügbaren liquiden Mittel der Bank ergeben sich aus der Summe der Salden der Konten zuzüglich des Eigenkapitals.

Mit diesen Datentypen implementieren wir jetzt folgende Operationen für unsere Bank:

[1] Mehr dazu hier `http://www.guardian.co.uk/money/2012/jun/29/natwest-customers-without-money` und hier `http://www.theregister.co.uk/2012/06/25/rbs_natwest_what_went_wrong/`.

[2] Normalerweise würde sich das Eigenkapital durch Ausgabe von Aktien, Gewinnrücklagen oder Verluste verändern.

- einen neuen Kunden anlegen;
- ein neues Konto für einen bestehenden Kunden anlegen;
- eine Einzahlung (oder externe Überweisung) auf ein Konto;
- eine Auszahlung (oder externe Überwesung) von einem Konto;
- eine Überweisung zwischen zwei Kunden;
- ein Kontoauszug (ggf. über einen Zeitraum) für ein Konto anfordern.

Beachten Sie dabei Rand- und Fehlerfälle. Beispielsweise muss bei einer Überweisung das Quellkonto immer genügend Deckung aufwiesen, und bei einer Auszahlung darf das Überziehungslimit des Kontos nicht überschritten werden, und es müssen für die Auszahlung ausreichend liquide Mittel der Bank vorhanden sein.[3] Diese Fälle sollten keine Laufzeitfehler (`error "..."` auslösen, sondern ein geeignetes wohldefiniertes Ergebnis liefern. Kontonummer und Kundennummer sollen als Datentyp verkapselt werden, und daher können Sie davon ausgehen, dass die übergebenen Konto- und Kundennummer existieren; hier ist die Fehlerbehandlung durch einen Laufzeitfehler ausreichend.

4.2 *Easy Money?* *10 Punkte*

In dieser Aufgabe wird die *Bank of Haskell* um eine Kreditvergabe erweitert. Konkret bedeutet das die Implementation zweier zusätzlicher Operationen:

- die Vergabe eines Kredits über eine gewünschte Summe und Laufzeit (in Monaten) auf ein gewünschtes Konto,
- sowie die Rückzahlung der Rate eines Kredits (von einem Konto).

Die Vergabe von Krediten ist das Kerngeschäft einer Bank, und unterliegt vielerlei Regularien, von denen wir hier größtenteils abstrahieren. Wir betrachten nur Darlehen zu einem einheitlichen Zinssatz, die über eine Laufzeit zurückgezahlt werden sollen (also keine Annuitätskredite, bei denen die Tilgung über der Laufzeit variiert). Bei der Vergabe eines Darlehens soll folgendes geprüft werden:

- Die *Bonität* des Kunden soll geprüft werden, was wir hier wie folgt abstrahieren: die gesamten an diesen Kunden vergebenen Kredite sollen das Saldo über alle Konten des Konten multipliziert mit seinem *credit rating* (siehe unten) nicht überschreiten;
- die *Eigenkapitalquote* der Bank soll 4.5% nicht unterschreiten (sog. Basel III-Richtlinie), wobei hier die Eigenkapitalquote der Anteil des Eigenkapitals an der Summe der insgesamt vergebenen Kredite ist.

Es soll für jeden Kunden ein *credit rating* geführt werden, das die Kreditwürdigkeit eines Kunden abstrahiert. Das *credit rating* ist initial und maximal 10. Jeder gezahlte Kredit halbiert das *credit rating* (wobei Werte unter 1 auf 0 gerundet werden, d.h. es wird kein Kredit mehr vergeben), und jede Rückzahlung eines Kredites verdoppelt das *rating*, bzw. setzt das *rating* auf 1, wenn es 0 war.

Im nächsten Aufgabenblatt erweitern wir dann die *Bank of Haskell* mittels *collateralized debt obligations* und *credit default swaps* zu einer Investmentbank, so dass wir am Ende des Semesters von den zu erwartenden Boni in den wohlverdienten Ruhestand gehen können.

? *Verständnisfragen*

1. Was kennzeichnet einfach rekursive Funktionen, wie wir sie in der Vorlesung kennengelernt haben, und wie sind sie durch die Funktion `foldr` darstellbar?
2. Welche anderen geläufigen Funktionen höherer Ordnung kennen wir?
3. Was ist η-Kontraktion, und warum ist es zulässig?
4. Wann verwendet man `foldr`, wann `foldl`, und unter welchen Bedingungen ist das Ergebnis das gleiche?

[3] Es ist also möglich, dass ein Kunde kein Geld abheben kann obwohl sein Konto gedeckt ist, weil das Geld anderweitig verliehen wurde.