Universität Bremen

Praktische Informatik 3 WS 14/15

Christoph Lüth
Sandor Herms
Daniel Müller
Jan Radtke
Henrik Reichmann
Sören Schulze
Felix Thielke

7. Übungsblatt

Ausgabe: 02.12.14
Abgabe: 12.12.14

Bei der Open Night of Code an der Uni Bremen entbrannte vor einigen Tagen eine große Diskussion über die beste Implementierung einer *deterministischen* Turing-Maschine. [1]

Da viele Studenten der Praktischen Informatik 3 anwesend waren, wurde hierfür einheitlich Haskell als beste Wahl einer effizienten Programmiersprache angesehen. Wie jeder Informatiker weiß ist eine Turingmaschine ein Tupel $T = (Q, \Sigma, q_0, q_f, \square, \delta)$, wobei Q die endliche Menge der Zustände, Σ das Bandalphabet, $q_0, q_f \in Q$ der Start- und Endzustand, $\square \in \Sigma$ das Leerzeichen und $\delta \subseteq (Q \times \Sigma) \times (Q \times \Sigma \times \{L, N, R\})$ die Zustandsübergangsrelation ist.[2] Diese Modellierung kann man direkt in Haskell übersetzen, aber bei der Implementation der Zustandsübergangsrelation teilte sich der Raum in zwei Lager: die eine Gruppe vertrat felsenfest die Meinung, diese sollten als Liste von Paaren implementiert werden, während die andere sich auf die Vorlesung berief, in der stattdessen der Datentyp `Data.Map` als besonders nützlich dargestellt worden sei.

In diesem Übungsblatt wollten wir beiden Sichtweisen auf den Grund gehen und selbst ermitteln, welche Implementierung am besten ist.

Hinweis: Verwenden Sie in den Aufgaben die Listenkombinatoren `map`, `foldr`, `foldl`, `filter` usw., wo es möglich und sinnvoll ist.

7.1 *Turingmaschinen in drei Schritten* — *12 Punkte*

1. Wir implementieren zunächst das Band der Turing-Maschine als einen abstrakten Datentypen `Tape`. Das Band soll nach links und rechts potentiell unendlich sein. Wir implementieren das Band mittels zweier Listen, die jeweils das Band links und rechts von der aktuellen Position darstellen; als Konvention ist der Kopf der zweiten Liste die Position unter dem Schreib/Lesekopf:

   ```
   data Tape σ = Tape [σ] [σ]
   ```

   Jetzt benötigen wir Funktionen, um vom Band zu lesen, auf das Band zu schreiben (jeweils an der aktuellen Position des Kopfes), sowie den Kopf zu bewegen:

   ```
   read  :: Tape σ → Maybe σ
   write :: σ → Tape σ → Tape σ
   move  :: σ → Move → Tape σ → Tape σ
   ```

   Das erste Argument von `move` ist das Leerzeichen, mit welchem das Band erweitert wird, falls der Kopf über das bisherige Ende hinaus bewegt wird. `Move` ist ein algebraischer Datentyp, der angibt, in welche Richtung der Kopf bewegt wird (links, rechts, oder gar nicht).

   Ferner benötigen wir zwei Funktionen, welche aus einer Liste von Eingabezeichen (einem Eingabewort) ein Band machen (wobei der Kopf auf dem ersten Zeichen der Liste steht), und eine Funktion, welches das gesamte Band als Ausgabewort zurückgibt:

   ```
   init :: [σ] → Tape σ
   dump :: Tape σ → [σ]
   ```

   (*4 Punkte*)

2. Im zweiten Schritt modellieren wir die Zustandsübergangsrelation. Um die Implementierung austauschbar zu halten, implementieren wir sie als ADT mit folgenden Operationen:

[1] Siehe http://de.wikipedia.org/wiki/Turingmaschine
[2] Details der Modellierung können abweichen (manchmal wird eine Teilmenge des Bandalphabets als Eingabealphabet ausgezeichnet, oder es gibt mehrere Endzustände), und sind kein Reklamationsgrund.

```
data TransRel α β
next    :: Ord α⇒ TransRel α β → α→ Maybe β
empty   :: TransRel α β
targets :: TransRel α β → [β]
insert  :: Ord α⇒ TransRel α β → α→ β→ TransRel α β
```

Hierbei gibt next t a für eine Zustandsübergangsrelation t Just b zurück, falls genau ein Übergang a nach b in der Relation enthalten ist, und ansonsten Nothing. empty ist die leere Zustandsübergangsrelation (keine Übergänge), targets gibt alle b zurück, für die ein Übergang von irgendeinem a nach b enthalten ist, und insert t a b fügt einen Zustandsübergang von a nach b hinzu. (Hierbei werden keine Invarianten geprüft.)

Implementieren Sie ein Modul TransList, welches die Zustandsübergangsrelation durch eine Assoziativliste (Listen von Paaren (α, β)) modelliert.

(*4 Punkte*)

3. Jetzt können wir in einem Modul Turing Turing-Maschinen implementieren. Eine Turing-Maschine ist über dem Typ α von Zuständen und dem Bandalphabet σ parametrisiert, und besteht aus folgenden Komponenten:

   - einem Startzustand,
   - einem Endzustand,
   - dem Leerzeichen,
   - der Zustandsübergangsrelation,
   - dem aktuellen Zustand, sowie
   - dem aktuellen Band.

   Eine Turing-Maschine wird mit folgenden Funktionen erzeugt:

```
data Turing α σ
create ::  α→ α→ σ→ Turing α σ
insertTransition :: (Ord α, Ord σ)⇒ Turing α σ → (α, σ)→ (α, σ, Move)→ Turing α σ
start :: (Ord α, Ord σ)⇒ Turing α σ→ [σ]→ Turing α σ
```

   - create i f b erzeugt eine Turingmaschine mit Startzustand i, Endzustand f, Leerzeichen b sowie einer leeren Zustandsübergangsrelation.
   - insertTransition fügt eine Transition hinzu. Es dürfen nur Transitionen hinzugefügt werden, die vom Startzustand oder dem Zielzustand einer existierenden Transition ausgehen. Ferner darf keine Transition mit dem gleichen Startzustand und Eingabezeichen bereits existieren.
   - start setzt die aktuelle Position auf den Startzustand, und das Band auf den gegeben Wert (das Eingabewort).

   Mit den folgenden zwei Funktionen läßt sich die Turing-Maschine animieren:

```
step :: (Ord α, Ord σ)⇒ Turing α σ→ Res α σ
run  :: (Ord α, Ord σ)⇒ Turing α σ→ Maybe [σ]
```

   Hierbei führt step einen Schritt aus, mit folgenden drei Ergebnismöglichkeiten (die durch den algebraischen Datentyp Res $\alpha\sigma$ kodiert werden sollen):

   - entweder die Turingmaschine ist danach in einem neuen Zustand,
   - oder es gibt keine Folgezustand und die Turingmaschine ist im Endzustand, dann wird das gesamte Band zurückgegenen und die Eingabe wurde akzeptiert;
   - oder es gibt keine Folgezustand, und die Turingmaschine ist nicht im Endzustand, dann wurde die Eingabe ncht akzeptiert.

   run führt solange step durch, bis eine der beiden letzten Möglichkeiten eintritt. (*4 Punkte*)

Hinweis: Nutzen Sie zum Testen Ihrer Turingmaschine die in Aufgabe 7.3 beschriebenen Möglichkeiten.

7.2 *Zustandsübergängle als Abbildungen* *4 Punkte*

Implementieren Sie jetzt ein Modul TransMap mit derselben Signatur wie TransList, welches die Zustandsübergangsrelation durch eine endliche Abbildung (Map aus dem Modul Data.Map) implementiert.

Jetzt können Sie in der Implementierung der Turingmaschine die Zustandsübergangsrelation ersetzen, indem Sie **import** TransList mit **import** TransMap ersetzen.

Hinweis: Folgende Funktionen aus Data.Map sind dabei nützlich: lookup, empty, elems und insert.

7.3 *Praxiseinsatz!* *4 Punkte*

Um Ihre Implementierung zu testen, steht auf der Homepage der Veranstaltung ein Modul TuringTest bereit, welches eine Beispielturingmaschine implementiert.

Erweitern Sie das Testmodul, indem Sie eine weitere Turingmaschine erstellen, die Wörter der Form $\{abba\}^*$ erkennt. Wir beschränken uns auf ein Eingabealphabet $A = \{a, b\}$. Ein Wort gilt als erkannt, wenn die Maschine im Endzustand hält.

Schreiben Sie dazu eine Funktion check :: String $\rightarrow$ Bool, in die beliebige Wörter eingegeben werden können. Das Ergebnis ist True, falls das Wort erkannt wurde, ansonsten False.

Testen Sie die öffentlich sichtbaren Methoden im Modul aus Aufgabe 7.1, indem Sie die Ergebnisse der unterschiedlichen Implementierungen der Zustandsübergangsrelation vergleichen. Vergleichen Sie zusätzlich mit den *erwarteten* Ergebnissen!

Hinweis: Implementieren Sie geeignete Show-Instanzen für Turing und Tape, um sich die Ableitungen anzeigen zu lassen. Dabei sollten von einer Turingmaschine nur der aktuelle Zustand und das Band angezeigt werden. Sie können jetzt die Funktion stepper aus dem auf der Webseite bereitgestellten Modul TuringTest nutzen, um Ihre Turingmaschine schrittweise laufen zu lassen.

? *Verständnisfragen*

1. Was ist ein abstrakter Datentyp (ADT)?
2. Was sind Unterschiede und Gemeinsamkeiten zwischen ADTs und Objekten, wie wir sie aus Sprachen wie Java kennen?
3. Wozu dienen Module in Haskell?