## Theorie der Informatik

M. Helmert
F. Pommerening
Frühjahrssemester 2016

Universität Basel
Fachbereich Informatik

# Übungsblatt 8

**Abgabe: Mittwoch, 27. April 2016**

*Anmerkung:* Bei diesem Übungsblatt handelt es sich um eine Programmieraufgabe, in der Sie bestehenden Java-Code vervollständigen sollen. Wir erwarten, dass Sie Ihre Implementierung selbständig, das heisst ohne Anwendung von fremdem Code z.B. aus dem Internet erstellen. Die Java Standardbibliothek dürfen Sie selbstverständlich ohne Einschränkung verwenden.
Bei technischen Schwierigkeiten oder Verständnisproblemen helfen wir gerne weiter. Bitte wenden Sie sich dazu *mit genügend zeitlichem Abstand zum Abgabetermin* an Ihren Tutor.
Geben Sie Ihre Lösung bitte nicht auf Papier ab, sondern laden Sie sie im Courses-Portal hoch. Falls Sie Testfälle schreiben, dann geben Sie sie bitte ebenfalls mit ab (dies kann sich nur positiv auf die Bewertung auswirken).

**Aufgabe 8.1** (Turing-Maschinen; 3+2+3+2 Punkte)

Im Rahmen dieser Aufgabe soll ein Simulator für deterministische Turing-Maschinen implementiert werden.
Ein Gerüst dafür finden Sie auf der Vorlesungsseite. Implementieren Sie die vorgegebenen Methoden und wählen Sie geeignete private Felder für die Klassen. Implementieren Sie nur die durch `// TODO` gekennzeichneten Stellen; fügen Sie keine weiteren Methoden, Klassen oder öffentliche Felder hinzu.

(a) Implementieren Sie die folgenden Methoden der Klasse `Tape`, die das beidseitig unendliche Band einer Turing-Maschine (inklusiv der Position des Schreib-/Lesekopfes) repräsentieren soll:

Der Konstruktor `Tape(word, blank)` bekommt die initiale Bandbelegung und das zu verwendende Blank-Symbol übergeben. Der Schreib-/Lesekopf soll sich initial auf dem ersten Zeichen der Eingabe befinden.

Die Methode `read()` soll das Zeichen unter dem Kopf zurückgeben.

Die Methode `write(symbol)` soll das Zeichen `symbol` an der Position des Kopfes auf das Band schreiben.

Die Methoden `moveLeft()` und `moveRight()` sollen den Kopf um eine Position nach rechts oder nach links verschieben.

Die Methode `dumpAlpha()` soll den Teil des bisher verwendeten Bands ausgeben, der links der aktuellen Kopfposition ist (exklusive der aktuellen Position).

Die Methode `dumpBeta()` soll den Teil des bisher verwendeten Bands ausgeben, der rechts der aktuellen Kopfposition ist (inklusive der aktuellen Position).

Die Methode `usedSpace()` soll die Grösse (die Anzahl der Bandfelder) des bisher verwendeten Bandes zurückgeben.

(b) Implementieren Sie den Konstruktor `TuringMachine(Q, Sigma, Gamma, delta, q0, blank, E)`, wobei die einzelnen Parameter den normalen Elementen einer Turing-Maschine entsprechen. Bei der Erstellung der Turing-Maschine soll sichergestellt werden, dass sie korrekt spezifiziert ist: Ist `delta` eine totale Funktion? Ist `Sigma` Teilmenge von `Gamma`? Ist das `blank` in `Gamma`, aber nicht in `Sigma`? Ist `E` eine Teilmenge von `Q`? Werfen Sie eine `InvalidSpecificationException`, wenn die Spezifikation nicht in Ordnung ist.

(c) Implementieren Sie die restlichen Methoden der Klasse `TuringMachine`:

`initialize(word)` soll die Turing-Maschine initialisieren (d.h. die initiale Konfiguration herstellen). Stellen Sie bitte sicher, dass nur Symbole aus dem Eingabealphabet in `word` vorkommen (sonst werfen Sie eine `InvalidSpecificationException`).

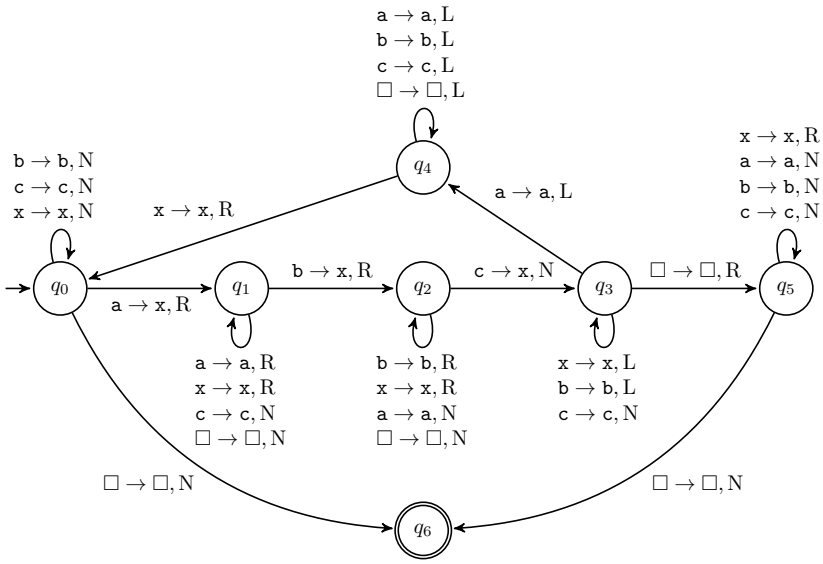
`step()` soll einen Schritt der Turing-Maschine ausführen. Werfen Sie eine `RuntimeException`, wenn die Maschine nicht initialisiert wurde oder in einem Endzustand ist.

Die Methode `dumpConfiguration()` soll die aktuelle Konfiguration der Turing-Maschine ausgeben.

Die Methode `dumpStatistics()` soll ausgeben, wieviele Schritte bereits gemacht wurden und wieviel Platz auf dem Band verwendet wurde.

`run(maxSteps)` soll die Turing-Maschine laufen lassen. Die Maschine soll nur anhalten wenn sie in einem Endzustand ist oder wenn die Anzahl Schritte `maxSteps` erreicht hat. Dies ist nützlich, um eine Maschine auf einem Wort testen zu können, auf dem sie nicht hält. Die Methode `run()` ruft `run(Integer.MAX_VALUE)` auf, um ein unbeschränktes Laufen der Turingmaschine zu simulieren. Rufen Sie `dumpConfiguration()` am Anfang der Simulation und nach jedem Schritt auf, und rufen Sie `dumpStatistics()` am Ende der Simulation auf.

(d) Betrachten Sie die folgende Turing-Maschine.




Implementieren Sie die `main`-Methode der Klasse `SimulateTM`, die ein Objekt der Klasse `TuringMachine` erstellen soll, welches die oben dargestellte Turing Maschine repräsentieren soll. Testen Sie verschiedene Eingaben für die Turing Maschine. Darunter mindestens:

- das leere Wort, und
- zwei Wörter, die in der Sprache liegen welche die Turing Maschine akzeptiert (aber nicht das leere Wort oder *aabbcc*), und
- zwei Wörter, die nicht in der Sprache liegen welche die Turing Maschine akzeptiert.

Dabei soll bei jedem Schritt die Konfiguration ausgegeben werden. Für Wörter, die nicht in der Sprache liegen, verwenden Sie bitte einen kleinen Wert für `maxSteps`, bei dem man schon sehen kann, dass die TM nicht mehr halten wird. Für Wörter die in der Sprache liegen, rufen Sie `run()` ohne Parameter auf.

Zu Ihrer Kontrolle: Auf der Eingabe *aabbcc* benötigt die Turing-Maschine 29 Schritte und verwendet 8 Bandzellen.