Prof. Dr. Christoph Bockisch
MSc Steffen Dick
Fachbereich Mathematik und Informatik
AG Programmiersprachen und -werkzeuge

Philipps Universität Marburg

Übungen zur Vorlesung
Objektorientierte Programmierung: Wintersemester 2021/2022

Nr. 10, Abgabe bis 31.01.2022

Hinweis: Sie können auf diesem Zettel mehr als die üblichen 12 Punkte erreichen, es zählen allerdings nur 12 zu Ihrem Soll. Alle weiteren erreichten Punkte werden als Bonuspunkte gewertet.

Aufgabe 10.1: NieR: Automata — 16 Punkte

Ein deterministischer endlicher Automat (DFA) lässt sich durch folgendes Quintupel eindeutig definieren: (Q, Σ, δ, q_0, F).

Q ist die Menge der Zustände eines Automaten. In dieser Menge sind alle Zustände enthalten, die ein Automat annehmen kann. Bei Σ spricht man von einem Alphabet. In einem beliebigen Alphabet, Σ, sind Symbole, σ, enthalten, mit denen ein Automat abgefragt werden kann. δ bezeichnet die Übergangs- oder Transitionsfunktion. Die Transitionsfunktion ist dabei wie folgt definiert: $\delta : Q \times \Sigma \to Q$. Dies bedeutet, dass die Funktion δ einen Zustand aus Q und ein Symbol aus Σ entgegennimmt und einen neuen Zustand aus Q als Ergebnis liefert. Bei q_0 handelt es sich um den Startzustand. Dieser wird in der Regel als q_0 bezeichnet, muss allerdings nicht diesen Namen tragen. Zuletzt handelt es sich bei F um die Menge aller akzeptierender Zustände, wobei $F \subseteq Q$ gilt.

Ein DFA akzeptiert dann ein Wort (eine Anreihung von σ), wenn der letzte erreichte Zustand in der Menge der akzeptierenden Zustände ist, also $q_n \in F$. Das besondere an einem DFA ist, dass die Funktion δ für jedes $\sigma \in \Sigma$ und jedes $q_n \in Q$ eindeutig definiert ist (es gibt für jedes Symbol und jeden Zustand genau einen Folgezustand).

In dieser Aufgabe sollen Sie einen DFA in Java programmieren.

Hinweis: Sie dürfen bei der Implementierung Ihres Automaten das leere Wort (ϵ) ignorieren.

a) Implementieren Sie die folgenden Klassen und stellen Sie sicher, dass Sie die korrekten Sichtbarkeiten verwenden und über Konstruktoren verfügen: — 1.5
   - `State`, der eine ID vom Typen `String` hält und ein `boolean`, der **`true`** gesetzt wird, wenn es sich um einen akzeptierenden Zustand handelt.
   - `Transition`, die zwei `String` IDs von Zuständen hält (eine für den Start und eine für das Ende der Transition) und einen `Character` für das Symbol der Transition.
   - `Alphabet`, das ein Array von `Character` enthält, in welchem die verfügbaren Symbole im Alphabet gespeichert werden sollen. Implementieren Sie außerdem eine Methode, die überprüft, ob ein übergebenes Symbol im Alphabet enthalten ist.

b) Legen Sie die folgenden geprüften `Exceptions` an: `StateAlreadyExists`, `StateDoesNotExist`, `SymbolNotInAlphabet` und `TransitionAlreadyExists`. Stellen Sie sicher, dass jede dieser Exceptions eine Aussagekräftige Fehlermeldung ausgibt. — 1

c) Legen Sie als Nächstes eine abstrakte Klasse `GenericAutomaton` an. Die Klasse soll über die Felder `State[] states`, `Transition[] transitions`, `Alphabet alphabet` und `String start` (hält die ID des Startzustands) verfügen. Stellen Sie sicher, dass Ihre Felder über korrekte Sichtbarkeiten verfügen (gegebenenfalls sollen erbende Klassen auf Felder zugreifen können). Legen Sie einen Konstruktor an, welchem ein `Alphabet` übergeben wird, der alle notwendigen Felder mit Default-Werten initialisiert. [1]

d) Fügen Sie `GenericAutomaton` die abstrakten Methoden `void reset()` und `boolean isAccepting()` hinzu. [1]

e) Implementieren Sie die Methode `void addState(State state, boolean isStart)` in `GenericAutomaton`, die einen neuen Zustand zur Menge der Zustände hinzufügen soll. Existiert bereits ein Zustand mit der `id` des übergebenen Zustands, soll die `Excecption StateAlreadyExists` geworfen werden. Wenn `isStart` auf `true` gesetzt wurde, soll `start` mit der ID des neuen Zustandes überschrieben werden. [2]

f) Implementieren Sie die Methode `public State findState(String id)`, die den Zustand mit der ID `id` zurückgibt. Sofern kein Zustand mit dieser ID vorhanden sein sollte, geben Sie `null` zurück. [1]

g) Implementieren Sie die Methode `protected void addTransition(Transition transition)`, die eine Transition in die Menge der Transitionen aufnimmt. Sollte einer der beiden Zustände nicht im Automaten vorhanden sein, soll eine `StateDoesNotExistException` geworfen werden. Wenn das Symbol nicht im Alphabet vorkommt, soll eine `SymbolNotInAlphabetException` geworfen werden. [2]

h) Legen Sie nun die Klasse `DFA` an. `DFA` soll von `GenericAutomaton` erben und um ein privates Feld `String current` für das Speichern der ID des aktuellen Zustands erweitern. Stellen Sie sicher, dass Ihre Klasse den Konstruktor der übergeordneten Klasse verwendet. [0.5]

i) Implementieren Sie die beiden abstrakten Methoden `void reset()` und `boolean isAccepting()` in `DFA`. Die Methode `void reset()` soll den aktuellen Zustand (`current`) auf den Startzustand zurücksetzen. Die Methode `boolean isAccepting()` soll dann `true` zurückgeben, wenn der aktuelle Zustand ein akzeptierender Zustand ist und `false` ansonsten. [1]

j) Implementieren Sie die Methode `public void makeTransition(String q1, String q2, Character symbol)`, die Ihrem Automaten eine neue Transition hinzufügt. Verwenden Sie bei Ihrer Implementierung `addTransition`. Sollte bereits eine Transition mit dem gleichen Startzustand und Symbol vorhanden sein, werfen Sie eine `TransitionAlreadyExistsException`. Stellen Sie außerdem sicher, dass die beiden `Exceptions` von `addTransition` korrekt weiter gereicht werden. [1]

k) Um die Zustände und Transitionen verwenden zu können, benötigen wir eine weitere Methode. Schreiben Sie die Methode `public String delta(Character symbol)` in `DFA`, die unter Verwendung einer ForEach-Schleife alle Transitionen durchgeht und mithilfe des aktuellen Zustands und des übergebenen Symbols den neuen aktuellen Zustand setzt und zurückgibt. [2]

l) Überlegen Sie sich ein Szenario mit einem DFA, der über mindestens 2 Zustände, 2 Symbole im Alphabet und hinreichend viele Transitionen verfügt. Verwenden Sie anschließend Ihr Szenario, um alle nach außen verfügbaren Methoden von DFA mit JUnit-Tests zu überprüfen. [2]