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. 13, Abgabe bis 21.02.2022

Hinweis: Bei diesem Zettel handelt es sich um einen Bonuszettel. Die Punkte auf diesem Zettel zählen zu Ihrem Haben und nicht zu Ihrem Soll.

Aufgabe 13.1: Nier: Replicant — 6 Punkte

Wie ein DFA (Deterministic Finite Automaton) lässt sich auch ein NFA (Non-deterministic Finite Automaton) durch folgendes Quintupel eindeutig definieren: (Q, Σ, δ, q_0, F).

Die Bedeutung der Symbole ist in diesem Fall exakt gleich. Der einzige Unterschied zwischen einem DFA und einem NFA besteht in der Transitionsfunktion δ. Wir erinnern uns:
$\delta : Q \times \Sigma \to Q$ war die Definition für den DFA. Bei einem NFA lautet die Definition
$\delta : Q \times \Sigma \to 2^Q$. Dies bedeutet, dass δ nicht wie beim DFA nur einen Zustand als Ergebnis liefern kann, sondern eine Menge an Zuständen. Innerhalb dieser Menge ist ebenfalls die leere Menge als Ergebnis erlaubt. Ein NFA ist dann akzeptierend, wenn in der aktuellen Zustandsmenge wenigstens ein Zustand aus F enthalten ist, also $\exists q_n \in Q^* | q_n \in$ F.

In dieser Aufgabe benötigen Sie Ihre Lösungen des vorangegangenen Zettels 10. Sollten Sie den Zettel nicht korrekt gelöst haben, finden Sie im Ilias eine Datei *dfa.zip* mit den für diese Aufgabe benötigten Klassen.

Hinweis: Sie dürfen davon ausgehen, dass nur ein Startzustand zulässig ist. Außerdem dürfen Sie ϵ-Transitionen ignorieren.

a) Schreiben Sie eine Klasse `NFA`, die von `GenericAutomaton` erbt. Implementieren Sie außerdem einen geeigneten Konstruktor und ein Feld für die Menge der aktuellen Zustände. — **1**

b) Implementieren Sie unter Verwendung der Methode aus der Super-klasse eine Methode zum Hinzufügen von Transitionen in `NFA`. — **1**

c) Implementieren Sie die Methode `public ArrayList<State> delta(Character symbol)`, die für jeden aktuellen Zustand eine neue Menge an Zuständen anlegt und am Ende die Menge der aktuellen Zustände mit dieser überschreibt. — **2**

d) Implementieren Sie die benötigten Methoden aus `GenericAutomaton`. — **1**

e) Testen Sie Ihre Implementierung von `NFA` hinreichend mit JUnit-Tests. — **1**