Praktische Informatik II SS 2009

Übungsblatt 8

Abgabe: 08.06.2009

Aufgabe 1 Eine allgemeine Klasse zur Verwendung von DFA

In der Vorlesung wurden **Deterministic Finite Automata** mit Hilfe einer Adjazentmatrix implementiert und ein auf dieser Datenstruktur operierender Minimierungsalgorithmus vorgestellt. Stellt eine neue Implementierung her, die in Bezug auf die Wiederverwendbarkeit besser geeignet ist und ermöglicht, den DFA mit Hilfe von Operationen systematisch aufzubauen:

1. Definiert eine Klasse `Node`, welche die Zustände des DFA repäsentiert:
   - Jeder `Node` trägt einen Namen vom Typ `String` als beliebigen Bezeichner.
   - Jeder `Node` verwendet eine **Adjazenzliste**, um die Liste der ausgehenden Kanten vom Typ `Edge` zu erfassen. Verwendet `java.util.Vector<E>` für die Adjazenzlisten, anstatt ein eigenes Listenpaket zu programmieren.
   - Ein Flag ermöglicht es, einen beliebigen `Node` als akzeptierend zu kennzeichnen.
   - Ein weiteres Flag ermöglicht es, einen beliebigen `Node` als Initialzustand zu kennzeichnen.
2. Definiert eine Klasse `Edge`, mit welcher die Kanten eines DFA spezifiziert werden können. Jede Kante trägt folgende Attribute:
   - Zielzustand, der über diese Kante erreicht werden kann.
   - Objektreferenz auf das **Label** aus dem Alphabet des DFA, welches den Zustandsübergang über diese Kante auslöst.
3. Definiert eine Klasse `DFA`, welche folgende Attribute verwaltet:
   - Der Name des Automaten als String.
   - Eine Liste von Knoten, welche die Zustände des DFA repräsentieren.
   - Eine Liste von Labeln, welche die erlaubten Elemente des Alphabets repräsentieren. Label können beliebige Objekte sein; die (z. B. bei der DFA-Minimierung benötigte) Gleichheit von Labeln wird durch die Methode `equals()` geprüft.

   Verwendet die Java-Klasse `java.util.Vector<E>` zur Speicherung von Knoten und Alphabetelementen.
4. Über den Konstruktor wird ein leerer Automat erzeugt, der nur einen Namen erhält.

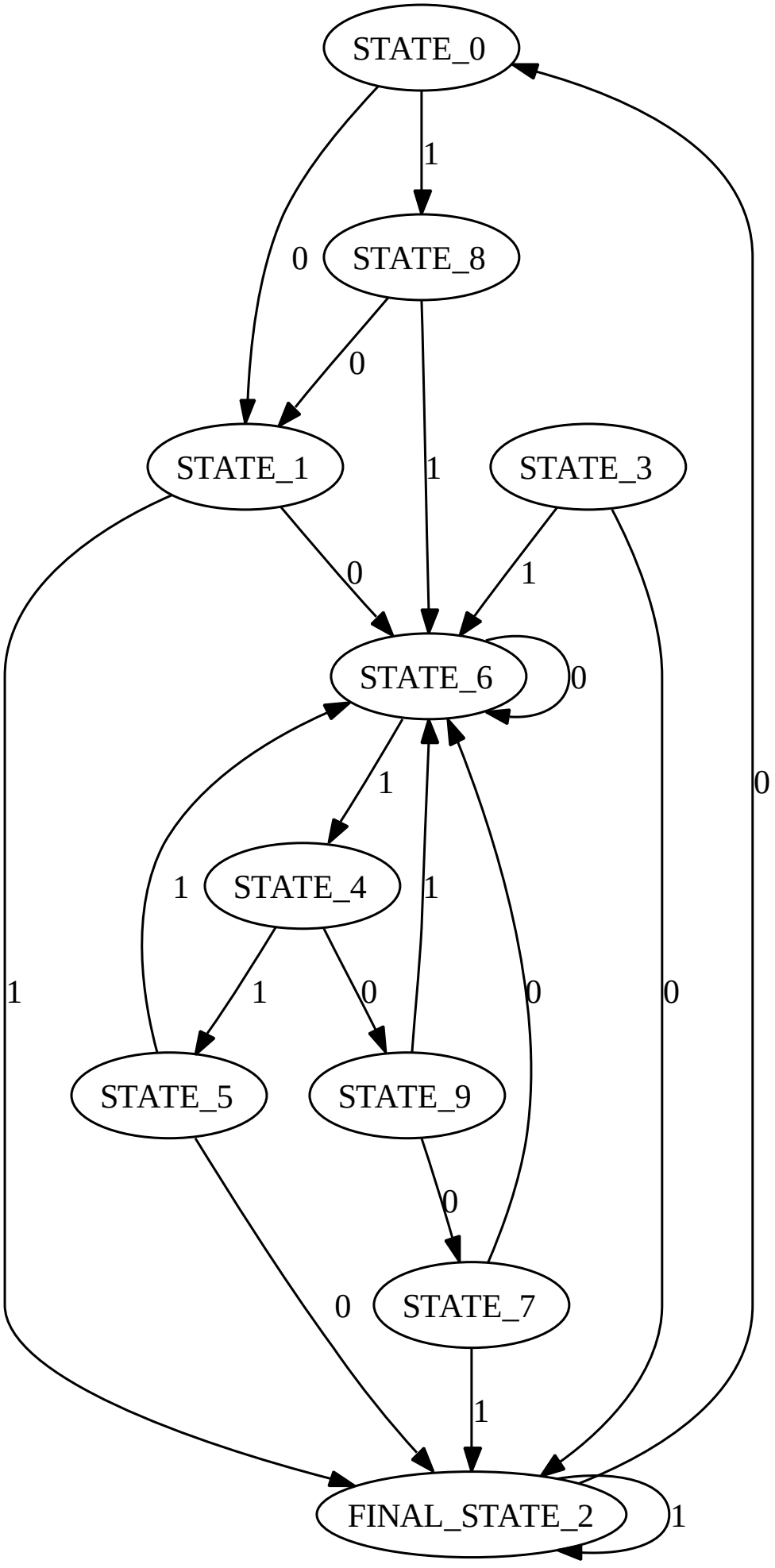
5. Entwickelt Methoden, über die man einen anfangs leeren Automaten schrittweise aufbauen kann:

   - Methode `addNode(Node n)` fügt dem DFA einen neuen Knoten hinzu. Die Knotennamen müssen eindeutig sein.
   - Methode `addLabel(Object label)` erweitert das Alphabet um `label`.
   - Methode `createEdge(String from, Object label, String to)` erzeugt eine neue Kante mit Label `label` und Zielknoten `to` und fügt diese in die Adjazenzliste von `from` ein.

6. Entwickelt folgende weitere Methoden:

   - `checkDfa()` prüft die Wohldefiniertheit des DFA nach folgenden Regeln:
     - Es existiert genau ein Initialzustand.
     - Es existiert mindestens ein akzeptierender Zustand.
     - Es werden nur erlaubte Label aus dem definierten Alphabet verwendet.
     - Für jeden Zustand ist für jedes Element des Alphabetes ein Zustandsübergang definiert.
     - Der Automat ist deterministisch: Jedes Label kommt bei jedem Knoten in genau einer Kante vor.
   - `minimiseDfa()` minimiert den DFA. Verwendet `java.util.LinkedList<E>` für die Liste, welche hierzu in der Breitensuche benötigt wird.
   - `printDfa()` gibt den DFA im dot-Format aus.
   - **Optionale Methode (+20%):** `parseDfa()` parsiert eine Datei im dot-Format (Syntaxumfang wie in der Beispieldatei `dfa.dot` aus der Vorlesung) und baut den DFA mittels `addNode()`, `addLabel()` und `createEdge()` automatisch auf. Hierbei wird angenommen, dass akzeptierende Zustände immer mit dem Präfix `FINAL_` gekennzeichnet sind und dass das Alphabet nur Strings als Label enthält.

7. Testet Eure DFA-Minimierungsprogramm an folgendem DFA (Abb. 1):


STATE_0
1
0
STATE_8
0
STATE_1
1
STATE_3
0
1
STATE_6
0
1
1
STATE_4
1
1
0
0
0
STATE_5
STATE_9
0
0
STATE_7
1
FINAL_STATE_2
1