Prof. Dr. Andreas Podelski
Dr. Matthias Heizmann
Alexander Nutz

Abgabe: 21. November 2018

5. Übungsblatt zur Vorlesung
Informatik III

Aufgabe 1: Sternoperator 2 Punkte

In Lemma 2.11 sahen wir eine Konstuktion, die für einen ε-NEA $\mathcal{B} = (\Sigma, Q, \delta, q^{\mathsf{init}}, F)$ einen ε-NEA lieferte der Sprache $L(\mathcal{B})^*$ akzeptiert. Die Konstuktion sah auf den ersten Blick unnötig kompliziert aus, da ein zusätzlicher Zustand hinzugefügt wurde. In dieser Aufgabe betrachten wir nun eine einfache Konstruktion und analysieren warum man mit dieser keinen Automaten für $L(\mathcal{B})^*$ erhält.
Wir definieren dafür die *Alternative für den Sternoperator* als den ε-NEA

$$\mathcal{B}_{\mathsf{AfS}} = (\Sigma, Q, \delta_{\mathsf{AfS}}, q^{\mathsf{init}}, F \dot\cup \{q^{\mathsf{init}}\})$$

dessen Transitionsfunktion wie folgt definiert ist.

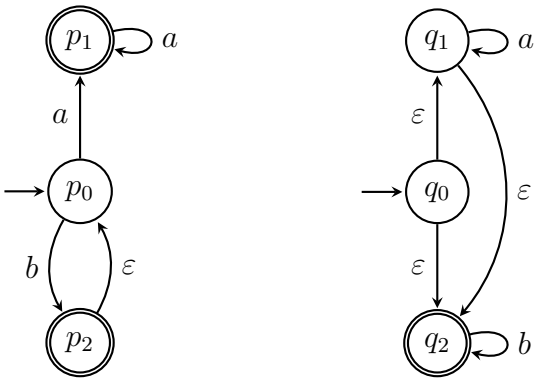
$$\delta_{\mathsf{AfS}}(q, x) = \begin{cases} \delta(q, x) & q \notin F \text{ oder } x \neq \varepsilon \\ \delta(q, x) \cup \{q^{\mathsf{init}}\} & q \in F \text{ und } x = \varepsilon \end{cases}$$

Zeigen Sie, dass der ε-NEA $\mathcal{B}_{\mathsf{AfS}}$ im Allgemeinen nicht die Sprache $L(\mathcal{B})^*$ erkennt.

Hinweis: Sie müssen als Gegenbeispiel also sowohl einen ε-NEA $\mathcal{B}$ als auch ein Wort $w \in \Sigma^*$ angeben. Den Automaten $\mathcal{B}_{\mathsf{AfS}}$ müssen Sie nicht angeben, aber (wie immer) kann Ihnen eine detaillierte Dokumentation Ihres Lösungsweges (hier u.a. durch Angabe von $\mathcal{B}_{\mathsf{AfS}}$) im Falle eines Fehlers zu Teilpunkten verhelfen.

Aufgabe 2: Konkatenation und ε-Eliminierung 1+3 Punkte

Betrachten Sie die folgenden ε-NEAs $\mathcal{B}_1$ und $\mathcal{B}_2$ über dem Alphabet $\Sigma = \{a, b\}$:



(a) Konstruieren Sie einen ε-NEA $\mathcal{B}_3$, der die Sprache $L(\mathcal{B}_1) \cdot L(\mathcal{B}_2)$ akzeptiert. Verwenden Sie dazu die Konstruktion aus der Vorlesung (Definition 2.21).

(b) Eliminieren Sie anschließend die ε-Kanten aus $\mathcal{B}_3$. Verwenden Sie wieder die Konstruktion aus der Vorlesung (Definition 2.20).

Aufgabe 3: Reguläre Ausdrücke 2,5 Punkte

Geben Sie reguläre Ausdrücke an, welche die folgenden Sprachen über dem Alphabet $\Sigma = \{a, b\}$ beschreiben.

(a) $L_1 = \{w \in \Sigma^* \mid$ auf jedes a in w folgt direkt ein b $\}$

(b) $L_2 = \{w \in \Sigma^* \mid w$ enthält das Teilwort $bb\}$

(c) $L_3 = \{w \in \Sigma^* \mid w$ enthält das Teilwort bb nicht$\}$

(d) $L_4 = \left\{ w \in \Sigma^* \;\middle|\; \begin{array}{l} w \text{ enthält genau zweimal das Symbol } a \text{ oder } w \text{ enthält} \\ \text{genau einmal das Symbol } b \end{array} \right\}$

(e) Die Sprache der Wörter mit einer geraden Anzahl b's am Ende:

$L_5 = \left\{ w \in \Sigma^* \;\middle|\; \begin{array}{l} \text{die Länge des längsten Suffixes von } w, \text{ welches nur aus} \\ b\text{'s besteht, ist gerade} \end{array} \right\}$

Aufgabe 4: Regulärer Ausdruck $\rightsquigarrow$ endlicher Automat 2 Punkte

Betrachten Sie den folgenden regulären Ausdruck über $\Sigma = \{a, b\}$.

$$(b \cdot (a \cdot b))^*$$

Konstruieren Sie zu diesem Ausdruck einen äquivalenten endlichen Automaten. Verwenden Sie jeweils die im Skript vorgestellten Konstruktionen (Satz 2.12 „$\Leftarrow$“). Zustände dürfen Sie zur Vereinfachung umbenennen.
Es genügt, den resultierenden Automaten anzugeben. Wie immer gilt: Je gründlicher sie die einzelnen Schritte dokumentieren, desto eher bekommen Sie im Falle eines Fehlers noch Teilpunkte.

Aufgabe 5: grep 2,5 Punkte

Das Unix Kommandozeilentool `grep` dient dem Finden und Filtern von Zeichenketten. Das Tool is auf allen gängingen Linuxdistributionen vorinstalliert und Sie haben es in der Vorlesung bereits kurz gesehen. Für einen regulären Ausdrucks `REGEXP` betrachtet der Befehl

```
grep -Er "REGEXP" .
```

alle Dateien im aktuellen Verzeichnis und dessen Unterverzeichnissen und liefert jeweils alle Zeilen, die eine Zeichenkette enthalten, die in der Sprache von `REGEXP` liegt.
Ihre Aufgabe ist es nun, für jede der folgenden Teilaufgaben einen regulären Ausruck für `grep` anzugeben. Wir wollen dabei `grep` mit dem Argument `-E` verwenden (`E` für *Extended* Regular Expressions).
Die Syntax von `grep` ist sehr reichhaltig. Wir wollen uns deshalb auf die folgenden Konstrukte beschränken.

Einzelne ASCII-Zeichen	Entsprechen den Elementen des Alphabets. Dabei müssen Sonderzeichen typischerweise mit einem Backslash maskiert werden. Wir schreiben also z.B. „`\?`“ statt „`?`“.
Punkt „.“	Beschreibt ein beliebiges Zeichen.
Runde Klammern „(“, „)“	Entsprechen den runden Klammern in der Notation der Vorlesung.
Strich-Operator „|“	Entspricht dem Plus-Operator für reguläre Ausdrücke aus der Vorlesung.
Stern-Operator „*“	Entspricht dem Stern-Operator für reguläre Ausdrücke aus der Vorlesung.
Eckige Klammern „[“, „]“	Mit eckigen Klammern lässt sich eine *Zeichenauswahl* beschreiben. So steht z.B. `[A-Za-z0-9]` für einen beliebigen Buchstaben oder eine beliebige Ziffer.
Hutsymbol „`^`“	Zeilenanfang
Dollarsymbol „$“	Zeilenende

Geben Sie die grep-Ausdrücke für die folgenden Suchkriterien an.

(a) Zeilen, die eine Freiburger Telefonnummer enthalten.

Eine Freiburger Telefonnummer beginnt mit `+49` oder `0`, anschließend kommt die Folge `761` und anschließend eine beliebig lange Folge von Ziffern.

(b) Zeilen, die einen *schönen Domainnamen* aus Fidschi enthalten.

Ein schöner Domainname besteht nur aus kleinen Buchstaben, Ziffern und Punkten. Dabei darf der Domainname nicht mit einem Punkt beginnen und es dürfen niemals zwei Punkte aufeinander folgen. Außerdem soll am Ende die Top-Level-Domain von Fidschi `.fj` stehen.

(c) Zeilen, die eine *schöne E-Mailadresse* aus Fidschi enthalten.

Eine schöne E-Mailadresse beginnt mit einer nicht leeren Folge, die aus kleinen Buchstaben, Ziffern und dem Minuszeichen besteht. Es folgt das At-Symbol (@) und ein schöner Domainname aus Fidschi.

(d) Zeilen, die höchstens Leerzeichen enthalten.

(e) Zeilen, die einen HTML-Tag ohne Attribut enthalten.

HTML-Tags beginnen und enden mit spitzen Klammern. Ein HTML-Tag ohne Attribut hat zwischen diesen Klammern genau eine nicht leere Folge von Buchstaben und Ziffern. Zwischen dieser Folge und den Klammern ist aber noch eine beliebig lange Folge von Leerzeichen erlaubt. Beispiele sind `<h1>`, `< title >`.