Fragen, Antworten, Kommentare und Hinweise

Evaluation schließt am 13.6. Wenn nicht getan, bitte teilnehmen (auch beim Gedanken „ist ok“, das schreibt schon jemand anders): https://evaluation.hs-osnabrueck.de/unizensus/de/sl/df4oT7KjDs13

Das Video zur Lösung der Aufgaben 43 und 44 finden Sie unter: https://youtu.be/B6gJlszyNnY (Aufgabennummern im Video stimmen nicht.)

Das Video zur Lösung der Aufgaben 45 und 46 finden Sie unter: https://youtu.be/lT1IfiXixiE (Aufgabennummern im Video stimmen nicht.)

Frage: zur Sicherheit, zum Ausdruck a(b)* gehört auch das Wort a.

Antwort: Genau, da der Kleene-Stern auch für „hoch 0“ also ε steht. Außerhalb des Hilfstools zur Überprüfung kann dann auch ab* geschrieben werden, der Kleene-Stern bindet stärker.

Frage: Wie geht das mit dem Pumping-Lemma genauer?

Antwort: Schauen wir uns $L = \{a^n b^p c^m \mid n<m, p > 0\}$ an. Der erste Versuch ist immer einen Automaten anzugeben, der die Sprache akzeptiert. Wenn der nicht gefunden wird, wird mit dem Pumping-Lemma versucht zu zeigen, dass es keinen Automaten gibt. Dazu muss als Gegenbeispiel nur ein nicht aufpumpbares oder/und schrumpfbares Wort gefunden werden. Im Beispiel ist es das Wort $a^n bc^{n+1}$. Dann muss argumentiert werden, dass es kein Teilwort x w=vxy zum Aufpumpen/Schrumpfen gibt. Hier:

x=a (oder mehrere a) verstößt irgendwann beim Aufpumpen gegen n<m.

x=ab (oder mehrere b) verstößt gegen die Struktur, da z. B. das Teilwort abab nicht erlaubt ist, nicht zu Wörtern der Sprache gehört

x=b, pumpen geht, aber schrumpfen oder weglassen ist nicht erlaubt, da ein b gefordert wird

x=bc (oder mehrere c) verstößt wieder gegen die geforderte Struktur beim Aufpumpen

x=c (oder mehrere c), pumpen geht, aber schrumpfen oder weglassen ist nicht erlaubt, da dann die Anzahl der a und b gleich werden würde

Beweise sehen oft ähnlich auf und basieren darauf, dass Automaten nicht beliebig weit zählen können.

Die Beweisidee klappt nicht, wenn mathematische Konstruktionen bei der Angabe der Sprache erscheinen.

Frage: Wie geht nochmal der zweite Schritt bei der Äquivalenzberechnung der Zustände, nachdem ich die End- und Nichtendzustände getrennt habe.

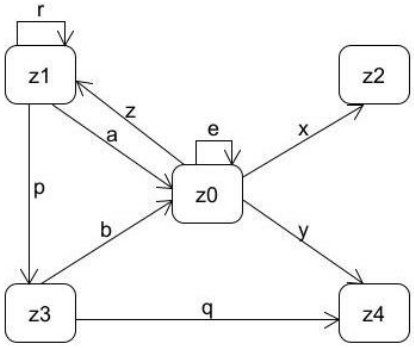
Antwort: Der 2. geht wie jeder folgende n + 1. Schritt. Sie haben ein Zustandspaar (qi,qj) was aktuell nach n Schritten noch äquivalent ist. Dann berechnen Sie für jedes Zeichen des Automaten, z. B. z das

Ergebnis ueber(qi,z)=ri und ueber(qj,z)=rj. Sie prüfen dann, ob (ri,rj) in der bisherigen Tabelle äquivalent sind, wenn ja, passiert nichts, wenn nicht, wird dies eingetragen.

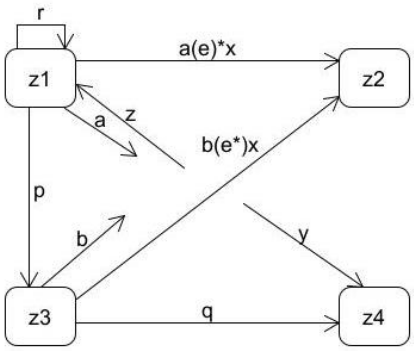
Achtung beim 2. Schritt ist das die Prüfung ob ri und rj beide Endzustände oder beide Nicht-Endzustände sind (wird im Video auch einmal an dieser Stelle argumentiert). Die Idee ist aber ab dem 3. Schritt nicht hilfreich, da Sie jetzt nicht mehr auf erreichte End- oder Nicht-Endzustände prüfen können, da jetzt ja Worte der Länge 2 also 2 Schritte betrachtet werden.

Frage: Gibt es noch ein Beispiel zur Ableitung eines regulären Ausdrucks aus einem Automaten?

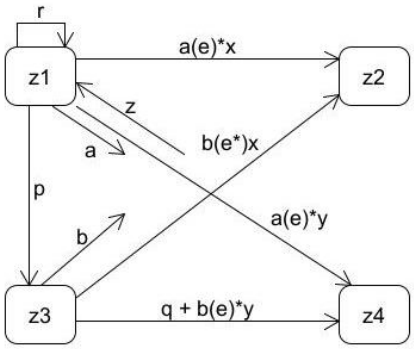
Antwort: Das Beispiel rechts zeigt schrittweise, wie ein Zustand entfernt wird. Die obere Abbildung zeigt dabei eine Ausgangssituation, die Teil einer komplexeren Berechnung sein soll und deshalb keinen Start- oder Endzustand hat. Der Zustand z0 soll entfernt werden. Zunächst ist zu erkennen, dass es zwei Kanten gibt, die nach z0 hineinlaufen und drei Kanten, die herauslaufen. Daraus folgt, dass wir insgesamt 6 (= 2*3) Berechnungen durchführen müssen.



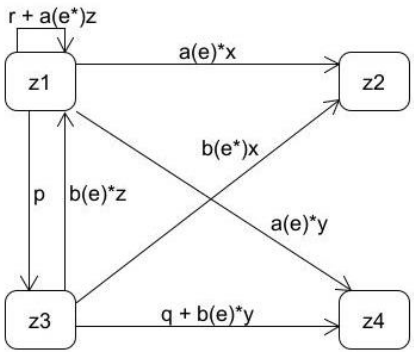
Im ersten Schritt wird die Kante von z0 mit x nach z2 entfernt. Dazu müssen für beide eingehende Kanten neue Ziele berechnet werden. Ein Weg ist von z1 über z0 nach z2 möglich. Dies führt zu einer neuen Kante von z1 nach z2. Auf dem weg wird einmal die Kante von z1 nach z2, beliebig oft die Schleife in z0 und einmal die Kante von z0 nach z2 genutzt. Dabei wird der reguläre Ausdruck a€*x abgearbeitet. Die Berechnung von z3 über z0 nach z2 laufen analog und führen zur Ausführung von b(e)*x.



Nun ist die Kante von z0 mit y nach z4 zu entfernen. Der Ansatz zur Berechnung des Weges von z1 über z0 nach z4 sollte bekannt sein. Beim Weg von z3 über z0 nach z4 ist zu beachten, dass die Kante von z3 nach z4 bereits existiert. Der neu berechnete Weg wird dann als Alternative (+) zum bereits bekannten Weg hinzugeführt.



Für die Kante von z0 mit z nach z1 wird der Weg von z3 über z0 nach z1 mit dem vorherigen Ansatz mit der neuen Kante b(e*)z ersetzt. Mit den Kanten von z1 mit a nach z0 und von z0 mit z nach z1 entsteht durch die Ersetzung eine Schleife von z1 nach z1. Der zugehörige Ausdruck a(e*)z ergänzt die schon existierende Kante als Alternative. Dies führt zum abschließenden Ergebnis in der unteren Abbildung.



Frage: Dieser domänengetriebene Ansatz ist interessant, ist eigentlich der ganze Code so aufgebaut?

Antwort: Dies gilt nur für die Basisklassen, wie Zustand und Zeichen. Generell finde ich den domänengetriebenen Ansatz sehr sinnvoll; er ist aber bei der Erstellung meist deutlich aufwändiger. Würde der Ansatz konsequent genutzt, würde z. B. die Klasse für Automaten (kommt später) nicht so anfangen:

```
public class EndlicherAutomat {
  protected List<Zustand> zustaende = new ArrayList<>();
  protected List<Zustand> endzustaende = new ArrayList<>();
  protected List<Terminal> alphabet = new ArrayList<>();
  protected Zustand start;
  protected AutomatUeberfuehrungsfunktion ueber;
```

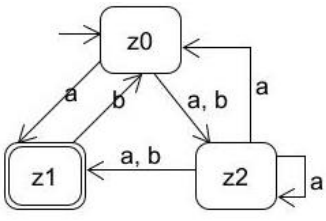
sondern so (deutlich cooler, intuitiver)

```
public class EndlicherAutomat {
  protected Zustandsmenge zustaende = new Zustandsmenge();
  protected Zustandsmenge endzustaende = new Zustandsmenge();
  protected Alphabet alphabet = new Alphabet();
  protected Zustand start;
  protected AutomatUeberfuehrungsfunktion ueberfuehrungsfunktion;
```

weitere Wiederholungsaufgaben

Die angegebenen Tests müssen aus theorieAufgabeWiederholung.zip in die Testpakete der theoriesammlung kopiert werden.

a) Begründen oder widerlegen Sie, dass folgende Sprachen von endlichen Automaten akzeptiert werden können.
   I. $L1 = \{a^i b^j c^k \mid i > 0, j > 1, k < 4\}$
   II. $L2 = \{a^i b^j c^k \mid i > 0, j > 1, k > i \text{ oder } k > j\}$



b) Wandeln Sie den Automaten auf der rechen Seite in einen sprachäquivalenten regulären Ausdruck um. Test mit test.endlicherAutomat.AusdruckAusAutomatWiederholungTest.java.

c) Geben Sie zu den beiden Sprachen aus a) kontextfreie Grammatiken an, die diese Sprachen erzeugen.
Test mit test.kontextfreieGrammatik.KFGWiederholung_L1_L2_Test.java.

d) Transformieren Sie folgende Grammatik in eine sprachäquivalente Grammatik ohne ε-Regeln.
Start -> aABb | aA A -> aaa | bBAa | B B -> ab | ba | ε
Test mit test.kontextfreieGrammatik.KFGOhneEpsWiederholungTest.java.

e) Transformieren Sie folgende Grammatik in eine sprachäquivalente Grammatik ohne Ketten-Regeln.
Start -> aABb | A A -> aaa | bBAa | B B -> ab | ba | Start
Test mit test.kontextfreieGrammatik.KFGOhneKetteWiederholungTest.java.

Das Video zur Lösung der Aufgabe ac finden Sie unter https://youtu.be/Xo2OdvxtyKk.

Das Video zur Lösung der Aufgabe b finden Sie unter https://youtu.be/TQ5aTFUYKew.

Das Video zur Lösung der Aufgabe de finden Sie unter https://youtu.be/V0o0dzCsh8U.