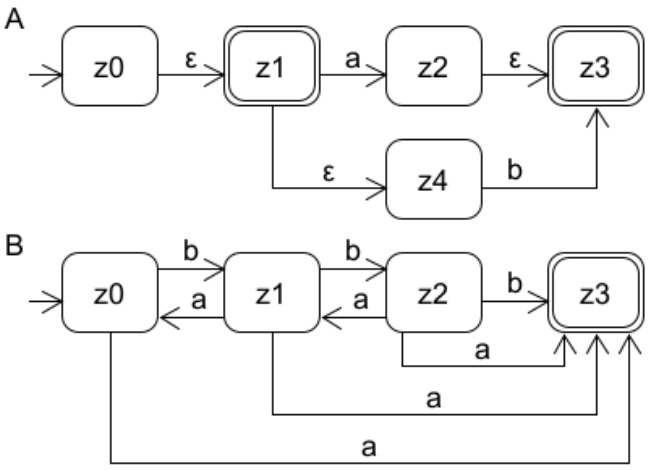
Prof. Dr. Stephan Kleuker
Hochschule Osnabrück
Fakultät Ing.-Wissenschaften und Informatik
- Software-Entwicklung -

Theoretische Informatik

Sommersemester 2023
10. Aufgabenblatt

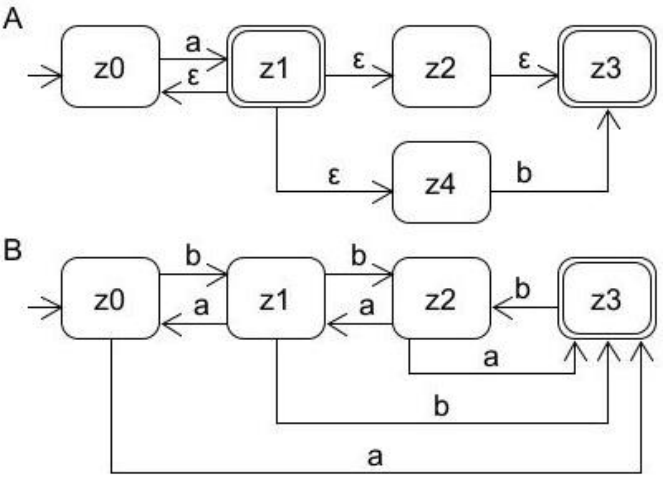
Aufgabe 39 (endliche Automaten mit ε-Übergängen [klausurähnlich], 3+7+1 Punkte)

a) Geben Sie zum Automaten A das Zustandsdiagramm eines sprachäquivalenten Automaten ohne Epsilon-Übergänge durch Anwendung des passenden Transformationsalgorithmus der Veranstaltung an.
b) Geben Sie zum Automaten B das Zustandsdiagramm eines sprachäquivalenten vollständigen deterministischen Automaten durch Anwendung des passenden Transformationsalgorithmus der Veranstaltung an.
c) Welche Sprache akzeptiert der Automat A?



Aufgabe 40 (endliche Automaten mit ε-Übergängen [klausurähnlich], 3+7+1 Punkte)

a) Geben Sie zum Automaten A das Zustandsdiagramm eines sprachäquivalenten Automaten ohne Epsilon-Übergänge durch Anwendung des passenden Transformationsalgorithmus der Veranstaltung an.
b) Geben Sie zum Automaten B das Zustandsdiagramm eines sprachäquivalenten vollständigen deterministischen Automaten durch Anwendung des passenden Transformationsalgorithmus der Veranstaltung an.
c) Welche Sprache akzeptiert der Automat A?



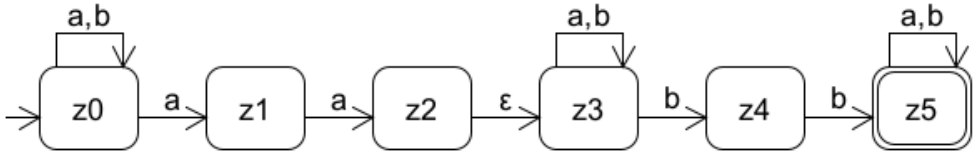
Aufgabe 41 (endliche Automaten mit ε-Übergängen)

Lesen Sie Kapitel 5.2 der theoriesammlung-Doku.

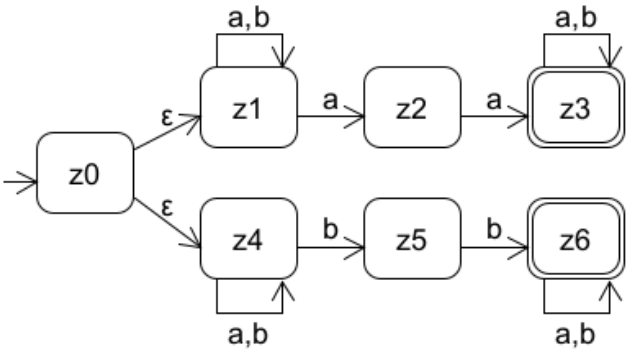
a) Geben Sie Ihren Ergebnis-Automaten aus der Aufgabe 39a) in die Datei /theoriesammlung/beispiele/endlicheautomaten/Probeklausur1.atm ein und überprüfen Sie Ihr Ergebnis mit test.endlicherAutomat.AutomatProbeklausur1Test.java.
b) Geben Sie Ihren Ergebnis-Automaten aus der Aufgabe 39b) in die Datei /theoriesammlung/beispiele/endlicheautomaten/Probeklausur2.atm ein und überprüfen Sie Ihr Ergebnis mit test.endlicherAutomat.AutomatProbeklausur2Test.java.
c) Geben Sie Ihren Ergebnis-Automaten aus der Aufgabe 40a) in die Datei /theoriesammlung/beispiele/endlicheautomaten/Probeklausur3.atm ein und überprüfen Sie Ihr Ergebnis mit test.endlicherAutomat.AutomatProbeklausur3Test.java.

d) Geben Sie Ihren Ergebnis-Automaten aus der Aufgabe 40b) in die Datei /theoriesammlung/beispiele/endlicheautomaten/Probeklausur4.atm ein und überprüfen Sie Ihr Ergebnis mit test.endlicherAutomat.AutomatProbeklausur4Test.java.

C



D



e) Geben Sie zu den Automaten C und D jeweils das Zustandsdiagramm eines sprachäquivalenten Automaten ohne Epsilon-Übergänge durch Anwendung des passenden Transformationsalgorithmus der Veranstaltung an. Geben Sie die Automaten in die Dateien AutomatCoeps.atm und AutomatDoeps.atm ein und überprüfen Sie Ihre Ergebnisse mit test.endlicherAutomat.AutomatOhneEpsTest.java.

f) Geben Sie zu den Automaten aus e) jeweils das Zustandsdiagramm eines sprachäquivalenten vollständigen deterministischen Automaten durch Anwendung des passsenden Transformationsalgorithmus der Veranstaltung an. Geben Sie die Automaten in die Dateien AutomatCdet.atm und AutomatDdet.atm ein und überprüfen Sie Ihre Ergebnisse mit test.endlicherAutomat.AutomatDeterministischTest.java.

g) Schreiben Sie einen deterministischen Automaten über dem Alphabet {a, b}, der nur Worte akzeptiert, deren Länge durch 3 teilbar ist. Geben Sie den Automaten in die Datei Automat3malZeichen.atm ein und überprüfen Sie Ihr Ergebnis mit test.endlicherAutomat.Automat3malZeichenTest.java.

Aufgabe 42 (2^n Zustände)

a) Überlegen Sie sich einen nichtdeterministischen Automaten ea mit 4 Zuständen, dessen sprachäquivalenter Automat nach Minimierung 16 Zustände hat. Bevor Sie weiterlesen sollten Sie überlegen ob Ihnen selbst eine Lösung einfällt. Ein erster Ansatz kann daraus bestehen, dass es 2^n Zeichen gibt und jedes Zeichen zu einem anderen Element der Potenzmenge verzweigt. Ein anderer, in der Vorlesung grob skizzierter, Ansatz nutzt ein Zeichen um zwischen allen Zuständen umzuschalten. Dann werden weitere Zeichen genutzt um in alle unterschiedliche Varianten von 1-, 2-, …, n-elementigen Teilmengen zu verzweigen. Überprüfen Sie Ihre Lösung mit der theoriesammlung:

```
ea.getZustaende().size() == 4
ea.deterministisch(false).minimieren().getZustaende().size() == 16
```

b) (schwierig) Schreiben Sie ein Programm, das eine Methode enthält, die eine Zahl n übergeben bekommt und als Ergebnis einen nichtdeterministischen Automaten mit n Zuständen liefert, dessen deterministischer Minimalautomat ca. 2^{n-1} Zustände hat („n-1“ da das Verfahren in theoriesammlung gegebenenfalls einen absorbierenden Zustand

ergänzt, alternativ können Sie `deterministisch(false)` aufrufen, wobei dann kein absorbierender Zustand ergänzt wird und 2^n erreichbar ist). Lassen Sie Ihren Automaten durch die theoriesammlung minimieren. Messen Sie dann die Zeit für aufsteigende Werte von n, wie lange die Minimierung dauert. Zur Zeitmessung kann einfach `long start = System.currentTimeMillis();` genutzt werden

Hinweise zur Umsetzung von 42b): Da sich die Bibliothek bisher genutzte Zustände und Zeichen merkt, ist es sinnvoll vor der Erstellung eines neuen Automaten, diese Mengen zu löschen. Dazu existieren folgende Methoden:

```
Terminal.reset(); // statt der Klasse Zeichen ist die davon abgeleitete
                  // Klasse Terminal zu nutzen
Zustand.reset();
```

Um einen beliebigen garantiert neuen Zustand zu erzeugen kann die Methode neu(), z. B.

```
Zustand tmp = Zustand.neu();
```

genutzt werden.
Um ein Zeichen zu erzeugen, kann terminal(.), z. B.

```
Terminal z1 = Terminal.terminal("a");
```

genutzt werden. Beachten Sie, dass sie beliebig viele unterschiedliche Zeichen in der Aufgabe nutzen können. Eventuell sollten Sie dazu einen Generator schreiben.
Die Klasse EndlicherAutomat hat Methoden um Zeichen zum Alphabet und Zustände zu den passenden Mengen zuzuordnen.

```
ea.addZustand(tmp);
ea.setStart(tmp);
ea.addEndzustand(tmp);
```

Zeilen zur Überführungsfunktion werden z. B. wie folgt ergänzt.

```
ea.addUeberfuehrung(tmp, z1, tmp); // vom ersten Zustand zum zweiten
                                   // im Beispiel eine Schleife
```

Abhängig vom gewählten Algorithmus, kann eine Methode hilfreich sein, die für eine Ausgangsmenge alle Teilmengen berechnet, die n Elemente enthalten, also z. B.

```
/** Berechnet alle n-Stelligen Kombinationen aus den
* Zahlen 0 bis aus-1.
* @param n Anzahl der Ergebniswerte pro Menge
* @param aus Groesse der Ausgangsmenge
* @return berechnete Kombinationen
*/
public Set<Set<Integer>> kombinationen (int n, int aus){
```

z. B: liefert `System.out.println(lsg.kombinationen(2, 5));`
`[[0, 1], [0, 2], [1, 2], [0, 3], [1, 3], [0, 4], [2, 3], [1, 4], [2, 4], [3, 4]]`

Ein Beispielergebnis sieht wie folgt aus, Anzahl gibt die Anzahl der Zustände des nichtdeterministischen Automaten, Zustaende die Anzahl der Zustände des deterministischen Automaten, bei Zeichen steht die Anzahl benötigter unterschiedlicher Zeichen und abschließend wird die benötigte Rechenzeit angegeben:

```
Anzahl      Zustaende   Zeichen     Zeit (in ms)
2           4           2           4
3           8           2           2
4           16          3           3
5           32          3           10
```

6	64	4	27
7	128	4	76
8	256	6	206
9	512	8	992
10	1024	12	5835
11	2048	18	52072
12	4096	31	272495
13	8192	50	3835226
14	16384	86	26707387

Im nächsten Schritt stellt sich die Frage, ob nicht zwei Zeichen ausreichen. Die Antwort ist ja. Suchen Sie nach so einer Lösung basierend auf dem vorherigen Ansatz oder direkt. Die Idee muss wieder sein, dass alle Elemente der Potenzmenge erreicht werden können und dass diese Zustände nicht äquivalent sind, also es für zwei Zustände garantiert ein Wort gibt, mit dem einmal ein Endzustand erreicht wird und einmal nicht.

Anzahl	Zustaende	Zeichen	Zeit (in ms)
2	4	2	4
3	8	2	4
4	16	2	6
5	32	2	21
6	64	2	51
7	128	2	135
8	256	2	557
9	512	2	3606
10	1024	2	33918
11	2048	2	317984
12	4096	2	2475399