1.3 Endliche Automaten

Nachdem wir uns veranschaulicht haben, wie man Sprachen formalisieren kann, beschäftigen wir uns mit dem für den Computer häufigeren Fall, dass der Computer eine Eingabe eines Menschen interpretieren muss.

Der erste Schritt beim Interpretieren einer menschliche Eingabe ist die Überprüfung, ob die Eingabe der korrekten Syntax folgt. Das bedeutet, dass der Computer eine Zeichenkette bekommt und ausgibt, ob die gegebene Zeichenkette zur Sprache gehört.

Der Computer arbeitet hierbei auch nicht anders, als ein Mensch - er geht alle Zeichen der Reihe nach durch und überprüft, ob dieses Zeichen passt. Um dem Computer das zu ermöglichen, muss man festlegen, welche Zeichen an welcher Stelle erlaubt ist. Die EBNF gibt uns einen guten Start:

Nehmen wir die EBNF `a{b}c` als Beispiel:

1. Es wird immer zuerst ein „a“ erwartet
2. Nun kommt ein „b“ oder ein „c“
   a) Wenn ein „b“ war, wieder zurück zu 2.
   b) Wenn ein „c“ war, darf nichts mehr kommen und wir sind fertig

Wir kennen bereits ein Diagramm, das diese Art Zustandswechsel darstellen kann: das Zustandsdiagramm. Man erweitert es um zwei Ergänzungen. Zum einen muss man definieren, welches der Startzustand ist und zum anderen muss klar sein, an welcher Stelle das Wort zu Ende sein muss. Wir nennen dies einen Endzustand. Ein Automat besteht somit aus einem Eingabealphabet (die Terminalsymbole bilden die Zustandsübergänge), einer Menge von Zuständen (hier: `Z = {z0,z1,z2}`), einem Startzustand und einer Menge von Endzuständen.

Beispiel:

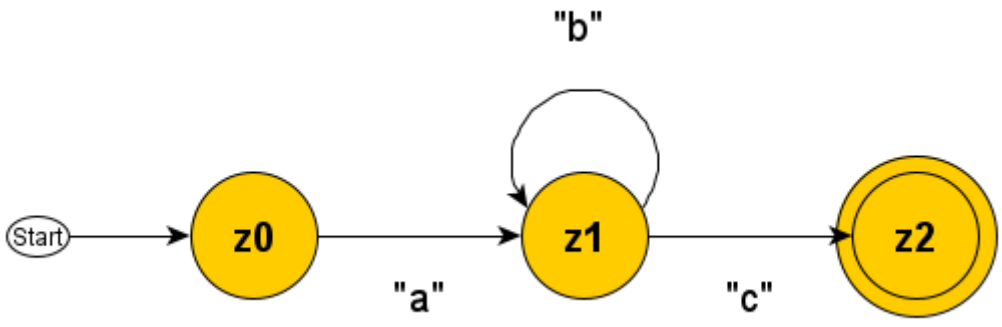
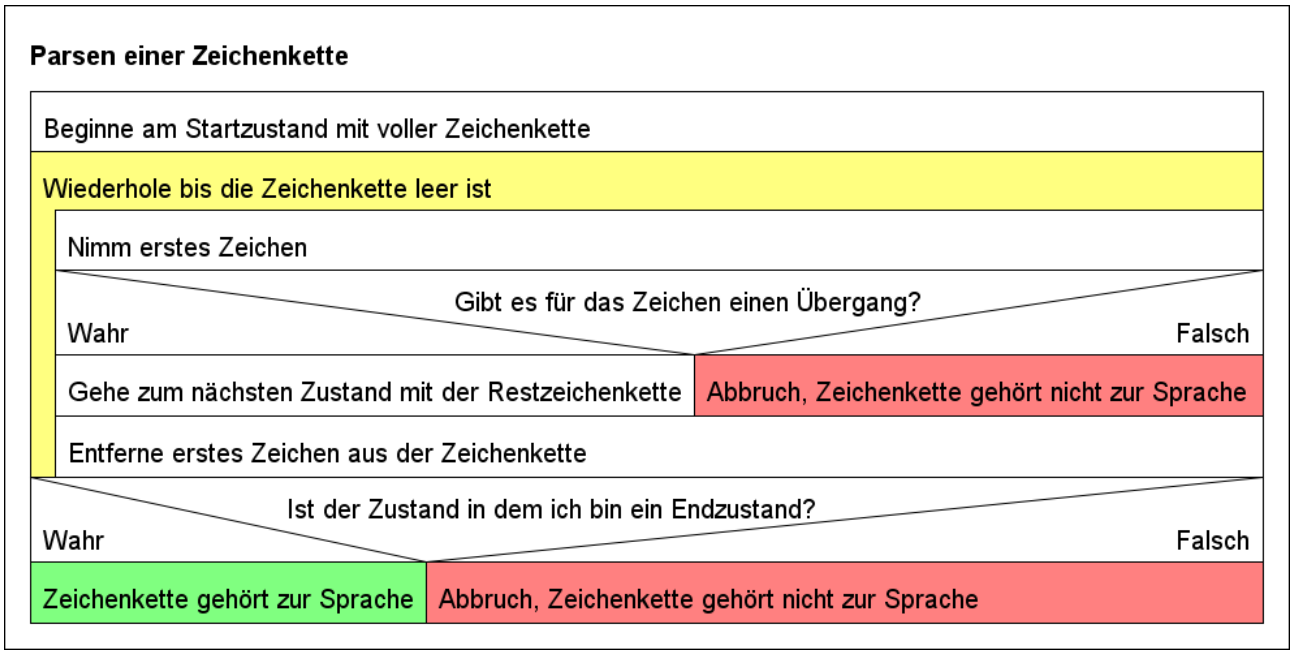


Abb. 1: Automat der Sprache a{b}c

Der Zustand `z0` ist dabei der Startzustand. Den Endzustand erkennt man am „Doppelkreis“ (hier: `z2`). Man nennt einen solchen Automaten einen **endlichen Automat**, da er endlich viele Zustände hat. Ein Automat wird **vollständiger Automat** genannt, wenn es zu jedem Zustand für jedes Eingabezeichen des Alphabets einen gültigen Zustandsübergang gibt.

Das Parsen eines Wortes läuft dann so ab, dass vom Startzustand aus alle Zeichen der Reihe nach überprüft werden. Dabei wird jeweils zum nächsten Zustand gesprungen, zu dem ein Zustandsübergang existiert. Akzeptiert wird das Wort nur dann, wenn die **Restzeichenkette leer ist und man in einem Endzustand gelandet ist**.

Struktogramm:



Beispiel: Erkennen des Wortes „`abbc`“:

Schritt	0	1	2	3	4
Zustand	z0	z1	z1	z1	z2
Restwort	`abbc`	`bbc`	`bc`	`c`	

Das Wort wird akzeptiert, da die Zeichenkette leer ist und der Automat in einem Endzustand (`z2`) anhält.

Umsetzung eines Automaten in Java

Aus unseren Vorüberlegungen wissen wir bereits einiges über die Umsetzung.

Wir brauchen eine globale Variable, die speichert, in welchem Zustand wir uns befinden. Je nachdem in welchem Zustand wir sind muss der Automat anders auf die Eingabe des nächsten Zeichens reagieren. Also brauchen wir für jeden Zustand eine Methode, die das Verhalten des Zustandes definiert. Zum Schluss brauchen wir noch eine Methode, die überprüft, ob wir am Ende in einem Endzustand gelandet sind.

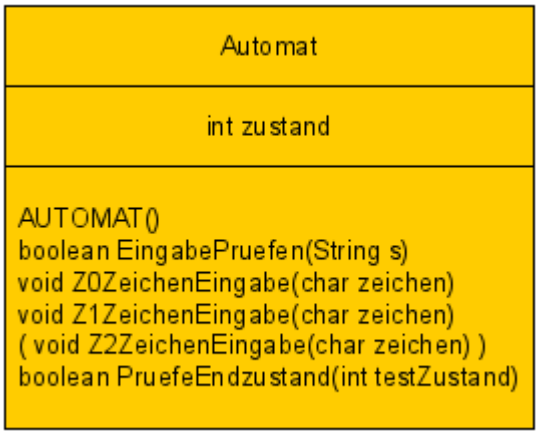


Abb. 2: Klassendiagramm

Kommen wir zu den Struktogrammen der Methoden. Die Eingabe leitet hauptsächlich die Zeichenkette Zeichen für Zeichen an die jeweilige Methode der Zustände weiter. Das macht Sie so lange, bis die Zeichenkette leer ist. Dann fragt sie, ob der Zustand, in dem angehalten wurde ein gültiger Endzustand ist:

boolean EingabePruefen(String s)

Zustand auf Startzustand
Lokale Variablen für nächstes Zeichen und Rest erstellen

Wähle erstes Zeichen aus
Setze Rest Zeichenkette neu

Zustand			
0	1	2	Fehler
Z0ZeichenEingabe(erstes Zeichen)	Z1ZeichenEingabe(erstes Zeichen)	...	return false

Solange Rest nicht leer

Ist aktueller Zustand Endzustand?	
Wahr	Falsch
Zeichenkette akzeptiert	Zeichenkette nicht akzeptiert

Für jeden Zustand muss eine Methode geschrieben werden, die das Verhalten der Zustandes definiert. Schauen wir uns das Struktogramm der Methode für Zustand 2 an, da diese in unserem Beispiel am interessantesten ist. Je nach Eingabe 'b' oder 'c' wechselt der Zustand. Alle anderen Eingaben werden als Fehler weitergegeben.

void Z1ZeichenEingabe(char zeichen)

zeichen		
'b'	'c'	sonst
Zustand bleibt bei 1	Zustand wird 2	Übergang nicht definiert Zustand z.B. auf -1 setzen, um Fehler weiter zu geben

Zum Abschluss noch die Methode `PruefeEndzustand()`. Sie überprüft mit einer Mehrfachauswahl den aktuellen Zustand und gibt einen `boolean` aus.

boolean PruefeEndzustand(int z)

z	
2	sonst
return true	return false

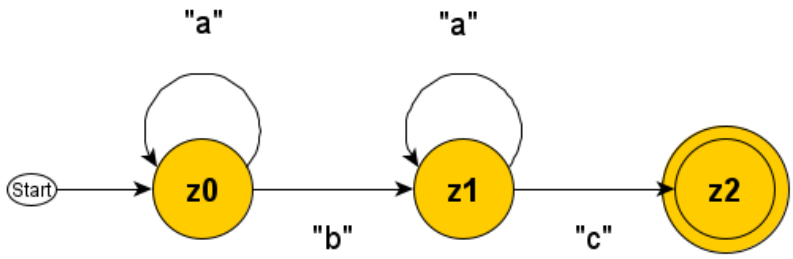
Aufgaben:

1) Erstellen Sie einen Automaten, der alle Wörter über dem Alphabet `A = {a,b,c}` akzeptiert, die mit zwei „b“ enden.

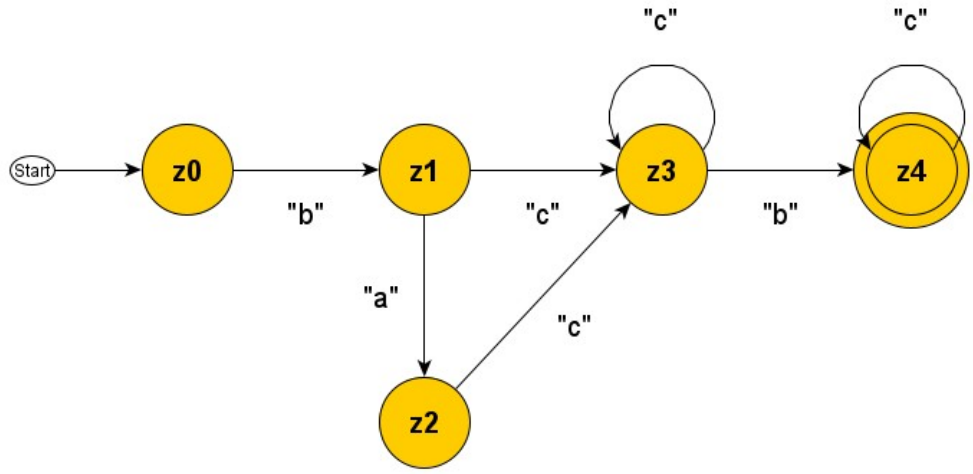
2) Erstellen Sie einen Automaten, der nur Zeichenketten über dem Alphabet `A = {d,e}` akzeptiert, in denen entweder „dd“ oder „ee“ im Wort vorkommt.

3) Erstellen Sie einen Automaten, der alle Wörter über dem Alphabet `A = {f,g,h}` akzeptiert, die mit „g“ beginnen, auf „h“ enden und deren Länge gerade ist.

4) Geben Sie an, welche Sprache von den folgenden Automaten erkannt wird:



(i) Automat



(ii) Automat

5) Programmieren Sie den Automaten zum Beispiel aus dem Hefteintrag. Programmieren Sie anschließend zu den Aufgaben 1-4 jeweils einen Automaten, der die jeweilige formale Sprache akzeptiert.