Leitprogramm

Automaten, Sprachen, Grammatiken und reguläre Ausdrücke

Stand 21. Juni 2026

Inhaltsverzeichnis

1. Vorwort **3**

2. Transduktoren **4**
2.1. Ticketautomat 7
2.2. Fehler bei der Eingabe 8

3. Akzeptoren **10**
3.1. Endlicher Automat 12
3.2. DEA und NEA 13
3.2.1. Ein Übergang ohne Eingabe: Der ϵ-Übergang 15
3.2.2. Vergleich von DEA und NEA 16
3.2.3. Vom NEA zum DEA 17

4. Umsetzung eines Automaten in ein Programm **21**

5. Grammatiken **24**
5.1. Reguläre Grammatiken 25

6. Reguläre Sprachen **28**

7. Reguläre Ausdrücke **30**

8. Nicht endliche Automaten **33**
8.1. Kellerautomat 33
8.2. Turingmaschine 35

9. Sammlung von Aufgaben **37**

A. Hinweise **39**

B. Lösungen **42**

C. Umsetzungen in Programmiersprachen **50**
C.1. Java 50
C.2. Python 51

Literatur **52**

Kapitel 1.

Vorwort

Eine alltägliche Bedeutung der Begriffe Automaten und Sprachen ist jeder Schülerin und jedem Schüler einer gymnasialen Oberstufe klar. Im Bereich der theoretischen Informatik haben diese beiden Wörter eine andere Bedeutung und beschreiben jeweils ein abstraktes Konstrukt. So geht es besonders im Bereich der Automaten nicht primär darum, wie ein Ticketautomat oder ein Geldautomat programmiert wird. Vielmehr dienen Automaten im Zusammenhang mit der Automatentheorie als Grundlage für Beschreibungen und theoriegeleitete Aussagen über Klassen von Automaten, ihre prinzipiellen Möglichkeiten und grundsätzliche Grenzen, die auch formal nachgewiesen werden können.

In diesem Leitprogramm geht es zunächst um sogenannte endliche Automaten mit Ein- und Ausgabe. Daran anschließend geht es um solche Automaten, die eine Sprache akzeptieren, wie sie von regulären Ausdrücken aufgestellt werden. Hierbei werden zwei weitere Arten unterschieden, bei denen sich zeigen lässt, dass die gleichwertig sind. Von den regulären Ausdrücken mit möglichen Anwendungen wird ein besonderer Blick auf den Begriff der Sprache geworfen. Zum Abschluss gibt es einen Ausblick zu den Kellerautomaten und der Turingmaschine, die beide keine endlichen Automaten sind.

Kapitel 2.

Automaten mit Ein- und Ausgabe (Transduktoren)

In der Automatentheorie werden zwei Arten von Automaten unterschieden. Automaten, die neben der Eingabe auch eine Ausgabe haben, werden als Transduktoren bezeichnet. Sie kommen real existierenden Automaten nah. Daher soll zu Beginn auch ein solcher realer Automat betrachtet werden: Der Schrankenautomat an der Ausfahrt des Parkplatzes des Münsterraner Zoos.



Abbildung 2.1.: Schrankenautomat des Münsterraner Zoos

Dieser Automat steuert die Schranke, indem er sie nach passender Bezahlung öffnet und nach der Durchfahrt eines Autos wieder schließt. Danach muss erneut die passende Bezahlung geleistet werden, damit der Automat die Schranke wieder öffnet.

Die Beschreibung (Automatenmodellierung) eines theoretischen Automaten setzt sich aus mehreren Angaben zusammen. Bei einigen dieser Angaben werden mehrere Elementen angegeben. Wie in der Mathematik bezeichnet man eine solche Sammlung von Elementen als Menge. Ein Automat ist nur dann vollständig angegeben, wenn alle Angaben zu ihm aufgelistet werden. Zwei dieser Angaben sind die Eingabemenge Σ und die Ausgabemenge A. Dabei zählt man alle möglichen einzelnen Eingaben und Ausgaben des Automaten auf.



Aufgabe 2.1 H

a) Zählen Sie alle möglichen einzelnen Eingaben auf, die an den Schrankenautomaten gestellt werden können.

b) Zählen Sie auch alle Ausgaben des Automaten auf. Als Ausgabe wird dabei auch das bezeichnet, was eine Benutzerin oder ein Benutzer sehen kann.

Welche Eingabe beim Automaten erfolgt ist, speichert dieser durch den Wechsel in einen entsprechenden (häufig anderen) Zustand. Die Namen aller Zustände, auch als Zustandsmenge Z bezeichnet, gehört dabei mit zu den Angaben eines Automaten, wie auch die Regeln, wann in welchen Zustand gewechselt werden soll. Die Benennung der Zustände eines Automaten ist nicht einheitlich geregelt. Oft werden sie durchnummeriert oder bekommen einen Buchstaben mit durchnummerierten Indizes. In diesem Dokument werden bis auf besondere Ausnahmen alle Zustände mit s_1, s_2 usw. bezeichnet.

Die Regeln des Automaten werden durch die Zustandsübergangsfunktion φ beschrieben. Diese wird üblicherweise graphisch in einem Zustandsübergangsdiagramm oder tabellarisch in einer Zustandsübergangstabelle dargestellt. Zur Erläuterung der graphischen Darstellung wird nur eine Schranke mit der Einwurfmöglichkeit einer Parkmünze betrachtet. Bei dieser öffnet sich die Schranke, sobald die Parkmünze eingeworfen wurde. Ist anschließend das Auto durchgefahren, so schließt sich die Schranke wieder. Die Zustände des Automaten werden dabei mit Kreisen dargestellt und die Übergänge mit Pfeilen. Die Eingaben und Ausgaben werden dabei an die Pfeile geschrieben. Gibt es sowohl Ein- als auch Ausgaben, so werden diese durch einen Schrägstrich getrennt. Außerdem ist es üblich, die Namen abzukürzen. In unserem Fall steht PM für »Parkmünze«, Ad für »Auto durchgefahren«, Sa für »Schranke auf« und Sz für »Schranke zu«. Der mit »start« markierte Pfeil an den Zustand s_1 zeigt an, in welchem Zustand der Automat

beginnt. In anderen Darstellungen kann dieses auch ein Pfeil ohne Bezeichnung sein. Dieser Startzustand z_0 gehört mit zu den nötigen Angaben eines Automaten.

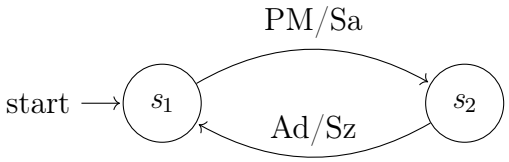


Abbildung 2.2.: Zustandsübergangsdiagramm eines Parkschrankenautomaten nur mit Parkmünze

In der tabellarischen Darstellung stehen in der obersten Zeile alle möglichen Eingaben. Die möglichen Zustände sind in der vordersten Spalte zu finden. So kann den Zellen entnommen werden, welcher neue Zustand und welche Ausgabe zu jeder Eingabe mit dem vorherigen Zustand gehören.

	PM	Ad
s_1	s_2/Sa	
s_2		s_1/Sz

Tabelle 2.1.: Zustandsübergangstabelle eines Parkschrankenautomaten nur mit Parkmünze

Aufgabe 2.2

H L

a) Geben Sie die Zustandsmenge Z des Automaten für die Parkschranke an, der die von Ihnen angegebene Eingabemenge nutzt.

b) Geben Sie den zur Zustandsmenge Z zugehörigen Startzustand z_0 des Parkschrankenautomaten an.

c) Erstellen Sie ein Zustandsübergangsdiagramm, dass die Zustandsübergangsfunktion φ angibt.

Bei der Angabe der Mengen wird die Mengenschreibweise der Mathematik verwendet. Dabei werden alle Elemente innerhalb von geschweiften Klammern geschrieben. Die Beschreibung des gesamten Automaten $(\Sigma, A, Z, z_0, \varphi)$ sieht damit wie folgt aus:

$$\Sigma = \{\text{PM, Ad, 1, 2}\}$$
$$A = \{\text{2, 1, Sa, Sz_3}\}$$
$$Z = \{s_1, s_2, s_3, s_4\}$$
$$z_0 = s_1$$
$$\varphi =$$

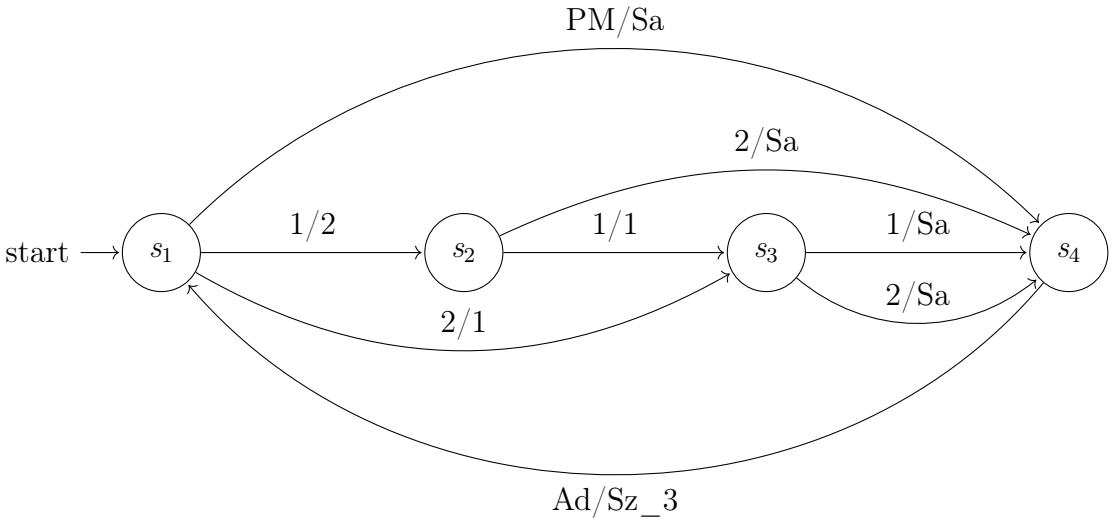


Abbildung 2.3.: Kompletter Parkscheinautomat

Wenn man sich diese Lösung ansieht, so fällt auf, dass bei der Eingabemenge und der Ausgabemenge gleiche Symbole verwendet werden. So geben die Zahlen bei der Eingabemenge die entsprechenden Geldstücke an. Bei der Ausgabe entsprechen die Zahlen der Anzeige. Dadurch bedingt werden nach dem Durchfahren des Autos zwei Sachen gleichzeitig getätigt: Die Schranke schließt sich und die Anzeige geht zurück auf 3. Dieses muss aber in einer Ausgabe »Sz_3« zusammengefasst werden.

2.1. Ticketautomat

Um das Verständnis für solche Automaten zu stärken, wird als zweites Beispiel ein einfacher Ticketautomat betrachtet. Dieser gibt nach dem Einwurf von 2,50 Euro ein Ticket aus. Wird mehr gezahlt, so gibt er zum Ticket auch das passende Rückgeld mit aus. Er soll keine Anzeige der bisher gezahlten Summe erhalten. Gezahlt werden kann nur mit 1 Euro, 2 Euro oder 50 Cent Münzen.



Aufgabe 2.3 H L

a) Erstellen Sie die vollständige Darstellung eines Automaten, der der obigen Beschreibung entspricht.

b) Erläutern Sie, weshalb der Automat nicht verändert werden muss, wenn Sie die Anzeige des bisher bezahlten Betrags realisieren möchten.

2.2. Fehler bei der Eingabe

Beim Parkschrankenautomaten wird davon ausgegangen, dass bei bestimmten Zuständen nur spezielle Eingaben folgen. Besonders bei dem folgenden Kapitel zu den Akzeptoren werden solche Annahmen gemacht. Doch was passiert, wenn eine andere Eingabe vorgenommen wird? Das beste Beispiel dafür wäre, dass ohne Einwurf passender Münzen und bei geschlossener Schranke das Auto durch fährt. Dazu wird der Automat noch einmal betrachtet, bei dem nur eine Parkmünze zum Öffnen der Schranke erlaubt war (Abbildung 2.2). Bezieht man diesen Fall auf die Realität, so wird jedem schnell klar, die Schranke ist anschließend defekt. Diesen überträgt man auch auf die Automaten. Sie bekommen einen zusätzlichen »Fehlerzustand«. In diesen führen alle Eingaben, die keinen anderen Übergang von dem betrachteten Zustand aus haben. Aus diesem Zustand kommt man anschließend aber auch nicht mehr heraus. Die oben betrachtete Zustandsübergangsfunktion würde im Diagramm dann so aussehen:

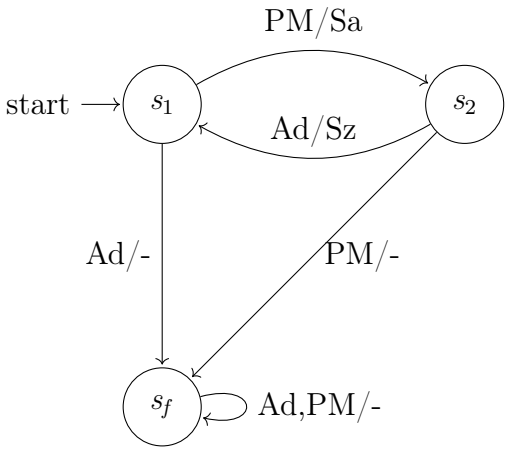


Abbildung 2.4.: Parkautomat nur mit Parkmünze incl. Fehlerzustand

Durch diesen Fehlerzustand wird ein Automat, besonders in der Zeichnung, sehr aufgebläht. Auch die Übersichtlichkeit kann dadurch verloren gehen. Deshalb ist es üblich auf den Fehlerzustand in der Darstellung der Zustandsübergangsfunktion zu verzichten.

Aufgabe 2.4

L

Betrachten Sie die Lösung des Schrankenautomaten. Welche weiteren Fälle von Eingaben wurden bei diesem nicht betrachtet. Geben Sie mit an, ob bei diesen Fällen der Übergang in den »Fehlerzustand« sinnvoll ist.

Kapitel 3.

Akzeptierende Automaten (Akzeptoren)

Die zweite Art von Automaten wird als Akzeptoren bezeichnet. Sie haben im Gegensatz zu Transduktoren keine Ausgabe, sondern ihre Aufgabe besteht darin, die nacheinander erfolgten Eingaben und ihre Reihenfolge daraufhin zu überprüfen, ob diese korrekt ist. Dieses lässt sich daran erkennen, wenn sich der Automat nach der vollständigen Eingabe in einem sogenannten akzeptierten Zustand befindet. Dabei können auch mehrere Zustände am Ende akzeptiert werden, so dass man von der Menge der akzeptierten Endzustände E spricht.

Als Beispiel soll hier ein Akzeptor für Klassenbezeichner betrachtet werden. Dieser soll prüfen, ob die Eingabe, die gemacht wird, die Konvention für Klassenbezeichner erfüllt.

$$\Sigma = \{\text{A}, \ldots, \text{Z}, \text{a}, \ldots, \text{z}, 0, \ldots, 9\}$$
$$Z = \{s_0, s_1\}$$
$$z_0 = s_0$$
$$E = \{s_1\}$$
$$\varphi =$$

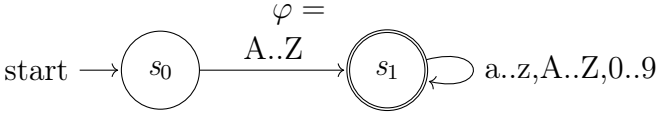


Abbildung 3.1.: Akzeptor für Klassenbezeichner

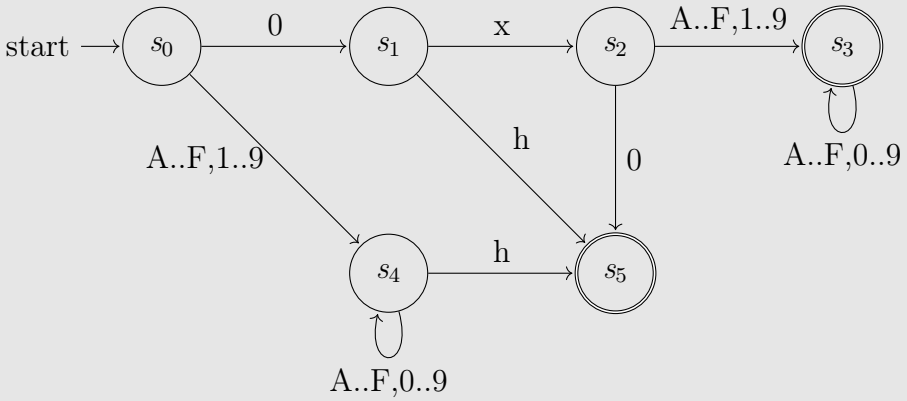
Wie in der Graphik zu erkennen, werden die akzeptierten Endzustände durch einen doppelten Kreis deutlich gemacht. Außerdem werden bei den Buchstaben Abkürzungen genutzt, um nicht alle einzelnen Zeichen aufschreiben zu müssen. Bei dem Übergang von s_1 zu sich selbst hätte man auch ein Σ an den Pfeil schreiben können, um zu verdeutlichen, dass hier alle Elemente der Eingabemenge zulässig sind. Teilweise findet man dafür in einigen Diagrammen auch ein Asterisk (*).

Aufgabe 3.1

H L

a) Beschreiben Sie die Konvention für Klassenbezeichner anhand des abgebildeten Akzeptors.

b) Gegeben ist das Zustandsübergangsdiagramm eines Akzeptors.



Geben Sie die Eingabemenge und die Menge der akzeptierten Endzustände an. Erläutern Sie außerdem, welche Eingaben von dem Automaten akzeptiert werden.

c) Erstellen Sie einen Akzeptor, der überprüft, ob eine Eingabe eine gültige Kombination aus Vor- und Nachname darstellt. Sonderzeichen müssen dabei nicht berücksichtigt werden.

3.1. Endlicher Automat

Wird in der Informatik von einem Automaten gesprochen, so meint man in den meisten Fällen einen endlichen Automaten. Diesen Namen haben die Automaten durch ihre endliche Anzahl an möglichen Zuständen. Es ist also möglich, für jeden endlichen Automaten anzugeben, wie viele Zustände er hat. Somit handelt es sich auch bei allen bisher diskutierten Automaten um endliche Automaten.

Die Endlichkeit der Automaten hat größere Auswirkungen, als man im ersten Moment vermuten würde. Sie beschränkt die Möglichkeiten der Automaten in einem besonderen Maße. Alle Zählaufgaben sind nur eingeschränkt möglich, denn für jeden Zählschritt, der durchgeführt werden soll, muss ein einzelner Zustand existieren. Verdeutlicht werden soll dies an einem Automaten, der das Vorkommen aller Neunen in einer Zahl abzählt:

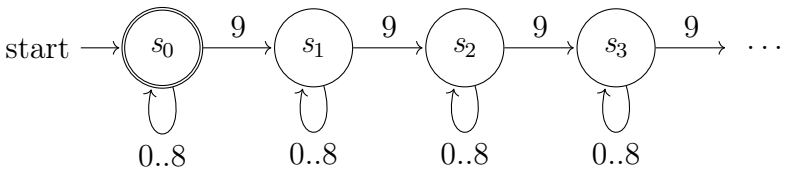


Abbildung 3.2.: Beispiel für einen unendlichen Automaten

Anhand der Nummer des Zustandes kann man die Anzahl ablesen. Da es keine Beschränkung der Länge der übergebenen Zahl gibt, benötigt der Automat unendlich viele Zustände, um seine Aufgabe voll erfüllen zu können.

Aufgabe 3.2

H L

a) Ein weiteres Beispiel für die Einschränkung eines endlichen Automaten ist ein Akzeptor, der überprüft, dass es zu jeder öffnen Klammer auch eine schließende gibt. Erstellen Sie das Zustandsübergangsdiagramm für einen Automaten, der bis zu einer Klammerungstiefe von drei dieses überprüfen kann.

b) Zeichnen Sie das Zustandsübergangsdiagramm zu einem Automaten, der eine beliebige Klammertiefe verarbeiten kann. Orientieren Sie sich dabei an der vorherigen Lösung und erklären Sie, weshalb diese Anforderung nicht mit einem endlichen Automaten realisiert werden kann.

c) Diskutieren Sie, ob es sich bei einem Computer um einen endlichen Automaten handelt.

3.2. Deterministischer und nicht deterministischer endlicher Automat

Einen Akzeptor zu entwickeln, der alle gewünschten Eigenschaften hat, ist teilweise nicht einfach. Als Beispiel soll hier die Erkennung aller Wörter dienen, die auf »ter« enden. Eine einfache Lösung dafür sieht wie folgt aus:

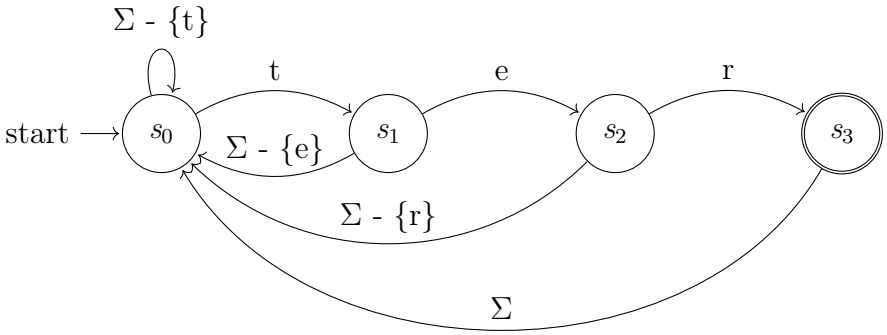


Abbildung 3.3.: Zustandsübergangsdiagramm eines Akzeptors für alle Wörter, die auf »ter« enden

Aufgabe 3.3 L

Testen Sie den oben angegebenen Akzeptor an folgenden Wörtern auf seine Korrektheit: Peter, Peterson, Wetter und lauter. Geben Sie zu jedem Zustandsübergang den entsprechenden Buchstaben und nächsten Zustand an.

Die Lösung der obigen Aufgabe zeigt, dass der angegebene Automat nicht in der Lage ist, alle Wörter korrekt zu erkennen. So sorgt beim Fall von Wetter die Kombination

aus den doppelten t dafür, das aus dem Zustand s_1 wieder in den Zustand s_0 gewechselt wird. Dadurch werden die folgenden Buchstaben nicht wie gewünscht verarbeitet.

Zur Lösung dieses Problems bietet sich ein anderer Akzeptor an, der auch als nicht deterministischer Automat (NEA) bezeichnet wird:

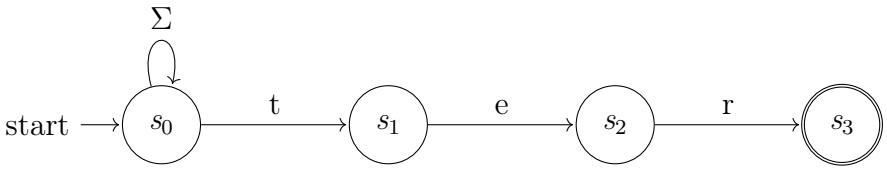


Abbildung 3.4.: Zustandsübergangsdiagramm eines nicht deterministischen Automaten für alle Wörter, die auf »ter« enden

Ein solcher NEA hat eine ganz besondere Eigenschaft: Er kann in mehreren Zuständen gleichzeitig sein. So wechselt er beim Wort »Wetter« mit dem ersten t gleichzeitig in den Zustand s_0 und s_1. Dadurch hat er die Möglichkeit beim folgenden t wieder korrekt in den Zustand s_1 zu wechseln.

Hinweis

Zum besseren Verständnis kann man einen NEA auch als Automat mit einem Orakel bezeichnen. Dieses sagt dem Automaten, wann es bei einer Eingabe welche Zustandsänderung machen soll. Durch das Vorausschauen in die Zukunft wählt der Automat so die richtigen Übergänge um korrekte Eingaben auch zu akzeptieren.

Der Begriff Determinismus kommt aus dem Lateinischen lässt sich mit »bestimmbar« übersetzen. Bei allen Aktzeptoren bis zu diesem Abschnitt lässt sich eindeutig bestimmen, in welchem Zustand sie sich nach einer Eingabe befinden. Deshalb werden diese auch als deterministische endliche Automaten oder kurz DEA bezeichnet. Genau diese Eigenschaft weist ein NEA nicht auf. Er kann sich zu jedem Zeitpunkt in verschiedenen Zuständen befinden und auch der Startzustand muss nicht eindeutig festgelegt sein. Durch diese Möglichkeiten kann man auch für komplexere Anforderungen einen Automaten mit nur wenigen Zuständen erstellen.

Aufgabe 3.4 H L

a) Bei der DNA von Zellen befinden sich an ihrem Anfang und Ende jeweils spezielle Sequenzen die sich auch in der Regel mehrfach wiederholen. Sie haben eine wichtige Steuerfunktion bei der Reproduktion der DNA. Erstellen Sie einen NEA, mit dem überprüft werden kann, ob die Startsequenz AATG und die Endsequenz TCAC in einer gegebenen DNA vorhanden ist.

b) Gegeben ist nur die Zustandsübergangstabelle eines NEA. Dabei sind die Zustände, die der Automat bei gleicher Eingabe von einem Zustand aus erreichen kann, durch ein Komma getrennt. Geben Sie an, welche Eingaben der Automat akzeptiert. Zeichnen Sie außerdem das Zustandsübergangsdiagramm auf. Startzustand ist s_0 und der akzeptierte Endzustand ist s_5.

	A	C	G	T
s_0	s_1	-	-	-
s_1	-	-	-	s_2
s_2	s_1, s_3	s_3	s_3	s_3
s_3	s_3	s_3, s_4	s_3	s_3
s_4	-	-	s_5	-
s_5	-	s_4	-	-

3.2.1. Ein Übergang ohne Eingabe: Der ϵ-Übergang

Neben der Eigenschaft, in verschiedenen Zuständen gleichzeitig zu sein, kann ein NEA auch einen Übergang in einen anderen Zustand machen, ohne dass es dazu eine Eingabe gibt. Dieser Übergang wird ϵ-Übergang genannt.

Dieser Übergang kann z. B. dann genutzt werden, wenn man einen bestehenden Automaten einfach um eine weitere Eigenschaft erweitern will. Ein Automat, der alle Eingaben akzeptiert, die eine beliebige Anzahl von »t« und »r« haben und mit einem »r« enden, kann so auch erweitert werden, dass er auch eine Wiederholung dieser Wörter akzeptiert.

Natürlich lassen sich solche Erweiterungen auch auf andere Weise realisieren. So kann man den ϵ-Übergang vom Startzustand zu einem anderen Zustand auch direkt durch einen zweiten Startzustand realisieren. Dieses bedeutet, dass es zu jedem NEA mit

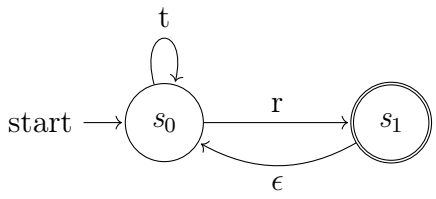


Abbildung 3.5.: Zustandsübergangsdiagramm eines Automaten mit ϵ-Übergang

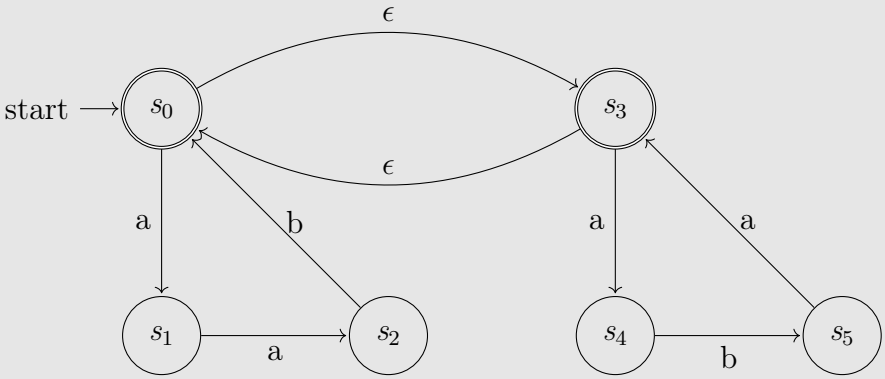
ϵ-Übergängen auch einen NEA ohne solche Übergänge gibt, der die gleiche Aufgabe erfüllt.

Aufgabe 3.5

H L

a) Erstellen Sie zu dem Zustandsübergangsdiagramm aus der Abbildung 3.5 eines entsprechendes ohne ϵ-Übergang.

b) Betrachten Sie das folgende Übergangsdiagramm für einen Akzeptor. Beschreiben Sie den Aufbau der Wörter, die dieser Automat aktzeptiert.



c) Entwickeln Sie auch zum Übergangsdiagramm aus der vorherigen Teilaufgabe ein weiteres ohne ϵ-Übergang.

3.2.2. Vergleich von DEA und NEA

Beim Vergleich von einem DEA und einem NEA ist zu erkennen, dass beide die gleiche Aufgabe erfüllen: Eingaben zu akzeptieren oder abzulehnen. Daher stellt sich die Frage, ob ein NEA durch seine zusätzliche Möglichkeit, in mehreren Zuständen gleichzeitig zu sein, wirklich mehr kann als ein DEA. Sollte dieses nicht der Fall sein, so gibt es zu jedem NEA auch einen DEA, der genau die gleichen Eingaben akzeptiert.

Betrachten wir deshalb noch einmal den ersten NEA 3.4 in diesem Leitprogramm, der alle Wörter akzeptiert, die auf ter enden. Es gibt auch einen DEA, der diese Aufgabe

erfüllt:

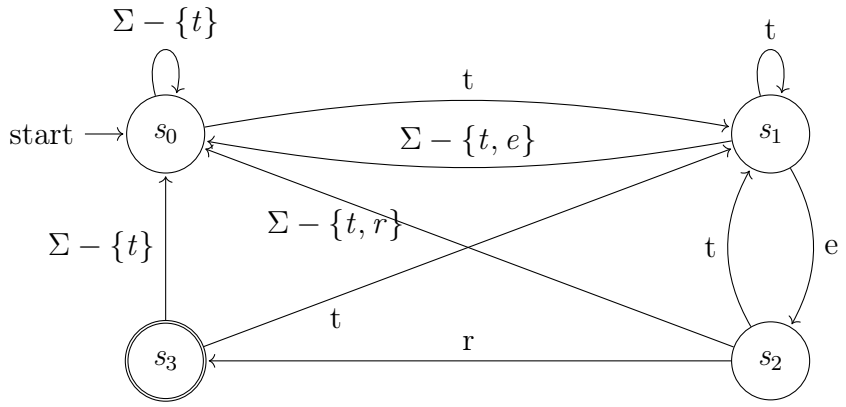


Abbildung 3.6.: Zustandsübergangsdiagramm eines deterministischen Automaten für alle Wörter, die auf »ter« enden

Für diesen speziellen Fall lässt sich also ein passender DEA zum NEA finden. Gesucht wird daher zu einem beliebigen NEA ein passender DEA. Dieses ist auch nicht so schwer, wie es auf den ersten Blick erscheint. Betrachtet man einen NEA mit n Zuständen, so kann sich dieser auch in mehreren Zuständen gleichzeitig befinden. Die Anzahl der möglichen Kombinationen von Zuständen, in den sich der NEA befinden kann, ist aber begrenzt: Es gibt nur 2^n mögliche Kombinationen. Daher ist es möglich, einen DEA zu finden, der $m \leq 2^n$ Zustände hat.

3.2.3. Vom NEA zum DEA

Um für einen NEA den entsprechenden DEA zu finden, gibt es einen einfachen Algorithmus. Dieser soll an einem einfachen NEA demonstriert werden, dessen Zustandsübergangsdiagramm in Abbildung 3.7 dargestellt ist.

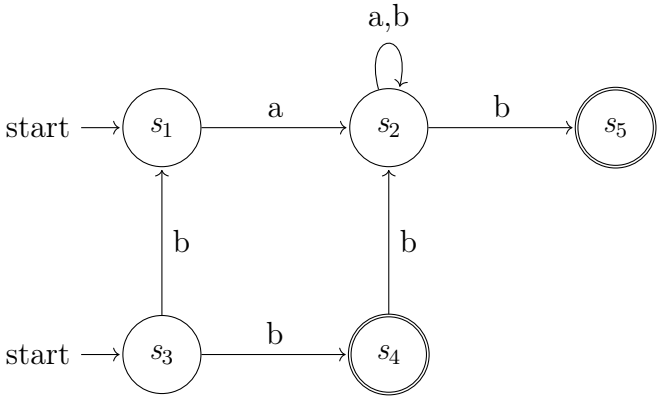


Abbildung 3.7.: Beispiel eines einfachen NEA

Zuerst wird eine Tabelle angelegt, bei der beginnend ab der zweiten Spalte alle Elemente der Eingabemenge als Spaltenkopf eingetragen werden. Die Zeilen werden in der ersten Spalte alle möglichen Zustände und Zustandskombinationen enthalten. In den folgenden Spalten werden die Zustände eingetragen, die durch entsprechende Eingabe erreicht werden können. Dazu beginnt mit allen Startzuständen des NEA, die in die erste Zelle eingetragen werden. Für den Beispiel-NEA sieht die Tabelle nun so aus:

	a	b
s_1, s_3		

Dann wird für jede Eingabe herausgesucht, in welche Zustände gewechselt werden kann vom dem Ursprungszustand aus. Diese werden dann in die entsprechende Spalte eingetragen. Von den Zuständen s_1 und s_3 gelangt man beim Beispielautomaten durch die Eingabe von einem a nur in den Zustand s_2. Daher wird dieser in der Tabelle in die entsprechende Zelle für die Eingabe a und die Kombination der Zustände Zuständen s_1 und s_3 eingetragen. Mit einem b kann man zu den Zuständen s_1 und s_4 kommen. Dementsprechend werden diese beide Zustände in die entsprechende Zelle eingetragen. Die Tabelle hat anschließend folgenden Stand:

	a	b
s_1, s_3	s_2	s_1, s_4

Im nächsten Schritt werden für alle neuen erreichbaren Zustände oder Zustandskombinationen eine neue Zeile angelegt. Im Beispiel wären dieses s_2 und s_1, s_4.

	a	b
s_1, s_3	s_2	s_1, s_4
s_2		
s_1, s_4		

Dann wird wieder überprüft, welche Zustände erreicht werden können. Auch werden ggf. neue Zeilen angelegt, wenn für einen Zustand bzw. eine Zustandskombination nicht bereits eine Zeile existiert. Deshalb sieht die Tabelle nach dem nächsten Schritt so aus:

	a	b
s_1, s_3	s_2	s_1, s_4
s_2	s_2	s_2, s_5
s_1, s_4		
s_2, s_5		

Dieses Vorgehen wird solange wiederholt, bis keine neue Zeile mehr hinzugefügt werden muss. Die vollständige Tabelle, wenn alle Kombinationen untersucht wurden, hat dann die folgende Gestalt:

	a	b
s_1, s_3	s_2	s_1, s_4
s_2	s_2	s_2, s_5
s_1, s_4	s_2	s_2
s_2, s_5	s_2	s_2, s_5

Daraus ergibt sich, dass ein DEA, der die gleichen Eingaben akzeptiert, aus vier Zuständen besteht. Diese kann man in Anlehnung an die ursprünglichen Zustände des NEA wie folgt bezeichnen: $s_{1,3}$, s_2, $s_{1,4}$ und $s_{2,5}$. Dabei sind alle Zustände als akzeptierte Zustände anzusehen, bei denen mindestens einer der Zustände, aus denen sie bestehen, selber ein akzeptierter Zustand war. In dem Beispiel sind dieses $s_{1,4}$ und $s_{2,5}$, da im NEA s_4 und s_5 akzeptierte Zustände sind.

So hat man direkt einen DEA mit Zustandsübergangstabelle. Aus dieser lässt sich auch auf Wunsch ein Zustandsübergangsdiagramm erstellen:

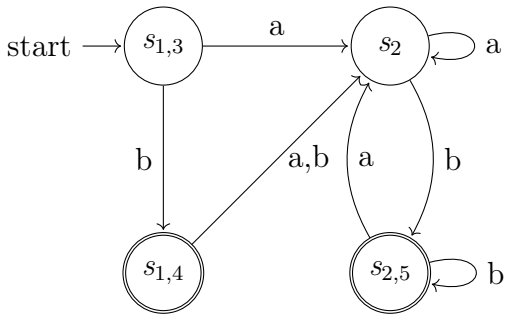


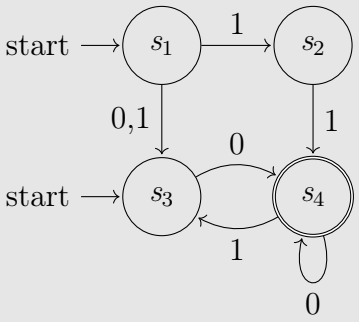
Abbildung 3.8.: Aus dem Beispiel NEA erzeugte DEA

Mit dieser Vorgehensweise lässt sich zu jedem NEA der passende DEA erzeugen. Dieses geschieht auch vor dem Hintergrund, dass die praktische Umsetzung eines NEA in einem Programm wesentlich komplizierter ist als die eines DEA. Sucht man aber zu einem Problem einen passenden Akzeptor, so lässt sich dieser aber in der Regel einfacher mit einem NEA realisieren. Bei der Aufgabe, ein Problem in ein Programm zu überführen, kann man sich diese Umstände zunutze machen, indem man zuerst einen NEA erstellt, diesen in einen DEA überführt und daraus ein Programm entwickelt. Die Übersetzung eines DEA in ein Programm wird im nächsten Kapitel erläutert.

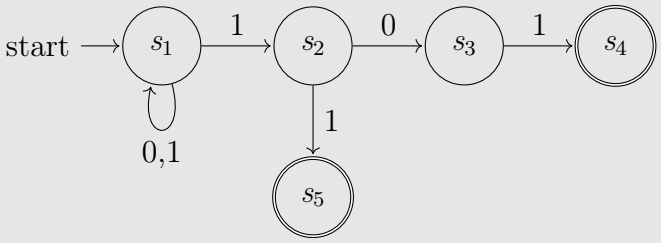
Aufgabe 3.6 L

Erstelle zu den drei gegebenen NEA jeweils den passenden DEA mit dem passenden Zustandsübergangsdiagramm:

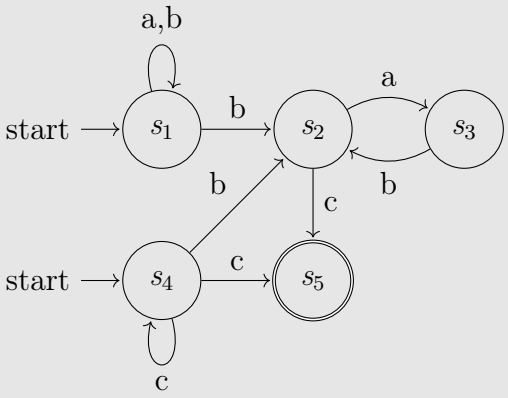
1.


start
s_1
1
s_2
0,1
1
0
start
s_3
s_4
1
0

2.


start
s_1
1
s_2
0
s_3
1
s_4
0,1
1
s_5

3.


a,b
a
start
s_1
b
s_2
s_3
b
b
c
start
s_4
c
s_5
c

Kapitel 4.

Umsetzung eines Automaten in ein Programm

Bei der Umsetzung von Automaten in Programme sollen an dieser Stelle nur Akzeptoren betrachtet werden, die auch als Parser bezeichnet werden. Um die Aufgabe eines Akzeptors übernehmen zu können, muss ein Programm folgende Teile realisieren:

1. Den aktuellen Zustand speichern und zu Beginn den Startzustand einnehmen.
2. Die Eingabe Zeichen für Zeichen durchgehen.
3. Aus dem aktuellen Eingabeelement und aktuellem Zustand auf den neuen Zustand schließen und diesen einnehmen.
4. Am Ende der Eingabe zurückgeben, ob ein akzeptierter Zustand vorliegt.

Eine Methode, die diese Elemente umsetzt, soll nun schrittweise an dem folgenden DEA vorgestellt werden. Dabei wird ein Pseudocode verwendet. Die Umsetzung in bekannte Programmiersprachen ist im Anhang (C) zu finden.

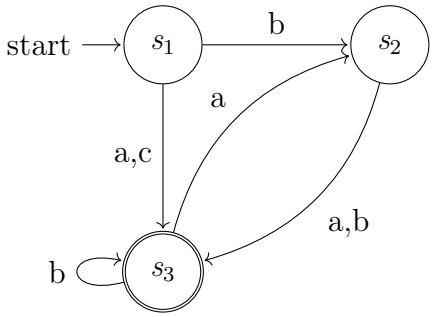


Abbildung 4.1.: DEA, der in ein Programm übersetzt werden soll

Der erste Teil, die Speicherung des aktuellen Zustands, lässt sich mit einer einfachen Variable für Zahlen realisieren. Dieser wird der Startzustand übergeben.

```
zustand <- 1
```

Für den zweiten Teil wird die Zeichenkette zeichenweise vom Anfang bis zum Ende durchgegangen. Dieses erfolgt mit einer einfachen Wiederholungsstruktur, die läuft, bis das Ende der Zeichenkette erreicht ist.

```
pos <- 0
Solange pos < Länge von wort:
    symbol <- Zeichen von wort an der Stelle pos
    ...
    pos <- pos + 1
```

Innerhalb dieser Wiederholung muss jeweils aus der Kombination von aktuellem Zustand und dem aktuellen Zeichen ein neuer Zustand gewählt werden. Dieses geschieht durch doppelt geschachtelte bedingte Anweisungen. In der ersten Ebene wird nach dem aktuellen Zustand unterschieden.

```
Wenn zustand = 1:
    ...
Sonst Wenn zustand = 2:
    ...
Sonst:
    ...
```

Da der Beispielautomat nur aus drei Zuständen besteht, werden auch nur diese drei unterschieden. Zu beachten ist dabei, dass immer mit »Sonst« gearbeitet wird, da innerhalb des Bereiches ein anderer Zustand schon gesetzt wird und damit das aktuelle Zeichen noch auf einen weiteren Zustand angewandt wird.

Innerhalb der Unterscheidung nach Zuständen, findet die Unterscheidung bezüglich des aktuellen Zeichens statt. Dabei wird dann der neue Zustand zugewiesen oder die Methode mit der Rückgabe von »Nicht wahr« beendet. Letzteres passiert, wenn es einen Übergang in den »Müllzustand« gibt. Dieses wird hier am Beispiel für den Zustand 1 verdeutlicht:

```
Wenn symbol = 'a' oder symbol = 'c':
    zustand <- 3
Sonst Wenn symbol = 'b':
    zustand <- 2
Sonst:
    Gib zurück 'Nicht Wahr'
```

Zum Ende der Methode, wenn die komplette Zeichenkette durchlaufen wurde, muss sie eine Überprüfung und entsprechende Rückgabe machen, ob sich der Automat in einem akzeptierten Endzustand befindet.

```
Gib zurück das Ergebnis von (zustand = 3)
```

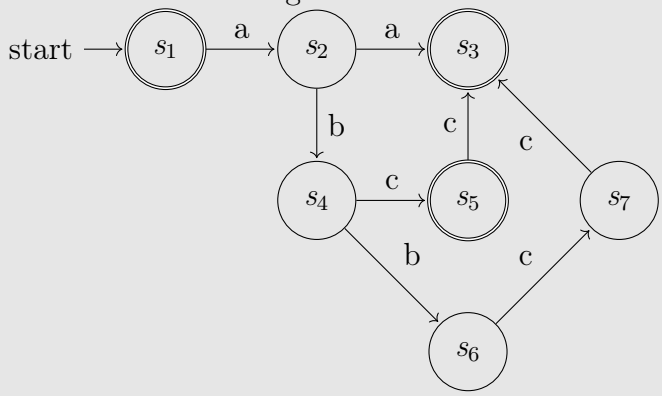
Zusammengesetzt sollte sich darauf die folgende Methode parser ergeben:

```
definiere parser (wort):
    zustand <- 1
    pos <- 0
    Solange pos < Länge von wort:
        symbol <- Zeichen von wort an der Stelle pos
        Wenn zustand = 1:
            Wenn symbol = 'a' oder symbol = 'c':
                zustand <- 3
            Sonst Wenn symbol = 'b':
                zustand <- 2
            Sonst:
                Gib zurück 'Nicht Wahr'
        Sonst Wenn zustand = 2:
            Wenn symbol = 'a' oder symbol = 'b':
                zustand <- 3
            Sonst:
                Gib zurück 'Nicht Wahr'
        Sonst:
            Wenn symbol = 'b'
                zustand <- 3
            Sonst Wenn symbol = 'a':
                zustand <- 2
            Sonst:
                Gib zurück 'Nicht Wahr'
        pos <- pos + 1
    gib zurück das Ergebnis von (zustand = 3)
```



Aufgabe 4.1

Erstellen Sie zu dem folgenden Akzeptor eine passende Methode `parser(wort)` in der im Unterricht verwendeten Programmiersprache. Testen Sie ihre Methode an verschiedenen Eingaben.



Kapitel 5.

Grammatiken

Den Aufbau der Zeichenketten, die von einem Automaten akzeptiert werden, lassen sich auch mit Hilfe von formalen Grammatiken darstellen. Als kleines Beispiel wird die Grammatik betrachtet, die nur die beiden Eingaben »abb« und »aaa« erlaubt:

$N = \{\text{A, B, C, D}\}$
$\Sigma = \{\text{a, b}\}$
$P = \{\text{A}\rightarrow\text{aB, B}\rightarrow\text{aC}|\text{bD, C}\rightarrow\text{a, D}\rightarrow\text{b}\}$
$S = \text{A}$

Diese Grammatik besteht aus Produktionsregeln P, die auf Symbole angewendet werden, um andere Symbole zu erhalten. Alle diese Symbole bilden das Vokabular V, das sich aus den Terminalsymbole Σ und den Nichtterminalsymbole N zusammensetzt. Die Menge der Terminalsymbole entspricht dabei der Eingabemenge eines Automaten. Für die Nichtterminalsymbole werden in der Regel Großbuchstaben gewählt, die zum Aufstellen der Produktionsregeln benötigt werden. Begonnen wird dabei mit dem in S angegebenen Nichtterminalsymbol, dem Startsymbol.

Bei der Bildung der durch die Grammatik beschriebenen Wörter, wird immer ein Nichtterminalsymbol durch die Anwendung einer entsprechenden Produktionsregel ersetzt. So wird zur Bildung von »abb« aus A erst aB, dann abD und zum Schluss abb. Der vertikale Strich bei den Produktionsregeln steht dabei für ein Oder. Dadurch wird es ermöglicht, dass zwischen verschiedenen Möglichkeiten bei der Ersetzung ausgewählt werden kann. Außerdem ist die Ersetzung durch ϵ möglich, das als das leere Wort bezeichnet wird. In diesem Fall wird das Nichtterminalsymbol entfernt.



Aufgabe 5.1 H L

a) Geben Sie eine Erklärung zur Wortwahl bei den Begriffen Terminalsymbole und Nichtterminalsymbole an.

b) Gegeben ist folgende Grammatik:

$N = \{A, B, C, D, E\}$
$\Sigma = \{a, b, c\}$
$P = \{A \rightarrow Ba, B \rightarrow aC|bD, C \rightarrow \epsilon|aB, D \rightarrow bC|cE, E \rightarrow aB|\epsilon\}$
$S = A$

Analysieren Sie, welche Wörter damit gebildet werden können und geben Sie das Zustandsübergangsdiagramm eines entsprechenden Automaten an.

c) Erstellen Sie eine Grammatik für Wörter die aus einer fast beliebigen Reihenfolge von a und b bestehen. Einschränkung dabei soll aber sein, dass wenn exakt nur zwei a oder b folgen, soll danach ein c stehen (Beispiel: ababbcaaabaac). Vergleichen Sie diese Grammatik mit dem Ergebnis von einem anderen Mitschüler.

5.1. Reguläre Grammatiken

Eine Untergruppe der Grammatiken sind die regulären Grammatiken. Bei ihnen werden besondere Bedingungen an die Produktionsregeln gestellt. Bei ihnen dürfen nur Ersetzungen durch ein Terminalsymbol, das leere Wort oder die Kombination von einem Terminal- und einem Nichtterminalsymbol genutzt werden. Dabei müssen die Kombinationen innerhalb der Grammatik immer die gleiche Reihenfolge haben. Je nach Reihenfolge wird dabei von einer rechtsregulären oder einer linksregulären Grammatik gesprochen.

Bei der linksregulären Grammatik, teilweise auch linkslineare Grammatik genannt, dürfen also nur Produktionsregeln der Form A→Ba, A→a oder A→ϵ genutzt werden. Wörter die dadurch zusammengebaut werden, wachsen in die linke Richtung, wodurch sich auch der Name ableitet. Analog sind bei der rechtsregulären (rechtslineare) Grammatik nur Form A→aB, A→a oder A→ϵ erlaubt. Dadurch verlängern sich die Wörter immer nach rechts.

Diese regulären Grammatiken haben den Vorteil, dass von ihnen sehr einfach ein Akzeptor abgeleitet werden kann. Die Gründe dafür werden später noch genauer beleuchtet. Es geht dabei soweit, dass sich für die akzeptierten Worte eines Akzeptors einfach eine Grammatik aufstellen lässt. Der Umkehrschluss, dass sich zu einer nicht regulären Grammatik kein Akzeptor bauen lässt, trifft aber nicht zu.



Aufgabe 5.2

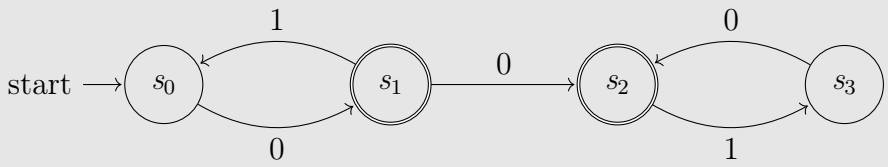
H L

a) Geben Sie aufgrund der Produktionsregeln an, ob die dazugehörige Grammatik regulär ist und wenn ja, in welche Richtung.

1. A→Aa|Ba, B→Cb, C→Cb|Da, D→Daa|Eb, E→b
2. A→aB|bB, B→bC, C→ϵ|g|d|bA
3. A→aBa, B→aB|bC, C→bC|bD, D →aE|bE, E→b|bE

b) Modifizieren Sie die in der vorherigen Aufgabe angegebenen Grammatiken so, dass sie regulär sind.

c) Erstellen Sie zum folgenden, im Zustandsübergangsdiagramm angegebenen Akzeptor eine linksreguläre Grammatik.


start
s_0
s_1
s_2
s_3
1
0
0
0
1

Kapitel 6.

Reguläre Sprachen

Mit dem Begriff Sprachen werden Sie wahrscheinlich als erstes die natürlichen und zum größten Teil gesprochene Sprachen verbinden. Daneben gibt es aber auch die konstruierten Sprachen. Zu diesen gehören auch die formalen Sprachen. Beispiele für formale Sprachen sind die Programmiersprachen, aber auch die in der theoretischen Informatik genutzten Sprachen. Zu diesen gehören auch alle regulären Sprachen.

Die allgemeine Definition einer Sprache L ist, dass sie aus Wörtern besteht, die aus einem Alphabet Σ gebildet werden. Jedes Wort besteht also nur aus aneinander gehangenen Zeichen eines festen Alphabets. Um die Wörter einer Sprache anzugeben, bedient man sich einer Aufzählung sowie der Benutzung verschiedener Hilfsmittel. So besteht die Sprache $L = \{\text{abc}, \text{bbc}\}$ nur aus eben den beiden angegeben Wörtern. Die Sprache $L = \{\text{a}^n\text{b}^m | n, m \geq 0\}$ aus allen Wörtern, die aus einer beliebigen Anzahl von a, gefolgt von einer beliebigen Anzahl von b zusammen gesetzt sind. Das Leere Wort ϵ ist damit auch Teil dieser Sprache.

Damit eine Sprache zu den regulären Sprachen gezählt wird, muss sie einige Eigenschaften erfüllen: Es muss einen endlichen Automaten geben, der alle Wörter der Sprache und kein weiteres Wort akzeptiert. Alternativ muss es möglich sein, zur Sprache eine reguläre Grammatik aufstellen zu können. Auch müssen sich die Wörter der Sprache mit einem regulären Ausdruck beschreiben lassen. Diese Ausdrücke werden im nächsten Kapitel genauer behandelt. Diese vier Angaben sind äquivalent zueinander: Die Sprache, die ein endlicher Automat akzeptiert, ist eine reguläre Sprache und sie lässt sich auch durch eine reguläre Grammatik beschreiben. Wie aber bereits im vorherigen Kapitel angedeutet, kann man aber nicht sagen, dass z. B. die Sprache einer nicht regulären Grammatik keine reguläre Sprache ist.

Durch diese Bedingungen ergeben sich die gleichen Einschränkungen, wie auch für Automaten. So sind alle Sprachen, in denen gezählt werden muss, wie z. B. bei Klammerausdrücken, keine regulären Sprachen. Dieses gilt also für Programmiersprachen, bei denen darauf geachtet werden muss, dass es zu jeder sich öffnenden Klammer auch eine schließende gibt. Alle möglichen Bezeichner einer Methode bilden aber eine reguläre Sprache.

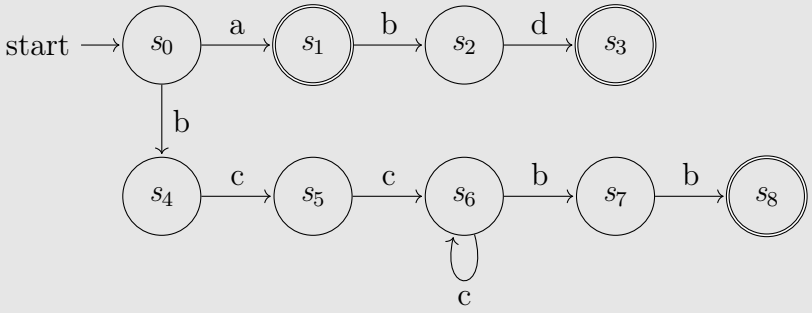
Aufgabe 6.1

L

a) Geben Sie begründet an, welche der folgenden Sprachen regulär sind und welche nicht:

1. Die erlaubten Kommentare in Ihrer Programmiersprache
2. Die erlaubten HTML-Texte
3. Die korrekten arithmetischen Ausdrücke
4. Die erlaubten Dateinamen
5. Die erlaubten WWW-Adressen

b) Geben Sie die Sprache an, die folgender Automat akzeptiert:



Kapitel 7.

Reguläre Ausdrücke

Eine Möglichkeit eine reguläre Sprache zu beschreiben sind die regulären Ausdrücke. Obwohl die Grundidee hinter regulären Ausdrücken einfach ist, können einige Ausdrücke so kompliziert werden, dass selbst erfahrene Benutzer zwar noch in der Lage sind, sie selbst zu schreiben, nicht aber einen solchen komplizierten Ausdruck von anderen Personen sofort zu verstehen. Dabei passt ein Ausdruck auf eine bestimmte Menge von Wörtern, die dann die zugehörige Sprache bilden. Dabei ist es auch möglich, dass er im einfachsten Fall nur auf ein einziges Wort passt.

Eigentlich werden reguläre Ausdrücke eher im Bereich der Suche nach Mustern genutzt, das auch **Pattern Matching** genannt wird. Ein sehr bekanntes Beispiel für den Einsatz von regulären Ausdrücken ist das Programm »grep«, mit dem man in Dateien nach bestimmten Vorkommnissen suchen kann. Grep liefert dann alle Zeilen, in denen ein Teil der Zeile auf den regulären Ausdruck passt:

```
$ grep -n -E "Kuss" romeo_julia.txt
2513:Im Kusse sich verzehrt.  Die Süßigkeit
4430:Des Odems, siegelt mit rechtmäßgem Kusse
4441:Dein Trank wirkt schnell.--Und so im Kusse sterb ich.
$
```

Hier wurde in der Datei »romeo_julia.txt« *Free ebooks - Project Gutenberg* 2014, `https://www.gutenberg.org/ebooks/6996` nach dem Vorkommen von Kuss gesucht. Das Ergebnis liefert die Zeilen, die »Kusse« enthalten. Der Text enthält das Wort Kuss nicht. Bei einer Zeile wird durch grep geprüft, ob der reguläre Ausdruck auf einen Teil der Zeile passt. Dieses ist bei dem Wort Kusse der Fall, da das Wort Kuss Teil von Kusse ist.

Der Parameter »-n« bei grep ist dabei verantwortlich dafür, dass die Zeilennummer angeben wird. Durch das »-E« wird die moderne Syntax der regulären Ausdrücke genutzt. Dieser Parameter sollte immer mit angeben werden, da es für reguläre Ausdrücke keine einheitliche Schreibweise gibt und so die hier angegebene genutzt werden kann.

Bei dem Ausdruck »Kuss« wurden nur die vier Zeichen aneinander gehangen. Um aber auch gleichzeitig nach Küssen zu suchen muss man die beiden Teile durch ein |voneinander trennen. Es stellt, wie bei den Grammatiken den Operator »Oder« dar. Der reguläre Ausdruck wäre dann »Kuss|küssen«.

```
$ grep -n -E "Kuss|küssen" romeo_julia.txt
572:Die schöner Frauen Stirne küssen, bringt
2513:Im Kusse sich verzehrt.  Die Süßigkeit
3439:Ich steig hinab; laß dich noch einmal küssen!
4430:Des Odems, siegelt mit rechtmäßgem Kusse
4441:Dein Trank wirkt schnell.--Und so im Kusse sterb ich.
4552:Dir deine Lippen küssen.  Ach, vielleicht
$
```

Hierbei wurde aber nicht beachtet, dass »küssen« auch als Nomen oder am Satzanfang verwendet werden kann. Anstatt drei Wörter direkt aneinander zu hängen, kann man auch mit Klammern einen entsprechenden Vorrang schaffen: Kuss|(K|k)üssen.

Ein weiteres wichtiges Hilfsmittel bei regulären Ausdrücken ist der sogenannte Kleene-Stern (»*«). Wird der Kleene-Stern hinter ein Zeichen gesetzt, so bedeutet dies, dass dieses Zeichen keinmal bis beliebig oft wiederholt werden darf. Damit findet grep bei »of*e« nicht nur Kartoffeln und Kaminofen, sondern auch Soest, da hier das e direkt auf das o folgt.

Sehr ähnlich verhalten sich die Zeichen ? und +. Beim Fragezeichen soll das Zeichen ein bis keinmal vorkommen. Ein mindestens einmaliges Vorkommen wird durch das + erreicht. Alle drei Zeichen können natürlich auch an eine Klammer gesetzt werden, so dass sie sich auf den Inhalt der Klammer beziehen.

Auch wenn bei Verwendung der regulären Ausdrücke bei der Suche Wörter gefunden werden, bei denen nur ein Teil mit dem Ausdruck übereinstimmt, so sind diese nicht Teil der Sprache, die durch den regulären Ausdruck beschrieben werden. Bei den Wörtern der Sprache muss der reguläre Ausdruck genau stimmen. So enthält die Sprache, die von »of*e« beschrieben wird nur die Wörter oe, ofe, offe, offfe usw..



Aufgabe 7.1 L

a) Geben Sie alle Wörter der Sprache an, die durch den Ausdruck »(1|2)(34)(5|6)« gebildet wird.

b) Suchen Sie alle Zeilen in der Datei, in denen Julia oder Romeo vorkommt. Ihre Anzahl können sie über den Parameter »-c« für count bekommen.

c) Geben Sie alle Worte von »(ab|cab)*« an, die eine maximale Länge von 6 haben.

Es gibt noch eine weitere Menge von Möglichkeiten bei regulären Ausdrücken, von denen hier nur ein paar aufgelistet werden sollen:

- Der Punkt als Sonderzeichen steht für ein beliebiges Zeichen.
- Wenn ein Zeichen aus einer bestimmten Auswahl stammen soll, so kann man den Ausdruck mit eckigen Klammern abkürzen. Aus »0|1|2|3« wird einfach »[0123]«. Hängen die Zeichen zusammen kann man auch ein - verwenden: »[0-3]«.
- Kommt direkt nach einer eckigen Klammer ein ^, so wird der Inhalt negiert. Der Ausdruck »[^abc]« bedeutet also ein beliebiges Zeichen außer ein a,b oder c.
- Mit einem ^wird sonst auch der Anfang eines Wortes angegeben und mit $ dessen Ende. So müssen Wörter für »^a.*z$« die Wörter mit einem a beginnen und einem z enden. Dazwischen dürfen beliebige Zeichen mit in einer unbeschränkten Anzahl sein.

Kapitel 8.

Nicht endliche Automaten

Durch ihre Endlichkeit bei den Zuständen haben die Automaten eine Einschränkung, so dass bestimmte Problemfälle nicht mit ihnen bearbeitet werden können. Das prominenteste unter ihnen ist das Problem, nur Wörter zu akzeptieren, bei denen die Anzahl der sich öffnenden Klammern mit der sich schließenden Klammern übereinstimmt. Theoretischen Automaten benötigen dafür eigentlich eine nicht beschränkte Anzahl von Zuständen, sie müssen also ein unendlicher Automat sein. Um diese Beschränkung zu umgehen gibt es zwei Möglichkeiten: Der Kellerautomaten und die Turingmaschine, die beide in den nächsten beiden Abschnitten erklärt werden.

8.1. Kellerautomat

Die Bezeichnung Kellerautomat hat seine Ursache in dem Keller, den dieser Automatentyp zusätzlich gegenüber den bisher bekannten Automaten aufweist. Die Bezeichnung Keller wird im englischen Sprachraum mit Stack übersetzt. Da etliche ausgezeichnete Informatiklehrwerke und Umsetzungen in Programmiersprachen englischsprachige Bezeichner enthalten, wird der Begriff Stack auch im Deutschen verwendet. Eine weitere, allgemeinverständliche, deutsche Bezeichnung ist Stapel. In einen Keller/Stack/Stapel kann ein Automat theoretisch unendlich viele Zusatzeingaben ablegen. Um einen Stack nutzen zu können, ist der Zustandsübergang bei diesem Automaten nicht nur von der Eingabe und dem bisherigen Zustand abhängig, sondern auch vom aktuell obersten Element auf dem Stack. Dieses oberste Element wird vor jedem Übergang vom Stack herunter genommen. Es erfolgt neben dem Übergang in einen neuen Zustand auch das Ablegen von einem oder mehreren Elementen auf den Stack.

Zur Realisierung eines Akzeptors ist, gegenüber der Definition eines bisher bekannten endlichen Automaten, es für einen Kellerautomaten auch nötig das Alphabet der Zustände Γ sowie ein Zeichen # für die Startbelegung des Stacks mit anzugeben. Wenn dieses das oberste Element des Stacks ist, so ist dieser als leer anzusehen. Auch beim Zustandsübergangsdiagramm gibt es Veränderungen: So wird hier bei einem Übergang neben der Eingabe auch das zugehörige oberste Stackelement und das oder die Elemente,

die wieder auf den Stack gelegt werden soll mit angegeben. Die Schreibweise ist dabei <Eingabe, oberstes Element; neue Elemente>.

Um den Aufbau noch einmal zu verdeutlichen, wird die Definition eines Kellerautomat genauer betrachtet, der die Sprache $abc^n d^n b$ akzeptiert. Das Alphabet des Kellers besteht dabei aus zwei Zeichen, wobei 0 als Startwert # für den Keller gewählt wurde. Immer dann wenn ein c an der passenden Stelle in der Eingabe kommt, wird auch ein c zusätzlich zu dem bisher oben aufliegenden Zeichen auf den Stack gelegt. Jedes d entfernt dann das oberste c, indem die leere Eingabe ϵ aufgelegt wird.

$$\Sigma = \{a, b, c, d\}$$
$$Z = \{s_0, s_1, s_2, s_3, s_4, s_5\}$$
$$z_0 = s_0$$
$$E = \{s_5\}$$
$$\Gamma = \{c, 0\}$$
$$\# = 0$$
$$\varphi =$$

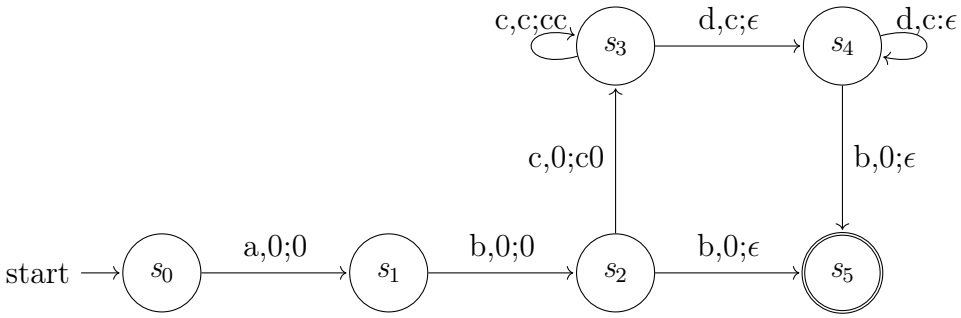


Abbildung 8.1.: Kellerautomat der $abc^n d^n b$ akzeptiert.

Möchte man mit einem solchen Kellerautomaten die Sprache $a^n b^n$ akzeptieren, so muss sichergestellt sein, dass am Ende des Wortes nicht nur ein akzeptierter Zustand erreicht wurde, sondern auch der Keller leer ist. Bei dem vorherigen Beispiel trat dieses Problem nicht auf, da am Ende ein einzelnes b stehen musste. Mit einer simplen Ergänzung lässt sich die benötigte Bedingung realisieren: Der letzte Übergang zu einem akzeptierenden Zustand bezieht sich nur auf das oberste Element des Stacks und als Eingabe wird ϵ angegeben, da am Ende des eingelesenen Wortes keine weitere Eingabe mehr erfolgt.

Aufgabe 8.1

H L

Erstellen Sie einen Kellerautomaten, der das Klammerproblem löst, also alle Eingaben akzeptiert bei denen es zu jeder öffnenden Klammer auch eine schließende Klammer gibt. Verwenden sie zur Vereinfachung neben den Klammern für alle anderen Symbole in der Eingabe ein Asterisk.

8.2. Turingmaschine

Die Turingmaschine ist ein rein theoretischer Automat, den Alan M. Turing 1936 beschrieben hat. Mit ihm lässt sich der Begriff der Berechenbarkeit in der Mathematik fassen. Die Turingmaschine besteht aus einem Schreib- und Lesekopf, der wie ein Automat verschiedene Zustände hat. Der Schreib- und Lesekopf kann sich an einem unendlich langem Band entweder vor oder zurück bewegen.

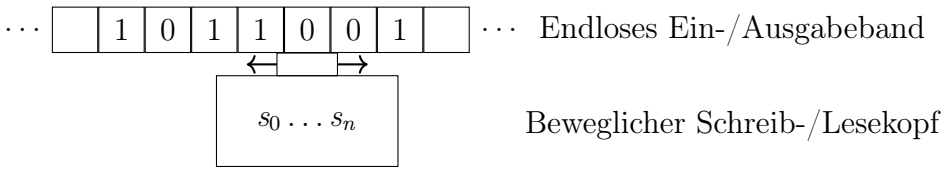


Abbildung 8.2.: Prinzip der Turingmaschine

Je nach Zustand des Schreib-/Lesekopfs und dem Inhalt der aktuellen Position am Band ändert sich der Zustand des Kopfs. Außerdem wird davon abhängig gemacht, ob der Kopf sich in eine Richtung bewegt oder etwas an der aktuellen Position auf das Band geschrieben wird. Wenn man das Band in die beiden Richtungen jeweils als Keller auffasst, so lässt sich die Turingmaschine auch als Kellerautomat mit zwei Kellern beschreiben.

Die Zustandsübergänge der Turingmaschine sollen so gewählt werden, dass man in Abhängigkeit vom Inhalt des Bandes Aktionen auf andere Inhalte des Bandes anwenden kann. Es lässt sich also ein Programm auf das Band schreiben, dass Daten an anderen Stellen des Bandes modifiziert. Damit kann die Turingmaschine alles, was ein moderner Rechner auch kann. Man spricht davon, dass sie gleich mächtig sind.

Bei der Analyse der Berechenbarkeit eines Problems geht es bei der Turingmaschine darum, dass diese nach Ausführung des Programms zu Berechnung des Problems am Ende in einem Stoppzustand ankommt. Erreicht die Turingmaschine diesen Zustand, so ist das

Problem berechenbar. Kann sie diesen Zustand nicht erreichen, so ist das Problem nicht berechenbar. Durch die gleiche Mächtigkeit bedeutet dieses, dass ein solches Programm dann auch auf einem normalen Rechner nicht berechenbar ist. Dieses zeigt die aktuelle Bedeutung der Turingmaschine, da sie durch ihren einfachen Aufbau es einfacher beweisbar macht, dass ein Problem berechenbar ist oder nicht.

Kapitel 9.

Sammlung von Aufgaben

Zur Übung und Vertiefung sind hier verschiedene Aufgaben angegeben.



Aufgabe 9.1 L

Geben Sie an, welche fünf Elemente zur vollständigen Beschreibung eins endlichen Automaten gehören. Gehen Sie dabei auf die Unterschiede zwischen einem Transduktor und einem Akzeptor ein.



Aufgabe 9.2

Betrachten Sie den Akzeptor für Klassenbezeichner aus Abbildung 3.1 und verändern ihn so, dass der Klassenname mindestens vier Zeichen lang ist. Sie müssen nur das Zustandsübergangsdiagramm angeben.



Aufgabe 9.3

Erstellen Sie einen Automaten, an dem für 1,50 Euro in 1 Euro oder 50 Cent Stücken, ein spezielles Buch ausgeliehen werden kann. Die Besonderheit dieses Automaten soll darin liegen, dass er nur maximal drei dieser Bücher beherbergen kann. Entsprechend soll er auf Eingaben reagieren, wenn er aktuelle keine Bücher mehr hat bzw. zu viele Bücher versucht werden zurückzugeben.



Aufgabe 9.4

H

Bei einer Verkehrsampel dürfen die Lichter nur in bestimmten Kombinationen leuchten. Erstellen Sie einen Akzeptor, dem die Schaltzustände der drei Lampen in der Reihenfolge rot, gelb und grün als ein oder aus übergeben werden. Er soll nur die gültigen Farbkombinationen akzeptieren.

Anhang A.

Hinweise

Hinweis zu Aufgabe 2.2.1 ←

a) Jede Eingabe löst im Automaten eine Aktion aus. Also schreiben Sie alle diese Möglichkeiten auf, wie z. B. den Einwurf einer 1 Euro Münze.

b) Denken Sie dabei neben der Anzeige auch an die Schranke selber und die Geldrückgabe.

Hinweis zu Aufgabe 2.2.2 ←

An dieser Stelle soll der vorstellte Automat so erweitert werden, dass er auch die Geldmünzen verarbeiten kann.

Hinweis zu Aufgabe 2.2.3 ←

a) Wenn auf eine Eingabe keine Ausgabe kommt, so kann man dieses auch mit einem waagerechten Strich deutlich machen.

b) Der Automat enthält bereits an die Information, welcher Betrag gezahlt wurde.

Hinweis zu Aufgabe 3.3.1 ←

b) Der Akzeptor überprüft zwei Schreibweisen eines besonderen Zahlensystems.

c) Stellen Sie zuerst einfache Regeln auf, anhand derer überprüft werden kann, ob es ein gültiger Name ist. Nehmen sie dabei gültige und nicht gültige Beispiele. So sollten »Hermman Hesse«, »Ludwig van Beethoven« und »Luisa Maria Peters« akzeptiert werden. Diese Fälle aber nicht: »Gustav«, »MArtin Mars« und »von Beethoven, Ludwig«.

Hinweis zu Aufgabe 3.3.2 ←

a) Überlegen Sie genau, was bei einer schließenden Klammer zu beachten ist. Nehmen Sie zum Testen auch Beispiele wie (x(x)x((x)x))(x).

c) Stellen Sie verschiedene Aspekte bei einem Computer gegenüber. Betrachten Sie dabei auch den Compiler für eine beliebige Programmiersprache, der auf einem Computer ausgeführt werden kann. Vor der eigentlichen Übersetzung in ein maschinenlesbares Programm, wird in der Regel die Syntax des Quellcodes überprüft.

Hinweis zu Aufgabe 3.3.4 ←

a) Das Eingabealphabet beschränkt sich bei der DNA nur auf vier verschiedene Möglichkeiten: ACGT. Zu beachten ist nur der Anfang und das Ende der Eingabe. Dabei muss der entsprechende Einstieg gefunden werden.

Hinweis zu Aufgabe 3.3.5 ←

a) Versuchen Sie alle Schritte nach dem ϵ-Übergang nachzuvollziehen und diese entsprechend umzusetzen.

b) Die Wörter setzen sich aus Teilwörtern zusammen.

c) Die Übergänge müssen durch die möglichen nächsten Schritte ersetzt werden.

Hinweis zu Aufgabe 5.5.1 ←

a) Das Adjektiv terminal stammt vom lateinischen terminare: abgrenzen, begrenzen, beenden.

b) Es ist hilfreich die Produktionsregeln einfach anzuwenden und bei den Auswahlmöglichkeiten beliebig zu wählen.

Hinweis zu Aufgabe 5.5.2 ←

a) Bevor Sie ihre Lösung kontrollieren, sollten Sie die Produktionsregeln mit den drei jeweils gültigen Mustern vergleichen.

b) Bei allen gegebenen ist dieses möglich, was aber nicht immer der Fall sein muss. Wenn es möglich ist, müssen die Regeln, die zu einem Konflikt führen, durch passende ersetzt werden.

c) Es gibt zwei Möglichkeiten dazu: Man fängt beim Automaten hinten an und geht die Pfeile rückwärts oder man erstellt eine rechtsreguläre Grammatik und versucht sie umzustellen. Dabei kann man, bis auf das Startsymbol sagen: Ein Zustand ein Nichtterminalsymbol.

Hinweis zu Aufgabe 8.8.1

Der letzte Abschnitt vor der Aufgabe sollte beachtet werden und die Anzahl der Zustände ist sehr gering.

Hinweis zu Aufgabe 9.9.4

Es gibt nur vier verschiedene Farbkombination, die akzeptiert werden. Schreiben Sie sich diese explizit auf um dafür den Akzeptor zu erstellen.

Anhang B.

Lösungen

Lösung 2.2

Die Lösung wird im Text darunter vorgestellt.

Lösung 2.3

a) Erfolgt keine Ausgabe, so wird dieses mit einem waagerechten Strich deutlich gemacht. Die Semikolons in den Mengen wurden verwendet, da Kommata bereits in den einzelnen Elementen Verwendung finden.

$$\Sigma = \{0{,}5;\ 1;\ 2\}$$
$$A = \{\text{-};\ \text{T};\ \text{T_0,5};\ \text{T_1};\ \text{T_1,5};\}$$
$$Z = \{s_0, s_{0,5}, s_1, s_{1,5}, s_2\}$$
$$z_0 = s_0$$
$$\varphi =$$

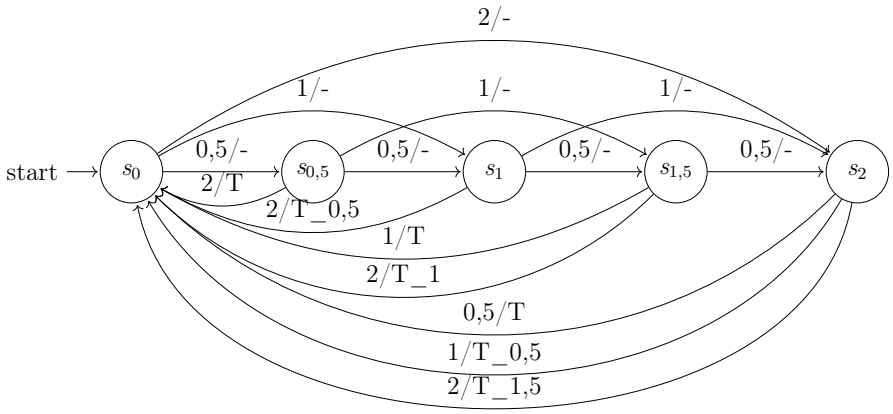


Abbildung B.1.: Lösung für den Ticketautomat

b) In den Zuständen ist gespeichert, welcher Betrag bereits gezahlt wurde. Daher könnte man eine Anzeige direkt mit den Zuständen verknüpfen.

Lösung 2.4 ←

Betrachtet wurden alle Fälle nicht, bei denen erst Geld und anschließend die Parkmünze eingeworfen wurde. Auch die Fälle, in denen die Schranke geöffnet ist und Geld bzw. die Parkmünze eingeworfen wird, sind nicht eingebaut. Für diese Fälle ist es eher sinnvoll, die Zustandsübergangsfunktion zu erweitern.

Lösung 3.1 ←

a) Die Eingabe muss mit einem Großbuchstaben beginnen und darf anschließend nur weitere Buchstaben und Ziffern, aber keine Sonderzeichen enthalten. Die Groß- und Kleinschreibung ist bei den weiteren Zeichen nicht von Bedeutung.

b) Die Eingabemenge ist $\Sigma = \{\text{A}, \ldots, \text{F}, 0, \ldots, 9, \text{x}, \text{h}\}$ und die akzeptierten Endzustände sind $E = \{s_3, s_5\}$. Der Automat akzeptiert die beiden Schreibweisen für hexadezimale Zahlen, die entweder mit einem 0x beginnen oder mit einem h enden. Dabei sind keine führenden Nullen zugelassen. Die Null selber aber schon.

c) Der Akzeptor muss nur prüfen, ob mindestens zweimal mehrere Zeichen hintereinander hängen und durch ein Leerzeichen getrennt sind. Dabei darf nur der erste Buchstabe jeweils groß sein. In der Mitte dürfen auch noch weitere Zeichen aneinander hängen, die komplett klein geschrieben sind.

Für die bessere Lesbarkeit ist das Leerzeichen durch ein _ im folgenden Automaten ersetzt worden.

$$\Sigma = \{_, \text{A}, \ldots, \text{Z}, \text{a}, \ldots, \text{z}\}$$
$$Z = \{s_1, s_2, s_3, s_4\}$$
$$z_0 = s_0$$
$$E = \{s_3\}$$
$$\varphi =$$

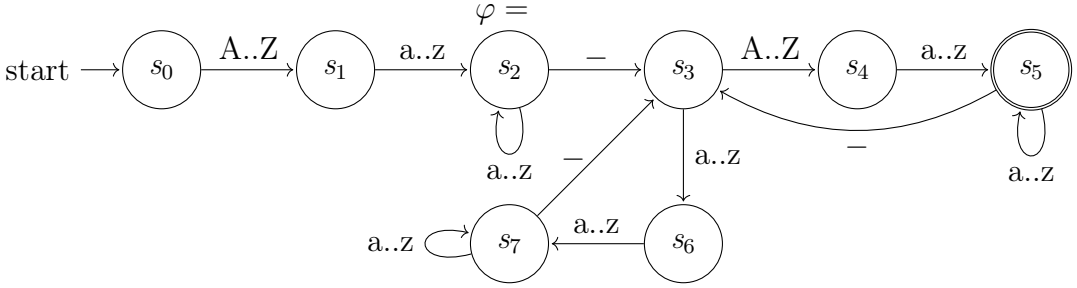


Abbildung B.2.: Lösung für den Namensakzeptor

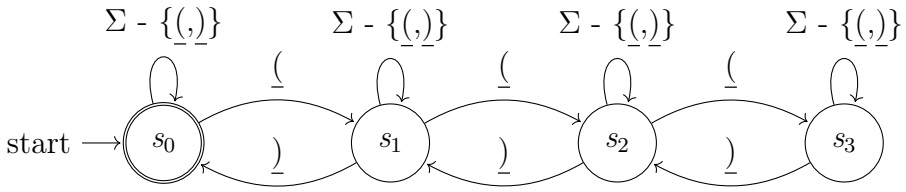


Abbildung B.3.: Akzeptor für eine korrekte Klammerung bis zur Tiefe von drei.

Lösung 3.2 ←

a) Bei der Lösung sind die Klammern unterstrichen, um deutlich zu machen, dass es sich hierbei um Eingabesymbole handelt.

b) Die Lösung für den Automaten sieht dem Automaten für eine fest Klammerungstiefe sehr ähnlich: Der wichtige Unterschied zwischen den beiden Automaten besteht

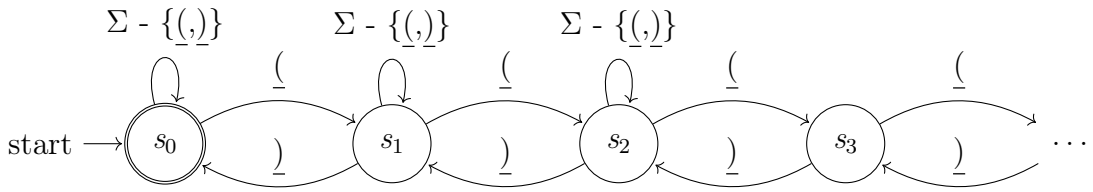


Abbildung B.4.: Zustandsübergangsdiagramm für einen unendlichen Automaten, der die korrekte Klammerung überprüft

in diesem, dass bei ihm nicht vorher angegeben werden kann, wie viele Zustände benötigt werden, da für jede Stufe der Klammerung ein weiterer Zustand benötigt wird. Es handelt sich deshalb um keinen endlichen Automaten.

c) Vom Prinzip her handelt es sich bei einem Computer um einen unendlichen Automaten. Dieses lässt sich auch daran festmachen, dass bei Programmen die korrekte Klammerung unabhängig von der Klammerungstiefe geprüft werden kann. Auch andere Zählungen sind ohne größere Probleme möglich. Beachtet man aber, dass der Zustand eines Automaten eine mögliche Belegung des Speichers darstellt, ist der Computer kein unendlicher Automat. Dieses liegt daran, dass bei jedem Computer der Speicher beschränkt ist und damit auch die Möglichkeiten, wie der Speicher genutzt wird.

Lösung 3.3 ←

Wie den folgenden Lösungen zu entnehmen ist, werden nicht alle Fälle korrekt erkannt:

Peter: $0 \overset{\text{P}}{\Longrightarrow} 0 \overset{\text{e}}{\Longrightarrow} 0 \overset{\text{t}}{\Longrightarrow} 1 \overset{\text{e}}{\Longrightarrow} 2 \overset{\text{r}}{\Longrightarrow} 3$
Peterson: $0 \overset{\text{P}}{\Longrightarrow} 0 \overset{\text{e}}{\Longrightarrow} 0 \overset{\text{t}}{\Longrightarrow} 1 \overset{\text{e}}{\Longrightarrow} 2 \overset{\text{r}}{\Longrightarrow} 3 \overset{\text{s}}{\Longrightarrow} 0 \overset{\text{o}}{\Longrightarrow} 0 \overset{\text{n}}{\Longrightarrow} 0$
Wetter: $0 \overset{\text{W}}{\Longrightarrow} 0 \overset{\text{e}}{\Longrightarrow} 0 \overset{\text{t}}{\Longrightarrow} 1 \overset{\text{t}}{\Longrightarrow} 0 \overset{\text{e}}{\Longrightarrow} 0 \overset{\text{r}}{\Longrightarrow} 0$
lauter: $0 \overset{\text{l}}{\Longrightarrow} 0 \overset{\text{a}}{\Longrightarrow} 0 \overset{\text{u}}{\Longrightarrow} 0 \overset{\text{t}}{\Longrightarrow} 1 \overset{\text{e}}{\Longrightarrow} 2 \overset{\text{r}}{\Longrightarrow} 3$

Lösung 3.4

a) Bei dem Automaten müssen die Start- und Endsequenz direkt hintereinander mit einzelnen Übergängen aufgebaut sein. In der Mitte sind alle vier Möglichkeiten als Übergang erlaubt.

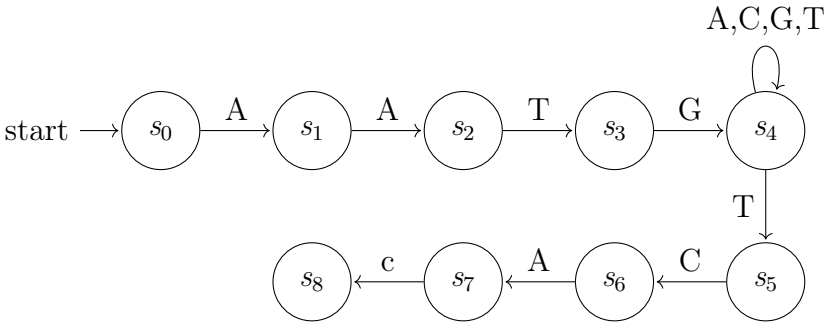


Abbildung B.5.: Zustandsübergangsdiagramm für einen NEA bei dem nur die Start- und Endsequenz überprüft werden muss

b) Der Automat akzeptiert Eingaben, die mit eine Wiederholung von AT beginnen und einer Wiederholung von CG enden. Dazwischen muss mindestens ein Zeichen aus A, T, C oder G stehen.

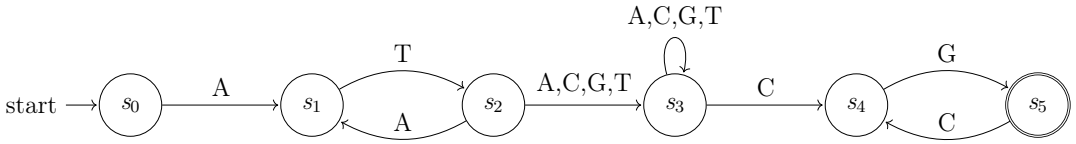


Abbildung B.6.: Zustandsübergangsdiagramm für den Automaten, dessen Zustandsübergang in der Aufgabe als Tabelle dargestellt wurde

Lösung 3.5

a) Es muss der ϵ-Übergang nur durch ein t ersetzt werden:

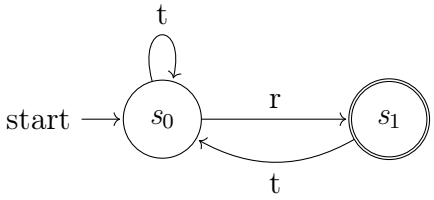


Abbildung B.7.: Übergangsdiagramm ohne ϵ-Übergang

b) Wörter die sich aus dem Vielfachen von aab und aba zusammensetzten.

c) Hier müssen die Übergänge durch a ersetzt werden. Dabei bekommen sie aber auch ein anderes Ziel.

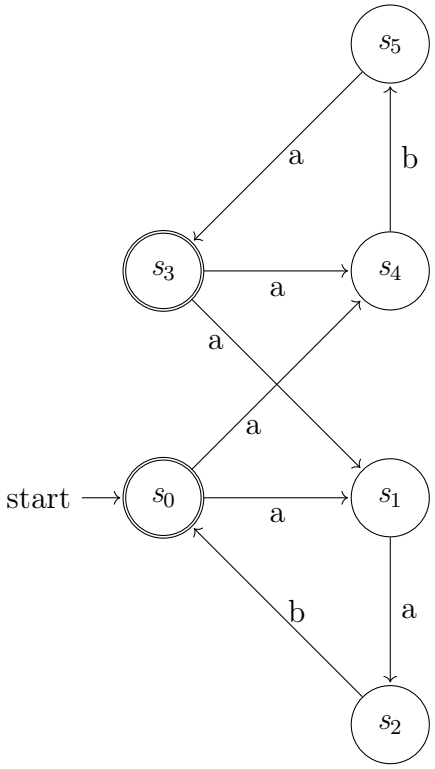
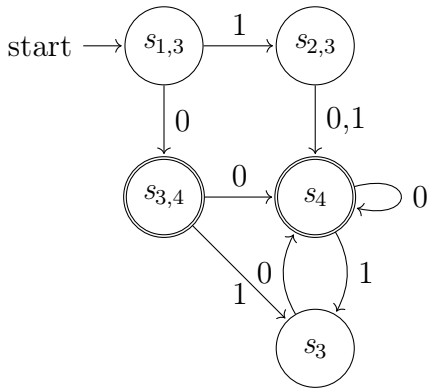


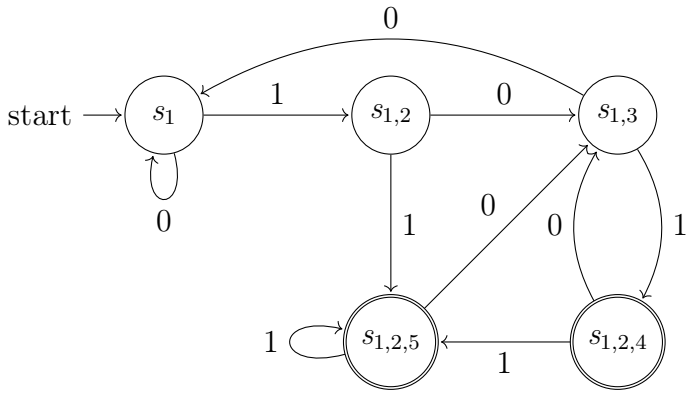
Abbildung B.8.: Übergangsdiagramm ohne ϵ-Übergang

Lösung 3.6

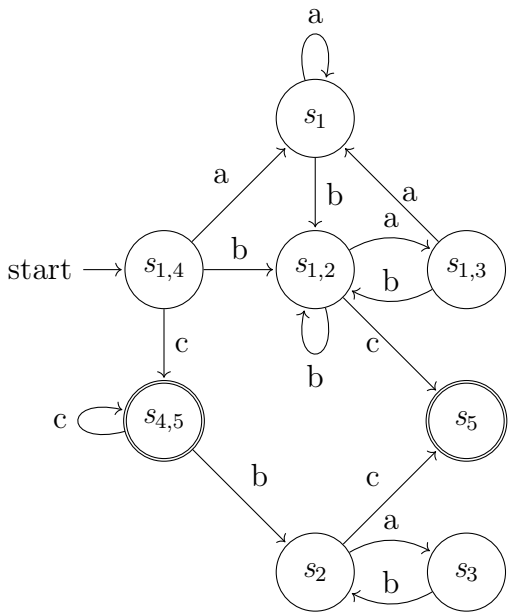
1.



2.



3.



Lösung 5.1 ←

b) Alle Wörter enden auf aa, bba oder bca. Davor können aa, bba und bca in beliebiger Reihenfolge und Menge gesetzt werden. In der Regel ergibt jedes Nichtterminalsymbol einen Zustand, wie in Abbildung B.9 dargestellt.

c) Es gibt, ähnlich wie bei Automaten, keine eindeutige Lösung. Eine mögliche Lösung ist:

$N = \{\text{A, B, C, D, E}\}$
$\Sigma = \{\text{a, b, c}\}$
$P = \{\text{A}\rightarrow\text{aB}|\text{bC}|\epsilon, \text{B}\rightarrow\text{acbC}|\text{aaD}|\text{ac}|\epsilon, \text{C}\rightarrow\text{bcaB}|\text{bbE}|\text{bc}|\epsilon, \text{D} \rightarrow\text{aD}|\text{bC}|\epsilon, \text{E}\rightarrow\text{bE}|\text{aB}|\epsilon\}$
$S = \text{A}$

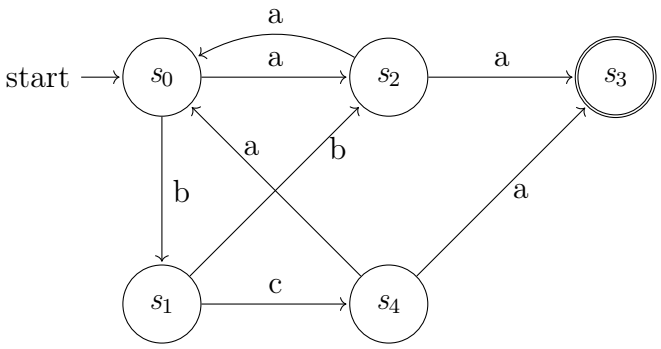


Abbildung B.9.: Zustandsübergangsdiagramm für einen Automaten, der gleiche Wörter akzeptiert, wie die gegebene Grammatik

Lösung 5.2 ←

a)
1. Sie ist nicht regulär, da D→Daa gegen die Regeln verstößt.
2. Sie ist rechtsregulär.
3. Sie ist nicht regulär, da A→aBa gegen die Regeln verstößt.

b)
1. Die Regel D→Daa|Eb muss überführt werden in die beiden Regeln D→Fa|Eb und F→Da.
2. Sie ist bereits regulär.
3. Die Regel A→aBa muss überführt werden in die Regel A→aB und die Regel E→b|bE in die Regel E→bF|bE und F →a.

c) $N = \{\text{A, B, C, D, E}\}$
$\Sigma = \{1, 0\}$
$P = \{\text{A}\rightarrow 0\text{B}|0\text{C}|0\text{E}, \text{B}\rightarrow\epsilon|1\text{C}, \text{C}\rightarrow 0\text{B}, \text{D}\rightarrow 0\text{C}|0\text{E}, \text{E}\rightarrow 1\text{D}\}$
$S = \text{A}$

Lösung 6.1 ←

a) Bei den Sprachen spielt die Abzählung die entscheidende Rolle:
1. Ist regulär.
2. Ist nicht regulär, da zu jedem geöffneten Tag auch ein schließender gehört.
3. Ist nicht regulär, da hier auch auf öffnende und schließende Klammern geachtet werden muss.
4. Ist regulär.
5. Ist regulär.

b) $L = \{\text{a}, \text{abd}, \text{bc}^n\text{bb} | n \geq 2\}$

Lösung 7.1

a) L = {1345, 1346, 2345, 2346}

b) Es sind genau 156 Zeilen

c) ϵ, ab, abab, ababab, cab, cabcab, abcab, cabab

Lösung 8.1

Der zweite Zustand bei der Lösung wird nur gebraucht, um am Ende den Stack darauf zu überprüfen, dass dieser leer ist.

$\Sigma = \{a, \ldots, z, A, \ldots, Z, (,)\}$
$Z = \{s_0, s_1\}$
$z_0 = s_0$
$E = \{s_1\}$
$\Gamma = \{(), 0\}$
$\# = 0$
$\varphi =$

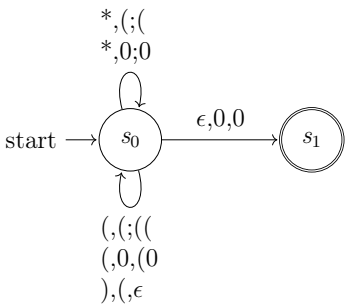


Abbildung B.10.: Lösung für das Klammerproblem mit einem Kellerautomaten

Lösung 9.1

Für einen Transduktor benötigt man die Menge der Zustände Z, die Eingabe Σ, die Ausgabe A, den Startzustand z_0 und die Zustandsübergangsfunktion φ. Letztere kann entweder durch eine Tabelle oder ein Diagramm erfolgen. Beim Akzeptor wird anstelle der Ausgabemenge die Menge der akzeptierten Endzustände E als Angabe benötigt.

Anhang C.

Umsetzungen in Programmiersprachen

C.1. Java

Die Umsetzung des Parsers in der Programmiersprache Java:

```
public boolean parser(String wort) {
    int zustand = 1;
    int pos = 0;
    while (pos < wort.length()) {
        char symbol = wort.charAt(pos);
        if (zustand == 1) {
            if (symbol == 'a' || symbol == 'c') {
                zustand = 3;
            } else if (symbol == 'b') {
                zustand = 2;
            } else {
                return false;
            }
        } else if (zustand == 2) {
            if (symbol == 'a' || symbol == 'b') {
                zustand = 3;
            } else {
                return false;
            }
        } else { //zustand == 3
            if (symbol == 'b') {
                zustand = 3;
            } else if (symbol == 'a') {
                zustand = 2;
            } else {
                return false;
```

```
            }
        }
        pos ++;
    }
    return (zustand == 3);
}
```

C.2. Python

Die Umsetzung des Parsers in der Programmiersprache Python:

```
def parser(wort):
    zustand = 1
    pos = 0
    while pos < len(wort):
        symbol = wort[pos]
        if zustand == 1:
            if symbol == 'a' or symbol == 'c':
                zustand = 3
            elif symbol == 'b':
                zustand = 2
            else:
                return False
        elif zustand == 2:
            if symbol == 'a' or symbol == 'b':
                zustand = 3
            else:
                return False
        else:  #zustand == 3
            if symbol == 'b':
                zustand = 3
            elif symbol == 'a':
                zustand = 2
            else:
                return False
        pos += 1
    return zustand == 3
```

Literatur

Clausing, Achim (Jan. 2003). *Informatik III – Grundlagen der Theoretischen Informatik – Wintersemester 2002/2003*. Münster.

Free ebooks - Project Gutenberg (Nov. 2014). Redwood City : Oracle. URL: https://www.gutenberg.org (besucht am 12. 11. 2014).

MSW-NW (2023). *Abitur Gymnasiale Oberstufe – Informatik – Übersichtsseite: Vorgaben, Fachliche Hinweise und sonstige Materialien, Operatoren und Konstruktionsvorgaben, Aufgabenbeispiele. Zentralabitur – **Rahmenvorgaben** für 2023, 2024 und 2025*. MSW-NW – Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen. URL: https://t1p.de/9ygk (besucht am 01. 07. 2023).

Rechenberg, Peter (2000). *Was ist Informatik? Eine allgemeinverständliche Einführung*. 3., überarbeitete und erweiterte Auflage. München: Carl Hanser Verlag. ISBN: 3-446-21319-8.

Tantau, Till (Jan. 2013). *Einführung in die Informatik 1 und 2 – Wintersemester 2012, Sommersemester 2013*. Lübeck.