Formale Sprachen und Automatentheorie

Horst Gierhardt
Städtisches Gymnasium Bad Laasphe
horst@gierhardt.de
http://www.oberstufeninformatik.de/theorie

Version vom 10. Februar 2015

Inhaltsverzeichnis

1 Natürliche und formale Sprachen

1.1 Loriot

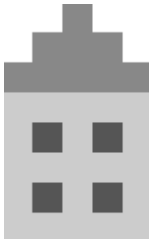
Das Frühstücksei

Er: Berta!
Sie: Ja...
Er: Das Ei ist hart!
Sie: (schweigt)
Er: Das Ei ist hart!!!
Sie: Ich habe es gehört...
Er: Wie lange hat das Ei denn gekocht?
Sie: Zu viele Eier sind gar nicht gesund!
Er: Ich meine, wie lange dieses Ei gekocht hat ..?
Sie: Du willst es doch immer viereinhalb Minuten haben...
Er: Das weiß ich...
Sie: Was fragst du denn dann?
Er: Weil dieses Ei nicht viereinhalb Minuten gekocht haben kann!
Sie: Ich koche es aber jeden Morgen viereinhalb Minuten.
Er: Wieso ist es dann mal zu hart und mal zu weich?
Sie: Ich weiß es nicht ...ich bin kein Huhn!
Er: Ach! ... Und woher weißt du, wann das Ei gut ist?
Sie: Ich nehme es nach viereinhalb Minuten heraus, mein Gott!
Er: Nach der Uhr oder wie?
Sie: Nach Gefühl ... eine Hausfrau hat das im Gefühl...
Er: Im Gefühl? Was hast du im Gefühl?
Sie: Ich habe es im Gefühl, wann das Ei weich ist...
Er: Aber es ist hart ... vielleicht stimmt da mit deinem Gefühl was nicht...
Sie: Mit meinem Gefühl stimmt was nicht? Ich stehe den ganzen Tag in der Küche, mache die Wäsche, bring deine Sachen in Ordnung, mache die Wohnung gemütlich, ärgere mich mit den Kindern rum und du sagst, mit meinem Gefühl stimmt was nicht?
Er: Jaja ... jaja ... jaja ...wenn ein Ei nach Gefühl kocht, kocht es eben nur zufällig genau viereinhalb Minuten.
Sie: Es kann dir doch ganz egal sein, ob das Ei zufällig viereinhalb Minuten kocht ... Hauptsache, es kocht viereinhalb Minuten!
Er: Ich hätte nur gern ein weiches Ei und nicht ein zufällig weiches Ei! Es ist mir egal, wie lange es kocht!
Sie: Aha! Das ist dir egal ...es ist dir also egal, ob ich viereinhalb Minuten in der Küche schufte!
Er: Nein – nein
Sie: Aber es ist nicht egal ... das Ei muss nämlich viereinhalb Minuten kochen...
Er: Das habe ich doch gesagt...
Sie: Aber eben hast du doch gesagt, es ist dir egal!
Er: Ich hätte nur gern ein weiches Ei...
Sie: Gott, was sind Männer primitiv!
Er: (düster vor sich hin) Ich bringe sie um ...morgen bringe ich sie um!

1.2 PGM

Die Sprache PGM (Portable Graymap) wird zur Darstellung von Bildern benutzt. Hier ein Beispiel:

```
P2
# beispiel.pgm
5 8
15
15 15  8 15 15
15  8  8  8 15
 8  8  8  8  8
12 12 12 12 12
12  5 12  5 12
12 12 12 12 12
12  5 12  5 12
12 12 12 12 12
```



Aufgabe: Schreibe den Text mit einem Texteditor und speichere ihn als beispiel.pgm ab. Dann sieh Dir das Ergebnis z. B. mit dem Programm *IrfanView* an. Dieses Programm „glättet“ die Pixel automatisch. Um dies zu verhindern, muss nach dem Vergößern die Fenstergröße einmal verändert werden[1].

Einige Regeln der Sprache:

- Die erste Zeile (P2) kennzeichnet das PGM-Format in der ASCII-Version.
- Die zweite Zeile enthält einen Kommentar, den man auch weglassen kann.
- Die beiden Zahlen „5 8“ in der dritten Zeile beschreiben die Aufteilung der Pixel in Spalten und Zeilen (hier 5 Spalten und 8 Zeilen).
- Die Zahl 15 in der vierten Zeile beschreibt die Anzahl der Graustufen - hier von Stufe 0 bis Stufe 15. Beachte, dass maximal 256 Graustufen festgelegt werden können.
- In den folgenden Zeilen werden die einzelnen Grauwerte mit Hilfe von Zahlen dargestellt. Diese sind hier - der besseren Lesbarkeit wegen - bereits im Quelltext genauso angeordnet wie in der beabsichtigten Bilddarstellung. Eine solche Anordnung ist aber nicht erforderlich. Man könnte z. B. alle Zahlen in eine einzige Zeile schreiben oder auch jede dieser Zahlen in eine neue Zeile.

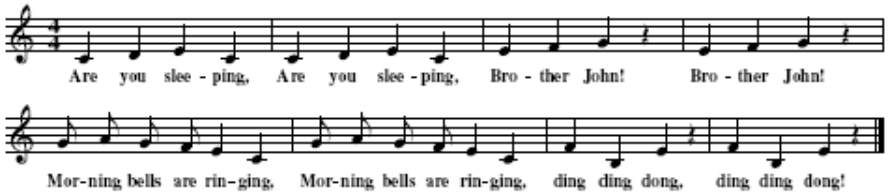
1.3 ABC

Der folgende Text ist in der Sprache „ABC“ verfasst.

```
X: 1
T: Brother John
C: Traditional
L: 1/4
```

[1] „Is this a bug or is this a feature?“

```
K: C
CDEC|CDEC|EFGz|\ % continues
w: Are you slee-ping, Are you slee-ping, Bro-ther John!\ % continues
EFGz|
w: Bro-ther John!
G/ A/ G/ F/ EC|G/ A/ G/ F/ EC|\ % continues
w: Mor-ning bells are rin-ging, Mor-ning bells are rin-ging,\
FB,Ez|FB,Ez|]
w: ding ding dong, ding ding dong!
```



Aufgabe: Erkläre die Sprache ABC mit Blick auf den Notentext.
Weitere Informationen: The ABC Plus Project, http://abcplus.sourceforge.net

1.4 LaTeX

Diese Eingabe in der Sprache TeX bzw. genauer in der Sprache LaTeX

```
\textbf{Satz von Taylor:} Es sei $I$ ein beliebiges Intervall und
$x_0 \in I$ im Innern von $I$. $f$ sei beliebig oft differenzierbar
auf $I$. Dann gilt:
\[ f(x) = \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!} (x-x_0)^i
          + R_n (x, x_0)
\ \ \mbox{mit}\ \
R_n (x, x_0) = \frac{1}{n!}\int\limits_{x_0}^x (x-t)^{n}
                f^{(n+1)}(t)\; dt \]
```

erzeugt nach diversen Übersetzungen diese Ausgabe:

Satz von Taylor: Es sei I ein beliebiges Intervall und $x_0 \in I$ im Innern von I. f sei beliebig oft differenzierbar auf I. Dann gilt:

$$f(x) = \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i + R_n(x, x_0) \quad \text{mit} \quad R_n(x, x_0) = \frac{1}{n!}\int_{x_0}^{x} (x - t)^n f^{(n+1)}(t)\, dt$$

Dieses Dokument ist komplett in dieser Sprache geschrieben.

1.5 Ipogesien

In Ipogesien hört man im Mathematikunterricht der 1. Klasse ständig folgende Wörter:

```
ipigisi, isipigisisi, ipisigisisi, isisipigisisisi,
isipisigisisisi, ipisisigisisisi, ...
```

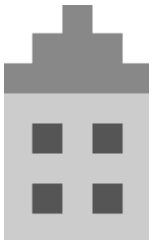
Nachdem der Schüler Ipo das Wort `isipisisigisisisisisisi` sagt, gibt es ein Aufmurren. Warum wohl?
Aufgabe:

1. Wie sind die Wörter der „ipigisi-Sprache“ aufgebaut?
2. Hast du eine Idee, was die Wörter der „ipigisi-Sprache“ bedeuten könnten? Tipp: Es handelt sich um Additionsaufgaben.

1.6 Sprachen in der Informatik

Informatik wird oft als Wissenschaft von der automatisierten Verarbeitung von Information bezeichnet. Information muss dabei zunächst in Form von Daten formal dargestellt werden, bevor sie verarbeitet werden kann. Daten können dann mit Hilfe von Computerprogrammen zu neuen Daten verarbeitet werden oder auch zu anderen Computern transportiert werden. Wenn die hierdurch erzeugten bzw. transportierten Daten gedeutet werden, entsteht neue Information. Sprachen spielen bei diesem Vorgehen eine zentrale Rolle. Dies soll am folgenden Beispiel verdeutlicht werden.

Die Bildinformation muss zunächst in einer bestimmten Form dargestellt werden. Im vorliegenden Beispiel wird hierzu die Sprache PGM (siehe oben) benutzt. Diese Sprache erlaubt es, Grauwerte von Pixeln mit Hilfe von Zahlen zu beschreiben.



$\Longrightarrow$

```
P2
5 8
15
15 15  8 15 15
15  8  8  8 15
 8  8  8  8  8
12 12 12 12 12
12  5 12  5 12
12 12 12 12 12
12  5 12  5 12
12 12 12 12 12
```

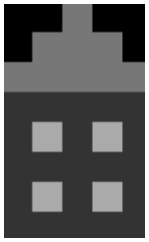
Die Bilddaten (in einer Datei) können jetzt mit einem Programm bearbeitet werden.

```
1     public void dateiInvertiertKopieren(String dateiname, String zieldateiname)
2    { String ersteZeile;
3      int spaltenanzahl, zeilenanzahl, graustufen, wert;
4      boolean okay;
5      In.open(dateiname);
6      if (In.done())
7           {  Out.open(zieldateiname);
8              ersteZeile = In.readLine();     Out.println(ersteZeile);
9              spaltenanzahl = In.readInt();   Out.print(spaltenanzahl + " ");
10             zeilenanzahl = In.readInt();    Out.println(zeilenanzahl);
```

```
11              graustufen = In.readInt();      Out.println(graustufen);
12              for (int zeile=1; zeile<= zeilenanzahl; zeile++)
13              {    for (int spalte=1; spalte <= spaltenanzahl; spalte++)
14                   {  wert = graustufen - In.readInt();
15                      Out.print(wert + " ");
16                   }
17                   Out.println();
18              }
19              Out.close();
20              In.close();
21          }
22      else {  Out.println("Die Datei " + dateiname + " existiert nicht."); }
23
24    } // dateiInvertiertKopieren
```

Die Java-Methode liest die Werte für die Graustufen aus einer Datei, invertiert sie durch eine Subtraktion und schreibt die Werte in eine neue Datei. Diese neuen Daten lassen sich jetzt wieder als Bild interpretieren.

```
P2
5 8
15
0 0 7 0 0
0 7 7 7 0
7 7 7 7 7
3 3 3 3 3
3 10 3 10 3
3 3 3 3 3
3 10 3 10 3
3 3 3 3 3
```

$\Longrightarrow$ 

Im Beispiel kommen also zwei Sprachen zum Einsatz. Die Sprache PGM wird zur Darstellung von Information benutzt, die Sprache Java zur Verarbeitung von Daten.

Es gibt eine Vielzahl von **Sprachen zur Darstellung von Information**, die eine Weiterverarbeitung mit dem Computer ermöglichen:

- HTML: Sprache zur Darstellung von Hypertexten
- SVG: Sprache zur Darstellung von Vektorgrafiken
- PBM, PGM, PPM: Sprachen zur Darstellung von Pixelgrafiken
- ABC: Sprache zur Darstellung von Musik

Es gibt ebenso eine Vielzahl von **Sprachen zur Beschreibung der Verarbeitung von Daten**. Zu diesen Sprachen zählen alle Programmiersprachen: Java, C, SQL, ...
Computersprachen haben viele Gemeinsamkeiten mit natürlichen Sprachen.

- Man nutzt sie, um bestimmte Sachverhalte zu beschreiben.
- Sie dienen der Kommunikation.
- Man muss bestimmte Regeln beachten, wenn man sie nutzt.
- Sprachelemente haben eine ganz bestimmte Bedeutung.

- ...

Es gibt aber auch wesentliche Unterschiede: Computersprachen müssen eindeutig und präzise sein. Bei der Verwendung von Computersprachen führen bereits kleinste Ungenauigkeiten dazu, dass der Computer den mit der Sprache beschriebenen Sachverhalt nicht mehr „versteht“. Das hast du sicher schon beim Programmieren des öfteren erlebt. Bei natürlichen Sprachen kann man sich meist darauf verlassen, dass bei sprachlichen Ungenauigkeiten der Kommunikationspartner schon versteht, was man meint. Allerdings kommt es dabei auch oft zu Missverständnissen. Solche Missverständnisse gilt es bei Computersprachen durch Präzision und Eindeutigkeit zu vermeiden.

1.7 Zusammenfassung:

Ein **Alphabet** ist eine nicht-leere endliche (geordnete Menge) von Symbolen und wird mit dem Symbol Σ (Sigma) bezeichnet.

Beispiele:
Das Alphabet zur römischen Zahldarstellung ist die Menge

$$\Sigma = \{I, V, X, L, C, D, M\}.$$

Das Alphabet zur Zahldarstellung der natürlichen Zahlen im Dezimalsystem ist die Menge

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

Das Alphabet zur „ipigisi-Sprache“ ist die Menge

$$\Sigma = \{i, p, g, s\}.$$

Ein **Wort** über einem Alphabet ist eine Hintereinanderreihung endlich vieler Symbole aus einem vorgegebenen Alphabet, auch wenn es nicht den Regeln der Sprache entspricht. Das Wort, das aus keinem Symbol besteht, heißt **leeres Wort** und wird mit ε (Epsilon) bezeichnet.

Beispiel: Mit dem Alphabet der römischen Zahldarstellung lassen sich eine Vielzahl von Wörtern bilden, u.a. LXX, XXL und LILLI.

Die **Menge aller Wörter** über einem Alphabet Σ wird mit Σ^* bezeichnet.

Beispiel: Zum Alphabet der römischen Zahldarstellung erhält man die Menge Σ^*, indem man systematisch (alphabetisch) alle Wörter über Σ erzeugt, auch wenn dabei ungültige Zahlen auftreten:

$$\Sigma^* = \{\varepsilon, I, V, X, \ldots, M, II, IV, IX, \ldots, IM, VI, \ldots, VM, \ldots, MM, III, IIV, \ldots\}$$

Eine **(formale) Sprache** über einem Alphabet Σ ist eine bestimmte Teilmenge der Menge Σ^* aller möglichen Wörter über Σ.

Beispiel: Die Sprache

$$L = \{I, II, III, IV, V, VI, VII, VIII, IX, X, \ldots\}.$$

ist einen Teilmenge von Σ^*, weil sie nur diejenigen Wörter enthält, die gültige römische Zahlen darstellen.
Eine andere Sprache über dem selben Alphabet ist die Sprache der großen Größen (für T-Shirts usw.):

$$L = \{L, XL, XXL, XXXL, \ldots\}.$$

Für diese Sprache könnte man natürlich auch ein reduziertes Alphabet $\Sigma = \{L, X\}$ benutzen.

Bei der Kommunikation mit natürlichen Sprachen können Zweideutigkeiten und Missverständnisse auftreten. Dies kann man durch Verwendung einer **formalen Sprache** verhindern, die als Menge zulässiger Zeichenketten über einem Alphabet definiert wird. Die Regeln zur Bildung zulässiger Zeichenketten stellen die **Syntax** einer Sprache dar. Die Bedeutung eines Wortes nennt man auch **Semantik**.

1.8 Übungen

1. PGM (Portable Graymap) kann als Sprache zur Beschreibung von Graustufenbildern aufgefasst werden. Das folgende Bild ist mit dieser Sprache angefertigt worden.

   (a) Übersetze das Bild in die Sprache PGM.

   (b) Welches Alphabet Σ liegt der Sprache PGM zu Grunde? Gib Beispiele für Wörter über Σ an, die zu PGM bzw. nicht zu PGM gehören.

   (c) Verdeutliche am Beispiel PGM, was man unter Syntax und Semantik einer Sprache versteht.

2. Zur Sprache „hallihallo“ gehören die folgenden Wörter: hallo, hallihallo, hallihallihallo, hallihallihallihallo,...

   (a) Welche Alphabete könnte man hier als Grundlage der Sprache wählen?

   (b) Beschreibe die Regeln, nach denen die Wörter der „hallihallo“-Sprache gebildet werden.

3. Bunny Banana ist der Teenie-Pop-Star im Biberland. Alle jungen Biber würden gerne so singen wie Bunny. Bunny Banana erklärt den Fans, wie die Lieder gemacht sind:

   - Eine Silbe wird aus einem Konsonanten (z. B.: d, l, n, s) und aus einem Vokal (a, e, i, o, u) gebildet. Beispiele: do, nu, la.
   - Ein Vers besteht aus einer ungeraden Anzahl der gleichen Silbe, wobei der mittleren Silbe ein „p di“ angehängt wird. Beispiele: da dap di da, ne ne nep di ne ne.
   - Ein Lied besteht aus einem oder mehreren Versen. Wenn ein Lied mehrere Verse hat, darf es mit „yeah“ enden, muss aber nicht.

   Beim Biber-Song-Contest haben vier Biber versucht, wie Bunny Banana zu singen. Aber nur einer war erfolgreich. Entscheide, welcher Biber die Bunny-Banana-Sprache beherrscht und korrekte Wörter dieser Sprache gebildet hat.

   - Biber-Max: si sip di si su dup di su
   - Biber-Tina: da da dap di da da yeah
   - Biber-Paul: nu nu nup di nu nu di di dip di di
   - Biber-Trixi: sa sa sap di sa sa lu lu lup di lu lu yeah

4. Die Programmiersprache *Java* ist auch eine formale Sprache. Das Alphabet von *Java* besteht aus den Schlüsselwörtern (z. B. `if`, `for`, `class` und `public`) und Zeichen (z. B. den Klammern), die in der Definition von *Java* festgelegt sind. Was entspricht dann den Wörtern der formalen Sprache *Java*?

2 Grammatiken formaler Sprachen

2.1 Beispiel: E-Mail-Adresse

Für die Syntax einer E-Mail-Adresse gelten die folgenden Regeln:

- Eine E-Mail-Adresse hat links vom @-Zeichen eine Benutzerkennung und rechts davon den Namen einer Internet-Domäne.
- Die Benutzerkennung besteht aus mindestens einem Zeichen. Es dürfen Kleinbuchstaben, Zahlen, Punkt, Ausrufezeichen, Binde- und Unterstrich verwendet werden.
- Die Internet-Domäne setzt sich zusammen aus mindestens einer Unterdomäne gefolgt von einer Top-Level-Domäne wie `de` oder `com`. Dazwischen steht ein Punkt.
- Die Top-Level-Domäne besteht nur aus Buchstaben. Es sind mindestens zwei und höchstens vier Buchstaben.
- Jede Unterdomäne kann Buchstaben, Ziffern und den Bindestrich enthalten. Es sind mindestens zwei Zeichen nötig. Am Anfang und am Ende darf kein Bindestrich stehen.

Aufgabe: Welche der folgenden E-Mail-Adressen sind syntaktisch korrekt, welche nicht?

`senf@brat.wurst.de, brat@-kartoffel.salat.net,`
`jaeger.schnitzel@pommes-frites.de, brat.kartoffel@s.piegel.ei,`
`x3@xx.it, brat@wurst@scharf_senf_dazu.de, x123557@issmi.ch`

Die Vorgaben für eine gültige E-Mail-Adresse lassen sich in Regeln umwandeln:

Regel 1:

```
E-Mail-Adresse = Benutzerkennung "@" Domain
```

`E-Mail-Adresse`, `Benutzerkennung` und `Domain` sind sogenannte **Nichtterminalsymbole**, d.h. so etwas wie Platzhalter, die noch durch weitere Regeln erklärt werden müssen. Das Zeichen `@` ist ein **Terminalsymbol**, also ein Zeichen der gültigen Zeichenkette. Das erste Nichtterminalsymbol `E-Mail-Adresse` nennt man **Startsymbol**.

Regel 2:

```
Benutzerkennung = Einzelzeichen {Einzelzeichen}
```

Dies bedeutet, dass die Benutzerkennung aus mindestens einem Einzelzeichen besteht. Die geschweiften Klammern bedeuten, dass es kein, ein oder mehrere Zeichen sind, die an das erste Einzelzeichen angehängt werden können. Die geschweifte Klammer gehört nicht zur erzeugten Sprache, sondern ist ein **Metazeichen**.

Regel 3:

```
Einzelzeichen = Buchstabe | Ziffer | "-" | "_" | "." | "!"
```

Regel 4:

```
Ziffer = "0" | "1" | "2" | ... | "9"
```

Regel 5:

```
Buchstabe = "a" | "b" | "c" | ... | "z"
```

Bei Regel 3 bis Regel 5 steht das Metazeichen `"|"` für **Alternativen** und kann als „oder“ gelesen werden..

Regel 6:

```
Domain = Subdomain { "." Subdomain } "." TopLevelDomain
```

Regel 7:

```
TopLevelDomain = Buchstabe Buchstabe [Buchstabe] [Buchstabe]
```

Regel 8:

```
Subdomain = (Buchstabe | Ziffer) { Buchstabe | Ziffer | "-" }
            (Buchstabe | Ziffer)
```

Mit den runden Klammern kann man den Ausdruck `Buchstabe | Ziffer` geeignet **gruppieren**, d.h. hier das erste und letzte Zeichen festlegen.

2.2 Grammatik-Definition

Die eindeutige Festlegung einer formalen Sprache mit Regeln heißt **Grammatik**.

Definition: Eine **Grammatik** G wird durch ein 4-Tupel (N, T, S, P) spezifiziert:

- N ist die Menge der **Nichtterminalsymbole**.
- T ist die Menge der **Terminalsymbole**.
- $S \in N$ ist das **Startsymbol**.
- P ist die Menge der Regeln oder **Produktionen**.

Eine Grammatik erzeugt eine Sprache, die aus allen Wörtern besteht, die sich durch Anwendung der Regeln der Grammatik erzeugen (**ableiten**) lassen.

Die in den oben angegebenen Regeln benutzte Darstellungsform der Regeln mit Optionsklammern `{}`, Wiederholungsklammern `[]` und Gruppierungsklammern `()` neben der Alternative `|` nennt man **erweiterte Backus-Naur-Form (EBNF).**

2.2.1 Beispiel 1:

Die Grammatik der E-Mail-Adressen kann so angegeben werden:
$N = \{$ E-Mail-Adresse, Benutzerkennung, Domain, Einzelzeichen, Buchstabe, Ziffer, Subdomain, TopLevelDomain$\}$
$T = \{$`"a", "b",...,"z", "0", "1",..., "9", "@", "-", "_", ".", "!"`$\}$
$S =$ E-Mail-Adresse
$P =$ {Benutzerkennung = Einzelzeichen {Einzelzeichen},...}

2.2.2 Beispiel 2:

Eine Grammatik $G = (N, T, S, P)$ ist wie folgt gegeben:
$N = \{s_0, A, B\}$
$T = \{0, 1\}$
$S = s_0$
$P = \{s_0 \longrightarrow 1s_0 \,|\, 0A, A \longrightarrow 1s_0 \,|\, 0B, B \longrightarrow 1s_0 \,|\, 0B \,|\, 0\}$

Eine gültige Ableitung ist z. B.

$$s_0 \longrightarrow 1s_0 \longrightarrow 10A \longrightarrow 100B \longrightarrow 1000.$$

1000 ist ein Wort dieser Sprache. Die Sprache besteht aus bestimmten Binärzahlen.

2.2.3 Reguläre Grammatiken

Wenn eine **Grammatik** $G = (N, T, S, P)$ mit $N = \{A, B, C, ...\}$ und $T = \{a, b, c, ...\}$ nur Regeln der Form

$$A \longrightarrow aB \,|\, B \,|\, a \,|\, \epsilon$$

besitzt (rechts steht höchstens ein Nichtterminalsymbol hinter höchstens einem Terminalsymbol), dann nennt man die Grammatik **rechtslinear** oder **rechtsregulär**.
Insbesondere gehört auch die rekursive Regel

$$A \longrightarrow aA$$

zu einer rechtsregulären Grammatik.
Entsprechend heißt die Grammatik mit diesen Regeln

$$A \longrightarrow Ba \,|\, B \,|\, a \,|\, \epsilon$$

(links steht höchstens ein Nichtterminalsymbol vor höchstens einem Terminalsymbol), **linkslinear** oder **linksregulär**.
Eine Grammatik heißt **regulär**, wenn sie linksregulär oder rechtsregulär ist.

Die Grammatik von Beispiel 2 ist rechtslinear bzw. rechtsregulär.

2.2.4 Klassifizierung von Sprachen

Reguläre Sprachen (CHOMSKY-Typ 3) sind Sprachen, die von einer regulären Grammatik erzeugt werden.

Kontextfreie Sprachen (CHOMSKY-Typ 2) werden von Grammatiken erzeugt mit einer Regel der Form $A \longrightarrow \alpha$, wobei α eine nichtleere Symbolfolge aus Terminal- und/oder Nichtterminalsymbolen ist. Bei dieser Ersetzung kommt es nicht auf den Kontext, d.h. die Umgebung von A an.

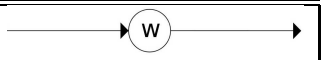
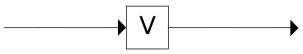
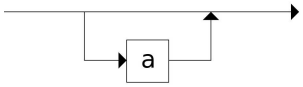
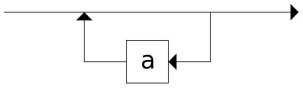
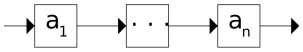
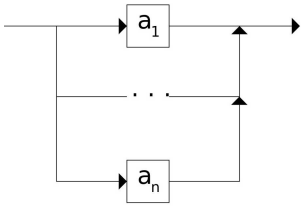
Kontextsensitive Sprachen (CHOMSKY-Typ 1) werden von Grammatiken erzeugt, deren Regeln alle die Form $\beta A \gamma \longrightarrow \beta \alpha \gamma$ haben, d.h. die Ersetzung des Nichtterminalsymbols A erfolgt nur, wenn A zwischen den Symbolfolgen β und γ, d.h. *im Kontext* von β und γ auftritt.

Allgemeine Sprachen (CHOMSKY-Typ 0) unterliegen keinerlei Einschränkungen bezüglich der Ersetzungsregeln.

Wir interessieren uns hier nur für Sprachen vom Typ 1 bis 3, zu denen die Programmiersprachen gehören.

2.3 Syntaxdiagramme

Die Regeln einer Grammatik lassen sich auch mit einem **Syntaxdiagramm** veranschaulichen:

Bedeutung	EBNF	Diagramm
Terminalsymbol	w	
Nichtterminalsymbol	V	
Optionsklammer: einmal oder keinmal	[a]	
Wiederholungsklammer: keinmal oder beliebig oft	{a}	
Aneinanderreihung (Konkatenation)	$a_1 \ldots a_n$	
Alternative: oder	$a_1 \mid \ldots \mid a_n$	

2.4 Übungen

1. Regeln einer Grammatik sind in EBNF gegeben:

Ziffer	=	0|1|2|...|9
Buchstabe	=	a|b|c|...|z
Bezeichner	=	Buchstabe {Buchstabe | Ziffer}

   (a) Welche Nichtterminalsymbole und welche Terminalsymbole enthält die Grammatik?

   (b) Gib ein Wort der Sprache an und ein Wort, das nicht zur Sprache gehört.

   (c) Zeichne ein Syntaxdiagramm dazu.

2. Gegeben sei die Grammatik G mit $N = \{X, C\}$, $T = \{a, b\}$ und $S = X$. Das Symbol ϵ soll das leere Wort darstellen. Die Regeln:

   $$\begin{aligned} X &\longrightarrow aCa \\ C &\longrightarrow \epsilon \mid bbC \end{aligned}$$

   Prüfe, ob die folgenden Wörter in der Grammatik abgeleitet werden können:

   aa abba abbbba abbba

3. Gegeben sei die Grammatik G mit $N = \{X, C\}$, $T = \{a, b, d\}$ und $S = X$. Das Symbol ϵ soll das leere Wort darstellen. Die Regeln:

   $$\begin{aligned} X &\longrightarrow abC \\ C &\longrightarrow \epsilon \mid abC \mid dC \end{aligned}$$

   Prüfe, ob die folgenden Wörter in der Grammatik abgeleitet werden können:

   ababd abdddabdabab abdabbabd

4. In seinem Buch *Gödel, Escher, Bach: Ein endloses geflochtenes Band* entwickelt Douglas R. Hofstadter eine Formale Sprache, das „MIU-System“, wie folgt:

   - Regel 1: **MI** ist Bestandteil des Systems.
   - Regel 2: In jeder Zeichenkette, deren letztes Zeichen **I** ist, darf **U** hinzugefügt werden.
   - Regel 3: Wenn x eine beliebige Zeichenfolge mit den Symbolen **M**, **I** bzw. **U** ist, dann darf aus der Zeichenfolge **M**x die Kette **M**xx gebildet werden.
   - Regel 4: In jeder Zeichenfolge darf **III** durch **U** ersetzt werden.
   - Regel 5: Wenn **UU** irgendwo vorkommt, kann man es streichen.

   (a) Welche Terminalsymbole und welche Nichtterminalsymbole enthält die Sprache?

   (b) Leite die Worte **MIUIU**, **MUIUIU** und **MUIIIU** ab.

   (c) Leite mindestens drei weitere Worte ab.

(d) Ist **MU** ein Wort der Sprache?

5. Gegeben sei die Grammatik G mit $N = \{s_0, s_1, s_2\}$, $T = \{0, 1\}$ und $S = s_0$. Das Symbol ϵ soll das leere Wort darstellen. Die Regeln:

$$\begin{aligned} s_0 &\longrightarrow \epsilon \,|\, 0s_0 \,|\, 1s_1 \\ s_1 &\longrightarrow 0s_2 \,|\, 1s_0 \\ s_2 &\longrightarrow 0s_1 \,|\, 1s_2 \end{aligned}$$

(a) Die Worte können als Dualzahlen interpretiert werden. Stelle fest, welche der Dualzahlen in der Grammatik ableitbar sind.

Wort	Dezimalzahl	ableitbar?	Wort	Dezimalzahl	ableitbar?
0			1000		
1			1001		
10			1010		
11			1011		
100			1100		
101			1101		
110			1110		
111			1111		

(b) Erläutere die Ableitung am Beispiel 10101.

6. Entwickle eine Grammatik in EBNF für Postanschriften. Beachte die folgenden Aspekte:

- Eine Postanschrift besteht aus einem Personenteil, gefolgt von einer Straße, gefolgt von der Stadt.
- Der Personenteil besteht aus einem Titelteil und einem Namensteil, gefolgt von einem Zeilenende.
- Der Titelteil besteht aus einem Titel oder ist leer.
- Der Vornamenteil besteht aus einem Vornamen oder einem Initial, auf den ein Punkt folgt.
- Der Namensteil besteht aus einem Vornamensteil, einem Nachnamen oder aus einem Vornamensteil und wiederum aus einem Namensteil.
- Eine Straße besteht aus einem Straßenname, gefolgt von einer Hausnummer, gefolgt von einem Zeilenende.
- Eine Stadt besteht aus einer Postleitzahl, gefolgt von einem Stadtnamen, gefolgt von einem Zeilenende.

7. In Ipogesien gibt es folgende Produktionen:

   $S \longrightarrow iAi$

   $A \longrightarrow siAis \,|\, piBis$

   $B \longrightarrow siBis \,|\, g$

   Welche der Wörter ipigisi, isipisigisi, isipisigisisisi können mit den Produktionen abgeleitet werden?

8. NOAMESISCH (aus dem Informatik-Biber-Wettbewerb, Altersstufe 11–13, schwer): Auf der kleinen Insel Noam sprechen die Eingeborenen eine ganz besondere Sprache. In langjähriger Arbeit konnten Sprachforscher folgendes feststellen:

   - Es gibt die Wörter ba, di und du.
   - Durch Verdopplung eines Worts entsteht ein neues Wort; z. B. baba.
   - Ein neues Wort entsteht auch, indem ein Wort vorne und hinten an ein anderes Wort angefügt wird; z. B. baduba.

   (a) Als man versuchte, mit den Noamesen zu reden, funktionierte das ganz gut, doch bei einem der folgenden Worte zuckten die Noamesen nur verständnislos mit den Achseln. Bei welchem?

   A) dudubabadudubabadudu B) didudubadududi

   C) dudubadibadibadu D) dididudidibadibadididudidi

   (b) Gib eine Grammatik zu dieser Sprache an.

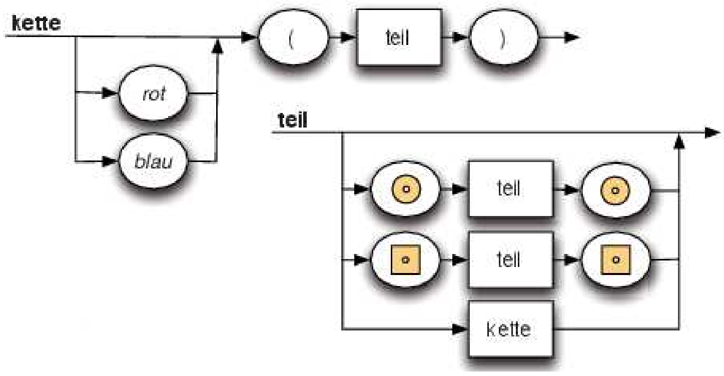
9. Bunte Perlenketten (aus dem Informatik-Biber-Wettbewerb, Altersstufe 11–13, schwer): Die Kinder der kreativen Biberdame Grace basteln Perlenketten. Sie haben verschiedene Holzperlen (quadratisch und kreisförmig), die sie rot oder blau einfärben können. So können sie beispielsweise die folgende Kette basteln:

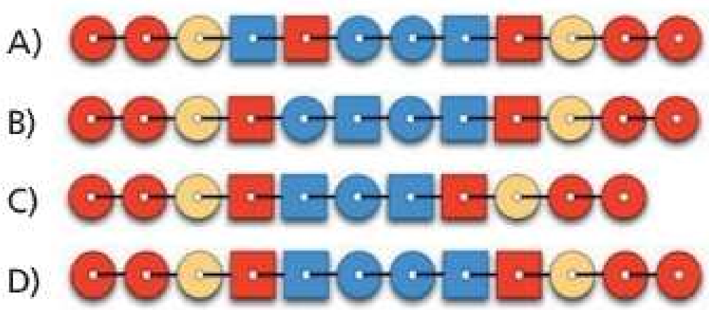
   Grace erklärt den Kindern, dass diese Kette die folgende Kettenbeschreibung hat:

   Grace fertigt nun zwei Zeichnungen an, die „kette“ und „teil“ heißen. Sie möchte nur Ketten haben, deren Kettenbeschreibung man erhalten kann, wenn man den Pfeilen in den Zeichnungen folgt:



Die kleinen Biber basteln vier Ketten. Leider passt nur eine zu Graces Zeichnungen. Welche?



10. Als der Münchner Dienstmann Alois Hingerl in den Himmel kam, überreichte ihm St. Petrus eine Harfe und wies ihn in die himmliche Sprache *Frohlocken* ein. Einige der Wörter dieser Sprache:

    halleluja!

    halleleluja!

    hahahalleluuja!

    luja!

    halleleleluuuuja!

    Auch Wiederholungen sind möglich:

    halleluja,luja!

    halleluja,halleluja,luja!

    Beschreibe diese Sprache mit einem Syntaxdiagramm und stelle die zugehörige Grammatik dar.

    Doch der neue Engel Aloisius tat sich mit der Sprache so schwer, dass Petrus ihn im himmlischen Chor nicht gebrauchen konnte und Gott ihn mit göttlichem Rat an die bayerische Landesregierung auf die Erde schickte. Und weil er unterwegs am Hofbräuhaus vorbeikam und dort sitzen blieb, wartet die Regierung noch heute darauf.[2]

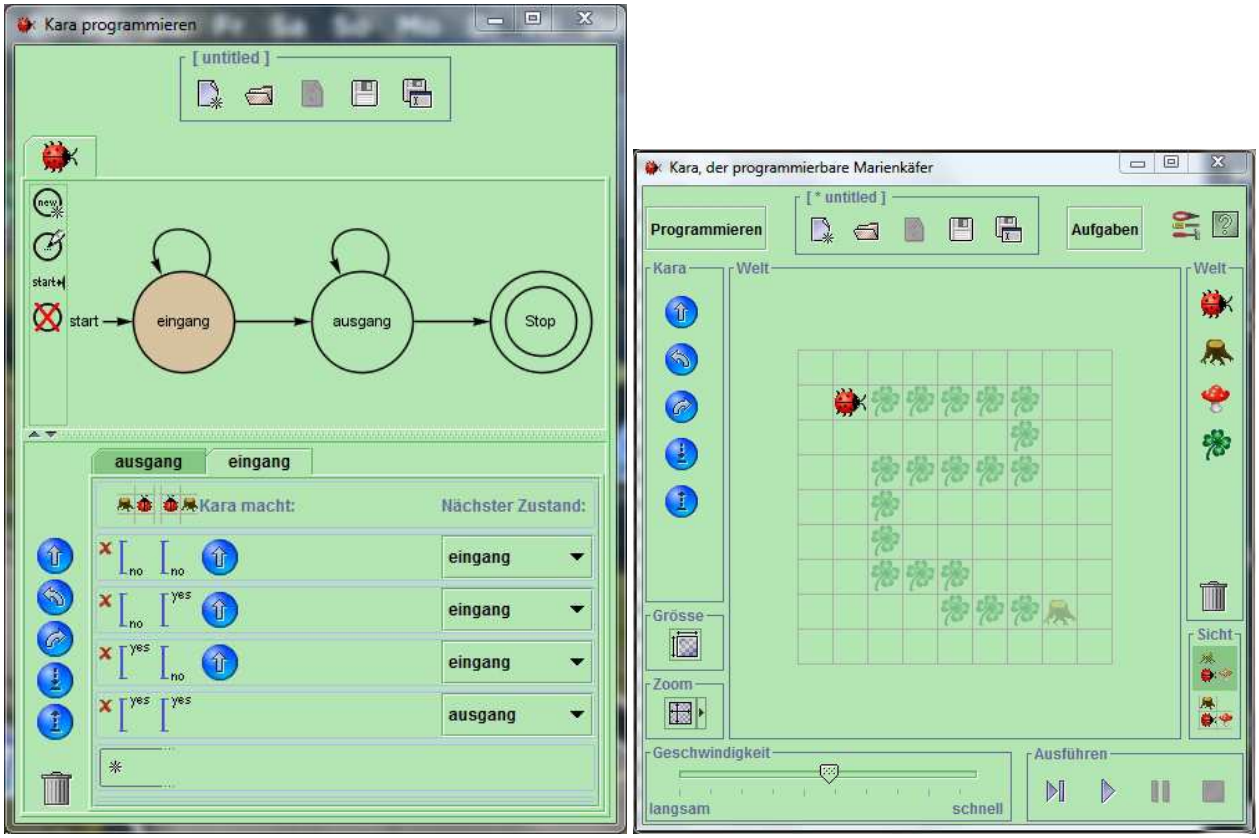
[2] Frei nach Ludwig Thoma, Neufassung hier: http://www.youtube.com/watch?v=USlmFs2A-fU Beim Informatik-Bundeswettbewerb 1990 war das Syntaxdiagramm dieser Sprache vorgegeben. Die Aufgabe bestand darin, einen Parser (Teil eines Compilers) in einer Programmiersprache zu schreiben, der feststellt, ob eine bestimmte Eingabe zur Sprache *Frohlocken* gehört.

3 Erkennung formaler Sprachen und endliche Automaten

3.1 Endliche Automaten

3.1.1 Der Marienkäfer Kara als endlicher Automat

Nach dem Start von *kara.jar*[3] sieht man zuerst das rechte Fenster mit der Welt von Kara. Es gibt in dieser Welt nur Bäume, Pilze und Kleeblätter. Kara kann einen Schritt gehen (er darf aber nicht gegen einen Baumstamm laufen), er kann dabei Pilze verschieben und Blätter aufnehmen und ablegen. Im Weltfenster kann man ihn mit den blauen Symbolen direkt steuern.
Durch Klick auf **Programmieren** öffnet sich das linke Fenster, in dem man die Zustände des Automaten definiert. Der Automat reagiert auf seine Sensoren (Baum vorne?, Baum links? Baum rechts? Blatt unten? Pilz vorne?). Die Aktionen und **Zustandsübergänge** programmiert man graphisch in einer **Übergangstabelle**. Nach Festlegung eines **Startzustandes** kann im rechten Fenster der Automat gestartet werden. Die Zustandsübergänge werden im linken Fenster während des Programmlaufs auch im Diagramm angezeigt (Hinweis: Die dargestellte Welt hat nichts mit dem links dargestellten Automaten zu tun).

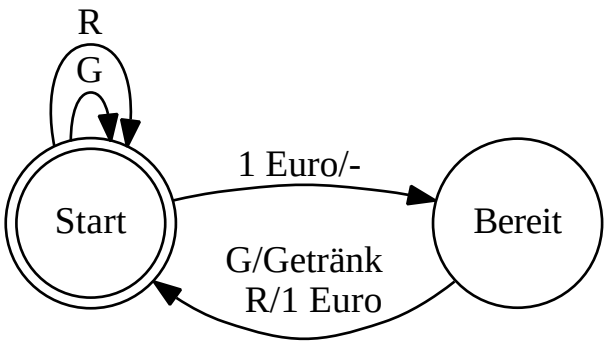


Im linken Fenster ist hier ein Automat mit zwei Zuständen und einem Endzustand dargestellt. Man erkennt die Aktionen und Zustandsübergänge in Abhängigkeit von den Sensoren.

[3] Erhältlich auf http://www.swisseduc.ch/informatik/karatojava/download.html

3.1.2 Einfaches Beispiel mit Ausgabe

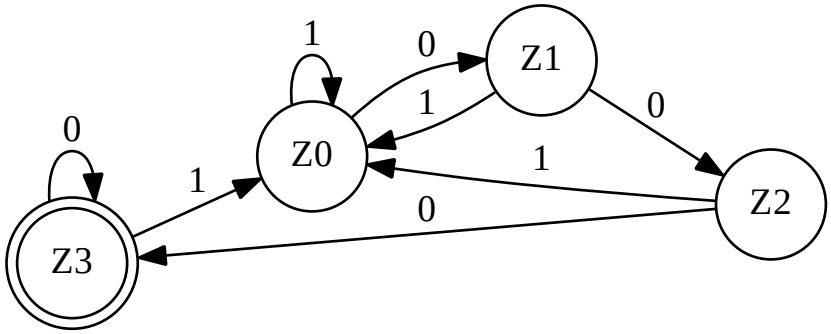
Ein einfacher Getränkeautomat bietet nur ein Getränk an. Wenn man einen Euro eingeworfen hat, kann mit der Getränketaste G das Getränk ausgegeben werden. Drückt man nach dem Einwurf von 1 Euro die Rückgabetaste R, wird das Geld wieder ausgegeben. Ist kein Geld eingeworfen, dann haben G und R keine Wirkung. Das Verhalten eines solchen Automaten kann folgendermaßen veranschaulicht werden:



Der Automat befindet sich anfangs im **Zustand** *Start*. Die Eingaben R und G führen zu keiner **Zustandsänderung**. Nach Einwurf von 1 Euro wechselt der Automat in den Zustand *Bereit*. Wenn man jetzt die Rücktaste R drückt, wird ein Euro ausgegeben und der Automat befindet sich wieder im Zustand *Start*. Nach Druck der Taste G wird ein Getränk ausgegeben und der Automat ist ebenso wieder im Zustand *Start*.

3.1.3 Einfaches Beispiel ohne Ausgabe

Der folgende Automat hat den **Startzustand** Z0. Z3 ist der **Endzustand**.



Tabellarisch kann das Verhalten des Automaten auf zwei Arten als **Übergangstabelle** dargestellt werden:

Zustand	Eingabe	Folgezustand
Z0	0	Z1
Z0	1	Z0
Z1	0	Z2
Z1	1	Z0
Z2	0	Z3
Z2	1	Z0
Z3	0	Z3
Z3	1	Z0

Zustand / Eingabe	0	1
Z0	Z1	Z0
Z1	Z2	Z0
Z2	Z3	Z0
Z3	Z3	Z0

3.1.4 Definition eines endlichen Automaten

Die Beispiele zeigen, dass es Automaten mit und ohne Ausgabe gibt. Automaten mit Ausgabe heißen **Transduktoren**, Automaten ohne Ausgabe nennt man **Akzeptoren**.

Ein endlicher Automat ohne Ausgabe (**Akzeptor**) wird durch ein 5-Tupel

$$A = (Z, \Sigma, z_0, Z_E, \delta)$$

spezifiziert. Dabei gilt:

- Z ist die endliche und nichtleere Menge der Zustände.
- Σ ist das Eingabealphabet, d.h. eine endliche, nichtleere Menge von Symbolen.
- $z_0 \in Z$ ist der Anfangszustand.
- Z_E ist die Menge der Endzustände mit $Z_E \subset Z$.
- $\delta : Z \times \Sigma \to Z$ ist die Zustandsübergangsfunktion.
- Ein Wort, das den Automaten vom Start- in einen Endzustand überführt, gilt als **akzeptiert**.

Einige Vereinbarungen:

- Akzeptierende Zustände werden als doppelter Kreis gezeichnet.
- Es ist in Zeichnungen erlaubt, dass in einem Zustand zu einer bestimmten Eingabe kein Pfeil vorliegt. Dann wird das Wort sofort verworfen. Man kann sich dazu den Übergang in einen *Fehlerzustand* vorstellen. Ein solcher Automat heißt *unvollständig*, sonst *vollständig*.
- Startzustände können auch mit einem Startpfeil gekennzeichnet sein.

Ein endlicher Automat mit Ausgabe (**Transduktor**) wird durch ein 7-Tupel

$$A = (Z, \Sigma, Y, z_0, Z_E, \delta, \lambda)$$

spezifiziert. Dabei gilt:

- Z ist die endliche und nichtleere Menge der Zustände.
- Σ ist das Eingabealphabet, d.h. eine endliche, nichtleere Menge von Symbolen.
- Y ist die endliche und nichtleere Menge der möglichen Ausgaben.
- $z_0 \in Z$ ist der Anfangszustand.
- Z_E ist die Menge der Endzustände mit $Z_E \subset Z$.
- $\delta : Z \times \Sigma \to Z$ ist die Zustandsübergangsfunktion, die jedem Zustand und Eingabezeichen einen Folgezustand zuordnet.
- $\lambda : Z \times \Sigma \to Y$ ist die Ausgabefunktion, die jedem Zustand und Eingabezeichen eine Ausgabe zuordnet.
- Ein Wort, das den Automaten vom Start- in einen Endzustand überführt, gilt als **akzeptiert**.

3.1.5 Übungen

Bild 1

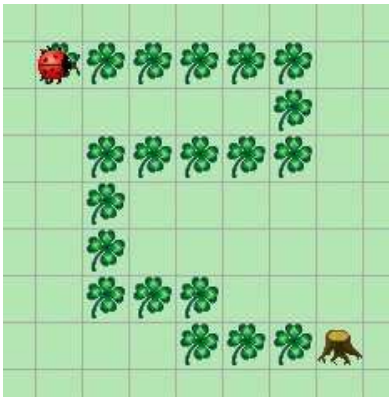


Bild 2

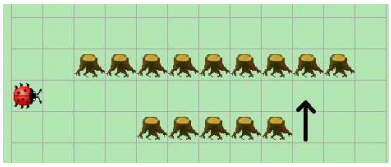
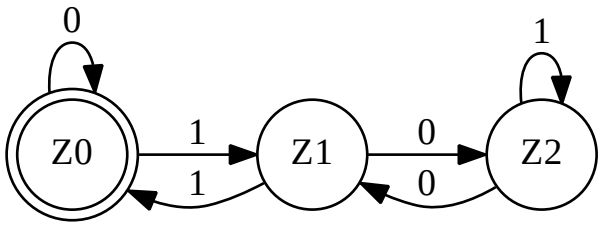


Bild 3

1. (Zu Bild 1:) Kara soll bis zum Baum laufen und auf dem Weg alle Kleeblätter aufsammeln
2. (Zu Bild 2:) Kara soll wie Pacman die Kleeblattspur bis zum Baum „auffressen“.
3. (Zu Bild 3:) Kara soll den Ausgang des Tunnels finden (markiertes Feld). Dazu muss er zunächst den Tunnel durchqueren. Schreibe ein Programm, das ihn auf dem ersten Feld nach dem Tunnel anhalten lässt - er soll nicht bis zum Ende der Galerie laufen! Das Programm soll nicht nur für diesen Tunnel korrekt laufen, sondern für alle geraden Tunnel mit oder ohne Galerien davor oder dahinter. Hinweis: Die Lösung erfordert zwei Zustände! Überlege dir, warum ein Zustand nicht ausreichen kann.

4. Beim dargestellten Automaten ist $Z0$ gleichzeitig Start- und Endzustand.



   (a) Die Worte können als Dualzahlen interpretiert werden. Stelle fest, welche der Dualzahlen vom Automaten akzeptiert werden.

Wort	Dezimalzahl	akzeptiert?	Wort	Dezimalzahl	akzeptiert?
0			1000		
1			1001		
10			1010		
11			1011		
100			1100		
101			1101		
110			1110		
111			1111		

   (b) Stelle fest, ob 10101 vom Automaten akzeptiert wird.

5. Bestimme einen Akzeptor für Dezimalzahlen in Computerschreibweise (Dezimalpunkt statt -komma, e für Zehnerpotenzen) in mehreren Schritten.

   (a) Natürliche Zahlen, z. B. 13456

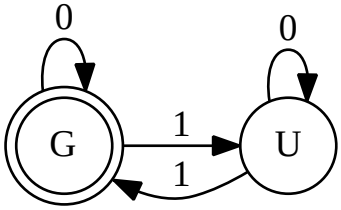
   (b) Ganze Zahlen mit und ohne Vorzeichen, z. B.-3855, 22555, +23

   (c) Dezimalzahlen mit Dezimalpunkt, z. B. 0.345, -14.3, +13.456, 355

   (d) Dezimalzahlen in Exponentendarstellung, z. B.-2.3e24, 4.3e-5, 3e17

6. Bestimme einen Akzeptor für Java-Bezeichner. Diese beginnen mit einem Buchstaben und können außer weiteren Buchstaben noch Ziffern und den Unterstrich enthalten.

7. Begründe, was vom folgenden Akzeptor akzeptiert wird. G ist gleichzeitig Anfangs- und Endzustand.



8. Beschreibe mit einem „Seelenautomaten“ die Theologie des *Thomas von Aquin*. Der Mensch wird infolge der Erbsünde im Stand der Schuld geschaffen. Durch die Taufe

gelangt er in den Stand der Gnade. Durch Handlungen wie Götzendienst, Gotteslästerung oder Ehebruch gelangt er in den Stand der Schuld. Beichte, Reue und Absolution stellen den Stand der Gnade wieder her. Die Wirkung des Todes hängt vom Zustand der Seele ab. Im Stand der Gnade führt er zur Erlösung, im Stand der Schuld zur Verdammnis.

9. Entwirf einen Akzeptor, der alle Binärzahlen akzeptiert, die durch 4 teilbar sind, also mit der Ziffernfolge 00 enden.

10. Ein Automat wird als Filter für Webseiten in einer Schule eingesetzt. Er soll alle Webadressen blockieren, die irgendwo die Buchstabenfolge „sex“ enthalten.

11. Die *Zentralstelle für feministische Wahrheitsfindung und Sprachbereinigung* möchte auch in englischsprachigen Beiträgen alle unliebsamen Texte erfassen und durchsuchen können. Entwirf dazu einen Akzeptor, der alle Wörter akzeptiert, die mit „man“ enden. Akzeptiert werden also Herman, Müsliman, maman und Mangaman, nicht aber command oder manamana.

12. Entwirf einen Geldspielautomaten, der nach Einwurf eines 1-Euro-Stücks und Drücken eines Hebels gestartet wird. Er hat zwei Walzen, die jeweils als Eingabe betrachtet werden sollen. Die Walzen zeigen Apfel, Birne und Pflaume. Bei zwei Birnen oder Pflaumen werden 2 Euro, bei zwei Äpfeln 3 Euro ausgezahlt.

13. Entwickle einen Akzeptor mit dem Eingabealphabet $\Sigma = \{0, 1\}$, der nur Worte akzeptiert, die

    (a) mit 101 enden.

    (b) irgendwo die Zeichenfolge 101 enthalten.

    (c) nirgendwo die Zeichenfolge 101 enthalten.

    (d) eine ungerade Anzahl von Einsen enthalten.

    (e) eine gerade Anzahl von Nullen und eine gerade Anzahl von Einsen enthalten.

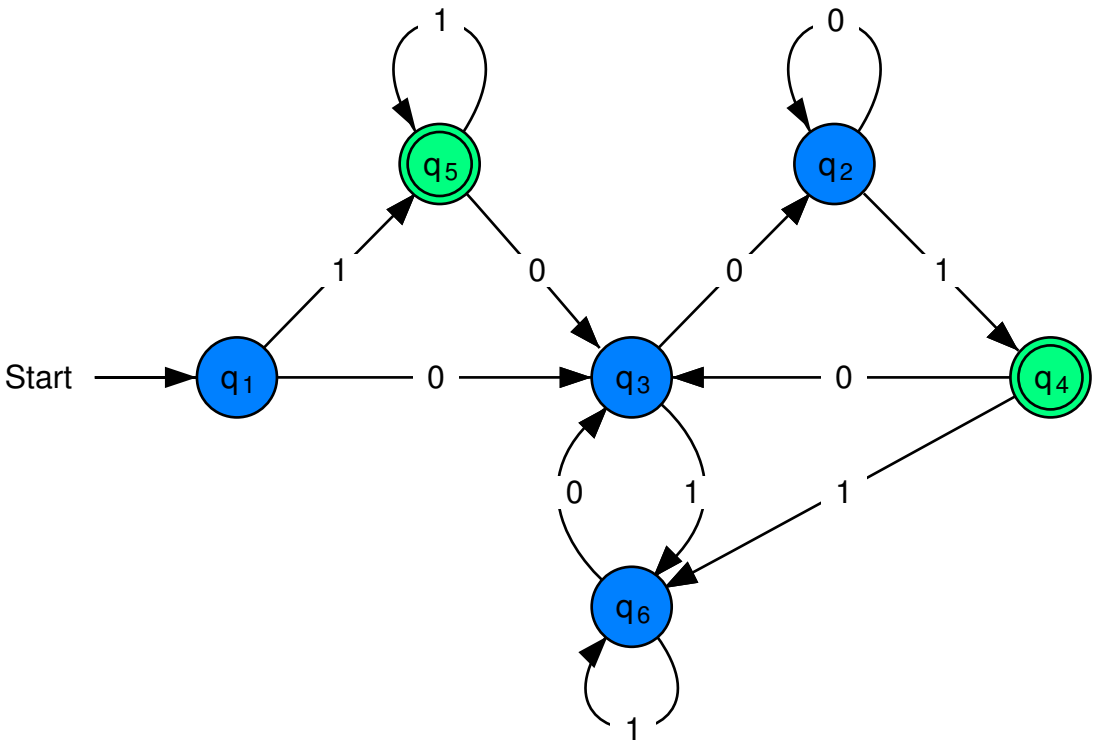
    (f) zwei aufeinanderfolgende Nullen oder zwei aufeinanderfolgende Einsen enthalten.

14. Gib das Zustandsdiagramm zu folgendem Automaten an, der q_1 als Start- und q_3 als Endzustand hat.

Zustand / Eingabe	0	1
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_2	q_4
q_4	q_3	q_5
q_5	q_4	q_5

15. Entwickle einen Akzeptor für die Sprache $L = \{a^m b^n | m \geq 0, n > 0, n \text{ ungerade}\}$. Z.B. soll das Wort *aabbbbb* akzeptiert werden.

16. Der folgende Automat steuert als Programm den Zugang zum Internet auf einem Schulrechner. Schüler mit gültigem Code (als Dualzahl) können im Internet arbeiten, wenn der Code vom Automaten akzeptiert wird.

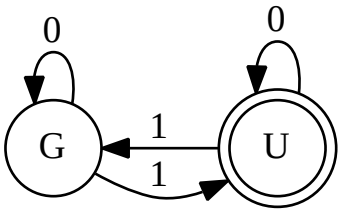


(a) Welche der folgenden Codes werden vom Automaten akzeptiert? 11111, 1001, 11010, 1011101, 0010101001, 10101010

(b) Auch ohne Kenntnis des Automatenaufbaus kamen andere Schüler ohne Code durch Beobachtung der Eingabe gültiger Codes darauf, welches System den gültigen Codes zugrunde liegt. Was konnte man beobachten?

3.2 Automaten und Grammatiken

Der dargestellte Automat akzeptiert Binärzahlen mit ungerader Parität. Sein Startzustand ist G und sein Endzustand U.



Wir suchen nun eine Grammatik, die die gleichen Binärzahlen produziert.

- Als Nichtterminalsymbole nehmen wir die Zustände: $N = \{G, U\}$
- Die Terminalsymbole bestehen aus dem Eingabealphabet: $T = \{0, 1\}$
- Als Startzustand nehmen wir das Anfangssymbol: $z_0 = G$
- Regeln:
  - Aus dem Zustand G ergibt sich bei 1 der Zustand U (und weitere), bei 0 der gleiche Zustand G oder bei 1 der Endzustand U.

    $$G \longrightarrow 1U \,|\, 0G \,|\, 1$$

  - Aus dem Zustand U erreicht man mit einer 1 den Zustand G, bei 0 wieder den Zustand U (und weitere) oder mit 0 den Endzustand U.

    $$U \longrightarrow 1G \,|\, 0U \,|\, 0$$

Das Wort 10110 wird vom Automaten folgendermaßen akzeptiert:

$$G \xrightarrow{1} U \xrightarrow{0} U \xrightarrow{1} G \xrightarrow{1} U \xrightarrow{0} U$$

Ebenso lässt sich das Wort mit den gefundenen Ersetzungsregeln produzieren. Wenn man die sechs Regeln wie folgt nummeriert,

$$G \longrightarrow \overbrace{1U}^{1} \mid \overbrace{0G}^{2} \mid \overbrace{1}^{3}$$

$$U \longrightarrow \underbrace{1G}_{4} \mid \underbrace{0U}_{5} \mid \underbrace{0}_{6}$$

dann kann man folgende Produktion machen:

$$G \xrightarrow{1} 1U \xrightarrow{5} 10U \xrightarrow{4} 101G \xrightarrow{1} 1011U \xrightarrow{6} 10110$$

Zu jedem Akzeptor

$$A = (Z, \Sigma, z_0, Z_E, \delta)$$

kann man eine reguläre Grammatik

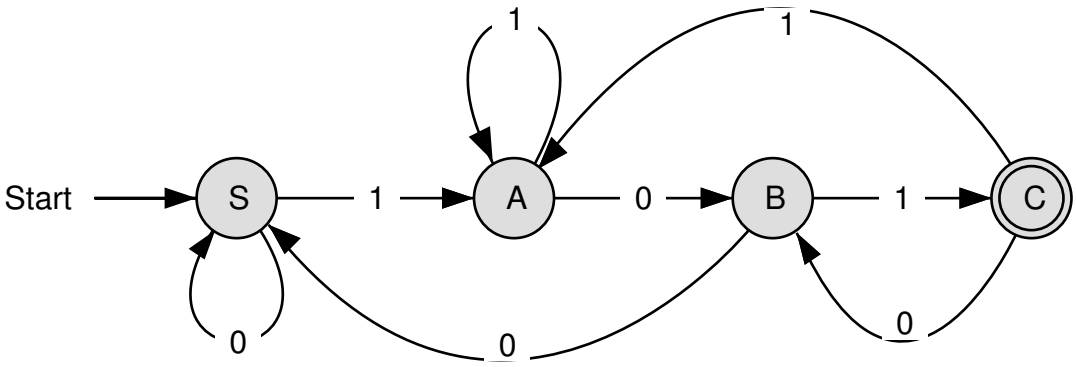
$$G = (N, T, S, P)$$

konstruieren und umgekehrt, so dass Grammatik und Akzeptor die gleiche Sprache beschreiben.

3.2.1 Übungen

1. Konstruiere eine Grammatik zum Akzeptor für ganze Zahlen.

2. Wir haben schon in einer Übungsaufgabe einen endlichen Automaten konstruiert, der alle Binärzahlen mit 101 am Ende akzeptiert.



   (a) Gib dazu nach dem dargestellten Verfahren eine Grammatik an.

   (b) Überprüfe die Richtigkeit der Grammatik mit dem Programm *kfG Edit* des Programmpakets AtoCC.

3. Gegeben ist die Grammatik $G = (N, T, S, P)$ mit

   $N = \{Z, V\}$ und

   $T = \{0, 1, 2, ..., 9, +, -\}$und

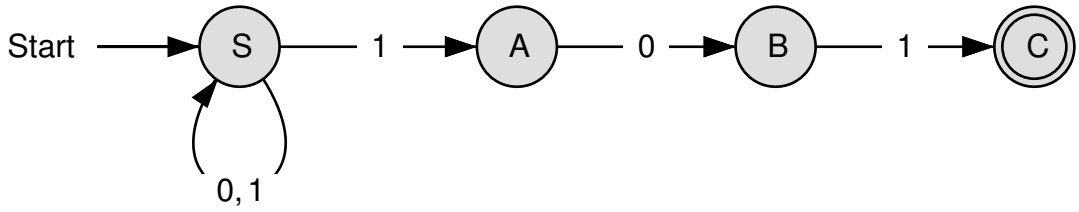
   $P = \{Z \longrightarrow 0|1|2|...|9|0V|1V|2V|...|9V; \quad V \longrightarrow +Z| - Z.\}$

   Konstruiere einen äquivalenten Akzeptor.

4. Eine Grammatik $G = (N, T, S, P)$ ist wie folgt (siehe auch Seite 13) gegeben:
   $N = \{s_0, A, B\}$
   $T = \{0, 1\}$
   $S = s_0$
   $P = \{s_0 \longrightarrow 1s_0 \,|\, 0A, A \longrightarrow 1s_0 \,|\, 0B, B \longrightarrow 1s_0 \,|\, 0B \,|\, 0\}$
   Konstruiere einen äquivalenten Akzeptor. Welche Binärzahlen werden akzeptiert?

3.3 Nichtdeterministische endliche Automaten

Unsere bisher betrachteten Automaten werden auch als **deterministische** endliche Automaten (DEA) bezeichnet[4]. Das bedeutet, dass in jeder Situation das Verhalten des Automaten eindeutig festgelegt ist. Der Automat hat niemals eine Wahl. Das gilt auch für alle Computer, zumindest für die jetzige Computerarchitektur. Bei den für die (ferne?) Zukunft geplanten Quantencomputern ist der Determinismus nur noch im Rahmen der quantenphysikalischen Vorgaben gewährleistet. Da diese Computer mit Licht arbeiten, hat dann ein Photon z. B. an einem halbdurchlässigen Spiegel die Wahl, ob es reflektiert werden oder durch den Spiegel hindurch gehen will. Diese Problematik soll hier nicht weiter erörtert werden.
In der theoretischen Informatik hat aber das Modell eines **nichtdeterministischen** endlichen Automaten (NEA) große Bedeutung. Dieser Automatentyp wurde von Michael O. Rabin und Dana S. Scott 1959 erfunden bzw. erdacht. Für diese Idee erhielten sie 1976 den Turing-Award — was in der Informatik so etwas wie ein Nobelpreis ist.
Auf der vorhergehenden Seite haben wir den DEA für Binärzahlen mit 101 am Ende in einer Übungsaufgabe behandelt. Der folgende nichtdeterministische endliche Automat leistet das Gleiche, weil er die gleiche Menge an Binärzahlen akzeptiert.



Wir machen uns zuerst die Gleichwertigkeit der beiden Automaten durch Betrachtung der zugehörigen Grammatiken klar. Wenn man das im vorhergehenden Abschnitt benutzte Verfahren zur Erzeugung einer Grammatik „ohne langes Nachdenken“ einmal anwendet, erhält man die Produktionsregeln

$$S \longrightarrow 0S \,|\, 1S \,|\, 1A$$

$$A \longrightarrow 0B$$

$$B \longrightarrow 1$$

Aufgabe: Überprüfe diese Grammatik mit dem Programm *kfG Edit* und vereinfache dann die Grammatik so, dass die Produktionsregel nur noch aus einer Zeile besteht.

Wie kann man nun das „Akzeptieren“ eines Wortes beim NEA verstehen? Beim dargestellten Automaten tritt das Problem bei einer 1 im Zustand S auf. Es gibt zwei Möglichkeiten für einen Übergang. Welcher soll gewählt werden? Für die „Arbeitsweise“ des Automaten gibt es drei Interpretationsmuster:

- **Das Gute-Fee-Konzept:** Der Automat hat eine „gute Fee“ eingebaut, die vorher schon weiß, ob ein Wort akzeptiert werden kann. Sie weiß, durch welche Zustandsübergänge dann ein Endzustand erreicht werden kann. Bei der Konstruktion muss man sich überlegen, ob die gute Fee überhaupt einen Weg in einen Endzustand finden kann

[4] deterministisch (lat. determinare = abgrenzen, bestimmen)

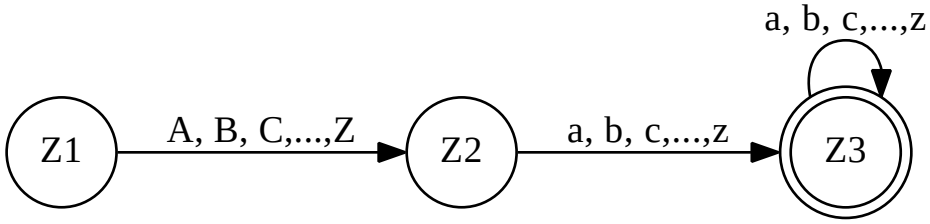
- **Das Cloning-Konzept:** Für jeden Folgezustand wird ein neuer Automat durch „Klonen“ gebildet. Die beiden Klone arbeiten dann zeitlich parallel. So entstehen beim Durchlaufen eines Automaten viele Klone. Ein Wort gilt dann als akzeptiert, wenn es unter den vielen geklonten Automaten wenigstens einen gibt, der in einem Endzustand stoppt. Das Programm *AutoEdit* zeigt die geklonten Automaten am Bildschirm an.

- **Das Rekursions-Konzept:** Bei jeder Verzweigung denkt man sich zwei Zweige einer Baumstruktur aufgebaut. Der Automat läuft die Baumstruktur durch und geht jeweils an die letzte Verzweigung zurück, wenn ein Zweig nicht zu einem Blatt (Endzustand) führte.

Die genaueren Zusammenhänge zwischen Grammatiken, deterministischen und nichtdeterministischen Automaten werden im Informatik-Studium in der theoretischen Informatik geklärt. Uns soll an dieser Stelle genügen, dass wir mit Hilfe eines nichtdeterministischen endlichen Automaten meist viel einfacher eine Grammatik zu einem Problem finden können als mit einem deterministischen.
Es gibt allerdings auch einen Nachteil: Ein nichtdeterministischer Automat ist viel schwieriger zu programmieren als ein deterministischer!

3.4 Vermischte Übungen

1. Das Diagramm beschreibt die Regeln, nach denen gültige Benutzernamen in einem Schulnetzwerk gebildet werden:



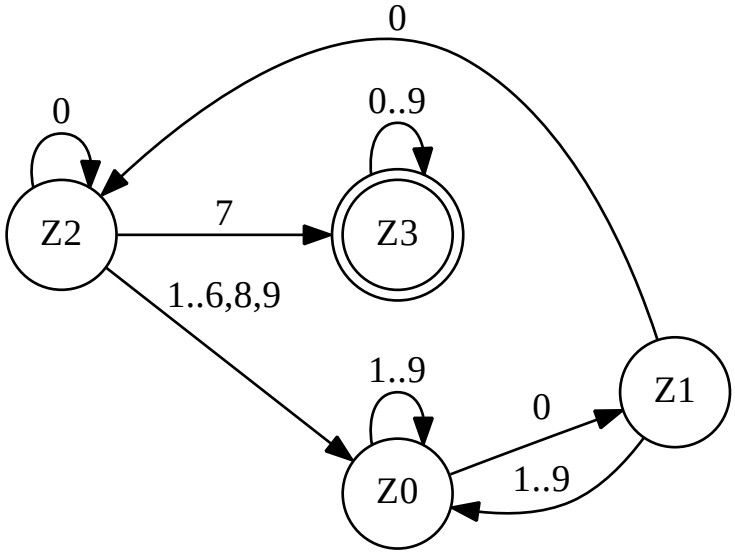
   Ein Benutzername beginnt mit einem Großbuchstaben, gefolgt von mindestens einem Kleinbuchstaben. Es können beliebig viele Kleinbuchstaben folgen. Anna und Moritz sind gültige Benutzernamen. Um auch Nachnamen und weitere Vornamen zu berücksichtigen, werden die Regeln erweitert, so dass zum Beispiel CarolineSchilling, JanPhilippRoth und JackMcGyver gültige Benutzernamen sind.

   (a) Durch welchen zusätzlichen Übergangspfeil muss das Diagramm ergänzt werden, damit es die neuen Regeln beschreibt?

   (b) Formuliere jeweils eine entsprechende Grammatik für die Automaten.

2. (*Aus dem NRW-Zentralabitur 2008*) Der frühere russische Geheimdienst KGB hörte jahrelang den Nachrichtenstrom des britischen Geheimdienstes MI6 ab. Aus Sicherheitsgründen hatte der MI6 sämtliche Nachrichten so codiert, dass diese nur noch aus einer Folge von Ziffern bestanden. Ziffern in der Originalnachricht wurden unverändert übernommen. Die „Mithörer“ des KGB waren insbesondere an Nachrichten interessiert, in denen es um den britischen Geheimagenten James Bond, genannt 007, ging.

   Mitarbeiter des KGB hatten den folgenden Automaten konstruiert, um Nachrichten, in denen das Kürzel 007 enthalten ist, abzufangen.



   (a) Geben Sie das Eingabealphabet, die Menge der Zustände, die Menge der Endzustände und den Startzustand an.

Geben Sie drei verschiedene Eingabeworte an, die der oben angegebene Automat akzeptiert. Dabei sollen nicht alle Worte auf der Ziffer 7 enden.

Zeigen Sie, dass der Automat das Eingabewort 120006007006 akzeptiert.

Erläutern Sie die Funktionsweise dieses Automaten.

(b) Übertragen Sie den Automaten in die Form einer Zustandstabelle.

(c) Der oben abgebildete Automat akzeptiert unter anderem die Teilsprache all der Worte, die auf 007 enden.

Entwickeln Sie eine erzeugende Grammatik für genau diese Teilsprache.

(d) Durch Spione fand der KGB heraus, dass ein weiterer Geheimagent Charlie Brown, genannt 707, sein Unwesen trieb. Auch sein Name tauchte in den abgefangenen Geheimnachrichten des MI6 auf.

Erweitern Sie den oben dargestellten Automaten so, dass dieser alle Nachrichten akzeptiert, in denen einer der beiden Agenten oder auch beide erwähnt werden.

Zeigen Sie die Funktionsfähigkeit Ihres Automaten an zwei verschiedenen Beispielnachrichten, indem Sie die jeweilige Zustandsfolge angeben.

(e) Nachdem der MI6 erkannt hatte, dass der KGB alle Geheimagenten-Namen aus den abgehörten Nachrichten herausfiltern konnte, wurde nach einer sicheren Codierung gesucht. Eines Tages kam Q (der Erfinder in den James-Bond-Filmen) auf die Idee, alle Geheimagenten mit beliebig langen Namen zu versehen, die nur aus den Ziffern 0 und 7 bestehen. Lediglich die Anzahl der Nullen und Siebenen in einem Namen sollte gleich sein. James Bond erhielt beispielsweise den Namen 00000770777770.

Begründen Sie, dass es dem KGB auf diese Weise unmöglich war, einen endlichen Automaten zu konstruieren, der alle Nachrichten erkennen konnte, in denen ein beliebiger Geheimagent erwähnt wurde.

3. Die Sprache des Münchner Dienstmanns Alois Hingerl ist schon auf Seite 19 beschrieben. Gib dazu eine **reguläre** Grammatik an.

4. Die Grammatik des 007-Automaten lässt sich sehr einfach gestalten, wenn man den entsprechenden nichtdeterministischen Automaten zur Verfügung hat. Erstelle aus dem nichtdeterministischen Automaten die zugehörige Grammatik.

5. Ein nichtdeterministischer Automat soll alle durch 8 teilbaren Binärzahlen akzeptieren.

3.5 Implementation eines endlichen Automaten in Java

Die Programmierung eines Automaten in Java stellt sich als erstaunlich einfach heraus:

- Den aktuellen *Zustand* speichert man in einer `int`-Variable.
- Die Symbole des *Eingabealphabets* sind `char`-Werte.
- Die *Zustandsübergänge* kann man durch diverse `if`-Konstruktionen abbilden.

Ein Beispiel ist hier angegeben:

```
1  class Automat
2  {  // Deterministischer endlicher Automat
3    final int Z_0 = 0;  // Startzustand
4    final int Z_1 = 1;
5    final int Z_2 = 2;
6    final int Z_3 = 3;
7    // ...
8    final int Z_F = -1; // Fehlerzustand
9
10   int     zustand;
11
12   public Automat()
13   {   } // nichts zu tun
14
15   public void verarbeiteEingabe(char zeichen)
16   {  if (zustand == Z_0)
17           {  if      (zeichen == '0') zustand = Z_0;
18              else if (zeichen == '1') zustand = Z_1;
19              else                     zustand = Z_F; // Fehlerzustand
20           }
21      else if (zustand == Z_1)
22           {  if      (zeichen == '0') zustand = Z_F;
23              else if (zeichen == '1') zustand = Z_1;
24              else                     zustand = Z_F; // Fehlerzustand
25           }
26   }
27
28   public boolean akzeptiert(String s)
29   {  zustand = Z_0; // Startzustand setzen
30      for (int i = 0; i < s.length(); i++)
31           verarbeiteEingabe(s.charAt(i)); // buchstabenweise Automat fuettern
32      // Ist ein Endzustand erreicht?
33      return (zustand == Z_1);  // Anpassen an den Endzustand oder die Endzustaende
34   }
35 }
```

```
1  class AutomatenTest
2  { SimpleInput in;
3    String eingabe;
4    Automat automat;
5
6
7    public void action()
8    { in = new SimpleInput();
9      automat = new Automat();
10     do
11     {     eingabe = in.getString("Hier das Eingabewort fuer den Automaten eingeben:");
12           if (automat.akzeptiert(eingabe))
13                System.out.println("Der Automat hat " + eingabe + " akzeptiert.");
14           else System.out.println("Der Automat hat " + eingabe + " NICHT akzeptiert.");
15     } while (eingabe.length() > 0);
16
17   }
18 }
```

3.6 Grenzen von endlichen Automaten

Gibt es einen Automaten, der einfache, aber beliebig lange Klammerstrukturen erkennen kann?

`(())`	ok
`((())`	Fehler
`(((((())))))`	ok

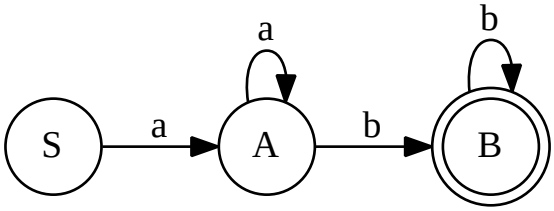
Anders formuliert: Wir ersetzen eine öffnende Klammer durch „a“ und eine schließende Klammer durch „b“. Dann stellt sich die Frage, ob es einen Automaten gibt, der die Sprache L mit $L = \{a^n b^n \mid n = 1, 2, 3, \ldots\}$ erkennt.

$aabb$	ok
$aaabb$	Fehler
$aaaaabbbbb$	ok
a^5b^5	ok
a^5b^4	Fehler

Ein endlicher Automat mit z. B. 300.018 Zuständen muss zählen, wie oft eine öffnende Klammer bzw. ein „a“ erscheint. Zählen geht nur über einen Zustandswechsel. So kann ein Automat mit 300.018 Zuständen nicht bis 300.019 zählen, weil er sonst einen Zustand doppelt annehmen müsste, z. B. den Zustand für 300.015 Klammern. Damit kann er aber auch nicht mehr unterscheiden zwischen 300.015 oder 300.019 Klammern.

Satz über die Grenzen von endlichen Automaten:
Die Sprache L mit $L = \{a^n b^n \mid n = 1, 2, 3, \ldots\}$ kann nicht von einem endlichen Automaten erkannt werden. Sie ist also nicht regulär.

Erstaunlich ist, dass die Sprache L mit $L = \{a^m b^n \mid m, n \geq 1 \text{ bel.}\}$ völlig problemlos mit einem einfachen endlichen Automaten erkannt werden kann:



Daraus erhält man relativ leicht eine Grammatik mit diesen Regeln:

$$S \longrightarrow A$$

$$A \longrightarrow aA \mid bB \mid b$$

$$B \longrightarrow bB \mid b$$

Vereinfacht man die Grammatik, dann stellt auch diese die Sprache dar:

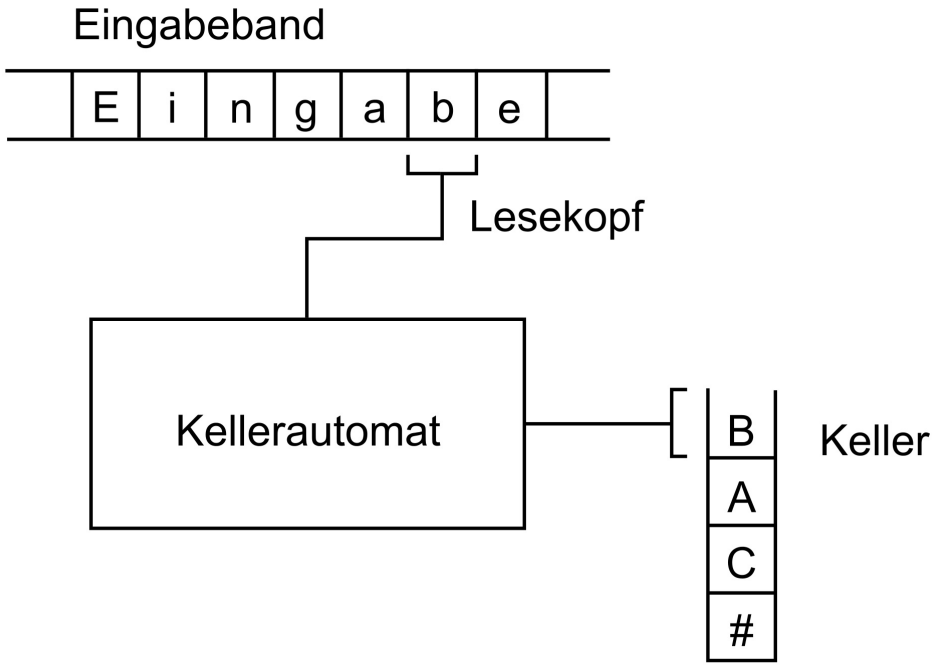
$$A \longrightarrow aA \mid aB$$

$$B \longrightarrow bB \mid b$$

4 Nicht-reguläre Grammatiken

4.1 Kellerautomat

Ein **Kellerautomat** ist ein endlicher Automat, der um einen **Kellerspeicher** erweitert wurde. Ein Kellerspeicher heißt auch **Stapel** oder **Stack**, dementsprechend heißt der Automat auch **Stackmaschine**. Im Englischen spricht man von *pushdown automaton*, kurz **PDA**.
Das Prinzip des Kellerspeichers wurde 1962 in den USA und 1972 in Deutschland von Friedrich Ludwig Bauer[5] und Klaus Samelson[6] zum Patent angemeldet.



Der Keller enthält Inhalte des **Kelleralphabets**, wobei es nur Zugriff auf das oberste Element gibt. Dieses Element kann durch **pop** vom Stapel genommen werden. Mit **push** wird ein neues Element auf den Stapel gelegt.
Das Kellerprinzip ist grundlegend für das Verständnis von Programmiersprachen und der Arbeitsweise von Computern. Erinnert sei an dieser Stelle an die rechnerinternen Vorgänge bei rekursiven Algorithmen.

[5] Friedrich Ludwig Bauer (geb. 10. Juni 1924) ist ein Pionier der deutschen Informatik. Er erfand 1957 das Prinzip des Stapels und hielt 1967 an der Technischen Universität München die erste offizielle Informatik-Vorlesung in Deutschland.

[6] Klaus Samelson (geb. 21. Dezember 1918, gest. 25. Mai 1980) war Mathematiker, promovierter Physiker und Informatik-Pionier. Er arbeitete maßgeblich an der Entwicklung programmgesteuerter Rechenanlagen und im Compilerbau.

Ein **Kellerautomat** wird durch ein 7-Tupel

$$A = (Z, \Sigma, Y, z_0, Z_E, \delta, \lambda)$$

spezifiziert. Dabei gilt:

- Z ist die endliche und nichtleere Menge der Zustände.
- Σ ist das Eingabealphabet, d.h. eine endliche, nichtleere Menge von Symbolen.
- K ist die endliche und nichtleere Menge der möglichen Kellerinhalte:

  $$K = \{k_1, k_2, k_3, \ldots k_p\}$$

- $z_0 \in Z$ ist der Anfangszustand.
- Z_E ist die Menge der Endzustände mit $Z_E \subset Z$.
- $\delta : Z \times \Sigma \times K \to Z$ ist die Zustandsübergangsfunktion, die jedem Zustand, Eingabezeichen und Kellerelement einen Folgezustand zuordnet.
- $\lambda : Z \times \Sigma \times K \to K$ ist die Ausgabefunktion, die jedem Zustand, Eingabezeichen und Kellerinhalt einen Kellerinhalt zuordnet.
- Ein Wort, das den Automaten vom Start- in einen Endzustand überführt, gilt als **akzeptiert**, wenn am Ende der Keller leer ist.
- Das Zeichen $\# \in K$ bezeichnet das Anfangssymbol (leerer Keller).

4.1.1 Beispiel 1: $a^n b^n$

Der Kellerautomat für die Sprache L mit $L = \{a^n b^n \mid n = 1, 2, 3, \ldots\}$ sieht sehr einfach aus. Dabei muss man bedenken, dass das Diagramm die Vorgänge auf dem Stack nicht darstellen kann.

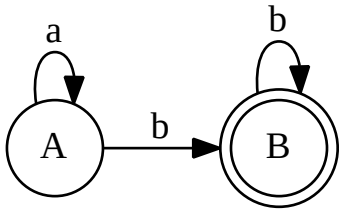
$Z = \{A, B\}$

$\Sigma = \{a, b\}$

$K = \{a\}$

$z_0 = A$

$Z_E = \{B\}$



Die Übergangsfunktionen stellt man am besten mit einer Tabelle dar:

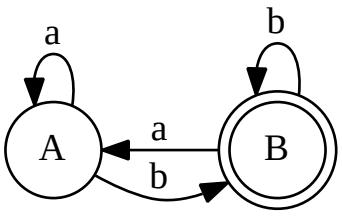
Zustand	Eingabe	Keller	Folgezustand	Kelleroperation
A	a	#	A	push a
A	a	a	A	push a
A	b	a	B	pop a
B	b	a	B	pop a

Die dazugehörige Grammatik sieht verblüffend einfach aus, ist aber keine reguläre Grammatik mehr, da links und rechts von einem Nichtterminalsymbol Terminalsymbole stehen:

$$A \longrightarrow aAb \,|\, ab$$

4.1.2 Beispiel 2: „Wohlgeformte“ Klammerterme

Unter einem „wohlgeformten“ Klammerterm sei ein im Sinne der Mathematik gültiger Ausdruck mit richtiger Anzahl an öffnenden und schließenden Klammern wie z. B. (()) (()()) Wenn wir die öffnende Klammer mit a und die schließende Klammer mit b bezeichen, dann ergibt sich dieser relativ einfache Automat. Auf dem Stapel wird im Prinzip die Anzahl der eingegebenen öffnenden Klammern gespeichert. Jede Eingabe einer schließenden Klammer reduziert die Anzahl der noch vorhandenen öffnenden Klammern.



Zustand	Eingabe	Keller	Folgezustand	Kelleroperation
A	a	#	A	push a
A	a	a	A	push a
A	b	a	B	pop a
B	b	a	B	pop a
B	a	a	A	push a
B	a	#	A	push a

Da die gesamte Information auf dem Stack gespeichert ist, wird auch schnell klar, dass ein zweiter Zustand gar nicht nötig ist. So geht es auch mit nur einem Zustand A:

Zustand	Eingabe	Keller	Folgezustand	Kelleroperation
A	a	#	A	push a
A	a	a	A	push a
A	b	a	A	pop a

Auch hier sieht die dazugehörige Grammatik verblüffend einfach aus, ist aber natürlich keine reguläre Grammatik mehr:

$$A \longrightarrow aAb \,|\, ab \,|\, AA$$

4.1.3 Implementation eines Kellerautomaten in Java

Der schon vorgestellte Java-Automat wird einfach um eine Stack-Klasse erweitert.
Ein Beispiel ist hier angegeben:

```
1  class Kellerautomat
2  {  // Deterministischer endlicher Kellerautomat
3    Stapel st;
4    final int A = 0;  // Startzustand
5    final int B = 1;
6    // ...
7    final int Z_F = -1; // Fehlerzustand
8
9    int    zustand;
10
11   // Beispiel fuer die Sprache L = {a^n b^n | n=1,2,3,... }
12
13   public Kellerautomat()
14   {    } // nichts zu tun
15
16   public void verarbeiteEingabe(char zeichen)
17   {   char ablage; // Platz fuer gepoppte Stackelemente
18       if (zustand == A)
19           {  if (zeichen == 'a')
20                  { if (st.isEmpty())           { zustand = A;  st.push('a'); }
21                    else if (st.readTop()=='a') { zustand = A;  st.push('a'); }
22                    else zustand = Z_F;
23                  }
24              else if (zeichen == 'b')
25                  { if (st.isEmpty())           { zustand = Z_F; }
26                    else if (st.readTop()=='a') { zustand = B;  ablage = st.pop(); }
27                    else zustand = Z_F;
28                  }
29              else        zustand = Z_F; // Fehlerzustand
30           }
31      else if (zustand == B)
32           {  if (zeichen == 'b')
33                  { if (st.isEmpty())           { zustand = Z_F; }
34                    else if (st.readTop()=='a') { zustand = B;  ablage = st.pop(); }
35                    else zustand = Z_F;
36                  }
37              else        zustand = Z_F; // Fehlerzustand
38           }
39   }
40
41   public boolean akzeptiert(String s)
42   {  zustand = A; // Startzustand setzen
43      st = new Stapel();
44      for (int i = 0; i < s.length(); i++)
45           verarbeiteEingabe(s.charAt(i)); // buchstabenweise Automat fuettern
46      // Ist ein Endzustand erreicht?
47      return (zustand == B && st.isEmpty());  // Anpassen an die Endzustaende
48   }
49 }
```

```
1  class KellerautomatenTest
2  { SimpleInput in;
3    String eingabe;
4    Kellerautomat automat;
5
6    public void action()
7    { in = new SimpleInput();
8      automat = new Kellerautomat();
9      do
10     {    eingabe = in.getString("Hier das Eingabewort fuer den Automaten eingeben:");
11          if (automat.akzeptiert(eingabe))
12               System.out.println("Der Automat hat " + eingabe + " akzeptiert.");
13          else System.out.println("Der Automat hat " + eingabe + " NICHT akzeptiert.");
14     } while (eingabe.length() > 0);
15
16   }
17 }
```

```
1  public class Stapel
2  {
3    private class Knoten // ------------------------------------------------------------
4    {  char ch;
5       Knoten naechster;
6
7      public Knoten(char c, Knoten next)
8      { this.ch = c;
9        this.naechster = next;
10     }
11
12     public char getInhalt()
13     { return ch; }
14
15     public Knoten getNachfolger()
16     { return naechster; }
17   }  // Ende von Knoten // ------------------------------------------------------------
18
19     private  Knoten top;
20
21     public Stapel()
22     { top = null; }
23
24     public boolean isEmpty()
25     { return (top == null);  }
26
27     public void push(char c)
28     { Knoten neu = new Knoten(c, top);
29       top = neu;
30     } // push
31
32     public char pop()
33     { char ablage = ' '; // Rueckgabe bei leerem Stapel
34       if (!isEmpty())
35            { ablage = top.getInhalt();
36              top = top.getNachfolger();
37            }
38       return ablage;
39     } // pop
40
41     public char readTop()
42     {  if (!isEmpty())
43              return top.getInhalt();
44        else return ' ';
45     } // readTop
46 } // Ende von Stapel
```

4.1.4 Aufgaben

1. Zeige ausführlich mit Darstellung des Stacks, dass der Kellerautomat von Beispiel 1 *aaabbb* akzeptiert. und *aab* nicht akzeptiert. Teste das dargestellte Java-Programm mit verschiedenen Eingaben.

2. Schreibe das abgedruckte Java-Programm so um, dass „wohlgeformte“ Klammerterme akzeptiert werden.

3. In der 007-Abituraufgabe (siehe Seite 32) geht es darum, dass ein endlicher Automat eine Folge von Nullen und Siebenern nicht auf gleiche Anzahl untersuchen kann. Zeige, dass dies mit einem Kellerautomaten nun möglich ist. Es genügt ein Zustand.

   (a) Stelle die Übergangstabelle her.

   (b) Schreibe das dazu passende Java-Programm.

4.2 Kontextfreie Grammatiken

4.2.1 Begriffe

Eine Grammatik heißt **kontextfrei** (CHOMSKY-Typ 2), wenn alle Regeln von der Form $A \longrightarrow \alpha$ sind, wobei α eine nichtleere Symbolfolge aus Terminal- und/oder Nichtterminalsymbolen ist. Bei dieser Ersetzung kommt es nicht auf den Kontext, d.h. die Umgebung von A an.

Eine Ableitung heißt **Linksableitung**, wenn immer nur das am weitesten links stehende Nichtterminalsymbol ersetzt wird.
Eine Ableitung heißt **Rechtsableitung**, wenn immer nur das am weitesten rechts stehende Nichtterminalsymbol ersetzt wird.
Satz: (Ohne Beweis) Die Klasse der kontextfreien Sprachen ist gleich der Klasse der Sprachen, die von (nichtdeterministischen) Kellerautomaten akzeptiert werden.

Eine Grammatik heißt **eindeutig**, wenn es für jedes Wort der Sprache nur eine Linksableitung (oder nur eine Rechtsableitung) gibt, sonst **mehrdeutig**.

4.2.2 Aufgaben

1. Eine Satzgliederungsgrammatik hat folgende Regeln (Nichtterminalsymbole sind durch spitze Klammern gekennzeichnet):

<Satz>	$\longrightarrow$	<Nominalphrase> <Verbalphrase>
<Nominalphrase>	$\longrightarrow$	<Eigenname> | <Artikel> <Substantiv>
<Verbalphrase>	$\longrightarrow$	<Verb> | <Verb> <Nominalphrase>
<Eigenname>	$\longrightarrow$	Anne | Bernd | Carla
<Substantiv>	$\longrightarrow$	Katze | Pferd | Heu | Buch | Bild | Zimmer
<Artikel>	$\longrightarrow$	der | die | das
<Verb>	$\longrightarrow$	jagt | frisst | liest | betritt

   Leite die Sätze „Anne liest“ und „das Buch jagt die Katze“ ab. Finde weitere Sätze.

2. Konstruiere eine möglichst einfache Grammatik, die alle Terme mit Klammern und den vier Grundrechenarten erzeugt.

3. Finde mit der Grammatik der vorhergehenden Aufgabe verschiedene Ableitungen für den Term $3 + 4 \cdot 5$.

4. Ergänze die Grammatik der Aufgabe 1 durch folgende Regeln:

<Nominalphrase>	$\longrightarrow$	<Artikel> <Substantiv> <Präpositionalphrase>
<Verbalphrase>	$\longrightarrow$	<Verb> <Nominalphrase> <Präpositionalphrase>
<Präpositionalphrase>	$\longrightarrow$	<Präposition> <Nominalphrase>
<Präposition>	$\longrightarrow$	mit | in | auf | unter
<Artikel>	$\longrightarrow$	dem | den

Leite den Satz „Anne betritt das Zimmer mit dem Bild“ mit zwei verschiedenen Syntaxbäumen ab. Was bedeuten die Varianten?

5. Gegeben sei eine Grammatik mit den Nichtterminalsymbolen S (Anweisung) und B (Bedingung), den Terminalzeichen a und b (Anweisungen), p und q (Bedingungen) sowie IF, THEN, ELSE (reservierte Wörter) und den Produktionen

   S $\longrightarrow$ a | b | IF B THEN S | IF B THEN S ELSE S

   B $\longrightarrow$ p | q

   Zeige, dass die Anweisung IF p THEN IF q THEN a ELSE b zwei verschiedene Linksableitungen und damit auch zwei verschiedene Syntaxbäume besitzt.

6. Gegeben sei eine Grammatik ähnlich wie in der vorhergehenden Aufgabe, aber mit den folgenden Produktionen

S	$\longrightarrow$	S1 | S2
S1	$\longrightarrow$	T | IF B THEN S1 ELSE S2
S2	$\longrightarrow$	T | IF B THEN S | IF B THEN S1 ELSE S2
T	$\longrightarrow$	a | b
B	$\longrightarrow$	p | q

   Zeige, dass die Anweisung IF p THEN IF q THEN a ELSE b in dieser Grammatik nur eine einzige Linksableitung bzw. einen einzigen Syntaxbaum besitzt.

7. Welche Mehrdeutigkeiten treten bei den folgenden Sätzen auf?

   (a) Ich warte neben der Bank.

   (b) Sie suchte den Nachtwächter mit der Lampe.

   (c) Rudolf hat sich in München verliebt.

   (d) Er nahm den Kuchen vom Teller und aß ihn.

   (e) Eines Tages, als die Tochter der Fremden die Speisen in das Zimmer brachte, gab es eine Überraschung.

   (f) Der Diebstahl einer Putzfrau ist nicht über die Haftpflichtversicherung gedeckt.

   (g) Weibliche Mitglieder sind verpflichtet, die in der weiblichen Natur begründeten regelmäßigen Störungen der Bühnenleitung spätestens sechs Tage vor dem Beginn derjenigen Woche zu melden, in der die Störungen zu erwarten sind.

   (h) Wenn der Tierarzt dem Hund einen Zahn zieht, muss die Arzthelferin die Schnauze halten.

   (i) Wir essen jetzt Opa![7]

[7] Korrekte Zeichensetzung rettet Leben!

4.3 Grenzen von Kellerautomaten

Ein Kellerautomat ist leitungsfähiger als ein endlicher Automat. Ein endlicher Automat kann reguläre Sprachen erkennen. Ein Kellerautomat kann aber auch kontextfreie Sprachen erkennen. Es gibt aber noch komplexere Sprachen, die auch ein Kellerautomat nicht mehr bewältigen kann.
Eine solche für einen Kellerautomaten nicht mehr zu verarbeitende Sprache ist verblüffend einfach:

$$L = \{a^n b^n c^n \mid n \in \mathbb{N} \text{ beliebig}\}$$

Beispiel: Ein Automat soll in einem Text einer Textverarbeitung ein unterstrichenes Wort in ein kursiv gesetztes Wort umwandeln. Ein unterstrichenes Wort besteht dabei aus einer Buchstabenfolge gefolgt von der gleichen Anzahl von Backspace-Zeichen und der gleichen Anzahl von Unterstrichen.
Aufgabe: Begründe, warum ein Kellerautomat die Sprache L nicht erkennen kann.
Aufgabe: Die Grammatik mit den Terminalzeichen a, b und c hat die folgenden Produktionen:

S	$\longrightarrow$	a S B C | a B C
C B	$\longrightarrow$	B C
a B	$\longrightarrow$	a b
b B	$\longrightarrow$	b b
b C	$\longrightarrow$	b c
c C	$\longrightarrow$	c c

Diese Grammatik kann die Sprache $L = \{a^n b^n c^n | n \in \mathbb{N} \text{ beliebig}\}$ erzeugen. Zeige dies beispielhaft an den Wörtern *abc* und *aabbcc*.
Eine solche Grammatik heißt **kontextsensitiv**, denn die Nichtterminalzeichen können nur ersetzt werden, wenn ein gewisser Kontext gegeben ist. Z.B. darf das Zeichen C nur dann ersetzt werden, wenn es im Kontext als bC oder cC vorkommt.
Bedeutung für die Informatik: Mit kontextsensitiven Grammatiken können Programmiersprachen definiert und Übersetzer (Compiler) konstruiert werden.

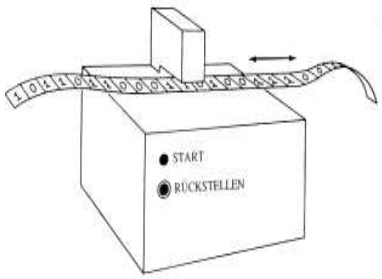
5 Die Turing-Maschine

5.1 Allgemeines

Der Kellerautomat hat offensichtliche Einschränkungen:

- Das Eingabeband kann nur in einer Richtung gelesen werden.
- Im Kellerspeicher ist nur das oberste Element im direkten Zugriff.

Hebt man diese Beschränkungen nach Ideen von Alan Turing auf, kommt man zur Maschine, die zu seinen Ehren benannt wurde.



Die Turing-Maschine besteht aus

1. aus einem Schaltwerk mit einer festen Anzahl von Zuständen Z_1 bis Z_n,
2. einem beidseitig unendlich langen Band (z. B. aus Papier zu denken) als Speicher
3. und einem Schreib-Lese-Kopf.

Das Band ist in Zellen eingeteilt, wobei jede Zelle ein Zeichen eines gegebenen Alphabets aufnehmen kann. Das Band kann zellenweise nach rechts oder links bewegt werden. Man stellt sich allerdings üblicherweise vor, dass der Schreib-Lese-Kopf nach links oder nach rechts läuft.
Zu Beginn enthält das Band die Eingabedaten. Der Kopf steht auf einer Zelle des Bandes. Was passiert nach dem Start der Maschine?

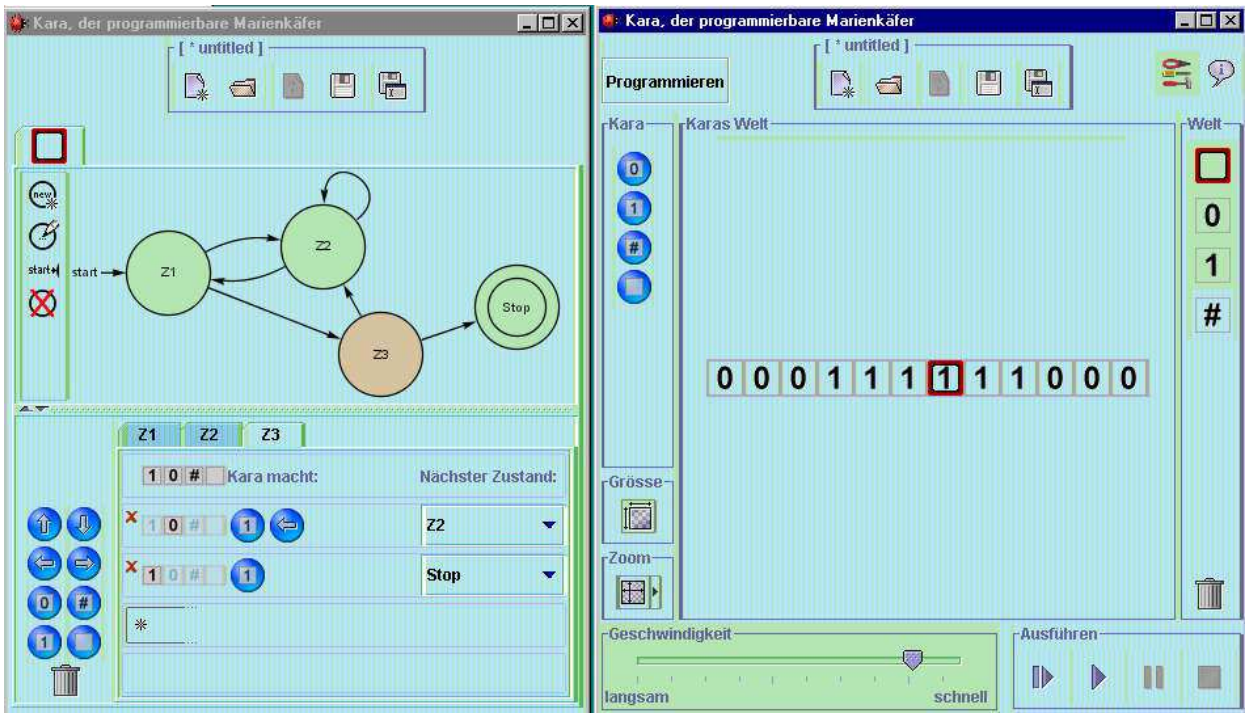
1. Der Schreib-Lese-Kopf liest das Zeichen auf dem Band und gibt es an das Schaltwerk weiter. Aus dem gegenwärtigen Zustand und dem gelesenen Zeichen bestimmt das Schaltwerk aus einer Tabelle
   - ein neues Zeichen,
   - einen neuen Zustand und
   - eine Bewegungsrichtung für den Schreib-Lese-Kopf, die nur links oder rechts lauten kann.
2. Das neue Zeichen wird an der aktuellen Stelle auf das Band geschrieben, der Schreib-Lese-Kopf geht um eine Zelle nach rechts oder links und das Schaltwerk geht in den neuen Zustand über.

3. Einer der Zustände im Schaltwerk muss als Startzustand gesetzt werden.

4. Einer der Zustände im Schaltwerk muss der Stoppzustand sein. Wird dieser Zustand erreicht, stoppt die Maschine.

Diese Maschine ist wirklich sehr primitiv aufgebaut. Man braucht dazu keine komplizierte Mechanik oder Elektronik. Denkbar wäre z. B. eine Turing-Maschine, die aus einer Rolle Toilettenpapier (wegen der hilfreichen Zelleneinteilung) mit weiteren unendlich vielen angeklebten Rollen rechts und links davon, einer Markierung (z. B. ein großer Stein) für die Position des Schreib-Lese-Kopfs und einem Zettel, auf dem die verschiedenen Zustände und Aktionen je nach gelesenem Zeichen, besteht. Als einfaches Alphabet wären die Zeichen 0 und 1 denkbar, wobei man eine 1 durch einen kleinen Stein auf dem Toilettenpapier darstellen könnte und die 0 durch ein steinfreies Blatt repräsentiert wird. Ein Mensch könnte dann den Ablauf einer solchen „Maschine“ in Ruhe in einer geröllreichen Umgebung durch Steinelegen und Nachgucken auf dem Zettel für die Zustände verfolgen.

Eine sehr schöne Simulation einer Turing-Maschine ist mit der Kara-Version möglich.



5.2 Die Turing-Maschine im praktischen Einsatz

Dargestellt ist jeweils die Zustands- bzw. Übergangstabelle. Die Zustände sind mit Z1, Z2,... bezeichnet. S ist der Zustand, der die Maschine stoppt. L bedeutet eine Bewegung des Schreib-Lese-Kopfes nach links, R eine Bewegung nach rechts. Die Turing-Maschine soll immer im Zustand Z1 starten.

1. Beispiel:

Gelesenes Zeichen	0			1		
Zustand	Schreibe	Neuer Zustand	Nach	Schreibe	Neuer Zustand	Nach
Z1	1	Z2	L	0	Z1	L
Z2	0	S		1	Z2	R
S (Stopp)						

Die folgenden Bänder sollen benutzt werden, wobei zur Orientierung eine Positionsnummer angegeben ist, die für die Turing-Maschine irrelevant ist. Die Maschine starte jeweils auf der fettgedruckten Ziffer.

Positionsnr.	01	02	03	04	05	06	07	08	09	10	11	12
Inhalt	0	0	0	0	0	0	0	0	**0**	0	0	0

Positionsnr.	01	02	03	04	05	06	07	08	09	10	11	12
Inhalt	0	0	0	0	0	0	0	0	**1**	0	0	0

Positionsnr.	01	02	03	04	05	06	07	08	09	10	11	12
Inhalt	0	0	0	0	0	0	0	1	**1**	0	0	0

Positionsnr.	01	02	03	04	05	06	07	08	09	10	11	12
Inhalt	0	0	0	1	0	1	1	1	**1**	0	0	0

Was bewirkt diese Turing-Maschine?

2. Beispiel:

Gelesenes Zeichen	0			1		
Zustand	Schreibe	Neuer Zustand	Nach	Schreibe	Neuer Zustand	Nach
Z1	1	Z2	R	1	Z3	L
Z2	1	Z1	L	1	Z2	R
Z3	1	Z2	L	1	S	
S (Stopp)						

Positionsnr.	01	02	03	04	05	06	07	08	09	10	11	12
Inhalt	0	0	0	0	0	0	**0**	0	0	0	0	0

Die Vorgänge macht man sich am besten mit einer Programmablauftabelle klar.

Positionsnr.	Zustand	gelesenes Zeichen	zu schreibendes Zeichen	Neuer Zustand	Kopf-bewegung
7	Z1	0			

3. Beispiel:

Gelesenes Zeichen	0			1		
Zustand	Schreibe	Neuer Zustand	Nach	Schreibe	Neuer Zustand	Nach
Z1	0	Z1	R	0	Z2	R
Z2	1	Z3	R	1	Z2	R
Z3	0	S		1	S	
S (Stopp)						

Positionsnr.	01	02	03	04	05	06	07	08	09	10	11	12
Inhalt	**0**	0	1	1	1	1	0	1	1	0	0	0

Die Vorgänge macht man sich wieder am besten mit einer Programmablauftabelle klar.

Positionsnr.	Zustand	gelesenes Zeichen	zu schreibendes Zeichen	Neuer Zustand	Kopf-bewegung
1	Z1	0			

4. Beispiel:

Beim vorliegenden Band soll an die Einser-Kette rechts eine 1 angefügt werden. Bestimme eine geeignete Zustands- bzw. Übergangstabelle zur Lösung dieses Problems.

Positionsnr.	01	02	03	04	05	06	07	08	09	10	11	12
Inhalt	**0**	0	0	1	1	1	1	0	0	0	0	0

Gelesenes Zeichen	0		1	
Zustand	Schreibe	Neuer Zustand	Schreibe	Neuer Zustand
Z1				
Z2				
Z3				
S (Stopp)				

Erweiterung: Stelle diese Turing-Maschine mit KARA dar. Ein Kleeblatt sei eine 1, ein leeres Feld eine 0. Zur Erinnerung: Kara ist nicht Javakara, sondern arbeitet als endlicher Automat mit Zuständen.
Besser: Benutze **Turing-Kara!**

Fleißige Biber (busy beaver):

Turing-Maschinen, die entlang ihres Bandes hin- und herlaufen, hier ein Symbol unverändert lesen und dort ein Symbol schreiben, könnten uns an fleißige Biber erinnern, die den Fluss zwischen Damm und Wald fleißig durchqueren und bei jeder Tour ein Stöckchen (eine 1) für die Konstruktion ihres Dammes heranschaffen. Wie fleißig kann nun eine Turing-Maschine sein? Wie viele Stöckchen können sie legen? Dabei ist aber zu bedenken, dass eine Turing-Maschine anhalten muss. Eine Maschine zu entwerfen, die unendlich viele Einsen auf ein Band schreibt, ist nicht schwer. Wie ist es aber, wenn die Maschine nur möglichst viele Einsen schreiben soll und irgendwann anhalten soll?
TIBOR RADO, ein ungarischer Mathematiker, dachte sich das aus, was heute als *Busy-beaver-Problem* bezeichnet wird: Gegeben sei eine Turing-Maschine mit n Zuständen und

einem zweielementigen Alphabet bestehend aus 0 und 1. Was ist nun die maximale Anzahl an Einsen, die eine solche Turing-Maschine auf das Band schreiben kann?
Der augenblickliche Wissensstand (2014) sieht so aus:

Anzahl der Zustände	maximale Anzahl der Einsen
1	1
2	4
3	6
4	13
5	≥ 4098
6	$> 3{,}514 \cdot 10^{18276}$

1984 erschien in *Scientific American* ein Artikel über den damals fleißigsten bekannten 5-Zustands-Biber von Uwe Schult, einem deutschen Informatiker. Sein Biber konnte 501 Einsen vor dem Anhalten erzeugen. Als Antwort auf den Artikel führte George Uhing, ein amerikanischer Programmierer, eine Computersuche nach fleißigen Bibern mit 5 Zuständen durch und fand einen, der 1915 Einsen produzieren konnte. 1989 setzten Jürgen Buntrock und Heiner Marxen in Deutschland auf einem Hochgeschwindigkeitsrechner ein Programm zur Suche ein, dass nach drei Tagen einen Biber mit 4098 Einsen fand. Hier ist seine Turing-Tabelle:

Gelesenes Zeichen	0			1		
Zustand	Schreibe	Neuer Zustand	Nach	Schreibe	Neuer Zustand	Nach
Z1	1	Z2	L	1	Z1	L
Z2	1	Z3	R	1	Z2	R
Z3	1	Z1	L	1	Z4	R
Z4	1	Z1	L	1	Z5	R
Z5	1	S	R	0	Z3	R
S (Stopp)						

Es ist natürlich berechtigt, zu fragen, was „dieser Quatsch“ soll, mit dem sich Informatiker mit „viel Freizeit“ beschäftigen. Ich kann mich selbst daran erinnern, dass zu der Zeit, als ich in die Informatik Anfang der 80er Jahre einstieg, in diversen Computerzeitschriften Busy-Beaver-Kolumnen existierten, was ich damals überhaupt nicht verstehen konnte und wollte.
Warum ist dieses Problem nun so berühmt geworden? Die Antwort ist kurz und ernüchternd:

> Das Busy-beaver-Problem ist mit einer Turing-Maschine nicht berechenbar und damit überhaupt nicht berechenbar.

Kein Computer kann jemals herausfinden, wieviel Stöckchen ein fleißiger Biber mit einer bestimmten Anzahl von Zuständen maximal legen kann. Die Informatik hat also ihre Grenzen und damit muss man sich leider abfinden.
Der Beweis des Satzes sei auf das Informatik-Studium verschoben.

5.3 Aufgaben zur Turing-Maschine

1. Die Sprache

   $$L = \{a^n b^n c^n | n \in \mathbb{N} \text{ beliebig}\}$$

   kann von einem Kellerautomaten nicht erkannt werden. Zeige, dass eine Turing-Maschine mit der unten angegebenen Tabelle diese Sprache verarbeiten kann und bei korrektem Wort in einem Endzustand ankommt. Teste die Maschine mit verschiedenen Bändern.

   Damit man Turingkara benutzen kann, soll ein anderes Alphabet benutzt werden. Das Band sieht etwa so aus:

Positionsnr.	01	02	03	04	05	06	07	08	09	10	11	12	13
Inhalt			0	0	0	#	#	#	1	1	1		

   Man startet auf Position 3, d.h. bei der am weitesten links stehenden 0. Auf das Band werden die Zeichen $\rightarrow$, $\uparrow$ und $\downarrow$ als Hilfsmarkierungen geschrieben.

	Zustand	gelesenes Zeichen	zu schreibendes Zeichen	Neuer Zustand	Kopf-bewegung
	Z1	0	$\rightarrow$	Z2	R
	Z1	$\uparrow$	$\uparrow$	Z5	R
	Z2	0	0	Z2	R
	Z2	$\uparrow$	$\uparrow$	Z2	R
	Z2	#	$\uparrow$	Z3	R
	Z3	#	#	Z3	R
	Z3	1	$\downarrow$	Z4	L
	Z3	$\downarrow$	$\downarrow$	Z3	R
	Z4	0	0	Z4	L
	Z4	#	#	Z4	L
	Z4	$\rightarrow$	$\rightarrow$	Z1	R
	Z4	$\uparrow$	$\uparrow$	Z4	L
	Z4	$\downarrow$	$\downarrow$	Z4	L
	Z5	$\uparrow$	$\uparrow$	Z5	R
	Z5	$\downarrow$	$\downarrow$	Z5	R
	Z5	space	space	Stop	L

2. Eine natürliche Zahl ist in **unärer** Darstellung gegeben. Z.B. ist die Zahl 5 durch 5 Einsen auf dem Band dargestellt:

Positionsnr.	01	02	03	04	05	06	07	08	09	10	11	12	13
Inhalt			1	1	1	1	1						

Konstruiere einen **Verdoppler**, der aus dem obigen Band folgendes macht (die Positionsnummern sind hierbei unwichtig):

Positionsnr.	01	02	03	04	05	06	07	08	09	10	11	12	13
Inhalt			1	1	1	1	1	1	1	1	1	1	

6 Quellen

Viele Inhalte und Beispiele habe ich ohne direkte Quellenangabe von diversen Autoren übernommen. Ich beanspruche keineswegs, hier eine eigene geistige Leistung dokumentiert zu haben.

- Rüdeger Baumann, Informatik für die Sekundarstufe II (Band 2), Klett-Verlag 1993 (nicht mehr im Buchhandel erhältlich)
- Klaus Becker u.a. auf http://www.inf-schule.de
- Dr. Jürgen Czischke und Horst Hildebrecht, Materialien einer Fortbildung zum Thema „Automatentheorie“ am Freiherr-vom-Stein-Gymnasium Lünen
- Tino Hempel auf http://www.tinohempel.de/
- Douglas R. Hofstaedter, Gödel, Escher, Bach - ein Endloses Geflochtenes Band, Klett-Cotta 1985
- Eckart Modrow, Automaten Schaltwerke Sprachen, Dümmlers Verlag 1988
- Bernhard Schriek, Informatik mit Java - Band III, Nili-Verlag Werl 2007
- Christian Wagenknecht, Michael Hilscher, Formale Sprachen, abstrakte Automaten und Compiler: Lehr- und Arbeitsbuch für Grundstudium und Fortbildung, Vieweg + Teubner 2009
- Informatik 5, Lehrwerk für Gymnasien, Ernst Klett Verlag 2010
- Informatik Oberstufe 2, Oldenbourg Schulbuchverlag 2010
- http://de.wikipedia.org
- http://www.informatik-biber.de
- http://do.nw.schule.de/goethe-gymnasium/joom/index.php/faecher /naturwissenschaften/informatik/skripten, dort Automatentheorie von Dieter Lindenberg

7 Software

- *AtoCC*: http://www.atocc.de; from **a**utomaton **to** **c**ompiler **c**onstruction

  Ist zwar nur ein Windows-Programm, aber auf der Webseite findet man Anleitungen, wie das Programm auf einem Mac oder unter Linux mit *wine* zum Laufen gebracht werden kann. Der ganze Bereich vom Automaten bis zum Compiler wird abgedeckt. Deutsche Menüführungen, Hilfetexte und Videos! U.A. EPS als Ausgabeformat.

- *JFlap*: http://www.jflap.org; **J**ava **F**ormal **l**anguages and **a**utomata **p**ackage

  Läuft als Java-Programm betriebssystemunabhängig.

- *Kara* und *TuringKara*: http://www.swisseduc.ch/informatik/karatojava/download.html