Formale Sprachen

Problemerfassung

In unserem Alltag ist Sprache ein wichtiger Bestandteil von Kommunikation. Dabei kann ein Sender Sprache nutzen, um Informationen darzustellen und an den Empfänger zu übermitteln. Natürliche Sprachen wie Deutsch, Englisch, aber auch Dialekte wie Bayrisch haben sich historisch entwickelt und entwickeln sich ständig weiter. Typisch für natürliche Sprachen ist, dass sie Mehrdeutigkeiten enthalten können. Das kann zu Missverständnissen führen, aber auch bewusst genutzt werden, um unterschiedliche Informationen zu übermitteln. → **Präsentation 1 Einführung FS**

Abbildung 1

Erkenntnis:

1. Menschen sind bzgl. Sprachabweichungen sehr tolerant.
2. Maschinen kommen mit Sprachabweichungen nicht zurecht.

Fazit:

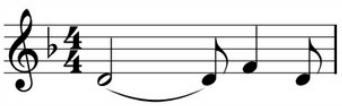
Insbesondere bei der Kommunikation zwischen Computern oder Mensch und Computer ist eine eindeutige Kommunikation sogar zwingend notwendig.

Syntax und Semantik

In natürlichen Sprachen regelt die ***Grammatik*** die korrekte Verwendung von Wörtern im Satz. Ein zentraler Teil davon ist die ***Syntax***, also der Satzbau. Trotz syntaktischer Fehler oder Mehrdeutigkeiten erschließen Menschen dank ***Kontext*** oft die ***Bedeutung (Semantik)***, sodass Kommunikation auch mit Fehlern funktioniert. **Formale Sprachen** dagegen sind so gestaltet, dass ***keinerlei Mehrdeutigkeiten*** entstehen

Fachgebiet	künstliche Sprache	Semantik
Chemie	CO_2	Ein Kohlenstoffdioxidmolekül besteht aus einem Kohlenstoffatom und zwei Sauerstoffatomen.

Informatik	`SELECT *`
	`FROM Kleidung`
	`WHERE geschlecht = "w"`
		Gib alle Informationen in der Tabelle Kleidung zu Kleidungsstücken aus, die dem weiblichen Geschlecht zugeordnet werden.

Musik		Spiele im 4/4 Takt d als halbe Note und übergebundene Achtelnote, danach f als Viertel- und danach d als Achtelnote.

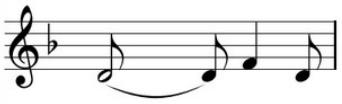
Beispiele:

1. ***Anton trit mit fus gegen bal.***

   Man kann den Sinn des Satzes erahnen. Die Semantik ist also eigentlich richtig. Syntaktisch ist der Satz allerdings falsch, denn er enthält Rechtschreib- und Grammatikfehler, weshalb es bei einer Maschine gar nicht erst dazu kommen würde, dass die Semantik erkannt wird!

Fachgebiet	künstliche Sprache	Syntaxfehler
Chemie	C^2O	Die Anzahl der Atome im Molekül steht tiefgestellt nach dem Atom.

Informatik	`FROM Kleidung`
	`SELECT *`
	`WHERE geschlecht = "w"`
		Die Tabellenangabe mit FROM muss nach der Spaltenauswahl mit SELECT erfolgen.

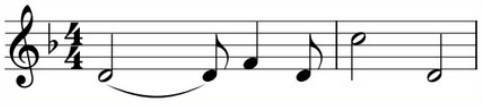
Musik		Die Angabe einer Taktart ist nötig. Bei der halben Note ist der Notenhals falsch (die halbe Note hat einen Hals ohne „Fähnchen“).

2. ***Der Ball tritt mit dem Fuß gegen Anton.***

   Hier ist die Syntax korrekt, denn gegen den Satzbau und die Rechtschreibung ist nichts einzuwenden.
   Der Satz gibt aber keinen Sinn, weshalb er semantisch unbrauchbar ist.

Fachgebiet	künstliche Sprache	semantischer Fehler
Chemie	CO_6	Es gibt kein stabiles Molekül mit einem Kohlenstoff- und sechs Sauerstoffatomen.

Informatik	`SELECT *`
	`FROM Kleidung`
	`WHERE größe = "Hose"`
		Es gibt den Wert „Hose“ in der Spalte „größe“ nicht.

Musik		In D-Moll ist der vierte Ton des Lieds „Wade in the water“ f statt c.

3. ***Hans sieht den Wanderer mit dem Fernrohr.***

   (Wer hat das Fernrohr?)

   Hier ist die Syntax korrekt, aber die Semantik **nicht eindeutig**.

Fazit:
Bei der Kommunikation zwischen Mensch und Maschine bzw. zwischen Maschinen untereinander braucht man sehr strikt definierte – gegenüber der Alltagssprache stark eingeschränkte – Sprachen, die ***keinerlei syntaktische Toleranz*** aufweisen.

In der Technik ist die Semantik einer Aussage ist erst nach korrekter syntaktischer Formulierung wahrnehmbar!

neue Herausforderung:

Wie können solche Regeln aussehen und wie kann man die zu einer Sprache gehörigen Wörter unterscheiden von Wörtern, die nicht zu dieser Sprache gehören? Hierzu gehört auch die Struktur der Sprache einzuhalten → siehe Java.

Ansatz:

Wir versuchen zunächst die bestehende formale Sprachen zu analysieren und formal korrekt zu beschreiben. Beispiel: → ***2 Präsentation E-Mail-Adresse***

Nichtterminale und Terminale

Symbole, die nicht weiter ersetzt werden müssen nennen wir Terminalsymbole oder kurz ***Terminale***.

Symbole, die noch weiterer Erklärung bedürfen und erst mittels weiterer Erklärungen ersetzt werden müssen (Platzhalter), nennen wir Nichtterminalsymbole oder ***Nichtterminale***.

Das erste Nichtterminal, bei dem mit der Erzeugung einer gültigen Zeichenkette begonnen wird, nennt man ***Startsymbol***.

Erweiterte Backus-Naur Form (EBNF)

Die ***erweiterte Backus-Naur Form*** ist eine kompakte Darstellungsform für die Struktur einer Sprache. Man verwendet Nichtterminale, Terminale und einige Metazeichen für Wiederholungen oder Alternativen. Alle Nichtterminale müssen dabei durch weitere Regeln erklärt werden bis nur noch Terminale übrig bleiben. Die Terminale bilden dann ein Wort der Sprache.

Folgende Zusatzzeichen stehen zur Verfügung:

- ***( ) runde Klammern*** zum „Bündeln“ (wie in der Mathematik)
- ***[ ] eckige Klammern*** für Optionen (ein Mal oder kein Mal)
- ***{ } geschweifte Klammern*** (kein Mal oder beliebig oft)
- ***| senkrechter Strich*** bedeutet „oder“

Beispiele:

1. Binärzahlen mit genau zwei Nullen =
   {1} 0 {1} 0 {1}
2. Binärzahlen mit mindestens zwei Nullen =
   {1} 0 {1} 0 {0|1}
3. Auto Kennzeichen (Kurzform ohne Benutzung von Nichtterminalen) =

(A|..|Z|Ä|Ö|Ü) [A|..|Z|Ä|Ö|Ü] [A|..|Z|Ä|Ö|Ü] "-" (A|..|Z|Ä|Ö|Ü) [A|..|Z|Ä|Ö|Ü] "-" (1|..|9) [0|..|9] [0|..|9] [0|..|9] [E|H]

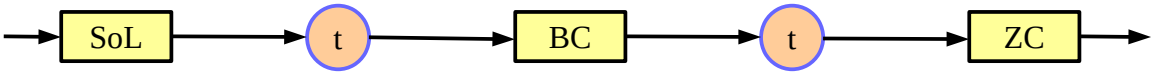
Terminale gehören eigentlich immer in Anführungszeichen. Aus Gründen der Übersichtlichkeit wurde das hier vermieden (außer beim Trennzeichen).

Syntaxdiagramme

Das Syntaxdiagramm kann als graphische Darstellung der EBNF gesehen werden. Beide Werkzeuge sind gleich mächtig und können leicht ineinander umgewandelt. Wir betrachten das Syntaxdiagramm anhand der Autokennzeichen (ohne E und H):

Autokennzeichen: Stadt oder Landkreis → Buchstabencode → Zahlencode

Autokennzeichen:



Terminale werden in einen **Kreis** gezeichnet.

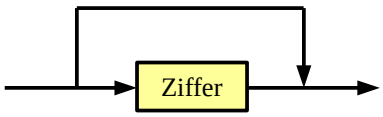
Nichtterminale werden in einem **Rechteck** gezeichnet.

Wiederholungen im Syntaxdiagramm darstellen

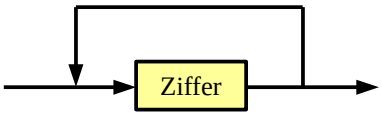
SoL → **ein Buchstabe** oder **zwei Buchstaben** oder **drei Buchstaben**

BC → **ein Buchstabe** oder **zwei Buchstaben**

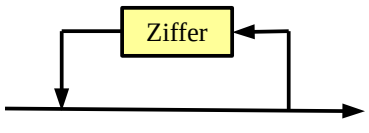
ZC → **eine Ziffer** oder **zwei Ziffern** oder **drei Ziffern** oder **vier Ziffern**



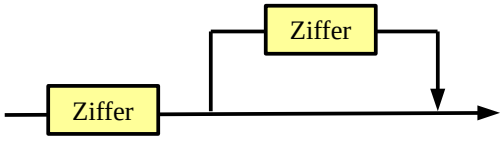
ein Mal oder kein Mal



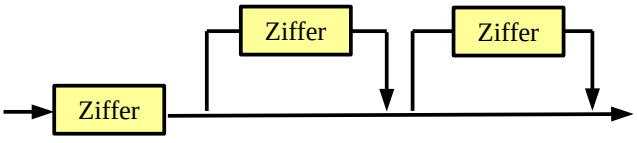
ein Mal oder beliebig oft



kein Mal oder beliebig oft



ein Mal oder zwei Mal



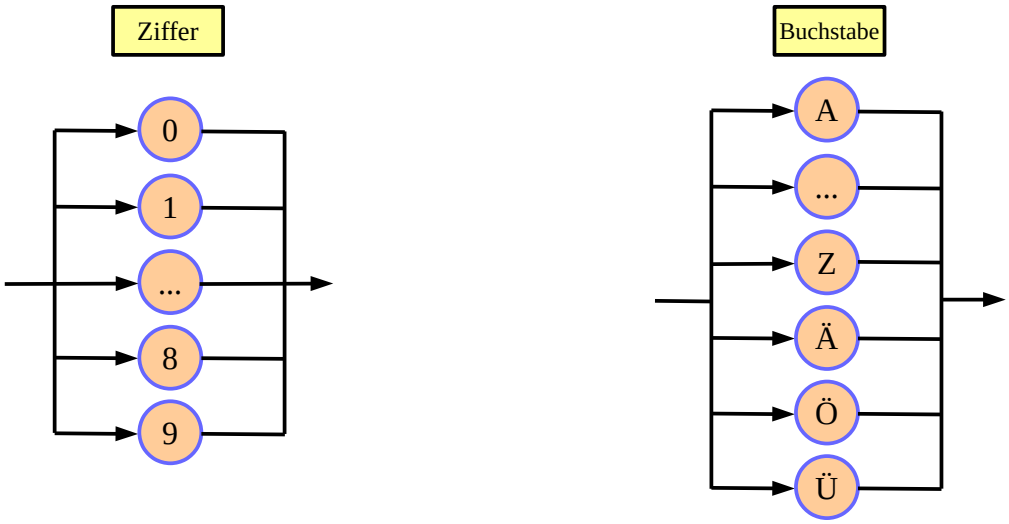
ein Mal oder zwei Mal oder drei Mal

Alternativen im Syntaxdiagramm darstellen

Buchstabe → **A** oder **B** oder **C** … oder **Z** oder **Ä** oder **Ö** oder **Ü**

Ziffer → **1** oder **2** oder **3** oder … oder **9** oder **0**

In den Erklärungen zu den fünf Nichtterminalen kommt sehr oft das Wort „oder“ vor. Solche Alternativen stellt man durch Verzweigungen dar, die auch wieder zusammenführen müssen.



Zusammenfassung Syntaxdiagramm

- Ein ***Syntaxdiagramm*** kann aus ***Terminalen*** und ***Nichtterminalen*** bestehen.
- Jedes **Nichtterminal** muss **in einem weiteren Syntaxdiagramm** in Terminale umgewandelt werden.
  Dieser Vorgang wird so lange wiederholt, bis es keine Nicht-Terminale mehr gibt.
- Die **Pfeile** geben an, in welche Richtung man das Syntax-Diagramm durchlaufen darf.
  Gegen die Pfeilrichtung zu laufen ist verboten!
- **Alternativen und Wiederholungen** realisiert man durch **Verzweigungen** bei denen sowohl weiter nach vorne als auch weiter zurück gegangen werden kann.
  Ein Syntaxdiagramm kann also „Loops“ enthalten.

Aufgabe 1: E-Mail-Adresse

Entwerfe ein Syntaxdiagramm für die formale Sprache der E-Mail-Adressen.

Aufgabe 2: Eurobeträge

Entwerfe ein Syntaxdiagramm, das Eurobeträge darstellt. Eurobeträge besitzen einen beliebig großen Eurobetrag, gefolgt von einem Komma und dem Centbetrag. Zum Schluss gibt es immer ein €-Zeichen. Der Betrag für die vollen Euros dürfen keine führenden Nullen besitzen, aber es darf natürlich ein reinen Centbetrag geben.

Gültige Wörter: 5,23€ 0,00€ 74353,72€ 0,50€ usw.

Stelle anschließend passend zu deinem Syntaxdiagramm die EBNF auf.

Grammatiken

Die bisher kennengelernten Modelle EBNF und Syntaxdiagramm dienen eher der Strukturierung von Wörtern einer Sprache. Grammatiken sind Modelle die eher zum Erzeugen „neuer“ Wörter einer Sprache geeignet sind.

Eine allgemeine ***Grammatik*** wird durch folgende vier Merkmale beschrieben (fachlich: Eine Grammatik ist ein 4-Tupel mit **G = (N, T, P, S)**):

- ***Nichtterminale***
- ***Terminale***
- ***Produktionsregeln***
- ***Startsymbol***

Nichtterminale und Terminale kennst du bereits.

Produktionsregeln geben an, wie Nichtterminale nach und nach in Terminale umgewandelt werden.

Das ***Startsymbol*** ist ein Nichtterminal, mit dem jede Anwendung der Grammatik beginnt. (erste Produktionsregel)

Produktionsregeln von (kontextfreien) Grammatiken

Eine ***(kontextfreie) Grammatik*** ist eine Grammatik mit speziellen Regeln für die Produktionen. Ein Nichtterminal darf ausschließlich folgendermaßen abgebildet werden:

- ***S → AbSaAa*** (S, A sind Nichtterminale und a,b sind Terminale.
  Linke Seite der Produktion: genau ein Nichtterminal
  Rechte Seite der Produktion: beliebige Kombinationen aus Nichtterminalen und Terminalen (auch ein einzelnes Nichtterminal bzw. Terminal)
- ***A → ε*** (***ε*** ist das Zeichen für das „***leere Wort***“, also für nichts)

Die Produktionsregeln einer Grammatik funktionieren analog zur EBNF mit dem

Unterschied, dass man ***keine Option oder Wiederholung*** verwenden darf. Damit wird auch die Gruppierung nicht mehr benötigt.

Beispiel für eine Grammatik

G = (N, T, P, S) mit N = {S, A, B}, T = {0, 1}, S = S und P =

S → A1A1A

A → 0A | ε

Aufgabe 3:

Ermittle 3 Wörter, die die obige Grammatik aufstellt.

Bestimmt die Sprache, die die obige Grammatik aufstellt.

Aufgabe 4:

Stelle für die folgenden Sprachen eine Grammatik auf der Webseite flaci.com auf. Hier müssen Terminale in ‘ ‘ Hochkommata geschrieben werden oder man lässt Leerzeichen zwischen den Zeichen weg. Zeichen die durch eine weitere Regel beschrieben werden, werden automatisch als Nichtterminale gekennzeichnet:

a) *Eurobeträge*

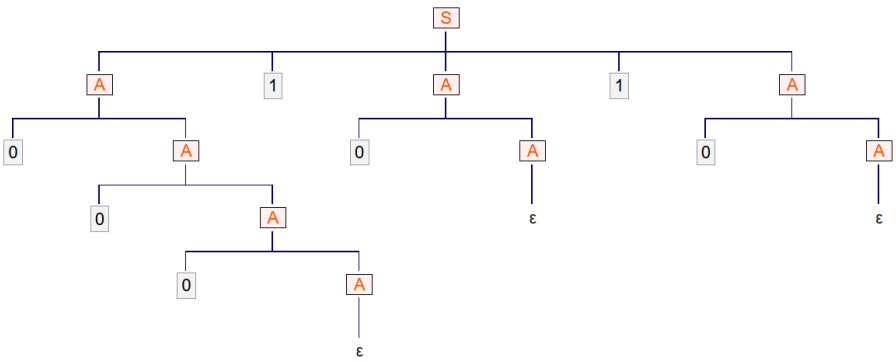
b) *Autokennzeichen*

c) *E-Mail-Adresse*

Lass die Grammatik einige Wörter ableiten und betrachte die Ableitungsbäume.

Ableitungsbäume

Unter einem ***Ableitungsbaum*** versteht man eine baumartige Darstellung aller Regeln, die nacheinander angewandt werden müssen, damit – beginnend mit der Startregel – ein bestimmtes Wort nach den Regeln einer Grammatik gebildet werden kann.



Die obenstehende Skizze zeigt den Ableitungsbaum für das Wort ***0001010*** der Beispielgrammatik.

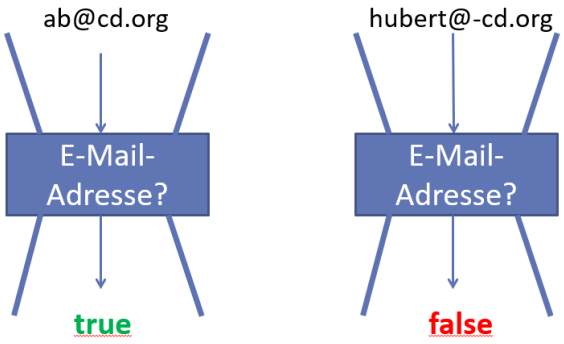
Schreibt man dies simpel in eine Zeile, so spricht man nur von einer ***Ableitung***, nicht von einem Ableitungsbaum.

S → A1A1A → 0A1A1A → 00A1A1A → 000A1A1A → 0001A1A → 00010A1A → 000101A → 0001010A → 0001010

Automaten (Deterministischer endlicher Automat)

Computer denken sich keine E-Mail-Adressen aus. Sie überprüfen nur, ob die Eingabe korrekt ist.

Für die Überprüfung, ob ein gegebenes Wort Teil einer Sprache ist, fehlt uns noch ein Modell: der Automat.



Ein ***deterministischer endlicher Automat*** ist ein Modell, das durch folgende fünf Bestandteile beschrieben (fachlich: Ein DEA ist ein 5-Tupel mit **$A = (Z, \Sigma, \delta, z_0, E)$**):das aus verschiedenen Bausteinen besteht. Es gibt

- ***Zustände Z***
  Ein Automat hat **endlich** viele Zustände. ***Genau einer*** davon ist sein ***Startzustand z_0***, in dem er immer seine Arbeit beginnt. Es muss auch ***mindestens einen Endzustand E*** geben. Nur wenn hier das Eingabewort endet, wird ein Wort akzeptiert (Teil der Sprache)
  Einen Zustand zeichnen wir immer als ***Kreis***.
- ***ein Alphabet Σ***
  Das ist die Menge an Zeichen, die zum Bauen eines Wortes verwendet werden dürfen → Terminale
- ***Übergänge δ***
  Der Automat kann durch das Einlesen eines weiteren Zeichens in einen anderen Zustand übergehen.
  Einen Übergang zeichnen wir immer als ***Pfeil***.
  Bewirkt ein Zeichen einen Zustandsübergang, dann wird das Zeichen am Pfeil notiert.

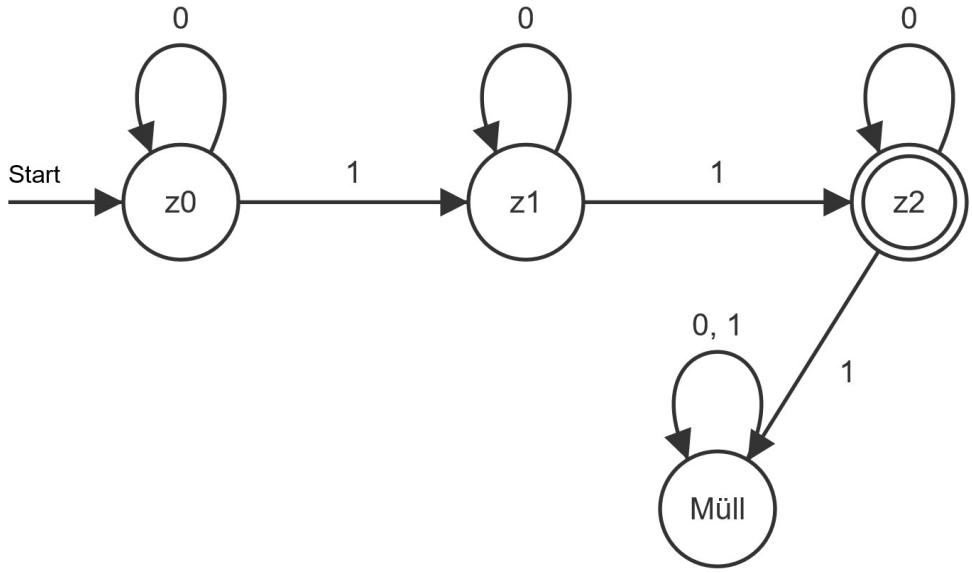
Beispielautomat

Der Beispielautomat A erkennt Binärzahlen, die ***genau zwei 1sen*** enthalten.

$A = (Z, \Sigma, \delta, z_0, E)$ mit

- ***$Z = \{ z_0, z_1, z_2,$ Müll $\}$***

- ***Σ = { 0, 1 }***
- ***δ als Zustandübergangsdiagramm:***



- $z_0 = z_0$
- $E = \{ z_2 \}$

Das Wort ***01100*** wird **akzeptiert**, weil der Automat nach Abarbeitung des Wortes in einem Endzustand stehen bleibt.

Konfigurationenfolge(n) für: 0 1 1 0 0

K1	0	1	2	3	4	5
▶	z0	z0	z1	z2	z2	z2
	01100	1100	100	00	0	

Das Wort ***1000*** wird **nicht akzeptiert**, weil der Automat nach Abarbeitung des Wortes nicht in einem Endzustand stehen bleibt.

Konfigurationenfolge(n) für: 1 0 0 0

K1	0	1	2	3	4
▶	z0	z1	z1	z1	z1
	1000	000	00	0	

Allgemein:

Ein ***DEA*** (***D***eterministischer ***E***ndlicher ***A***utomat) besteht aus ***endlich*** vielen Zuständen und ***akzeptiert*** ein Wort als „zu seiner Sprache gehörig“, wenn er nach Abarbeitung des Wortes in einem ***Endzustand*** stehen bleibt. Bleibt er nach Abarbeitung des Wortes in einem ***Nicht-Endzustand stehen***, so ***lehnt*** er dieses Wort ***ab***.

Bei einem ***deterministischen*** Automaten gibt es in jedem Zustand zu jedem Zeichen des Alphabets maximal einen Zustandsübergang.

Ein Automat heißt ***vollständig***, wenn es in jedem Zustand für jedes Zeichen des Alphabets mindestens einen Zustandsübergang gibt.

Aufgabe 5:

Stelle für die folgenden Sprachen einen Automaten auf der Webseite flaci.com auf.

a) *Eurobeträge*

b) *Autokennzeichen*

Allgemein:

Bei einem ***NEA*** (***N***ichtdeterministischen ***E***ndlichen ***A***utomaten) sind zu einem Zeichen mehrere Zustandsübergänge pro Zustand möglich. Ein ***NEA*** akzeptiert eine Eingabe, wenn es ***mindestens einen*** „Weg“ gibt, sodass er nach der Abarbeitung des Eingabewort in einem ***Endzustand*** steht.

Weitere Übungsaufgaben → ***3 Präsentation Übungen Zu DEAs und NEAs***

Übergangstabellen

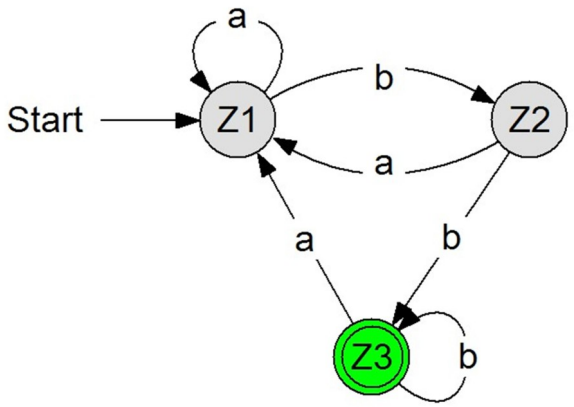
Ein Automat sieht grundlegend aus wie die Struktur eines Graphen. Die Kanten eines Graphen kann man durch eine Adjazenzmatrix darstellen uns so ist es logisch, dass sich auch die Übergänge unseres Automaten in Form von Tabellen angeben lassen.

Aufgabe 6:

a) *Erfasse die Struktur der nebenstehenden Übergangstabelle des Automaten der Binärzahlen mit genau zwei 1sen.*

b) *Bestimme die Sprache, die dieser Automat aufstellt.*

δ	0	1
Müll	Müll	Müll
z0	z0	z1
z1	z1	z2
z2	z2	Müll



c) *Stelle die Übergangstabelle zu diesem gegebenen Automaten auf*

Übergangstabelle → Code

Die ***Struktur der Tabellen*** kann man durch eine **verschachtelte bedingte Anweisung** direkt übernehmen:

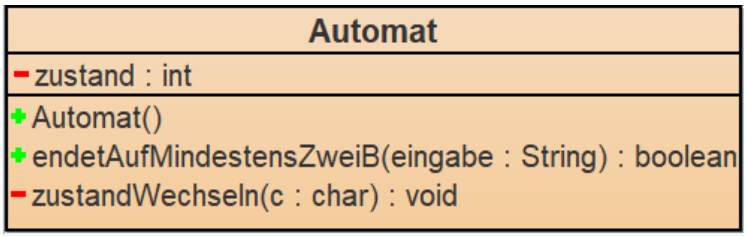
- Jeder **Zustand** entspricht einem **Fall in der äußeren Fallunterscheidung.**
- Jeder **Übergang** entspricht einem **Fall in einer inneren Fallunterscheidung.**

```
if  (zustand == 0)   {                          // erste Zeile/Zustand
        if (zeichen == 'c')   { zustand = 0; }          //erste Spalte
        else if (zeichen == 'a')   { zustand = 1; }     //zweite Spalte
        else if (zeichen == 'b')   { zustand = 2; }     //dritte Spalte
        else { zustand = -1; }                          //ungültiges Zeichen oder
                                                        //Übergang in den Müllzustand
}
else if  (zustand == 1)   {                     // zweite Zeile/Zustand
        …
}
…
```

Aufgabe 7:

*Programmiere den Automaten von **Aufgabe 6b**.*

Nutze hierfür die Methoden length() und charAt(…) der Klasse String, die Funktionsweise des Automaten zu imitieren. Siehe Java-API String



Ausblick formale Sprachen und Zusammenhänge

Unter dem Begriff ***formale Sprachen*** fasst man all diejenigen Sprachen zusammen, welche rein mathematisch (also formal) analysiert werden können. Unsere Alltagssprache gehört nicht zu den formalen Sprachen! Formale Sprachen können mit „Ungenauigkeiten“ (Dialekt, Akzent, Grammatik-Fehler, Rechtschreib-Fehler) nicht umgehen.

Bei einer Sprache unterscheiden wir zwischen ***Syntax*** (Sprachrichtigkeit) und ***Semantik*** (Was bedeutet das eigentlich?). Bei der Kommunikation zwischen Mensch und Maschine bzw. zwischen Maschinen untereinander ist eine formale Sprache notwendig. Dabei kann eine Semantikprüfung erst erfolgen, nachdem eine Syntaxprüfung fehlerfrei durchlaufen wurde.

Zur Darstellung der Struktur einer Sprache gibt es unterschiedliche Möglichkeiten:

- ***Automat*** (z.B. DEA – auch andere ...)
- ***Syntaxdiagramm***
- ***erweiterte Backus-Naur Form*** (***EBNF***)
- ***Grammatik*** (z.B. kontextfreie – auch andere)

Nicht alle Darstellungsarten sind gleich mächtig. Es gibt unterschiedliche Arten von Sprachen, die mit der einen Methode dargestellt werden können und mit einer anderen nicht.

Man kann formale Sprache je nach Komplexität in eine klare Hierarchie einteilen. Man nennt diese ***Chomsky-Hierarchie***

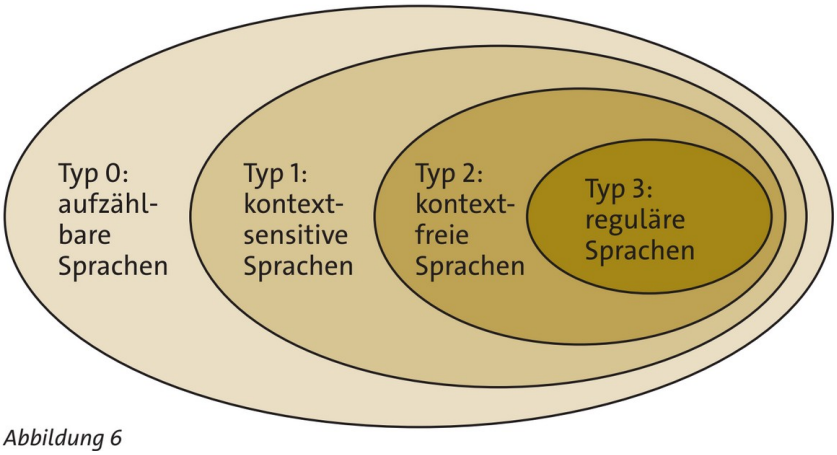


Abbildung 6

Sprache	Grammatik	Automat	Andere Möglichkeiten
regulär	Reguläre Grammatik Nur S→ aA	***DEA***	Regulärer Ausdruck
kontextfrei	***Kontextfreie Grammatik*** S→ aaAb; S→A1B rechte Seite: beliebige Kombinationen aus Nichtterminalen und Terminalen	Kellerautomat: kann sich gelesene Zeichen mit einem Stapel/ Stack/Keller merken	***Syntaxdiagramm, EBNF***
kontextsensitiv	Kontextsensitive Grammatik SA → AS linke Seite darf Kombinationen auf Nichtterminalen und Terminalen enthalten	Linear beschränkte Turingmaschine	
unbeschränkt semi-entscheidbar	Phrasenstrukturgrammatik DR→ R man darf die linke Seite verkürzen bzw. löschen	Turingmaschine	

- Die reguläre Grammatik, der regulärer Ausdruck und der DEA sind gleich mächtig. Sie bieten beide exakt dieselben Möglichkeiten eine reguläre Sprache zu beschreiben, können aber nicht kontextfreie Sprachen beschreiben z.B. korrekte Klammerung, d.h. es gibt genauso viele öffnende wie schließende Klammern.

- Das Klammerproblem kann aber mit Syntaxdiagramm, EBNF und kontextfreier Grammatik behandelt werden. Diese drei Darstellungsarten sind wiederum gleich mächtig, aber mächtiger als regulärer Ausdruck, reguläre Grammatik und DEA und können damit auch reguläre Sprachen beschreiben.
- Alle uns bekannten Formen versagen allerdings bei Sprachkonstrukten wie $a^n b^n c^n$, also allen Wörtern die n mal das Terminal a gefolgt von n Mal dem Terminal b gefolgt von n Mal dem Terminal c enthalten. Hierfür gibt es wiederum andere Arten von Automaten bzw. Grammatiken, die aber immer komplizierter werden.

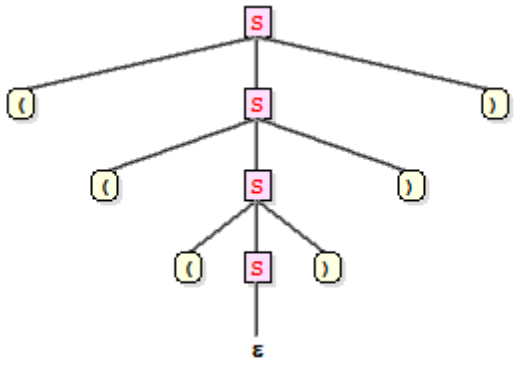
Grenzen des DEAs/Abgrenzung der regulären Sprache

Wie bereits erwähnt, kann man mit DEAs bestimmte Sprachkonstrukte – wie korrekte Klammerung – nicht erkennen. Allerdings bieten die meisten Programmiersprachen die Möglichkeit beliebig vieler Klammerebenen. Der Compiler (Übersetzer) erkennt, wenn die Anzahl der öffnenden Klammern nicht der Anzahl der schließenden Klammern entspricht.

Aufgabe 12:

*Stelle für die folgenden Sprachen entweder eine EBNF **oder** eine Grammatik **oder** ein Syntaxdiagramm auf.*

a) *vereinfachte Klammerung*
$L = \{ (^n)^n \mid n \in \mathbb{N} \}$, also erst alle öffnenden dann alle schließenden Klammern. Hilfestellung: siehe Ableitungsbaum



b) *alle Klammerausdrücke, z.B. **()(), (()(()))()***

c) ***Versuche einen DEA aufzustellen, der die (einfache) Klammersprache erkennt. Auf welches Problem stößt du beim Aufstellen?***

d) *Abituraufgabe 2021 IV 1c*

Zusatzaufgaben:

S. 46/T1