Formale Sprachen und Compiler

Überblick

Prof. Dr. Franz-Karl Schmatzer
schmatzf@dhbw-loerrach.de

Literatur

- C.Wagenknecht, M.Hielscher; Formale Sprachen, abstrakte Automaten und Compiler; 3.Aufl. Springer Vieweg 2022;
- A.V.Aho, M.S.Lam,R.Savi,J.D.Ullman, *Compiler – Prinzipien,Techniken und Werkzeuge*. 2. Aufl., Pearson Studium, 2008.
- Güting, Erwin; *Übersetzerbau –Techniken, Werkzeuge, Anwendungen*, Springer Verlag 1999
- Sipser M.; Introduction to the Theory of Computation; 2.Aufl.; Thomson Course Technology 2006
- Hopecroft, T. et al; Introduction to Automata Theory, Language, and Computation; 3. Aufl. Pearson Verlag 2006

Grundlegende Konzepte

- **Motivation**
- **Sprachprozessoren**
- **Grundbegriffe**
- **Formale Sprachen**

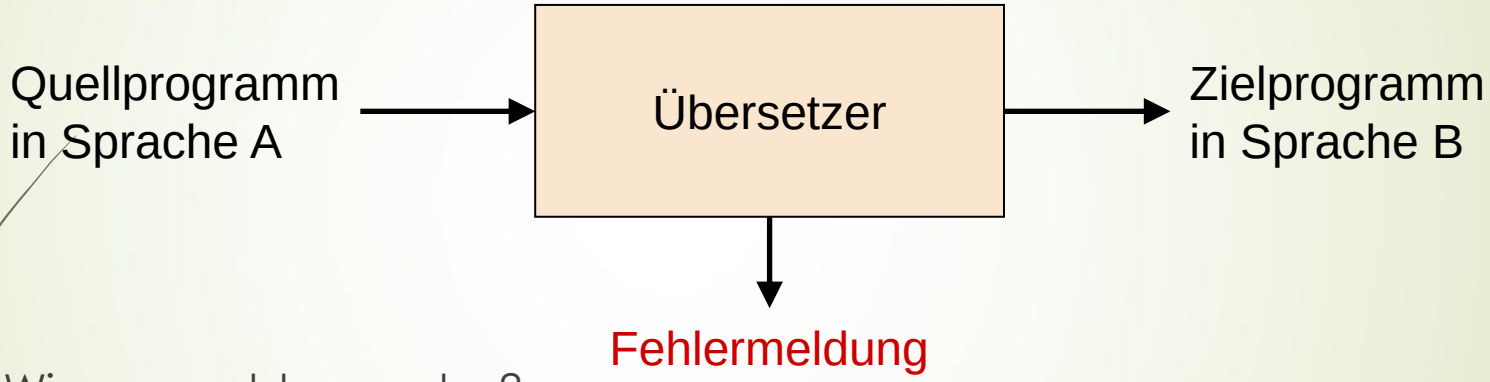
Motivation

Wieso werden heute noch formale Sprachen und Compiler gelehrt?

- Gehört zur Allgemeinbildung eines Informatikers wie
  - Datenbanken
  - Erlernen einer Programmiersprachen
  - Internet-Technologien
- Einzelne Techniken und Tools werden immer wieder verwendet:
  - Beschreibung des Verhaltens von Objekten
  - Entwickeln von Beschreibungssprachen (LaTex, HTML, SGML)
  - Datenbankanfragesprachen (SQL, XQuery)
  - VLSI Entwurfsprachen (Layout von Chips)
  - Entwickeln von Protokollen in verteilten Systemen
  - Entwickeln von Sprachen für spezielle Systeme

Sprachprozessoren
Einführung

- Bau eines Übersetzers (Compiler) für formale Sprachen im weitesten Sinnes. D.h. Das Übersetzen von einem Quellprogramm in ein Zielprogramm und der Ausgabe einer Fehlermeldung.



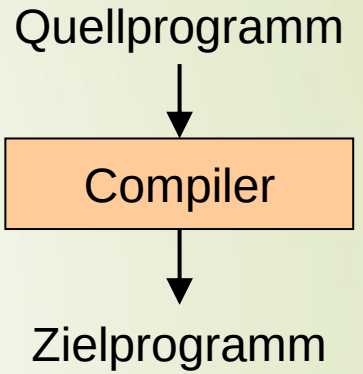
- Wieso macht man das?
  - Man kann etwas besser in Sprache A beschreiben, aber die Maschine versteht nur Sprache B, oder man hat nur eine Maschine die Sprache B versteht.
- Solche Systeme nennt man auch allgemein Sprachprozessoren
- Man unterscheidet i. W. 2 Typen von Sprachprozessoren
  - Compiler und Interpreter

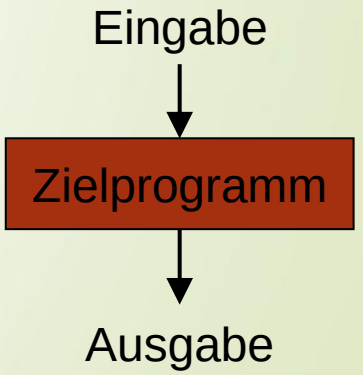
Aufgabe Sprachprozessoren

- Erläutern Sie die Funktion eines Compilers und eines Interpreters.
- Wo werden diese eingesetzt?
- Was sind die Vor- und Nachteile dieser Systeme?
- Wie realisiert Java die Übersetzung von Quellcode in Maschinencode.

Sprachprozessoren

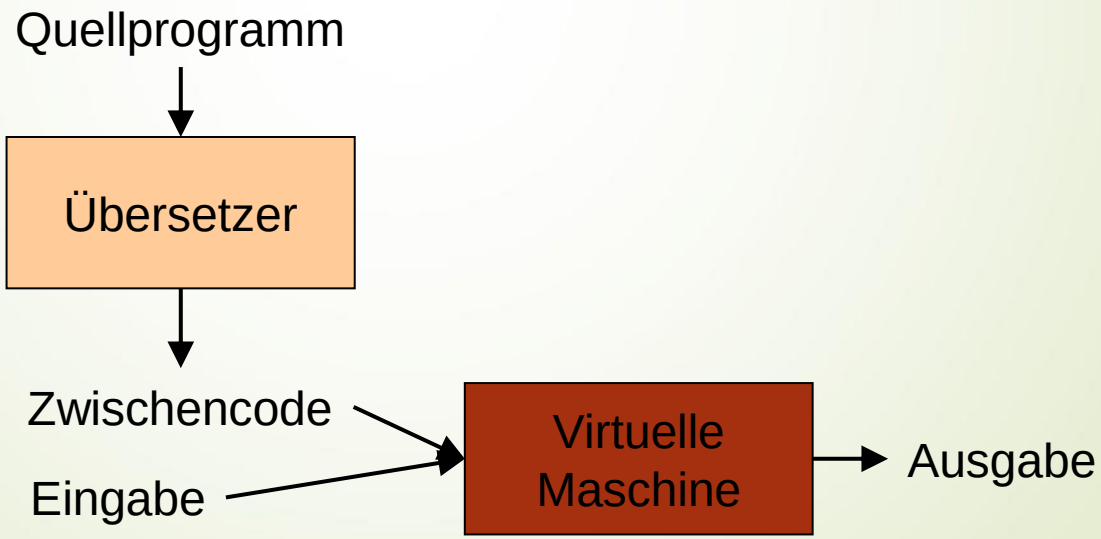
- Was ist ein Compiler?
  - Übersetzer eines Programms von einer Quellsprache in eine Zielsprache
  - Wichtige Rolle des Compilers ist eine Fehleranalyse
  - Ist die Zielsprache ein ausführbares Programm in Maschinensprache, kann es anschließend direkt aufgerufen werden.
- Was ist ein Interpreter?
  - ein andere Form eines Sprachprozessor, der die Operationen direkt ausführt, ohne ein Zielprogramm zu erstellen.
- Compilergenerierte Programme sind normalerweise viel schneller als eine Ausführung durch einen Interpreter
- Aber eine Fehleranalyse eines Interpreters ist meistens besser.





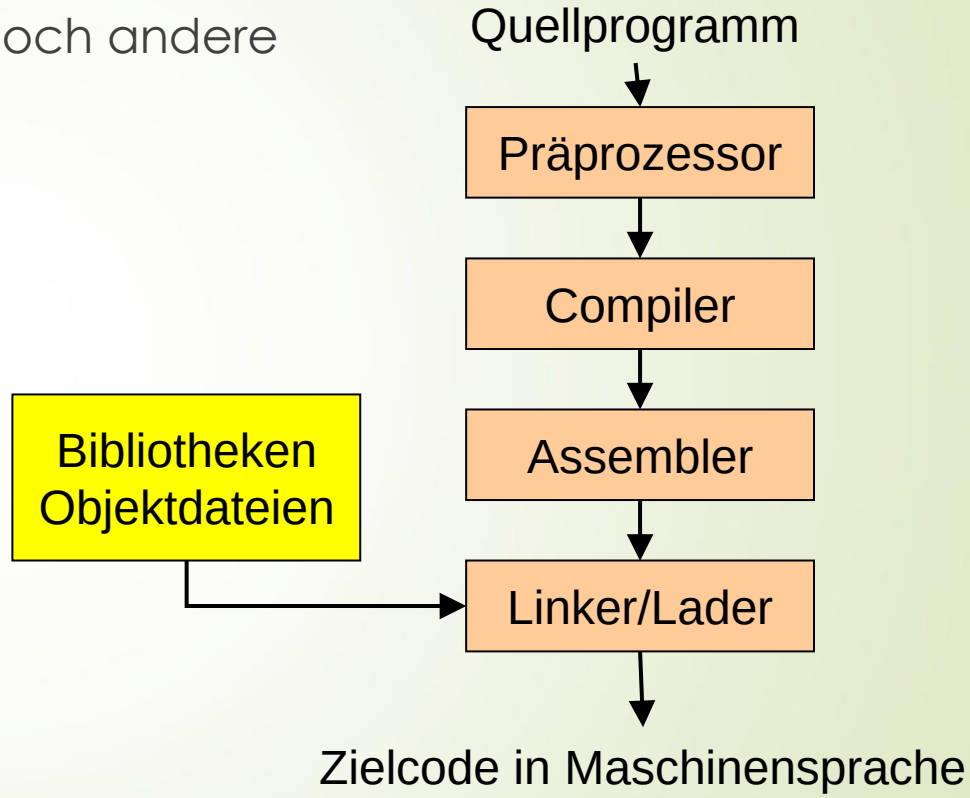
Sprachprozessoren
Beispiel 1

- Java Sprachprozessoren
  - kombinieren Kompilierung und Interpretation
  - Es wird eine Zwischenform , sogenannter Bytecode, erzeugt.
  - Der Bytecode wird von der virtuellen Maschine interpretiert



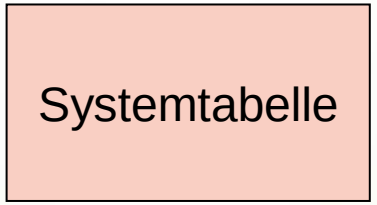
Sprachprozessoren
Übersetzungsprozess Compiler

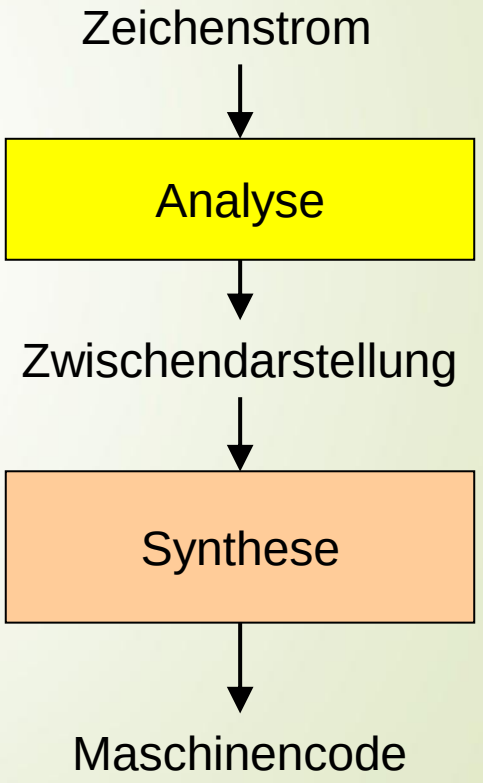
- Neben dem Compiler können noch andere Programme benötigt werden
  - Aufspalten in Module
  - Verwenden eines Präprozessors
  - Assembler
  - Linker/Lader



Sprachprozessoren
Struktur eines Compilers

- Ein Compiler teilt man in 2 große Blöcke
  - Analyse und
  - Synthese
  - mit einer Systemtabelle





Grundbegriffe

- Alphabet und Zeichen
- Worte, Wortlänge und Verkettung
- Wortmenge
- Die Sprache

Alphabet und Zeichen

- Ein Alphabet ist eine beliebige endliche, nichtleere Menge. Die Elemente dieser Menge heißen Zeichen.
- Beispiel:
  - A_1 = {a,b,c,d,…,z}
  - A_2 = {(,),[,],+,+,*,/,a}
  - A_3 = {begin, end, for, while, do ,repeat, until}

Worte, Wortlänge und Verkettung

- Irgendeine auch eine leere Zeichenmenge ist ein Wort.
- Man sagt:
  - eine Zeichenkette „w“ ist ein Wort über dem Alphabet Σ, wenn sämtliche Zeichen von w aus Σ stammen.
- Das Symbol für das leere Wort ist ε.
- Die Länge eines Wortes w, kurz |w| ist bestimmt durch die Anzahl aller Zeichen, die das Wort w enthält.
- Die Verkettung • von zwei Zeichen ergibt wieder ein Wort.
  - Σ ={a,b,c} dann w=a•b = ab ist ein Wort.
  - Verkettung zwei Worte w_1 = ab mit w_2 = bc

    w3 = w_1 • w_2 = abbc

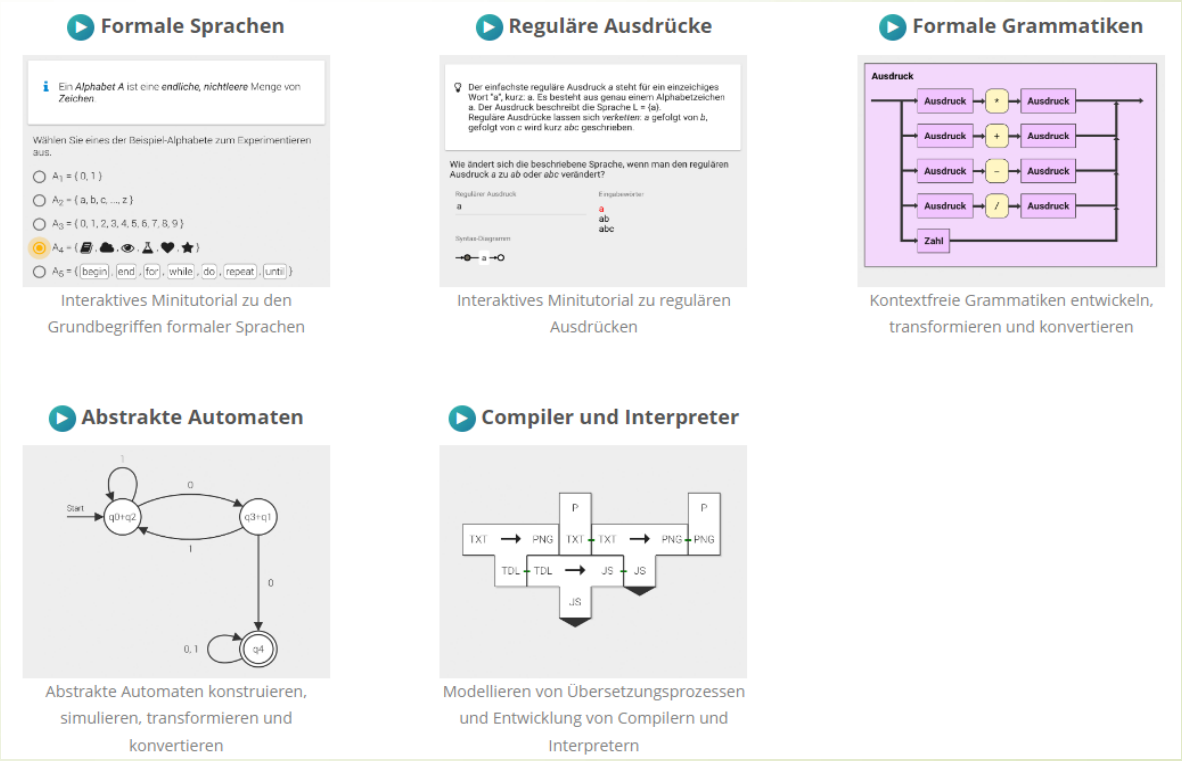
Tools zu Sprachen/Automaten

- Nutzen Sie die Tools der Webseite AtoCC
  - Es erlaubt Sprachen zu untersuchen, Automaten und Grammatiken aufzubauen und zu simulieren
  - Zugang https://atocc.de/cgi-bin/atocc/site.cgi?lang=de&site=main

Toolbox FLACI

(Formale Sprachen, abstrakte Automaten, Compiler und Interpreter)

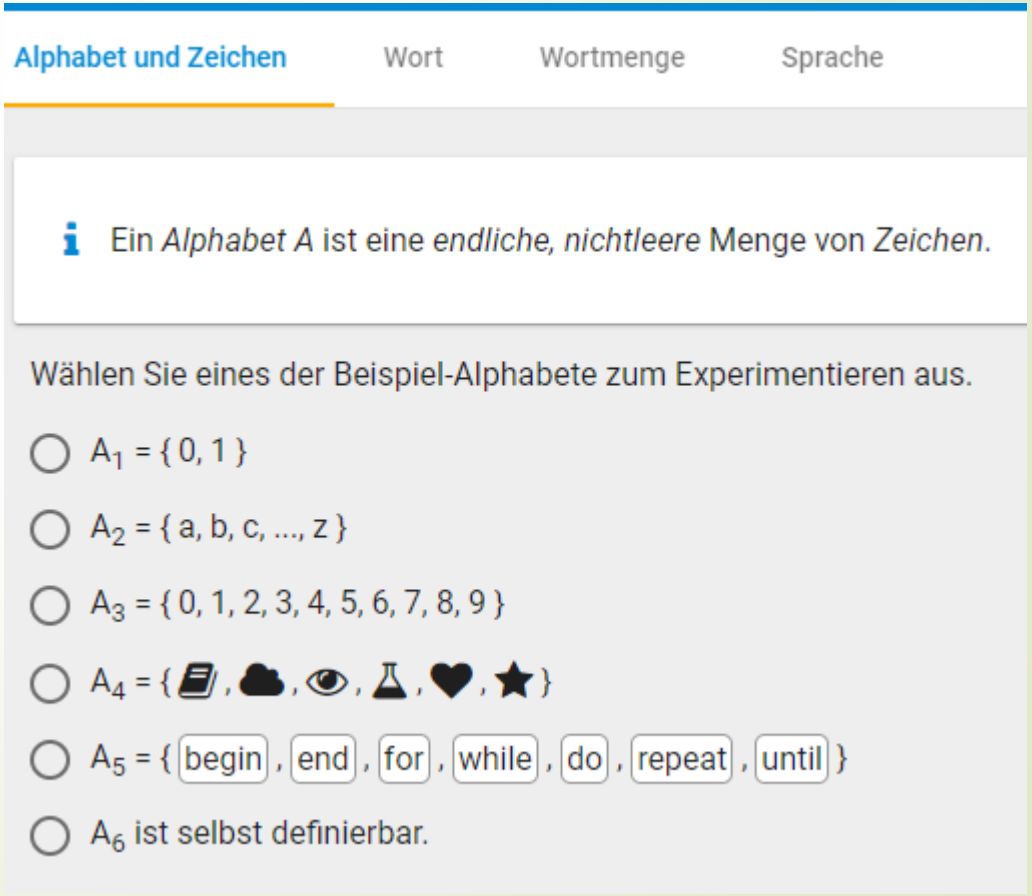
- Zugang https://flaci.com/home/
- Das Tool „Formale Sprachen“ erlaubt Sprachen, Worte und Wortmengen aufzubauen und zu untersuchen.
- Rufen Sie das Tool „ Formale Sprachen“ auf und melden Sie sich an dem System an.



Toolbox FLACI

Formale Sprachen

- Hier können sie bestehende Alphabete nutzen oder eigene Alphabete erstellen.
- Die Worte, Wortmengen und die Sprache untersuchen.



Aufgabe Zeichenmenge

Wortmengen

- Erstellen Sie für folgende Zeichenmenge alle Wörter der Länge 1,2,3
  - A_1 = {|} „Bierdeckel“
  - A_2 = {01,11}
  - A_3 = {a , be, c}
- Geben Sie Worte an, die nicht zur Menge gehören.
- Wie viele Worte können Sie für A_1, A_2 der Länge k bilden.

Lösung A_1

- Alphabet
- Worte bis mindestens zur Länge 3

Alphabet A = { 1 }

A^* = {

"", "1", "11", "111", "1111", "11111", .

}

WEITERE ELEMENTE ANZEIGEN

Lösung A_2

- Alphabet

Alphabet A = { 01 , 11 }

Wort w = "" = ε

- Worte bis mindestens zur Länge 3

Alphabet A = { 01 , 11 }

Stellen Sie eine Sprache L durch Anklicken einzelner *Wörter* aus A^* zusammen.

A^* = {
"", "01", "11", "0101", "0111", "1101", "1111",
"010101", "010111", "011101", "011111",
"110101", "110111", "111101", "111111",
"01010101", ...
}

L = {
}

Lösung A_3

- Alphabet

Klicken Sie wiederholt auf Zeichen des Alphabets *A*, um ein Wort *w* zu konstruieren.

Alphabet A = { a , be , c }

Wort w = "" = ε

- Worte bis mindestens zur Länge 3

A^* = {
"", "a", "be", "c", "aa", "abe", "ac", "bea", "bebe",
"bec", "ca", "cbe", "cc", "aaa", "aabe", "aac",
"abea", "abebe", "abec", "aca", "acbe", "acc",
"beaa", "beabe", "beac", "bebea", "bebebe",
"bebec", "beca", "becbe", "becc", "caa",
"cabe", "cac", "cbea", "cbebe", "cbec", "cca",
"ccbe", "ccc", "aaaa", ...
}

L = {
}

Wortmenge

Σ und Σ*

- Die Menge aller Wörter über Σ nennt man die Wortmenge Σ*. Das leere Wort ε gehört auch dazu.
- Das Eingabealphabet Σ

  Σ = $\{e_1,..,e_n\}$ eine nicht leere Menge von Zeichen

  z.B. {0,1} oder {a,b,c}
- Worte
  - Endliche Zeichenfolge die aus dem Eingabealphabet gebildet werden können.

  z.B w = 0101101101 oder w = abbabccbacbb
- Σ* : = die Menge aller Wörter, die über das Alphabet gebildet werden können.

Wortmenge

Formale Definition von Σ^*

Formale Definition von Σ^* (rekursiv definiert)

1. Das leere Wort ε gehört zu Σ^*, d.h. $\varepsilon \in \Sigma^*$
2. Jeder Buchstabe $e \in \Sigma$ ist in Σ^*, d.h. $e \in \Sigma^*$
3. Sei $v,w \in \Sigma^*$ dann ist auch $vw \in \Sigma^*$

   (Konkatenierung von v mit w)

- Beispiel:

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

  $\Sigma^* = \{\varepsilon, a , b, aa, ab, ba, bb, aaa, \ldots\ldots\}$

Sprache L(A)

- Sei Σ ein Alphabet. Jede Teilmenge $L \subseteq \Sigma^*$ heißt Sprache über Σ.
- Eine Sprache besteht aus Wörter
  - $L_1 = \varnothing$ oder $L_2 = \Sigma^*$ sind Sprachen
  - $L_3 = \{ w \in N_0 \mid \text{mod}(w/2) = 0 \}$
- Sprachen können endlich viele oder auch unendlich viele Worte enthalten
  - $L_3 = \{0,1,2,3,4,5\}$
  - $L_4 = \{ w \in N_0 \mid \text{Sqrt}(w) \in N_0 \}$

Aufgabe Sprachen

- Geben Sie für folgende Sprachen das Alphabet und die Sprache mathematisch korrekt an.

1. L sei die Sprache, die bei Übertragung von Bytes nur Worte mit gerader Parität erzeugen.
2. L sei die Sprache, die bei Übertragung von Bytes nur Worte mit genauso viele 0- als auch 1-Zeichen erzeugen.
3. L sei die Sprache, die nur aus Quadratzahlen besteht.
4. L sei die Sprache, die ganze Zahlen kennzeichnet.

Lösung Aufgabe Sprachen

- L sei die Sprache, die bei Übertragung von Bytes nur Worte mit gerader Parität erzeugen.

  $\Sigma = \{0,1\}$ $L = \{w \in \Sigma^* \mid |w| = 8 \text{ und } |w|_1 \bmod 2 = 0\}$

  Bem: $|w|_1$ = Anzahl der 1-Zeichen im Wort.

- L sei die Sprache, die bei Übertragung von Bytes nur Worte mit genauso viele 0- als auch 1-Zeichen erzeugen.

  $\Sigma = \{0,1\}$ $L = \{w \in \Sigma^* \mid |w| = 8 \text{ und } |w|_1 = |w|_0\}$

  Bem: $|w|_0$ = Anzahl der 0-Zeichen im Wort.

- L sei die Sprache, die nur aus Quadratzahlen besteht.

  $\Sigma = N_0$ mit $L = \{ w \in N_0 \mid \text{Sqrt}(w) \in N_0 \}$

- L sei die Sprache, die ganze Zahlen kennzeichnet.

  $\Sigma = N_0$ mit $L = \{ w \mid w = w' \text{ oder } w = -w' \text{ und } w' \in N_0 \}$

Formale Definition von Sprachen
Grammatiken

Grammatiken werden formal definieren als:

- Eine Grammatik G besteht aus 4 Komponenten (N, $\sum$, P,S) mit:
  - N eine endliche Menge von Variablen (Nichtterminale).
  - $\sum$ ist ein Alphabet aus Terminalen mit $N \cap \sum = \varnothing$. Statt $\sum$ verwendet man oft T (**T**erminalen) für das Alphabet.
  - P ist eine Menge von Produktionen (Regeln).
  - Eine Produktion ist eine Element einer Relation P
    - $P=(L,R) \in (N \cup \sum)^* N (N \cup \sum)^* \times (N \cup \sum)^*$
    - Mit $l \in L$ und $r \in R$
    - Man schreibt statt (l, r) besser: $l \rightarrow_P r$ bzw. $l \rightarrow r$
  - $S \in N$ ist eine Startvariable

Beispiel einer Grammatik I

Grammatik G = (N,T,P,s) mit

N = {S, A, B}

T = {a, b, c}

{S → AS | ccSb, cS → a, AS → Sbb, cSb → c}

s = S

- Es gibt Nicht Terminale N
- Terminale T
- Produktionsregeln und Startzustand

Beispiel einer Grammatik II

Grammatik G = (N,T,P,s) mit

N = {S, A, B}

T = {a,b,c}

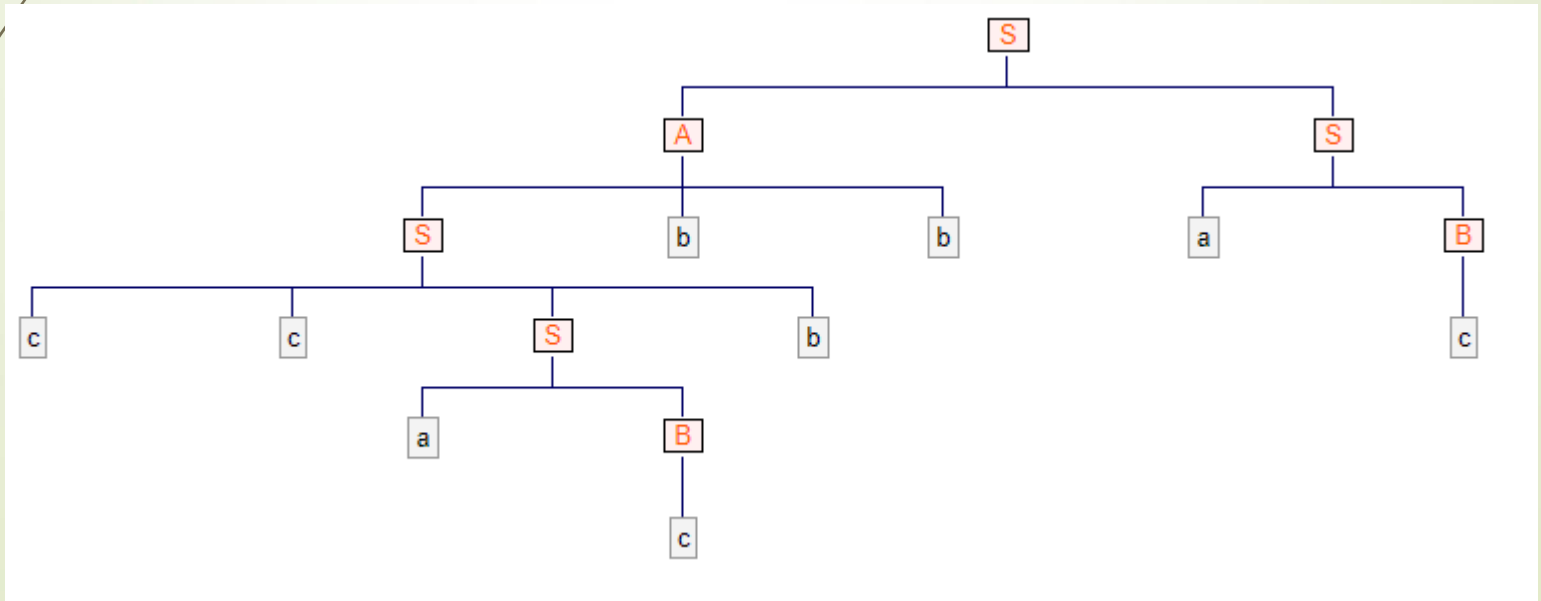
P = {S → AS | ccSb, S → aB, A → Sbb, B → c}

s = S

- Diese Grammatik besitzt eine besondere Eigenschaft:
- Sämtliche linke Regelseiten bestehen aus genau einem Nichtterminal.
- Grammatiken dieser Bauart nennt man kontextfreie Grammatiken, kurz: kfG.
- Erzeugen Sie ein Wort mit der Sprache?

Das kleinste Wort der Sprache

- Wort: ccacbbbac
- S-> AS-> SbbS-> ccSbbbS-> ccacbbbS-> ccacbbbaB-> ccacbbbac



Grammatiken

Formale Einteilung

- CHOMSKY hat 1956 die formalen Grammatiken in 4 Klassen (Typen) eingeteilt und hat damit hierarchische Sprachklassen beschrieben.
- Auf den ersten Blick sieht das recht willkürlich aus. Diese Typisierung ist für die Theorie der formalen Sprachen und die Automatentheorie fundamental.
- Das klassifizierende Merkmal dieser Typen ist die Regelgestalt.
- Regeln sind von der Form:

$$\alpha \rightarrow \beta \mid \alpha \in (N \cup T)^* \setminus T^* \text{ und } \beta \in (N \cup T)^*$$

Grammatiken

Formale Einteilung

Typ	Klas-sen	definiert	Regelgestalt
0	uG uS	unbeschränkt	Keine Einschränkung
1	ksG ksS	kontextsensitiv	wie Typ 0 und zusätzlich: ¨ $\lvert\alpha\rvert \leq \lvert\beta\rvert$ (langenmonoton) ¨ Ausnahme: $s \rightarrow \varepsilon$ zulässig, wenn ¨ s in keiner Regel auf der rechten Seite steht.
2	kfG kfS	kontextfrei	wie Typ 1 und zusätzlich: $\alpha \in N$ Ausnahme: Regeln der Form $\alpha \rightarrow \varepsilon$ zulässig
3	rG rS	regulär	wie Typ 2 und zusätzlich: ¨ $\lvert\beta\rvert \leq 2$, genauer: Entweder $\alpha \rightarrow x$ und $\alpha \rightarrow Ax$ (linkslinear) oder $\alpha \rightarrow x$ und $\alpha \rightarrow xA$ (rechtslinear) mit $x \in T$ und $A \in N$

Grammatiken

Formale Einteilung

Sprachklasse	definiert	Name der Klasse
L_3	{L(G) | G ist regulär}	Regulär, Typ 3
L_2	{L(G) | G ist kontextfrei}	Kontextfrei, Typ 2
L_1	{L(G) | G ist kontextsensitiv}	Kontextsensitiv, Typ 1
	{L(G) | G ist beschränkt}	
L_0	{L(G) | G ist eine Grammatik}	Rekursiv aufzählbar, Typ 0
L	{$L \subseteq T^*$ | T ist ein Alphabet}	Sprache

Chomsky-Hierarchie

Es gilt: $L_3 \subset L_2 \subset L_1 \subset L_0 \subset L$

d.h. jeder der Sprachen L_i ist eine echte Obermenge zu der nächsten Sprache L_{i+1}

ε -Sonderregel

- Eine ε-Regel ist für kontextsensitive, kontextfreie und reguläre Grammatiken problematisch.
  1. Sie verstoßen gegen die Längenmonotonie.
  2. Die rechte Seite (β) soll mindestens genauso lang sein, wie die linke (α) Seite, also $|\alpha| \leq |\beta|$.
- Ohne ε-Regel wäre es nicht möglich, das leere Wort abzuleiten.
- Für Sprachen vom Typ 1, 2 und 3 ist das aber erforderlich.
- Abhilfe schafft folgender Satz:

- **Zu jeder ε-freien ksG G = (N,T,P,s) gibt es eine äquivalente ksG G‘ = (N‘,T‘ ,P‘ ,s‘) und mit L(G‘) = L(G)∪ {ε}**

Beweis

- Die erforderliche Grammatiktransformation für kfG und analog für ksG geschieht folgen dermaßen:

1. Wähle ein noch nicht in N vorhandenes Nichtterminal s‘ als Spitzensymbol bzw. Startsymbol von G .
2. Ergänze die Regeln s‘ → s und s‘ → ε.

Aus G entsteht die Grammatik G‘ = (N ∪ {s‘},T,P ∪ {s‘ → s | ε},s‘)

Vorsicht!

Typ 3 Grammatiken sind problematisch da eine Regel s‘→ s nicht erlaubt ist. Folgende Transformation ist zielführend:

1. Wähle ein noch nicht in N vorhandenes Nichtterminal s‘ als Spitzensymbol von G‘
2. Ergänze für alle Regeln s → β in P die Regeln

   s‘→ β sowie s‘ → ε.

Diese Transformation ändert den jeweilige Typ der Grammatik nicht.

Aufgabe zu ε -Regel

- Geben
  - $G_1 = (\{S,A\},\{0,1,2\},\{S \rightarrow 0A,\ A \rightarrow 2S \mid 1\},S)$
  - $G_2 = (\{S,A\},\{0,1,2\},\{S \rightarrow 0S \mid 1A,\ A \rightarrow 1A \mid 2\},S)$
- Fügen Sie einfach die Regel $S \rightarrow \varepsilon$ hinzu. Sind dann die neue Grammatiken G‘ mit der alten Grammatik G identisch?
- Falls nicht, wandeln Sie die Grammatik G in Grammatik G‘ um.

Lösung Aufgabe zu ε -Regel

G = ({S,A},{0,1,2},{S → 0A, A → 2S | 1},S)

- Da S‘ → S für reguläre Grammatiken nicht erlaubt ist, nehmen wir einfach S → ε hinzu. Erhalten wir dann bis auf das leere Wort die gleiche Grammatik?

Neue Grammatik mit leerem Wort:

G‘ = ({S,A},{0,1,2},{S → 0A, A → 2S | 1, S → ε },S)

- G und G‘ sind außer dem leerem Wort nicht identisch.
- w = 02 gehört zu G‘ aber nicht zu G.
- Richtige Umsetzung

G‘ = (N ∪ {s‘},T, P ∪ {s‘ → 0A | ε}, s‘)

Lösung Aufgabe zu ε -Regel

G = ({S,A},{0,1,2},{S → 0S |1A, A → 1A | 2},S)

- Da S‘ → S für reguläre Grammatiken nicht erlaubt ist, nehmen wir einfach S → ε hinzu. Erhalten wir dann bis auf das leere Wort die gleiche Grammatik?

Neue Grammatik mit leerem Wort:

G‘ = ({S,A},{0,1,2},{S → 0S | 1A , A → 1A | 2, S → ε },S)

- G und G‘ sind außer dem leerem Wort nicht identisch.
- w = 0 gehört zu G‘ aber nicht zu G.
- Richtige Umsetzung

G‘ = (N ∪ {s‘},T, P ∪ {s‘ → 0S | ε}, s‘)

ε -Sonderregel

- Wie kann man für eine kfG ε-Freiheit herstellen?
- KfG mit ε-freien Regeln werden oft benötigt.
- Es gilt aber der Satz:

Zu jeder kfG G = (N,T,P,s) mit ε-Regeln der Form $A \rightarrow \beta$, mit $A \in N$ und $\beta \in (N \cup T)^*$, gibt es eine äquivalente kfG $G' = (N', T', P', s')$ ohne ε-Regeln (ggf. bis auf $s \rightarrow \varepsilon$).

Beweis

- Ein konstruktiver Beweis:
- Alle möglichen ε-Ersetzungen in den betreffenden Produktionen ausführen, so dass sich die ε-freier Regeln erübrigen.
  - Hierfür müssen zunächst alle Nichtterminale $A_i \in N$ bestimmt werden, die in beliebig vielen Schritten zu ε abgeleitet werden können.
  - Beginne mit $N_\varepsilon = \{A_i\}$, mit $A_i \rightarrow \varepsilon$ in P.
  - Ergänze im nächsten Schritt A in N_ε , wenn $A \rightarrow A_1 A_2 \ldots A_k$ in P, wobei $k \geq 1$, $A_i \in N$ und für alle A_i $(1 \leq i \leq k)$ gilt $A_i \in N_\varepsilon$.
  - Das Verfahren stoppt, wenn sich im jeweils nächsten Schritt keine weitere Veränderung in N_ε ergibt.
  - Da N endlich ist, terminiert das Verfahren.
  - Anschließend entferne alle Regeln der Gestalt $A_i \rightarrow \varepsilon$ aus P.
  - Für jede Regel $B \rightarrow \beta A_i \gamma$ in P, mit $A_i \Rightarrow^* \varepsilon$, d.h. $A_i \in N_\varepsilon$, ergänze $B \rightarrow \beta\ \gamma$.
  - β und γ sind Satzformen, von denen höchstens eine das leere Wort bezeichnet. Die ursprünglichen Regeln $B \rightarrow \beta A_i \gamma$ in P bleiben erhalten

Beispiel Transformation

- Gegeben

  $G1 = (\{X,B,K\},\{a,c\},\{X \rightarrow aB,\ B \rightarrow cB \mid K,\ K \rightarrow a \mid \varepsilon\},X)$.

- Die zu G1 äquivalente Grammatik G 1 ohne ε-Regeln ergibt sich nach der Transformation:
- $G'1 = (N', T', P', s')$, mit $N' = N = \{X,B,K\}$, $T' = T = \{a, c\}$, $P' = \{X \rightarrow aB \mid a,\ B \rightarrow cB \mid c \mid K,\ K \rightarrow a\}$, $s' = s = X$
- Führen Sie die Umsetzung durch:
- Aus welchen Elementen besteht die Menge N_ε?

Das Wortproblem

- Das (allgemeine) Wortproblem besteht aus der Frage nach der Existenz eines allgemeingültigen Entscheidungsverfahrens, das für jedes beliebige Wort w und jede beliebige Grammatik G in endlicher Zeit feststellt, ob entweder $w \in L(G)$ oder $w \notin L(G)$
- Man beginnt mit dem Startsymbol und leitet alle mögliche Satzformen ab.
- Die betrachtete Satzform besteht ausschließlich aus Terminalen und stimmt mit w überein. Diese Satzform wird nicht in S_{i+1} übernommen.
- Wenn die betrachtete Satzform eine Länge besitzt, die größer als $n = |w|$ ist, wird sie nicht in S_{i+1} (Längenmonotonie).
- Die betrachtete Satzform besteht ausschließlich aus Terminalen und stimmt mit w überein. Das Entscheidungsverfahren antwortet mit true und wird beendet.

Das Wortproblem

- Das Verfahren beginnt mit S0 = {s}, wobei s das Spitzensymbol der betrachteten Grammatik bezeichnet, und endet
  - entweder wenn $w \in S_k$, dann ist die Ausgabe des Algorithmus true, s.o.,
  - oder wenn $S_{k+1} = S_k$, d.h., es sind keine weiteren Satzformen ableitbar. Dann lautet die Ausgabe des Algorithmus false.

Beispiel

- Gegeben:

G = ({A,B}, {a,b,c}, P, A) mit

P = {A → aABc | aBc, cB → Bc, aB → ab, bB → bb}

Das Wort sei w = aabbcc

S0 = {A}

S1 = {A, aABc, aBc}

S2 = {A, aABc, aBc, aaABcBc, aaBcBc, abc}

w = aaABcBc und w=abc streichen. Erfüllen die Bedingung w=6 nicht.

S2' = {A, aABc, aBc, aaBcBc}

S3 = {A, aABc, aBc, aaBcBc, aabcBc,aaBBcc},

S4 = {A, aABc, aBc, aaBcBc, aabcBc, aaBBcc, aabBcc}

S5 = {A,aABc,aBc,aaBcBc,aabcBc,aaBBcc,aabBcc,aabbcc}

Stopp, da aabbcc ∈ S5.

Aufgabe

- Zeigen Sie, dass das Wort w=acb nicht in L(G), mit

G = ({A,B},{a,b,c},P,A) und

P = {A → aABc | aBc, cB → Bc, aB → ab, bB → bb} enthalten ist

Das Wortproblem

- Das Wortproblem ist für Typ-1,2,3-Sprachen allgemein entscheidbar, jedoch nicht für Sprachen vom Typ 0.
- Beispiel Typ-0-Sprache

N = {A,B,C},

T = { a, b},

P = { A → aBC | aA, aB → bCBa, CBaC → a },

s = A

Das Wort w = aba gehört zur Sprache, lässt sich aber nicht mit dem vorherigen Algorithmus ableiten.

A ⇒ aA ⇒ aaBC ⇒ abCBaC ⇒ aba