

# 1 Begriffe

**Alphabet**  $\Sigma = \{\dots\}$  ist die Menge von Zeichen, aus welchen die Wörter der definierten Sprache aufgebaut werden.

**Menge aller Wörter eines Alphabets**  $\Sigma^*$  ist die Menge aller aus dem Alphabet  $\Sigma$  gebauten Wörter. Beispiel:  $\Sigma = \{a, b\}$

$$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, \dots\}$$

**Wort**  $w \in \Sigma^*$  ist eine Kombination aus Zeichen aus dem Alphabet.

**Leeres Wort**  $\varepsilon \in \Sigma^*, |\varepsilon| = 0$  ist das Wort mit der Länge 0.

**Sprache**  $L(A)$  ist die Sprache, welche von einem Automaten akzeptiert wird. Man sagt  $A$  akzeptiert  $L$ .

**regulär** Eine Sprache ist dann regulär, wenn es einen DEA gibt, der sie akzeptiert.

**DEA** Deterministischer Endlicher Automat

**deterministisch** Der Determinismus ist die Auffassung, dass zukünftige Ereignisse durch Vorbereidungen eindeutig festgelegt sind (für jedes Zeichen ist in jedem Zustand ein Übergang vorhanden).

**nicht deterministisch** Gegenteil von zuvor (es kann für ein Zeichen mehrere Übergänge geben und es werden nur Übergänge eingezeichnet, welche für das Erreichen des Akzeptanzzustandes nötig sind).

**NEA** Nichtdeterministischer Endlicher Automat. Kann auch  $\varepsilon$ -Übergänge enthalten, dabei wird von einem  $NEA_\varepsilon$  gesprochen.

**akzeptieren** Ein Wort wird von einem Automaten akzeptiert, wenn sich der Automat nach dem einlesen des gesamten Wortes in einem Akzeptanzzustand befindet.

**akzeptierte Sprache eines DEA**  $L(A) = [w \in \Sigma^* | w \text{ wird von } A \text{ akzeptiert}]$  beinhaltet alle Wörter, welche vom Automaten  $A$  akzeptiert werden.

**regulärer Ausdruck** Notation nach Vorgaben, welche einen Automaten mit \*-Operationen, Alternativen und Verkettungen beschreibt.

**Grammatik** Beschreibung einer kontextfreien Sprache in der Form  $S \rightarrow SS$  oder  $S \rightarrow s$ , welche beschreibt, wie sich Nichtterminalsymbole (in diesem Fall  $S$  und  $SS$ ) "entwickeln" können (i entwickelt sich  $S$  zu  $SS$  oder dem Terminalsymbol  $s$ ).

**kontextfrei** Im Bezug auf CFGs bedeutet kontextfrei, dass die Variable auf der Linken Seite (oben  $S$ ) nicht von ihrem Kontext beeinflusst ist.

**Variable** Nichtterminalsymbol

**Regel** Vorschrift, wie sich die Variable in einer Grammatik entwickeln darf. z.B.  $S \rightarrow s$

**Ableitung** Erstellen einer Kombination von Terminalsymbolen (a.k.a. Wort) aus einer Variable (Frage ist jeweils: Kann  $w$  aus  $S$  abgeleitet werden?)

**Parse Tree** "Baum", welcher beim Ableiten eines Wortes aus einer Variable entsteht. Falls ein Wort mehrere Parse Trees erlaubt, ist die Grammatik nicht in Chomsky-Normalform.

**erzeugte Sprache einer Grammatik** Alle Wörter, die aus einer Grammatik abgeleitet werden können.

**Stackautomat** NEA, welcher über einen Speicher (Stack) verfügt, bei welchem er jeweils auf das Oberste Element zugreifen kann.

**Turing-Maschine** NEA, der über einen unendlichen Speicher (Band) verfügt und einen Lesekopf beliebig auf diesem Band bewegen kann.

**erkannte Sprache einer TM**  $L(M)$  ist die Sprache  $L$ , welche von einer Turing-Maschine  $M$  erkannt werden kann. Beschreibung in **28**.

**Entscheider** Ein Entscheider ist eine TM, die auf jeden beliebigen Input anhält.

**entscheidbare Sprache** Eine Sprache  $L$  ist entscheidbar, wenn es einen entscheider  $M$  gibt mit  $L = L(M)$ . Man sagt  $M$  entscheidet  $L$ .

**Akzeptanzproblem** Es kann nicht entschieden werden, ob:

$$A_{DEA} = \{\langle A, w \rangle \mid A \text{ ein DEA, der } w \text{ akzeptiert.}\}$$

**abzählbar unendlich** Die Elemente einer abzählbar unendlichen Menge lassen sich mit  $n \in \mathbb{N}$  nummerieren.

**überabzählbar unendlich** Die Elemente einer überabzählbar unendlichen Menge werden mit  $r \in \mathbb{R}$  nummeriert. Es lässt sich also noch immer eine Zahl finden, die in mindestens einer Stelle zu allen anderen unterschiedlich ist.

**Reduktion** Reduktion dient dazu, zwei Probleme zu vergleichen. Wird in **37.** erklärt.

**P und NP** Komplexitätsklassen: Die Klasse P besteht aus den sprachen, die mit einem Entscheider (also einer deterministischen Turing-Maschine) in polynomieller Laufzeit entschieden werden können. Die Klasse NP besteht aus den Sprachen, die mit einer nichtdeterministischen Turingmaschine in polynomieller Laufzeit entschieden werden können. Alle Sprachen in NP haben einen polynomiellen Verifizierer. P ist eine unterklasse von NP. **Merksatz:** *P ist deterministisch polynomiell, NP ist nichtdeterministisch polynomiell.*

**NP-vollständig** NP-vollständige Probleme sind die schwersten Probleme in NP. Ein Problem ist NP-vollständig, wenn sich jedes andere Problem in NP darauf reduzieren lässt.

**polynomiale Reduktion** Wird in **37.** erklärt.

**SAT** "Erfüllbarkeitsproblem der Aussagenlogik", wird in **39.** erklärt.

**3SAT** Dasselbe wie SAT, es sind aber maximal drei Literale pro Klausel enthalten. Beispiele dafür sind:

$$F = \overbrace{( \underbrace{x_0}_{\text{Literal}} \vee x_1 \vee x_2 )}^{\text{Klausel}}$$

$$G = (x_0 \vee x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (\bar{x}_2 \vee x_4) \wedge ()$$

**Clique** Beim CLIQUE- oder auch k-CLIQUE-Problem wird danach gefragt, ob es in einem ungerichteten Graphen  $G$  eine Clique/einen Subgraphen gibt, deren/dessen  $n$  (Anzahl) Knoten alle paarweise untereinander verbunden sind.

**Turing-vollständig** Die Turing-vollständigkeit ist die Eigenschaft einer Sprache, die besagt, dass in dieser Sprache ein Turing-Maschinen-Simulator implementiert werden kann. Hierzu wird ein WHILE- oder GOTO-Konstrukt benötigt, da sonst garantiert wäre, dass der Simulator anhält, was die Eigenschaften einer Turing-Maschine verletzen würde.

**LOOP** Ein LOOP ist eine Block-Anweisung einer Programmiersprache, welche die Instruktionen in ihrem Block begrenzt oft ausführt. Auch eine verschachtelung aus LOOP-Anweisungen wird immer terminieren.

## 2 Fragen

### 1. Was ist $\Sigma^*$ ?

Die Menge aller Wörter aus dem Alphabet  $\Sigma$ .

### 2. Was ist der Unterschied zwischen $\varepsilon$ , $\emptyset$ und $\{\varepsilon\}$ ?

$\varepsilon$  ist das leere wort mit der länge  $|w| = 0$ ,  $\emptyset$  ist die leere Sprache, welche gar keine Wörter enthält und  $\{\varepsilon\}$  ist Menge/Sprache, die nur das leere Wort enthält.

**3. Wie unterscheidet sich ein *DEA* von einem *NEA*?**

Ein *DEA* hat zu jedem Zustand für jedes Zeichen genau einen Übergang (Voraussetzung für den Determinismus). Ein *NEA* wiederum darf für jeden Zustand beliebige Übergänge haben, also für jedes Zeichen mehrere, einen oder gar keinen Übergang. Ein  $NEA_\varepsilon$  kann auch sogenannte  $\varepsilon$ -Übergänge haben, welche ohne Input funktionieren.

**4. Wie kann man zwei endliche Automaten vergleichen?**

Man bildet von beiden Automaten den minimalen Automaten z.B. mittels "Kreuzchenalgorithmus". Diese Minimalautomaten sind bei gleichen Automaten ebenfalls gleich.

**5. Nennen Sie drei Methoden, mit denen sie zeigen können, dass eine Sprache regulär ist.**

Eine Sprache ist regulär, wenn sie einen *DEA* besitzt, durch einen regulären Ausdruck beschreiben werden kann oder nur die Operationen Alternative, Verkettung oder  $*$ -Operation verwendet wird.

**6. Nennen Sie zwei Methoden, mit denen sie zeigen können, dass eine Sprache nicht regulär ist.**

Pumping Lemma oder wenn sie keinen *DEA* besitzt.

**7. Beschreiben Sie den Zusammenhang zwischen *DEAs*, *NEAs* und regulären Ausdrücken.**

Ein *NEA* kann zu einem *DEA* gemacht werden (Zustandsmengenalgorithmus), ein *DEA* kann dann zu einem regulären Ausdruck gemacht werden.

**8. Beschreiben Sie die fünf typischen Schritte, die notwendig sind, um mit dem Pumping Lemma zu beweisen, dass eine Sprache nicht regulär ist.**

- Annehmen, dass die Sprache  $L$  regulär ist.
- Es gibt eine Pumping Length  $N$ .
- Ein Wort  $w \in L$  mit  $|w| \geq N$  Festlegen.
- Wort aufteilen  $w = xyz$ ,  $|xy| \leq N$ ,  $|y| > 0$ .
- Was passiert, wenn mit  $y$  gepumpt wird? Ist das neue Wort immer noch in der Sprache?
- Widerspruch und Schlussfolgerung.

**9. Beschreiben Sie *DEAs* für die Sprachen  $\emptyset$ ,  $\{\varepsilon\}$  und  $\Sigma^*$ .**

**10. Warum sind endliche Sprachen regulär?**

Siehe Wikipedia: Myhill-Nerode "Endliche Sprachen sind regulär". Sofern mit dem Satz von Myhill-Nerode bewiesen werden kann, dass der Automat einer Sprache mit endlich vielen Zuständen beschrieben werden kann, ist die Sprache regulär. Da der Automat einer endlichen Sprache auch mit endlich vielen Zuständen beschrieben werden kann, ist die Sprache regulär.

**11. Geben Sie ein typisches Beispiel für eine nicht reguläre Sprache.**

$$L = \{0^n 1^n \mid n \geq 0\}$$

**12. Was ist eine kontextfreie Grammatik?**

Eine kontextfreie Grammatik  $G$  besteht aus Variablen  $V$ , Terminalsymbolen  $\Sigma$  ( $V$  und  $\Sigma$  müssen disjunkt sein), den Regeln  $R$  und der Startvariable  $S \in V$ . Also ist  $G = (V, \Sigma, R, S)$

**13. Was bedeutet  $w \in L(G)$ ?**

$w$  ist ein Wort aus der Sprache  $L$  welche durch die Grammatik  $G$  definiert ist.

**14. Welche Eigenschaften hat eine kontextfreie Grammatik in Chomsky-Normalform?**

Jedes Wort aus einer CFG in CNF hat einen eindeutigen Parse Tree.

**15. Wie kann man eine Grammatik in Chomsky-Normalform bringen?**

Neue Startvariable festlegen,  $\varepsilon$ -Regeln entfernen, Unit-Rules entfernen, Verkettungen ersetzen.

**16. Ist die Chomsky-Normalform eindeutig?**

Ja, es gibt für jede Grammatik nur eine Chomsky-Normalform.

**17. Was müssen Sie tun um nachzuweisen, dass eine Sprache kontextfrei ist?**

Eine Sprache ist dann kontextfrei, wenn ein Stackautomat, Regex oder eine Grammatik dazu existiert.

**18. Wie funktioniert ein Stack-Automat?**

Ein Stackautomat/Pushdown Automata  $P$  ist **immer nichtdeterministisch**. Er besteht aus Zuständen  $Q$ , dem Eingabealphabet  $\Sigma$ , dem Stackalphabet  $\Gamma$ , den Übergängen  $\delta$ , einem Startzustand  $S \in Q$  z.B.  $q_0$  und einer Menge von Akzeptierzuständen  $F$ . Somit ist  $P = (Q, \Sigma, \Gamma, \delta, S, F)$

**19. Welche Eigenschaften muss eine Sprache haben, damit es einen Stack-Automaten gibt, der sie akzeptieren kann?**

Die Sprache muss kontextfrei sein.

**20. Beschreiben Sie eine Technik, mit der Sie zeigen können, dass eine Sprache nicht kontextfrei ist.**

Das Pumping Lemma für kontextfreie Sprachen kann angewendet werden.

**21. Geben Sie Grammatiken an für die Sprachen  $\emptyset$ ,  $\{\varepsilon\}$  und  $\Sigma^*$ .**

Die Menge der Regeln muss Leer sein:  $V = \emptyset$ , eine "Nicht-Regel" wie  $S \rightarrow$  darf nicht existieren, da kein anderes Symbol erreicht wird.

**22. Beschreiben Sie die fünf typischen Schritte, die notwendig sind, um mit dem Pumping Lemma zu beweisen, dass eine Sprache nicht kontextfrei ist.**

- Annehmen, dass die Sprache  $L$  kontextfrei ist.
- Es gibt eine Pumping Length  $N$ .
- Ein Wort  $w \in L$  mit  $|w| \geq N$  Festlegen.
- Wort aufteilen  $w = uvxyz$ ,  $|vxy| \leq N$ ,  $|vy| > 0$ .
- Was passiert, wenn  $v$  und  $y$  gepumpt werden? Ist das neue Wort immer noch in der Sprache?
- Widerspruch und Schlussfolgerung.

**23. Geben Sie ein typisches Beispiel für eine nicht kontextfreie Sprache.**

$$L = \{a^n b^n c^n \mid n \geq 0\}$$

**24. Gibt es eine Turing-Maschine mit nur einem Zustand?**

Nein, da diese Maschine keinen Sinn machen würde. Es muss  $q_{accept} \neq q_{reject}$  gelten, da die Maschine die Eigenschaften einer Turing-Maschine sonst nicht erfüllt.

**25. Wieviele verschiedene Sprachen können von Turing-Maschinen mit zwei Zuständen erkannt werden. Warum?**

Eine Turing-Maschine mit nur zwei Zuständen ( $q_{accept}$  und  $q_{reject}$ ) kann nur keinen Input annehmen und akzeptieren oder irgendetwas annehmen und verwerfen. Somit sind die einzigen erkennbaren Sprachen  $\emptyset$  und  $\Sigma^*$ .

**26. Zählen Sie drei Varianten von Turing-Maschinen auf.**

Mehrbandmaschine, Mehrspurmaschine, nicht deterministische Turingmaschine

**27. Sei  $M$  eine nicht deterministische Turing-Maschine und  $w \in \Sigma^*$ . Was heisst  $w \in L(M)$ ?**

Das Wort  $w$  ist in der Sprache, die von der Turingmaschine  $M$  entschieden wird.

**28. Warum gibt es Sprachen, die nicht Turing-erkennbar sind?**

Die Menge der Turing-Maschinen ist abzählbar unendlich, die Menge aller Sprachen ist aber überabzählbar unendlich. Somit kann es Sprachen geben, die nicht Turing-erkennbar sind.

**29. Was ist der Unterschied zwischen einer Turing erkennbaren Sprache und einer Turing entscheidbaren Sprache?**

Eine Sprache  $L$  heisst Turing-erkennbar, wenn es eine Turing-Maschine  $M$  gibt mit  $L = L(M)$ . Eine Sprache  $L$  ist dann Turing-entscheidbar, wenn die Turing-Maschine  $M$ , welche die Sprache definiert, auf jeden beliebigen Input anhält, während bei einer nur erkennbaren Sprache einzelne Input-Wörter auch dazu führen können, dass die Turing-Maschine endlos weiterrechnet. Somit ist klar, dass eine Turing-entscheidbare Sprache auch Turing-erkennbar ist.

**30. Was müssen Sie tun um nachzuweisen, dass eine Turing-erkennbare Sprache entscheidbar ist?**

Eine Turing-erkennbare Sprache ist dann entscheidbar, wenn es dafür einen Entscheider (ein Entscheider ist eine Turingmaschine, die auf jedem Input  $w \in \Sigma^*$  anhält) gibt.

**31. Beschreiben Sie das prototypische nicht entscheidbare Problem für Turing-Maschinen. Halteproblem?**

**32. Was bedeutet Reduktion eines Problems auf ein anderes?**

Ein Problem wird z.B. auf ein bekanntes NP-vollständiges Problem reduziert, um zu beweisen, dass das Problem ebenfalls NP-vollständig und somit von einer nichtdeterministischen Turing-Maschine in polynomieller Zeit entschieden werden kann bzw. von einer Turing-Maschine in polynomieller Zeit verifiziert werden kann.

**33. Erklären Sie eine Standardtechnik, mit der man nachweisen kann, dass ein Problem nicht entscheidbar ist.**

Reduktion auf ein nicht entscheidbares Problem oder die Anwendung des Satzes von Rice.

**34. Geben Sie ein Beispiel eines nicht entscheidbaren Problems.**

Das Halteproblem.

**35. Was besagt das Halte-Theorem?**

Das Haltetheorem nach Turing besagt, dass anhand der Beschreibung einer Maschine/eines Programms nicht entschieden werden kann, ob sie/es anhalten wird. Beliebte Probleme hierzu sind VirensScanner, welche Programme auf ihr Verhalten überprüfen sollen. Dies ist nicht möglich, da ein solcher Scanner das Halteproblem lösen könnte.

**36. Wie ändert sich die Laufzeit eines Algorithmus, wenn man von einer Variante einer TM zu einer Standard TM übergeht?**

Die Laufzeit wird exponentiell.

**37. Was bedeutet polynomielle Reduktion?**

$A$  und  $B$  sind entscheidbar und  $A$  soll auf  $B$  reduziert werden. Die Abbildung

$$f : \begin{cases} \Sigma^* \rightarrow \Sigma^* \\ w \mapsto f(w) \end{cases}$$

heisst **polynomielle Reduktion von  $A$  auf  $B$**  mit  $w \in A \equiv f(w) \in B$  und  $f(w)$  in polynomieller Zeit in  $|w|$  berechenbar.  $A \leq_p B$  heisst "A ist polynomiell leichter entscheidbar als B". So können Aussagen über die Entscheidbarkeit getätiggt werden: Ist  $B$  entscheidbar, so muss auch  $A$  entscheidbar sein. Ist  $A$  nicht entscheidbar, so darf auch  $B$  nicht entscheidbar sein.

**38. Wie ändert sich die Laufzeit, wenn man eine nichtdeterministische Maschine auf einer deterministischen Maschine simuliert?**

Die Laufzeit wird exponentiell.

**39. Was bedeutet das Problem SAT?**

Das Satisfiability (zu Deutsch "Erfüllbarkeitsproblem der Aussagenlogik") ist ein NP-vollständiges Problem, welches sich folgendermassen beschreiben lässt:  $F$  ist eine aussagenlogische Formel in konjunktiver Normalform. Gibt es eine Belegung der Variablen von  $F$  so, dass die Aussage wahr wird? Als NP-Vollständiges Problem wird SAT von einer deterministischen Turingmaschine (z.B. einem Computer) in exponentieller Zeit entschieden.

- 40. Was unterscheidet ein Problem wie SAT von einem Problem wie die ganzzahlige Division von Zahlen?**

Die division kann von einer deterministischen Turing-Maschine in polynomieller Zeit entschieden werden, SAT nicht.

- 41. Wie können Sie herausfinden, ob eine Programmiersprache Turing-vollständig ist?**

Eine Programmiersprache ist dann Turing-vollständig, wenn sich darin ein Turing-Maschinen-Simulator schreiben lässt.

- 42. Warum ist die Sprache LOOP nicht Turing-vollständig?**

LOOP ist nicht Turing-vollständig, da auch verschachtelte LOOP-Statements immer terminieren werden, da bereits zu Beginn festgelegt ist, wie oft die Anweisungen ausgeführt werden. Eine Turing-Maschine kann also nicht in LOOP simuliert werden.

- 43. Gibt es Probleme, die man WHILE lösen kann, nicht aber GOTO?**

Nein, WHILE und GOTO sind äquivalente Operationen. Der Code, welcher mit WHILE geschrieben wird, ist nur einiges schöner und verständlicher.

- 44. Ist es möglich, einen Compiler zu schreiben, der C-Code in Brainfuck übersetzt?**