

# Automaten und Sprachen

## Inhaltsverzeichnis

1 Reguläre Sprachen .....	1
1.1 Deterministische endliche Automaten (DEA) .....	1
1.1.1 Myhill-Nerode Automat .....	2
1.1.2 Algorithmus für den Minimalautomaten .....	2
1.1.3 Pumping Lemma für reguläre Sprachen .....	2
1.2 Mengenoperationen .....	2
2 Reguläre Operationen und Ausdrücke .....	2
2.1 Alternative .....	2
2.2 Verkettung .....	2
2.3 *-Operation .....	2
2.4 Reguläre Ausdrücke .....	2
2.4.1 Erweiterungen .....	2
3 Kontextfreie Sprachen .....	2
3.1 Reguläre Operationen .....	2
3.1.1 Alternative .....	3
3.1.2 Verkettung .....	3
3.1.3 *-Operation .....	3
3.2 Reguläre Sprachen sind kontextfrei .....	3
3.2.1 Bausteine .....	3
3.3 Chomsky Normalform .....	3
3.3.1 Anforderungen an eine Grammatik .....	3
3.3.2 Transformation in CNF .....	3
4 Nicht kontextfreie Sprachen .....	3
4.1 Pumping-Lemma für kontextfreie Sprachen .....	3
5 Turing-Maschinen .....	4
5.1 Berechnungsgeschichte .....	4
6 Entscheidbarkeit .....	4
6.1 Defintion Entscheidbarkeit .....	4
6.2 Satz von Rice .....	4
7 Komplexität .....	4
7.1 P – NP .....	4
7.2 Aufüllrätsel .....	4
7.3 Reduktion Sudoku → CONSTRAINTS → SAT .....	5
7.3.1 SAT .....	5
7.4 Karp-Katalog .....	5
7.4.1 SAT .....	5
7.4.2 Folgerungen aus P = NP .....	5
7.4.3 Folgerungen aus P ≠ NP .....	5
7.4.4 NP-vollständig .....	5
7.4.5 k-CLIQUE .....	5
7.4.6 SET-PACKING .....	5
7.4.7 VERTEX-COVER .....	5
7.4.8 FEEDBACK-NODE-SET .....	6

7.4.9 FEEDBACK-ARC-SET .....	6
7.4.10 HAMPATH (Hamiltonischer Pfad) .....	6
7.4.11 UHAMPATH .....	6
7.4.12 SET-COVERING .....	7
7.4.13 BIP .....	7
7.4.14 3SAT .....	7
7.4.15 VERTEX-COLORING .....	7
7.4.16 CLIQUE-COVER .....	7
7.4.17 EXACT-COVER .....	7
7.4.18 3D-MATCHING .....	8
7.4.19 STEINER-TREE .....	8
7.4.20 HITTING-SET .....	8
7.4.21 SUBSET-SUM .....	8
7.4.22 SEQUENCING .....	8
7.4.23 PARTITION .....	9
7.4.24 MAX-CUT .....	9
8 Turing-Vollständigkeit .....	9
8.1 Die universelle Turing-Maschine .....	9
8.2 Programmiersprachen und Turing-Vollständigkeit .....	9
8.3 Turing-Vollständigkeit von Programmiersprachen .....	9
8.3.1 LOOP .....	9
8.3.2 Turing-Vollständigkeit .....	9
9 Logik .....	9
9.1 Prädikate .....	9
9.2 Verknüpfungen .....	9
9.3 Distributivgesetze .....	9
9.4 De Morgan .....	9
9.5 Quantoren .....	9
9.5.1 Morgan 2.0 .....	9
10 Alphabet und Wort .....	10
10.1 Wortlänge .....	10
11 Beispiele für DEAs: .....	10
12 Kontextfreie Grammatik .....	10
12.1 Grammatik für Klammerausdrücke .....	10
12.1.1 Beispiel für Ableitung von Klammerausdruck .....	10
12.2 Parse Tree .....	10
12.2.1 regex → CFG .....	10
12.3 Arithmetische Ausdrücke .....	10
12.3.1 Anwendungen der Chomsky-Normalform .....	10
12.3.2 Deterministisches Parsen .....	10
12.3.3 CYK-Ideen .....	10
12.3.4 CYK-Algorithmus .....	10
12.4 Stack-Automaten .....	11
12.4.1 Visualisierung eines Stacks .....	11
12.4.2 Grammatik → Stackautomat .....	11
12.4.3 Backus-Naur-Form (BNF) .....	11
12.5 PDA zu CFG .....	11
12.5.1 Stackautomat standardisieren .....	11
12.5.2 Regeln .....	12
12.6 Vergleich von Maschinen .....	12
12.7 Unendlichkeit .....	12
12.7.1 Grammatik ablesen .....	12
12.8 Spielregeln .....	12
12.9 Zustandsdiagramm .....	12
12.10 Von einer TM erkannte Sprache .....	12
12.11 Turing Maschinen und moderne Computer .....	13
12.12 Aufzähler .....	13
12.13 Vergleich von Maschinen .....	13
12.14 Unendlichkeit .....	13
12.15 Berechenbare Zahlen .....	13
12.15.1 Wieviele berechenbare Zahlen gibt es? .....	13
12.16 Hilberts 10. Problem .....	13
12.17 Sprachprobleme .....	13
12.17.1 $\epsilon$ -Akzeptanzproblem für endliche Automaten .....	13
12.17.2 Gleichheitsproblem für DEAs .....	14
12.17.3 Akzeptanzproblem für DEAs .....	14
12.17.4 Akzeptanzproblem für CFGs .....	14
12.18 Nicht entscheidbare Probleme .....	14
12.19 Reduktion .....	14
12.19.1 Reduktionsbeispiele .....	14
12.19.2 Folgerungen aus $P = NP$ .....	14
12.19.3 Folgerungen aus $P \neq NP$ .....	14
12.19.4 NP-vollständig .....	14
13 Beispielprüfung .....	14

## 1 Reguläre Sprachen

### 1.1 Deterministische endliche Automaten (DEA)

**Definition 1.1.1:** Ein DEA ist ein Quintupel  $(Q, \Sigma, \delta, q_0, F)$ , wobei:

1.  $Q$ , Beliebige endliche Menge von **Zuständen**
2.  $\Sigma$ , **Alphabet** (Endliche Menge)
3.  $\delta : Q \times \Sigma \rightarrow Q$ , **Übergangsfunktion**
4.  $q_0 \in Q$ , **Startzustand**
5.  $F \subset Q$ , Menge der **Akzeptierzustände**

**Definition 1.1.2:** Sei  $A$  ein endlicher Automat, dann ist  $L(A)$  die Sprache

$$L(A) = \{w \in \Sigma^* \mid w \text{ überführt } A \text{ in einen Akzeptierzustand}\}$$

### 1.1.1 Myhill-Nerode Automat

Wir müssen für beliebige Wörter herausfinden, mit welchem weiteren Input der Automat in einen Akzeptierzustand übergeht. Das leere Wort  $\varepsilon$  ist speziell, wir benötigen die Sprache  $L$  selbst, um zum Akzeptierzustand zu kommen. Beispiel:

$$\Sigma = \{0, 1\}$$

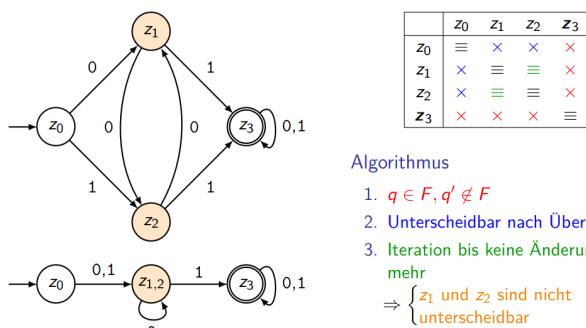
$$L = \{w \in \Sigma^* \mid |w|_0 \text{ gerade}\}$$

w	$L(w)$	Q
$\varepsilon$	$L(\varepsilon) = L$	$q_0$
0	$L(0) = \{w \in \Sigma^* \mid  w _0 \text{ ungerade}\}$	$q_1$
1	$L(1) = \{w \in \Sigma^* \mid  w _0 \text{ gerade}\} = L$	$q_0$

### 1.1.2 Algorithmus für den Minimalautomaten

Wir füllen jeweils nur die untere Hälfte der Tabelle aus:

1. Tabelle erstellen mit allen Zuständen in den Zeilen und Spalten
2. Äquivalente Zustände (die Diagonale) mit  $\equiv$  markieren
3. Akzeptierzustände von normalen Zuständen unterscheiden mit  $\times$ .
4. Falls man von einem Zustandspaar  $(a, b)$  mit einem Übergang bei einem Paar mit  $\times$  landet, kann man das Paar  $(a, b)$  auch mit  $\times$  markieren.



### 1.1.3 Pumping Lemma für reguläre Sprachen

**Satz 1.1.3.1:** Ist  $L$  eine reguläre Sprache, dann gibt es eine Zahl  $N$ , die Pumping Length, so dass jedes Wort  $w \in L$  mit  $|w| \geq N$  in drei Teile  $w = xyz$  zerlegt werden kann, so dass:

$$|y| > 0$$

$$|xy| \leq N$$

$$xy^kz \in L, \forall k \geq 0$$

Um zu beweisen, dass eine Sprache nicht regulär ist, nimmt man an, dass sie regulär ist und führt das Pumping Lemma durch. Gibt es einen Widerspruch, ist die Sprache nicht regulär (Widerspruchsbeweis).

Beispiel ( $L = \{0^n 1^n \mid n \geq 0\}$ ):

1. Annahme:  $L$  ist regulär
2.  $\exists N \in \mathbb{N}$ , Pumping Length
3.  $w = 0^N 1^N$
4. Unterteilung  $w = xyz$

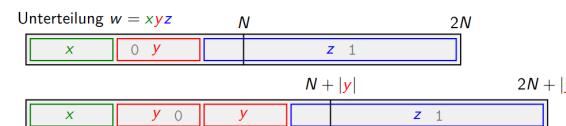


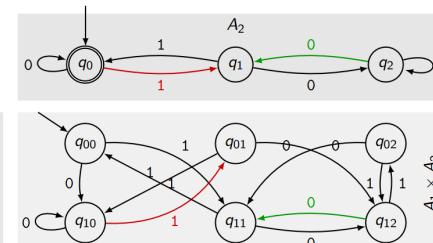
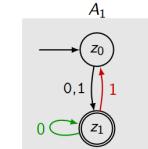
Abbildung 1: Pumping Lemma

5. Pumpen: nur die Anzahl der 0 wird erhöht, Anzahl 1 bleibt
6.  $xy^kz \notin L$  für  $k \neq 1$ , im Widerspruch zum Pumping-Lemma

### 1.2 Mengenoperationen

#### Produktautomat

$$L_i = L(A_i)$$



$$\text{Schnittmenge } L_1 \cap L_2 \\ F = F_1 \times F_2$$

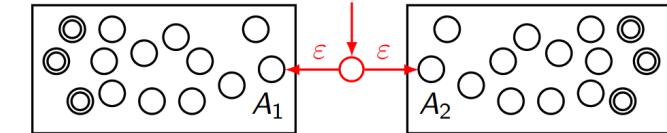
$$\text{Vereinigung } L_1 \cup L_2 \\ F = F_1 \times Q_2 \cup Q_1 \times F_2$$

$$\text{Differenz } L_1 \setminus L_2 \\ F = F_1 \times (Q_2 \setminus F_2)$$

## 2 Reguläre Operationen und Ausdrücke

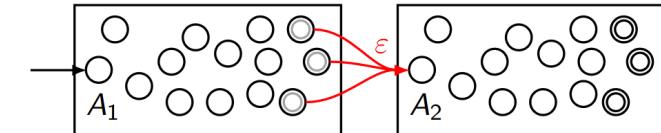
### 2.1 Alternative

$$L = L_1 \cup L_2 = L(A_1) \cup L(A_2)$$



### 2.2 Verkettung

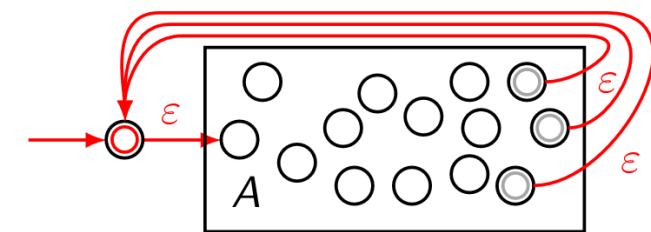
$$L = L_1 L_2 = L(A_1) L(A_2)$$



### 2.3 \*-Operation

$$L^* = \{\varepsilon\} \cup L \cup L^2 \cup \dots$$

$$= \bigcup_{k=0}^{\infty} L^k$$



### 2.4 Reguläre Ausdrücke

- Metazeichen () [] \* ? | . \
- [abc] = a|b|c, [^abc] = nicht [abc] (das ^-Symbol bedeutet innerhalb [] «nicht»)

#### 2.4.1 Erweiterungen

- {n, m}: zwischen n und m
- +: mindestens eines {1, }
- ?: optional {0, 1}
- \d Ziffern, \s whitespace, \w Wortzeichen (inkl. Ziffern), [:space:], [:lower:], [:xdigit:]

## 3 Kontextfreie Sprachen

### 3.1 Reguläre Operationen

$L_1, L_2$  kontextfreie Sprachen mit Grammatiken  $G_i = (V_i, \Sigma, R_i, S_i)$ . Grammatik für reguläre Operationen:

- Neue Startvariable  $S_0$
- $V = V_1 \cup V_2 \cup \{S_0\}$

- Geeignete erweiterte Regeln  $R$

$$\Rightarrow G = (V, \Sigma, R, S_0)$$

### 3.1.1 Alternative

Regeln für  $L_1 \cup L_2$ :

$$R = R_1 \cup R_2 \cup \{S_0 \rightarrow S_1, S_0 \rightarrow S_2\}$$

Im Wesentlichen bleiben die einzelnen Regeln dieselben, es müssen nun einfach alle Regeln der Sprachen  $L_1, L_2$  untereinander geschrieben werden. Wichtig ist jedoch, dass wir einen neuen Startzustand konstruieren, den wir mit Regeln in  $S_1, S_2$  überführen.

### 3.1.2 Verkettung

Regeln für  $L_1 L_2$ :

$$R = R_1 \cup R_2 \cup \{S_0 \rightarrow S_1 S_2\}$$

### 3.1.3 \*-Operation

Regeln für  $L_1^*$ :

$$R = R_1 \cup \{S_0 \rightarrow S_0 S_1, S_0 \rightarrow \epsilon\}$$

Diese Regel besagt einfach, dass  $S_1$  beliebig oft wiederholt werden kann.

**Satz 3.1.3.1:** Die Klasse der kontextfreien Sprachen ist abgeschlossen unter regulären Operationen.

## 3.2 Reguläre Sprachen sind kontextfrei

### 3.2.1 Bausteine

- Einzelne Buchstaben:
  - $A \rightarrow a$
  - $B \rightarrow b$
  - ...
  - $Z \rightarrow z$
- Leeres Wort
  - empty  $\rightarrow \epsilon$

## 3.3 Chomsky Normalform

### 3.3.1 Anforderungen an eine Grammatik

1. Keine Unit-Rules  $A \rightarrow B$
2. Keine Regeln  $A \rightarrow \epsilon$  außer wenn nötig  $S \rightarrow \epsilon$  (Startvariable  $S$ )
3. Keine Regeln mit mehr als 2 Variablen auf der rechten Seite

$\Rightarrow$  Rechte Seite enthält genau zwei Variablen oder genau ein Terminal-Symbol

### 3.3.2 Transformation in CNF

1. Neue Startvariable  $S_0 \rightarrow S$

2.  $\epsilon$ -Regeln
3. Unit-Rules
4. Verkettungen auflösen

Beispiel:

$$\begin{aligned} S &\rightarrow ASA \mid aB \\ A &\rightarrow B \mid S \\ B &\rightarrow b \mid \epsilon \end{aligned}$$

**Neue Startvariable:**

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow ASA \mid aB \\ A &\rightarrow B \mid S \\ B &\rightarrow b \mid \epsilon \end{aligned}$$

$\epsilon$  eliminieren:

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow ASA \mid aB \mid a \\ A &\rightarrow B \mid S \mid \epsilon \\ B &\rightarrow b \end{aligned}$$

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow ASA \mid SA \mid AS \mid S \mid aB \mid a \\ A &\rightarrow B \mid S \\ B &\rightarrow b \end{aligned}$$

**Unit-Rules:**

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow ASA \mid SA \mid AS \mid aB \mid a \\ A &\rightarrow b \mid S \\ B &\rightarrow b \\ S_0 &\rightarrow ASA \mid SA \mid AS \mid aB \mid a \\ S &\rightarrow ASA \mid SA \mid AS \mid aB \mid a \\ A &\rightarrow ASA \mid SA \mid AS \mid aB \mid a \\ B &\rightarrow b \end{aligned}$$

**Verkettungen, Terminalsymbole:**

$$\begin{aligned} S_0 &\rightarrow AA_1 \mid SA \mid AS \mid aB \mid a \\ S &\rightarrow AA_1 \mid SA \mid AS \mid aB \mid a \\ A &\rightarrow AA_1 \mid SA \mid AS \mid aB \mid a \\ B &\rightarrow b \\ A_1 &\rightarrow SA \end{aligned}$$

$$\begin{aligned} S_0 &\rightarrow AA_1 \mid SA \mid AS \mid UB \mid a \\ S &\rightarrow AA_1 \mid SA \mid AS \mid UB \mid a \\ A &\rightarrow AA_1 \mid SA \mid AS \mid UB \mid a \mid b \\ B &\rightarrow b \\ A_1 &\rightarrow SA \\ U &\rightarrow a \end{aligned}$$

**Definition 3.3.2.1** (Chomsky-Normalform): Eine CFG ist in Chomsky-Normalform (CNF), wenn  $S$  auf der rechten Seite nicht vorkommt und jede Regel von der Form  $A \rightarrow BC$  oder  $A \rightarrow a$  ist, zusätzlich ist die Regel  $S \rightarrow \epsilon$  erlaubt.

## 4 Nicht kontextfreie Sprachen

### 4.1 Pumping-Lemma für kontextfreie Sprachen

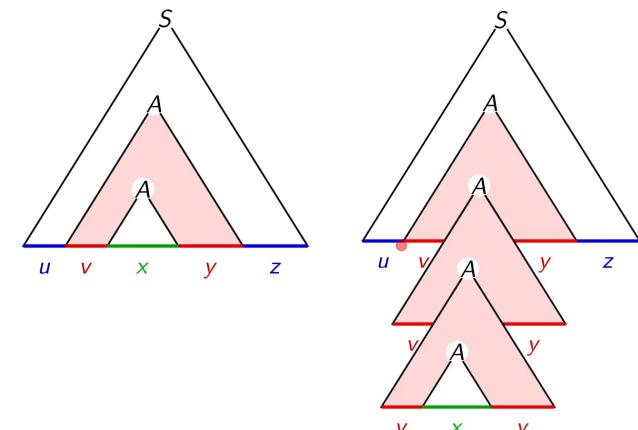


Abbildung 2: Herleitung Pumping Lemma für CFGs

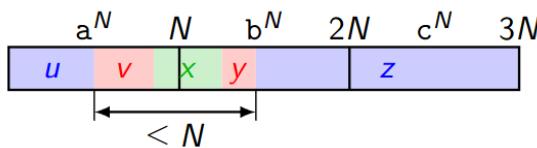
**Definition 4.1.1** (Pumping Lemma für CFL): Ist  $L$  eine CFL, dann gibt es eine Zahl  $N$ , die Pumping Length, derart, dass jedes Wort  $w \in L$  mit  $|w| \geq N$  zerlegt werden kann in fünf Teile  $w = uvxyz$  derart, dass:

1.  $|vy| > 0$
2.  $|vxy| \leq N$
3.  $uv^kxy^kz \in L, \forall k \in \mathbb{N}$

Mit dem Pumping-Lemma kann man beweisen, dass eine Sprache nicht kontextfrei ist.

- Beispiel:  $\{a^n b^n c^n \mid n \geq 0\}$   
1. Annahme  $L$  kontextfrei

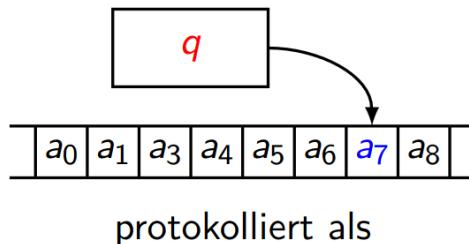
2. Pumping Length  $N$
3. Wort:  $w = a^N b^N c^N$
4. Zerlegungen (mehrere):



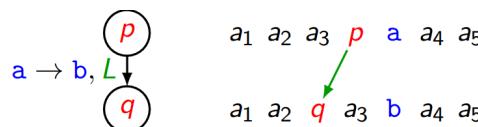
5. Beim Pumpen nimmt die Anzahl der  $a$  und  $b$  zu, nicht aber die Anzahl der  $c \Rightarrow uv^kxy^kz \notin L, \forall k \neq 1$
6. Widerspruch:  $L$  nicht kontextfrei

## 5 Turing-Maschinen

### 5.1 Berechnungsgeschichte



$a_0 a_1 a_2 a_3 a_4 a_5 a_6 q a_7 a_8$



## 6 Entscheidbarkeit

Eine Sprache heisst entscheidbar, wenn es einen Entscheider gibt, der prüfen kann, ob ein Wort  $w$  in der Sprache liegt.

Beispiel: Sei  $A$  ein DEA. Dann kann man folgenden Algorithmus  $M_A$  mit Input  $w$  bauen:

1. Simuliere  $A$  auf  $w$
2. Falls  $A$  in Akzeptanzzustand:  $q_{\text{accept}}$
3. Andernfalls:  $q_{\text{reject}}$

Daraus folgt  $L(A) = L(M_A)$ , oder  $M_A$  ist ein Entscheider für die Sprache  $L(A)$ .

**Satz 6.1:** Reguläre Sprachen sind entscheidbar.

Beispiel: Sei  $G$  eine CFG. Dann kann man folgenden Algorithmus  $M_G$  mit Input  $w$  bauen:

1. Wandle  $G$  in Chomsky-Normalform  $G'$  um
2. Wende den CYK-Algorithmus auf  $G'$  und Wort  $w$  an
3. Wenn der CYK-Algorithmus das Wort  $w$  akzeptiert:  $q_{\text{accept}}$
4. Andernfalls:  $q_{\text{reject}}$

### 6.1 Definition Entscheidbarkeit

**Definition 6.1.1** (Entscheider): Ein Entscheider ist eine Turing Maschine, die auf jedem beliebigen Input anhält.

**Definition 6.1.2** (Entscheidbar): Eine Sprache  $L$  heisst entscheidbar, wenn es einen Entscheider  $M$  gibt mit  $L = L(M)$ . Man sagt,  $M$  entscheidet  $L$ .

Jedes Problem kann in ein Sprachproblem übersetzt werden:

$$L_P = \{w \in \Sigma^* \mid w \text{ ist Lösung des Problems } P\}$$

Beispiel (Leerheitsproblem):

$$E_{\text{DEA}} = \{\langle A \rangle \mid A \text{ ein DEA und } L(A) = \emptyset\}$$

Beispiel (Gleichheitsproblem):

$$EQ_{\text{CFG}} = \{\langle G_1, G_2 \rangle \mid G_i \text{ CFGs und } L(G_1) = L(G_2)\}$$

Beispiel (Akzeptanzproblem):

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

Beispiel (Halteproblem):

$$HALT_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ hält auf Input } w\}$$

**Satz 6.1.1** (Halteproblem): Das spezielle Halteproblem

$$HALT_{\varepsilon_{\text{TM}}}$$

$$= \{\langle M \rangle \mid M \text{ ist eine Turingmaschine und } M \text{ hält auf leerem Band}\}$$

ist nicht entscheidbar.

**Satz 6.1.2** (Allgemeines Halteproblem): Das allgemeine Halteproblem

$$\begin{aligned} & HALT_{\text{TM}} \\ & = \{\langle M, w \rangle \mid M \text{ ist eine Turingmaschine und } M \text{ hält auf Input } w\} \\ & \text{ist nicht entscheidbar.} \end{aligned}$$

Weitere nicht entscheidbare Probleme:

- Leerheitsproblem  $E_{\text{TM}}$

### 6.2 Satz von Rice

Eigenschaften Turing-erkennbarer Sprachen

- **REGULAR**:  $L(M)$  ist regulär
- **$E$** :  $L(M)$  ist leer

**Definition 6.2.1**: Eine Eigenschaft  $P$  Turing-erkennbarer Sprachen heisst nichttrivial, wenn es zwei Turingmaschinen  $M_1$  und  $M_2$  gibt, mit:

$L(M_1)$  hat Eigenschaft  $P$

$L(M_2)$  hat Eigenschaft  $P$  nicht

**Satz 6.2.1** (Rice): Ist  $P$  eine nichttriviale Eigenschaft Turing-erkennbarer Sprachen, dann ist

$$P_{\text{TM}} = \{\langle M \rangle \mid L(M) \text{ hat Eigenschaft } P\}$$

nicht entscheidbar.

## 7 Komplexität

### 7.1 P – NP

### 7.2 Aufüllrätsel

Viele Ausfüllrätsel wie z.B. Sudoku sind exponentiell lösbar. Man versucht dabei einfach jede Möglichkeit und falls eine Möglichkeit nicht zu einer korrekten Lösung führt, machen wir ein Backtracking. Man spricht von einer Nicht-Deterministischen Turing-Maschine, bei welcher wir alle Möglichkeiten durchprobieren müssen. Eine solche Maschine kann aber polynomiell verifiziert werden, in dem man den Pfad durchgeht, welcher verwendet worden ist für die Lösung des Rätsels und prüft, ob dieser in  $q_{\text{accept}}$  landet.

### 7.3 Reduktion Sudoku → CONSTRAINTS → SAT

Sudoku-Regeln werden als logische «Constraints» formuliert. Dabei müssen alle Constraints erfüllt sein. Es gibt Software/Libraries wie python-constraint für diese Probleme.

**Allgemein:** Jedes Ausfüllrätsel lässt sich auf CONSTRAINTS = SAT reduzieren.

#### 7.3.1 SAT

Eine logische Formel ist in SAT genau dann, wenn sie erfüllt werden kann. Constraints werden miteinander «verundet», diese Formel soll wahr ergeben.

$$\text{SAT} = \{\varphi \mid \varphi \text{ erfüllbar}\}$$

Seine Sprache in NP  $\Rightarrow$  Es gibt eine nichtdeterministische TM  $M$ , die  $A$  in polynomieller Zeit  $O(t(n))$  entscheidet.  $\Rightarrow A$  kann polynomiell auf SAT reduziert werden:  $A \leq_P \text{SAT}$

Beschreibe das Finden der Berechnungsgeschichte von  $M$  als polynomielles Ausfüllrätsel.

### 7.4 Karp-Katalog

#### 7.4.1 SAT

Das SAT-Problem prüft im Allgemeinen, ob ein Ausdruck true wird. Dann ist er satisfiable. Es sind folgende Dinge gegeben:

- Aussagenlogische Formel

#### 7.4.2 Folgerungen aus P = NP

1. Für jedes Problem in NP gibt es einen polynomiellen Algorithmus
2. Es gibt keine «schwierigen» Probleme
3. Mit Moore's Law lässt sich jedes Problem «lösen»
4. Es gibt keine sichere Kryptographie

#### 7.4.3 Folgerungen aus P ≠ NP

1. NP-vollständige Probleme haben nicht skalierende Lösungen
2. Moore's Law hilft nicht in  $NP \setminus P$

#### 7.4.4 NP-vollständig

Eine entscheidbare Sprache  $B$  heisst NP-vollständig, wenn sich jede Sprache  $A$  in NP polynomiell auf  $B$  reduzieren lässt:

$$A \leq_P B, \forall A \in \text{NP}$$

Wenn ein Problem NP-vollständig ist:

- Lösung braucht typischerweise exponentielle Zeit
- Korrektheit der Lösung ist leicht (in polynomieller Zeit) zu prüfen
- Andernfalls wären Tests schon exponentiell und somit in Software nicht sinnvoll

Falls  $P \neq \text{NP}$ , dann können NP-vollständige Probleme nicht in polynomieller Zeit gelöst werden.

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Variablen (boolesche Werte)
- Aktion, die den Wahrheitszustands einer Variable verändert
- (falls vorhanden) Zwischenausdrücke von logischen Formeln
- Finaler Logischer Ausdruck zur Erfüllung des Ausgangproblems

SAT eignet sich für Probleme, bei welchen Elemente (Variablen) nur zwei Zustände einnehmen können. Jedes Ausfüllrätsel ist mit SAT beschreibbar. Man kann Wahrheitstabellen bilden. Jedes Problem, das sich auf SAT reduzieren lässt, ist NP-vollständig.

#### 7.4.5 k-CLIQUE

**Gegeben:** Graph  $G$  (bestimmte Anzahl und Anordnung von Knoten), Zahl  $k$

**Fragestellung:** Gibt es  $k$  Knoten, die alle miteinander verbunden sind? Diese Knoten bilden eine sogenannte k-CLIQUE.

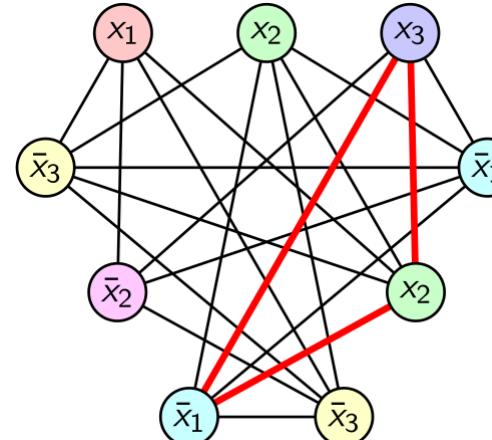


Abbildung 3: k-Clique

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Knoten (die miteinander via Kanten verbunden werden)
- Kanten
- $k$  (Anzahl verbundener Knoten, bzw. Größe der Clique)
- Clique (Was bildet die Clique?)

Eignet sich für Probleme, bei denen möglichst viele Elemente eine Bedingung erfüllen müssen.

#### 7.4.6 SET-PACKING

**Gegeben:** Eine Familie  $(S_i), i \in I$  und eine Zahl  $k \in \mathbb{N}$

**Fragestellung:** Gibt es eine  $k$ -elementige Teilstammie  $(S_i), i \in I$  mit  $J \subset I$  (also  $|J| = k$ ) derart, dass die Menge der Teilstammie paarweise disjunkt sind ( $S_i \cap S_j = \emptyset$ )?

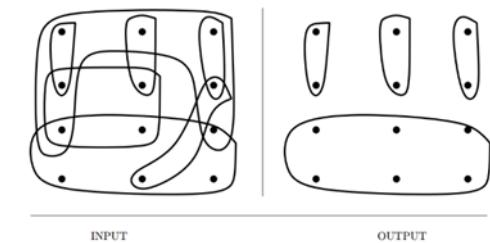


Abbildung 4: SET-PACKING

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Menge  $I$  (Eigenschaft zum Vergleich von Elementen)
- Familie  $S_i$  (Elemente mit Eigenschaften aus der Menge  $I$ )
- Teilmenge  $J$  von  $I$  (so dass  $|J| = k$ )
- Teilstammie, so dass die enthaltenen Mengen unterscheidbar sind

#### 7.4.7 VERTEX-COVER

**Gegeben:** Graph  $G$  und eine Zahl  $k$ .

**Fragestellung:** Gibt es eine Teilmenge von  $k$  Vertizes so, dass jede Kante des Graphen ein Ende in dieser Teilmenge hat? Jeder Knoten außerhalb des Graphen muss dementsprechend eine Kante zur Teilmenge besitzen.

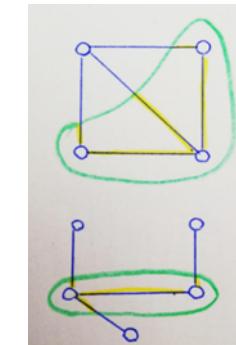


Abbildung 5: VERTEX-COVER

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Knoten des Graphen
- Kanten (Verbindung zwischen den Knoten)
- $k$  (Anzahl Knoten, so dass jeder andere Knoten in dieser Teilmenge eine Kante besitzt)

#### 7.4.8 FEEDBACK-NODE-SET

**Gegeben:** Gerichteter Graph  $G$  und eine Zahl  $k$

**Fragestellung:** Gibt es eine endliche Teilmenge von  $k$  Vertizes in  $G$  so, dass jeder Zyklus<sup>1</sup> in  $G$  einen Vertex in der Teilmenge enthält?

**Info 7.4.8.1:** Gerichtete Graphen sind eine Klasse von Graphen, die keine Symmetrie zwischen den Knotenpunkten gebildeten Kanten voraussetzen.

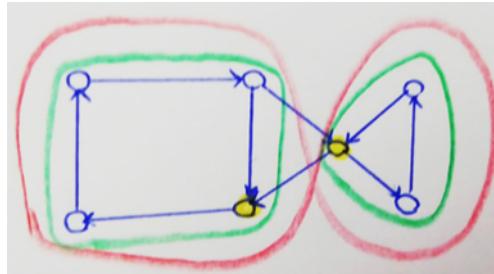


Abbildung 6: FEEDBACK-NODE-SET

**Reduktion:** Folgende Elemente müssen reduziert werden:

- gerichteter Graph
- Knoten
- Kanten
- Richtung
- Zyklen
- $k$  (Anzahl verbindende Knoten)
- Node Set (ausgewählte Knoten)

#### 7.4.9 FEEDBACK-ARC-SET

**Gegeben:** Gerichteter Graph  $G$ , Zahl  $k$

**Fragestellung:** Gibt es eine Teilmenge von  $k$  Kanten so, dass jeder Zyklus<sup>1</sup> in  $G$  eine Kante aus der Teilmenge enthält?

Im Vergleich zum FEEDBACK-NODE-SET wird im FEEDBACK-ARC-SET meist beschrieben, dass ein Vorgang während einer Verschiebung (auf einer Kante) gemacht wird.

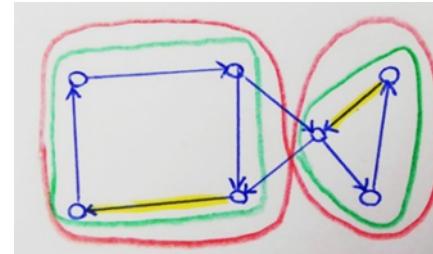


Abbildung 7: FEEDBACK-ARC-SET

**Reduktion:** Folgende Elemente müssen reduziert werden:

- gerichteter Graph
- Knoten
- Kanten
- Richtung
- Zyklen
- $k$  (Anzahl verbindende Kanten)
- Arc Set (ausgewählte Kanten)

#### 7.4.10 HAMPATH (Hamiltonischer Pfad)

Das HAMPATH-Problem beschreibt die Suche nach einem geschlossenen Pfad in einem gerichteten Graphen, der genau einmal durch jeden Knoten geht.

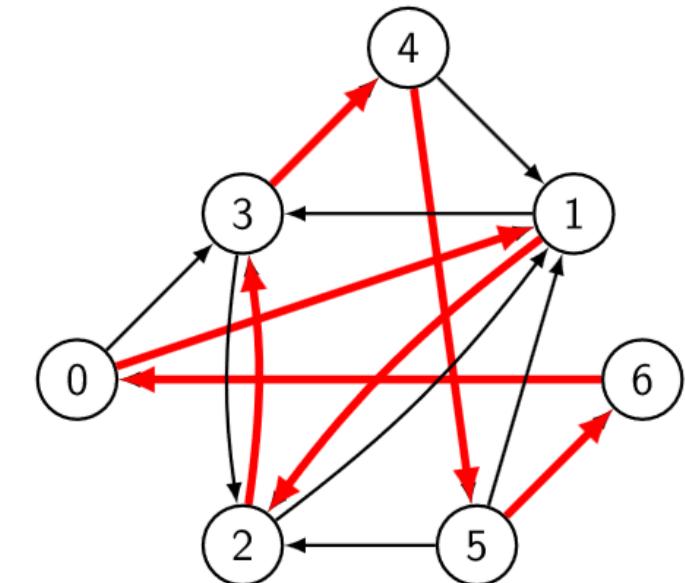


Abbildung 8: HAMPATH

#### 7.4.11 UHAMPATH

Das UHAMPATH-Problem beschreibt die Suche nach einem hamiltonischen Pfad in einem ungerichteten Graphen zu finden.

**Info 7.4.11.1:** Bei einem ungerichteten Graphen sind die Verbindungen zwischen den Knoten (Kanten) symmetrisch.

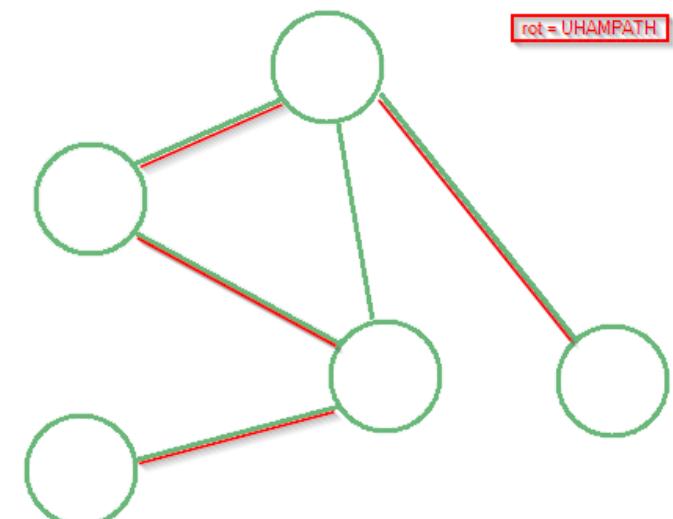


Abbildung 9: UHAMPATH

<sup>1</sup>Ein Zyklus ist ein sich wiederholender Ablauf

## 7.4.12 SET-COVERING

**Gegeben:** endliche Familie endlicher Mengen  $(S_j)_{1 \leq j \leq n}$  und eine Zahl  $k$

**Fragestellung:** Gibt es eine Unterfamilie bestehend aus  $k$  Mengen, die die gleiche Vereinigung hat? Kann man  $k$  Teilmengen bilden, welche die Menge  $S$  komplett abdecken?

Ziel ist es, alle Elemente abzudecken. Überschneidungen der Mengen sind möglich.

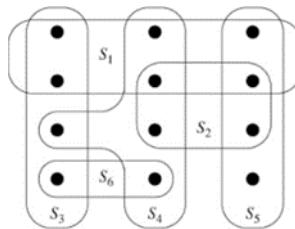


Abbildung 10: SET-COVERING

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Nummerierung von 1 bis  $n$
- Familie endlicher Mengen
- Unterfamilie bestehend aus  $k$  Mengen
- Bedingung der beiden Vereinigungen

## 7.4.13 BIP

Das BIP-Problem (Binary Integer Programming) beschreibt, dass zu einer ganzzahligen Matrix  $C$  und einem ganzzahligen Vektor  $d$  ein binärer Vektor  $x$  gefunden werden kann mit  $C \cdot x = d$ .

$$\begin{pmatrix} 1 & 3 & 0 \\ 0 & 2 & 5 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 5 \end{pmatrix}$$

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Matrix  $C$
- Vektor  $d$
- Resultat für die Suche nach dem Vektor  $x$  zur Erfüllung der Gleichung  $C \cdot x = d$

## 7.4.14 3SAT

3SAT ist eine Variante vom SAT-Problem, wo die aussagenlogische Formel als konjunktive Normalform mit 3 Variablen pro Klausel gegeben ist. Beispiel:

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$$

Mit Erfüllungsequivalenz darf jede SAT-Formel in eine 3SAT-Formel umgewandelt werden.

Die Reduktion ist identisch zu Abschnitt 7.4.1 SAT.

## 7.4.15 VERTEX-COLORING

Beim  $k$ -Vertex-Coloring-Problem sind folgende Dinge gegeben:

**Gegeben:** Graph  $G$ , Anzahl  $k$  Farben

**Fragestellung:** Kann man die Knoten so mit  $k$ -Farben einfärben, dass benachbarte Knoten verschiedene Farben haben?

**Fragestellung:**

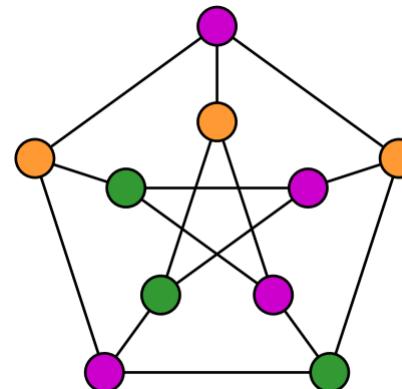


Abbildung 11: k-VERTEX-COLORING

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Vertizes
- Kanten
- Farben
- $k$ : Anzahl Farben

Die verbundenen Vertizes sollen alle verschiedene Farben haben:

- Die Farben stellen den (gesuchten) Unterschied zwischen den Vertizes dar (eine unterscheidbare Eigenschaft)
- Die Kanten zwischen den Vertizes stellen die Regeln für die unterscheidbaren Objekte dar.

Das VERTEX-COLORING-Problem eignet sich für Probleme, bei denen Elemente, die miteinander in Beziehung stehen, unterschieden werden müssen.

## 7.4.16 CLIQUE-COVER

**Gegeben:** Graph  $G$ , positive Zahl  $k$

**Fragestellung:** Gibt es  $k$  Cliques so, dass jede Ecke in genau einer der Cliques ist?

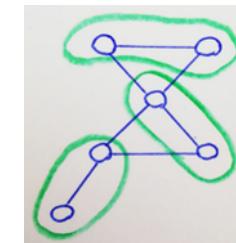


Abbildung 12: CLIQUE-COVER

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Vertizes
- Kanten
- $k$  (Anzahl Vertizes)
- Clique (verbund von möglichst vielen Vertizes)
- Covering (Bedingung)

## 7.4.17 EXACT-COVER

**Gegeben:** Eine Familie  $(S_j)_{1 \leq j \leq n}$  von Teilmengen einer Menge  $U$ .

**Fragestellung:** Gibt es eine Unterfamilie von Mengen, die disjunkt sind, aber dieselbe Vereinigung haben? Die Unterfamilie  $(S_{j_i})_{1 \leq i \leq m}$  muss also  $S_{ji} \cap S_{jk} = \emptyset$  und  $\bigcup_{j=1}^n S_j = \bigcup_{i=1}^m S_{ji}$  erfüllen.

Jedes Element in  $U$  soll genau in einer der Teilmengen einer Familie  $S$  vorkommen. Die gesuchte Menge bildet eine exakte Überdeckung. Es darf keine Überschneidungen geben, aber alle Elemente sollen abgedeckt werden.

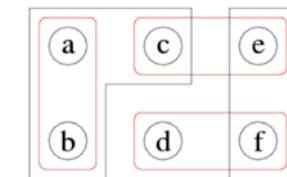


Abbildung 13: EXACT-COVER

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Menge  $U$
- Familie  $S_j \subset U$
- Unterfamilie  $S_{ji}$  (so, dass die gleiche Vereinigung wie die Familie  $S_j$  erzielt wird)
- Bedingung  $S_{ji} \cap S_{jk} = \emptyset$
- Bedingung  $\bigcup_{j=1}^n S_j = \bigcup_{i=1}^m S_{ji}$

**Info 7.4.17.1** (Unterschied Set-Covering, Exact-Cover, Set-Pack-ing):

- In SET-COVERING und SET-PACKING kommt eine Zahl  $k$  (z.B. maximale Anzahl Allergien) vor,

nicht aber in EXACT-COVER.

- In SET-COVERING dürfen sich die Mengen schneiden, müssen aber auch alles abdecken. In SET-PACKING

dürfen sich die Mengen nicht schneiden, müssen aber auch nicht alles abdecken.

- In EXACT-COVER dürfen sich die Mengen nicht schneiden, müssen alles abdecken, aber es kommt nicht auf ihre Anzahl an.

## 7.4.18 3D-MATCHING

**Gegeben:** Endliche Menge  $T$  und Menge  $U$  von Tripeln  $T^3$ .

**Fragestellung:** Gibt es eine Teilmenge  $W$  von  $U$  so, dass  $|W| = |T|$  und keine zwei Elemente von  $W$  in irgendeiner Koordinate übereinstimmen?

Für jeden Wert im Tripel  $(x, y, z)$  gibt es eine bestimmte Bedeutung abhängig vom Kontext.

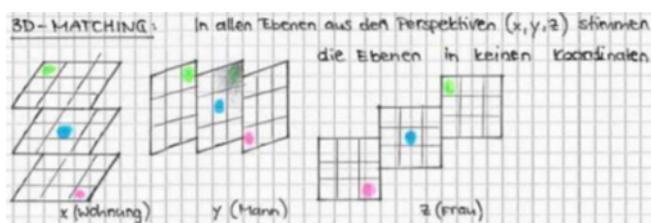


Abbildung 14: 3D-MATCHING

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Menge  $T$  (normalerweise eine bestimmte Anzahl  $n$  von Elementen)
- Tripel aus  $T^3$
- Menge  $U$  (bestehend aus Kombinationen der Tripel)
- Teilmenge  $W$  von  $U$  so, dass  $n = |W| = |T|$  (jedes Tripel muss in jedem Element für  $x, y, z$  eindeutig sein)

Beispiel:

- $T = \{0, 1\}$
- $U = \{(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)\}$
- $W = \{(1, 0, 1), (0, 1, 0)\}$  erfüllt die Bedingung

## 7.4.19 STEINER-TREE

**Gegeben:** Ein Graph  $G$ , eine Teilmenge  $R$  von Verzets, eine Gewichtsfunktion und eine positive Zahl  $k$ .

**Fragestellung:** Gibt es einen Baum mit Gewicht  $\leq k$ , dessen Knoten in  $R$  enthalten sind? Das Gewicht des Baumes ist die Summe der Gewichte über alle Kanten im Baum.

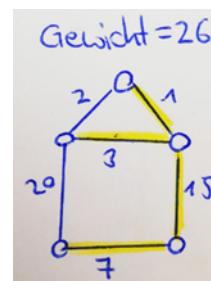


Abbildung 15: STEINER-TREE

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Vertizes
- Kanten (im Fall X)
- Gewichtsfunktion (zum Vergleich der Kanten und Wahl des Baumes)
- Knoten in  $R$  (bestimmte Auswahl von Vertizes aufgrund einer Bedingung)
- maximales Gewicht  $k$  (so, dass es sich noch «lohnt»)
- Baum (der die einzelnen Vertizes schliesslich verbinden soll)

## 7.4.20 HITTING-SET

**Gegeben:** Menge von Teilmengen  $S_i \subset S$

**Fragestellung:** Gibt es eine Menge  $H$ , die jede Menge in genau einem Punkt trifft ( $H \subset \bigcup_{i \in J} S_i$ ), also  $|H \cap S_i| = 1$ ?

Beispiel:

- Gegeben:  $A = \{1, 2, 3\}, B = \{1, 2, 4\}, C = \{1, 2, 5\}$
- Gesucht:  $H = \{3, 4, 5\}$

**Reduktion:** Folgende Elemente müssen reduziert werden:

- $i$  (Bedingung/Merkmal zur eindeutigen Auswahl eines Elements)
- Menge  $S_i$  (Menge von Teilmengen)
- Menge  $H$  (Resultat mit Bedingung  $|H \cap S_i| = 1$ )

## 7.4.21 SUBSET-SUM

**Gegeben:** Menge  $S$  von ganzen Zahlen

**Fragestellung:** Kann man eine Teilmenge finden, die als Summe einen bestimmten Wert  $t$  hat?

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Elemente, zwischen denen entschieden werden muss
- Menge  $S$  von Elementen
- bestimmter Wert  $t$
- Teilmenge  $T$ , so dass die Werte der Elemente dem Wert  $t$  entspricht:  $\sum_{x \in T} x = t$

Der Name «Rucksack»-Problem röhrt daher, dass man sich die Zahlen aus  $S$  als «Grösse» von Gegenständen vorstellt, und wissen möchte, ob man einen Rucksack der Grösse  $t$  exakt füllen kann mit einer Teilmenge von Gegenständen aus  $S$ .

## 7.4.22 SEQUENCING

**Gegeben:** Ein Vektor von Laufzeiten von  $p$  Jobs, ein Vektor von spätesten Ausführzeiten, ein Strafenvektor und eine positive ganze Zahl  $k$ .

**Fragestellung:** Gibt es eine Permutation der Zahlen  $1, \dots, p$ , sodass die Gesamtstrafe für verspätete Ausführung bei der Ausführung der Jobs nicht grösser ist als  $k$ ?

Vereinfachte Definition:

**Gegeben:** Eine Menge von Jobs, pro Job eine Ausführzeit, Deadline und eine Strafe sowie eine maximale Strafe  $k$ . Die Jobs müssen

sequenziell abgearbeitet werden. Wird ein Job zu spät fertig, muss eine Strafe gezahlt werden.

**Fragestellung:** Gibt es eine Reihenfolge von Jobs so, dass die Strafe kleiner gleich  $k$  ist?

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Ausführungszeit von Job
- Deadline
- Strafe
- Permutation/Reihenfolge
- Zusammensetzung der Gesamtstrafe

#### 7.4.23 PARTITION

**Gegeben:** Eine Folge von  $n$  ganzen Zahlen  $c_1, c_2, \dots, c_n$

**Fragestellung:** Kann man die Indizes  $1, 2, \dots, n$  in zwei Teilmengen  $I$  und  $\bar{I}$  teilen, so dass die Summe der zugehörigen Zahlen  $c_i$  identisch ist ( $\sum_{i \in I} c_i = \sum_{i \in \bar{I}} c_i$ )? Gibt es zwei disjunkte Teilmengen mit derselben Summe?

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Indizes  $i$  (konkretes Objekt zum Vergleich/zur Aufteilung)
- Werte der Vergleichsobjekte  $c_i$
- Aufteilung in zwei Teilmengen  $I$  und  $\bar{I}$  so, dass der Wert der entsprechenden Vergleichsobjekte identisch ist

Beispiel: Eine Reihe von Wassergläsern ist unterschiedlich gefüllt. Es sollen 2 Behälter gleich voll mit den Gläsern gefüllt werden. Welche Gläser müssen in welche Behälter geleert werden?

#### 7.4.24 MAX-CUT

**Gegeben:** Graph  $G$  mit einer Gewichtsfunktion  $w : E \rightarrow Z$  und eine ganze Zahl  $W$ .

**Fragestellung:** Gibt es eine Teilmenge  $S$  der Vertizes, so dass das Gesamtgewicht der Kanten, die  $S$  mit seinem Komplement verbinden, so gross wie  $W$ ?

$$\sum_{\{u,v\} \in E \wedge u \in S \wedge v \notin S} w(\{u,v\}) \geq W$$

Der Max-Cut eines Graphen ist eine Zerlegung seiner Knotenmenge in zwei Teilmengen, sodass das Gleichgewicht der zwischen den beiden Teilen verlaufenden Kanten mindestens  $W$  wird.

MAX-CUT sucht die maximalen Investitionen, die man in den Sand setzen muss, indem man eine Menge von Verbindungen durchschneidet.

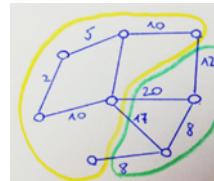


Abbildung 16: MAX-CUT

**Reduktion:** Folgende Elemente müssen reduziert werden:

- Vertizes
- Kanten
- Gewichtsfunktion der Kante (zum Vergleich der Kanten)
- Wert  $W$  (Ziel, das bei der Wahl der Teilmenge  $S$  der Vertizes überschritten werden sollte)
- Gesamtgewicht
- Teilmenge  $S$  der Vertizes (Trennlinie zwischen zwei Gruppierungen)

## 8 Turing-Vollständigkeit

### 8.1 Die universelle Turing-Maschine

Gibt es eine TM, die jede beliebige TM simulieren kann?

Charles Babbage, Ada Lovelace: Analytical Engine

Alan Turing: Es gibt eine Turing-Maschine, die jede beliebige andere Turing-Maschine simulieren kann<sup>2</sup>:

- Eigenes Band für die Codierung der Übergangsfunktion  $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- Eigenes Band für den aktuellen Zustand
- Arbeitsband
- Simulation dieser Maschine auf einer Standard-TM

### 8.2 Programmiersprachen und Turing-Vollständigkeit

**Definition 8.2.1:** Eine Programmiersprache heißt Turing-vollständig, wenn in ihr jede beliebige Turing-Maschine simuliert werden kann.

Frage: Gibt es einen in  $A$  geschriebenen Turing-Maschinen-Simulator?

## 8.3 Turing-Vollständigkeit von Programmiersprachen

### 8.3.1 LOOP

Führe ein Programm  $P$  genau  $x_i$  mal aus:

```
LOOP x_i DO
  P
END
```

Daraus kann eine if-Kontrollstruktur erstellt werden:

```
IF x_i THEN
  P
END
```

wird folgendermassen realisiert:

```
y := 0
LOOP x_i DO y := 1 END
LOOP y DO P END
```

### 8.3.2 Turing-Vollständigkeit

Für Turing-Vollständigkeit wird neben LOOP noch eine GOTO-Struktur benötigt.

## 9 Logik

### 9.1 Prädikate

- Aussagen über mathematische Objekte, wahr oder falsch
- Funktionen mit booleschen Rückgabewerten:

$P, Q(n), R(x, y, z)$

### 9.2 Verknüpfungen

- und:  $P \wedge Q$
- oder:  $P \vee Q$
- nicht:  $\neg P$
- Implikation  $P \Rightarrow Q = \neg P \vee Q$

### 9.3 Distributivgesetze

- $P \wedge (Q \vee R) \Leftrightarrow (P \wedge Q) \vee (P \wedge R)$
- $P \vee (Q \wedge R) \Leftrightarrow (P \vee Q) \wedge (P \vee R)$

### 9.4 De Morgan

- $\neg(P \wedge Q) \Leftrightarrow \neg P \vee \neg Q$
- $P \Rightarrow Q \Leftrightarrow (\neg Q \Rightarrow \neg P)$

### 9.5 Quantoren

- $\forall i \in \{1, \dots, n\}(P_i) =$  (Für alle  $i$  ist  $P_i$  wahr)
- $\exists i \in \{1, \dots, n\}(P_i) =$  (Es gibt ein  $i$  derart, dass  $P_i$  wahr ist)

#### 9.5.1 Morgan 2.0

«Nicht für alle» = «Es gibt einen Fall, für den nicht»

<sup>2</sup> Alan Turing (1936): On Computable Numbers, With an Application to the Entscheidungsproblem

$$\neg \forall i \in \{1, \dots, n\} (P_i)$$

$$\Leftrightarrow \neg(P_1 \wedge \dots \wedge P_n) \Leftrightarrow \neg P_1 \vee \dots \vee \neg P_n \Leftrightarrow \exists i \in \{1, \dots, n\} (\neg P_i)$$

## 10 Alphabet und Wort

- Alphabet:  $\Sigma$
- Wort: Ein  $n$ -Tupel in  $\Sigma^n = \Sigma \times \dots \times \Sigma$
- Menge aller Wörter:  $\Sigma^* = \{\varepsilon\} \cup \Sigma \cup \Sigma^2 \cup \dots = \bigcup_{k=0}^{\infty} \Sigma^k$

### 10.1 Wortlänge

- $|\varepsilon| = 0$
- $|01010|_0 = 3$
- $|(1201)^7| = 7 \cdot |1201| = 28, |w^n| = n \cdot |w|$

## 11 Beispiele für DEAs:

- Durch drei teilbare Zahlen
- Wörter mit einer geraden Anzahl  $a$
- Bedingungen an einzelne Zeichen, wie: Wörter, die mit einem  $a$  beginnen und genau ein  $b$  enthalten.

## 12 Kontextfreie Grammatik

### 12.1 Grammatik für Klammerausdrücke

$$\begin{aligned} A &\rightarrow \varepsilon \\ &\rightarrow AA \\ &\rightarrow (A) \end{aligned}$$

Oder (Kurzschreibweise):

$$A \rightarrow \varepsilon \mid AA \mid (A)$$

#### 12.1.1 Beispiel für Ableitung von Klammerausdruck

Wort  $((())()$  soll abgeleitet werden:

$$A \rightarrow (A) \rightarrow (AA) \rightarrow ((A)A) \rightarrow ((A)(A)) \rightarrow ((\varepsilon)(A)) \rightarrow ((\varepsilon)(\varepsilon)) \rightarrow ((())()$$

### 12.2 Parse Tree

**Definition 12.2.1:** Zwei Ableitungen eines Wortes  $w$  einer kontextfreien Sprache  $L(G)$  heißen äquivalent, wenn sie den gleichen Ableitungsbaum haben. Hat eine Sprache Wörter mit verschiedenen Ableitungen, heißt sie mehrdeutig (engl. ambiguous)

Ein Beispiel einer Grammatik, die nicht eindeutige Ableitungen hat, ist:

$$G = (\{S\}, \{0, 1\}, \{S \rightarrow 0S1 \mid 1S0 \mid SS \mid \varepsilon\}, S)$$

Ein Ausdruck 001011 produziert verschiedene Parsetrees (siehe Abbildung 17).

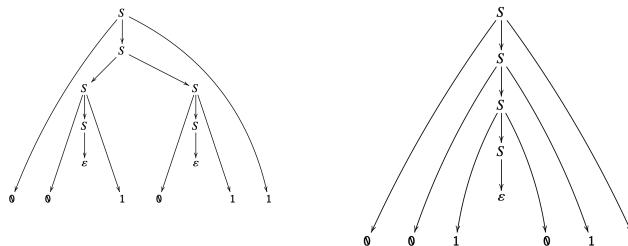


Abbildung 17: Nicht eindeutig ableitbare Grammatik

Von besonderer praktischer Bedeutung sind jedoch Grammatiken, die immer eindeutig ableitbar sind. Ein Beispiel solch einer Grammatik ist die expression-term-factor-Grammatik für einfache arithmetische Ausdrücke.

### 12.2.1 regex → CFG

Beispiel: Grammatik zum regulären Ausdruck  $(ab)^* \mid c$ :

Alternative:  $S \rightarrow U \mid C$

\*-Operation:  $U \rightarrow UP \mid \varepsilon$

Verkettung:  $P \rightarrow AB$

a:  $A \rightarrow a$

b:  $B \rightarrow b$

c:  $C \rightarrow c$

### 12.3 Arithmetische Ausdrücke

expression  $\rightarrow$  expression + term

$\rightarrow$  term

term  $\rightarrow$  term \* factor

$\rightarrow$  factor

factor  $\rightarrow$  (expression)

$\rightarrow$  N

N  $\rightarrow$  NZ

$\rightarrow$  Z

Z  $\rightarrow$  0|1|2|3|4|5|6|7|8|9

### 12.3.1 Anwendungen der Chomsky-Normalform

Gegeben: Grammatik  $G$  in Chomsky-Normalform

**Satz 12.3.1.1:** Ableitung eines Wortes  $w \in L(G)$  ist immer in  $2|w| - 1$  Regelanwendungen möglich.

Beweis:

- $|w| - 1$  Regeln der Form  $A \rightarrow BC$  um aus  $S$  ein Wort aus  $w$  Variablen zu erzeugen

- $|w|$  Regeln der Form  $A \rightarrow a$ , um das Wort  $w$  zu erzeugen

### 12.3.2 Deterministisches Parsen

**Aufgabe:**  $S \xrightarrow{*} w$

Kann das Wort  $w \in \Sigma^*$  von der Grammatik  $G = (V, \Sigma, R, S)$  erzeugt werden?

**Verallgemeinerte Aufgabe:**  $A \xrightarrow{*} w$

Kann das Wort  $w \in \Sigma^*$  aus der Variablen  $A$  in der Grammatik  $G = (V, \Sigma, R, s)$  abgeleitet werden?

### Deterministische Antwort:

Gesucht ist ein Programm mit der Signatur

```
boolean ableitbar(Variable a, String w);
```

welches entscheiden kann, ob ein Wort  $w$  aus der Variablen  $V$  ableitbar ist.

### 12.3.3 CYK-Ideen

1. Grammatik  $G = (V, \Sigma, R, S)$
2. Variable  $A \in V$
3. Wort  $w \in \Sigma^*$

**Frage:** Ist  $w$  aus  $A$  ableitbar? In Zeichen  $A \xrightarrow{*} w$

- Spezialfall  $w = \varepsilon: A \xrightarrow{*} \varepsilon \Leftrightarrow A \rightarrow \varepsilon \in R$
- Spezialfall  $|w| = 1: A \xrightarrow{*} w \Leftrightarrow A \rightarrow w \in R$
- Fall  $|w| > 1:$

$$A \xrightarrow{*} w \Rightarrow \exists \begin{cases} A \rightarrow BC \in R \\ w = w_1 w_2, w_i \in \Sigma^* \text{ mit } \\ \begin{cases} B \xrightarrow{*} w_1 \\ C \xrightarrow{*} w_2 \end{cases} \end{cases}$$

### 12.3.4 CYK-Algorithmus

```
boolean ableitbar(Variable A, String w) {
    if (w.length() == 0) {
        return A → ε ∈ R;
    }
    if (w.length() == 1) {
        return A → w ∈ R;
    }
    foreach Unterteilung w = w₁w₂ {
        foreach A → BC ∈ R {
            if (ableitbar(B, w₁) && ableitbar(C, w₂)) {
                return true;
            }
        }
    }
    return false;
}
```

## 12.4 Stack-Automaten

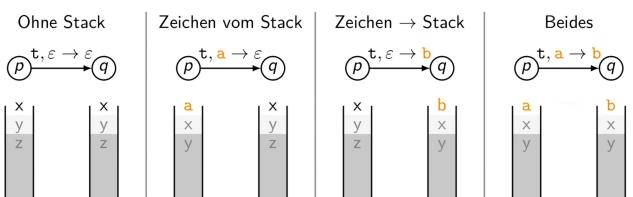


Abbildung 18: Stackübergänge

$a, b \rightarrow c$

- $a$ : vom Input
- $b$ : vom Stack entfernen (Bedingung)
- $c$ : auf den Stack

Beispiel:  $\{0^n 1^n \mid n \geq 0\}$

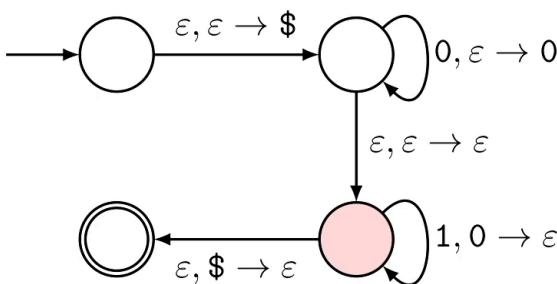


Abbildung 19: Stackautomat

### Definition 12.4.1 (Stackautomat):

$$P = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

1.  $Q$ : Zustände
2.  $\Sigma$ : Eingabe-Alphabet
3.  $\Gamma$ : Stack-Alphabet
4.  $\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$ , Regeln
5.  $q_0 \in Q$ : Startzustand
6.  $F \subset Q$ : Akzeptierzustände

Die Regeln hängen ab vom aktuellen Zustand, vom aktuellen Inputzeichen und vom Zeichen, das zuoberst auf dem Stack liegt.

$\delta$ : Funktion mit drei Inputs (aktueller Zustand, Input-Zeichen, oberstes Zeichen auf dem Stack) und gibt ein Tupel zurück mit neuem Zustand und neuem obersten Zeichen auf dem Stack.

#### Beachte:

- Immer nicht-deterministisch
- $\Gamma \neq \Sigma$  möglich (z.B.  $\$$ -Symbol auf den Stack)

### 12.4.1 Visualisierung eines Stacks

Beispiel:

Grammatik:

$$\begin{aligned} S &\rightarrow 0S1 \\ &\rightarrow \varepsilon \end{aligned}$$

Input: 0 0 0 1 1

1.  $\$$ -Symbol auf den Stack (um später zu prüfen, ob der Stack leer ist)
2. Variable  $S$  auf den Stack
3. Input 0 matcht nicht  $\rightarrow$  Variable  $S \rightarrow 0S1$  (1 auf den Stack, dann  $S$ , dann 0)
4. 0 matcht  $\rightarrow$  0 vom Stack entfernen und weiter
5. 0 matcht nicht  $S \rightarrow 0S1$

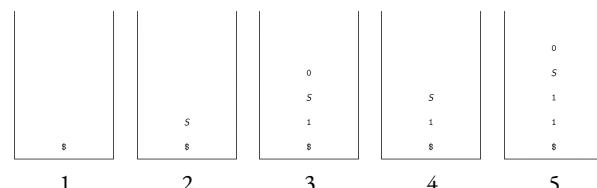


Abbildung 20: Stack-Visualisierung

#### Zustandsdiagramm:

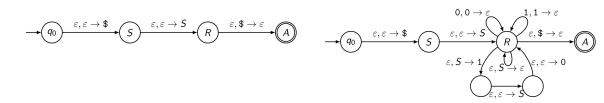


Abbildung 21: Zustandsdiagramm erstellen

## 12.4.2 Grammatik → Stackautomat

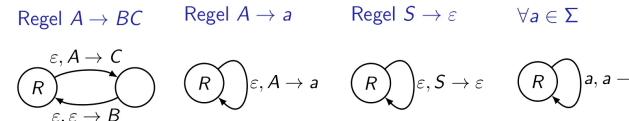


Abbildung 22: CNF zu Stackautomat

## 12.4.3 Backus-Naur-Form (BNF)

Grammatik:

$$\begin{aligned} S &\rightarrow \varepsilon \\ &\rightarrow 0S1 \end{aligned}$$

BNF:

$\langle \text{string} \rangle ::= '' \mid 0 \langle \text{string} \rangle 1$

Beispiel (Expression-Term-Factor):

```
<expression> ::= <expression> + <term> | <term>
<term> ::= <term> * <factor> | <factor>
<factor> ::= ( <expression> ) | <number>
<number> ::= <number> <digit> | <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

## 12.5 PDA zu CFG

### Variablen

Wörter beschreiben Pfade durch den Automaten:

Variable  $A_{pq} =$  Wörter, die von  $p$  nach  $q$  führen mit leerem Stack

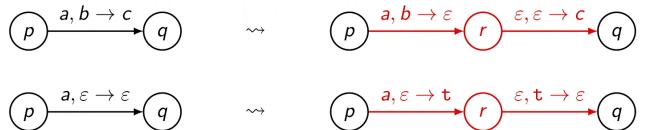
### Regeln

Regeln beschreiben, wie sich Wege zerlegen lassen.

$$A_{pq} \rightarrow A_{pr} A_{rq}$$

### 12.5.1 Stackautomat standardisieren

1. Nur ein Akzeptierzustand
  - $\forall q \in F: \text{Übergang in neuen Akzeptierzustand } q_a$
2. Stack leeren:
  - Markierungszeichen zu Beginn auf den Stack
  - Am Schluss:  $\varepsilon, . \rightarrow \varepsilon$  und dann  $\varepsilon, \$ \rightarrow \varepsilon$  von  $q_a$  nach  $q'_a$
3. Jeder Übergang legt entweder ein Zeichen auf den Stack oder entfernt eines



## 12.5.2 Regeln

$A_{pq}$  wird, falls sich der Stack dazwischen nicht leert, zu folgendem:

## 12.6 Vergleich von Maschinen

**Definition 12.6.1:** Eine TM  $M_1$  ist «leistungsfähiger» als eine TM  $M_2$ , wenn  $M_1$  die Maschine  $M_2$  simulieren kann

$$M_2 \leq_S M_1$$

$M_2$  ist simulierbar auf  $M_1$ .

Beispiele:

- TM  $\leq_S$  Minecraft
- 8-Bit CPU  $\leq_S$  Minecraft  $\leq_S$  TM

## 12.7 Unendlichkeit

Die Mengen  $\mathbb{N}$  und  $\mathbb{R}$  sind nicht gleich mächtig, da es keine bijektive Abbildung  $\mathbb{N} \rightarrow \mathbb{R}$  gibt.

**Definition 12.7.1:** Mengen  $A$  und  $B$  heissen gleich mächtig,  $A \simeq B$ , wenn es eine Bijektion  $A \rightarrow B$  gibt.

**Definition 12.7.2:** Eine Menge  $A$  heisst unendlich, wenn sie gleich mächtig wie eine echte Teilmenge ist.

**Definition 12.7.3:**  $A$  heisst abzählbar unendlich, wenn  $A \simeq \mathbb{N}$ , d.h.  $A$  gleich mächtig wie  $\mathbb{N}$  ist.

**Satz 12.7.1:** Die Potenzmenge  $P(A)$  einer abzählbar unendlichen Menge  $A$  ist immer überabzählbar unendlich.

**Satz 12.7.2:** Die Vereinigung von endlich vielen abzählbaren Mengen ist abzählbar. Das kartesische Produkt  $A \times B$  zweier abzählbaren Mengen  $A, B$  ist abzählbar.

## Anwendungen:

- Abzählbar unendlich:  $\Sigma^*$ , Menge aller DEAs/NEAs/PDAs/CFGs
- Überabzählbar unendlich: Menge aller Sprachen  $P(\Sigma^*)$

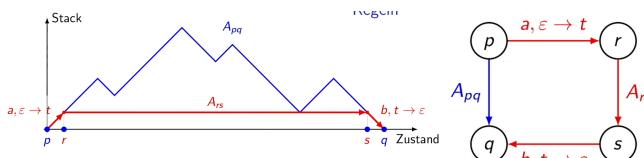


Abbildung 23: Regel, falls der Stack dazwischen nie leer wird

$$A_{pq} \rightarrow aA_{rs}b$$

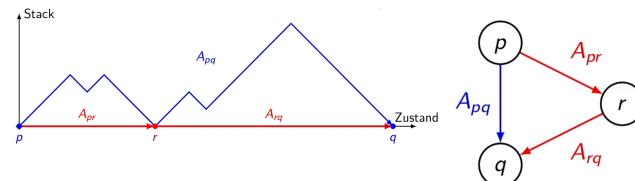


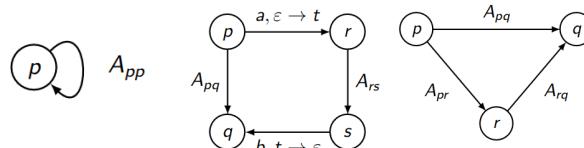
Abbildung 24: Falls der Stack zwischendurch leer wird

$$A_{pq} \rightarrow A_{pr}A_{rq}$$

### 12.7.1 Grammatik ablesen

Ausgangspunkt: Standardisierte Grammatik mit Startzustand  $q_0$  und  $F = \{q_a\}$ .

1. Startvariable  $A_{q_0, q_a}$
2. Regeln:

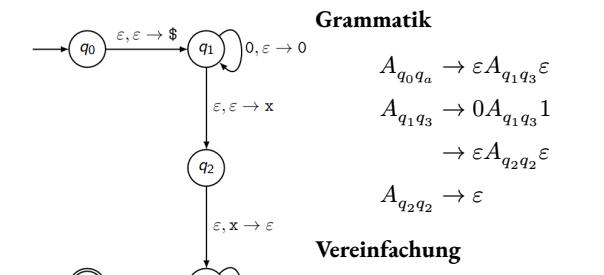


$$A_{pp} \rightarrow \epsilon$$

$$A_{pp} \rightarrow aA_{rs}b$$

$$A_{pp} \rightarrow A_{pr}A_{rq}$$

Beispiel (PDA zu CFG):



## Vereinfachung

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow 0S1 \\ &\quad \rightarrow \epsilon \end{aligned}$$

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

**Definition 12.7.1.1:** Eine Turing-Maschine ist ein 7-Tupel  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ , wobei:

1.  $Q$  heisst die Menge der Zustände
2.  $\Sigma$  heisst das Inputalphabet, es enthält das Blank-Zeichen nicht<sup>3</sup>
3.  $\Gamma$  ist das Bandalphabet, es enthält das Blank-Zeichen und  $\Sigma \subset \Gamma$
4.  $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$  ist die Übergangsfunktion

## 12.8 Spielregeln

1. Der Speicher ist unbegrenzt
2. In jeder Zelle steht genau ein Zeichen
3. Es ist immer nur eine Speicherzelle einsehbar
4. Der Inhalt der aktuellen Zelle kann beliebig verändert werden
5. Bewegung immer nur eine Zelle nach links oder rechts
6. Kein weiterer Speicher (nur die Zustände eines endlichen Automaten)

## 12.9 Zustandsdiagramm

$$\delta(p, a) = (q, b, L): \xrightarrow{p \rightarrow b, L} q$$

## 12.10 Von einer TM erkannte Sprache

$$L(M) = \{w \in \Sigma^* \mid M \text{ akzeptiert } w\}$$

**Definition 12.10.1:** Ein Entscheider ist eine Turing-Maschine, die auf jedem Input  $w \in \Sigma^*$  anhält. Eine Sprache heisst entscheidbar, wenn ein Entscheider sie erkennt.

<sup>3</sup> Wäre das Blank-Zeichen in  $\Sigma$ , könnte man das leere Band nicht vom Input unterscheiden.

Eine Turing-entscheidbare Sprache ist auch Turing-erkennbar. Die Eigenschaft «Turing-entscheidbar» unterscheidet sich von der Eigenschaft «Turing-erkennbar» nur dadurch, dass bei einer entscheidbaren Sprache die Turing-Maschine auf jedem beliebigen Input anhalten muss, während bei einer nur erkennbaren Sprache einzelne Input-Wörter auch dazu führen können, dass die Turing-Maschine endlos weiterrechnet.

## 12.11 Turing Maschinen und moderne Computer

### Turing Maschine

- 1. Zustände  $Q$
- 2. Band, unendlich grosser Speicher
- 3. Schreib-/Lesekopf
- 4. Anhalten,  $q_{\text{accept}}$  und  $q_{\text{reject}}$
- 5. Problemspezifisch

### Moderner Computer

- 1. Zustände der CPU: Statusbits, Registerwerte
- 2. Virtueller Speicher: praktisch unbegrenzt
- 3. Adress-Register, Programm-Zähler
- 4. `exit(EXIT_SUCCESS)`, `exit(EXIT_FAILURE)`
- 5. Kann beliebige Programme ausführen

## 12.12 Aufzähler

**Definition 12.12.1** (Aufzähler): Ein Aufzähler ist eine TM, die alle akzeptierbaren Wörter mit einem Drucker ausdrückt.

**Definition 12.12.2** (Rekursiv aufzählbare Sprache): Eine Sprache  $L$  heisst rekursiv aufzählbar, wenn es einen Aufzähler gibt, der sie aufzählt.

Aufzählbare Sprache  $\Leftrightarrow$  Turing-erkennbare Sprache.

## 12.13 Vergleich von Maschinen

**Definition 12.13.1:** Eine TM  $M_1$  ist «leistungsfähiger» als eine TM  $M_2$ , wenn  $M_1$  die Maschine  $M_2$  simulieren kann

$$M_2 \leq_S M_1$$

$M_2$  ist simulierbar auf  $M_1$ .

Beispiele:

- $\text{TM} \leq_S \text{Minecraft}$
- 8-Bit CPU  $\leq_S \text{Minecraft} \leq_S \text{TM}$

## 12.14 Unendlichkeit

Die Mengen  $\mathbb{N}$  und  $\mathbb{R}$  sind nicht gleich mächtig, da es keine bijektive Abbildung  $\mathbb{N} \rightarrow \mathbb{R}$  gibt.

**Definition 12.14.1:** Mengen  $A$  und  $B$  heissen gleich mächtig,  $A \simeq B$ , wenn es eine Bijektion  $A \rightarrow B$  gibt.

**Definition 12.14.2:** Eine Menge  $A$  heisst unendlich, wenn sie gleich mächtig wie eine echte Teilmenge ist.

**Definition 12.14.3:**  $A$  heisst abzählbar unendlich, wenn  $A \simeq \mathbb{N}$ , d.h.  $A$  gleich mächtig wie  $\mathbb{N}$  ist.

**Satz 12.14.1:** Die Potenzmenge  $P(A)$  einer abzählbar unendlichen Menge  $A$  ist immer überabzählbar unendlich.

**Satz 12.14.2:** Die Vereinigung von endlich vielen abzählbaren Mengen ist abzählbar. Das kartesische Produkt  $A \times B$  zweier abzählbaren Mengen  $A, B$  ist abzählbar.

### Anwendungen:

- Abzählbar unendlich:  $\Sigma^*$ , Menge aller DEAs/NEAs/PDAs/CFGs
- Überabzählbar unendlich: Menge aller Sprachen  $P(\Sigma^*)$

## 12.15 Berechenbare Zahlen

**Definition 12.15.1** (Berechenbare Zahl): Eine Zahl  $w$  heisst berechenbar, wenn es eine Turingmaschine  $M$  gibt, die auf leerem Band startet und auf dem Band nacheinander die Stellen der Zahl ausgibt.

Eine Zahl  $w$  ist somit berechenbar, wenn es eine Turing-Maschine gibt, die sie berechnet.

Beispiel:  $\pi, e, \sqrt{2}, \gamma, \varphi = \frac{\sqrt{5}-1}{2}$

### 12.15.1 Wieviele berechenbare Zahlen gibt es?

Sind alle reellen Zahlen berechenbar?

1. Es gibt höchstens so viele berechenbare Zahlen wie Turing-Maschinen
2. Die Menge der Turing-Maschinen ist abzählbar unendlich
3. Die Menge der reellen Zahlen  $\mathbb{R}$  ist überabzählbar unendlich
4.  $\Rightarrow$  fast alle reelle Zahlen sind nicht berechenbar

## 12.16 Hilberts 10. Problem

Gibt es ganzzahlige Lösungen für Polynomgleichungen?

$$\begin{aligned}x^2 - y &= 0 \\x^n + y^n &= z^n\end{aligned}$$

Eine TM, die verschiedene Möglichkeiten ausprobiert, ist hierfür nicht geeignet; falls es keine Lösung gibt, würde die TM nicht anhalten.  $\Rightarrow$  Es bräuchte einen Entscheider. Dieses Problem wurde jedoch 1970 von Yuri Matiyasevich als nicht entscheidbar bewiesen.

## 12.17 Sprachprobleme

Ein normales Problem soll zunächst in eine Ja/Nein-Frage übersetzt werden.

Beispiel: Problem: Finden sie eine Lösung der quadratischen Gleichung:

$$x^2 - x - 1 = 0$$

Dies ist jedoch nicht wirklich als Sprache formuliert. Wir können folgende Formulierung postieren:

Sei  $L$  die Sprache bestehend aus Wörtern der Form

$$w = a = a, b = b, c = c, x = x$$

wobei  $a, b, c, x$  Dezimaldarstellungen von Zahlen sind. Ein Wort gehört genau dann zur Sprache  $w$ , wenn  $ax^2 + bx + c = 0$  gilt. Ist  $a = 1, b = -1, c = -1, x = 3 \in L$ ?

Beispiel: Gegeben ist eine natürliche Zahl  $n$ , berechne die ersten 10 Stellen der Dezimaldarstellung von  $\sqrt{n}$ .

Dies ist wieder kein normales Sprachproblem  $\rightarrow$  Als Entscheidungsproblem mit Ja/Nein-Antwort formulieren:

Sind die ersten 10 Stellen der Dezimaldarstellung von  $\sqrt{2} = 1.414213562$ ?

**Als Sprache formuliert:** Sei  $L$  die Sprache bestehend aus Zeichenketten der Form  $n, x$  wobei  $n$  die Dezimaldarstellung einer natürlichen Zahl ist und  $x$  die ersten 10 Stellen einer Dezimalzahl. Gilt  $2, 1.414213562 \in L$ ?

## 12.17.1 $\varepsilon$ -Akzeptanzproblem für endliche Automaten

**Problem:** Kan der endliche Automat  $A$  das leere Wort akzeptieren?

**Als Sprachproblem:** Ist die Sprache  $L = \{\langle A \rangle \mid \varepsilon \in L(A)\}$  entscheidbar?

### Entscheidungsalgorithmus:

1. Wandle  $A$  in einen DEA um
2. Ist der Startzustand ein Akzeptierzustand  $q_0 \in F$ ?

## 12.17.2 Gleichheitsproblem für DEAs

**Problem:** Akzeptieren die endlichen Automaten  $A_1$  und  $A_2$  die gleiche Sprache,  $L(A_1) = L(A_2)$ ?

**Sprachproblem:** Ist die Sprache  $L = \{\langle A_1, A_2 \rangle \mid L(A_1) = L(A_2)\}$  entscheidbar?

**Entscheidungsalgorithmus:**

1. Wandle  $A_1$  in einen minimalen Automaten  $A'_1$  um
2. Wandle  $A_2$  in einen minimalen Automaten  $A'_2$  um
3. Ist  $A'_1 = A'_2$ ?

## 12.17.3 Akzeptanzproblem für DEAs

**Problem:** Akzeptiert der endliche Automat  $A$  das Wort  $w$ ?

**Sprachproblem:** Ist die Sprache  $L = \{\langle A, w \rangle \mid w \in L(A)\}$  entscheidbar?

**Entscheidungsalgorithmus:**

1. Wandle  $A$  in einen DEA  $A'$  um
2. Simuliere  $A'$  mit Hilfe einer TM
3. Hält die Turing-Maschine im Zustand  $q_{\text{accept}}$ ?

## 12.17.4 Akzeptanzproblem für CFGs

**Problem:** Kann das Wort  $w$  aus der Grammatik  $G$  produziert werden?

**Als Sprachproblem:** Ist die Sprache  $L = \{\langle G, w \rangle \mid w \in L(G)\}$  entscheidbar?

**Entscheidungsalgorithmus:**

1. Kontrollieren, dass  $\langle G \rangle$  wirklich eine Grammatik beschreibt
2. Grammatik in Chomsky-Normalform bringen
3. Mit dem CYK-Algorithmus prüfen, ob  $w$  aus  $G$  produziert werden kann

## 12.18 Nicht entscheidbare Probleme

**Satz 12.18.1** (Alan Turing):  $A_{\text{TM}}$  ist nicht entscheidbar.

Beweis: Konstruiere aus dem Entscheider  $H$  für  $A_{\text{TM}}$  eine Maschine  $D$  mit Input  $\langle M \rangle$ .

1. Lasse  $H$  auf Input  $\langle M, \langle M \rangle \rangle$  laufen
2. Falls  $H$  akzeptiert:  $q_{\text{reject}}$
3. Falls  $H$  verwirft:  $q_{\text{accept}}$

Wende jetzt  $D$  auf  $\langle D \rangle$  an:

- $D(\langle D \rangle)$  akzeptiert  $\Leftrightarrow D$  verwirft  $\langle D \rangle$   
 $D(\langle D \rangle)$  verwirft  $\Leftrightarrow D$  akzeptiert  $\langle D \rangle$

Widerspruch!

## 12.19 Reduktion

Berechenbare Abbildung  $f : \Sigma^* \rightarrow \Sigma^*$  so, dass

$$w \in A \Leftrightarrow f(w) \in B$$

Notation:  $f : A \leq B$ , « $A$  leichter entscheidbar als  $B$ »

Entscheidbarkeit:  $B$  entscheidbar,  $f : A \leq B \Rightarrow A$  entscheidbar

Beweis:  $H$  ein Entscheider für  $B$ , dann ist  $H \circ f$  ein Entscheider für  $A$ . ■

Folgerung:  $A$  nicht entscheidbar,  $A \leq B \Rightarrow B$  nicht entscheidbar.

### 12.19.1 Reduktionsbeispiele

In folgenden Beispielen ist  $M$  eine Maschine, die ein Wort  $w$  entweder akzeptiert oder verwirft. Wie wir bewiesen haben, ist es unmöglich, einen Entscheider für das Akzeptanzproblem  $A_{\text{TM}}$  zu konstruieren.

Beispiel (Reduktion für das spezielle Halteproblem):

Programm  $S$ :

1. Führe  $M$  auf  $w$  aus
2.  $M$  hält in  $q_{\text{accept}}$ : akzeptiere
3.  $M$  hält in  $q_{\text{reject}}$ : Endlosschleife

Wenn  $H$  ein Entscheider für  $\text{HALT}_{\epsilon_{\text{TM}}}$  wäre, könnte man daraus einen Entscheider für  $A_{\text{TM}}$  konstruieren:

1. Konstruiere das Programm  $S$
2. Wende  $H$  auf  $\langle S \rangle$  an

Beispiel (Reduktion für das Leerheitsproblem  $A_{\text{TM}} \leq E_{\text{TM}}$ ): Ist die Sprache  $L(M)$  leer?  $\rightarrow \overline{E}_{\text{TM}}$  ist  $L(M) \neq \emptyset$

Programm  $S$  mit Input  $u$ :

1.  $M$  auf  $w$  laufen lassen
2.  $M$  akzeptiert  $w$ :  $q_{\text{accept}}$
3. Andernfalls:  $q_{\text{reject}}$

$$M \text{ akzeptiert } w \Leftrightarrow S \text{ akzeptiert } L(S) = \Sigma^* \neq \emptyset$$

Wenn  $H$  ein Entscheider für  $E_{\text{TM}}$  wäre, könnte man daraus einen Entscheider für  $A_{\text{TM}}$  konstruieren:

1. Konstruiere das Programm  $S$
2. Wende  $H$  auf  $\langle S \rangle$  an

Beispiel (Regularitätsproblem  $A_{\text{TM}} \leq \text{REGULAR}_{\text{TM}}$ ): Ist die Sprache  $L(M)$  regulär?

Programm  $S$  mit Input  $u$ :

1.  $u \notin \{0^n 1^n \mid n \geq 0\} \rightarrow q_{\text{reject}}$
2.  $M$  auf  $w$  laufen lassen
3.  $M$  akzeptiert  $w$ :  $q_{\text{accept}}$

$$4. q_{\text{reject}}$$

$M$  akzeptiert  $w \Leftrightarrow S$  akzeptiert  $\{0^n 1^n \mid n \geq 0\}$ , nicht regulär

$M$  akzeptiert  $w$  nicht  $\Leftrightarrow S$  akzeptiert  $\emptyset$ , regulär

Wäre  $H$  ein Entscheider für  $\text{REGULAR}_{\text{TM}}$ , könnte man daraus einen Entscheider für  $A_{\text{TM}}$  konstruieren:

1. Konstruiere das Programm  $S$
2. Wende  $H$  auf  $\langle S \rangle$  an

### 12.19.2 Folgerungen aus $P = NP$

1. Für jedes Problem in  $NP$  gibt es einen polynomiellen Algorithmus
2. Es gibt keine «schwierigen» Probleme
3. Mit Moore's Law lässt sich jedes Problem «lösen»
4. Es gibt keine sichere Kryptographie

### 12.19.3 Folgerungen aus $P \neq NP$

1.  $NP$ -vollständige Probleme haben nicht skalierende Lösungen
2. Moore's Law hilft nicht in  $NP \setminus P$

### 12.19.4 NP-vollständig

Eine entscheidbare Sprache  $B$  heißt  $NP$ -vollständig, wenn sich jede Sprache  $A$  in  $NP$  polynomiell auf  $B$  reduzieren lässt:

$$A \leq_P B, \forall A \in NP$$

Wenn ein Problem  $NP$ -vollständig ist:

- Lösung braucht typischerweise exponentielle Zeit
- Korrektheit der Lösung ist leicht (in polynomieller Zeit) zu prüfen
- Andernfalls wären Tests schon exponentiell und somit in Software nicht sinnvoll

Falls  $P \neq NP$ , dann können  $NP$ -vollständige Probleme nicht in polynomieller Zeit gelöst werden.

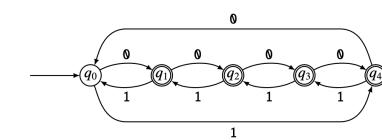
## 13 Beispielprüfung

1. Finden Sie einen deterministischen endlichen Automaten, der die Sprache

$$L = \{w = \{0, 1\}^* \mid |w|_0 \not\equiv |w|_1 \mod 5\}$$

bestehend aus Wörtern, deren Anzahl von Nullen und Einsen verschiedenen Rest bei Teilung durch fünf haben.

*Lösung.* Das Zustandsdiagramm



beschreibt diesen Automaten. Der Zustand  $q_k$  codiert den Fünferrest  $k$  der Differenz  $|w|_0 - |w|_1$ . ○

*Bewertung.* Korrekte Idee für ein Konstruktionsprinzip (I) 1 Punkt, Startzustand (S) 1 Punkt, Akzeptierzustände (A) 1 Punkt, Automat ist deterministisch (D) 1 Punkt, Reihenfolge von 0 und 1 spielt keine Rolle (R) 1 Punkt, Automat akzeptiert genau die Sprache  $L$  (L) 1 Punkt.

2. Sei  $f: \mathbb{N} \rightarrow \mathbb{N}$  eine streng monoton wachsende Funktion. Eine solche Funktion nimmt beliebig grosse Werte an und es gilt  $f(n) \geq n$  für alle  $n \in \mathbb{N}$ . Ist die Sprache

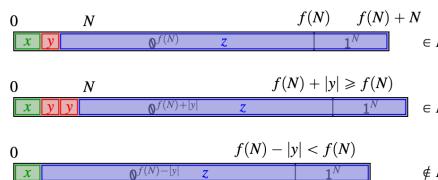
$$L = \{w \in \Sigma^* \mid |w|_0 \geq f(|w|_1)\}$$

über dem Alphabet  $\Sigma = \{\emptyset, 1\}$  regulär?

**Hinweis.** Die Sprache  $L$  besteht aus Wörtern, die immer mehr  $\emptyset$  als 1 enthalten. Es gilt aber ausserdem, dass die Anzahl der Nullen mindestens so schnell zunimmt wie die Anzahl der Einsen.

**Lösung.** Die Sprache ist nicht regulär, wie man mit dem Pumping-Lemma für reguläre Sprachen zeigen kann.

1. Annahme:  $L$  ist regulär.
2. Nach dem Pumping Lemma gibt es die Pumping Length  $N$ .
3. Wir wählen das Wort  $\emptyset^{f(N)} 1^N \in L$ .
4. Nach dem Pumping Lemma lässt sich das Wort  $w$  in drei Teile  $w = xyz$  zerlegen mit  $|xy| \leq N$  und  $|y| > 0$ . Weil  $f(N) \geq N$  enthält  $y$  ausschliesslich Nullen.



5. Beim Aufpumpen wird die Anzahl der Nullen grösser, das bedeutet aber nicht, dass das Wort nicht mehr in der Sprache enthalten ist, da nur  $|w|_0 \geq f(|w|_1)$  verlangt ist. Beim Abpumpen wird allerdings die Anzahl der Nullen kleiner,  $xz$  enthält nur noch  $f(N) - |y|$  Nullen. Damit haben wir

$$|xz|_0 = f(N) - |y| < f(N) = |xz|_1,$$

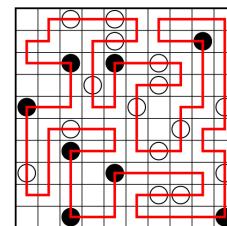
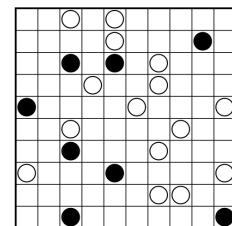
das Wort  $xz$  ist also nicht mehr in  $L$ , im Widerspruch zur Aussage des Pumping Lemmas.

6. Der Widerspruch zeigt, dass Annahme,  $L$  sie regulär, nicht haltbar ist. Also ist  $L$  nicht regulär.  $\circ$

**Bewertung.** Pumping-Lemma-Beweis: jeder Schritt ein Punkt: Annahme regulär (PL) 1 Punkt, Pumping Length (N) 1 Punkt, Wort (W) 1 Punkt, Aufteilung (A) 1 Punkt, Widerspruch beim Pumpen (P) 1 Punkt, Schlussfolgerung (S) 1 Punkt.

3. Masyu (japanisch ましゅ) ist ein Rätsel, welches auf einem  $n \times n$ -Spielfeld gespielt wird. In einigen Feldern sind schwarze und weisse Kreise eingezeichnet. Der Spieler muss einen geschlossenen Weg durch das Spielfeld finden, welcher folgenden Regeln genügt:

1. Der Weg betritt und verlässt ein Quadrat immer über eine Kante.
2. Der Weg darf sich selbst nicht schneiden.
3. Weisse Kreise werden in gerader Linie durchquert, aber der Weg muss vor oder nach dem weissen Kreis um  $90^\circ$  abbiegen.
4. Schwarze Kreise liegen auf einer Kurve, die um  $90^\circ$  abbiegt, die Felder neben den schwarzen Kreisen werden in gerade Linie durchlaufen.



Kann eine nichtdeterministische Turing-Maschine in polynomieller Zeit entscheiden, ob ein Masyu-Rätsel eine Lösung hat?

**Lösung.** Die Frage ist sicher entscheidbar, indem man alle endlich vielen möglichen Abfolgen von Feldern daraufhin testet, ob sie Pfade sind und den vier Bedingungen genügen. Dafür ist jedoch exponentielle Zeit nötig.

Eine nichtdeterministische Turing-Maschine kann die Frage in polynomieller Zeit entscheiden, wenn es einen polynomiellen Verifizierer gibt. Als Zertifikat wird der gesuchte Lösungspfad gefordert, eine Abfolge von höchstens  $n^2$  Feldern. Damit sind folgende Verifikationen vorzunehmen:

Regel	Verifikation	Laufzeit
1	Jedes Feld des Pfades ist ein Nachbarfeld des vorangegangenen Feldes.	$O(n^2)$
2	Jedes Element des Pfades mit jedem anderen Element vergleichen und kontrollieren, dass alle Elemente verschieden sind.	$O(\frac{n(n-1)}{2})$
3a	Kontrollieren, ob die Nachbarn eines Feldes mit weissem Kreis diametral gegenüberliegen.	$O(n^2)$
3b	Kontrollieren, ob auf einem der Nachbarfelder eines Feldes mit weissem Kreis eine $90^\circ$ -Abbiegung erfolgt	$O(n^2)$
4	Kontrolliere, ob auf jedem Element des Pfades mit einem schwarzen Feld die Nachbarelemente eine $90^\circ$ -Abbiegung bilden.	$O(n^2)$
Total		$O(n^3)$

Damit ist gezeigt, dass der Verifizierer polynomielle Laufzeit hat. Somit ist Masyu in NP.  $\circ$

**Diskussion.** Erich Friedmann hat 2002 bewiesen, dass Masyu NP-vollständig ist.

**Bewertung.** Entscheidbarkeit (E) 1 Punkt, Prinzip Verifizierer (V) 1 Punkt, Zertifikat spezifiziert (Z) 1 Punkt, Laufzeitabschätzung (L) 2 Punkte, Schlussfolgerung polynomieller Verifizierer (S) 1 Punkt.

**Lösung.** Es ist nicht möglich, ein solches Tool zu schreiben, wie man mit dem Satz von Rice zeigen kann. Die Eigenschaft **SORTIERT**, die der Output haben muss, ist, dass die Wörter im Output sortiert sind. Dies ist eine nichttriviale Eigenschaft, den von den Sprachen

$$\begin{aligned} L_1 &= \{ab\} \\ L_2 &= \{ba\} \end{aligned}$$

besteht die eine aus sortierten Wörtern, die andere nicht. Nach dem Satz von Rice ist es nicht möglich, zu entscheiden, ob die akzeptierte Sprache einer Turing-Maschine die Eigenschaft **SORTIERT** hat.  $\circ$

**Bewertung.** Erkenntnis, dass es um eine Eigenschaft der erkannten Sprache geht (Wörter sind sortiert) (E) 1 Punkt, Bezug zum Satz von Rice (R) 1 Punkt, zwei Beispilsprachen (L) 2 Punkte, Schlussfolgerung (S) 2 Punkte.

5. Ist die Sprache

$$L = \{\emptyset^j \emptyset^k \mid j > 0 \wedge k > 0\}$$

kontextfrei? Wenn ja, geben Sie eine kontextfreie Grammatik in Chomsky-Normalform dafür an.

**Lösung.** Die Sprache ist kontextfrei. Die Grammatik muss Wörter der Form  $\emptyset^l \emptyset^k$ ,  $l > 0$ , mit Wörtern der Form  $1^l \emptyset^k$ ,  $l > 0 \wedge k > 0$ , verketten. Für Wörter der ersten Art haben wir die Grammatik

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow \emptyset A1 \mid \emptyset 1 \\ B &\rightarrow 1B \mid B1 \mid 10 \end{aligned}$$

verwenden. Damit finden wir die Grammatik

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow NU \mid NE \\ U &\rightarrow AE \\ B &\rightarrow EB \mid BN \mid EN \\ N &\rightarrow \emptyset \\ E &\rightarrow 1 \end{aligned}$$

Diese Grammatik hat nicht Chomsky-Normalform. Aber ihre Startvariable kommt auf der rechten Seite nicht vor und sie hat weder  $\epsilon$ -Regeln noch Unit Rules. Es bleibt daher nur noch, die Terminal-symbole und die Dreierregeln zu "flicken":

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow NU \mid NE \\ U &\rightarrow AE \\ B &\rightarrow EB \mid BN \mid EN \\ N &\rightarrow \emptyset \\ E &\rightarrow 1 \end{aligned}$$

Damit hat die Grammatik Chomsky-Normalform.  $\circ$

**Bewertung.** Grammatik (3 Punkte) korrekt formuliert (G) 1 Punkt, Mehr 1 wie  $\emptyset$  (A) 1 Punkt, Mindestens eine  $\emptyset$  im zweiten Teil (N) 1 Punkt. CNF Umwandlung (3 Punkte): neue Startvariable (S) 1 Punkt, unit rules (U) 1 Punkt, Dreierregeln und Terminal-symbole (T) 1 Punkt.

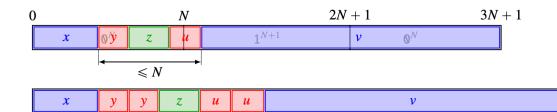
6. Ist die Sprache

$$L = \{\emptyset^j \emptyset^k \mid j > i\}$$

kontextfrei?

**Lösung.** Die Sprache ist nicht kontextfrei, wie man mit dem Pumping Lemma für kontextfreie Sprachen zeigen kann.

1. Annahmen:  $L$  ist kontextfrei.
2. Nach dem Pumping Lemma für kontextfreie Sprachen gibt es die Pumping Length  $N$ .
3. Wir wählen das Wort  $w = \emptyset^N 1^{N+1} \emptyset^N$ .
4. Das Wort lässt sich in fünf Teile  $xyzuv$  zerlegen:



5. Da  $|yzu| \leq N$  kann  $yzu$  nur Nullen aus einem der beiden Nullenblöcke enthalten. Beim Pumpen nimmt die Zahl der Nullen im betroffenen Block zu, jedoch nicht im anderen, das gepumpte Wort liegt daher nicht mehr in der Sprache.

Es aber noch eine zweite Situation, die beim Pumpen eintreten könnte. Es könnten in  $y$  und  $u$  nämlich nur Einsen enthalten sein. In diesem Fall würde beim Pumpen die Zahl der Einsen zunehmen, was der Sprachdefinition nicht widerspricht. Indem man  $y$  und  $u$  abpumpt, würde aber die Zahl der Einsen von  $N+1$  auf einen Wert  $\leq N$  reduziert, nach Voraussetzung  $w \in L$  muss aber die Anzahl der Einsen  $> N$  sein. Damit tritt auch in diesem Fall ein Widerspruch ein.

6. Dieser Widerspruch zu der Aussage des Pumping Lemma zeigt, dass die Annahme,  $L$  sei kontextfrei, nicht haltbar ist.  $\circ$

**Bewertung.** Annahme kontextfrei (PL) 1 Punkt, Pumping Length  $N$  (N) 1 Punkt, Wahl eines geeigneten Wortes (W) 1 Punkt, Aufteilung in 5 Teile (A) 1 Punkt, Widerspruch beim Pumpen (W) 1 Punkt, Schlussfolgerung (S) 1 Punkt.

7. Das Miniatur Wunderland<sup>1</sup> in Hamburg beherbergt die mit 16138 m Gleislänge auf 1545 m<sup>2</sup> Anlagefläche grösste Modelleisenbahnanlage der Welt. Auf der Anlage sind 1120 Züge, 10250 Autos und 47 Flugzeuge unterwegs. Die ganze Anlage mit 1392 Signalen, 3454 Weichen und über 497000 LEDs wird von 50 Computern gesteuert.

Um die Betriebssicherheit im Dauerbetrieb zu gewährleisten, müssen die Gleise ständig sauber gehalten werden. Dafür stehen Reinigungsziege zur Verfügung, die in regelmässigen Abständen auf vorprogrammierten Routen über die Anlage geführt werden. Natürlich sind auch noch viele andere Züge ebenfalls auf dem Schienennetz unterwegs, allerdings nicht ständig, die Loks sind Spielzeug, welches nicht für den Dauerbetrieb ausgelegt ist, sie müssen daher regelmässig Pausen zum Abkühlen einlegen. Die zeitliche Planung der Zugfahrten soll so gestaltet sein, dass innerhalb vorgegebener

<sup>1</sup><https://www.miniatur-wunderland.de/>

Zeitintervalle jeder Zug seine Fahrt durchführt. Die Zugfahrten der Reinigungsziege sind besonders wichtig. Fallen diese Fahrten aus oder finden Sie zu spät statt, kann es zu Ausfällen kommen, die den Besuch des Miniatur Wunderlandes weniger attraktiv machen. Man kann sogar versuchen, den dadurch verursachten Schaden zu beziffern. Da die Reinigungsziege nicht so interessant sind wie die normalen Modellziege, soll immer nur ein Reinigungszug aufs Mal unterwegs sein. Ziel ist natürlich, den Schaden durch Ausfälle möglichst klein zu halten.

Trotzdem scheint das MiWuLa keine Anstrengungen unternommen zu haben, die Reinigungsfahrten zum Beispiel in ein kleines Intervall vor der Öffnung am Morgen zu planen und trotzdem die Kosten für Betriebsausfälle unter der Schranke  $k$  zu behalten. Möglicherweise ist das einfach für eine so umfangreiche Anlage ein zu schwieriges Problem. Woran könnte das liegen?

*Lösung.* Das Planungsproblem für die Reinigungsziege ist äquivalent zum NP-vollständigen Problem SEQUENCING, wie man mit Hilfe einer polynomiellen Reduktion zeigen kann. Dazu seien die  $t_i$  die Fahrzeiten für jeden der  $p$  Reinigungsziege,  $d_i$  ist die späteste Fahrzeit, zu der die Zugfahrt von Zug  $i$  abgeschlossen sein muss, und  $s_i$  sind die Kosten für eine verspätete Reinigung durch den Zug  $i$ .

Fahrzeit von Zug  $i \leftrightarrow$  Ausführungszeit  $t_i$  von Job  $i$

Deadline für die Fahrt von Zug  $i \leftrightarrow d_i$

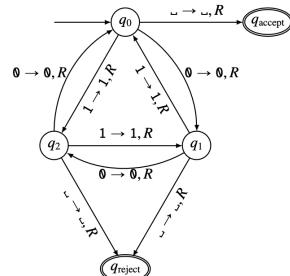
Kosten für Verspätete Reinigungsfahrt  $i \leftrightarrow s_i$

Reihenfolge der Fahrten  $\leftrightarrow$  Permutation  $\pi$

Die Aufgabe besteht darin, die Reihenfolge  $\pi$  der Reinigungszugfahrten so zu bestimmen, dass die Kosten für verspätete Reinigungen unter  $k$  zu halten, dies ist genau die Aufgabe des SEQUENCING-Problems. Damit ist gezeigt, dass das Reinigungszugsplanungsproblem NP-vollständig ist. Der Aufwand zur Planung der Reinigungszugsplanung steigt daher nach heutigem Wissen exponentiell mit der Grösse des Problems.  $\circlearrowright$

*Bewertung.* Reduktionsidee (R) 1 Punkt, Auswahl eines Vergleichsproblems (V) 2 Punkte, Mapping (M) 2 Punkte, NP-Vollständigkeit und Schlussfolgerung (S) 1 Punkt.

#### 8. Betrachten Sie die Turing-Maschine $M$ mit dem Zustandsdiagramm



- b) Welche der Wörter

01, 01000, 0100000, 111, 0111, 000111, 000111111, 00010101111

werden akzeptiert?

- c) Welche Sprache wird von  $M$  akzeptiert?

*Lösung.* a) Die Berechnungsgeschichte für das Wort  $w = 001011$  ist

Zustand								
$q_0$		0	0	1	0	1	1	
$q_1$		0	0	1	0	1	1	
$q_2$		0	0	1	0	1	1	
$q_1$		0	0	1	0	1	1	
$q_2$		0	0	1	0	1	1	
$q_1$		0	0	1	0	1	1	
$q_0$		0	0	1	0	1	1	
$q_{\text{accept}}$		0	0	1	0	1	1	

- b) Da nur Kopfverschiebungen nach rechts vorkommen, wird das Wort genau einmal durchgelesen, die Maschine realisiert also eigentlich nur einen endlichen Automaten. Mit jedem 0 auf dem Band ändert sich der Zustand im oberen Dreieck des Zustandsdiagramms im Urzeigersinn, mit jedem 1 auf dem Band im Gegenurzeigersinn. Akzeptiert wird, wenn der Zustand am Ende des Wortes  $q_0$  ist. Damit findet man schnell, dass

01, 01000, 0100000, 111, 000111, 000111111

akzeptiert und

0111, 00010101111

verworfen werden.

- c) Akzeptiert werden kann nur, wenn die Differenz der Anzahl der Nullen und Einsen die Maschine am Ende des Wortes im Zustand  $q_0$  lässt. Die akzeptierte Sprache besteht daher aus den Wörtern, deren Anzahlen von Nullen und Einsen gleichen Rest bei Teilung durch 3 haben, also

$$L = \{w \in \Sigma^* \mid |w|_0 \equiv |w|_1 \pmod{3}\}.$$

○

*Bewertung.* a) Berechnungsgeschichte (G) 1 Punkt, akzeptiert (A) 1 Punkt.

- b) Drei Punkte (B) 3 Punkte.

- c) Dreierrestbedingung (R) 1 Punkt.

- a) Wie wird das Wort 001011 verarbeitet?