Automaten und Sprachen | AutoSpr

Zusammenfassung

1. INHALT

Ablauf Prüfungsaufgaben

1. eine reguläre Sprache **S. 4**
2. eine nichtreguläre Sprache (Pumping Lemma) **S. 6**
3. eine kontextfreie Sprache (oft mit CNF) **S. 11**
4. eine nicht kontextfreie Sprache (Pumping Lemma) **S. 14**
5. ein nicht entscheidbares Problem (Rice oder Halteproblem) **S. 17**
6. ein Problem in NP (polynomieller Verifizierer) **S. 18**
7. ein NP-vollständiges Problem (polynomielle Reduktion) **S. 20**
8. etwas über Turing-Maschinen **S. 16**

2. LOGIK & QUANTOREN

2.1. JUNKTOREN / VERKNÜPFUNGEN

¬ Negation *nicht*
∧ Konjunktion *und*
∨ Disjunktion *oder*
⇒ Implikation *wenn … dann …*
⇔ Äquivalenz *… genau dann, wenn …*

2.2. QUANTOREN

Bei Aussagen, die Variablen enthalten, sind 2 Fälle besonders wichtig:

- ***Allquantor:*** $\forall n \in \mathbb{N}$ gilt $S(n) = \frac{1}{2}n(n+1)$ «für ***alle*** n aus den natürlichen Zahlen gilt…»
- ***Existenzquantor:*** $\exists n \in \mathbb{N}$ gilt $S(n) = \frac{1}{2}n(n+1)$ «es gibt ***eine*** natürliche Zahl n, für die gilt...»

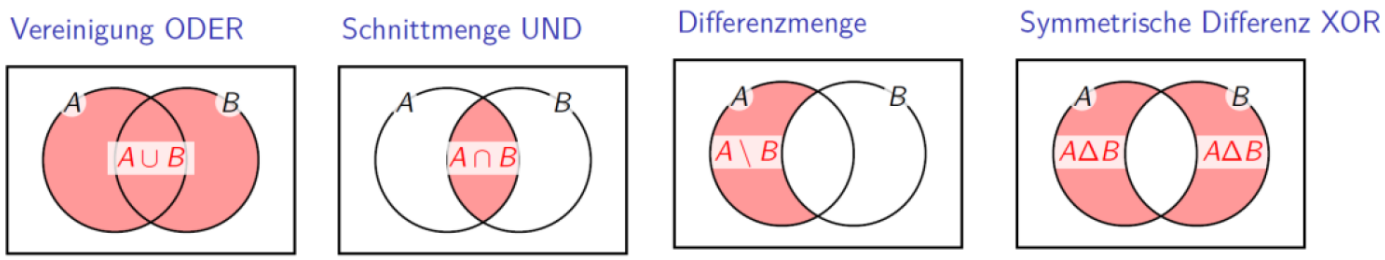
2.3. BEWEISEN

Ein Beweis zeigt die Gültigkeit einer Behauptung unter gewissen Voraussetzungen und Annahmen:

- ***Direkter Beweis:*** $A \Rightarrow B$
- ***Indirekter Beweis:*** $\neg B \Rightarrow \neg A$
- ***Widerspruchsbeweis:*** $A \wedge \neg B \Rightarrow F$
- ***Vollständige Induktion:*** $A(1) \wedge \big(A(n) \Rightarrow A(n+1)\big) \Rightarrow A(m), m \in \mathbb{N}$
- ***Vollständige Fallunterscheidung:*** Wenn $A \Leftrightarrow A_1 \vee A_2 \vee \dots A_n$ und es gilt, dass $(A_1 \Rightarrow B) \wedge (A_2 \Rightarrow B) \wedge \dots \wedge (A_n \Rightarrow B)$ ist, dann gilt auch $A \Rightarrow B$
- ***Schubfachprinzip:*** Wenn n + 1 Gegenstände (Objekte) auf n Schubladen (Kategorien) verteilt werden, dann befinden sich in mindestens einem Schubfach 2 Gegenstände.
- ***Diagonalverfahren (Georg Cantor):*** Anzahl rationaler Zahlen ist genau so gross wie die Anzahl natürlicher Zahlen.

3. MENGENLEHRE

«Primitive Datentypen der Mathematik», ***Zusammenfassung bestimmter Objekte***.
Konstruktion: Grundmenge G, Prädikat $P(x)$, $A = \{x \in G \mid P(x)\}$



3.1. PAARE, TUPEL & GRAPHEN

- ***Tupel:*** Zusammengesetzte Datentypen der Mathematik
- ***Paare (2-Tupel):*** A, B sind Mengen, Menge aller Paare: $A \times B = \{(a,b) \mid a \in A \wedge b \in B\}$
- ***N-Tupel:*** Mengen $A_0, \dots, A_n$, Menge aller n-Tupel

4. SPRACHEN

4.1. ALPHABET UND WORT

Eine nichtleere endliche Menge $\sum$ heisst ***Alphabet***. Die Elemente von $\sum$ heissen ***Zeichen***.

Wort

Eine Zeichenkette der Länge n ist ein n-Tupel in $\sum^n = \sum x \dots x \sum$. Ein Element von $\sum^n$ heisst «Wort der Länge n».

Abgekürzte Schreibweisen:

$(A,u,t,o,S,p,r) = AutoSpr$

$a^3b^5 = aaabbbbb$

Leeres Wort

Die Zeichenkette $\varepsilon \in \sum^0 = \{\varepsilon\}$ der Länge 0 heisst das leere Wort.

Menge aller Wörter

$$\sum^* = \{\varepsilon\} \cup \sum \cup \sum^2 \cup \sum^3 \cup \dots = \bigcup_{k=0}^{\infty} \sum^k$$

4.2. WORTLÄNGE

Wörter haben immer endliche Länge.

- ***Länge des Wortes w:*** $w \in \sum^n => |w| = n$
- ***Anzahl Zeichen im Wort w:*** Sei $w \in \sum^n$ und $a \in \sum$, dann ist $|w|_a$ die Anzahl Zeichen im Wort w.

Beispiele

- $|\varepsilon| = 0$
- $|01010|_0 = 3$
- $|01010|_1 = 2$
- $|a^3b^5| = 8$
- $|(1291)^7| = 7|1291| = 7 * 4 = 28$, $\boldsymbol{|w^n| = n * |w|}$

4.3. SPRACHE

Eine Sprache besteht aus diesen Wörtern. Eine Teilmenge von $\sum^*$ ist eine Sprache.

Beispiele

- $L = \sum^* \subset \sum^*$
- $L = \emptyset \subset \sum^*$, die leere Sprache
- $\sum = \{0,1\}$, Sprache aller Binärstrings : $L = \sum^*$
- $\sum = \{0,1\}, L = \{w \in \sum^* | \ |w|_0 = |w|_1\}$ => Alle Wörter beinhalten gleich viele Einsen und Nullen

4.4. SPRACHEN UND MASCHINEN

Der Compiler entscheidet, welche Speicherketten zur Sprache gehören.

M eine «Maschine»: $L(M) = \{w \in \sum^* |$ w wird von M akzeptiert $\}$

Beispiel: $\sum = Unicode, J = \{w \in \sum^* |$ w wird vom Java-Compiler akzeptiert $\}$

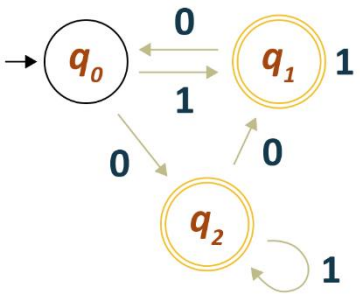
Zu jeder Maschine gibt es eine Sprache.

5. DETERMINISTISCHE ENDLICHE AUTOMATEN (DEA) UND REGULÄRE SPRACHEN

5.1. DEFINITION

$A = (Q, \Sigma, \delta, q_0, F)$

- Zustände: $Q = \{q_0, q_1, \dots, q_n\}$
- Alphabet: Σ
- Übergangsfunktion: $\delta: Q \times \Sigma \to Q$
- Startzustand: $q_0 \in Q$
- Akzeptierzustände: $F \subset Q$



		0	1	
Q	q_0	q_2	q_1	Σ
	q_1 /F	q_0	q_1	δ
F	q_2 /F	q_1	q_2	

Achtung: Nach Erstellung immer nachkontrollieren, ob der DEA vollständig ist!

5.2. AKZEPTIERTE SPRACHE EINES DEA

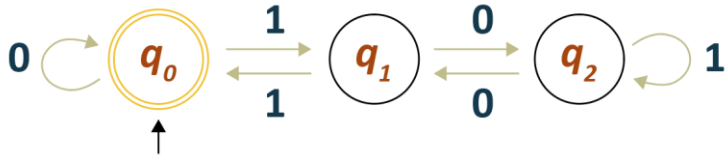
Der DEA $A = (Q, \Sigma, \delta, q_0, F)$ akzeptiert das Wort $w \in \Sigma^*$, wenn er A vom Startzustand in einen Akzeptierzustand $\delta(q_0, w) \in F$ überführt.

Die von A akzeptierte Sprache ist $L(A) = \{w \in \Sigma^* | A\ akzeptiert\ w\} = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$

Die Sprache $L \subset \Sigma^*$ heisst regulär, wenn es einen DEA A gibt, mit $L(A) = L$.

5.3. BEISPIEL: DURCH DREI TEILBARE BINÄRZAHLEN

- Zustände: Rest bei Teilung durch 3
- Übergänge: ein Bit anhängen (Shift) = Multiplikation mit 2



5.3.1. Zustandsmenge aus der Sprache ableiten

Wie findet man heraus, wie viele Zustände ein DEA wirklich braucht?

Bei diesem Beispiel: $L = \{\varepsilon,\ 0, 00, 11, 000, 011, 110, 0000, 0011, 0110, 1001, 1100, 1111, 00000, 00011, 00110, 01001, 01100, 01111, 10010, 10101, 11000, 11011, 11110, \dots\}$

Alle Werte durchprobieren, in Tabelle schreiben. Es wird Wiederholungen geben. Jede Wiederholung ist der Gleiche Zustand.

Daraus lässt sich der DEA generieren.

W	L(w)	Q (Start)
ε	{ε, 0, 00, 11, 000, …}	q_0
0	{ε, 0, 00, 11, 000, …}	q_0
1	{1, 10, 001, 100, 111, …}	q_1
00	{ε, 0, 00, 11, 000, …}	q_0
01	{1, 10, 001, 100, 111, …}	q_1
10	{01, 010, 101, . .}	q_2
11	{ε, 0, 00, 11, 000, …}	q_0
000	{ε, 0, 00, 11, 000, …}	q_0
001	{1, 10, 001, 100, 111, …}	q_1
010	{01, 010, 101, . .}	q_2
011	{ε, 0, 00, 11, 000, …}	q_0
100	{1, 10, 001, 100, 111, …}	q_1
101	{01, 010, 101, . .}	q_2
110	{ε, 0, 00, 11, 000, …}	q_0

Myhill-Nerode Automat

Gestattet es uns, zu einer beliebigen regulären Sprache den endlichen Automaten zu konstruieren. Nicht immer praktisch, da man zum Teil mit grossen Mengen hantieren muss.

Auch nützlich, um redundante Zustände in einem bestehenden DEA zu finden.
Für ein Wort $w \in \Sigma^*$ setze $L(w) = \{w' | ww' \in L\}$. Insbesondere $L(\varepsilon) = L$.

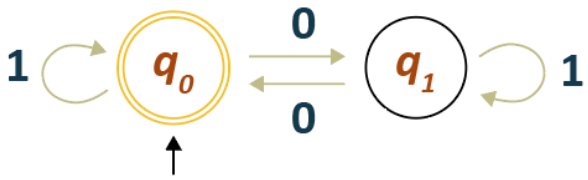
Gegeben: reguläre Sprache L über Σ. Rekonstruiere A mit:

- $Q = \{L(w) \mid w \in \Sigma^*\}$
- $q_0 = L(\varepsilon) = L$
- $F = \{L(w) \in Q \mid \varepsilon \in L(w)\}$
- $\delta(L(w), a) = L(wa)$

Beispiel: $\Sigma = \{0,1\}, L = \{w \in \Sigma^* \mid |w|_0 gerade\}$

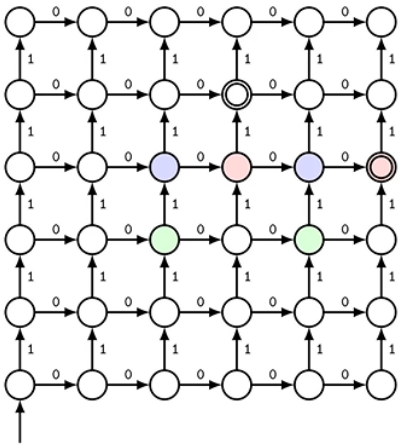
W	*L(w)*	*Q*
ε	$L(\varepsilon) = L$	q_0
0	$L(0)$ $= \{w \in \Sigma^* \mid |w|_0 ungerade\}$	q_1
1	$L(0) = \{w \in \Sigma^* \mid |w|_0 gerade\}$ $= L$	q_0
…	…	…

Ergibt folgenden Automaten:



Eine nicht-reguläre Sprache benötigt unendliche Zustände. Deshalb lässt sich für eine nicht-reguläre Sprache kein endlicher Automat erstellen.

Alternative Art, einen Automaten zu vereinfachen
Zustandspaare finden, die sich nicht zusammenlegen lassen.

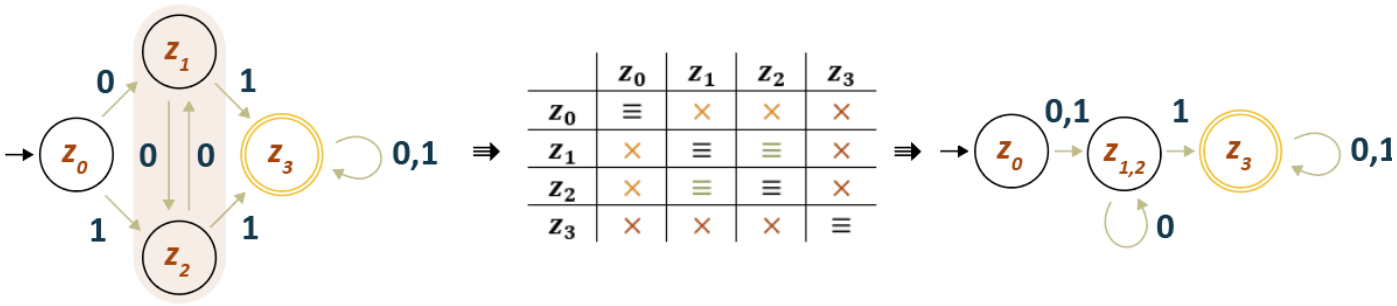


Plan
Nicht unterscheidbare Zustände zusammenlegen

Unterscheidbar?
1. Akzeptierzustand ≠ Nichtakzeptierzustand
2. Nach 0-Übergang unterscheidbar
3. Nach 1-Übergang in einem unterscheidbaren Paar
4. Iteration: alle unterscheidbaren Paare suchen

«Kreuzchenalgorithmus» - Minimalautomat finden
Dient als Beweis, ob zwei Automaten die gleiche Sprache akzeptieren.

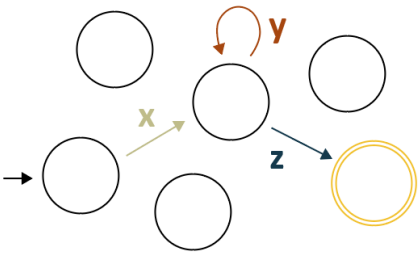
Tabelle erstellen, in der erkennbar ist, ob sich zwei Zustände unterscheiden. Ablauf:
- Alle Zustände zu sich selber äquivalent markieren: ≡
- Paare aus einem Akzeptier- und einem Nichtakzeptierzustand *(Zwei Zustände können nicht gleich sein, wenn einer davon ein Akzeptierzustand ist, der andere aber nicht)*: ×
- Zweifingertechnik: Von einem Paar alle Übergänge testen, führt zu einem Paar mit ×? Falls ja – ebenfalls ein × eintragen. Wiederholen für alle Paare ohne Zuordnung.
  Bsp. z_0 und z_1: $z_1$führt mit einem 1-Übergang zu einem Akzeptierzustand, z_0 hingegen nicht. ×
- Restliche Zustände sind Äquivalent ≡ und können minimiert werden.



6. PUMPING LEMMA, NEA, MENGENOPERATIONEN

6.1. PUMPING LEMMA

Wenn ein Wort länger sein kann als die Anzahl Zustände, dann muss mindestens ***ein Zustand mehrmals vorkommen im endlichen Automaten***: Ein Wort ist aufpumpbar (oder abpumpbar). Man kann damit ***nachweisen***, dass eine Sprache ***nicht regulär*** ist, wenn sie mindestens ein Wort enthält, das nicht aufpumpbar ist. ***Widerspruchsbeweis***.



Ist L eine reguläre Sprache, dann gibt es $N \in \mathbb{N}$, die pumping length so, dass jedes Wort $w \in L$ mit $|w| \geq N$ in drei Teile $w = xyz$ zerlegt werden kann mit

- $|xy| \leq N$
- $|y| > 0$
- $xy^kz \in L,\ \forall k \in \mathbb{N}$

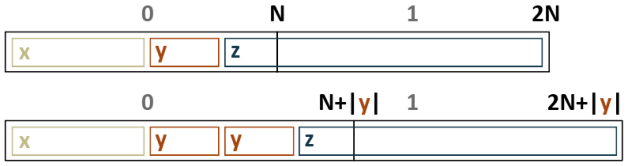
6.1.1. Beweis/Ablauf

- Annahme: L ist regulär
- Es gibt die Pumping Length N
- Wähle ein Wort $w \in L$ mit $|w| \geq N$, Definition mit ***N*** (nicht mit n) schreiben
- Aufteilung des Wortes gemäss Pumping Lemma $w = xyz, |xy| \leq N, |y| > 0$
- Auswirkung des Pumpens mit Begründung
- Widerspruch und Schlussfolgerung

Beispiele

$L = \{0^n1^n \mid n \geq 0\}$

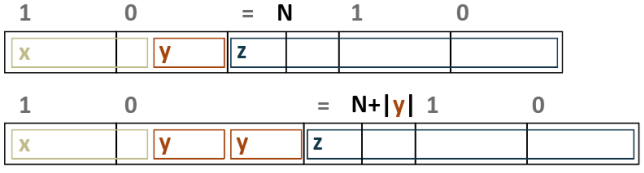
- Annahme: L ist regulär
- Pumping Length: $\exists N \in \mathbb{N}$
- $w = 0^N1^N$, N = Pumping Length
- Aufteilung $w = xyz$:



- Pumpen erhöht nur die Anzahl 0, jedoch nicht Anzahl 1
- Widerspruch: $xy^kz \notin L$, für $k \neq 1$, L ist also nicht regulär

$L = \{w = w \mid w \in \{0,1\}*\}$

- Annahme: L ist regulär
- Pumping Length: $\exists N \in \mathbb{N}$
- $w = 10^N$, N = Pumping Length
- Aufteilung $w = xyz$:



- Pumpen: Auf der linken Seite des Gleichheitszeichens wird eine grössere Binärzahl wie auf der rechten Seite stehen.
- Widerspruch: L ist nicht regulär

6.2. NICHTDETERMINISTISCHE ENDLICHE AUTOMATEN

Ein DEA kann sich aufhängen. Jede Verzweigung muss einzeln geprüft werden.

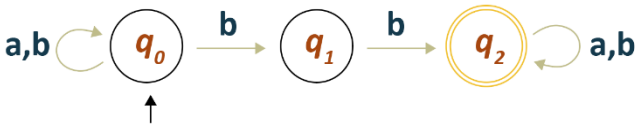
$A = (\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\delta}, q_0, \boldsymbol{F})$

- Zustände: $\boldsymbol{Q} = \{q_0, q_1, \dots, q_n\}$
- Alphabet: $\boldsymbol{\Sigma}$
- Übergangsfunktion: $\boldsymbol{\delta}$: $\boldsymbol{Q} \times \boldsymbol{\Sigma} \rightarrow \mathrm{P}(\mathbf{Q})$
- Startzustand: $q_0 \in \boldsymbol{Q}$
- Akzeptierzustände: $\boldsymbol{F} \subset \boldsymbol{Q}$

Beispiel:

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

$L = \{w \in \sum^* | \, w \, enthält \, zwei \, b \, nacheinander\}$



Definition Akzeptieren

Ein NEA A akzeptiert das Wort $w \in \sum^*$, wenn es eine Wahl von Übergängen gibt, derart, dass das Wort w den Automaten in einen Akzeptierzustand überführt.

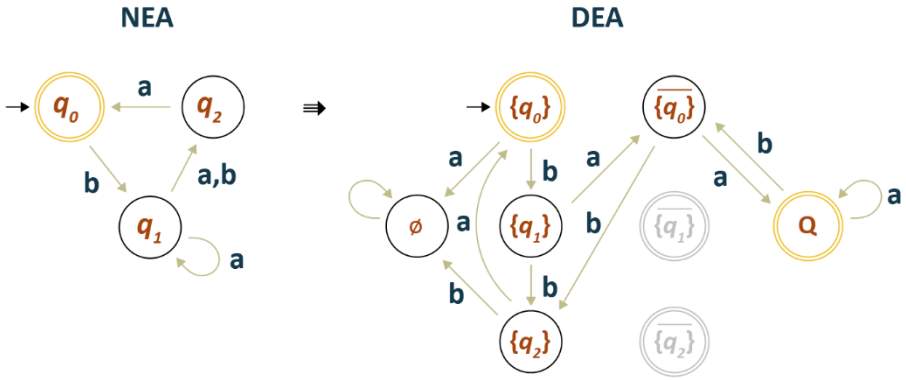
Faustregel

Nur genau diejenigen Pfeile einzeichnen, die man zum Akzeptieren braucht.

6.2.1. Thomspon-NEA: Transformation von NEA zu DEA («Könnte»-Automat)

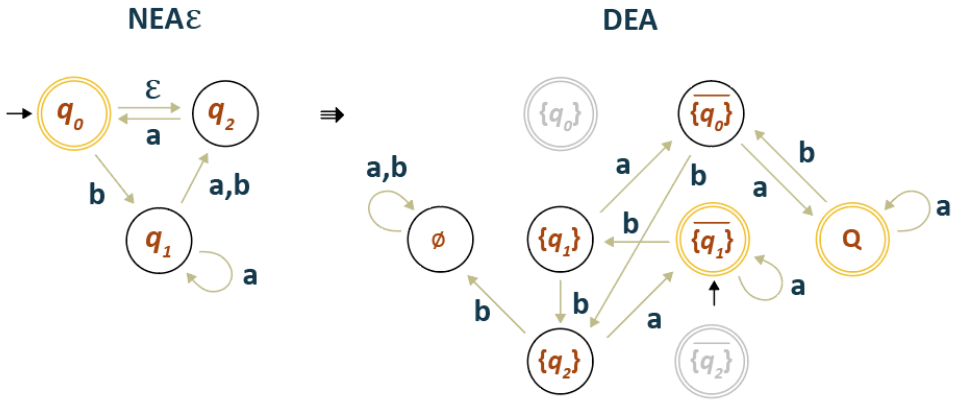
Alle Kombinationszustände einzeichnen, alle Zustände mit einem enthaltenen Akzeptierzustand sind Akzeptierzustände. Vom Startzustand ausgehend alle Übergänge einzeichnen.

DEA hat die Potenzmenge der Zustände vom NEA.



$\mathcal{E}$-Übergänge: Können ohne Verarbeitung eines Zeichens genommen werden. Sind gratis! Jeder NEA$\mathcal{E}$ kann in einen NEA umgewandelt werden.

Beispiel Thompson NEA$\mathcal{E}$ zu DEA

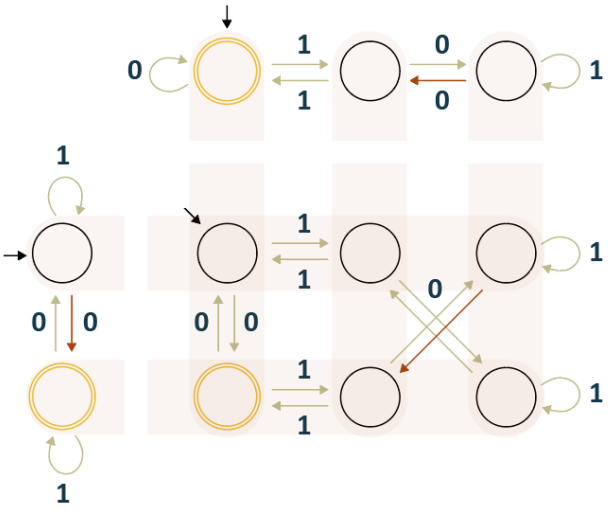


6.3. PRODUKTAUTOMAT

Beispiel im Bild ist die Schnittmenge

$L = \{w \in \sum^* \mid |w|_0 \, ungerade\} \cap \{w \in \sum^* \mid w \, ist \, eine \, durch \, drei \, teilbare \, Binärzahl\}$

- Beide Automaten auf eine Linie strecken / dehnen (falls möglich)
- Alle ***Übergänge einzeln eintragen***, Zustandswechsel des einten Automaten wirkt sich auf die Horizontale aus, die des andern auf die Vertikale.
- Akzeptierzustand ist bei der Schnittmenge dort, wo ***beide Automaten einen Akzeptierzustand*** haben.



Andere Akzeptierzustände:

- ***Vereinigung*** $L_1 \cup L_2$***:*** Alle Zustände, welche bei mindestens einem Automaten ein Akzeptierzustand sind.
- ***Differenz*** $L_1 \setminus L_2$***:*** Alle Zustände, welche nur beim ersten Automaten ein Akzeptierzustand sind.

6.4. MENGENOPERATIONEN

Wenn A1 und A2 DEA's sind, so kann man Mengenoperationen darauf anwenden und erhält so immer wieder einen DEA, also wieder eine reguläre Sprache.

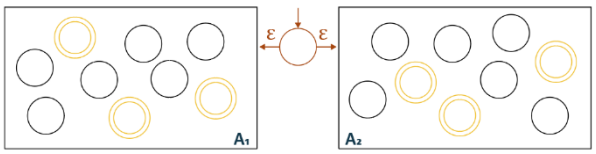
Operationen:

- Schnittmenge: $L_1 \cap L_2 : F = F_1 \times F_2$
- Vereinigung: $L_1 \cup L_2 : F = F_1 \times Q_2 \cup Q_2 \times F_2$
- Differenz: $L_1 \setminus L_2 : F = F_1 \times (Q_2 \setminus F_2)$

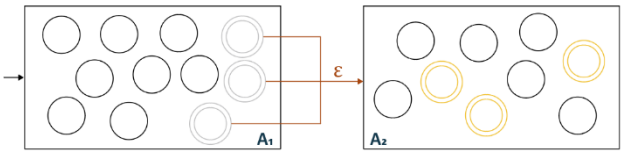
7. REGULÄRE OPERATIONEN UND REGULÄRE AUSDRÜCKE

7.1. REGULÄRE OPERATIONEN

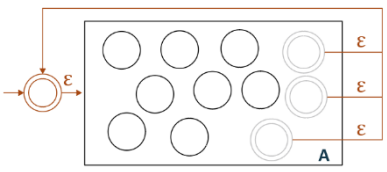
- ***Vereinigung/Alternative (Oder):***
  $L = L_1 \cup L_2 = L(A_1) \cup L(A_2)$
  Neuer Startzustand mit $\mathcal{E}$-Übergängen zu den einzelnen Automaten. Man kann entweder in den einen oder anderen Automaten gehen (ODER). Muss nicht wissen, was die einzelnen Automaten machen.



- ***Verkettung:***
  $L = L_1 L_2 = L(A_1)L(A_2) = \{w_1 w_2 | w_i \in L_i\}$
  A_1 Akzeptierzustände mit $\mathcal{E}$-Übergängen zu Startzugang A_2. Aus den vorherigen Akzeptierzuständen von A_1 $\mathcal{E}$ -Verknüpfung in A_2. A_1 hat keine Akzeptierzustände mehr.



- ****-Operation:***
  $L^* = \{\mathcal{E}\} \cup L \cup L^2 \cup \ldots = \bigcup_{k=0}^{\infty} L^k$
  Neuer Startzugang mit $\mathcal{E}$-Übergang zum alten Startzustand, $\mathcal{E}$-Verknüpfung aus den vorherigen Akzeptierzuständen von A zum neuen Startzugang. Leeres Wort ist ebenso akzeptiert.



7.2. REGULÄRE AUSDRÜCKE

VNEA ist ein NEA, dessen Pfeile mit regulären Ausdrücken angeschrieben sind.
Zu jedem regulären Ausdruck gibt es einen DEA.

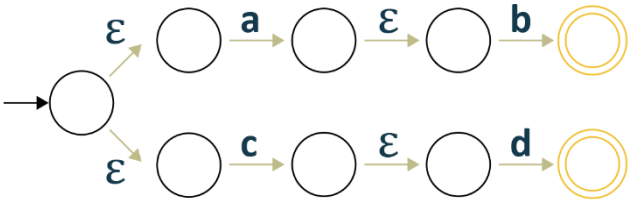
Ausdruck r	***Sprache*** $L = L(r)$	***Bedeutung***	***NEA***
$\emptyset$	$\emptyset$	Die leere Sprache	
ε	$\{\varepsilon\}$	Sprache, die nur das leere Wort enthält	
a	$\{a\}$	Sprache, die nur $a \in \sum$ enthält	
$[a-s]$	$\{a, b, \dots, s\}$	Sprache, die bestimmte Buchstaben enthält	
.	$\sum$	Jedes Zeichen aus $\sum$ wird akzeptiert	
$r_1 \mid r_2$	$L(r_1) \cup L(r_2)$	r_1 oder r_2 *Alternative / Vereinigung von zwei Sprachen*	
$r_1\, r_2$	$L(r_1)L(r_2)$	$r_1\, r_2$ *Verkettung von zwei Sprachen*	
$r_1 *$	$L(r_1 *)$	Beliebig viele r_1	
$r +$	$rr *$	Mindestens ein r plus beliebig viele r	
$r?$	$\varepsilon \mid r = \mid r$	Das leere Wort oder ein Mal r	
$r\{2\}$	rr	Genau zwei Mal r	
$r\{2,3\}$	$rr \mid rrr$	Zwei oder drei Mal r	
$r\{,3\}$	$\varepsilon \mid r \mid rr \mid rrr$	Höchstens drei Mal r	
$r\{3,\}$	$rrrr *$	Mindestens drei Mal r	

7.3. UMWANDLUNG REGEX -> NEA

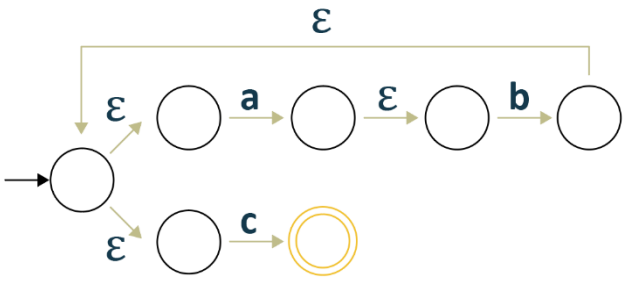
- Von innen nach aussen übersetzen (Klammern zuerst)
- Einzelen Teile mittel ε-Übergängen verbinden

Beispiele

$ab \mid cd$

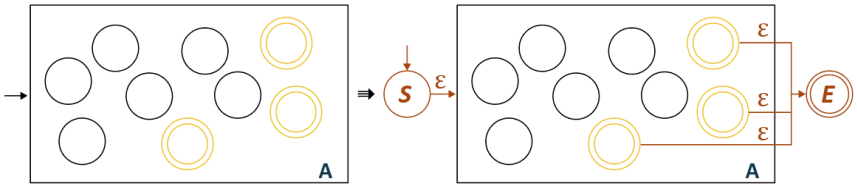


$(ab) * c$

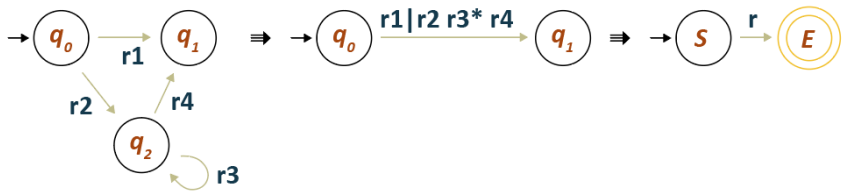


7.4. UMWANDLUNG VNEA ZU REGULÄREM AUSDRUCK

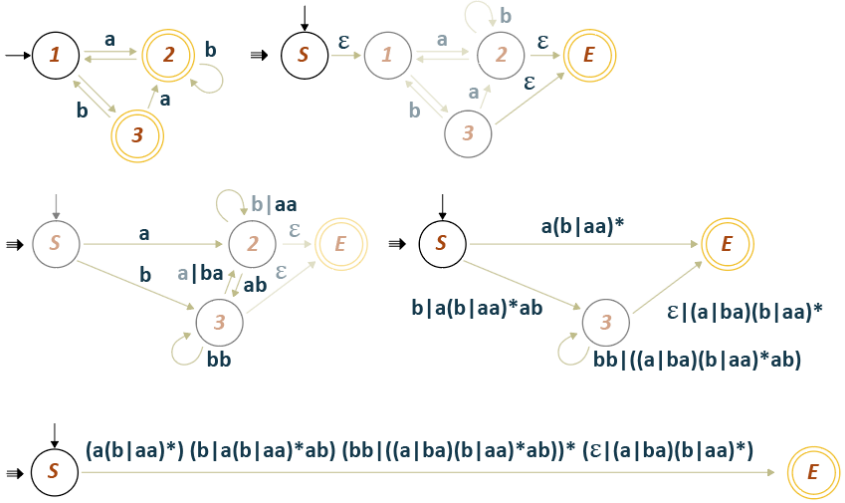
- Sichergehen, dass vom Startzugang zur Pfeile wegführen und beim Akzeptierzustand nur Pfeile ankommen: ***Neue Zustände S und E hinzufügen***



- Alle Zwischenzustände der Reihe nach ***entfernen***



7.4.1. Beispiel



8. KONTEXTFREIE SPRACHEN

Kontextfreie Sprachen können nur von nichtdeterministischen Stackautomaten erkannt werden. Existiert ein Stackautomat, Regex oder eine Grammatik, ist die Sprache kontextfrei.

Kontextfrei bedeutet, dass bei den Regeln bei der Variable auf der linken Seite der Kontext nicht relevant ist.

- ***Regeln ohne Kontext:*** $S \to A \mid C, A \to a, A \to aAb, C \to bA$
- ***Regeln mit Kontext:*** $S \to C \mid aC \mid bC, \boldsymbol{a}C \to A, \boldsymbol{b}C \to B$

8.1. KONTEXTFREIE GRAMMATIK

Kontextfreie Grammatik: $G = (V, \sum, R, S)$

- V: Endliche Menge von Variablen
- $\sum$: Endliche Menge von Zeichen, Terminalsymbole (Alphabet)
- R: Eine Menge von Regeln der Form $A \to x_1 x_2 \dots x_n$ mit $A \in V \cup \sum$
- $S \in V$: Startvariable

8.1.1. Beispiele

$L = \{w \in \{(,)\} \mid$ w gültige Klammerung $\}$

- $V = \{K\}$
- $\sum = \{(,)\}$
- $R = \{K \to \varepsilon, K \to KK, K \to (K)\}$
- $S = K$

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

- $V = \{Q\}$
- $\sum = \{0,1\}$
- $R = \{Q \to \varepsilon, Q \to 0Q1\}$
- $S = Q$

8.1.2. Grammatik für reguläre Operationen

L_1und L_2 sind kontextfreie Sprachen mit Grammatiken $G_i = (V_i, \sum, R_i, S_i)$. Grammatik für reguläre Operationen:

- Neue Startvariable S_0
- Variablen $V = V_1 \cup V_2 \cup \{S_0\}$
- Geeignet erweitere Regeln R

=> $G = (V, \sum, R, S_0)$

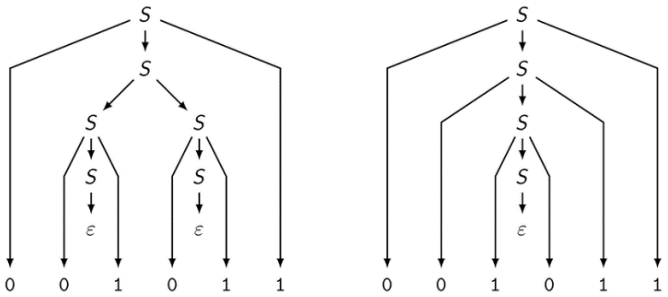
- ***Alternative / OR:*** Regeln für $L_1 \cup L_2$: $R = R_1 \cup R_2 \cup \{S_0 \to S_1, S_0 \to S_2\}$
- ***Verkettung / AND:*** Regeln für $L_1 L_2$ $R = R_1 \cup R_2 \cup \{S_0 \to S_1 S_2\}$
- ****-Operation:*** Regeln für ${L_1}^*$ $R = R_1 \cup \{S_0 \to S_0 S_1, S_0 \to \varepsilon\}$

8.2. PARSE-TREE

Zwei Ableitungen eines Wortes w einer kontextfreien Sprache L heissen ***äquivalent***, wenn sie den ***gleichen Ableitungsbaum*** (Parse-Tree) haben, Hat eine Sprache Wörter ***mit verschiedenen Parse-Trees***, heisst sie ***mehrdeutig***.

Beispiel mehrdeutiger Parse Tree

$\boldsymbol{S \to 0S1 \mid 1S0 \mid SS \mid \varepsilon}$



8.3. CHOMSKY-NORMALFORM (CNF)

Eine Regel ist in der Chomsky-Normalform, wenn S auf der ***rechten Seite nicht vorkommt*** und jede Regel von der Form $A \to BC$ oder $A \to a$ ist. Zusätzlich ist noch die Regel $S \to \varepsilon$ erlaubt.

8.3.1. Umwandlung in Chomsky-Normalform

- Neue Startvariable $S_0 \to S$
- ε-Regeln: $\left.\begin{matrix} A \to \varepsilon \\ B \to AC \end{matrix}\right\} \to A$ kann weggelassen werden $\Rightarrow \begin{cases} B \to AC \\ \to C \end{cases}$
- Unit-Rules: $\left.\begin{matrix} A \to B \\ B \to CD \end{matrix}\right\} \to$ aus A kann man wie aus B auch CD machen $\Rightarrow \begin{cases} A \to CD \\ B \to CD \end{cases}$
- Verkettungen: $A \to u_1 u_2 .. u_n$ ersetzen durch $A \to u_1 A_1, A_1 \to u_2 A_2, .. A_{n-2} \to u_{n-1} u_n$ und falls u_i ein Terminalsymbol ist: $A_{i-1} \to U_i A_i, U_i \to u_i$

8.3.2. Beispiel

Ausgangsgrammatik

- $S \to ASA \mid aB$
- $A \to B \mid S$
- $B \to b \mid \varepsilon$

Schritt 1: Neue Startvariable

- $S_0 \to S$
- $S \to ASA \mid aB$
- $A \to B \mid S$
- $B \to b \mid \varepsilon$

Schritt 2: ε nur aus Startvariable, das heisst A und B sind fakultativ

- $S_0 \to S$
- $S \to ASA \mid SA \mid AS \mid S \mid aB \mid a$
- $A \to B \mid S$
- $B \to b$

Schritt 3: Keine Unit-Rules

- $S_0 \to ASA \mid SA \mid AS \mid aB \mid a$
- $S \to ASA \mid SA \mid AS \mid aB \mid a$
- $A \to b \mid ASA \mid SA \mid AS \mid aB \mid a$
- $B \to b$

Schritt 4: Keine Dreier-Regeln und Terminalsymbole auf der rechten Seite

- $S_0 \to AA_1 \mid SA \mid AS \mid UB \mid a$
- $S \to AA_1 \mid SA \mid AS \mid UB \mid a$
- $A \to b \mid AA_1 \mid SA \mid AS \mid UB \mid a$
- $B \to b$
- $A_1 \to SA$
- $U \to a$

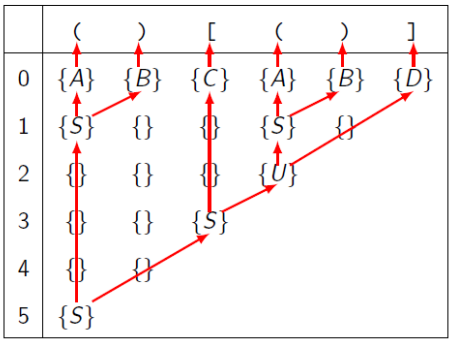
8.3.3. Anwendung der Chomsky-Normalform

Chomsky-Normalform ist nicht eindeutig, es kommt auf die Reihenfolge darauf an, wie die Schritte angewendet werden. Man kann sie deshalb nicht brauchen, um Grammatiken zu vergleichen.

- Ableitung eines Wortes $w \in L(G)$ ist immer in $\mathbf{2|w| - 1}$ Regelanwendungen möglich

$S \to AB \mid CD \mid AT \mid CU \mid SS$
$T \to SB$
$U \to SD$
$A \to ($
$B \to)$
$C \to [$
$D \to]$

Wort: ()[()]



8.3.4. Deterministisches Parsen

Ist w aus A ableitbar? In Zeichen $A \overset{*}{\Rightarrow} w$

Spezialfälle: $w = \varepsilon$ und $|w| = 1$ -> w ist ein Terminalsymbol. Falls $|w| > 1$:

$$A \overset{*}{\Rightarrow} w \quad \Rightarrow \quad \exists \begin{cases} A \to BC & \in R \\ w = w_1 w_2 & w_i \in \Sigma^* \end{cases} \quad \text{mit} \quad \begin{cases} B \overset{*}{\Rightarrow} w_1 \\ C \overset{*}{\Rightarrow} w_2 \end{cases}$$

8.4. CYK-ALGORITHMUS (COCKE-YOUNGER-KASAMI)

Damit lässt sich der Parse-Tree finden.
Nachteil: Laufzeit ist $0(n^3)$

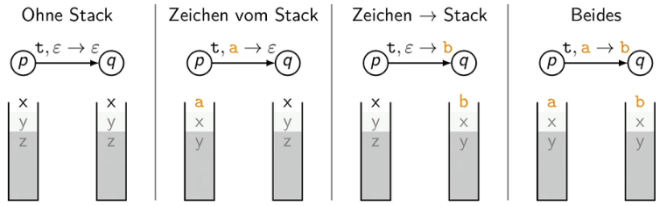
- ***Zeile 0:*** Terminalsymbol-Regeln
- ***Zeile 1:*** Variablen, die Paare produzieren können
- ***…***

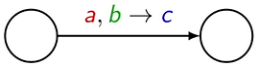
8.5. STACKAUTOMAT / PUSH DOWN AUTOMAT (PDF)

Ein Stackautomat kann nur ***kontextfreie Grammatiken erkennen***. Er ist immer ***nicht deterministisch***.

Stackautomat $P = (\mathrm{Q}, \Sigma, \Gamma, \delta, \mathrm{S}, \mathrm{F})$

Zeichen	***Bezeichnung***	***Beispiel***
Q	Zustände	$q0, q1, \ldots$
Σ	Eingabe-Alphabet	$\{0,1\}$
Γ	Stack-Alphabet	$\{0,1,\$\}$
δ	$Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$	$\varepsilon \to \$$
S	Startzustand	$q0$
F	Akzeptierzustände	$q3$



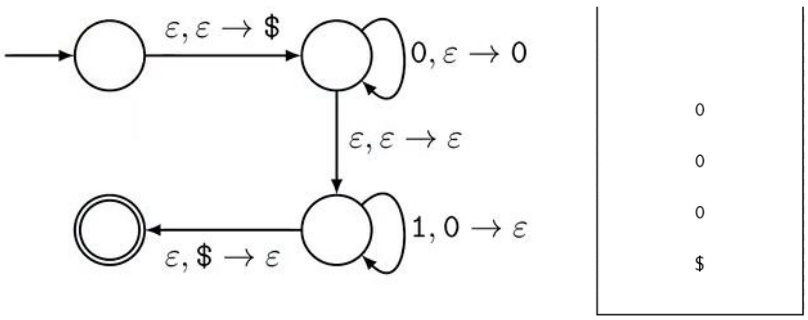


a vom Input
b vom Stack entfernen (Bedingung)
c auf den Stack

Übergänge: Siehe Bild, gelesen als «Verarbeite Input a und ersetze b auf dem Stack durch c».

8.5.1. Beispiel

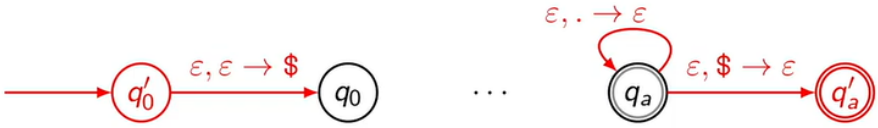
Jedes 0 wird auf den Stack gelegt. Bei jedem 1 wird wieder eine 0 vom Stack entfernt.



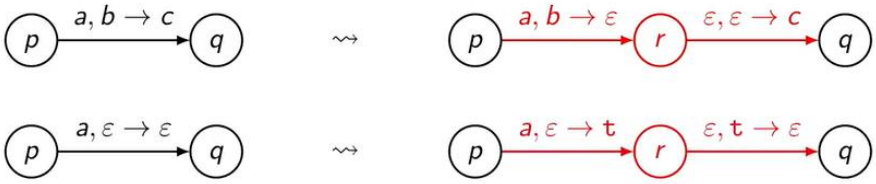
8.5.2. Stackautomat standardisieren *rote Inhalte ergänzen*

- ***Nur ein Akzeptierzustand:*** neuer Akzeptierzustand q_a und Übergänge $\forall q \in F$

- ***Stack leeren:*** Am Ende soll nichts mehr auf dem Stack sein.



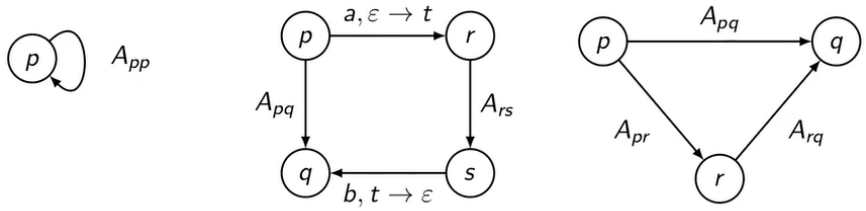
- Jeder Übergang ***legt*** entweder ein ***Zeichen*** auf den ***Stack*** oder ***entfernt*** eines:



8.5.3. Grammatik ablesen

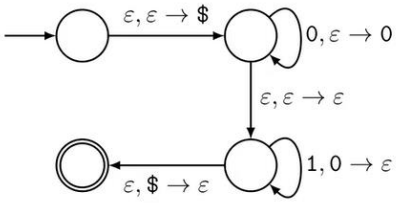
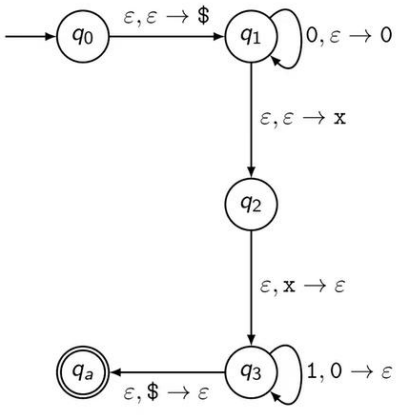
Ausgangspunkt: standardisierte Grammatik mit Starzustand q_o und $F = \{q_a\}$

- ***Startvariable:*** A_{q_o,q_a}
- ***Regeln:***



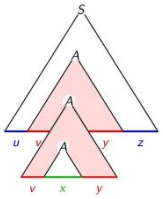
$A_{pp} \to \varepsilon$ $\quad$ $A_{pq} \to aA_{rs}b$ $\quad$ $A_{pq} \to A_{pr}A_{rq}$

Beispiel $L = \{0^n 1^n | n \geq 0\}$

Start-Automat	***Normalisierung Stackautomat***	***Grammatik ablesen***	***Vereinfachen***
		$A_{q_0q_a} \to \varepsilon A_{q_1q_3}\varepsilon$ $A_{q_1q_3} \to 0A_{q_1q_3}1$ $\to \varepsilon A_{q_2q_2}\varepsilon$ $A_{q_2q_2} \to \varepsilon$	$S \to 0S1$ $\to \varepsilon$

8.6. PUMPING LEMMA FÜR KONTEXTFREIE SPRACHEN

Es gibt auch Sprachen, die ***nicht kontextfrei*** sind. Z.B. die Sprache $L = \{a^n b^n c^n \mid n \geq 0\}$. Um zu ***beweisen, dass eine Sprache nicht kontextfrei ist***, kann man das Pumping Lemma verwenden, muss aber die Grammatik dazu nehmen.



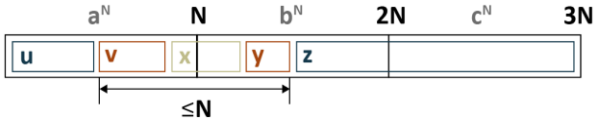
Ist L eine Kontextfreie Sprache (CFL), dann gibt es eine Zahl N, die Pumping Length, derart, dass jedes Wort $w \in L$ mit $|w| \geq N$ in fünf Teile $w = uvxyz$ zerlegt werden kann, dass

- $|vy| > 0$
- $|vxy| \leq N$
- $uv^k xy^k z \in L \; \forall k \in \mathbb{N}$

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

Beispiel: $\{a^n b^n c^n | n \geq 0\}$

- ***Annahme:*** L ist kontextfrei
- ***Pumping Length*** N
- ***Wort***: $w = a^N b^N c^N$ (Achtung, N und nicht n)
- ***Zerlegung***:



- Beim Pumpen nimmt die Anzahl der a und b zu, nicht aber die Anzahl der c ⇒ $uv^k xy^k z \notin L \forall k \neq 1$
- ***Widerspruch***: L ist nicht kontextfrei

9. TURING MASCHINEN

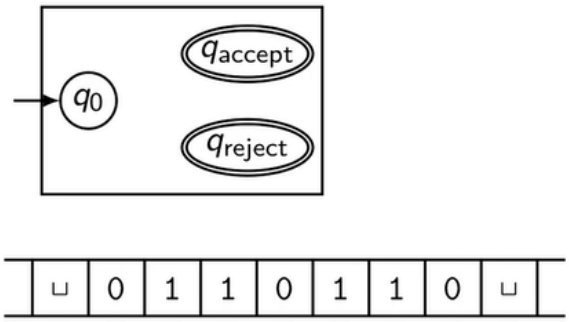
Eine Turing Maschine besteht aus:

- Einem ***unendlich langem Speicherband*** mit unendlich vielen sequentiell angeordneten Feldern. Pro Feld kann genau ein Zeichen aus einem vordefinierten Alphabet gespeichert werden.
- Einem ***Lese- und Schreibkopf***, der sich auf dem Speicherband feldweise bewegen kann.

Eine Sprache L heisst «Turing-Erkennbar», wenn es eine Turing-Maschine M gibt, die das Wort akzeptiert.

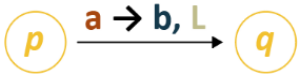
Turing Maschine $M = (\mathrm{Q}, \Sigma, \Gamma, \delta, \mathrm{S}, \mathrm{q_{accept}}, \mathrm{q_{reject}})$

Zeichen	***Bezeichnung***	***Beispiel***
Q	Zustände	$q0, q1, \ldots$
Σ	Eingabe-Alphabet	$\{0,1\}$
Γ	Stack-Alphabet	$\{0,1,\sqcup\}$
δ	$Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$	$a \rightarrow b, R$
S	Startzustand	$q0$
q_{accept}	Akzeptierzustand	
q_{reject}	Ablehnungszustand	



Zustandsdiagramm:

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



Übergang:

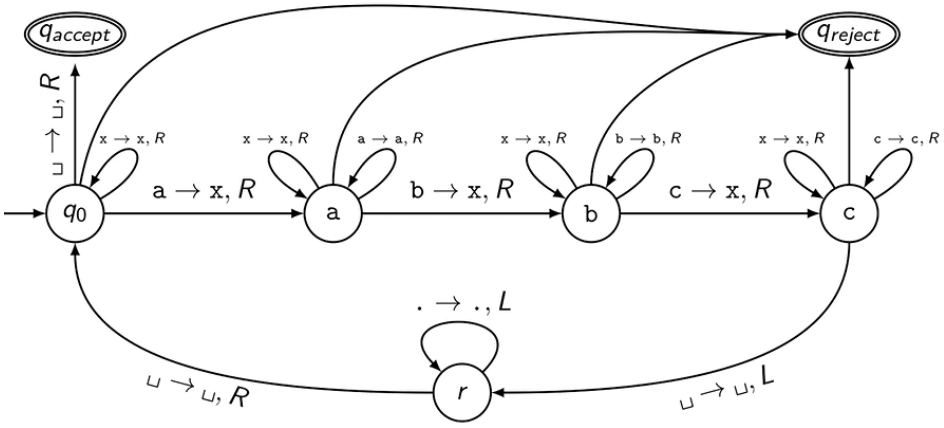
- Übergang möglich, wenn **a** unter dem Schreib-/Lese-Kopf
- Aktuelles Feld auf dem Band wird mit **b** überschrieben
- Kopfbewegung: **L** links, **R** rechts

9.1. ABLAUF PROGRAMM

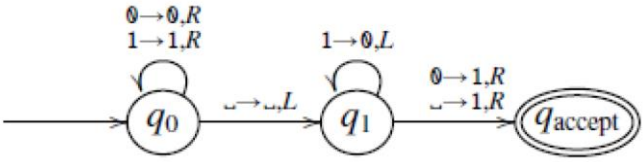
- Inputwort w auf Band schreiben
- Schreib- / Lesekopf auf erstes Zeichen Positionieren (meist ⊔)
- Maschine starten, $t(w)$ Einzelschritte ausführen
- Maschine hält in q_{accept}oder q_{reject} an und akzeptiert oder verwirft Wort entsprechend

9.2. BEISPIELE

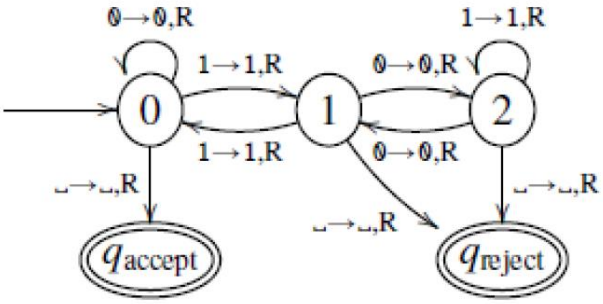
$L = \{a^n b^n c^n | n \geq 0\}$



Binärzahlen um 1 erhöhen



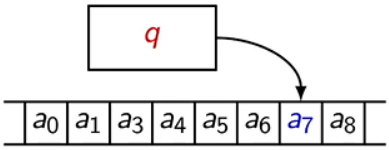
Binärzahlen durch 3 Teilbar (DEA in Turing umwandeln)



9.3. BERECHNUNGSGESCHICHTE

Kann auch auf ein Ausfüllrätsel reduziert werden und somit beweisen, dass SAT NP-Vollständig ist

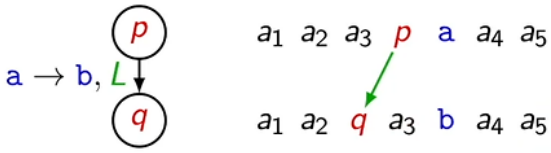
Notation für Maschinenzustand



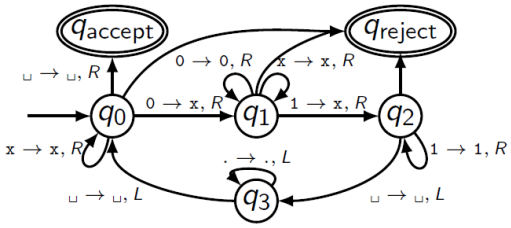
protokolliert als

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

Übergang



Protokoll einer Berechnung



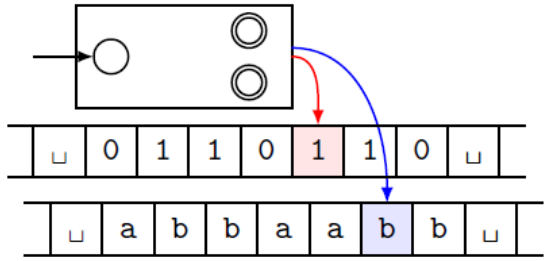
␣ q_0 0 0 1 1 ␣
␣ x q_1 0 1 1 ␣
␣ x 0 q_1 1 1 ␣
␣ x 0 x q_2 1 ␣
␣ x 0 x 1 q_2 ␣
␣ x 0 x q_3 1 ␣
␣ x 0 q_3 x 1 ␣
␣ x q_3 0 x 1 ␣
␣ q_3 x 0 x 1 ␣
. . .

9.4. VARIANTEN VON TURING-MASCHINEN

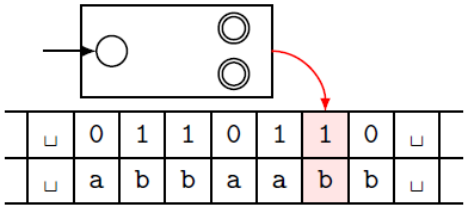
Jede Sprache, die von einer alternativen Turingmaschine erkann werden kann, kann auch von einer einspuringen Turingmaschine erkannt werden, ist einfach langsamer.

- ***Anderes Bandalphabet:*** Simuliert z.B. durch binäre Codierung, $O(t(n))$
- ***Mehrere Bänder (ein Pointer pro Band):*** Simulierbar mit $2n$ mehrspurigem Band, jedes Band bekommt eine zusätzliche Spur

für die Position. Die Position wird über ein Zeichen gelöst, welches verschoben wird und die Position anzeigt. $O(t(n)^2)$



- ***Mehrspurige TM (ein Pointer):*** Simuliert, in dem man die Spuren alle seriell hinterlegt. $O(t(n))$



- ***Nichtdeterministische TM:*** Bei jedem Übergang maximal N verschiedene Möglichkeiten. Mehrere Berechnungswege möglich, Akzetiert sobald ein Weg zu q_{accept}führt. Simulation: Maximal $2^{O(t(n))}$. 3 Bänder: Arbeitsband, Kopie von w und Liste aller Folgen von Wahlmöglichkeiten
- ***Aufzähler:*** Eine TM mit einem Drucker, bei dem gefundene Wörter «ausgedruckt» werden können. Kann endlos laufen. Wort kann mit «Ausdruck» verglichen werden um akzeptiert zu werden. Aufzählbare Sprache ⟺ Turing-Erkennbare Sprache

9.5. ENTSCHEIDBARKEIT

Ein ***Entscheider*** ist eine ***Turing-Maschine***, die auf jedem ***beliebigen Input anhält***. Eine ***Sprache*** L heisst ***entscheidbar***, wenn es einen ***Entscheider*** M gibt mit $L = L(M)$. Man sagt, M entscheidet L.

9.5.1. Turing-erkennbare Sprache

Eine Sprache L heisst Turing-erkennbar, wenn es eine TM M gibt, die die Sprache L als Input akzeptiert. Die TM kann auch unendlich lange laufen, wenn kein akzeptierter Input eingegeben wird. Ausser wenn M ein Entscheider ist, dann hält es immer an.

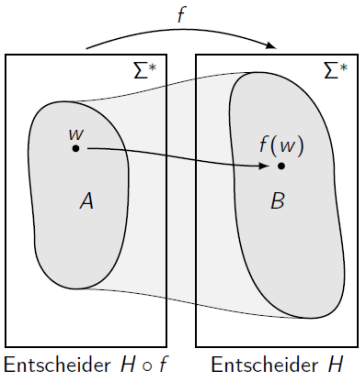
9.5.2. Entscheidbare Probleme

$E =$ Leerheitsproblem *M ist eine Turing-Maschine und M hält auf leerem Band*
$EQ =$ Gleichheitsproblem, $A =$ Akzeptanzproblem

Problem	***Wort***	***Bedingung***	***Entscheidbar***	***Entscheidungsalgorithmus***
E_{DEA}	$< A >$	$L(A) = \emptyset$	Ja	Minimalautomat hat keinen Akzeptierzustand
E_{CFG}	$< G >$	$L(G) = \emptyset$	Ja	Chomksy-Normalform
E_{TM}	$< M >$	$L(M) = \emptyset$	Nein	
EQ_{DEA}	$< A1, A2 >$	$L(A1) = L(A2)$	Ja	Vergleich der minimalen Automaten
EQ_{CFG}	$< G1, G2 >$	$L(G1) = L(G2)$	Nein	
EQ_{TM}	$< M1, M2 >$	$L(M1) = L(M2)$	Nein	
A_{DEA}	$< A, w >$	$w \in L(A)$	Ja	Regex-Engines simulieren beliebige DEAs auf beliebigen Input-Wörtern w
A_{CFG}	$< G, w >$	$w \in L(G)$	Ja	Cocke-Younger-Kasami, deterministischer Parse-Algorithmus
A_{TM}	$< M, w >$	$w \in L(M)$	Nein	Halteproblem

9.5.3. Reduktion

Mittels ***Reduktion*** kann bewiesen werden, dass eine ***TM nicht entscheidbar*** ist. Dazu ***reduziert*** man eine TM auf eine ***nicht entscheidbare TM*** (E_{TM}).
Idee: ***Neues Problem schaffen, welches allenfalls einfacher zu lösen ist***. (A ist leichter entscheidbar als B)



Berechenbare Abbildung: $w \in A \iff f(w) \in B$

Ist B ***entscheidbar***, dann ist auch A ***entscheidbar***. Ist B ***nicht entscheidbar***, dann ist auch A ***nicht entscheidbar***.

Reduktion für Halteproblem: $A_{TM} \leq HALT\varepsilon_{TM}$

A_{TM} ist nicht entscheidbar		Ist $HALT\varepsilon_{TM}$ entscheidbar?
Akzeptiert die Maschine M das Wort w?		Hält eine beliebige Maschine S an?
Es gibt keinen Entscheider		
$\langle M, w \rangle$	$\mapsto$	Programm S
		1. Führe M auf Inputwort w aus
		2. M hält in q_{accept}: akzeptiere
		3. M hält in q_{reject}: Endlosschleife
M akzeptiert w	$\Leftrightarrow$	S hält

Entscheider für A_{TM}
1. Konstruiere das Programm S
2. Wende H auf $\langle S \rangle$ an

Entscheider für $HALT\varepsilon_{TM}$
Wenn H ein Entscheider für $HALT\varepsilon_{TM}$ wäre, könnte man daraus einen Entscheider für A_{TM} konstruieren

Reduktion für das Leerheitsproblem: $A_{TM} \leq E_{TM}$

A_{TM} ist nicht entscheidbar		Ist E_{TM} entscheidbar?
Akzeptiert die Maschine M das Wort w?		Ist die Sprache $L(S)$ leer? $\rightarrow \bar{E}_{TM}$ ist $L(S) \neq \emptyset$?
Es gibt keinen Entscheider		
$\langle M, w \rangle$	$\mapsto$	Programm S mit Input u
		1. M auf w laufen lassen
		2. M akzeptiert w: q_{accept}
		3. q_{reject}
M akzeptiert w	$\Leftrightarrow$	S akzeptiert $L(S) = \Sigma^* \neq \emptyset$

Entscheider für A_{TM}
1. Konstruiere das Programm S
2. Wende H auf $\langle S \rangle$ an

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

Satz von Rice
Ist P eine nichttriviale Eigenschaft Turing-erkennbarer Sprachen, dann ist $P_{TM} = \{< M > \mid L(M) \text{ hat Eigenschaft } P\}$ nicht entscheidbar.

Voraussetzung: keine Einschränkung machen, wie Zahlengrösse reduzieren
Definition nichttrivial: Eine Eigenschaft P Turing-erkennbarer Sprachen heisst nichttrivial, wenn es zwei Turing-Maschinen $M1$ und $M2$ gibt, wobei $M1$ die Eigenschaft hat und $M2$ nicht.
Folgerung: Es ist nicht möglich, einem Programm anzusehen, ob die akzeptierte Sprache eine nichttriviale Eigenschaft hat.

Lösungsanleitung einer Prüfungsfrage:
- Nichttriviale Eigenschaft P aufschreiben
- Die beiden Sprachen $L(M1)$ und $L(M2)$ bilden
- Gibt es ein Programm, welches beide Sprachen erkennen kann? Sind beide Sprachen Turing erkennbar?
- Dann besagt der Satz von Rice, dass die Sprache nicht entscheidbar ist.

Beispiel 1
Zeigen Sie mit Hilfe des Satzes von Rice, dass die Frage, ob eine Turing-Maschine eine kontextfreie Sprache akzeptiert, nicht entscheidbar ist.

Ob eine Sprache kontextfrei ist oder nicht ist eine nichttriviale Eigenschaft. Um dies zu zeigen, muss man zwei Sprachen angeben, von denen die eine Kontextfrei ist und die andere nicht. Zwei mögliche Sprachen sind

$L_1 = \Sigma^*$ alle Wörter, kontextfrei dank der Grammatik $\begin{cases} S \rightarrow SZ \mid \varepsilon \\ Z \rightarrow \Sigma \end{cases}$

$L_2 = \{a^n b^n c^n \mid n \geq 0\}$ nicht kontextfrei, Standardbeispiel

Da die Spracheigenschaft, kontextfrei zu sein, eine nichttriviale Eigenschaft ist, folgt nach dem Satz von Rice, dass nicht entscheidbar ist, ob eine Turing-Maschine eine kontextfreie Sprache akzeptiert.

Beispiel 2

PPRIMES = "Sprache besteht nur aus den Primzahlen"

$L0=\{42\}$ $L1=\{Primzahlen\}$ $\Rightarrow Turing\ erkennbar$

Satz von Rice greift da, so PPRIMES nicht entscheidbar

Alternativ:

1. Reduktion auf Problem mit Erklärung
   a. Halteproblem
   b. Leerheitsproblem
   c. Regularitätsproblem
2. Problem ist nicht entscheidbar
3. Satz von Rice als zusätzliche Begründung

10. KOMPLEXITÄT

Eine Turing-Maschine mit mehreren Bändern und Laufzeit $t(n)$ kann in Laufzeit $O(t(n)^2)$ auf einer Standard Turing-Maschine simuliert werden.

Sei N aber eine Nichtdeterministische Turing-Maschine, dann kann diese in Laufzeit $2^{O(t(n))}$ simuliert werden. Dies gilt nur für NTM, die auch Entscheider sind.

10.1. POLYNOMIELLE UND EXPONENTIELLE LAUFZEIT

Polynomielle Probleme *Skalierbarkeit*

- Gauss-Alogrithmus $O(n^3)$
- FFT $O(n \log n)$
- Sortieren $\leq O(n^2)$

Exponentielle Probleme *Sicherheit*

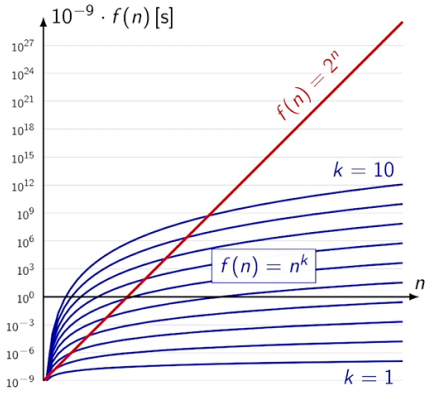
- Simplex-Algorithmus für lineare Optimierung
- Ganzzahlige Optimierung
- Faktorisierung von grossen Zahlen (RSA)
- Diskreter Logarithmus (Diffie-Hellmann)
- Hash-Kollision

Annahme
Zykluszeit 1 ns

Laufzeit = Anzahl Schritte

n	n^5	2^n
1	1ns	2ns
2	32ns	4ns
4	1.0μs	16ns
8	32.7μs	256ns
16	1.0ms	65.5μs
32	33.5ms	4.3ms
64	1.0s	584 Jahre
128	34.4s	10^{21} Jahre

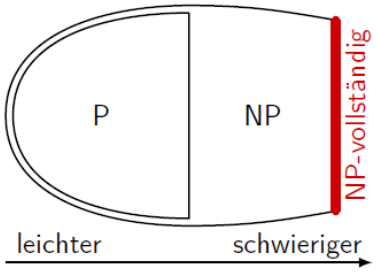
abhängig von der Inputlänge n



10.2. KLASSE P UND NP

Die ***Klasse P*** besteht aus den Sprachen, die mit einem ***Entscheider*** (DTM) in ***polynomieller Laufzeit entschieden*** werden können. Ist eine Teilmenge von NP.

Die ***Klasse NP*** besteht aus Sprachen, die mit einer ***nichtdeterminisitischen Turingmaschine*** (NTM) in ***polynomieller Laufzeit entschieden*** werden können. Eine Sprache ist genau dann in NP, wenn sie in ***polynomieller Zeit verifiziert werden kann***.



10.2.1. Polynomieller Verifizierer (Gleichbedeutend mit NP)

Ein polynomieller Verifizierer ist eine Turingmaschine, ***die die «Lösung» überprüfen*** kann.
Für die Sprache L ist das eine Turingmaschine V, so dass es für jedes $w \in \Sigma^*$ ein Wort c (das Lösungszertifikat) gibt, für das gilt:
$w \in L \iff V$ akzeptiert (w, c)

Die Laufzeit von V ist polynomiell in $|w|$.

Zertifikat *ist wie ein Wunschzettel, wo man alles angeben kann, dass einem bei der Verifikation hilft.*

Verifizierer für: _____

Entscheidbar?

Zertifikat?

Verifikationsalgorithmus

Nr.	Was ist zu tun?	Aufwand
1		
2		
3		
4		
5		
	Total	

Beispiele

Sudoku

Verifizierer für: Sudoku

Entscheidbar?
Ja, alle $\leq (n^2)^{n^2 \times n^2}$ Möglichkeiten durchprobieren

Zertifikat? $c =$ fehlende Ziffern

Verifikationsalgorithmus

Nr.	Was ist zu tun?	Aufwand
1	Für jedes Feld: Zeichen kommt auf der Zeile nicht mehr vor	$O(n^4 \cdot n^2)$
2	Für jedes Feld: Zeichen kommt in der Spalte nicht mehr vor	$O(n^4 \cdot n^2)$
3	Für jedes Feld: Zeichen kommt im Unterquadrat nicht mehr vor	$O(n^4 \cdot n^2)$
4		
5		
	Total	$O(n^6)$

Nurikabe

Verifizierer für: NURIKABE

Entscheidbar?
n= uv für die Anzahl Felder. Man kann alle 2^n möglichen Belegungen des Spielfelds mit schwarzen Felden prüfen ob es die Regeln einhalten. Braucht exponentielle Zeit, aber ist Entscheidbar

Zertifikat?

Verifikationsalgorithmus

Nr.	Was ist zu tun?	Aufwand
1	Für jedes Zahlfeld ($O(n)$) verwendet man einen Markieralgorithmus, der alle zum Gebiet dieses Zahlfeldes gehörenden weissen Felder bestimmt ($O(n)$ Durchgänge mit Aufwand $O(n)$).	$O(n^3)$
2	Trifft dieser Algorithmus auf ein weiteres Zahlfeld, ist der erste Teil von Regel 1 verletzt ($\rightarrow q_{\text{reject}}$).	$O(n)$
3	Weicht die Zahl der gefundenen weissen Felder vom Inhalt des Zahlfeldes ab, ist der zweite Teil von Regel 1 verletzt ($\rightarrow q_{\text{reject}}$).	$n \cdot O(n)$
4	Ebenfalls mit einem Markieralgorithmus wird überprüft, ob das schwarze Gebiet zusammenhängend ist.	$O(n^2)$
5	Für jedes schwarze Feld wird überprüft, ob es Teil eines 2×2-Tümpels ist.	$O(n)$
	Total	$O(n^3)$

Beim Aufstellen des Verifizierers immer auch ***prüfen*** (und begründen), ob das Problem ***überhaupt entscheidbar*** ist. Die Tatsache, dass es ein Zertifikat gibt, reicht dazu nicht aus. Man muss spezifizieren, ***was*** als Zertifikat angefordert wird. Am Ende muss eine ***Antwort auf die gestellte Frage*** stehen. Es reicht nicht, einfach nur die Schritte hinzuschreiben.

Hier geht es darum, ob eine Sprach in NP ist. Das Wort «NP-Vollständig» hat hier nichts verloren!

10.2.2. Reduktion

Dient dazu, zwei Probleme zu vergleichen.

- ***Fach ⟺ Knoten***
- ***Anmeldung ⟺ Kante***
- ***Zeitslot ⟺ Farbe***

Polynomielle Reduktion

Polynomieller Laufzeit-Vergleich

Seien A und B entscheidbare Sprachen. Eine berechenbare Abbildung

$$f : \Sigma^* \to \Sigma^* : w \mapsto f(w)$$

mit

1. $w \in A \Leftrightarrow f(w) \in B$ (Reduktion)
2. $f(w)$ ist berechenbar in polynomieller Zeit in $|w|$

heisst *polynomielle Reduktion* $A \leq_P B$. Lies: A ist polynomiell leichter entscheidbar als B.

Stundenplan und k-VERTEX-COLORING

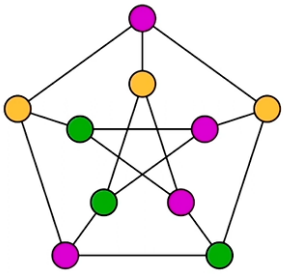
Stundenplan ↔ k-VERTEX-COLORING

Gegeben
- Fächer
- Anmeldungen von Studenten für Fächer
- k Zeitslots pro Woche

Gesucht
Stundenplan = Zuordnung von Zeitslots zu Fächern derart, dass jeder Student jedes angemeldet Fach besuchen kann

Gegeben
- Graph G
- k Farben

Gesucht
Kann man die Knoten so mit k Farben einfärben, dass benachbarte Knoten verschiedene Farbe haben?



10.3. NP-VOLLSTÄNDIG

Eine Entscheidbare Sprache B heisst NP-Vollständig, ***wenn sich jede Sprache A in NP polynomiell auf B reduzieren lässt***. Sind die ***schwersten Probleme*** in NP *Katalog von Karp*

Es genügt nicht, das Vergleichsproblem zu erkennen, es muss auch eine 1-1-Reduktion durchgeführt werden. Und am Ende muss eine Antwort auf die Frage stehen. Und irgendwo muss das Wort NP-vollständig stehen, NP reicht nicht!

11. KATALOG VON KARP

Probleme mit K Zahlen
- K-CLIQUE
- SET-PACKING
- VERTEX-COVER
- FEEDBACK-*-SET
- SET-COVERING
- CLIQUE-COVER
- VERTEX-COLORING
- SEQUENCING
- STEINER-TREE
- (MAX-CUT)

Aufteilung in zwei Teilmengen
- PARTITION
- MAX-CUT

Probleme mit Zahl 3
- 3D-MATCHING
- 3SAT

Unterschiede Hitting-Set, Exact-Cover
- HITTING-SET: Menge von Punkten
- EXACT-COVER: Menge von Teilmengen

Unterschiede Feedback-*-Set
- Node: Vertex entfernen
- Arc: Kanten entfernen

Unterschiede Set-*
- Covering: Endliche Familie endlicher Mengen
- Packing: Eine Familie von Mengen

	SET-COVERING	*SET-PACKING*	*EXACT-COVER*
Anzahl Mengen	k	k	
Vereinigung	ganze Menge		ganze Menge
paarweise Schnittmengen		$\varnothing$	$\varnothing$

Unterschiede *-Cover

Graph oder Mengen mit einer Zahl k

SAT(Logik), 3SAT (Logik)

Gibt es eine mögliche Lösung der **Booleschen Gleichung**, damit sie TRUE ergibt.

3SAT: Klausel besteht höchstens aus 3 Literalen pro Klausel und ist in KNF

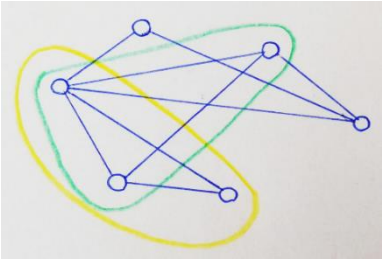
$F = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee \overline{x_2})$

Klausel = Klammerausdruck, Literale = Variable

Bsp. Elektriker Variablen: n-Schalter -> n-Variablen, m-Räume -> m-Klauseln, Überall Licht -> Aussage = true, Stromkreise -> Wertebereich (True, False)

Jedes Ausfüllrätsel ist mit SAT beschreibbbar. Man kann Wahrheitstabellen bilden. NP-Vollständig ist jedes Problem. Das sich auf SAT reduzieren lässt.

k-CLIQUE



Gibt es k Knoten, die alle miteinander verbunden sind?

k: Anzahl Vertizes in einere Clique (gegeben)
Gibt es eine solche Clique in dem Graphen (Siehe Bild)?
für $k = 3 \rightarrow$ ja (eingezeichnet)
für $k = 4 \rightarrow$ nein

Beispiel: *Job-Parallelisierbarkeit: gegeben eine Menge von n Jobs, die jeweils exklusiv auf m Ressourcen zugreifen, Jobs dürfen nicht gleichzeitig laufen. Zeigen Sie, dass das Problem zu entscheiden, ob mit diesen Jobs zu irgendeinem Zeitpunkt mehr als k Prozessoren ausgelastet werden können, NP-vollständig ist.*

Variablen: *Graph (Job-Parallelisierbarkeit), Knoten (n Job), Kante (m gleiche Ressourcen), k-Clique (Auswahl Knoten ohne Verbindung), k (Auslastbare Anzahl Prozessoren)*

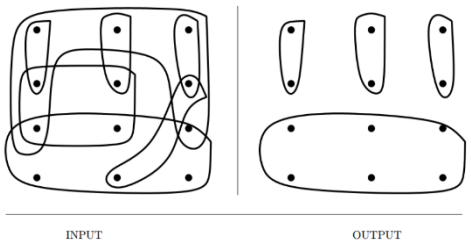
SET-PACKING

Gegeben: eine Familie $(S_i), i \in I$ und eine Zahl $k \in \mathbb{N}$
Gibt es eine k-elementeige Teilfamilie $(S_i), i \in J$ mit $J \subset I$, d. $|J| = k$ derart, dass die Menge der Teilfamilie paarweise diskunkt sind? Gleich wie Exact-Cover, nur Ausgangslage ist anders.

Menge S und Menge T von Teilmengen, Gibt es k Teilmengen in T, welche disjunkt sind und die Menge S abdecken?

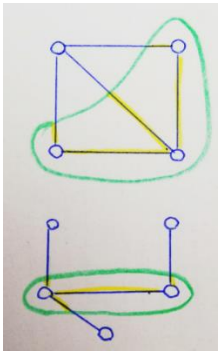
Beispiel 1 Küche: *In einer Küche hat man eine Menge an Zutaten und ein Rezeptbuch voller Rezepte. Nun möchte man möglichst viele der Rezepte kochen, ohne eine Zutat mehrmals zu verwenden.*

Beispiel 2 Medizinstudie: *Für eine medizinische Studie ist eine grosse Zahl von Probanden rekrutiert worden. Sie sind bereits auf Allergien getestet worden, man weiss also von jedem Probanden, auf welche Allergene er allergisch reagiert. Die Untersuchung soll sich auf eine Teilmenge von $k = 17$ oder noch mehr ausgewählten Allergenen beschränken, die so beschaffen ist ,dass kein Proband auf mehr als eines der ausgewählten Allergene reagiert.*



Variablen: *I (Allergene), Si (Auf Allergen i allergische Probanden), Ausschlussbedinungn zwischen Allergenen i und j, Teilmenge*

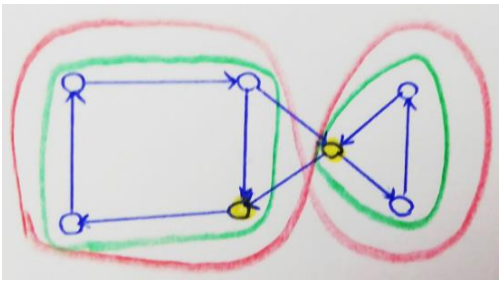
VERTEX-COVER



Gegeben: Ein Graph G und eine Zahl k. Gibt es eine **Teilmenge** von k Knoten so, dass jede **Kante** des Graphen ein Ende in dieser Teilmenge hat?

Beispiel: *Gibt es 4 Knoten, sodass jeder andere Knoten eine Kante zu diesem hat?*
Beispiel: *Ein Verkehrsnetz soll regelmässig durch Mitarbeiter kontrolliert werden, die ihre Basis an einzelnen Knotenpunkten des Netzes haben, Kann man auf effiziente Art herausfinden, an welchen Knotenpunkten man Kontrolleure stationieren muss, damit jede Strecke in einem Knoten mit Kontrolleur endet?*
Variablen: *Knotenpunkte -> Knoten, Strecke -> Kante, Anzahl Kontrolleure -> k*

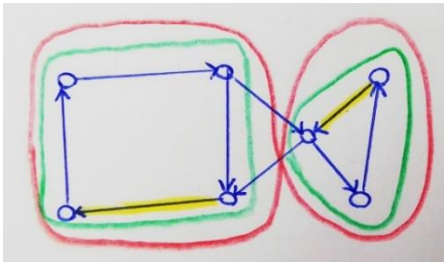
FEEDBACK-NODE-SET



Gegeben: Ein **gerichteter Graph G** und eine **Zahl k**. Gibt es eine endliche Teilmenge von k Vertizes/Knoten von G, sodass jeder Zyklus in G **einen Vertex** in der Teilmenge enthält?

Beispiel: *Es gibt mehrere Buslinien. Wo muss das Putzpersonal platziert werden, damit alle Linien geputzt werden können? Man möchte möglichst wenig Personal einsetzen.*
Bild: $k = 2$, *Alle 3 Umrahmungen = Zyklen. Knoten markieren*
Variablen: *Graph (Plan des Einkaufszentrums), Knoten (Kreuzungsstellen), Kanten (Gänge), Richtung, Anzahl k (Budget für Stationen), Arc set (Platzierungen der Desinfektionsstationen)*

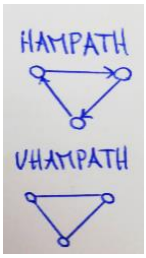
FEEDBACK-ARC-SET



Gegeben: Ein **gerichteter Graph G** und eine **Zahl k**. Gibt es eine endliche Teilmenge von k Kanten von G, sodass jeder Zyklus in G **eine Kante** aus der Teilmenge enthält?

Beispiel: *Es gibt mehrere Buslinien. Das Personal putzt während der Fahrt. Wo muss das Putzpersonal platziert werden, damit alle Linien geputzt werden können? Man möchte möglichst wenig Personal einsetzen.*
Bild: $k = 2$, *Kanten markieren*
Variablen: *Graph (Plan des Einkaufszentrums), Knoten (Kreuzungsstellen), Kanten (Gänge), Richtung, Anzahl k (Budget für Stationen), Arc set (Platzierungen der Desinfektionsstationen)*

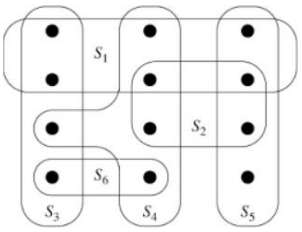
HAPMATH & UHAMPATH



Ein Hamilton-Pfad in einem **gerichteten Graphen** ist ein Pfad, der jeden Knoten **genau einmal** enthält (Haus des Nikolaus). Komme zu jedem Knoten genau einmal.

Ungerichtet ist es der **U**hampath.
Beispiel: *Ganze Schweiz bereisen ohne eine Stadt mehrmals zu besuchen.*
Variablen: *Knoten (Block), gerichtete Kante (Fahrt), Hamiltonischer Pfad*

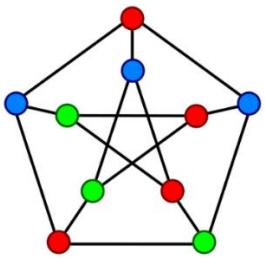
SET-COVERING



Gegeben: Eine **endliche Familie endlicher Mengen** $(S_j), 1 \leq j \leq n$ und eine **Zahl k**. Gibt es eine Unterfamilie bestehend aus k Mengen, die die gleiche Vereinigung hat? Kann man k Teilmengen bilden, welche die Menge S komplett abdecken?

Beispiel: *Von jedem Aussichtspunkt aus kann man eine endliche Anzahl Sehenswürdigkeiten erkennen. Ein Tourist, der es sehr eilig hat, möchte mit dem Besuch von nur k Aussichtspunkten alles sehen, was man an Sehenswürdigkeiten von allen Aussichtspunkten aus sehen könnte. Warum ist es schwierig, für den Touristen eine Auswahl zu treffen?*
Variablen: *Aussichtspunkt Sj, Sichtbare Sehenswürdigkeiten* $\cup_{j=1}^{n} S_j$, *Auswahl von Aussichtspunkten* $S_{j_1}, \ldots, S_{j_k}$, *Bedingung*

VERTEX-COLORING (Planungsproblem)



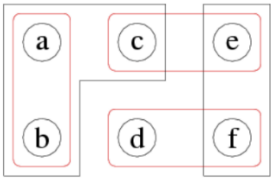
Kann man die Knoten so mit k Farben einfärben, dass benachbarte Knoten verschiedene Farben haben?

Beispiel: *Job-Planung*
Variablen: *n Job -> Knoten, m gleiche Ressourcen -> Verbindungen, parallele Jobs (bzw. Intervall) -> gleiche Farbe, N Intervalle -> n Farben*

EXACT-COVER

Gegeben: Eine **Familie** $(S_j), 1 \leq j \leq n$ **von Teilmengen** einer **Menge U**. Gibt es eine Unterfamilie von Mengen, die disjunkt sind, aber die gleiche Vereinigung haben? Jedes Element in U soll genau in einer der Teilmengen der Familie S vorkommen. Die gesuchte Menge bildet eine exakte Überdeckung.

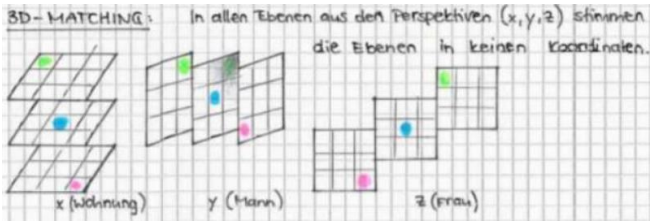
Beispiel: *Student Xaver Tecco soll im Rahmen einer Big-Data-Studienarbeit die Kunden einer grossen Shop-Website untersuchen und klassifizieren. Es steht eine grosse Zahl von binären Eigenschaften zur Verfügung, zum Beispiel ob Kunden ein bestimmtes Produkt gekauft haben, oder ob ein Kunde nur im Dezember einkauft. Herr Tecco soll herausfinden, ob es eine Teilmenge von Kriterien gibt, dass jeder Kunde genau eine der Eigenschaften hat.*
Variablen: *Eigenschaft -> Menge* S_j*, Teilmenge von Eigenschaften -> Unterfamilie* S_{ji}*, Genau eine der Eigenschaften, Alle Kunden erfasst*



3D-MATCHING

Gegeben: **Endliche Menge** T und Menge U von **Tripeln** $T \times T \times T$. Gibt es eine **Teilmenge** W von U so, dass $|W| = |T|$ und keine zwei Elemente von w stimmen in irgendeiner Koordinate überein?

Beispiel: *Ein Koch hat je n Rezepte für Vorspeisen, Hauptspeisen und Desserts. Nicht alle Vorspeisen lassen sich mit jeder Hauptspeise kombinieren, dasselbe gilt auch für Desserts. Damit jeder seiner Rezepte regelmässig zum Einsatz kommt, möchte der Koch eine Folge von n Menus zusammenstellen, sodass jedes Rezept in genau einem der Menus vorkommt.*
Variablen: *Nummern der Rezepte -> Menge T, Menuzusammenstellungen -> Tripel aus* $T \times T \times T$*, Menge der möglichen Menuzusammenstellungen -> Menge U, Gesuchte Menus n -> Teilmenge W, so* $n = |T| = |W|$



SUBSET-SUM (RUCKSACK-PROBLEM)

Gegeben: Menge S von ganzen Zahlen. Kann man eine Teilmenge finden, die als **Summe** einen bestimmten **Wert t** hat?

Beispiel: *Rucksack-Problem*
Variablen: *Grösse der Gegenstände -> Zahl, Menge der Gegenstände -> Menge S der Zahlen, Grösse des Rucksacks -> Wert t (Summe), Gegenstände im Rucksack -> Teilmenge*

SEQUENCING

Gegeben: Ein **Vektor** von Laufzeiten von p **Jobs**, ein Vektor von spätesten **Ausführzeiten**, ein **Strafenvektor** und eine **positive Zahl k**.
Gibt es eine Permutation der Zahlen 1, …, p sodass die Gesamtstrafe für verspätete Ausführung bei der Ausführung der Jobs nicht grösser ist als k? Gegeben: Eine Menge an 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. Gesucht: Eine Reihenfolge von Jobs so, dass die Strafe kleiner gleich k ist.

Beispiel: *Eine Firma hat eine bestimmte Anzahl laufende Verträge. Der Firma ist es nicht möglich, alle Verträge in einer bestimmten Zeit abzuarbeiten. Sie versucht also Schadensbegrenzung zu machen, indem sie möglichst viele Verträge in der verbleibenden Zeit abarbeitet, die eine hohe Strafe zur Folge haben.*
Variablen: *Ausführungszeit von Job, Deadline, Kosten, Permutation / Reihenfolge*

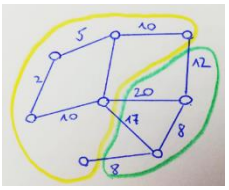
PARTITION

Gegeben: Eine **Folge von S ganzen Zahlen** $c_1, c_2, \dots, c_s$
Kann man die Indizes 1,2,…,S in zwei Teilmengen I und $\bar{I}$ teilen, sodass die Summe der zugehörigen Zahlen c_i identisch ist? Gibt es **zwei disjunktive Teilmengen** mit der **gleichen Summe**?

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?*

MAX-CUT

Gegeben ein Graph G mit einer **Gewichtsfunktion** $w: E \to \mathbb{Z}$ und eine Zahl W
Gibt es eine **Teilmenge** S der Vertizes, sodass das Gesamtgewicht der Kanten, die S mit seinem Komplement verbinden, mindestens so gross ist wie W?
Der Max-Cut eines Graphen ist eine Zerlegung seiner Knotenmenge in zwei Teilmengen, sodass das Gesamtgewicht der zwischen den beiden Teilen verlaufenden Kanten mindestens W wird.



Beispiel: *Feindliche Übernahme einer Firma, mit resultierender Aufteilung der Abteilung, dass diese möglichst ineffizient miteinander kommunizieren können.*
Variablen: *Abteilung -> Vertex, Kommunikationsbeziehung -> Kante, Kommunikationsvolumen -> Gewicht einer Kante*

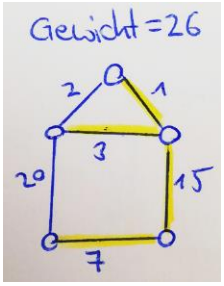
HITTING-SET

Gegeben: Eine **Menge von Teilmengen** $S_i \subset S$. Gibt es eine Menge H, die jede Menge in genau einem Punkt trifft, also $|H \cap S_i| = 1$

Gegeben: $A = \{1,2,\mathbf{3}\}, B = \{1,2,\mathbf{4}\}, C = \{1,2,\mathbf{5}\}$
Gesucht: $H = \{\mathbf{3},\mathbf{4},\mathbf{5}\}$
Variablen: *I (Teilmenge), i (Elemente in Teilmenge), S_i (Menge von Teilmengen), H (Resultat)*

STEINER-TREE

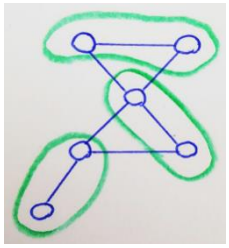
Gegeben: Ein Graph G, eine **Teilmenge R von Knoten**, eine **Gewichtsfunktion** und eine **positive Zahl k**.
Gibt es einen Baum mit Gewicht $\leq k$, dessen Knoten in R enthalten sind? Das Gewicht des Baumes ist die Summe der Gewichte über alle Kanten im Baum.



Beispiel: *Bau einer Zugstrecke oder Stromnetzes:*
Variablen: *Stromnetz / Zugstrecke -> Steiner-Tree, Ortschaften -> Knoten, zu erschliessende Ortschaften -> Knoten in R, Baukosten -> Gewicht w einer Kante, Budget -> maximales Gewicht k*

CLIQUE-COVER (EXOR mit bestimmter Zahl k)

Gegeben: Ein Graph G und eine **positive Zahl k**. Gibt es k Cliquen so, dass jede Ecke in genau einer der Cliquen ist?



Beispiel: *Für eine Gruppenarbeit sollen k Gruppen gebildet werden. Um die Zeit für das Kennenlernen möglichst kurz zu halten, sollen sich die Leute einer Gruppe bereist kennen. Alle Leute sollen eingeteilt sein.*
Variablen: *Teilnehmer -> Knoten, Kennen sich -> Kante, Anzahl Gruppen -> k, Gruppe -> Clique*

BIP

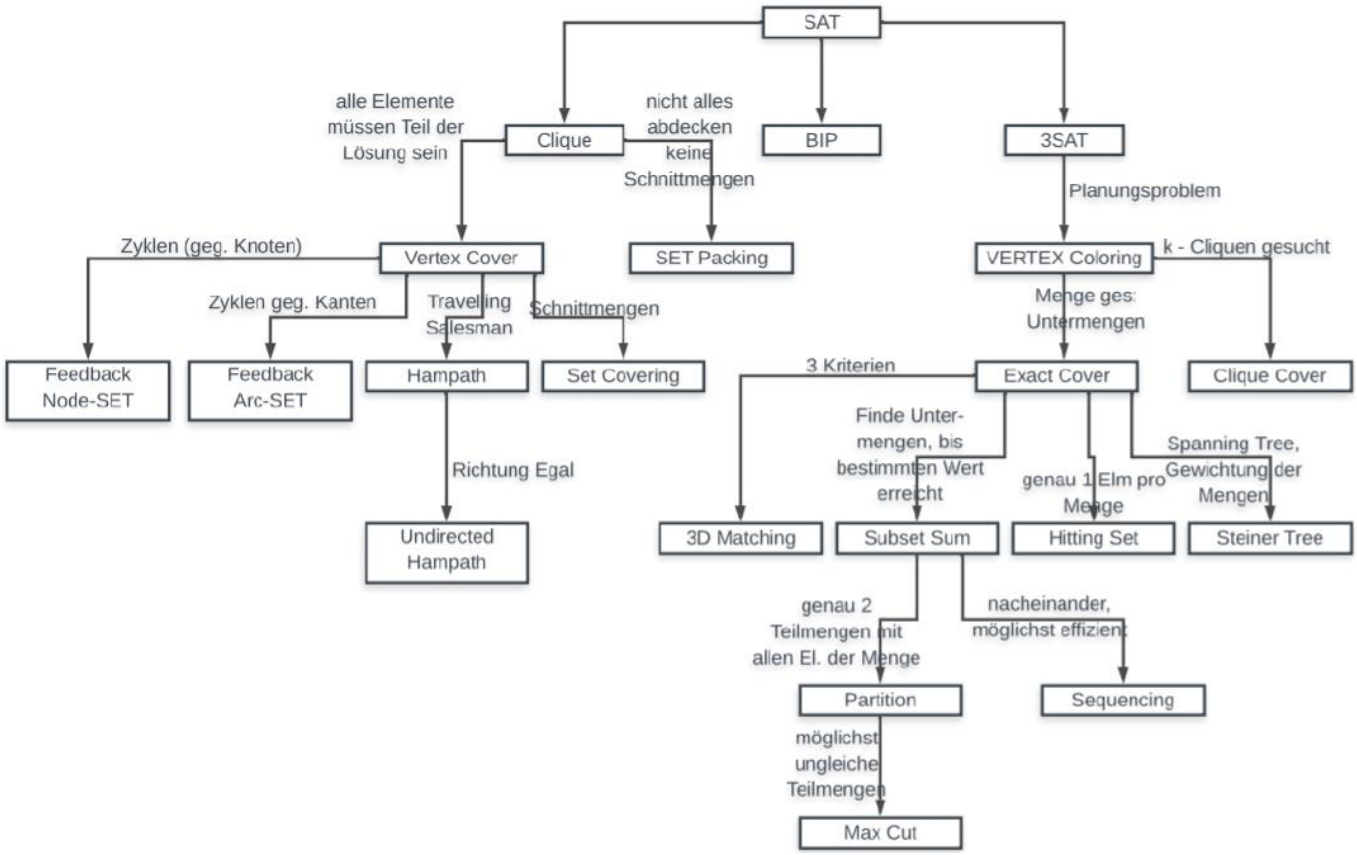
Zu einer ganzzahligen Matrix C und einem gannzahligen Vektor d, ist ein binärer Vektor x zu finden mit

$$C * x = d$$

$$\begin{bmatrix}1 & 3 & 0\\ 0 & 2 & 5\end{bmatrix} * \begin{bmatrix}1\\0\\1\end{bmatrix} = \begin{bmatrix}1\\5\end{bmatrix}$$

11.1. VORGEHEN BEI DER PRÜFUNG

- ***Problem aus Liste suchen***, dabei Punkte beachten wie: Ist es ein Mengenproblem, Planungsproblem, kommen k-Zahlen vor, Teilmengen?
- ***Reduktion des Problems*** auf Karp mit Pfeil-Liste darstellen (Eigenschaften)
- ***Zusammenfassung schreiben:*** «Problem x ist äquivalent mit Karp-Katalog-Problem y und deshalb gehört es zu den NP-Vollständigen Problemen. Daher ist es nicht einfach entscheidbar (Es gibt derzeit keinen polynomiellen Algorithmus).»



12. TURING-VOLLSTÄNDIGKEIT

Turing-Maschinen und moderne Computer haben einige ***Gemeinsamkeiten***. Ein ***Computer erfüllt alle Eigenschaften einer Turing-Maschine*** (Zustände, Band, Schreib-/Lesekopf). Einzig kann er ***mehrere Programme*** ausführen und eine ***Turing-Maschine ist problemspezifisch***. Es gibt aber die Universelle Turing-Maschine mit

- Eigenem Band für Codierung der Übergangsfunktionen
- Eigenem Band für aktuellen Zustand
- Arbeitsband

Eine solche Mehrbandmaschine kann auch auf einer Standard-Turingmaschine simuliert werden.
Es gibt aber Komponenten, die ein PC hat, eine Turing-Maschine aber nicht.

- ***Persistenter Speicher:*** Files nicht unterscheidbar von Daten, für TM egal
- ***Interaktion:*** Lösung eines spezifizierten Problems braucht keine Interaktion
- ***Input / Output:*** Output ist nicht wesentlich. Input «existiert nicht». Wenn TM angehalten wird, kann Kernel Band «manipulieren» und TM sieht nun geänderter Input, weiss aber nicht, ob er den Input selber geändert hat oder die Veränderung von aussen kommt.

12.1. VERGLEICH VON MASCHINEN

Eine TM M_1 ist «***leistungsfähiger***» als eine TM M_2, wenn M_1die Maschine M_2 ***simulieren*** kann.
$M_2 \leq M_1 \rightarrow M_2$ ist simulierbar auf M_1

12.2. PROGRAMMIERSPRACHEN

Eine Programmiersprache ist Turing-Vollständig, wenn sie folgende Eigenschaften erfüllt:

- Es lässt sich ein ***Infinite Loop*** erstellen
- Sie kann auf einen ***«unendlich» grossen Speicher*** zugreifen
- ***Bedingte Ausführung*** ist möglich

13. GLOSSAR

- Die Sprache $L \subset \Sigma^*$ heisst ***regulär***, wenn es einen DEA A gibt, mit $L(A) = L$.
  Beispiel: $\Sigma = \{0,1\}, L = \{w \in \Sigma^* | \, |w|_0 gerade\}$
- Die Sprache $L \subset \Sigma^*$heisst ***nicht regulär***, wenn es keinen DEA A gibt, mit $L(A) = L$.
  Beispiel: $0^n 1^n, n \geq 0$
- Die Sprache $L \subset \Sigma^*$ heisst ***kontextfrei***, wenn die Sprache nur von einem nichtdeterministischen Stackautomaten erkannt werden kann
- ***Beispiel:*** $\Sigma = \{(,)\}, L = \{w \in \Sigma^* | \, w \, ist \, ein \, korrekter \, Klammerausdruck\}$
- Die Sprache $L \subset \Sigma^*$ heisst ***nicht kontextfrei***, wenn das Pumping Lemma für Kontexfreie Sprachen einen Widerspruchbeweis liefert.
  Beispiel: $L = \{a^n b^n c^n \mid n \geq 0\}$
- Eine Sprache L heisst ***«Turing-Erkennbar»***, wenn es eine Turing-Maschine M gibt, die das Wort akzeptiert.
  Beispiel: $L = \{a^n b^n c^n \mid n \geq 0\}$

14. ANHANG

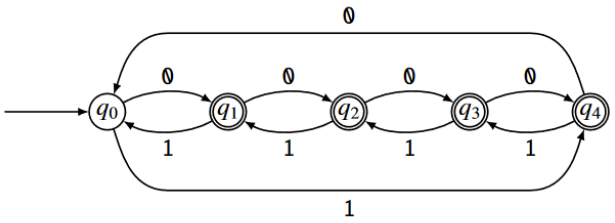
14.1. DEA

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



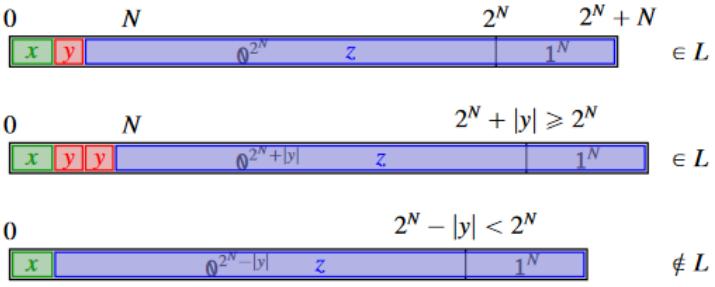
beschreibt diesen Automaten. Der Zustand q_k codiert den Fünferrest k der Differenz $|w|_0 - |w|_1$. ○

Wenn gleich statt ungleich, einfach Akzeptier- und nichtakzeptierzustände umkehren

14.2. PUMPING LEMMA

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

1. Annahme ist regulär.
2. Nach dem Pumping Lemma gibt es die Pumping Length N.
3. Wähle das Wort $w = 0^{2^N}1^N \in L$
4. Es gibt eine Aufteilung des Wortes $w = xyz$ mit $|xy| \leqslant N$ und $|y| > 0$. Der Teil y besteht ausschliesslich aus Nullen.



5. Beim Aufpumpen nimmt die Zahl der Nullen zu, dies steht aber nicht im Widerspruch zu der Sprachdefinition. Beim Abpumpen nimmt jedoch die Zahl der Nullen ab, es ist dann $|xz|_w = 2^N - |y|$. Die Zahl der Einsen ist immer noch N, damit haben wir

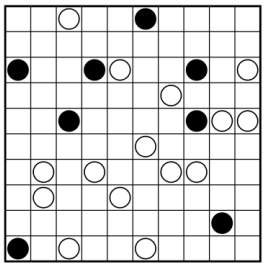
   $$|xy|_0 = 2^N - |y| < 2^N = |xy|_1,$$

   im Widerspruch zur Aussage des Pumping Lemmas
6. Dieser Widerspruch zeigt, dass die Sprache L nicht regulär ist. ○

Es gibt natürlich viele weitere Möglichkeiten, den gesuchten Widerspruch herzustellen. Das oben gewählte Wort ist möglicherweise dasjenigen, welches die meisten Studenten als erstes wählen würden, aber das Wort $w = 1^N0^{2^N}$ ist noch etwas einfacher. Beim Aufpumpen nimmt die Anzahl der Einsen zu, die Anzahl der Nullen aber nicht. Nach Sprachdefinition müsste aber auch die Anzahl der Nullen zunehmen, da ja $|w|_0 \geqslant 2^{|w|_1}$ sein muss. Der Widerspruch zeigt wieder, dass L nicht regulär sein kann.

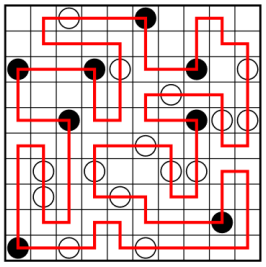
Bewertung. Pumping-Lemma-Beweis: jeder Schritt ein Punkt: Annahme regulär (**PL**) 1 Punkt, Pumping Length (**N**) 1 Punkt, Wort (**W**) 1 Punkt, Aufteilung (**A**) 1 Punkt, Widerspruch beim Pumpen (**P**) 1 Punkt, Schlussfolgerung (**S**) 1 Punkt. Falls das gleiche Wort wie in der Musterlösung gewählt wurde wird der Punkt für Schritt 5 nur gegeben, wenn erkannt worden ist, dass nur Abpumpen zu einem Widerspruch führt.

14.3. VERIFIZIERER



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° abbiegen.
4. Schwarze Kreise liegen auf einer Kurve, die um 90° 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.	$O(\frac{n(n-1)}{2})$
3*a*	Kontrollieren, ob die Nachbarn eines Feldes mit weissem Kreis diametral gegenüberliegen.	$O(n^2)$
3*b*	Kontrollieren, ob auf einem der Nachbarfelder eines Feldes mit weissem Kreis eine 90°-Abbiegung erfolgt	$O(n^2)$
4	Kontrolliere, ob auf jedem Element des Pfades mit einem schwarzen Feld die Nachbarelemente eine 90°-Abbiegung bilden.	$O(n^2)$
	Total	$O(n^2)$

Damit ist gezeigt, dass der Verifizierer polynomielle Laufzeit hat. Somit ist Masyu in NP. ○

14.4. SATZ VON RICE

In vielen Anwendungen wird verlangt, dass der Output sortiert wird. Es wäre daher nützlich für die Qualitätssicherung, wenn man ein Tool schreiben könnte, welches von einem Programm entscheiden kann, ob sein Output korrekt sortiert wird. Viele Programmiersprachen haben zum Sortieren Funktionen oder Klassen, die Daten sortieren können, man könnte testen, ob diese Klassen vom Code verwendet werden. Manchmal fallen die Daten aber auch automatisch sortiert an, zum Beispiel wenn in einer Datenbank ein Index verwendet wird. Es reicht also nicht, nur die Verwendung der genannten Klassen zu prüfen. Ist es möglich, so ein Tool zu schreiben?

Nehmen Sie der Einfachheit halber an, dass das Tool nur auf Programme angewendet werden soll, welche Wörter aus Zeichen in $\Sigma = \{a, b, \ldots, z\}$ erzeugen, und die Zeichen in einem Wort sollen alphabetisch aufsteigend sortiert sein.

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

$$L_1 = \{\mathtt{ab}\}$$
$$L_2 = \{\mathtt{ba}\}$$

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. ○

14.5. CHOMSKY-NORMALFORM

Ist die Sprache

$$L = \{\mathtt{0}^i\mathtt{1}^j\mathtt{0}^k \mid j > i > 0 \wedge k > 0\}\}$$

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

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

$$A \to \mathtt{0}A\mathtt{1} \mid \mathtt{01},$$

für die Wörter der zweiten Art können wir

$$B \to \mathtt{1}B \mid B\mathtt{0} \mid \mathtt{10}$$

verwenden. Damit finden wir die Grammatik

$$S \to AB$$
$$A \to \mathtt{0}A\mathtt{1} \mid \mathtt{01}$$
$$B \to \mathtt{1}B \mid B\mathtt{0} \mid \mathtt{10}$$

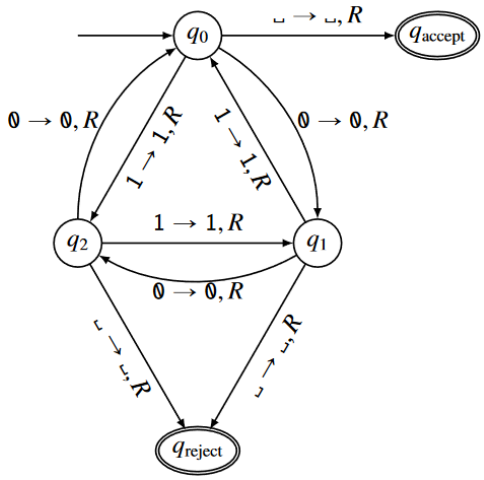
Diese Grammatik hat nicht Chomsky-Normalform. Aber ihre Startvariable kommt auf der rechten Seite nicht vor und sie hat weder ε-Regeln noch Unit Rules. Es bleibt daher nur noch, die Terminalsymbole und die Dreierregeln zu "flicken":

$$S \to AB$$
$$A \to NU \mid NE$$
$$U \to AE$$
$$B \to EB \mid BN \mid EN$$
$$N \to \mathtt{0}$$
$$E \to \mathtt{1}$$

Damit hat die Grammatik Chomsky-Normalform. ○

14.6. ZUSTANDSDIAGRAMM ANALYSIEREN

Betrachten Sie die Turing-Maschine M mit dem Zustandsdiagramm rechts.



- **Wie wird das Wort 001011 verarbeitet?**
  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_{accept}	⊔	0	0	1	0	1	1	⊔

- **Welche der Wörter 01, 01000, 01000000, 111, 0111, 000111, 000111111, 00010101111 werden akzeptiert?**
  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 Zustandsdiagramm im Urzeigersinn, mit jedem 1 auf dem Band im Gegenuhrzeigersinn. Akzeptiert wird, wenn der Zustand am Ende des Wortes q_0 ist. Damit findet man schnell, dass

  $$01,\ 01000,\ 01000000,\ 111,\ 000111\ 000111111$$

  akzeptiert und

  $$0111,\ 00010101111$$

  verworfen werden.

- **Welche Sprache wird von M akzeptiert?**
  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

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

Skizzieren Sie einen Algorithmus ….

Der folgende Algorithmus liefert einen regulären Ausdruck der verlangten Art:

1. Für die vorgegebene Wortlänge, verwende den analog zu Teilaufgabe a) gefundenen regulären Ausdruck und erzeuge daraus einen DEA.
2. Für jedes Zeichen mit bekannter Position erzeuge nach dem Muster von Teilaufgabe b) einen DEA, der genau die Wörter akzeptiert, die diese Bedingung erfüllen.
3. Für jedes Zeichen, das vorkommt, erzeuge nach dem Muster von Teilaufgabe c) einen DE, der genau die Wörter akzeptiert, die diese Bedingung erfüllen.
4. Aus den DEAs der Schritte 1–3 erzeuge den Produktautomaten, einen DEA, der genau die Wörter akzeptiert, die alle Bedingungen erfüllen.
5. Wandle den DEA von Schritt 4 mit dem in der Vorlesung dargestellten Algorithmus in einen regulären Ausdruck um. ○

Eine *endliche* Sprache hat nur endlich viele Wörter.

a) Wie kann man entscheiden, ob ein regulärer Ausdruck eine endliche Sprache akzeptiert?

b) Wie kann man mit einem Algorithmus entscheiden, ob ein DEA eine endliche Sprache akzeptiert?

c) Man nennt eine Turing-Maschine *endlich*, wenn sie eine endliche Sprache akzeptiert. Können Sie einen Algorithmus konstruieren, der entscheidet, ob eine Turingmaschine M endlich ist, d. h. eine endliche Sprache akzeptiert?

Lösung. a) Ein regulärer Ausdruck akzeptiert genau dann eine endliche Sprache, wenn er keinen Stern enthält.

b) Ist r der reguläre Ausdruck, der die gleiche Sprache akzeptiert wie der DEA A, dann ist die Sprache $L(r) = L(A)$ genau dann endlich, wenn der Ausdruck r keinen Stern enthält.

c) Nein, dies ist wegen des Satzes von Rice nicht möglich. Die Eigenschaft, endlich zu sein, ist eine nichttriviale Eigenschaft von Sprachen. Die leere Sprache $L_1 = \varnothing$ ist endlich, die Sprache $L_2 = \Sigma^*$ ist unendlich. Nach dem Satz von Rice ist die Eigenschaft, eine endliche Sprache zu akzeptieren, nicht entscheidbar, es kann also keinen Algorithmus der verlangten Art geben. ○

14.7. PUMPING LEMMA 2

Im Spiel *Corral* (vergleiche auch Aufgabe 3) muss ein geschlossener Pfad beschrieben werden. Eine Möglichkeit, einen solchen Pfad zu spezifizieren, ist, zunächst die Koordinaten des Startpunkts anzugeben und anschliessend eine Folge von Buchstaben aus der Mengen $\Sigma = \{\mathtt{u}, \mathtt{d}, \mathtt{l}, \mathtt{r}\}$, wobei jeder Buchstabe für ein Segment des Pfades steht, welches nach oben (u), unten (d), links (l) bzw. rechts (r) zeigt. Sie L die Sprache

$$L = \{w \in \Sigma^* \mid w \text{ beschreibt einen geschlossenen Pfad}\}$$

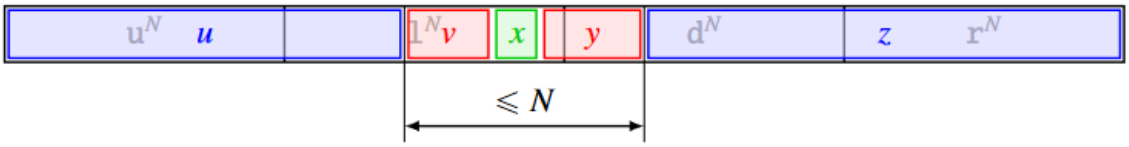
der Wörter, die geschlossene Pfade beschreiben. Können Sie eine kontextfreie Grammatik für L angeben?

Lösung. Damit ein Pfad geschlossen ist, müssen gleich viele u wie d und gleichviele l wie r in w vorkommen, die Sprache L ist also auch

$$L = \{w \in \Sigma^* \mid |w|_\mathtt{u} = |w|_\mathtt{d} \wedge |w|_\mathtt{l} = |w|_\mathtt{r}\} \tag{1}$$

Eine kontextfreie Grammatik für L gibt es aber nicht, denn L ist nicht kontextfrei, wie man mit dem Pumping Lemma für kontextfreie Sprachen zeigen kann.

1. Wir nehmen an, dass L kontextfrei ist.
2. Nach dem Pumping Lemma für kontextfreie Sprachen gibt es die Pumping Length N
3. Wir nehmen das Wort $w = \mathtt{u}^N\mathtt{l}^N\mathtt{d}^N\mathtt{r}^N$, es beschreibt ein im Gegenuhrzeigersinn durchlaufenes Quadrat mit Seitenlänge N.
4. Nach dem Pumping-Lemma gibt es eine Aufteilung des Wortes in fünf Teile $w = uvxyz$



mit $|x| > 0$ und $|vxy| \leqslant N$ derart, dass auch die aufgepumpten Wörter $uv^k xy^k z \in L$ sind.

5. Wegen $|vxy| \leqslant N$ kann sich beim Pumpen höchstens die Anzahl von zwei der Buchstaben ändern, niemals aber die Anzahl beider Buchstaben gleichzeitig, die jeweils gleich sein sollte.
6. Dieser Widerspruch zeigt, dass L nicht kontextfrei sein kann. ○

Bewertung. Bedingung (1) **(B)** 1 Punkt, Pumping Lemma und Pumping Length **(N)** 1 Punkt, Wahl des Wortes **(W)** 1 Punkt, Unterteilung **(U)** 1 Punkt, Widerspruch beim Pumpen **(P)** 1 Punkt, Schlussfolgerung L nicht kontextfrei **(S)** 1 Punkt.

Gegeben ist das Alphabet Σ bestehend aus den Buchstaben A–Z in den Farben schwarz, grün und oliv, so wie sie im Spiel *Wordle* verwendet werden (siehe auch Aufgabe 1).

a) Zeichnen Sie das Zustandsdiagramm einer Turing-Maschine, die ein auf dem Band eingegebenes, fünf Zeichen langes Wort von links nach rechts liest, ähnlich wie im Spiel mit dem Wort OCHSE[1] vergleicht und dabei die Zeichen genau wie folgt modifiziert:

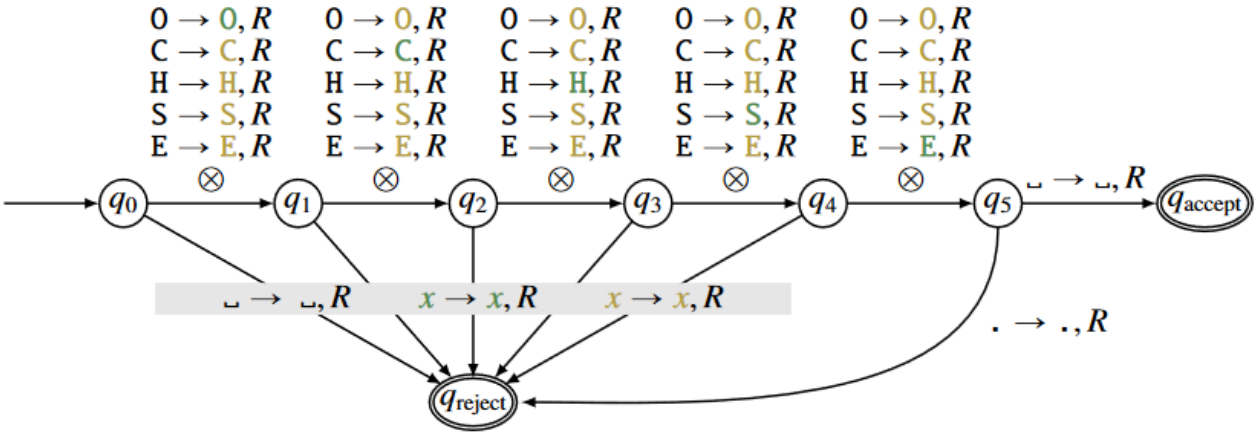
- Hat ein Zeichen bereits eine der Farben grün oder oliv, wird der Input verworfen.
- Ist das Zeichen richtig, soll es grün werden.
- Ist das Zeichen an dieser Stelle falsch, kommt aber an einer anderen Stelle im Wort vor, wird es oliv.

Falls das Wort auf dem Band nicht exakt fünf Zeichen hat, wird es verworfen.

Lösung. Das Alphabet besteht aus den Zeichen

$$\Sigma = \{\mathtt{A}, \mathtt{B}, \ldots, \mathtt{Z}, \mathtt{A}, \mathtt{B}, \ldots, \mathtt{Z}, \mathtt{A}, \mathtt{B}, \ldots, \mathtt{Z}, \textvisiblespace\}.$$

a) Das folgende Zustandsdiagramm löst die Aufgabe:



Dabei bedeutet das Zeichen ⊗ den Übergang $x \to x, R$ für jedes schwarze Zeichen x, welches nicht in $\{\mathtt{O}, \mathtt{C}, \mathtt{H}, \mathtt{S}, \mathtt{E}\}$ ist. Im grauen Balken sind ausserdem mit x und x alle farbigen Zeichen gemeint. Im Übergang vom Zustand q_5 in den Zustand q_{reject} bedeutet der Punkt . alle Zeichen ausser dem ␣.