

1. Kapitel Sprachen und Grammatiken

1.1 Grammatik

1.1.1 Alphabet und Wörter

Definition 1.1.1.1

Definition 1.1.1.2

Definition 1.1.1.3

Operationen auf Wörter

Potenz von Wörtern

Relationen für Wörter

1.1.2 Formale Sprache

Automaten und Berechenbarkeit > Definition 1.1.2.1

Operationen auf Sprachen

1.2 Einführung

2. Kapitel Reguläre Sprachen

2.1 Endliche Automaten

Definition 2.1.1

Satz 2.1.3

Satz 2.1.4

Satz 2.1.5

Satz 2.1.6

2.2 NFA

Definition 2.2.1

Definition 2.2.2

Definition 2.2.3

Satz 2.2.4

Satz 2.2.5

2.3 Reguläre Ausdrücke

Definition 2.3.1

Definition 2.3.2

Lemma 2.3.3

Satz 2.3.4 (Satz von Kleene)

Folgerung 2.3.5

2.4 Das Pumping Lemma

Satz 2.4.1 Pumping Lemma

2.5 Minimalautomaten und Äquivalenzrelationen

Definition 2.5.1

Satz 2.5.2 (Myhill Nerode)

2.6 Abschlusseigenschaften

Definition 2.6.1

3. Kapitel Kontextfreie Sprachen

Satz 3.0.1

3.1 Die Chomsky-Normalform

Definition 3.1.1

Satz 3.1.2

Definition 3.1.2

Satz 3.1.3

3.2 Pumping Lemma für Kontextfreie Sprachen

Satz 3.2.1

Lemma 3.2.2

3.3 Abschlusseigenschaften

Satz 3.3.1

5. Kapitel Berechenbarkeit

5.1 Intuitiver Begriff der Berechnbarkeit

5.2 Grundlagen

Satz 5.2.1

5.3 Turing Berechenbarkeit

Definition 5.3.1

Definition 5.3.2

5.4 Andere Typen von Turingmaschinen

Satz 5.4.1

5.5 Die Churchsche These

These 5.5.1 Churchs These

5.6 Deterministische und nicht deterministische TM

Satz 5.6.1

6. Kapitel Entscheidbarkeit und Aufzählbarkeit

6.1 Entscheidbarkeit

Definition 6.1.1

Satz 6.1.2

Satz 6.1.3 REC ist abgeschlossen bezüglich

6.2 Semi-Entscheidbarkeit und Aufzählbarkeit

[Definition 6.2.1](#)

[Definition 6.2.2](#)

[Satz 6.2.3](#)

[Folgerung 6.2.4](#)

[Satz 6.2.5](#)

[Satz 6.2.6](#)

[Satz 6.2.7](#)

[6.3 Beziehung zwischen REC und RE](#)

[Satz 6.3.1](#)

[Satz 6.3.2](#)

[6.4 Kodierung von Turingmaschinen über \$\{0,1\}\$](#)

[6.5 Das Halteproblem](#)

[6.5.1 Problemstellung und intuitive Argumentation](#)

[6.5.2 Formale Argumentation mittels TM](#)

[Definition 6.5.2.1](#)

[Satz 6.5.2.2](#)

[Satz 6.5.2.3](#)

[Satz 6.5.2.4](#)

[Satz 6.5.2.5](#)

[Automaten und Berechenbarkeit > 7. Kapitel - NP Vollständigkeit](#)

[7.1 Die Klasse P](#)

[Definition 7.1.1](#)

[Definition 7.1.2](#)

[7.2 Die Klasse NP \(nicht-deterministisch-polynomial\)](#)

[Definition 7.2.1](#)

[Definition 7.2.2](#)

[7.3 NP-Vollständigkeit](#)

[Definition 7.3.1](#)

[Definition 7.3.2](#)

[Satz 7.3.3](#)

[Satz 7.3.4](#)

[Satz 7.3.5](#)

[Satz 7.3.6](#)

[Satz 7.3.7](#)

[Satz 7.3.8](#)

1. Kapitel Sprachen und Grammatiken

1.1 Grammatik

1.1.1 Alphabet und Wörter

Definition 1.1.1.1

Ein Alphabet Σ ist eine endliche nicht leere Menge.

Die Elemente von Σ heißen Buchstaben.

z.B.:

$$\Sigma_{\text{latein}} := \{a, b, c, \dots, x, y, z\}$$

$$\Sigma_{\text{griech}} := \{\alpha, \beta, \gamma, \dots\}$$

$$\Sigma_{\text{binär}} := \{0, 1\}$$

$$\Sigma_{\text{Ziff}} := \{0, 1, 2, \dots, 9\}$$

$$\Sigma_{\text{Tastatur}} := \{a, b, c, \dots, z\}$$

$$\cup \{A, B, \dots, Z\}$$

$$\cup \{\ddot{a}, \dots, \ddot{A}, \dots\}$$

$$\cup \{0, \dots, 9\} \cup \{?, !, ., \dots\} \cup \{\square\}$$

Definition 1.1.1.2

Es sei $\Sigma := \{a_1, \dots, a_m\}$ ein Alphabet. Ein Wort w ist eine endliche Folge von Buchstaben aus Σ ,

etwa $w = a_{i1}, a_{i2}, \dots, a_{in}$ $a_{ij} \in \Sigma$. Die Länge $|w|$ eines Wortes ist die Anzahl seiner Buchstaben.

λ sei das eindeutig bestimmte Wort der Länge 0.

Definition 1.1.1.3

Es sei Σ ein Alphabet

$$(\Sigma)^0 := \{\lambda\}$$

$$(\Sigma)^{i+1} =_{df} (\Sigma)^i \cdot \Sigma =_{df} \{wa | w \in (\Sigma)^i, a \in \Sigma\}$$

Nebenrechnung:

$$(\Sigma)^0 \cdot \Sigma = \{\lambda\} \cdot \Sigma = \{wa | w \in \{\lambda\} \ a \in \Sigma\}$$

$$(\Sigma)^2 = \Sigma \cdot \Sigma = \{wa | w \in \Sigma \ a \in \Sigma\}$$

$$(\Sigma)^* =_{df} \bigcup_{i=0}^{\infty} (\Sigma)^i$$

$(\Sigma)^*$... Menge aller endlichen Folgen über Σ , d.h. Menge aller Wörter über Σ

$(\Sigma)^i$... Menge aller Wörter der Länge i über Σ

Bsp:

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

$(\Sigma)^*$... Menge aller endlichen Zeichenketten aus a's und b's und λ

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

Die Adjazenzmatrix eines Graphen ist als Wort über $\{0, 1, \#\}$ darstellbar.

- Wörter der Umgangssprache sind auch formelle Wörter über Σ_{latein} aber auch Sätze, Texte und Romane sind formale Wörter über Σ_{tastatur} .
Alle möglichen Daten (z.B. Eingabe von Algorithmen) und Informationen können als formale Wörter dargestellt werden.

Operationen auf Wörter

Für zwei Wörter $u = x_1x_2 \dots x_n$ und $v = y_1y_2 \dots y_k$ mit $x_i, y_i \in \Sigma$ ist

$u \circ v = x_1x_2 \dots x_ny_1y_2 \dots y_k$ die Konkatinetion von u und v Schreibweise: $w = uv$ oder auch $w = u \circ v$ (wie oben gezeigt).

Für ein Wort $w = x_1x_2 \dots x_n$, $x_i \in \Sigma$ ist das Spiegelwort wie folgt definiert:

$$Sp(w) = x_nx_{n-1} \dots x_2x_1 =_{df} w^R$$

Potenz von Wörtern

Es sei $w = x_1 \dots x_n$, $x_i \in \Sigma$ ein Wort der Länge n

$$w^0 = \lambda, (w^1 = w^0 \circ w = \lambda \circ w = w)$$

$$w^{k+1} = w^k \circ w; w^2 = w \circ w$$

Bsp: $w = abb$

$$\Rightarrow w^3 = abbabbabb = (abb)^3$$

Relationen für Wörter (über $(\Sigma)^*$)

$T \subseteq (\Sigma)^* \setminus (\Sigma)^* \dots$ Teilwortrelation

$$(u, v) \subseteq T \leftrightarrow \bigvee_{l, v \in (\Sigma)^*} v = luv$$

"u ist Teilwort von v"

$(a, b)P \subseteq (\Sigma)^* \times (\Sigma)^*$ Präfixrelation

"a ist Präfix von b"

Bsp.: (PAPA, PAPAJABAUM) $\in P$

$(a, b)S \subseteq (\Sigma)^* \times (\Sigma)^*$ Suffixrelation

"a ist Suffix von b"

Bsp.: (BUMM, KABUMM)

Behauptung: T und P sind reflexiv, transitiv, asymmetrisch

1.1.2 Formale Sprache

Definition 1.1.2.1

Jede $A \subseteq (\Sigma)^*$ heißt Formalsprache über Σ

Bsp: $(\Sigma)^* := \{a, b\}^*$

$$A_1 := \{a, abb, ba\}$$

$$A_2 := \{w \in \{a, b\}^* \mid w \text{ beginnt mit } a\}$$

$$\Sigma := \{0, 1\} \quad L_{\text{binär}} := \{ \bigvee_{u \in (\Sigma)^*} w = 1u \} \cup \{0\}$$

$$\Sigma = \{a, b, c, \wedge, \vee, \neg, \leftrightarrow, \rightarrow\}$$

$A \dots$ Menge aller aussagenlogischer Ausdrücke in den Atomen a, b, c

- Menge von Wörtern bestimmter Eigenschaften bilden Formale Sprachen

Operationen auf Sprachen

Formale Sprachen sind Mengen

\Rightarrow Alle Mengenoperationen sind für Sprachen definiert

$$L_1 \subseteq (\Sigma)_1^* ; L_2 \subseteq (\Sigma)_2^*$$

und somit $L_1, L_2 \in (\Sigma)^*$ mit $\Sigma = \Sigma_1 \Sigma_2$

Dann sind auch:

$$L_1 \cup L_2$$

$$L_1 \cap L_2$$

$$L_1 \setminus L_2$$

$$L_1 = \sum_1 \setminus L_2 = \{w \in \sum_1^* \mid w \in L_1\}$$

$$L_1 \times L_2$$

formale Sprachen

Definition 1.1.2.2

Es seien $L_1 \subseteq \Sigma_1^*, L_2 \subseteq \Sigma_2^*$

Dann ist $L_1 \circ L_2 = \{xy \mid x_1 \in L_1, y \in L_2\}$

die Verkettung (Konkatenation) der Sprachen L_1 und L_2

Definition 1.1.2.3

Es sei $L \subseteq \Sigma_1^*$

Dann ist $L^0 =_{def} \{\lambda\}$

$$L^{m+1} = L^m \circ L = \{uv \in \Sigma^* \mid u \in L^m, v \in L\}$$

und

$$L^* = \bigcup_{m \in \mathbb{N}} L^m \text{ die kleine Hülle von } L$$

Bemerkung:

$$L^1 = L^0 \circ L = \lambda L = L = \{uv \in L \mid u \in \{\lambda\}, v \in L\}$$

Bsp.:

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

$$A := \{a, ba\}$$

$$B := \{aa, ab, b\}$$

$$A \circ B = \{aaa, aa, ab, baaa, baab, bab\}$$

$$\Sigma = \{a\} \quad A = \{aa\}$$

$A^* = \{aa\}^*$ Menge aller Wörter über A die gerade Länge haben.

$$\Sigma = \{a, b, f, m, p, u\}$$

$$A = \{ba, umpf\}$$

$$A^2 = \{baba, baumpf, umpfba, umpfumpf\}$$

$a\{a, b\}^* \cup \{a, b\}^*b \dots$ Menge aller Wörter die mit "a" beginnen und mit "b" aufhören.

1.2 Einführung

Natürliche Sprachen durch Grammatikregeln, die konkrete Sätze erzeugen.

Ein Versuch:

SATZ \rightarrow SUBJEKT - PRÄDIKAT
SUBJEKT \rightarrow ATTRIBUT_SUBJEKT
PRÄDIKAT \rightarrow PRÄDIKAT_OBJEKT
OBJEKT \rightarrow ATTRIBUT_OBJEKT
PRÄDIKAT \rightarrow VERB
OBJEKT \rightarrow SUBJEKT

SUBJEKT \rightarrow Affen
SUBJEKT \rightarrow Menschen
SUBJEKT \rightarrow Autos
VERB \rightarrow frühstücken
VERB \rightarrow schlafen
ATTRIBUT \rightarrow klein
ATTRIBUT \rightarrow blau

kleine Autos frühstücken kleine Affen

Regeln und deren Hilfe genau die Wörter der Sprache gehören und alle anderen nicht

\rightarrow formale Grammatik

Die Erzeugung einer Sprache durch Ableitungsregeln bedarf folgende Dinge:

1. Symbole der Sprache(Buchstaben)
2. Hilfssymbole
3. Startsymbole
4. Ableitungsregeln

Definition 1.2.1

ein 4-Tupel $G = (\Sigma, N, S, R)$ heißt Grammatik \leftrightarrow_{df}

1. Σ ist eine endliche Menge der Buchstaben oder terminaler Symbole
2. N ist eine endliche Menge an Hilfssymbole oder nicht terminale Symbole
3. $S \in N$ Startsymbole
4. $R \subseteq (E \cup N)^* N (\Sigma \cup N)^* \times (\Sigma \cup N)^* \dots$ Regelmenge

$(p, q) \in R$ heißen Regeln, auch $p \rightarrow q$

Bsp:

$$G = \{\{a, b\}, \{s, x, y, z\}, S, R_1\}$$

$$R_1 = \{S \rightarrow SX, S \rightarrow bY, Y \rightarrow aZ, Z \rightarrow b, Z \rightarrow b\}$$

Am Beispiel von oben(mit den Autos die Affen frühstücken):

$$N = \{SATZ, SUBJEKT, PRÄDIKAT\}$$

$$S = SATZ$$

$$R = \{SATZ \rightarrow SUBJEKT PRÄDIKAT \dots\}$$

Definition 1.2.2

Es sei $G = (\Sigma, N, S, R)$ eine Grammatik.

w' heißt unmittelbare Ableitung von w bezüglich G

$$w \stackrel{\text{Ableitbar}}{\vdash}_G w' \leftrightarrow$$

$$1. w, w' \in (\Sigma \cup N)^*$$

$$2. \bigvee_{p_1 p_2 p q} \{p_1, p_2, p, q \in (\Sigma \cup N)^*, (p, q) \in R\} | w = p_1 p p_2, w = p_1 q p_2\}$$

w' heißt die Ableitung bezüglich w

$$w \vdash_G^* w' \leftrightarrow_{df}$$

$$1. w = w' \text{ oder}$$

$$2. \bigvee_{n \in \mathbb{N}} \bigvee_{w_0 w_1 \dots w_n} V \text{ wobei gelten muss:}$$

$$w_0 = w$$

$$w_n = w'$$

$$w_0 \vdash_G w_1$$

$$w_1 \vdash_G w_2 \dots$$

$$w_{n+1} \vdash_G w_n$$

Schreibweise

$w \vdash_G w'$ Ableitung der Länge n

\vdash_G, \vdash_G^* sind binäre Relationen

$$\vdash_G \subseteq (\Sigma \cup N)^* \times (\Sigma \cup N)^*$$

\vdash_G^* ist die reflexive und transitive Hülle von \vdash_G

Bsp:

G_1

$$S \vdash SX \vdash SXX \vdash bYXX$$

$$S \vdash bY \vdash baZ \vdash babY \vdash babaZ \vdash babab$$

Wir wollen Sprachen erzeugen, d.h. am Ende sollen keine Hilfssymbole mehr da sein.

Definition 1.2.3

Es sei $G = \{\Sigma, N, S, R\}$

eine Grammatik $L = \{w \in \Sigma^* \mid S \vdash_G^* w\}$

heißt die von G erzeugte Sprache

Schreibweise: $\mathcal{L}(G)$

Bsp: G_1

X , werden wir nicht mehr los $\rightarrow S \rightarrow BY, Y \rightarrow aZ, Z \rightarrow bY, Z \rightarrow b$

\rightarrow es entstehen Wörter der Form: $b(ab)^n, n \geq 1$

alle Wörter dieser Form sind ableitbar, alle anderen nicht $\rightarrow \mathcal{L}(G_1) = \{b(ab)^n \mid n \geq 1\}$

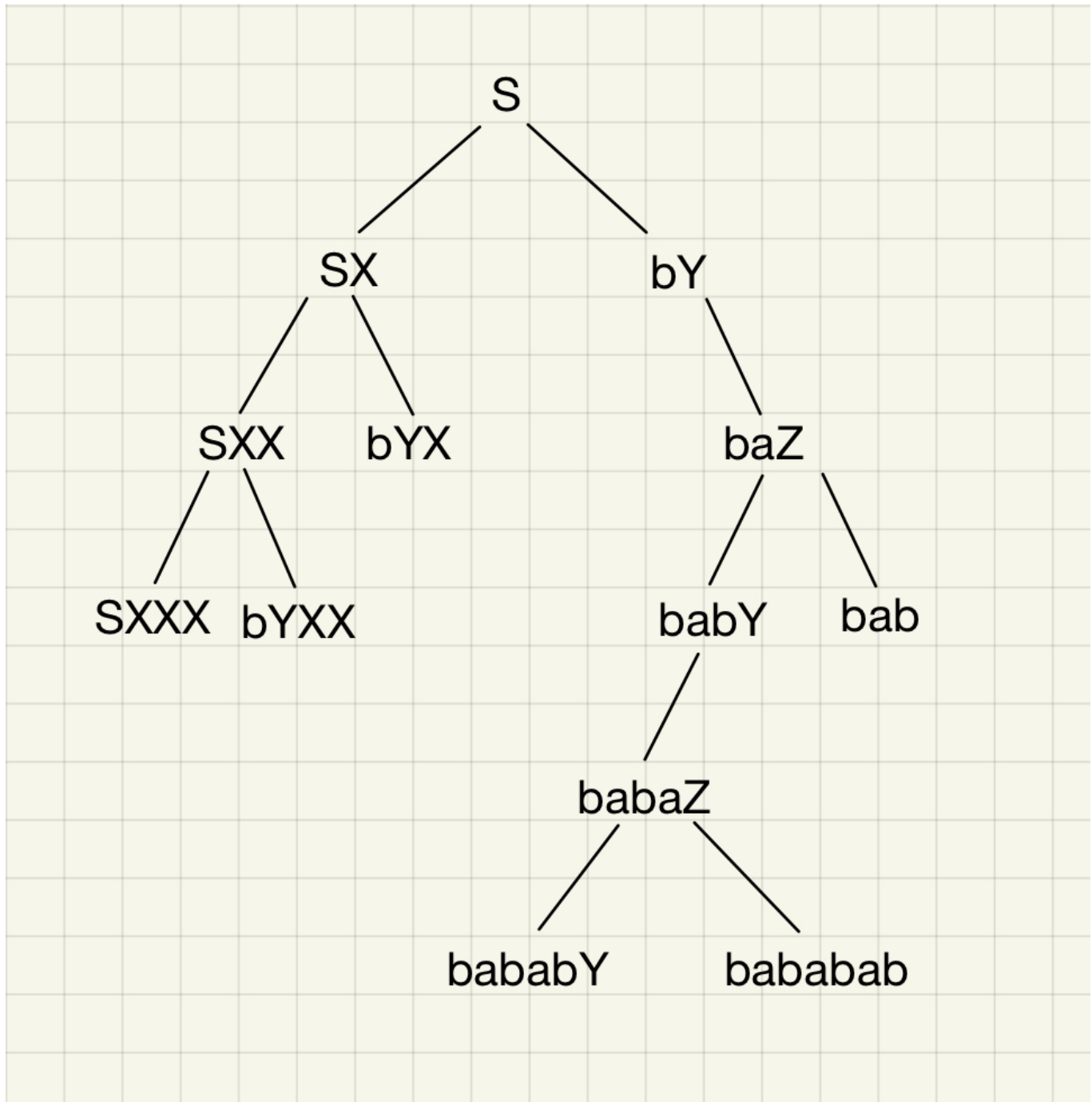
$$B = \{w \in \{a, b\}^* \mid w \text{ endet auf } a\}$$

$$G = (\{a, b\}, \{S\}, S, R)$$

$$R = \{S \rightarrow a \mid aS \mid bS\}$$

Ableitungsbaum

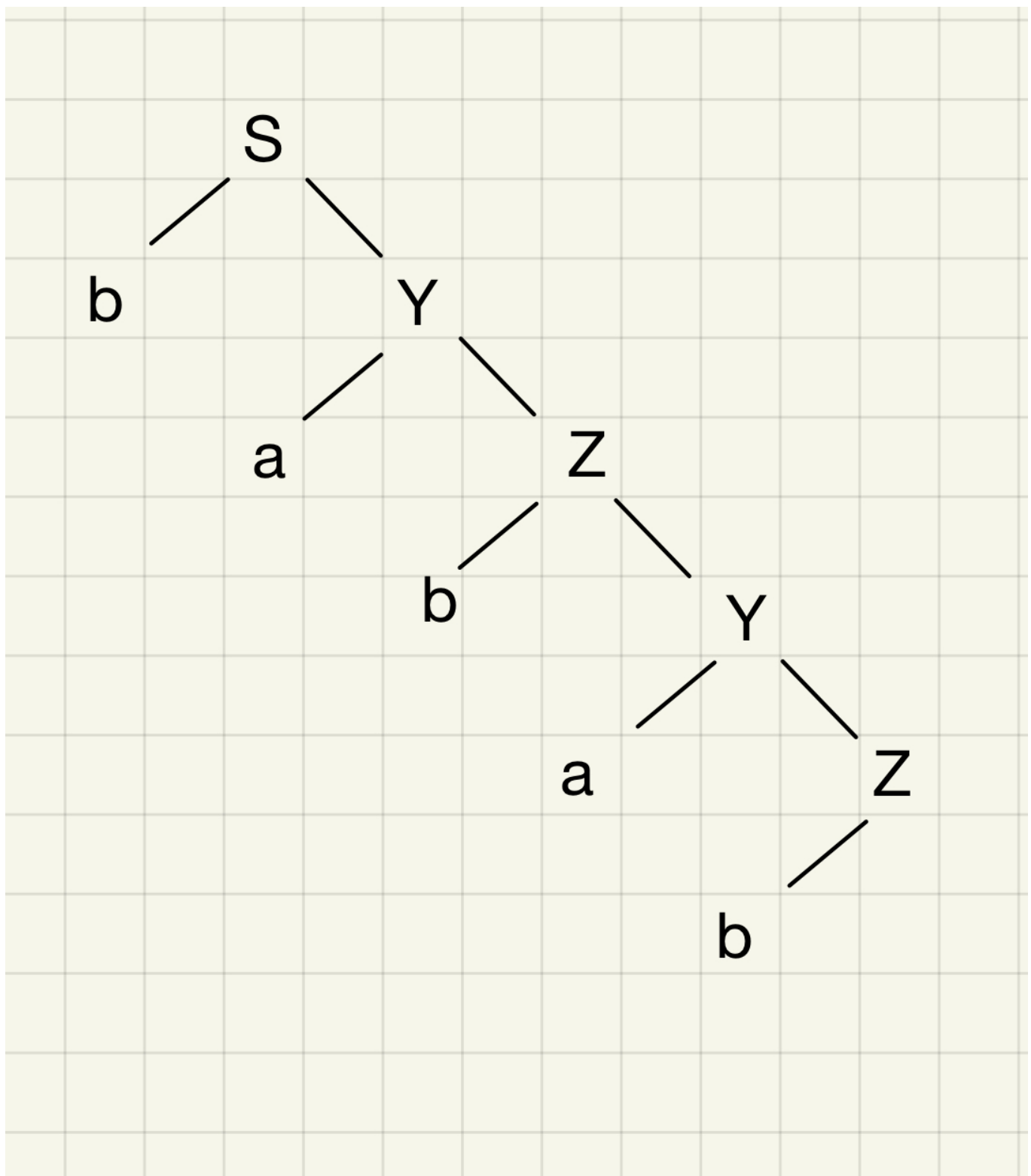
$G = (\Sigma, M, S, R)$ am Bsp G_1 Startsymbol ist die Wurzel. Zeichenketten aus $(\Sigma \cup N)^*$ sind Knoten Kinder eines Knoten w sind genau die Zeichenketten w' mit $w \vdash_G w'$



- Wörter aus $\mathcal{L}(G_1)$ Sind Blätter

Syntaxbaum für ein Wort nur möglich, wenn für alle Regeln $p \rightarrow q$ $p \in N$ gilt.

Syntaxbaum am Beispiel "babab":



Blätter von links nach rechts ergeben w .

Zu jeder Grammatik gibt es eine eindeutig bestimmte Sprache. Die Umkehrung ist falsch.

Bsp $G'_1 = (\{a, b\}, \{s, y, z\}, S, R)$

$$P_1 <' = \{S \rightarrow bY, Y \rightarrow aZ, z \rightarrow bY | b\}$$

\rightarrow erzeugt ebenfalls $\mathcal{L}(G_1)$

Definition 1.2.4

Zwei Grammatiken heißen äquivalent $G_1 \leftrightarrow_{df} G_2$ $\mathcal{L}(G_1) = \mathcal{L}(G_2)$

1.3 Die Chomsky-Hierarchie

Definition 1.3.1

Sei $G = (\Sigma, M, S, R)$ eine Grammatik

1. G heißt Typ 0 Grammatik
2. G heißt Kontextsensitiv (nicht verkürzend) (Typ 1) wenn für alle Regeln $(u, v) \in R$ gilt $|u| \leq |v|$
3. G heißt Kontextfrei (Typ 2), wenn G Kontextsensitiv ist und für alle Regeln $(u, v) \in R$ gilt $u \in N$
4. G heißt Regulär (rechtslinear) (Typ 3), wenn G Kontextfrei ist und falls für alle $(u, v) \in R$ $v \in \Sigma$ oder $v \in \Sigma \circ N$

Bsp:

1. $aA \rightarrow aa$
2. $BB \rightarrow b$, Verkürzung erlaubt
3. $A \rightarrow aA$
 $A \rightarrow aBa \mid aX$
4. nur Regeln $A \rightarrow b$
 $A \rightarrow bB$

die Terminale **stehen** nur links

Definition 1.3.2

Eine Sprache $L \subset \Sigma^*$ heißt vom Typ 0 (1,2,3), falls es eine Typ 0 (1,2,3)-Grammatik gibt, mit $\mathcal{L}(G) = L$

Eine Sprache heißt erzeugbar von einer Grammatik, falls es eine Grammatik gibt, die die Sprache erzeugt.

Sprachfamilien

\mathcal{L} ... Klasse der von Grammatik erzeugbaren Sprachen

\mathcal{L}_1 ... CS (Context-sensitive) Klasse der Kontextfreien Sprachen

\mathcal{L}_2 ... CF (Context-free) Klasse der Kontextfreien Sprachen

\mathcal{L}_3 ... REG (Regual) Klasse der Regulären Sprachen

Satz 1.3.3

$$REG \subseteq CF \subseteq CS \subseteq \mathcal{L}_0$$

Beobachtung: Folgt unmittelbar aus Definition 1.3.1 vom $Typ_{i+1}, 0 \leq i \leq 2$

2. Kapitel Reguläre Sprachen

2.1 Endliche Automaten

- Grammatiken erzeugen Wörter
- Automaten akzeptieren Wörter
 - entscheiden, ob ein Eingabewort zur Sprache gehören

Definition 2.1.1

Ein deterministischer endlicher Automat M ist ein 5-Tupel

$\rightarrow M = (\Sigma, Z, \delta, Z_0, Z_E)$ mit folgenden Eigenschaften

1. Σ ... ist eine endliche Menge Eingabealphabet
2. Z ... ist eine endliche Menge Zustandsmenge
3. $\delta : Z \times \Sigma \rightarrow Z$... ist eine endliche Menge Überföhrungsfunktion
4. $z_0 \in Z$... Startzustand
5. $z_E \in Z$... Endzustand

Satz 2.1.3

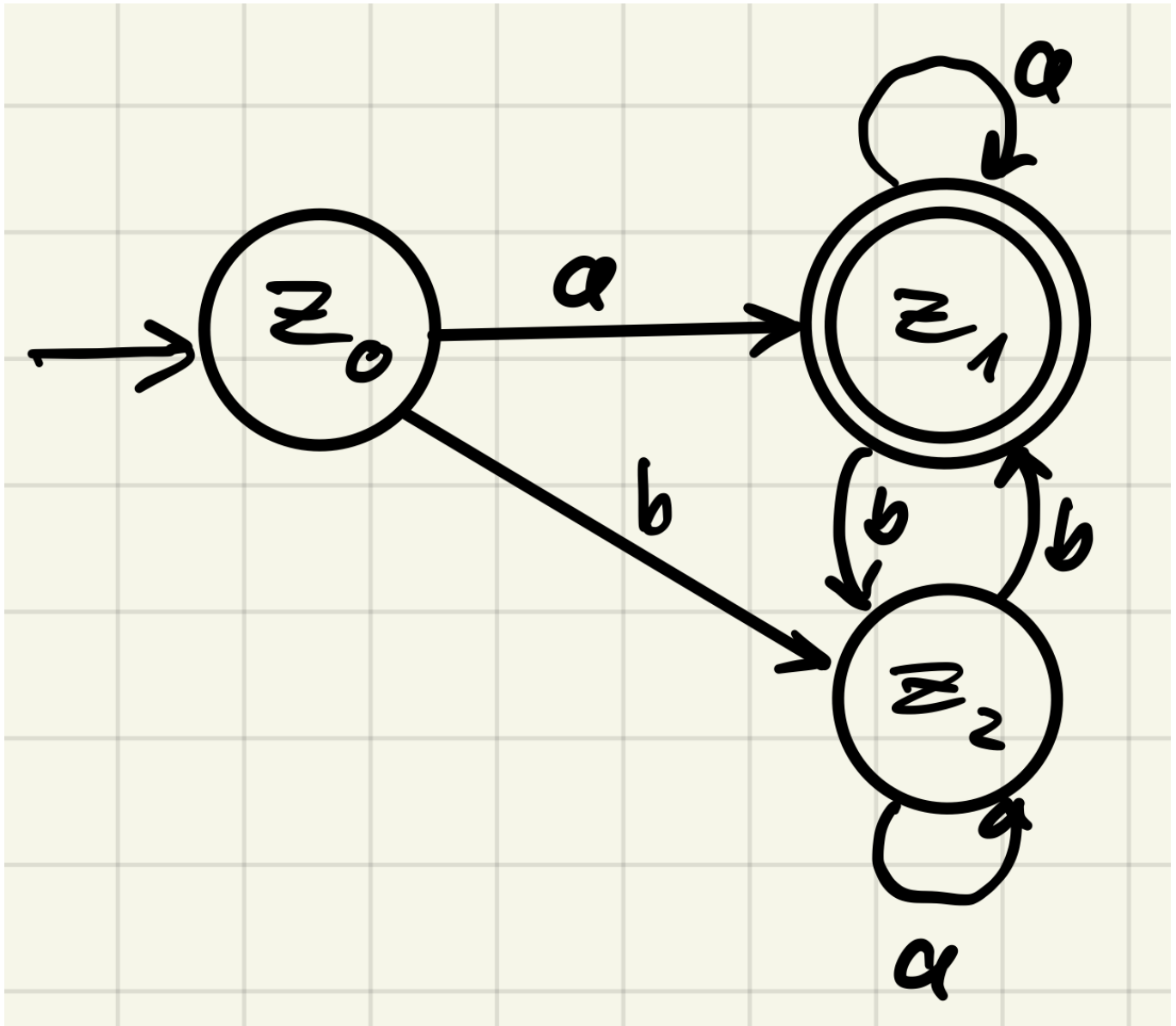
Jede Sprache die von einem DFA (DEA?) akzeptiert werden kann ist regulär.

Bsp: $A \subseteq \Sigma^*$ Sprache

$M = (\Sigma, Z, \delta, Z_0, Z_E)$ DEA mit $L(M) = A$ Wir geben eine reguläre Grammatik gerade $\mathcal{L}(G) = 1$ gilt.

1. Fall $\lambda \notin A$ (brauchen also keine Regel $S \rightarrow \lambda$)

Idee:



$$Z_0 \rightarrow aZ_1 | bZ_2 | a$$

$$Z_1 \rightarrow aZ_1 | bZ_2 | a$$

$$Z_2 \rightarrow aZ_2 | bZ_1 | b$$

$$\delta(z_0, a) = Z_1$$

$$Z_0 \rightarrow aZ_1$$

$$Z_0 \rightarrow bZ_2$$

$$Z_0 \rightarrow a$$

formel sei $G = (\Sigma, N, S, R)$

Wir setzen $N = Z$, $S = Z_0$

$$R = \{(Z \rightarrow aZ | \delta(Z, a) = Z')\}$$

$$\cup \{(Z \rightarrow a) | \delta(Z, a) = Z', Z' \in Z_E\}$$

$$w \in A \leftrightarrow w \in L(M)$$

\leftrightarrow es gibt eine Folge von Zuständen von M Z_0, \dots, Z_n mit Z_0 als Startzustand,

$Z_n \in Z_E$ und für alle $0 \leq i \leq n-1$ gilt $\delta(Z_i, a_{i+1}) = Z_{i+1}$

\leftrightarrow es gibt eine Folge von nicht terminalen Z_0, Z_1, \dots, Z_n mit Z_0 als Startsymbol und

$$Z_0 \vdash_G a_1 Z_1 \vdash a_1 a_2 z_2 \vdash_G \dots \vdash a_1 a_2 \dots a_{n+1} Z_{n+1} \vdash a_1 \dots a_n$$

$$\Leftrightarrow Z_0 \vdash^* w \Leftrightarrow w \in \mathcal{L}(G) \rightarrow L(M) = \mathcal{L}$$

2. Fall

folgt aus Fall 1 unter Berücksichtigung von Satz 1.3.4

$$S \rightarrow 0S | 1S | 1X$$

$$X \rightarrow 1$$

Definition 2.1.4

Es sei $M = (\Sigma, Z, \delta, Z_0, Z_E)$ ein DFA (DEA?) Dann mit

$$R_M = \{(x, y) \in \Sigma^* \times \Sigma^* | \delta^*(Z_0, x) = \delta^*(Z_0, y)\}$$

$(x, y) \in R_M \Leftrightarrow$ Bei Abarbeitung im DFA (DEA?) enden x und y im gleichen Zustand.

Satz 2.1.4

Es sei $M = (\Sigma, Z, \delta, Z_0, Z_E)$ ein DFA, dann ist

$$R_M = \{(x, y) \in \Sigma^* \times \Sigma^* | \delta^*(Z_0, x) = \delta^*(Z_0, y)\} \quad (x, y) \in R_M \Leftrightarrow \text{Bei Abarbeitung im}$$

DFA enden x und y im gleichen Zustand

Satz 2.1.5

Für einen DFA (DEA?) $M = \{\Sigma, Z, \delta, Z_0, Z_E\}$ ist R_M eine Äquivalenzrelation

Beobachtung:

Reflexivität

$$\bigwedge_{x \in \Sigma^*} (x, x) \in R_M, \text{ dann } \bigwedge_{x \in \Sigma^*} \delta^*(Z_0, x) = \delta^*(x, Z_0)$$

Symmetrie

Es seien $x, y \in \Sigma^*$ $(x, y) \in R_M \rightarrow \delta^*(Z_0, x) = \delta^*(Z_0, y) \Rightarrow (y, x) \in R_M$

Transitivität

Es seien $(x, y) \in R_M$ und $(y, x) \in R_M$, d.h. $\delta^*(Z_0, x) = \delta^*(Z_0, y)$ und $\delta^*(Z_0, y) = \delta^*(Z_0, u) \Rightarrow \delta^*(Z_0, x) = \delta^*(Z_0, u) \Rightarrow (x, u) \in R_M$

Satz 2.1.6

Sei $M = (\Sigma, Z, \delta, Z_O, Z_E)$ ein DFA (DEA??)

Dann gilt:

$$(x, y) \in R_M \Leftrightarrow \bigwedge_{u \in \Sigma^*} (xu, yu) \in R_M$$

R_M ist rechtsinvariant gegen Verkettung

Beobachtung : " \Rightarrow "

$$(x, y) \in R_M \rightarrow \delta^*(Z_O, x) = \delta^*(Z_O, y)$$

$$\Rightarrow \bigwedge_{u \in \Sigma^*} \delta^*(Z_O, xu) = \delta^*(Z_O, yu)$$

$$\Rightarrow \bigwedge_{u \in \Sigma^*} (xu, yu) \in R_M$$

$$A \rightarrow B$$

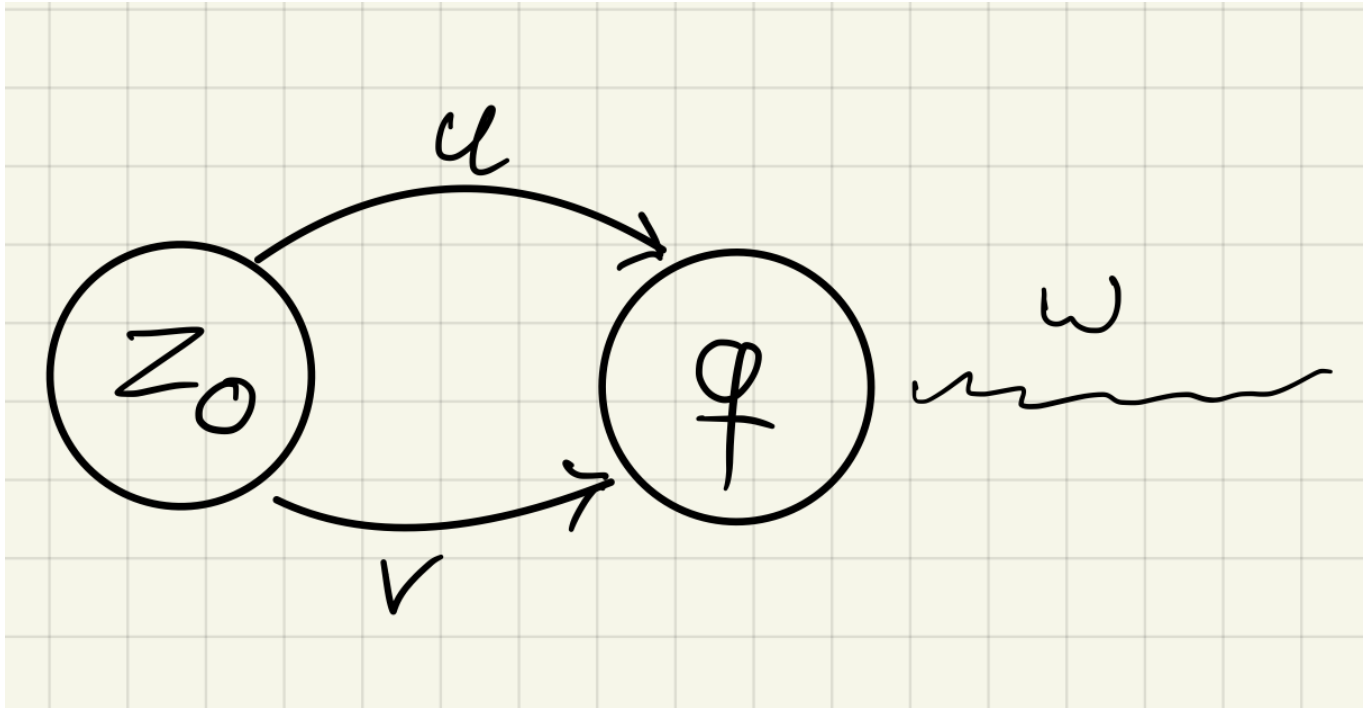
$$\Leftarrow \text{"Kontraposition"} \neg A \rightarrow \neg B$$

$$\text{z.Z. } (x, y) \notin R_M \rightarrow \bigvee_{u \in \Sigma^*} (xu, zu) \notin R_M$$

Das gilt für $u = \lambda$

Konstruktion deterministischer endlicher Automaten

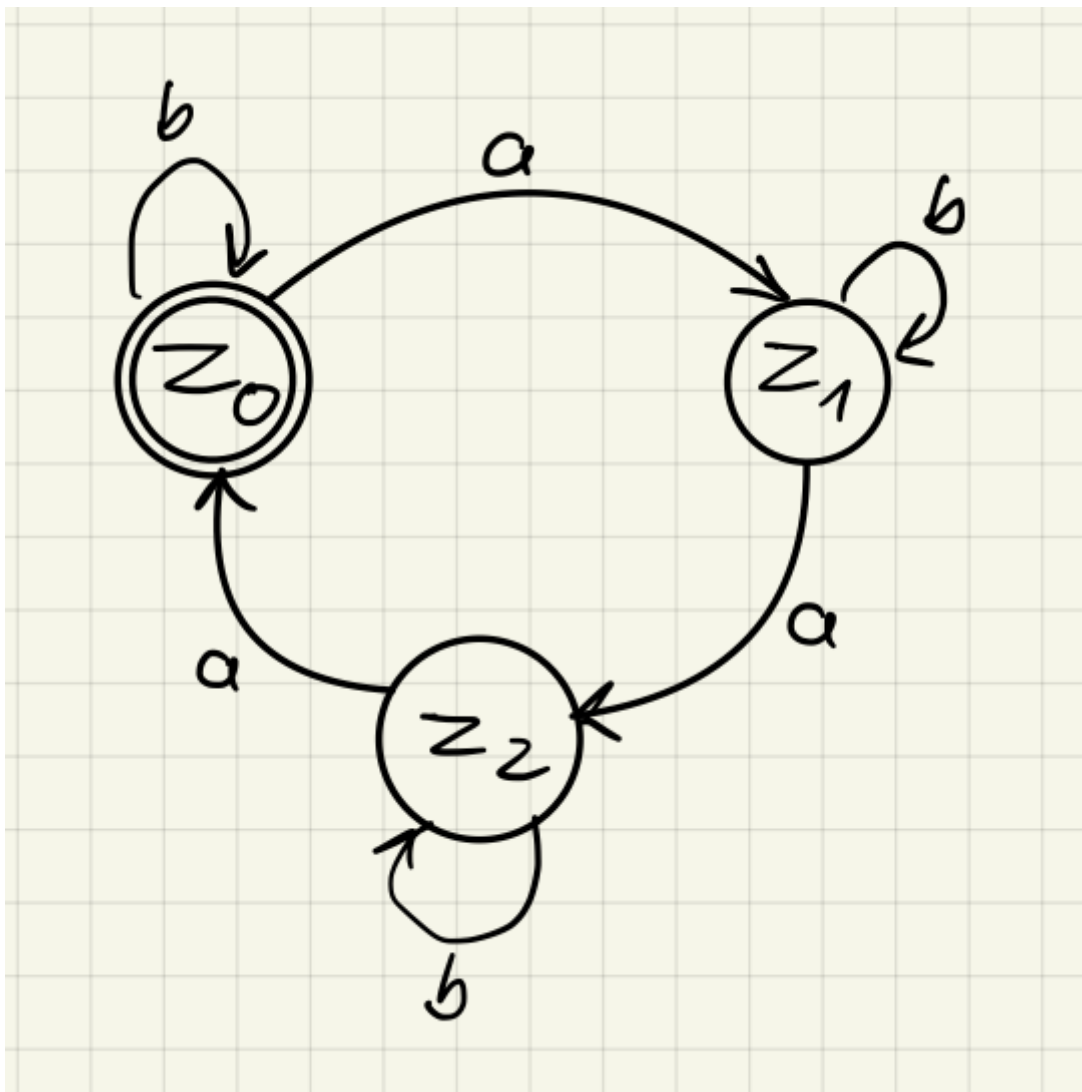
Seien $u, v \in \Sigma^*$ mit $(u, v) \in R_M$, d.h. $\delta^*(Z_O, z) = \delta^*(Z_O, v) = q$



Bemerkung: Für jedes beliebige Wort $w \in \Sigma^*$ kann M nicht zwischen uw und vw unterscheiden. Die einzige Möglichkeit sich den Zustand eines bereits gelesenen Wortes zu merken ist der Zustand des DFA (DEA??). Indem dieser Wortteil endet ($\delta^*(Z_O, u)$). Die für die Zugehörigkeit bzw. "nicht"-Zugehörigkeit eines Wortes u zu L relevanten Informationen müssen in q gespeichert werden.

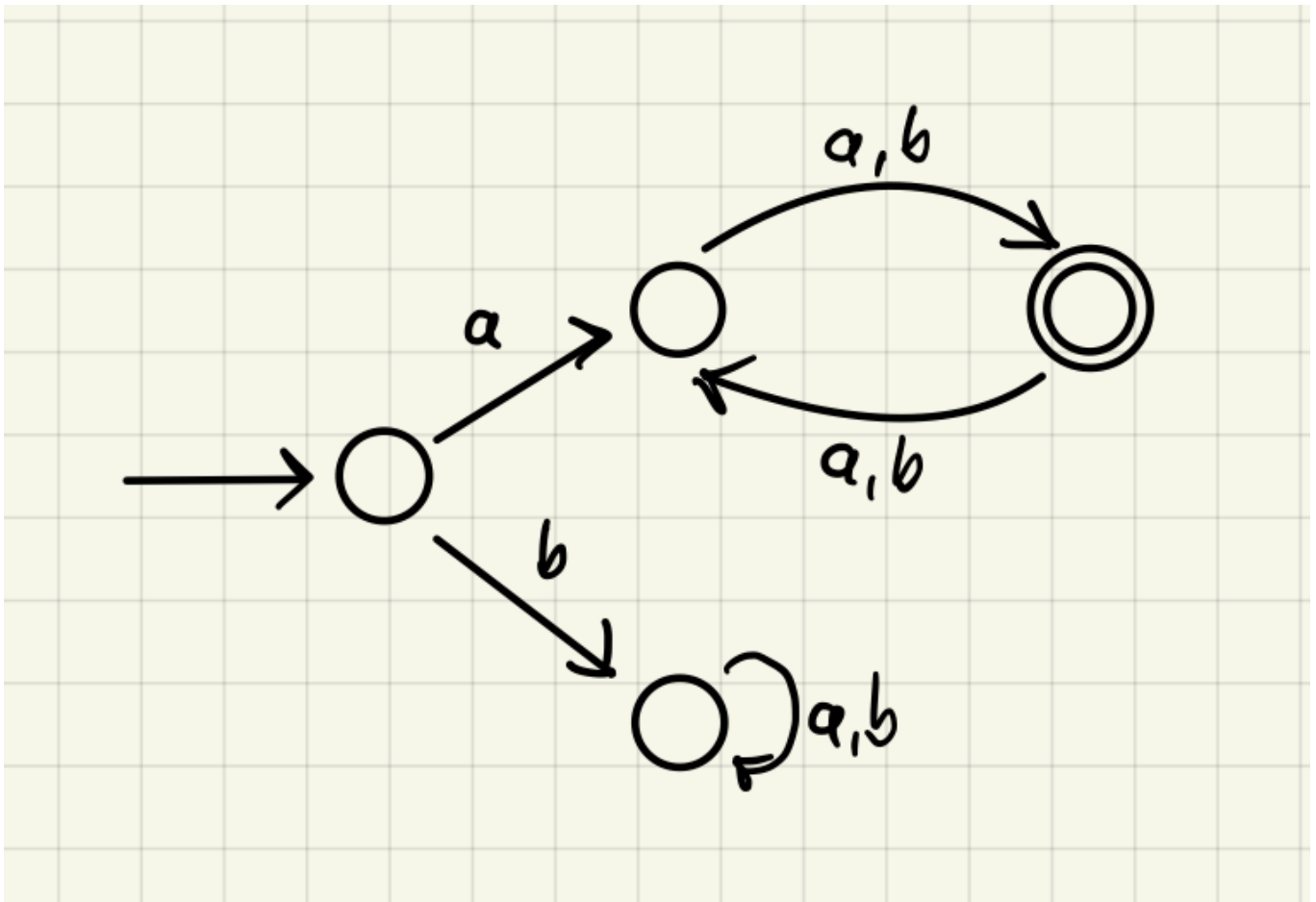
aababbbba

$$L_1 = \{w \in \{a, b\}^* \mid \#(w) \equiv_3 0\}$$



$$M = (\{a, b\}, \{z_1, z_2, z_3\}, \delta, z_0,)$$

$$L_2 = \{w \in \{a, b\}^* | w \text{ beginnt mit } a \text{ und } |w| \equiv_2 0\}$$



$$L_3 = \{w \in \{a, b\}^* \mid w \text{ beginnt mit } a \Leftrightarrow |w| \equiv_2 0\}$$

Beides gilt oder Beides gilt nicht !

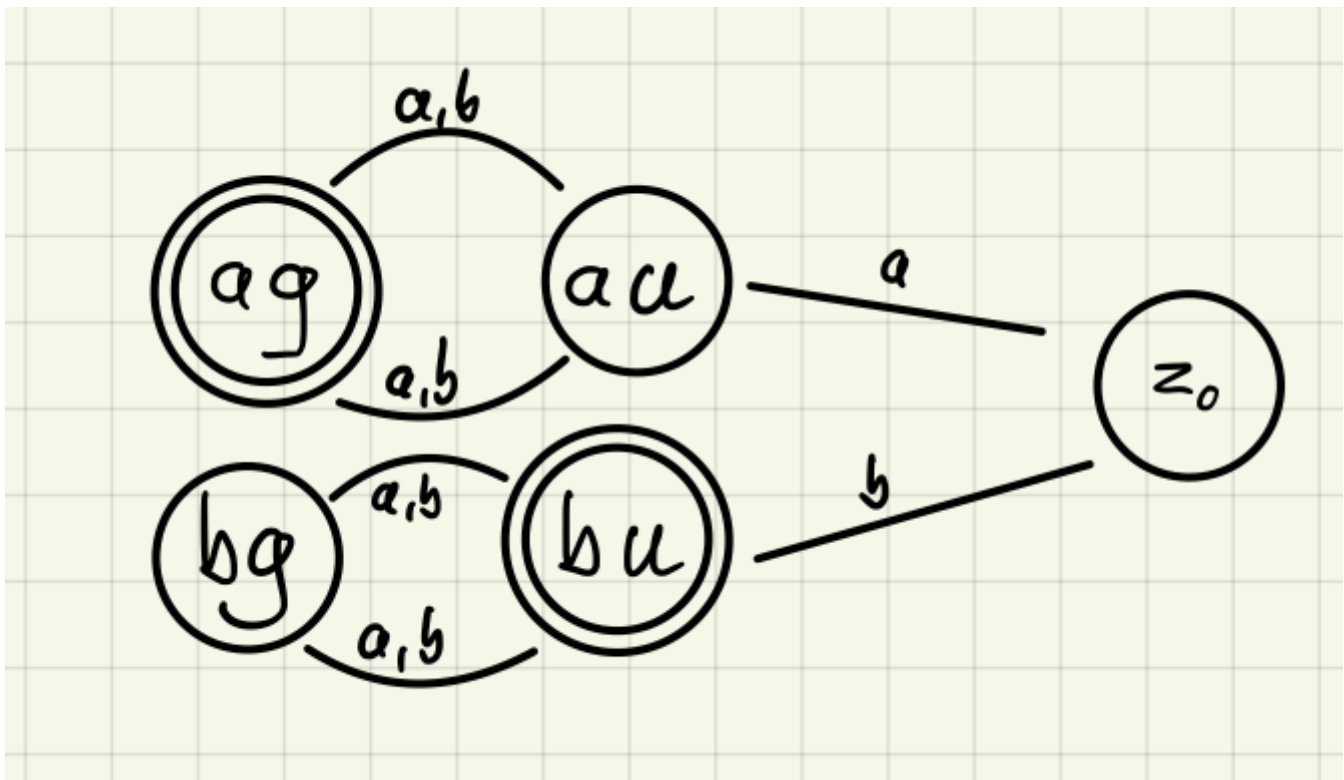
Bsp: $aa \in L_3$, beginnt mit a und gerade länge $\Rightarrow 1 \Leftrightarrow 1 = 1$

$\lambda \notin L_3$, beginnt nicht mit a und gerade länge $\Rightarrow 0 \Leftrightarrow 1 = 0$

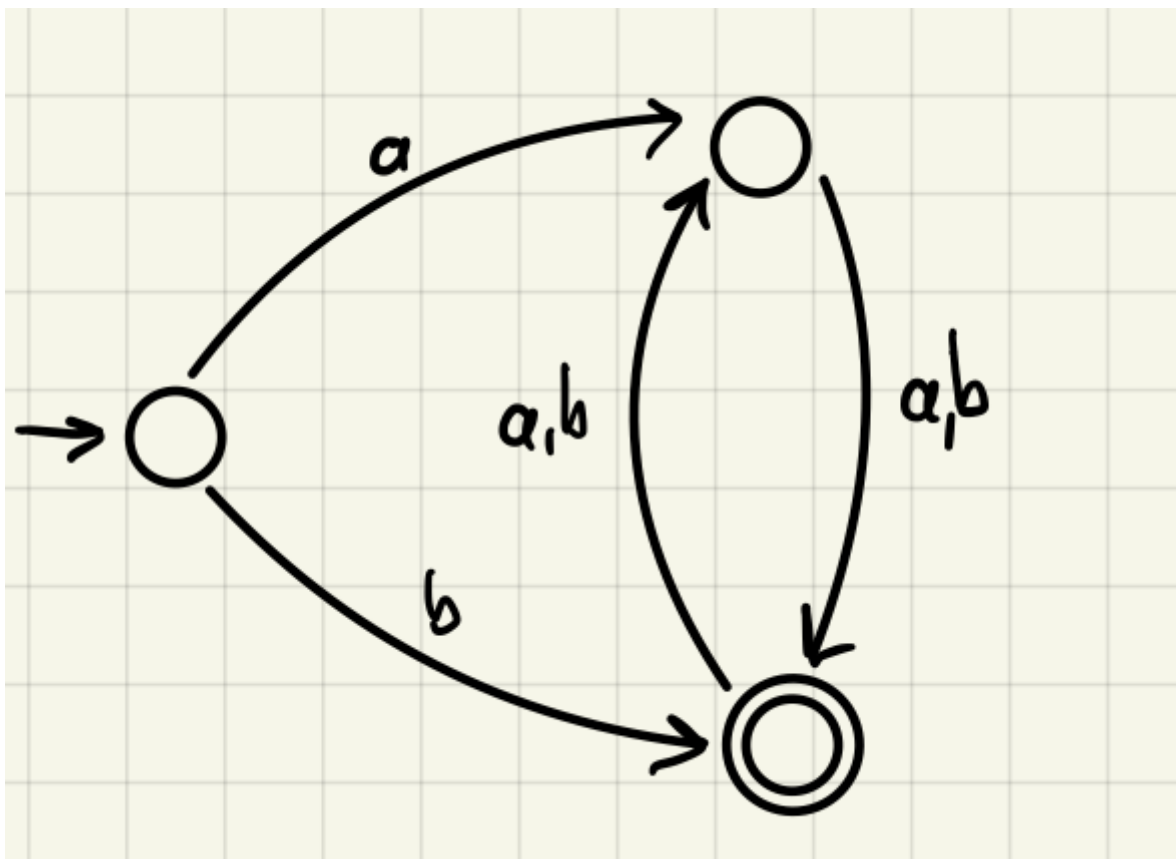
$b \in L_3$, beginnt nicht mit a und hat nicht gerade länge $\Rightarrow 0 \Leftrightarrow 0 = 1$

$baaaa \in L_3$, beginnt nicht mit a und hat nicht gerade länge $\Rightarrow 0 \Leftrightarrow 0 = 1$

Vorlesungsvariante:

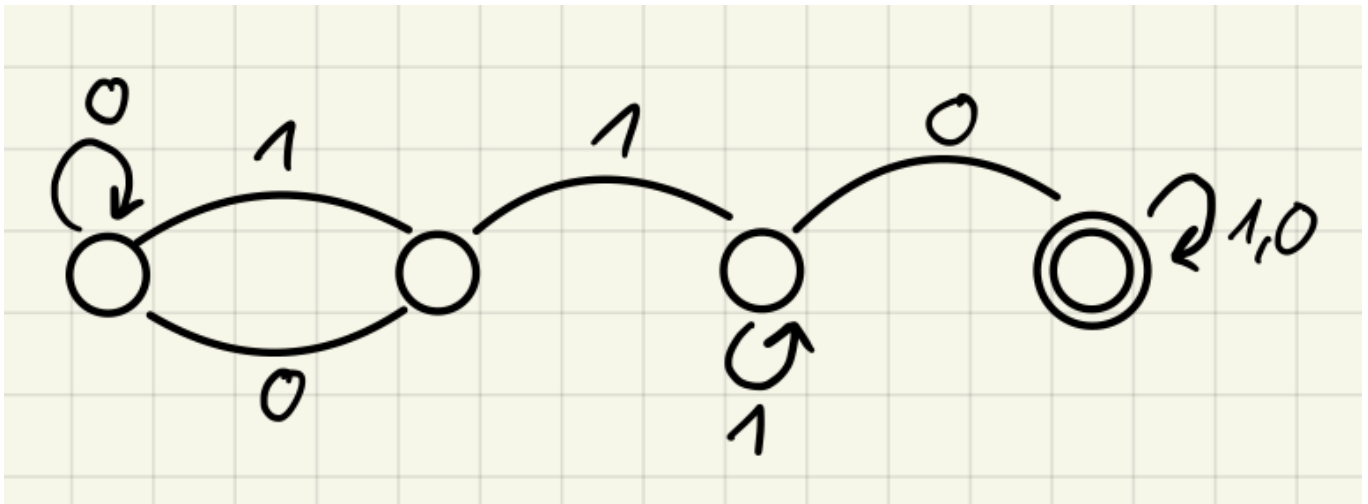


Wir haben eine optimierte Variante die richtig sein sollte!(Kein Gewähr):



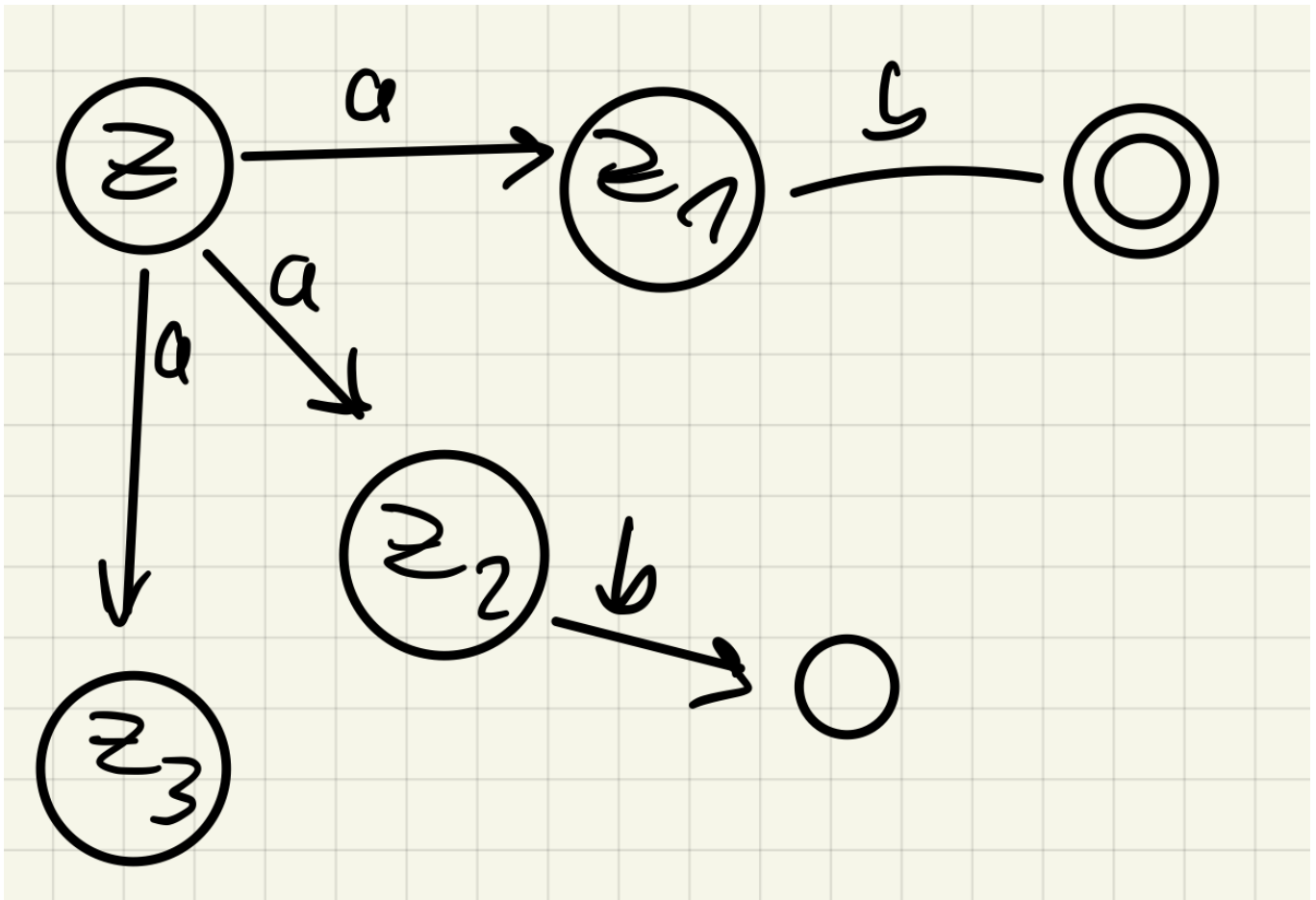
Letztes Beispiel:

$$L_4 = \{w \in \{0, 1\}^* \mid w \text{ enthält das Teilwort } 110\}$$



2.2 NFA

Idee: $\delta(z, a) = \{z_1, z_2, z_3\}$



$$\delta(z, b) = \emptyset$$

Ein NFA akzeptiert ein Wort w , wenn er bei Eingabe w eine Zustandsfolge durchlaufen kann, die zu einem Endzustand führt.

Definition 2.2.1

Ein NFA ist ein 5 Tupel $N = (\Sigma, Z, \delta, S, Z_E)$

1. Σ ist eine endliche Menge ... Eigenschaften
2. Z ist eine endliche Menge ... Zustandsmenge
3. $\delta : Z \times \Sigma \rightarrow P_{\text{otenzmenge}}(Z)$... Überföhrungsfunktion
4. $S \subseteq Z$... Menge Startzustände
5. $Z_E \subseteq Z$... Menge Endzustände

Definition 2.2.2

Die erweiterte Überföhrungsfunktion:

$\delta^* : P(Z) \times \Sigma^* \rightarrow P(Z)$, sei wie folgt definiert: Für alle $\Sigma \subseteq Z, a \in \Sigma, w \in \Sigma^*$ ist

$$\delta^*(\tilde{z}, \lambda) = \tilde{z}$$

$$\delta^*(\tilde{z}, aw) = \bigcup_{z \in \tilde{Z}} \delta^*(\delta(z, a), w)$$

$\delta(z, a)$... Menge von z nach Abarbeitung von a erreichabren Zuständen

$\delta^*(\tilde{z}, w)$... Menge der Zustände die erreicht werden, wenn in $z \in \tilde{Z}$ gestartet wird und w abgearbeitet wird.

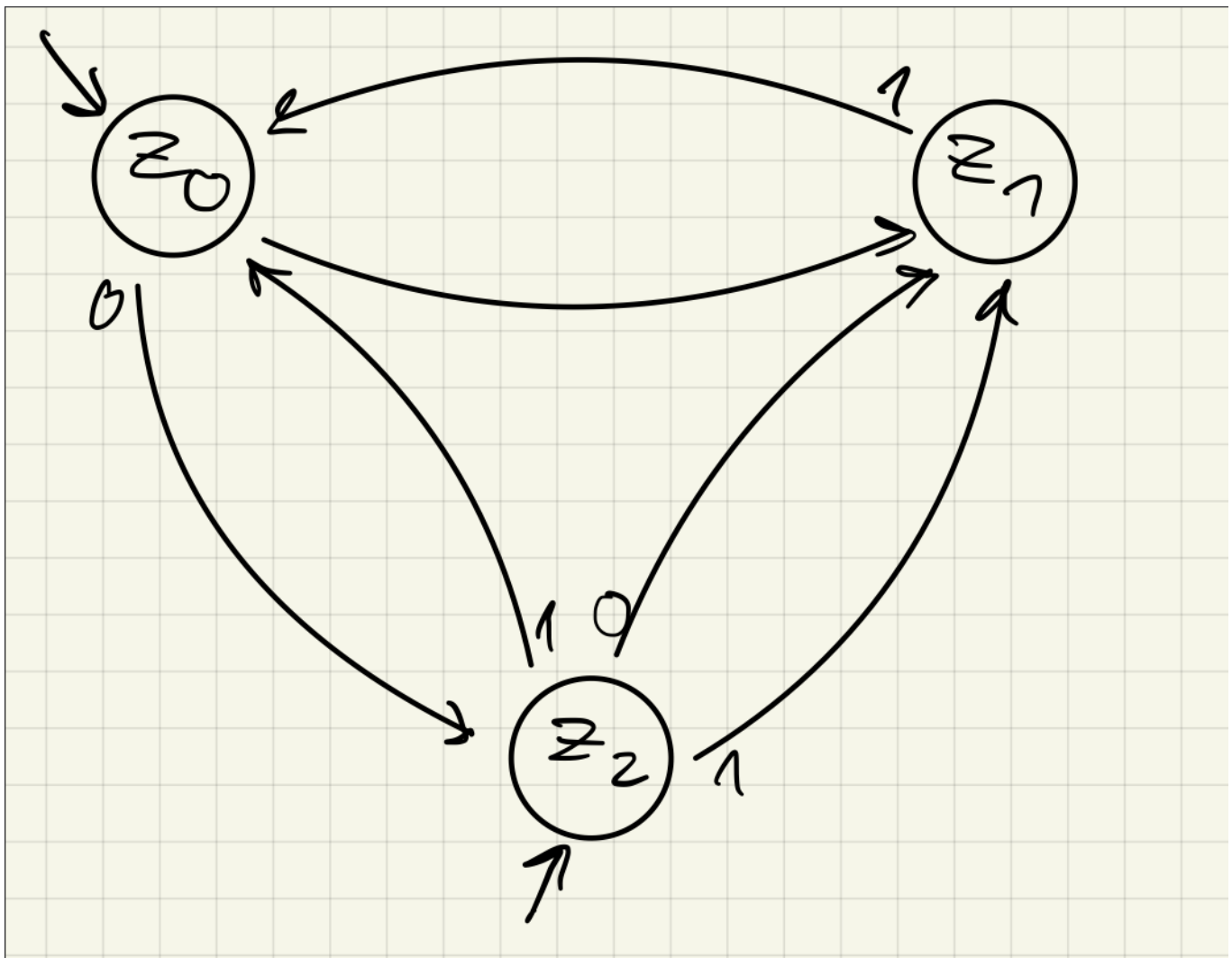
Definition 2.2.3

Die von einem NFA $M = \{\Sigma, Z, \delta, S, Z_E\}$ akzeptierte Sprache ist

$$L(M) = \{w \in \Sigma^* \mid \delta^*(s, w) \cap Z_E \neq \emptyset\}$$

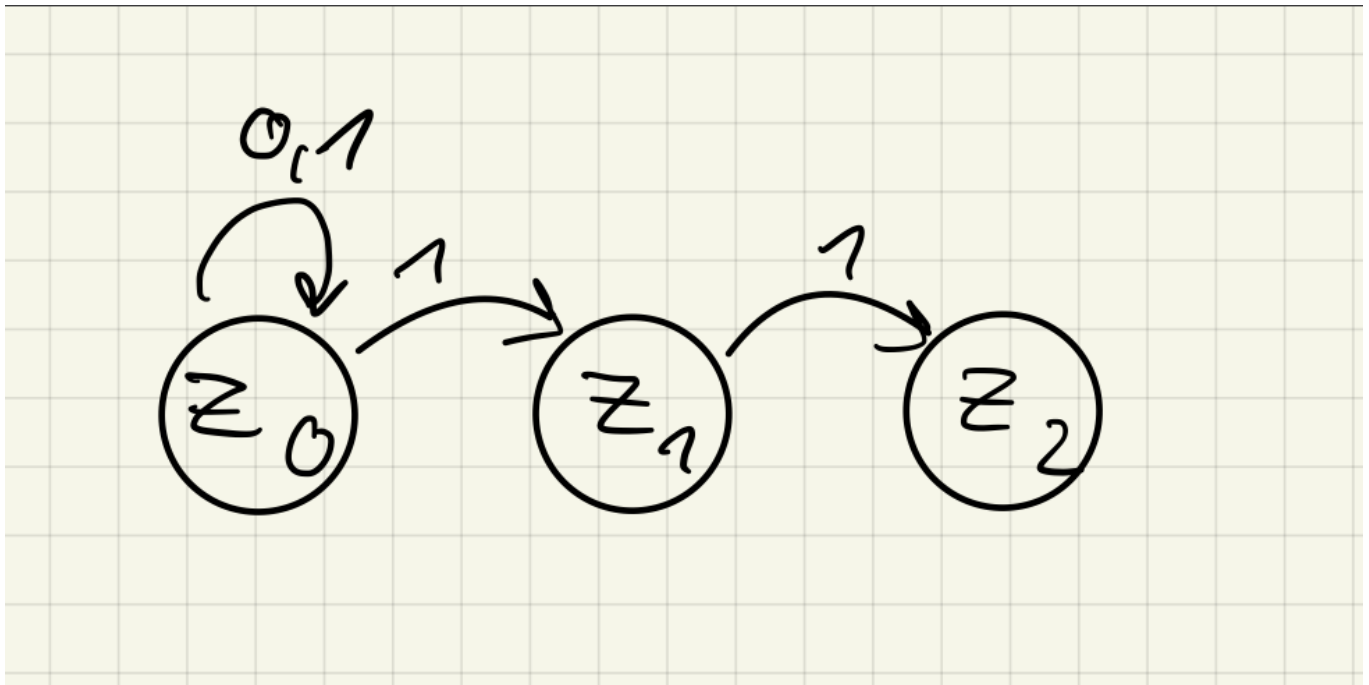
Bsp:

$$M = (\{0, 1\}, \{z_0, z_1, z_2\}, \delta, \{z_0, z_2\}, \{z_1\})$$



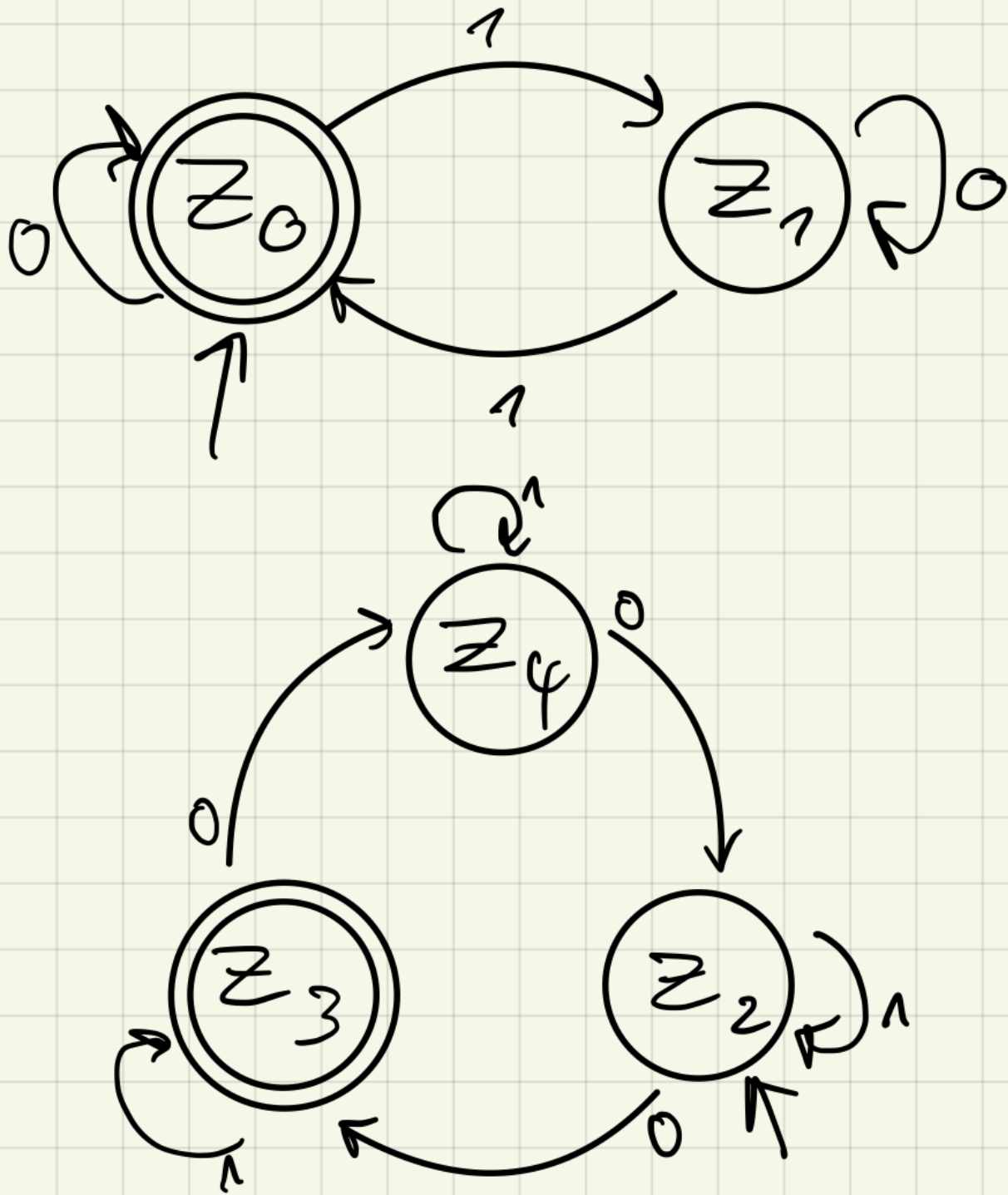
$$\begin{aligned}
 & \delta^*({z_0, z_2}, 100) \\
 &= \delta^*(\underbrace{\delta(z_0, 1)}_{=\emptyset}, 00) \cup \delta^*(\delta(z_2, 1), 00) \\
 &= \delta^*(\emptyset, 0) \cup \delta^*({z_2, z_1}, 00) \\
 &= \bigcup_{z \in \emptyset} \delta^*(\delta(z, 0), 0) \cup \delta^*(\delta(z_0, 0), 0) \cup \delta^*(\delta(z_1, 0), 0) \\
 &= \delta^*({z_1, z_2}, 0) \cup \underbrace{\delta^*({z_2}, 0)}_{=\emptyset} \\
 &= \delta^*(\delta(z_1, 0), \lambda) \cup \delta^*(\delta(z_2, 0), \lambda) \\
 &= \delta^*({z_2}, \lambda) \cup \delta^*(\emptyset, \lambda) = {z_2} \\
 &\Rightarrow 100 \text{ wird } \underline{\text{nicht}} \text{ akzeptiert}
 \end{aligned}$$

Bsp:



$$L(M) = w \in \{0, 1\}^* \mid \#_1(w) \equiv_2 0 \text{ oder } \#_0(w) \equiv_3 1$$

$$M = (\{0, 1\}, \{z_0, \dots, z_4\}, \delta, \{z_0, z_3\}, \{z_1, z_3\})$$



101000 ... wird akzeptiert
 100 ... wird nicht akzeptiert

Satz 2.2.4

Jede reguläre Sprache kann von einem NFA akzeptiert werden.

Beobachtung:

Sei $A \in REG$, $A \subseteq \Sigma^*$, nach Definition gibt es eine Reguläre Grammatik

$G = (\Sigma, N, S, R)$ mit $A = \mathcal{L}(G)$. Wir betrachten folgenden NFA $M = (\Sigma, Z, \delta, S, Z_E)$

$Z = N \cup \{F\}$ $F \notin N$

$S' = \{S\}$

$$Z_E = \begin{cases} \{F\} & \text{falls } (S \rightarrow \lambda) \notin R \\ \{F, S'\} & \text{falls } (S \rightarrow \lambda) \in R \end{cases}$$

$$\delta(X', a) = \begin{cases} \{Y' \mid X \rightarrow aY \in R\} & \text{falls } \{X \rightarrow a\} \notin R \\ \{Y' \mid X \rightarrow aY \in R\} \cup \{F\} & \text{falls } \{X \rightarrow a\} \in R \end{cases}$$

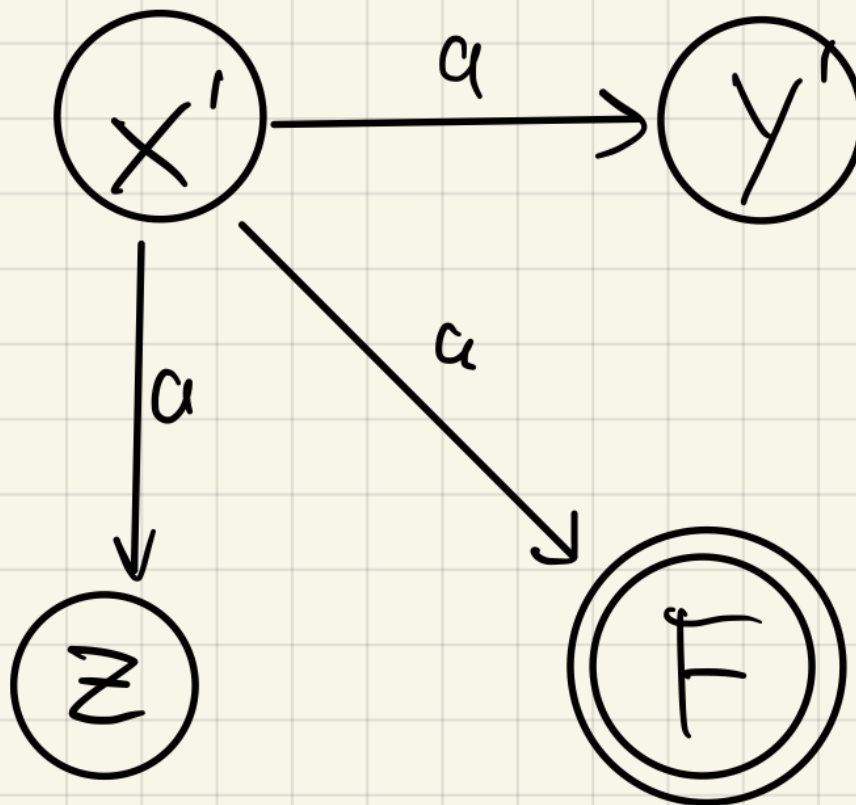
Anmerkung: " ' " an einem Buchstaben (wie in X' oder Y') steht für den Zustand eines Automaten.

Idee in Grammatik

$X \rightarrow aY$

$X \rightarrow aZ$

$X \rightarrow a$



Offenbar gilt für $w = a_1 a_2 \dots a_n \in A \Leftrightarrow a_1 \dots a_n \in \mathcal{L}(G)$

\Leftrightarrow es gibt eine Folge von Nichtdeterministischen

$X_1, X_2, \dots, X_{n-1} \in N$, so dass

$S \vdash_G a_1 X_1 \vdash_G a_1 a_2 X_2 \vdash_G \dots \vdash_G a_1 a_2 \dots a_{n-1} X_{n-1} \vdash_G a_1 \dots a_n$

\Leftrightarrow es gibt eine Folge von Zuständen

$X'_1, X'_2, \dots, X'_{n-1} \in Z$ mit $X'_1 \in \delta(S, a_1)$, $X'_2 \in \delta(X'_1, a_2)$, $X'_{n-1} \in \delta(X'_{n-2}, a_{n-1})$ und $X'_n \in \delta(X'_{n-1}, a_n)$

$\Leftrightarrow a_1 \dots a_n \in L(M)$

$DFA \rightarrow$ reguläre Grammatik $\rightarrow NFA$

Satz 2.2.5

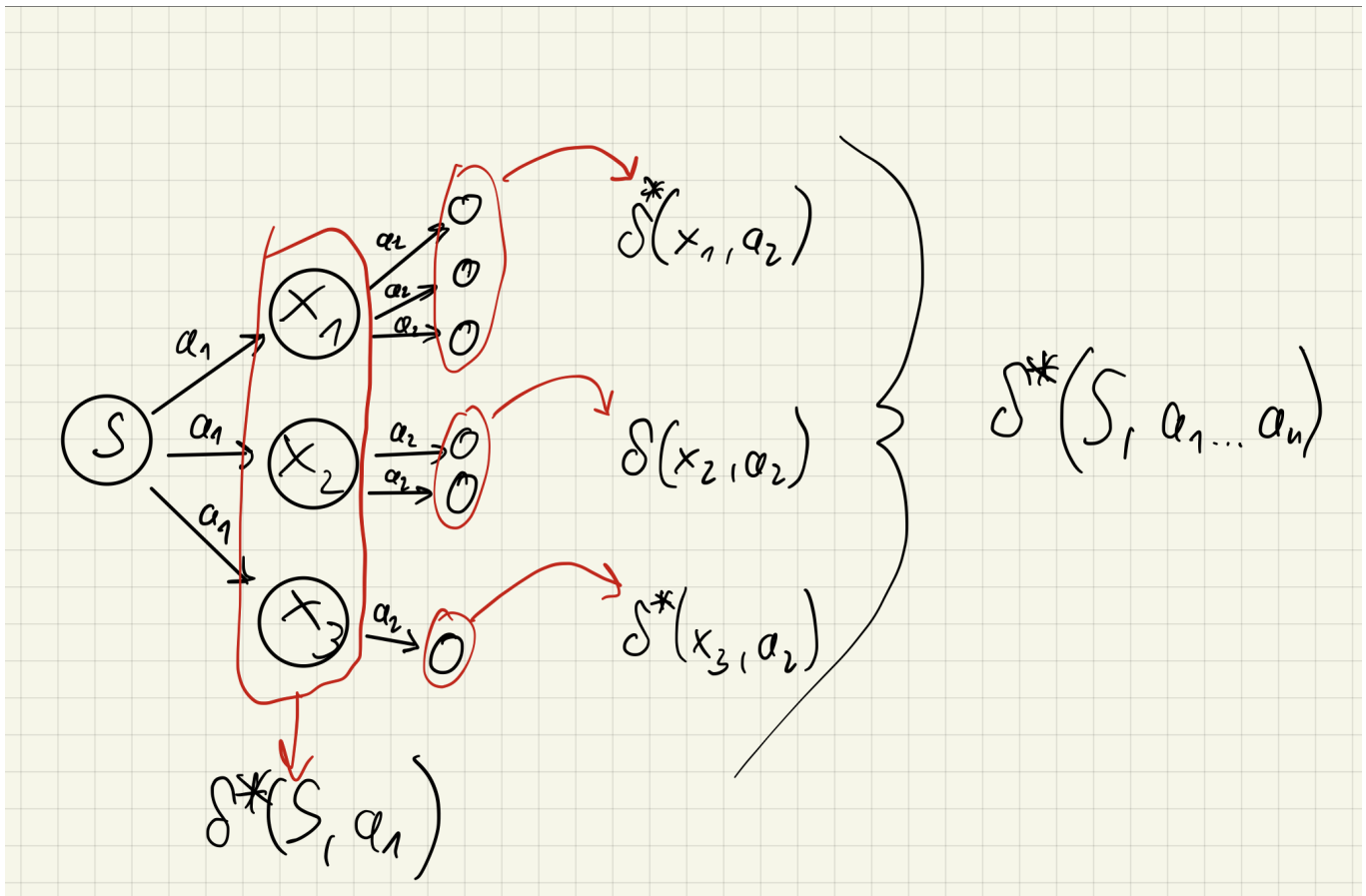
Jede Sprache, die von einem NFA akzeptiert wird, kann auch von einem DFA akzeptiert werden.

Beweis:

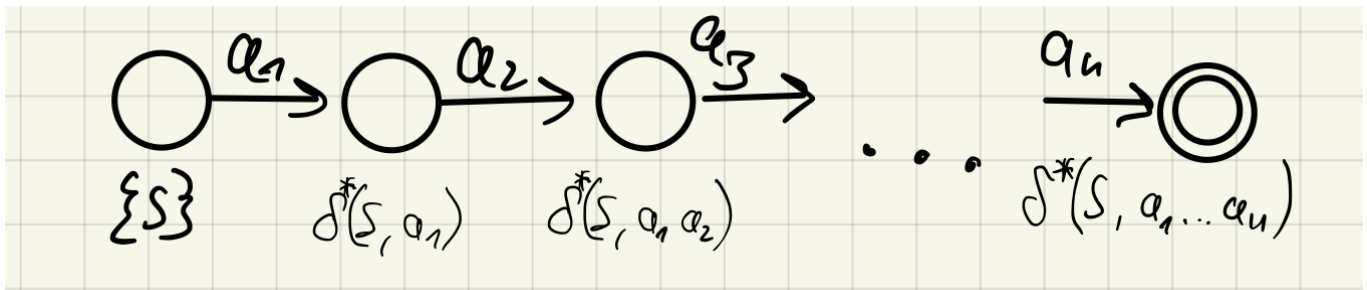
Sei $A \subseteq \Sigma^*$ und sei $M = (\Sigma, Z, \delta, S, Z_E)$ ein NFA mit $L(M) = A$

Idee: Abarbeitung von $w = a_1 \dots a_n$

NFA:



DFA:



Wir definieren DFA $M' = (\Sigma, P(M), \delta', S', Z'_E)$

$\delta'(Z', a) = \bigcup_{z \in Z'} \delta(z, a) = \delta^*(Z', a)$, für $Z' \in P(M)$ $Z' \in Z$

$$S' = S$$

$$Z'_E = \{Z' \subseteq Z \mid Z' \cap Z_E \neq \emptyset\}$$

Für jedes Wort $w = a_1 \dots a_n$ $a_i \in \Sigma^*$ gilt

$$w \in L(M) \Leftrightarrow \delta^*(S, w) \cap Z_E \neq \emptyset$$

\Leftrightarrow Es gibt eine Folge von Teilmengen $Z_1 \dots Z_n$ von Z mit $\delta^*(S, a_1) = Z_1$

$$\delta^*(Z_1, a_2) = Z_2 \dots \delta^*(Z_{n-1}, a_n) = Z_n \text{ und } Z_n \cap Z_E \neq \emptyset$$

\Leftrightarrow es gibt eine Folge von Zuständen Z_1, \dots, Z_n von M' mit

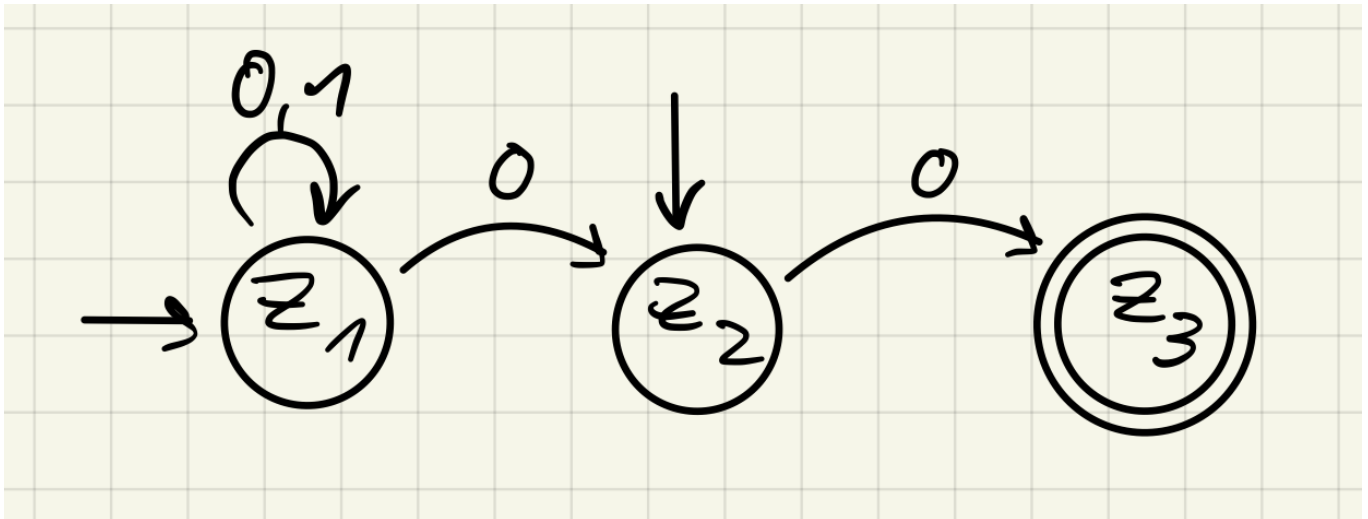
$$\delta'(S', a_1) = Z_1, \delta'(Z_2, a_2) = Z_2 \dots \delta'(Z_{n-1}, a_n) = Z_n \text{ und } Z_n \cap Z_E \neq \emptyset$$

$$\Leftrightarrow \delta^*(S, a_1 \dots a_n) \in Z'_E$$

Bsp:

$$NFA M = (\{0, 1\}, \{z_1, z_2, z_3\}, \delta, \{z_1, z_2\}, \{z_3\})$$

δ	0	1
z_1	$\{z_1, z_2\}$	$\{z_1\}$
z_2	$\{z_3\}$	\emptyset
z_3	\emptyset	\emptyset



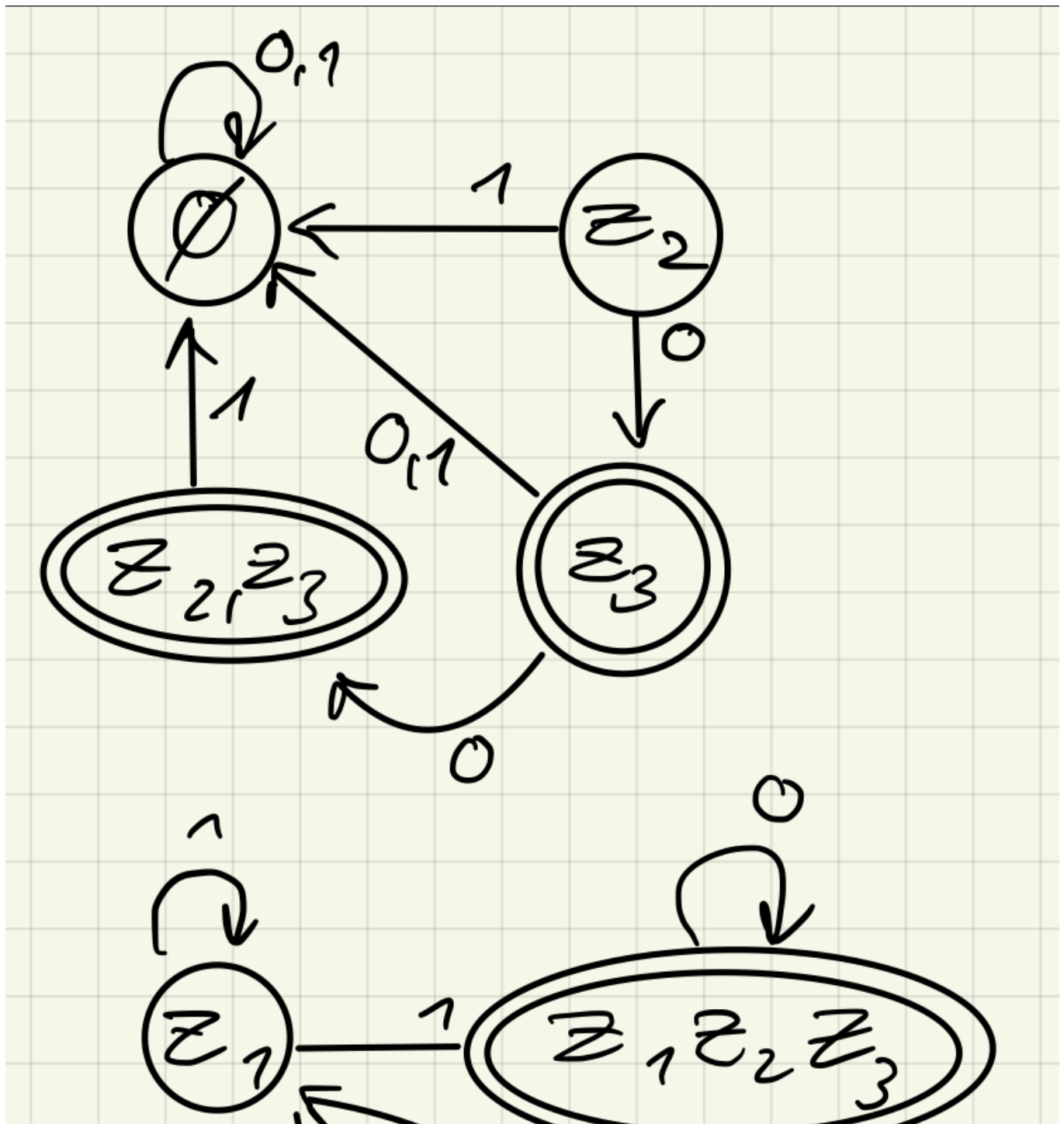
$$DFA M' = (\{0, 1\}, Z', \delta', S', Z_E)$$

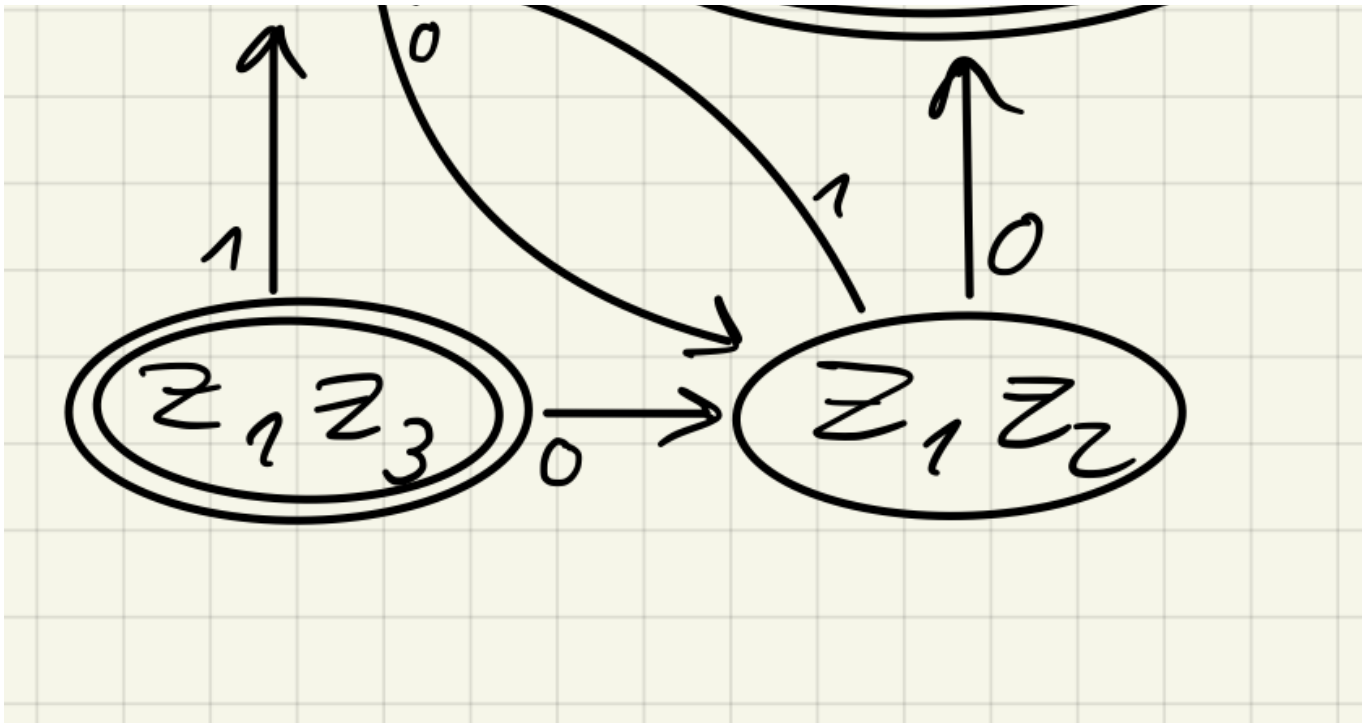
$$S' = \{z_1, z_2\} = q_4$$

$$Z_E = \{\underbrace{\{z_3\}}_{q_3}, \underbrace{\{z_1, z_3\}}_{q_5}, \underbrace{\{z_2, z_3\}}_{q_6}, \underbrace{\{z_1, z_2, z_3\}}_{q_7}\}$$

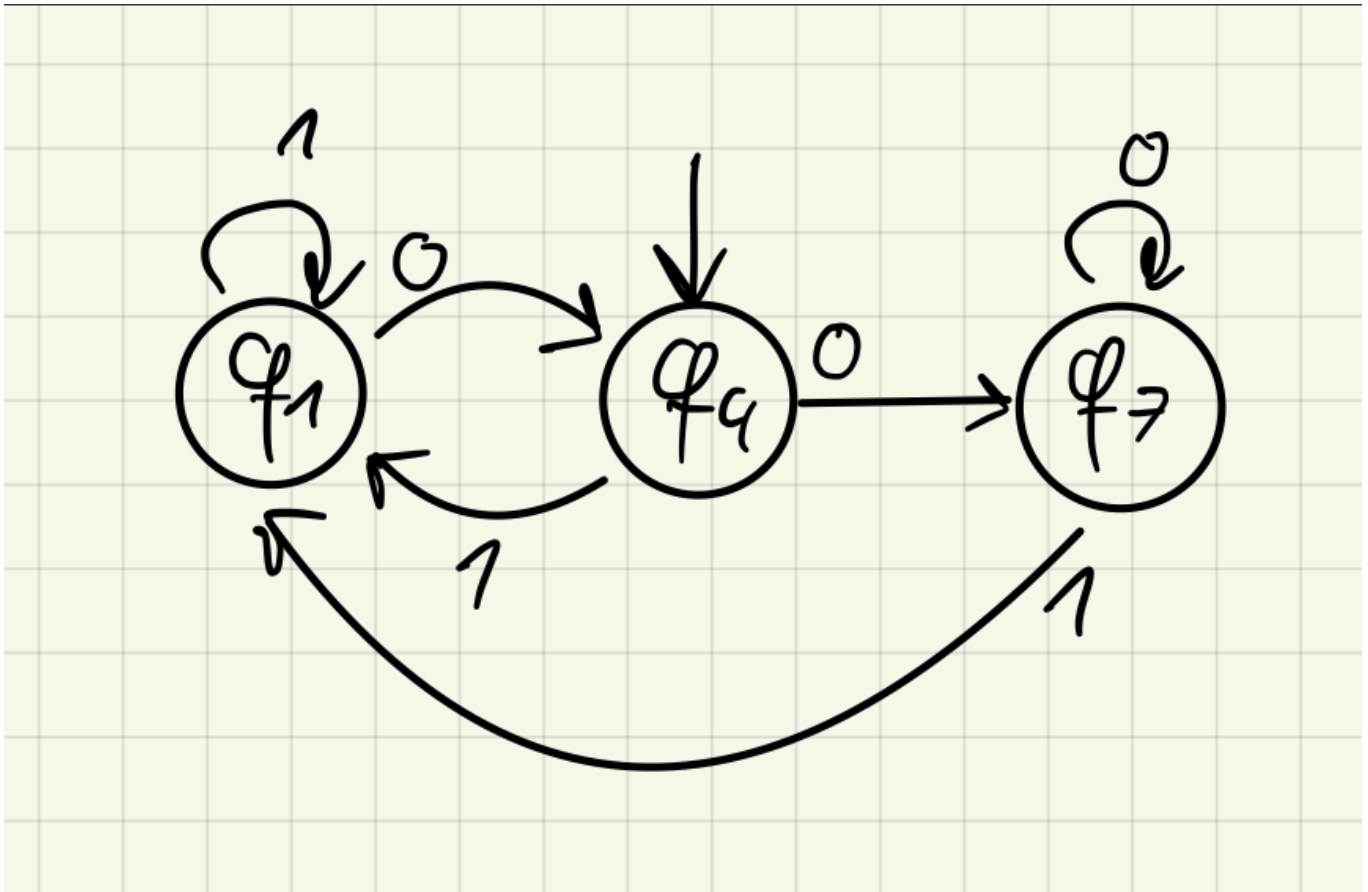
$$Z' = P(\{z_1, z_2, z_3\}) = \{\emptyset, \{z_1\}, \{z_2\}, \{z_3\}, \{z_1, z_2\}, \{z_1, z_3\}, \{z_2, z_3\}, \{z_1, z_2, z_3\}\}$$

	q ₀	q ₁	q ₂	q ₃	q ₄	q ₅	q ₆	q ₇
δ	∅	z ₁	z ₂	z ₃	z ₁ , z ₂	z ₁ , z ₃	z ₂ , z ₃	z ₁ , z ₂ , z ₃
0	∅	z ₁ , z ₂	z ₃	∅	z ₁ , z ₂ , z ₃	z ₁ , z ₂	z ₃	z ₁ , z ₂ , z ₃
1	∅	z ₁	∅	∅	z ₁	z ₁	∅	z ₁





Zustände die aus dem Startzustand nicht erreicht werden können, können weggelassen werden.



Beweis regulärer Ausdrücke:

Bemerkung:

- Zu gegebenen NFA erhält man durch Potenzmengenkonstruktion einen DFA mit 2^n Zuständen
- In unserem Beispiel ist unserer noch verkleinerbar
- Es gibt Beispiele bei denen alle 2^n Zustände gebraucht werden

2.3 Reguläre Ausdrücke

Definition 2.3.1

Es sei Σ ein Alphabet. Die Menge der regulären Ausdrücke über Σ , $RA(\Sigma)$ wird definiert durch:

1. λ und \emptyset sind reguläre Ausdrücke
2. Für jedes $a \in \Sigma$ ist a ein regulärer Ausdruck
3. Seien α und β reguläre Ausdrücke, so sind auch $\alpha \cdot \beta$, $(\alpha + \beta)$, $(\alpha)^*$ reguläre Ausdrücke
4. Andere reguläre Ausdrücke gibt es nicht

Bemerkung: $RA(\Sigma)$ ist eine Sprache über $\{\emptyset, \lambda, (,), \neg, *\} \cup \Sigma$

$RA(\Sigma)$ ist nicht regulär aber Kontextfrei

Definition 2.3.2

Es sei γ ein regulärer Ausdruck über Σ , $\gamma \in RA(\Sigma)$

Die Sprache $L(\gamma) \subseteq \Sigma^*$ ist wie folgt definiert:

1. $L(\emptyset) = \emptyset$ und $L(\lambda) = \lambda$, ($\gamma = 0, \gamma = \lambda$)
2. $L(a) = \{a\}$ für alle $a \in \Sigma$

$$3. \quad L(\gamma) = \begin{cases} L(\alpha) \cdot L(\beta) & , \text{ falls } \gamma = \alpha\beta \\ L(\alpha) \cup L(\beta) & , \text{ falls } \gamma = (\alpha + \beta) \\ L(\alpha) & , \text{ falls } \gamma = (\alpha)^* \end{cases}$$

Bsp: $\Sigma = \{a, b\}$

$$L\left(\left((a + b)\right)^*\right) = \{a, b\}^*$$

$$L\left((a(a+b))^* abba(a+b))^*\right) = \{a\}\{a,b\}^*\{abba\}\{a,b\}^*$$

$$L\left((b)^* a(b)^* a(b)^*\right) = \{b\}^* a \{b\}^* a \{b\}^*$$

Lemma 2.3.3

Jede endliche Sprache kann durch einen regulären Ausdruck beschrieben werden.

Beobachtung:

Sei

$$w_1 = a_{11}a_{12}\dots a_{1m_1}$$

$$w_2 = a_{21}a_{22}\dots a_{2m_2}$$

\vdots

$$w_n = a_{n1}a_{n2}\dots a_{nm_n}$$

Idee:

$$\left(((w_1 + w_2) + w_3) \dots + w_n \right) \\ \gamma \left(((a_{11}a_{12}\dots a_{1m_1} + a_{21}a_{22}\dots a_{2m_2})) + \dots + a_{n1}a_{n2}\dots a_{nm_n} \right)$$

Satz 2.3.4 (Satz von Kleene)

Eine Sprache ist genau dann regulär, wenn sie durch einen regulären Ausdruck beschrieben werden kann.

" \Rightarrow " Sei $A \subseteq \Sigma^*$ eine reguläre Sprache und sei $M = (\Sigma, Z, \delta, S, Z_E)$ ein DFA mit $L(M) = A$. Wir geben einen regulären Ausdruck γ an, für den gilt: $L(\gamma) = L(M)$ gilt. Sei $Z = \{z_1, \dots, z_n\}$. Wir definieren für $i, j \in \{1, \dots, n\}$ und $k \subseteq \{1, \dots, \}$

$$L_{i,j}^k = \{w = x_1 \dots x_m \in \Sigma^* \mid \delta^*(z_i, w) = z_j\}$$

$$\text{und } \bigwedge_{i \leq l \leq m-1} \delta^*(z_i, x_1 \dots x_l) = z_r \rightarrow r \leq k\}$$

(keiner der erreichbaren Zwischenzustände hat Index der gröÙe k)

$$k = 0 \quad \text{und} \quad i \neq j$$

$$L_{i,j}^0 = \{a \in \Sigma \mid \delta(z_i, a) = z_j\}$$

$$k = 0 \quad \text{und} \quad i = j$$

$$L_{i,j}^0 = \{a \in \Sigma \mid \delta(z_i, a) = z_j\} \cup \{\lambda\}$$

Offenbar ist $L_{i,j}^0$ endlich und lässt sich somit durch einen regulären Ausdruck beschreiben.

Beobachtung:

Induktion über k:

I.A.: $k = 0$ siehe oben

I.V.: Sei $K \geq 0$ $L_{i,j}^K$ lässt sich für jedes i, j durch regulären Ausdruck beschreiben

I.B.: Auch $L_{i,j}^{K+1}$ lässt sich durch regulären Ausdruck beschreiben

$k \rightarrow k+1$ Sei $i, j \in \{1, \dots, n\}$

beliebig aber fest gewählt

Offenbar ist

$$L_{\underbrace{i}_{z_{k+1}, \text{ wird nicht benutzt}}}^{k+1}, j = L_{i,j}^k \cup \underbrace{(L_{i,k+1}^k (L_{k+1,k+1}^k)^* L_{k+1,j}^k)}_{z_{k+1} \text{ wird ein oder mehrmals benutzt}}$$

Seien nun $\alpha_{i,j}^k, \alpha_{i,k+1}^k, \alpha_{k+1,k+1}^k, \alpha_{k+1,j}^k$ - reguläre Ausdrücke

für $L_{i,j}^k, L_{i,k+1}^k, L_{k+1,k+1}^k, L_{k+1,j}^k$

Diese gilt es nach I.V.:

Dann ist

$$L_{i,j}^{k+1} = L\left((\alpha_{i,j}^k \alpha_{i,k+1}^k (\alpha_{k+1,k+1}^k)^* \alpha_{k+1,j}^k)\right)$$

Nun lässt sich $L(M)$ ausdrücken als

$$L(M) = \bigcup_{z_i \in Z_E} L_{i,i}^n$$

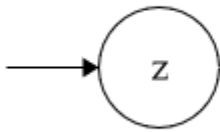
Sei i_1, i_2, \dots, i_m die Indizes der Endzustände so ist

$$L(M) = \left(\alpha_{1,i_1}^n + \left(\alpha_{1,i_1}^n + (\dots (\alpha_{1,i_1}^m) \dots) \right) \right)$$

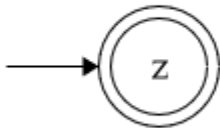
" \Leftarrow " Idee: Die Induktive Definition von $RA(\Sigma)$ | Kann mit NFA's noch vollzogen werden.

Sei $A \subseteq \Sigma^*$ und sie $\gamma \in RA(\Sigma)$ mit $L(\gamma) = A$ Wir geben einen *NFA* M für A an.

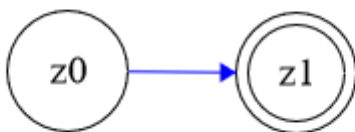
1.) Ist $\gamma = \emptyset$ $M = (\Sigma, \{z_0\}, \delta, \{z_0\}, \emptyset)$



2.) $\gamma = \{\lambda\}$ $M = (\Sigma, \{z_0\}, \delta, \{z_0\}, \{z_0\})$



3.) $\gamma = a \in \Sigma$ $M = (\Sigma, \{z_0, z_1\}, \delta, \{z_0\}, \{z_1\})$



4.) $\gamma = \alpha \cdot \beta; \gamma = (\alpha)^*$

Seien $M_1 = (\Sigma, Z_1, \delta_1, Z_{01}, Z_{E_1})$

und $M_2 = (\Sigma, Z_2, \delta_2, Z_{02}, Z_{E_2})$

$NFA's$ mit $L(M_1) = L(\alpha)$ und $L(M_2) = L(\beta)$

Ohne Beschränkung der Allgemeinheit (o.B.d.A) $Z_1 \cap Z_2 = \emptyset$

Wir konstruieren $M = (\Sigma, Z, \delta, Z_0, Z_E)$

mit $L(M) = L(\gamma)$

a) $\gamma = \alpha + \beta$

$Z = Z_1 \cup Z_2$

$$\delta(z, a) = \begin{cases} \delta_1(z, a) & , \text{ falls } z \in Z_1 \\ \delta_2(z, a) & , \text{ falls } z \in Z_2 \end{cases} \quad \begin{matrix} Z_0 = Z_{01} \cup Z_{02} \\ Z_E = Z_{E1} \cup Z_{E2} \end{matrix}$$

b) $\gamma = \alpha \cdot \beta$

$Z = Z_1 \cup Z_2$

$$Z_0 = \begin{cases} Z_{01} \cup Z_{02} & , \text{ falls } \lambda \in L(M_1) \\ Z_{01} & , \text{ falls } \lambda \notin L(M_1) \end{cases}$$

$Z_E = Z_{E2}$

Beweis Satz 2.3.4

" \Leftarrow " Wenn $L \subseteq \Sigma^*$ durch einen regulären Ausdruck beschreibbar ist, dann ist L Regulär

Zu zeigen: es gibt einen NFA der L akzeptiert

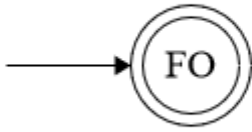
c) $\gamma = (\alpha)^*$ $NFA M = (\Sigma, Z, \delta, Z_0, Z_E)$

$M_1 = \Sigma, Z, \delta', Z'_0, Z'_E$

$\lambda \in L(M_1)$, so ist $M'_1 = M_1$

$\lambda \notin L(M_1)$,

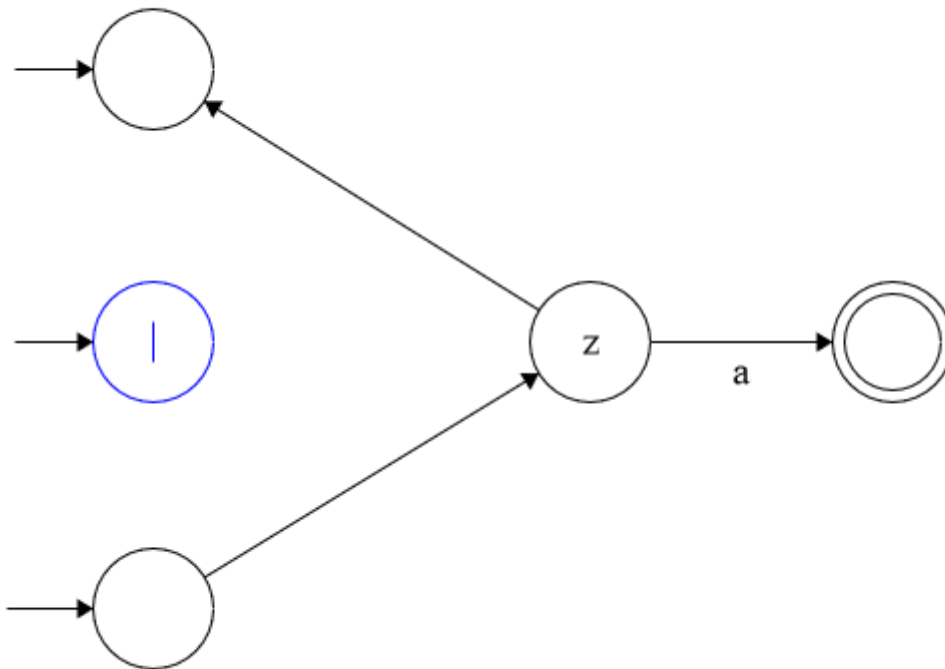
$M'_1 = (\Sigma, Z_1 \cup \{F0\}, \delta', Z_0 \cup \{F0\}, Z_E \cup \{F0\})$



Offenbar ist $L(M'_1) = L(M_1) \cup \{\lambda\}$

$M'_1 = \Sigma, Z', \delta'', Z'_0, Z'_E$ mit

$$\delta''(z, a) = \begin{cases} \delta'(z, a) \cup Z_0 & , \text{ falls } \delta'(z, a) \in Z'_E \\ \delta'(z, a) & \end{cases}$$



Folgerung 2.3.5

Sei $A \subseteq \Sigma^*$

Die folgenden Aussagen sind äquivalent

1. A ist eine reguläre Sprache
2. A kann von einem *DFA* akzeptiert werden

3. A kann von einem NFA akzeptiert werden
4. A kann durch einen regulären Ausdruck beschrieben werden

2.4 Das Pumping Lemma

bisher: Wir wissen wann eine Sprache regulär ist. Der Nachweis der Nichtregularität ist schwieriger, da man zeigen musste, kein DFA , kein NFA , keine reguläre Grammatik gibt, der die Sprache akzeptiert/erzeugt.

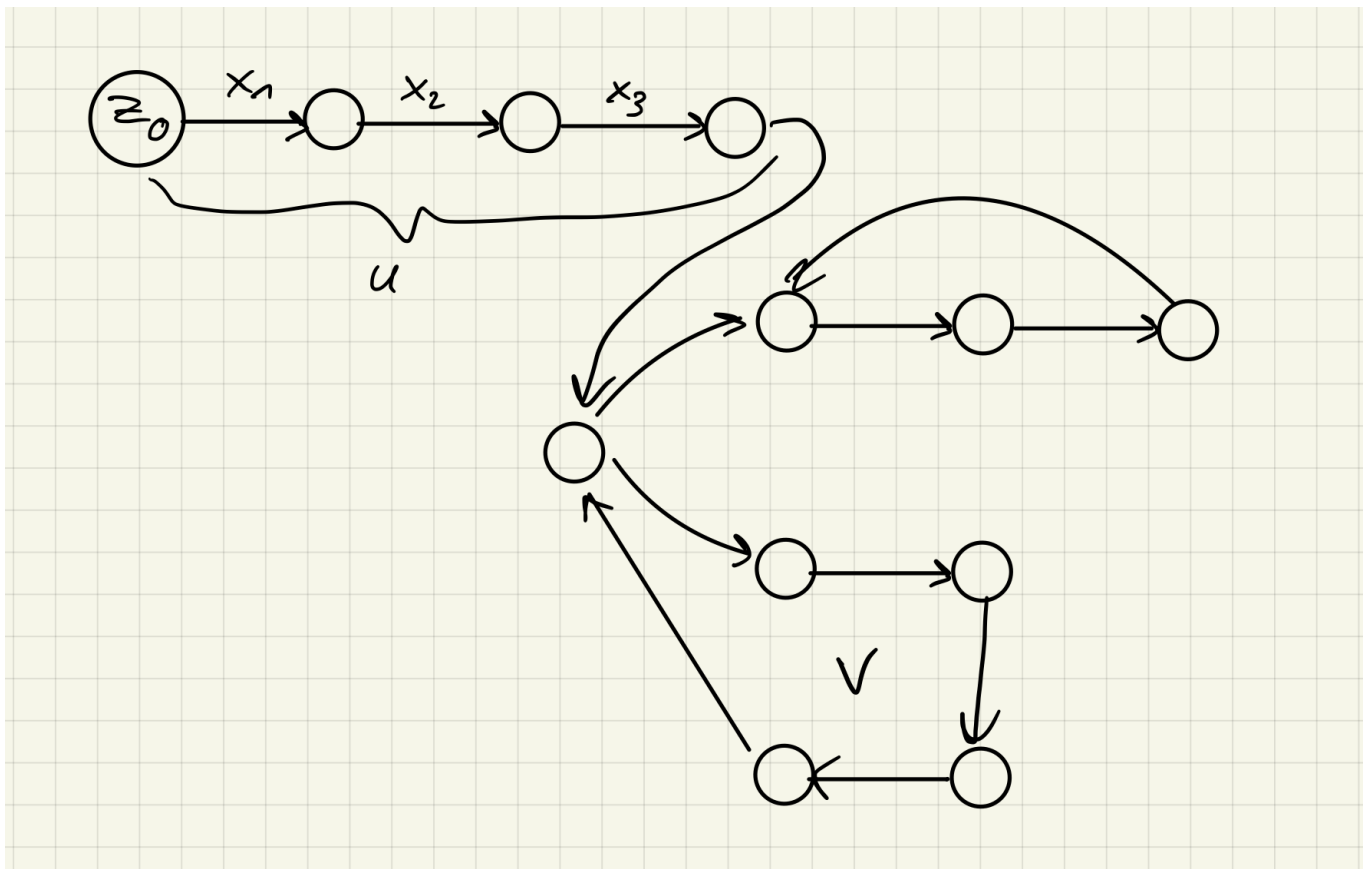
Satz 2.4.1 Pumping Lemma

Sei A eine reguläre Sprache. Dann gibt es ein $n \in \mathbb{N}$, so dass sich alle Wörter $x \in A$ mit $|x| \geq n$ so in $x = uvw$ zerlegen lassen, dass folgende Bedingungen erfüllt sind.

1. $|v| \geq 1$
2. $|uv| \leq n$
3. Für $i > 0$ $uv^i w \in A$

Beweis:

Sei $A \in REG$. Sei $M = \Sigma, Z, \delta, Z_0, Z_E$ ein DFA mit $A = L(M)$. Wir wählen $n = |Z|$. Sei nun $x \in A$ mit $|x| = n$. Beim Abarbeiten von x durchläuft $M : |x| + 1$ Zuständen (inklusive Startzustand). Da $|x| = n$, können nicht alle Zustände verschieden sein. M hat bei Abarbeitung x eine Schleife durch laufen.



Wir wählen nun u, v, w mit $x = uvw$, so dass $\delta^*(z_0, u) = \delta^*(z_0, uv)$

Man kann diese Zerlegung so wählen, dass $|v| \geq 1$ und $|uv| \leq n$ gilt.

Wegen $\delta^*(z_0, u) = \delta^*(z_0, uv)$ gilt auch $\delta^*(z_0, uw) = \delta^*(z_0, uvw)$

Mit anderen Worten

$$uvw \in A \Leftrightarrow uw \in A \Leftrightarrow uv^i w \in A$$

wegen $x \in A$ folgt Behauptung (B) \square

Bsp:

1.

$$A\{a^n b^n | n \geq 1\}$$

Annahme $A \in REG$, dann gibt es ein $n \in \mathbb{N}$, so dass sich alle $x \in A$ mit $|x| \geq n$ darauf in $x = uvw$ zerlegen lassen, dass gilt $|v| \geq 1$, $|uv| \leq n$, $uv^i w \in A$ für alle $i > 0$. Wähle $x = a^n b^n$ $|v| = 2n \geq n$.

Sei $x = uvw$ eine geeignete Zerlegung gemäß PL (Pumping Lemma), d.h. $|v| \geq 1$ und $|uv| \leq n$. Folglich kann v nur aus a's bestehen $v = a^k$ $1 \leq k \leq n$. Damit gibt $uv^0w = uw = a^{n-k}b^n \notin A$. Das ist ein Widerspruch und daraus folgt $A \notin REG$

2.

$$B = \{0^m | m \text{ ist Quadratzahl}\}$$

Annahme: $B \in REG$

Sei n die Pumpingzahl. Wähle $x = 0^{n^2}$ $x \in B$ $|x| \geq n$

Sei $x = uvw$ eine geeignete Zerlegung gemäß PL, d.h. $|v| \geq 1$ $|uv| \leq n$.

Sei $v = 0^k$ $1 \leq k \leq n$

$$n^2 = |x| = |uvw| \leq |uv^2w| \leq |uvw| + |v| = n^2 + k \leq n^2 + n \leq n^2 + n + 1 = (n+1)^2$$

$$\Rightarrow uv^2 \notin B. \text{ Damit ist } B \notin REG$$

Das Pumping Lemma liefert keine Charakterisierung der regulären Sprachen. Es stellt nur ein notwendiges Kriterium dar

$$A \in REG \Rightarrow \bigvee_n \bigwedge_{|x| \geq n} \bigvee_{u,v,w} x = uvw$$

$$|v| \geq 1 \quad |uv| \leq n, \bigwedge_{i \geq 0} uv^i w \in A$$

2.5 Minimalautomaten und Äquivalenzrelationen

Es kann jeder Sprache eine Äquivalenzrelation zugeordnet werden

Definition 2.5.1

Sei $A \subseteq \Sigma^*$

Dann ist $R_A = \{(x, y) \in \Sigma^* \times \Sigma^* \mid \bigwedge_{z \in \Sigma^*} (xz \in A \Leftrightarrow yz \in A)\}$

\Leftrightarrow beim Anhängen beliebiger Wörter an x und y dann verhalten sie sich bezüglich Mitgliedschaft zu A gleich.

R_A ist Äquivalenzrelation:

$$\bigwedge_{x \in \Sigma^*} (x, x) \in R_A$$

$$\bigwedge_{x, y \in \Sigma^*} (x, y) \in R_A \Rightarrow (y, x) \in R_A$$

$$\bigwedge_{x, y, u \in \Sigma^*} (x, y) \in R_A \Rightarrow (y, u) \in R_A \Rightarrow (x, u) \in R_A$$

Einschub/Erinnerung:

In 2.1 haben wir jedem DFA $M = \Sigma, Z, \delta, Z_0, Z_E$ eine Relation zu geordnet

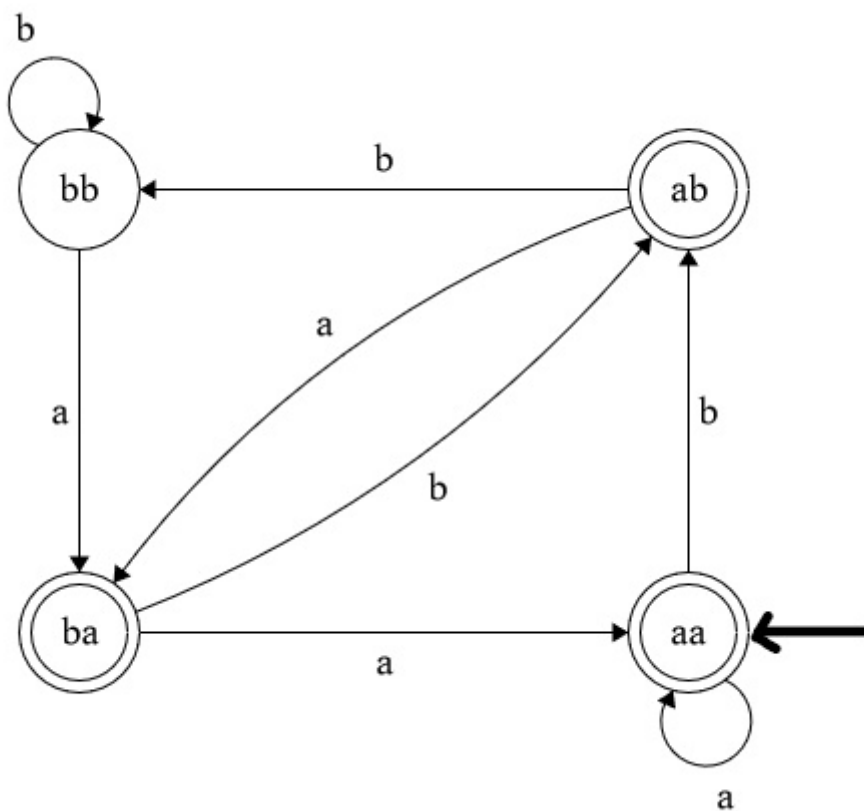
$$R_M = \{(x, y) \in \Sigma^* \times \Sigma^* \mid \delta^*(z_0) = \delta^*(z_0, y)\}$$

R_M ist Äquivalenzrelation

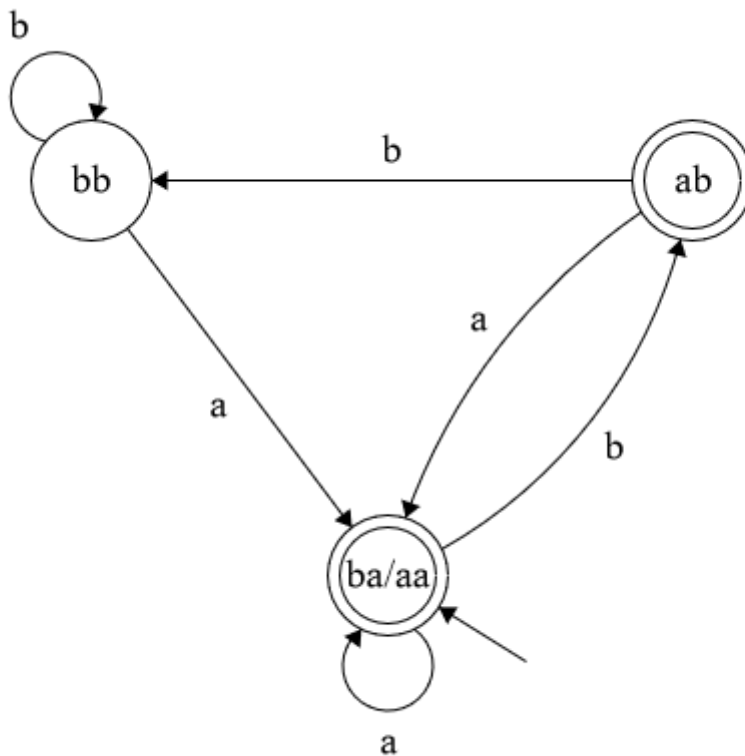
M_M ist rehtinvariant bzgl. Verkettung

$$(x, y) \in R_M \Leftrightarrow \bigwedge_{z \in \Sigma^*} (xz, yz) \in R_M$$

Bsp: endet nicht auf bb



Wird zu:



, da der Zustand ba äquivalenz zu aa ist

Wir werden zeigen das R_M eine Verfeinerung von R_A ist. Also

$\bigwedge_{x,y \in \Sigma^*} (x, y) \in R_M \Rightarrow (x, y) \in R_A$ Das liefert uns ein Kriterium für die Regularität.

Satz 2.5.2 (Myhil Neurode)

Die folgenden Aussagen sind äquivalent:

1. A ist regulär
2. A ist die Vereinigung von Äquivalenzklassen einer rechtsinvarianten Äquivalenzrelation mit endlichem Index
3. R_A hat endlichen Index

Beweis

1) \rightarrow 2) Sei $A \subseteq \Sigma^*$ Sei $M = (\Sigma, Z, \delta, Z_0, Z_E)$ ein DFA, ot $L = A$. Dann ist R_M eine rechtsinvariante Äquivalenzklasse von R_M kann mit genau einem Zustand $z \in Z$ identifiziert werden.

$[x]_M = \{y \in \Sigma^* \mid (x, y) \in R_M\} = \{y \in \Sigma^* \mid \delta^*(z_0, y)\}$, wird mit $\delta^*(z_0, x)$ identifiziert.

Nun ist A gerade die Vereinigung, derjenigen Äquivalenzklasse, die der Zuständen aus Z_E entsprechen. $A = \bigcup_{\delta^*(z_0, x) \in Z_E} [x]_M$

2) \rightarrow 3) Sei R eine rechtsinvariante Äquivalenzrelation und gelte 2) Wir zeigen, dass R eine Verfeinerung von R_A ist, also $\bigwedge_{x, y \in \Sigma^*} (x, y) \in R \Rightarrow (x, y) \in R_A$ gilt. Damit hält R_A

maximal so viele Klassen wie R . Sei $(x, y) \in R \xRightarrow{\text{rechtsinvariant}} \bigwedge_{z \in \Sigma^*} (xz, yz) \in R$

$\Rightarrow \bigwedge_{z \in \Sigma^*} xz \in [xz]_R$ und $yz \in [yz]_R$

$\Rightarrow \bigwedge_{z \in \Sigma^*} xz \in A \Leftrightarrow yz \in A$

(denn A ist die Vereinigung von Klassen von R)

$\Rightarrow (x, y) \in R_A$

3) \rightarrow 1)

R_A habe endlich viele Index

Sei $[x]_A = \{y \in \Sigma^* \mid (x, y) \in R\}$ für alle $x \in \Sigma^*$

$\Rightarrow Z = \{[x] \mid x \in \Sigma^*\}$... ist eine Menge

$Z = \{[x] \mid x \in A\}$... ebenfalls

Wir betrachten den DFA $M = (\Sigma, Z, \delta, \{\lambda\}, Z_+)$, wobei $\delta([x], a) = [xa]$ für alle $x \in \Sigma^*$, $a \in \Sigma$. Es gilt nun $A = L(M)$, da $x \in A \Leftrightarrow [x] \in Z_+$

$\Leftrightarrow \delta^*([\lambda], x) \in Z_+ \quad \square$

Bemerkung:

- Im Beweis wird der Äquivalenzklassenautomat konstruiert
- Äquivalenzklassenautomat(ÄkA) sind immer Minimalautomaten
- es gibt keinen Automaten der die selbe Sprache akzeptiert und weniger Zustände hat
- Satz von Myhill Nerode kann sowohl als Beweis von Regularität als auch zum Beweis von Nichtregularität genutzt werden

Bsp: $A_1 = \{w \in \{a, b\}^* \mid \#_a(w) \equiv_3 0\}$

Behauptung:

$K_1 = [\lambda] = \{w \in \{a, b\}^* \mid \#_a(w) \equiv_3 0\}$

$$K_2 = [a] = \{w \in \{a, b\}^* \mid \#_a(w) \equiv_3 1\}$$

$$K_3 = [aa] = \{w \in \{a, b\}^* \mid \#_a(w) \equiv_3 2\}$$

$$\text{z. Z.: } x, y \in K_i \ i = 1, 2, 3 \Rightarrow (x, y) \in R_{A_1}$$

$$K_1 \cup K_2 \cup K_3 = \{a, b\}^*$$

K_i sind paarweise disjunkt

$\Rightarrow \text{Index } R_{A_1} = 3 \rightarrow A_1 \text{ ist regulär}$

$$A_2 = \{a^n b^n \mid n \geq 1\}$$

Behauptung: Jedes $[ai] \ i \in \mathbb{N}$ bildet eine Äquivalenzklasse

$$\text{z.Z.: } [a^i, a^j] \notin R_{A_2} \text{ für } i \neq j$$

sei $z = b^i$

Dann gilt:

$$a^i z = a^i b^i \in A_2$$

$$a^j z = a^j b^i \notin A_2$$

$\Rightarrow \text{Index von } A_2 = \infty$

$\Rightarrow A_2 \notin REG$

Bemerkung:

Zum Beweis der Nichtregularität müssen nicht alle Äquivalenzklassen bestimmt werden, es genügt z.z., dass es unendliche viele Klassen gibt.

Bsp:

Für komplette Äquivalenzklassenzerlegung von Σ^* durch eine Sprache

$$A = \{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$$

$$[\lambda] = \{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$$

$$[a^i] = \{w \in \{a, b\}^* \mid \#_a(w) - \#_b(w) = i\}, \text{ für } i \geq 1$$

$$[a] = \{a, aba, aab, baa, aaabb, \dots\} \dots \text{ welche immer ein a mehr haben}$$

$$[b^i] = \{w \in \{a, b\}^* \mid \#_a(w) - \#_b(w) = -i\}, \text{ für } i \geq 1$$

2.6 Abschlusseigenschaften

Definition 2.6.1

Sei $f := \mathbb{P}(\Sigma^*)^k \rightarrow \mathbb{P}(\Sigma^*)$ eine Funktion.

Eine Sprachfamilie $\zeta \subseteq \mathbb{P}(\Sigma^*)^k$ heißt abgeschlossen bezüglich f (oder Anwendung von f) wenn gilt $\bigwedge_{A_1, \dots, A_k \subseteq \Sigma^*} \left(\bigwedge_{1 \leq i \leq k} A_i \in \zeta \right) \Rightarrow f(A_1, \dots, A_k) \in \zeta$

SATZ 2.6.2

Die Menge der regulären Sprachen (REG) ist abgeschlossen bezüglich:

1. Vereinigung
2. Durchschnitt
3. Komplement
4. Produkt
5. Kleene Abschluss
6. Differenz

Beobachtung: 1) 4) 5) folgen unmittelbar aus dem Beweis von Satz 2.3.4

3) Sei $A \subseteq REG$ und $M = (\Sigma, Z, \delta, Z_0, Z_E)$ ein DFA mit $L(M) = A$. Dann gilt für $M' = (\Sigma, Z, \delta, Z_0, Z \setminus Z_E)$ gerade $L(M') = \bar{A} = \Sigma^* \setminus A$

2) Wegen

PDF Variant:

$$A \cap B = \overline{\bar{A} \cup \bar{B}}$$

Non-PDF Variant:

$$A \cap B = \neg(\neg(A) \cup \neg(B))$$

beides äquivalente Aussagen nur die PDF Variante kann man im Markdown Editor nicht sehen.

6) Wegen $A \setminus B = A \cap \bar{B} = A \cap \neg(B)$ und 2) und 3)
direkter Beweis für 2):

Sei $A = L(M_1)$, mit $M_1 = (\Sigma, Z_1, \delta_1, Z_{0_1}, Z_{E_1})$

und $A = L(M_2)$, mit $M_2 = (\Sigma, Z_2, \delta_2, Z_{0_2}, Z_{E_2})$

Wir betrachten

$$M = (\Sigma, Z_1 \times Z_2, \delta, (Z_{0_1}, Z_{0_2}, Z_{E_1} \times Z_{E_2}))$$

$$\text{wobei } \delta((z_1, z_2), a) = (\delta_1(z_1, a), \delta_2(z_2, a))$$

$$\text{Es gilt } L(M) = A \cap B$$

3. Kapitel Kontextfreie Sprachen

für alle Regeln außer λ -Regeln gilt $|p| \leq |q|$

$$R \subseteq N \times (\Sigma \times N)^*$$

Bsp:

$$L = \{a^n b^n | n \geq 1\}$$

$$G = (\{a, b\}, S, S, \{S \rightarrow ab | \rightarrow aSb\})$$

Menge aller Palindrome

$$L = \{w \in \{a, b\}^* | w = w^R\}$$

$$G = (\{a, b\}, \{S, X\}, S, R)$$

$$R = \{S \rightarrow \lambda | X, X \rightarrow aXa | bXb | aa | bb | a | b\}$$

Satz 3.0.1

$$REG \subsetneq CF$$

Beobachtung: $L = \{a^n b^n | n \geq 1\}$ ist nicht regulär aber kontextfrei

3.1 Die Chomsky-Normalform

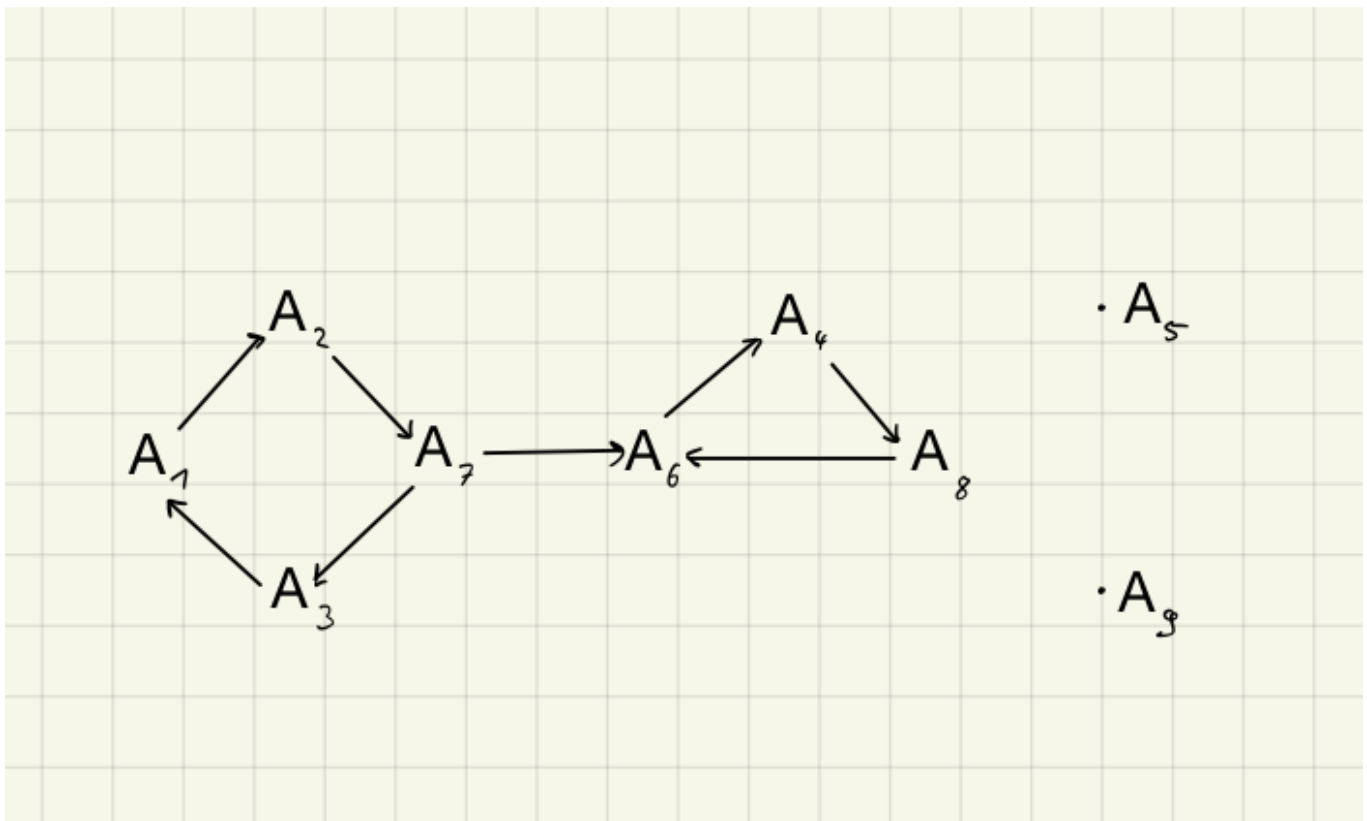
Definition 3.1.1

Eine Regel der Form $A \Rightarrow B; A, B \in N$ heißt Kettenregel

Satz 3.1.2

Zu jeder Kontextfreien Grammatik gibt es eine äquivalente Kontext freie Grammatik (K.f.G) G' ohne Kettenregel

Beweis: Sei $G = (\Sigma, N, S, R)$ eine k.f.G. Die Kettenregel von G definieren wir auf einen Gerichteten Graphen mit der Knotenmenge N .



1. Entfernen aller Zyklen

$$A_{i_1} \rightarrow A_{i_2}, A_{i_2} \rightarrow A_{i_3}, \dots, A_{i_{k-1}} \rightarrow A_{i_k}, A_{i_k} \rightarrow A_{i_1}$$

$$A_{i,j} \in N$$

In dem alle diese Regeln aus R entfernt werden und in den verbleibenden Regeln $A_{i,j}$ $1 \leq j \leq k$ durch A_i ersetzt wurden. Wir nummerieren die verbleibenden Nichtterminale so $\{B_1, \dots, B_k\}$, dass $B_i \rightarrow B_j$ stets $i < j$ folgt. Für $i = l-1, l-2, \dots$ ersetzen wir $B_i \rightarrow B_j$ $i < j$ so sind B_i als Regeln linke Seite $B_j \rightarrow a_1|a_2|\dots|a_n$ so entferne $B_i \rightarrow B_j$ und füge $B_i \rightarrow a_1|a_2|\dots|a_n$ hinzu.

Definition 3.1.2

Eine K.f.G $G = (\Sigma, N, S, R)$ heißt Grammatik in Chomsky Normalform, wenn jede Regel aus R folgende Form hat:

$$A \rightarrow BC \quad A, B, C \in N \quad A \neq B, A \neq C$$

$$A \rightarrow a \quad A \in N, a \in \Sigma$$

$S \rightarrow \lambda$ in diesem Fall darf S in keinem Fall auf der rechten Seite vorkommen

Satz 3.1.3

Jede Kontextfreie Sprache kann durch eine Grammatik in Chomsky Normalform erzeugt werden.

Beobachtung: Sei $A \subseteq CF$ gemäß 3.1.2 gibt es ein K.f.G. $G = (\Sigma, N, S, R)$ ohne Kettenregel mit $\mathcal{L}(G) = A$. Wir formen G in G' eine Chomsky-NF um.

1. Für jedes $a \in \Sigma$ führen wir ein neues Nichtterminal C_a ein $\{C_a | a \in \Sigma\} \cap N = \emptyset$
2. In jeder Regel wird a durch C_a ersetzt
3. Alle Regeln $C_a \rightarrow a$; $a \in \Sigma$ werden eingefügt/hinzugefügt
4. Nun kann es Regeln der Form $A \rightarrow B_1 B_2$
 B_m ; $m > 2$ geben

Für jede solcher Regeln führen wir $m - 1$ neue Nichtterminale D_1, D_2, \dots, D_{m-1} ein

Wir ersetzen $A \rightarrow B_1 D_1, D_1 \rightarrow B_2 D_2, D_2 \rightarrow B_3 D_3, \dots, D_{m-2} \rightarrow B_{m-1} D_{m-1}, D_{m-1} \rightarrow B_m$

Die nun entstehende Grammatik heie G' . Sie ist in Chomsky-NF und es gilt $\mathcal{L}(G') = \mathcal{L}(G)$

3.2 Pumping Lemma für Kontextfreie Sprachen

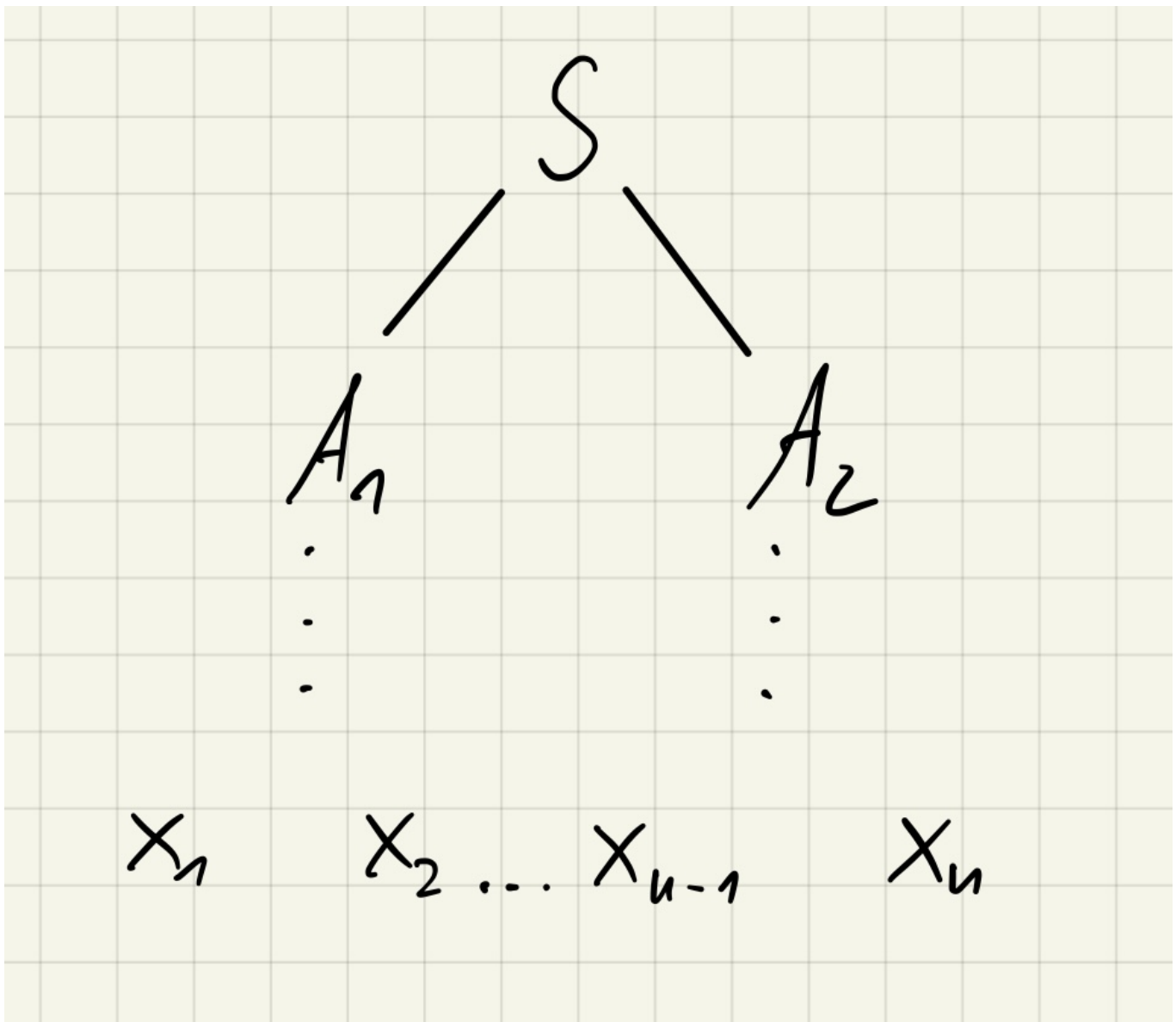
Satz 3.2.1

Sei $L \subseteq \Sigma^*$ eine Kontextfreie Sprache, dann gibt es eine Zahl in $n \in \mathbb{N}$, so dass sich alle $x \in L$ mit $|x| \geq n$ derart in $x = uv_1\tilde{v}v_2w$ das folgende Bedingungen erfüllt sind:

1. $|v_1 v_2| \geq 1$
2. $|v_1 \tilde{v} v_2| \leq n$
3. Für alle $i \geq 0$ gilt $uv_1^i \tilde{v} v_2^i w \in L$

Beweis: Sei $L \subseteq \Sigma^*, L \in CF$

Sei $G = (\Sigma, N, S, R)$ eine Grammatik in Chomsky-NF. Wir wählen $n = 2^{|N|}$. Sei $x \in L$ mit $|x| \geq n$. Der Syntaxbaum von x sieht so aus:



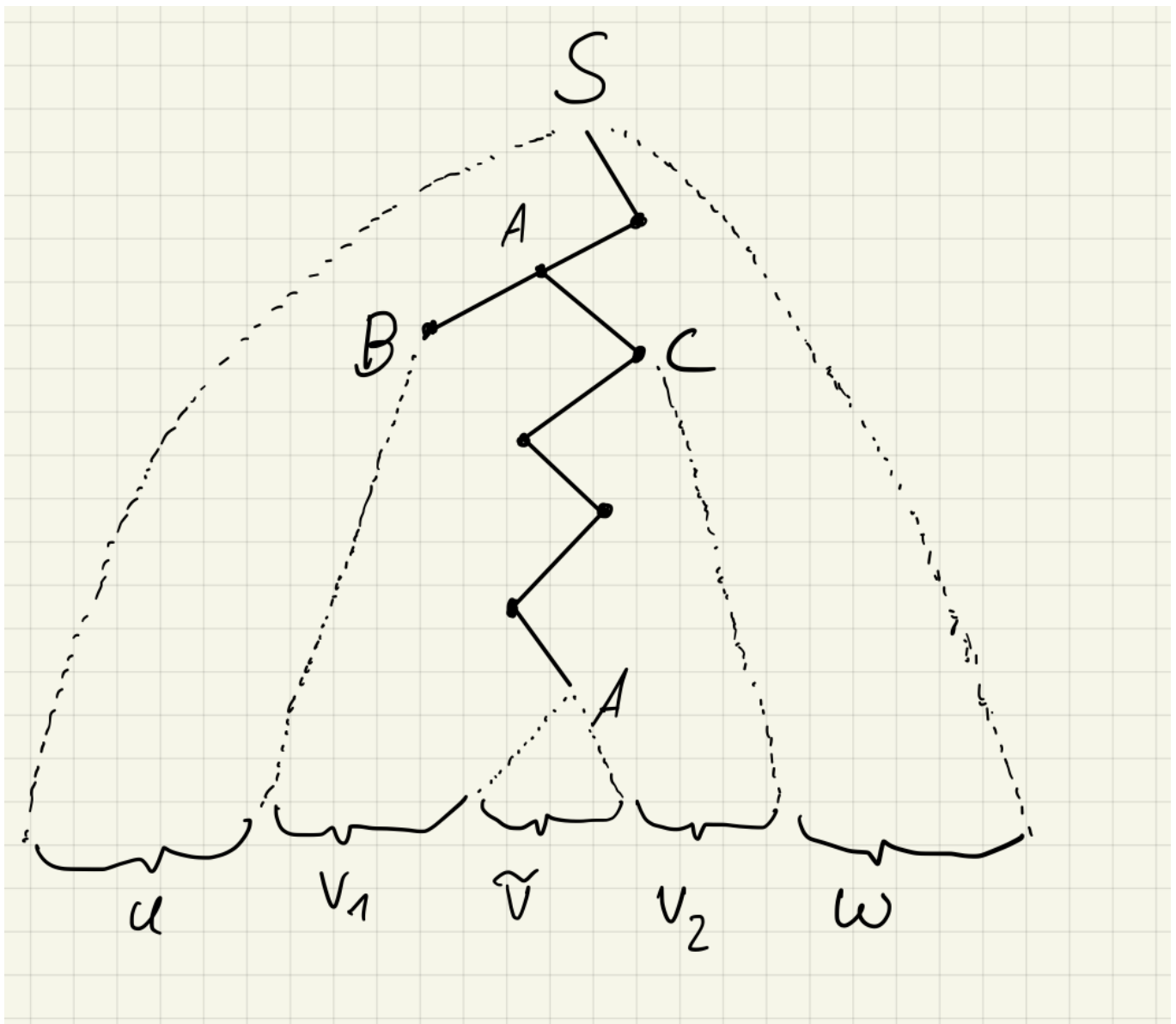
Dieser Baum hat $|x| \geq 2^{|N|}$ Blätter. Folglich muss es einen Pfad geben der mindestens Länge $|N|$ hat.

Lemma 3.2.2

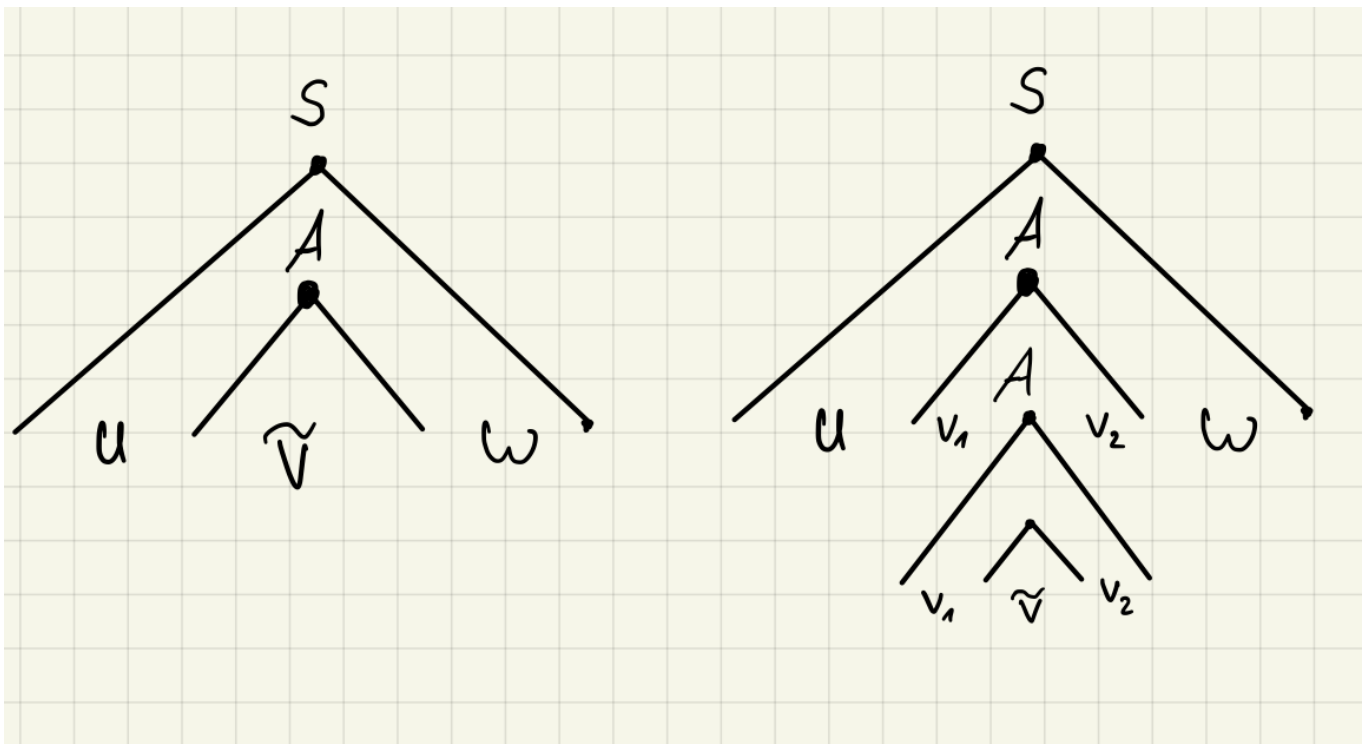
Jeder Binärbaum mit mindestens 2^k Blättern enthält einen Pfad der Länge $\geq k$.

Beobachtung: $k = 0$ Ein Baum mit $2^0 = 1$ Blättern hat einen Pfad der Länge 0.
 $K \rightarrow k + 1$.

Sei ein Binärbaum mit 2^{k+1} Blättern. Mindestens ein Teilbaum der Wurzel hat mindestens 2^k . In ihm gibt es nach IV einen Pfad der Länge k . Der wird zur Wurzel um eins verlängert. Wir fixieren einen Pfad p der maximale Länge. p hat mindestens $|N|$ Kanten und $|N| + 1$ Knoten \rightarrow auf p kommt ein Nichtterminal doppelt vor:



Wir fixieren ein solches Nichtterminal in dem wir auf p von unten nach oben, nach dem ersten Nichtterminal suchen das doppelt vorkommt. \rightarrow Das obere Vorkommen von A ist höchstens $|N|$ Kanten von der Blattebenen entfernt. Wir betrachten nun die Teilwörter, die aus den A 's abgeleitet werden können. Das gibt uns eine Zerlegung von x . Das oben A höchstens $|N|$ Kanten von der Blattebene entfernt ist, folgt $v_1 \tilde{v} v_2 \leq 2^m = n$ Aufgrund des doppelt vorkommens von A 's kann der Ableitungsbaum so modifiziert werden.



Das ergibt Ableitungsbäume für $uv_1\tilde{v}v_2w$ $i = 0, 2$ Analog erhält man Ableitungsbäume für $uv_1^i\tilde{v}v_2^i w$ für $i \geq 0$

$$1. L = \{a^n b^n c^n | n \geq 1\}$$

Annahme: $L \in CF$

Sei m die Pumpingzahl wir wählen $x = a^m b^m c^m$, $|x| = 3m \geq m$. Sei $x = uv_1\tilde{v}v_2w$ eine geeignete Zerlegung gemäß PL, das heißt $v_1v_2 \geq 1$

$$|uv_1v_2| \leq m$$

$$\underbrace{aaa \dots a}_{u} \underbrace{bb}_{v_1} \underbrace{b}_{\tilde{v}} \underbrace{b \dots b}_{v_2} \underbrace{bbccc \dots c}_{w}$$

Wegen $|v_2\tilde{v}v_2| \leq m$, kann v_1v_2 nicht a 's, b 's und c 's enthalten wegen $|v_1v_2| \geq 1$ ist und $v_1v_2 \neq \lambda$. Damit ist $uv_1\tilde{v}v_2w \notin L$, da $u\tilde{v}w$ nicht gleich viele a 's, b 's und c 's enthält (Widerspruch). $\rightarrow L \notin CF$

$$2. L = \{0^{2^n} | n \geq 1\} \notin CF$$

Annahme: $L \in CF$

Sei m die PZ (Pumpingzahl). Wähle $x = 0^{m^2}$, Sei $x = uv_1\tilde{v}v_2w$ eine geeignete Zerlegung gemäß PL.

$$\rightarrow |v_1 \tilde{v} v_2| \leq m, |v_1 v_2| \geq 1$$

$$\rightarrow v_1 v_2 = 0^k \quad 1 \leq k \leq m$$

$$m^2 \leq |uv_1^2 \tilde{v} v_2^2 w| = |uv_1 \tilde{v} v_2 w| + |v_1 v_2| = m^2 + |v_1 v_2|$$

$$m^2 + |v_1 v_2| \leq m^2 + m < (m+1)^2$$

$$\rightarrow uv_1 \tilde{v} v_2 w \notin L \text{ (Widerspruch)}$$

$$\rightarrow L \notin CF$$

3.3 Abschlusseigenschaften

Satz 3.3.1

CF ist eine abgeschlossen bezüglich

1. Vereinigung
2. Produkt
3. Kleene-Abschluss (Stern *)

CF ist noch nicht abgeschlossen bezüglich

4. Durchschnitt
5. Komplement
6. Differenz

Beweis: Seien $A, B \in CF$ und

$$G_1 = (\Sigma, N_1, S_1, R_1)$$

$$G_2 = (\Sigma, N_2, S_2, R_2) \text{ Kontextfrei}$$

Grammatiken mit $N_1 \cap N_2 = \emptyset$ und $\mathcal{L}(G_1) = A$ und $\mathcal{L}(G_2) = B$

$$1. \mathcal{L}((\Sigma, N_1 \cup N_2 \cup \{S\}, S, R_1 \cup R_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\})) = A \cup B$$

$$2. \mathcal{L}((\Sigma, N_1 \cup N_2 \cup \{S\}, S, R_1 \cup R_2 \cup \{S \rightarrow S_1 S_2\})) = A \cdot B$$

$$3. \mathcal{L}((\Sigma, N_1 \cup \{S\}, S, R_1 \cup R_2 \cup \{S \rightarrow S_1, S \rightarrow \lambda, S \rightarrow S_1, S \rightarrow S_1 S_1\} \setminus \{S_1 \rightarrow \lambda\})) = A \cup B$$

4. Wir betrachten zwei Sprachen:

$$L_1 = \{a^i b^j c^j \mid i, j > 0\}$$

$$L_2 = \{a^i b^i c^j \mid i, j > 0\}$$

$L_1, L_2 \in CF$, denn ... ÜA

$$\text{aber } L_1 \cap L_2 = \{a^n b^n c^n \mid n > 0\} \notin CF$$

5. Wenn CF bezüglich Komplement abgeschlossen. So müsste CF wegen
 $A \cap B = \neg(\neg A \cup \neg B)$

$$6. A \cap B = A \cup B \setminus ((A \setminus B) \cup (B \setminus A))$$

3.4 Nicht deterministischer Kellerautomat(Push-Down-Automat)

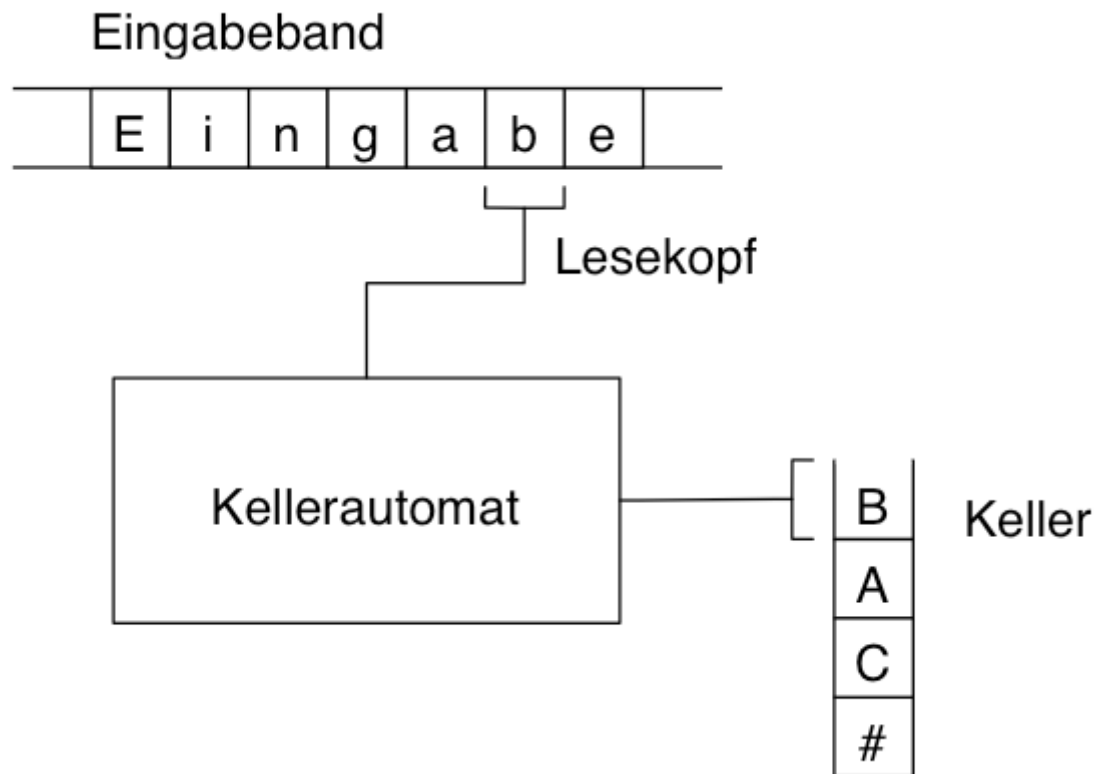
$$\{a^n b^n \mid n \geq 0\} \in CF \downarrow \text{Zähler wird gebraucht} \downarrow \text{Stack}$$

Definition 3.4.1

Ein nicht deterministischer Kellerautomat $M(PDA)$ ist ein 6-Tupel

$$M = \{\Sigma, \Gamma, Z, \delta, z_0, z_E\}$$

1. Σ ist eine endliche Menge (Eingabealphabet)
2. Γ ist eine endliche Menge (Kelleralphabet)
3. Z ist eine endliche Menge (Zustandsmenge)
4. $\delta : Z \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow \gamma_{endl}(Z \times \Gamma^*)$
5. $z_0 \in Z$ (Startzustand)
6. $z_E \in Z$ (Endzustand)



(Source: <https://upload.wikimedia.org/wikipedia/commons/4/42/Kellerautomat.png>
[10.01.2023])

Eingabeband:

- nur lesen
- jedes Symbol kann nur einmal gelesen werden
- am Anfang steht hier das Eingabewort

Keller:

- lesen und schreiben
- Topelement des Stacks wird gelesen und durch ein Wort $w \in \Gamma^*$ ersetzt
- neues Topelement ist der erste Buchstabe von w

Bei Start des Automaten ist der Keller leer (\square)

Bemerkungen:

1. $\delta(z, a, A) = \{(1, B_1, B_2), (z_2, \lambda), (z_3, BA)\}$
 $(z_1, B - 1 \dots B_k) \rightarrow A$ wird durch $B_1 B_2 \dots B_k$ ersetzt

- $(z_1, \lambda) \rightarrow A$ wird entfernt (pop)
- $(z_1, BA) \rightarrow$ neues Symbol B wird aufgesetzt (push)
- 2. $(\delta(z, \lambda, A) = \{(z_1, B_1 \dots B_k), (z_2, \lambda), (z_3, BA)\})$
 \downarrow ist auch erlaubt
 \rightarrow Veränderung ohne das ein Symbol der Eingabe gelesen wird
- 3. nur ein Symbolzustand ist keine Einschränkung
 $\rightarrow \delta(z_0, \lambda, \square) = \{(z'_0, \square), (z''_0, \square), (z'''_0, \square)\}$
 \rightarrow könnte man nutzen, um mit mehreren Startzuständen zu operieren

Die von einem PDA akzeptierte Sprache ist die Menge aller Wörter, bei deren Eingabe der PDA in einem Zustand z_E enden kann.

Definition 3.4.2

Sei $M = \{\Sigma, \Gamma, Z, \delta, z_0, z_E\}$ ein PDA

Eine Konfiguration K von M ist ein Tripel $K \in Z \times \Sigma^* \Gamma^*$

Dabei ist für $K = (z, u, v)$:

- $z \dots$ der aktuelle Zustand
- $u \dots$ der noch nicht gelesene Teil der Eingabe
- $v \dots$ der aktuelle Kellerinhalt

\rightarrow Anfangskonfiguration: $(z_0, w, \square), w \in \Sigma^*$

$\rightarrow (z_1, \lambda, p), z_1 \in z_E, p \in \Gamma^*$

Definition 3.4.3

Wir definieren auf $Z \times \Sigma^* \times \Gamma^*$

(Menge der Konfigurationen) eine binäre Relation (Nachfolgekongfiguration) wie folgt:

$(z, u, v) \Vdash_M (z', u', v') \leftrightarrow_{df} (*)$

$$(*) \quad \bigvee_{x \in \Sigma \cup \{\lambda\}} \bigvee_{y \in \Gamma \cup \{\lambda\}} \bigvee_{y' \in \Gamma^*} \bigvee_{r \in \Gamma^*} u = xu' \wedge v = y \cdot r \wedge v' = y'r \wedge (z', y') \in \delta(z, x, y)$$

\rightarrow Analog zu \vdash_G^* definiert man \vdash_M^\times als die reflexive und transitive Hülle von \vdash_M

$\rightarrow K \vdash_M^* K' \leftrightarrow_{df}$ es gibt endliche Konfigurationen K_1, \dots, K_n mit $K \vdash_M K_1,$

$K_1 \vdash_M K_2, \dots, K_{n-1} \vdash_M K_n$ und $K_n \vdash_M K'$

Definition 3.4.4

Die von einem PDA akzeptierte Sprache ist definiert als:

$$L(M) = \{w \in \Sigma^* \mid \bigvee_{z \in z_E} \dots \bigvee_{p \in \Gamma^*} ((z_0, w, \square) \vdash_M^* (z, \lambda, p))\}$$

Beispiel

1. $L = \{a^n b^n \mid n \geq 1\}$

$$M = (\{a, b\}, \{\square, A\}, \{z_0, z_1, z_{\text{ende}}\}, \delta, z_0, \{z_{\text{ende}}\})$$

$$\delta) (z_0, a, \square) \rightarrow (z_0, A\square)$$

$$(z_0, a, A) \rightarrow (z_0, AA)$$

$$(z_0, b, A) \rightarrow (z_1, \lambda)$$

$$(z_1, b, A) \rightarrow (z_1, \lambda)$$

$$(z_1, \lambda, \square) \rightarrow (z_{\text{ende}}, \square)$$

2. $L = \{w\$w^R \mid w \in \{a, b\}^*\}$

$$M = (\{a, b, \$\}, \{A, B, \square\}, \delta, z_0, \{z_{\text{ende}}\})$$

$$\delta)$$

$$(z_0, a, A) \rightarrow (z_0, A\square)$$

$$(z_0, a, A) \rightarrow (z_0, AA)$$

$$(z_0, a, B) \rightarrow (z_0, AB)$$

$$(z_0, b, \square) \rightarrow (z_0, B\square)$$

$$(z_0, b, A) \rightarrow (z_0, BA)$$

$$(z_0, b, B) \rightarrow (z_0, BB)$$

$$(z_0, \$, \square) \rightarrow (z_1, \square)$$

$$(z_0, \$, A) \rightarrow (z_1, A)$$

$$(z_0, \$, B) \rightarrow (z_1, B)$$

$$(z_1, a, A) \rightarrow (z_1, \lambda)$$

$$(z_1, b, B) \rightarrow (z_1, \lambda)$$

$$(z_1, \lambda, \square) \rightarrow (z_{\text{ende}}, \square)$$

Definition 3.4.5

Sei $M = \{\Sigma, \Gamma, Z, \delta, z_0, z_E\}$ ein PDA

Wir sagen M akzeptiert mit leerem Keller, wenn $\$$

A=

$\mathscr{L}(M) =$

{

$w \in \Sigma^*$

$\bigvee \{z \in Z\}$

(z_0, w, \Box)

\vdash_M

(z, λ, λ)

}

Bemerkung: auf z_{ende} kann verzichtet werden ##### Satz

$\bigvee \{u \in \Sigma^*\}$

$\bigvee \{A \in N\}$

$\bigvee \{q \in (\Sigma \cup N)^*\}$

$\bigvee \{v \in N^*\}$

$(p_i = uAv, p_{i+1} = uqv, (A \rightarrow q) \in R)$

oder

(

$p_1 = s_1, p_2 = \lambda, (s \rightarrow \lambda) \in R$

)

Behauptung: Sei $G = (\Sigma, N, S, R)$ mit $\mathscr{L}(G) \in CF$ $w \in \mathscr{L}(G)$ gdw. 1

A=

$\Big\{$

$w \in \Sigma^*$

$\bigvee \{z \in Z\}$

$\Big($

(z_0, w, \Box)

\vdash_M

(z, λ, λ)

$\Big)$

$\Big\}$

Diesen gibt es gemäß Satz 3.4.6 Wir nehmen an, dass für alle $z \in Z, a \in \Sigma, \gamma \in \Gamma$

$$|\delta(z,a,A)| + |\delta(z,\lambda,A)| \leq 1$$

Eine Sprache heißt deterministisch kontextfrei, falls es einen DPDA M gibt mit $\mathscr{L}(M) = L$

$\underset{w \in \Sigma^*}{\bigwedge}$

$\underset{z \in Z}{\bigvee}$

$\underset{a \in \Gamma}{\bigvee}$

(z_0, w, \Box)

\vdash_M

(z, λ, α)

Beweisidee: Sei $M = (\Sigma, \Gamma, Z, \delta, z_0, z_E)$ Ein DPDA mit $\mathscr{L}(M) = L$

$\vdash_M \begin{cases}$

$u \sim \alpha \sim \gamma \sim z' \sim v \ \& \ , \text{ falls } z \xrightarrow{\beta} z', \gamma, R \setminus$

$u \sim z' \sim \alpha \sim \gamma \sim v \ \& \ , \text{ falls } z \xrightarrow{\beta} z', \gamma, L \setminus$

$u \sim \alpha \sim z' \sim \gamma \sim v \ \& \ , \text{ falls } z \xrightarrow{\beta} z', \gamma, N \setminus$

\end{cases}

$\alpha \sim z \sim \beta \in \Gamma^* z \Gamma^* \implies \alpha \sim z \sim \beta \vdash_M \alpha' \sim z' \sim \beta' \implies \alpha' \sim z' \sim \beta' \in L$

$\mathscr{L}(M) = \{$

$w \in \Sigma^*$

$|$

es gibt eine akzeptierende Berechnung von M für w

$\}$

Bemerkung $w \notin L(M)$ gdw. alle Berechnungen sind verworfen

$\mathscr{L}(G) = L_1 \cup L_2$ und G kontextsensitiv

2. G_1, G_2 Normalformalgrammatiken

→ auf linken Seiten nur Nichtterminale, für jedes $a \in \Sigma$ $A_a \rightarrow a$ und in allen Regeln a durch A_a ersetzt

Bsp:

$S \rightarrow S_1 S_2 \quad G = (\Sigma, N_1 \cup N_2 \cup S, S, R)$

$AB \rightarrow Ba$

$R = R_1 \cup R_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\} \cup \{S \rightarrow \lambda\} \setminus \{S_1 \rightarrow \lambda, S_2 \rightarrow \lambda\}$ falls $\lambda \in L_1$ oder $\lambda \in L_2$

$$R' = \begin{cases} \{S \rightarrow S_1, S \rightarrow S_2, S \rightarrow \lambda\} & \text{falls } \lambda \in L_1 \text{ und } \lambda \in L_2 \\ \{S \rightarrow S_2\} & \text{falls } \lambda \in L_1 \text{ und } \lambda \notin L_2 \\ \{S \rightarrow S_1\} & \text{falls } \lambda \notin L_1 \text{ und } \lambda \in L_2 \\ \emptyset & \text{falls } \lambda \notin L_1 \text{ und } \lambda \notin L_2 \end{cases}$$

(*) sonst könnte es in $R_1 \cup R_2$ Regeln $\alpha \rightarrow \beta$ geben, die auf ein Teilwort α von $\gamma\delta$ mit $S_1 \vdash_{G_1} \gamma$ $S_2 \vdash_{G_2} \delta$ anwendbar sind, wobei α die Grenze zwischen γ und δ überragt

$\mathcal{L}(G) = L_1 L_2$ und G ist kontextsensitiv

$G = (\Sigma, N, S, R)$ $\mathcal{L}(G) = L$

$G' = (\Sigma, N \cup \{S'\}, S', R \cup \{S' \rightarrow \lambda, S' \rightarrow SS'\})$

$\mathcal{L}(G') = L^*$, aber G' ist nicht kontextsensitiv (S' kommt rechts vor)

$\rightarrow G' = (\Sigma, N \cup \{S', S''\}, S'', R \cup \{S'' \rightarrow \lambda, S'' \rightarrow S', S' \rightarrow S, S' \rightarrow SS'\} \setminus \{S \rightarrow \lambda\})$

3. ÜBUNG MUSS NACH GEHOLT WERDEN!!

4. $A \cap B = \overline{\overline{A} \cup \overline{B}}$

5. Kapitel Berechenbarkeit

5.1 Intuitiver Begriff der Berechnbarkeit

Was ist Berechnbar? $\rightarrow f(n, m) = m + n$

$\rightarrow 99!$

$\rightarrow f(n) = n!$

$\rightarrow g(n) = \text{Anzahl der Primfaktoren von } 2^{67 \cdot n} - 1$

$\rightarrow h(n, m) = ggT(n, m)$

Eine Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ heißt im intuitiven Sinne berechenbar, wenn es einen Algorithmus gibt, der zu beliebigen vorgegebenen Argumenten (x_1, \dots, x_k) aus dem Definitionsbereich von f nach endlich vielen Schritten den Funktionswert $f(x_1, \dots, x_k)$ liefert

Bsp: $f(x) = x + 1, f(x, y) = x + y$

$f(x_1, \dots, x_n) = (x_1, \dots, x_{in})$, wobei x_{i1}, \dots, x_{in} in paarweise verschiedenen und $x_{i1} \leq x_{i2} \leq \dots \leq x_{in}$

5.2 Grundlagen

$$\mathbb{N} = \{0, 1, 2, 3, \dots\}$$

$$\mathbb{N}^{\mathbb{N}} = \{f : f : \mathbb{N} \rightarrow \mathbb{N}\} \text{ "von } \mathbb{N} \text{ nach } \mathbb{N}", D_f = \mathbb{N}$$

$$\tilde{\mathbb{F}}_1 \mathbb{N}^{\mathbb{N}} \mathbb{F}_1 = \{f | f : \mathbb{N} \rightarrow \mathbb{N}\} \text{ "aus } \mathbb{N} \text{ in } \mathbb{N}", D_f \subseteq \mathbb{N}$$

$$\tilde{\mathbb{F}}_2 \mathbb{N}^{\mathbb{N} \times \mathbb{N}} \mathbb{F}_2 = \{f | f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}\}$$

$$\rightarrow \text{Menge der totalen beliebigstelligen Fkt. über } \mathbb{N}: \tilde{\mathbb{F}} = \bigcup_{i=1}^{\infty} \tilde{\mathbb{F}}_i$$

$$\rightarrow \text{Menge der beliebigstelligen Funktion über } \mathbb{N}: \tilde{\mathbb{F}} = \bigcup_{i=1}^{\infty} \tilde{\mathbb{F}}_i$$

Satz 5.2.1

Es gibt Funktionen(bereits in $\tilde{\mathbb{F}}_1$) die noch nicht berechenbar sind

Beweis: Behauptung 1: Es gibt überabzählbar viele Funktionen (in $\tilde{\mathbb{F}}_1$)

Beweis Diagonalisierung

Annahme: Es gibt nur abzählbar viele Funktionen: f_0, f_1, \dots

$$f_1 \quad f_1(0) \quad f_1(1) \quad f_1(2) \quad f_1(3) \quad f_1(3) \dots$$

$$f_2 \quad f_2(0) \quad f_2(1) \quad f_2(2) \quad f_2(3) \quad f_2(3) \dots$$

$$f_3 \quad f_3(0) \quad f_3(1) \quad f_3(2) \quad f_3(3) \quad f_3(3) \dots$$

$$f_4 \quad f_4(0) \quad f_4(1) \quad f_4(2) \quad f_4(3) \quad f_4(3) \dots$$

\vdots

Wir konstruieren die Funktion g : seien $i, j \in \mathbb{N} \quad i \neq j$

$$g(k) := \begin{cases} i & \text{falls } f_k(k) \neq i \\ j & \text{falls } f_k(k) = i \end{cases} \quad \text{für alle } k \in \mathbb{N}$$

$\rightarrow g$ ist somit von allen Funktion f_i verschieden und kommt in der Folge f_0, f_1, f_2, \dots nicht vor

\rightarrow Das ist ein Widerspruch zu der Annahme, dass die Folge alle Funktionen enthält

Behauptung 2: Die Menge der Berechnbaren Funktion abzählbar

Beweis - jeder Algorithmus repräsentiert eine berechnbare Funktion

\rightarrow es gibt höchstens so viele berechnbare Funktionen wie Algorithmen

- jeder Algorithmus ist als Wort über einen geeigneten Alphabet A darstellbar, da A^* abzählbar ist, ist die Menge der Algorithmen abzählbar und somit auch die Menge der berechnabren Funktionen

FAZIT: Das, was ein Computer kann, ist im Vergleich zu dem was er nicht kann, vernachlässigbar!

- Wie sehen sie denn nur aus, die nicht berechenbaren Funktionen?

→ Versuche: $f_1(n) = \begin{cases}$

1 & falls ~Anfangsabschnitt~der~Dezimalbruchentwicklung~von~ π ~ist~\

0 & sonst

\end{cases}

$$f_1(314) = 1, f_1(3) = 1, f_1(5) = 0$$

$f_2(n) = \begin{cases}$

1 & falls ~n~irgendwo~in~der~Dezimalbruchentwicklung~von~ π ~\

0 & sonst

\end{cases}

$$f_3(n) = \begin{cases} 1 & \text{irgendwo in der Dezimal... von } \pi \text{ gibt es } n \text{ mal hintereinander eine 7} \\ 0 & \text{sonst} \end{cases}$$

5.3 Turing Berechnbarkeit

bisher: Turingmaschine akzeptiert Sprachen

→ Wir wollen den Begriff modifizieren, um das Berechnen von Funktionen zu erfassen

→ Für Berechnbarkeit nutzen wir deterministische Turingmaschinen

→ δ ... partielle Funktionen

$$\delta : Z \times \Gamma \rightarrow Z \times \Gamma \times \{L, R, N\}$$

Vereinbarung: $\alpha z \beta :=$ für alle $a, \beta \in \Gamma^*$ (TM hält sobald ein Endzustand erreicht ist)

	NTM	DTM
Berechnung:	-maximale(nicht verlängerbare) Folge von Konfigurationen - viele Berechnungen für ein Wort möglich	$-z_0 w \vdash \alpha_1 z_1 \beta_1 \vdash \alpha_2 z_2 \beta_2$ genau eine Berechnung

	NTM	DTM
Art der Berechnung:	-TM hält an: -akzeptierend...endet in Konfiguration mit Endzustand verwerfend...endet in $\alpha_n z_n \beta_n$ $z_n \notin Z_E, \delta(z_n, b) = \emptyset$ für b 1. Zeichen von β_n -sonst unendlich	-TM hält an: -akzeptierend...endet in Konfiguration mit Endzustand verwerfend...endet in $\alpha_n z_n \beta_n$ $z_n \notin Z_E, \delta(z_n, b) = \emptyset$ für b 1. Zeichen von β_n -sonst unendlich
M akzept. Wort $w \in \Sigma^*$	-es gibt eine akzeptierende Berechnung von M für w mit $\bigvee_{z \in Z_E} \bigvee_{\alpha, \beta \in \Gamma^*} z_0 w \vdash_M^* \alpha z \beta$	-Die Berechnung von M für w ist akzeptierend, d.h. sie endet in $z \in Z_E$
$w \notin L(M)$	-alle Berechnung von M für w sind verwerfend oder endlos	-Die Berechnung von M für w ist verwerfend oder endlos

Definition 5.3.1

Eine Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ heißt Turing berechnbar gdw.: Es gibt eine DTM $M = (\{0, 1, 2, \dots, 9, \#\}, \Gamma, Z, \delta, z_0, Z_E, \square)$, so dass für alle $(n_1, \dots, n_i) \in \mathbb{N}^i$ gilt

1) $(n_1, \dots, n_i) \in D_f$ gdw. $\bigvee_{z \in Z_E} \bigvee_{\alpha, \beta \in \Sigma^*} z_0 n_1 \# n_2 \# \dots \# n_i \vdash_M^* \alpha z \beta$

2) Für alle $(n_1, \dots, n_i) \in D_f$ gilt $f(n_1, \dots, n_i) = m$ gdw.

$\bigvee_{a \in \Gamma^*} \bigvee_{z \in Z_E} z_0 n_1 \# n_2 \# \dots \# n_i \vdash_M^* \alpha z m$

Definition 5.3.2

Eine Funktion $f : \Sigma^* \rightarrow \Delta^*$ heißt Turing berechenbar gdw.: Es gibt eine DTM $M = (\Sigma, \Gamma, Z, \delta, Z_0, Z_E, \square), \Delta \subseteq \Gamma \setminus \{\square\}$: so dass für alle $w \in \Sigma^*$ gilt:

1) $w \in D_f \leftrightarrow \bigvee_{z \in Z_E} \bigvee_{\alpha, \beta \in \Sigma^*} z_0 w \vdash_M^* \alpha z \beta$

2) $f(w) = y \leftrightarrow \bigvee_{z \in Z_E} \bigvee_{\alpha, \beta \in \Sigma^*} z_0 w \vdash_M^* \alpha z y$

Bsp 1

Nachfolgekonfiguration: $S : bin(n) \rightarrow b(n+1)$

$$M = (\{0, 1\}, \{0, 1, \square\}, \{z_0, z_1, z_2, z_E\}, \delta, z_0, \{z_E\}, \square)$$

δ	z_0	z_1	z_2
0	$(z_0, 0, R)$	$(z_1, 1, L)$	$(z_2, 0, L)$
1	$(z_0, 1, R)$	$(z_1, 1, R)$	$(z_2, 1, L)$
\square	(z_0, \square, L)	$(z_E, 1, N)$	(z_E, \square, N)

Bsp 2

nirgends definierte Funktion $f(w) = n \cdot d$

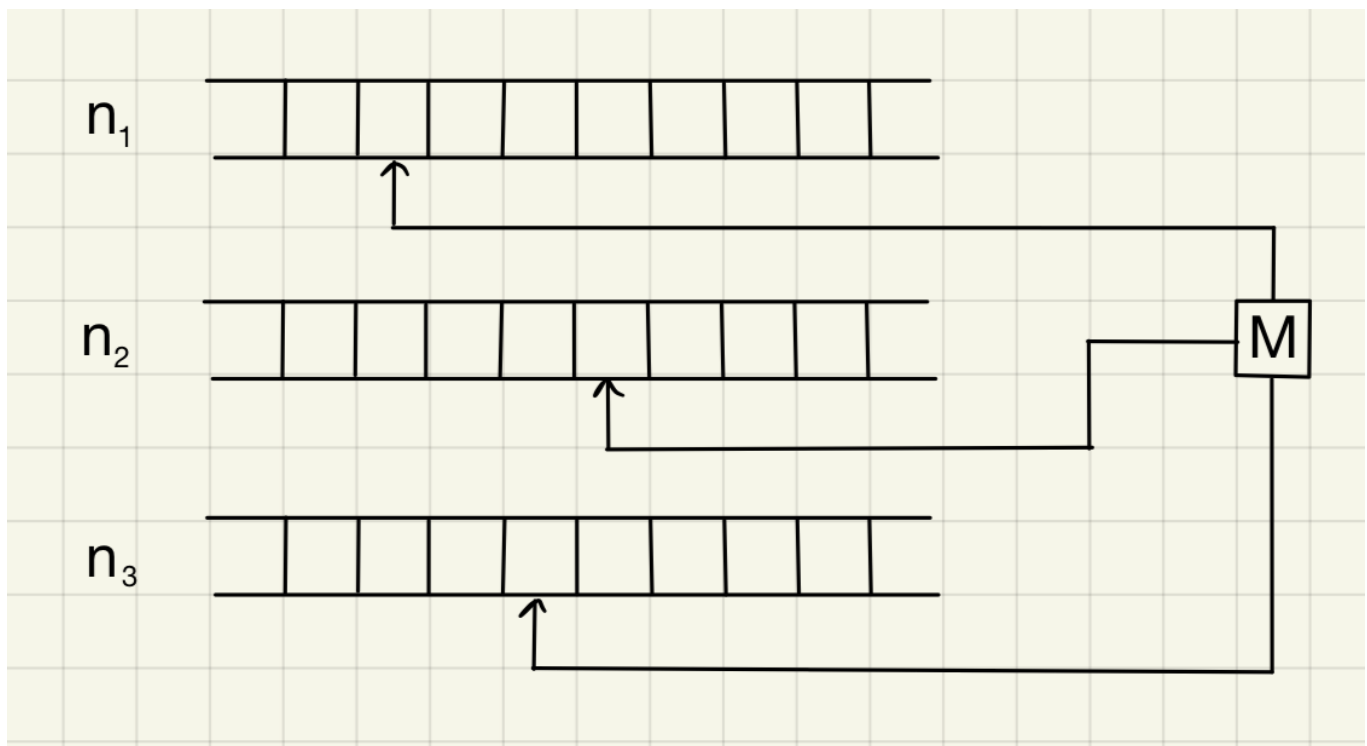
δ	z_0
a	(z_0, a, R)
b	(z_0, a, R)
\square	(z_0, a, R)

5.4 Andere Typen von Turingmaschinen

→ Der bisher eingeführte Typ wird oft als Standard TM bezeichnet

→ Man kann die Definition auf verschiedene Weisen erweitern

1. mehrbändige Turingmaschine:



$$\delta : Z \times \Gamma^k \rightarrow Z \times \Gamma \times \{L, R, N\}^k$$

k ... Anzahl der Bänder

Befehle haben die Form ($k = 3$)

$$zx_1x_2x_3 \rightarrow z'x'_1x'_2x'_3 \quad \sigma_1\sigma_2\sigma_3$$

x_i ... gelesenes Zeichen auf Band i

x'_i ... geschriebenes Zeichen auf Band i

σ_i ... Kopfbewegung auf Band i

→ die Eingabe steht auf Band 1, die anderen Bänder sind leer

→ nächster Schritt hängt vom aktuellen Zustand und den k -Bandinhalten ab und dem Lese-/Schreibkopf ab

→ jedes der k -Symbole kann überschrieben werden

→ jeder Kopf kann sich bewegen (unabhängig voneinander)

Konfiguration einer k -Band TM: $\langle \alpha_1, \alpha_2, \dots, \alpha_k \rangle z \langle \beta_1, \beta_2, \dots, \beta_k \rangle$

→ $k = 3$: - a's und b's auf Band 2 bzw Band 3 kopieren

- alle gleichzeitig vergleichen

$k = 2$: - a's auf Band 2 kopieren (als I)

- Anzahl b's und Anzahl I vergleichen

Band 1: von links nach rechts über b's

Band 2: von rechts nach links über I

- Anzahl c's mit Anzahl I vergleichen, auf Band 2 wieder von links nach rechts

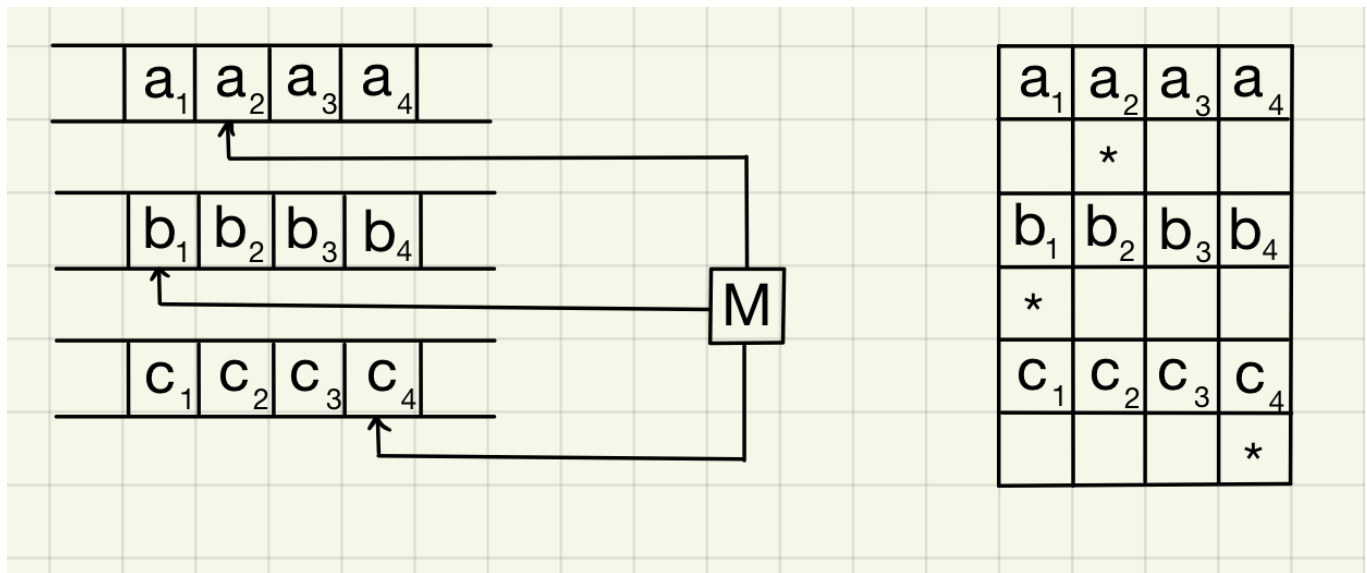
Satz 5.4.1

Zu jeder Mehrband TM M gibt es eine (Standard) TM M' mit $L(M) = L(M')$ (bzw. M' berechnet dieselbe Funktion wie M)

Beweis Idee: Sie k die Anzahl der Bänder von M .

Wir konstruieren: $M' = (\Sigma, \Gamma', Z', \delta', z'_0, Z'_E, \square)$

Das von M' ist in $2k$ Spuren unterteilt: $\Gamma' = \Gamma \cup (\Gamma \cup \{*\})^{2k}, * \notin \Gamma$



→ in jeder Zelle des Bandes steht ein 6-Tupel:

Spur $2i - 1 \dots$ Bandinhalt von Band i

Spur $2i \dots$ Kopfposition von Kopf i

→ Eingabe: $x_1, \dots, x_n \in \Sigma^*$

→ M' erzeugt Startkonfiguration in Spurendarstellung

	X_1	X_2	X_3	X_4	.	.	.	X_n	
	*	□	□						
□	□	□		□
	*								
	□	□	□		

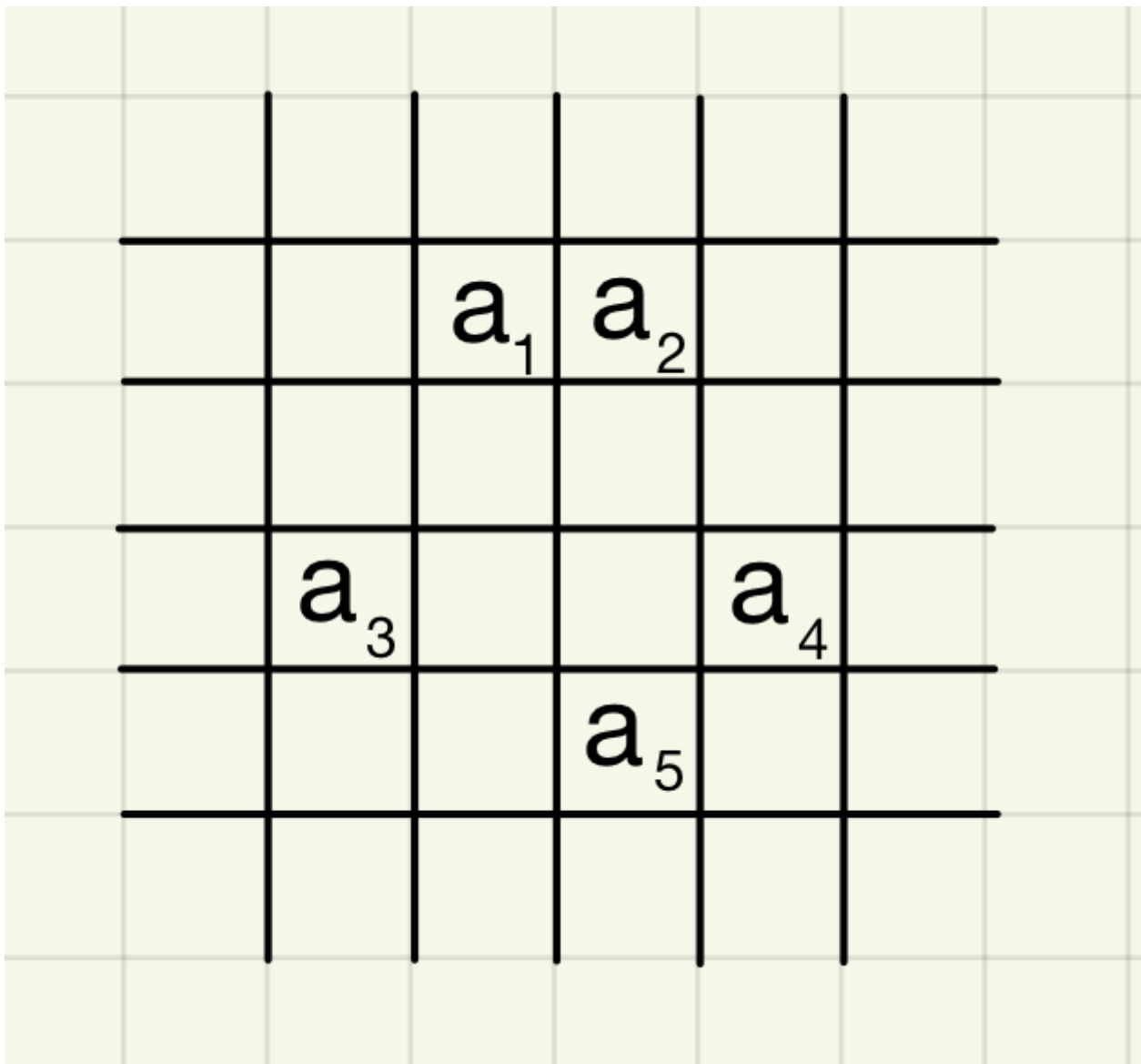
Simulation eines Arbeitsspeichers von M

- M' führt mehrere Schritte aus, um einen Schritt von M zu simulieren
- LS-Kopf steht zu Beginn links von allen $*$ -Markierungen
- M' läuft von links nach rechts, trifft sie auf eine $*$ -Zeile, merkt sie sich das gelesene Symbol (dazu werden (viele) neue Zustände benötigt)
- nach dem letzten $*$ sind alle zu lesenden Symbole bekannt $\rightarrow M'$ kann Überführungsregel aus δ anwenden
- auf dem Weg nach links über alle $*$ -Markierungen hinweg führt sie die entsprechenden Anweisungen aus

(2) Einige Bänder können reine Eingabebänder (read-only) oder reine Ausgabebänder (write-only) sein

(3) Mehrere Köpfe auf einem Band erlauben (Vorsichtig bei Kollisionen)

(4) Mehrdimensionale Bänder, z.B.: $d = 2$, $\delta : Z \times \Gamma \times B$ mit $B = \{N, S, W, O, H\}$



- auch hier ist eine Simulation durch eine (Standard) TM möglich
 - in jeder Zeile stehen endlich viele Symbole
 - der Bandinhalt kann durch ein Rechteck eingegrenzt werden
 - Zeile für Zeile auf eindimensionales Band schreiben
 - N - und S -Übergänge haben weiteren Weg auf eindimensionalem Band
- alle diese Typen können auf einer (Standard) TM simuliert werden
 - das funktioniert auch bei NTM
 - eine TM kann auf einer TM mit einseitig unendlichem Band simuliert werden

- es kann gezeigt werden, dass spezielle Programmierkonzepte (loop, while, goto-Berechnbarkeit) äquivalent zur Turing - Berechnbarkeit sind

5.5 Die Churchsche These

- Frage: Beschreibt der Begriff der Turing-Berechnbarkeit den intuitiven Begriff zufriedenstellend?
 - es gibt mehrere verschiedene Berechnungsmodelle
 - man kann zeigen, dass alle dieselbe Klasse von Funktionen beschreiben
 - alle anderen Berechnungsmodelle lassen sich durch TM's simulieren
 - ↓ dann gehen wir davon aus, dass es den intuitiven Begriff beschreibt (Beweis können wir es nicht)

These 5.5.1 Churchs These

→ Die durch die formale Definition der Turing-Berechenbarkeit erfasste Klasse von Funktionen stimmt mit der Klasse der im intuitiven Sinn berechnbaren Funktionen überein

5.6 Deterministische und nicht deterministische TM

DTM: $Z \times \Gamma \times \{L, R, N\}$ partielle

Satz 5.6.1

Jede von einer NTM akzeptierte Sprache kann auch von einer DTM akzeptiert werden

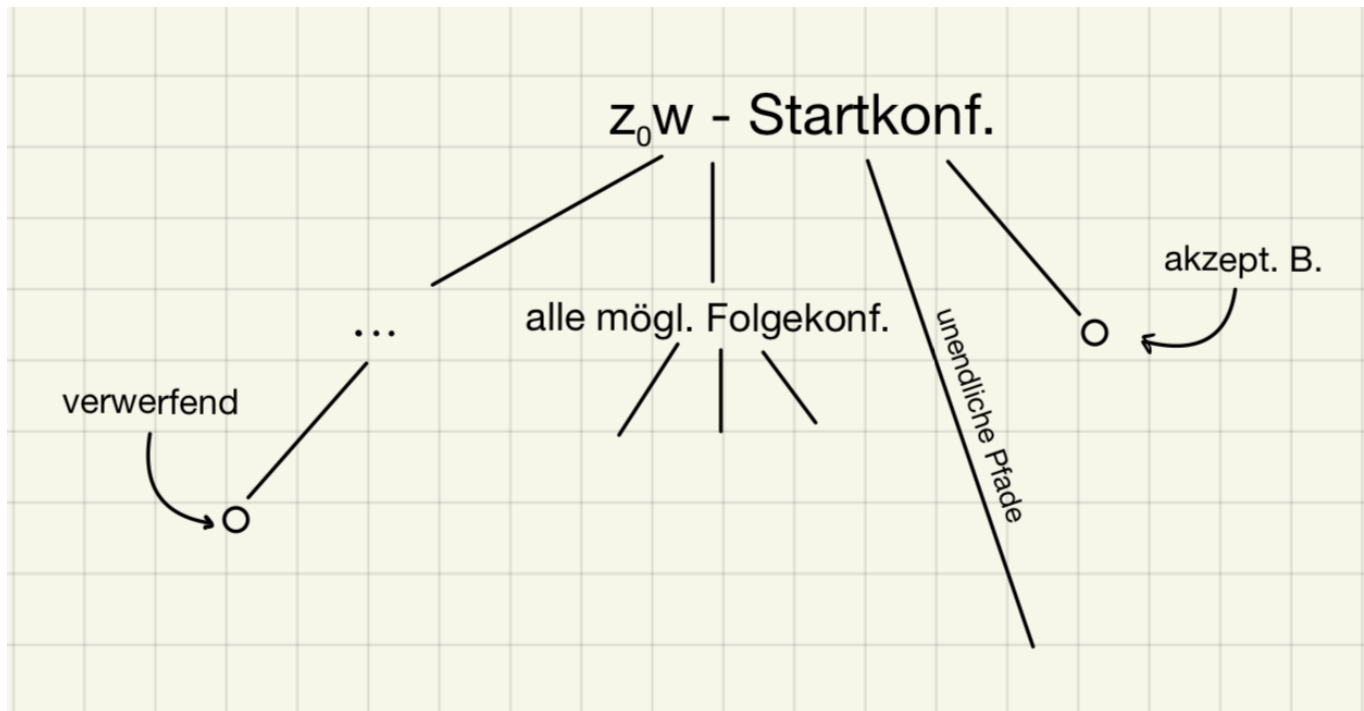
Berechnungsbaum einer NTM

Beweis

konstruiere eine DTM T' , die eine NTM T simuliert

- DTM durchläuft den Berechnungsbaum der NTM für w mit Breitensuche
- Gibt es eine akzeptierende Berechnung der NTM (akzeptierendes Blatt) wird die DTM sie finden
- DTM hält akzeptierend an

- gibt es keine akzeptierende Berechnung, verwirft die DTM (alle Blätter sind verwerfend)(weil es unendliche Pfade im Berechnungsbaum gibt)
 $\rightarrow \mathcal{L}(T) = \mathcal{L}(T')$



6. Kapitel Entscheidbarkeit und Aufzählbarkeit

6.1 Entscheidbarkeit

Funktionen	Mengen
Berechnbarkeit	Entscheidbarkeit, Aufzählbarkeit

\rightarrow Es sei $A \subseteq \Sigma^*$ die charakteristische Funktionen c_A von A ist wie folgt definiert:

$$c_A(w) = \begin{cases} 1 & w \in A \\ 0 & w \notin A \end{cases}$$

\rightarrow die Funktion ist total

Definition 6.1.1

Eine Menge $A \subseteq \Sigma^*$ heißt entscheidbar gdw. c_A berechenbar ist REC (recursive sets)

\rightarrow Menge aller entscheidbaren Mengen

Bemerkung:

- c_A ist Turingberechenbar bedeutet es gibt DTM, die für jedes $w \in \Sigma^*$ anhält, 1 ausgibt, falls $w \in A$, sonst 0
- Wegen Churcher These heißt das: Es gibt einen Algorithmus der 1 ausgibt, falls $w \in A$ und sonst 0
 - dieser Algorithmus entscheidet A

Bsp:

- $A = \{n \in \mathbb{N} | n \text{ ist Primzahl}\}$ IN: $n \in \mathbb{N}$, Frage: Ist n eine Primzahl?
- $B = \{n \in \mathbb{N} | n \text{ und } n + 2 \text{ sind Primzahlen}\}$
- $C = WORT_2 = \{(G, w) | G \text{ kf und } w \in \mathcal{L}(G)\}$ IN: $G \text{ kf}, w \in \Sigma^*$, Frage: $w \in \mathcal{L}(G)$?
- $D = \{f : 2^k - 1 \text{ ist Primzahl}\}$
- $E = \{n \in \mathbb{N} | n \text{ ist gerade und als Summe zweier Primzahlen darstellbar}\}$

Satz 6.1.2

1. $REG \subseteq REC$
2. $CF \subseteq REC$
3. $CS \subseteq REC$

→ Die Frage ob eine Sprachklasse in REC enthalten ist, ist gleichbedeutend mit der Frage ob das Wortproblem für die entsprechende Klasse entscheidbar ist

$$WORT_i = \{(G, w) | G \text{ ist von Typ } i \text{ und } w \in \mathcal{L}(G)\}$$

Beweis: Es würde genügen nur 3) zu zeigen trotzdem:

1. $WORT_1 \in REC$
 - DFA's liefern nach Abarbeitung eines Wortes stets eine Antwort in Form eines akzeptierenden oder nicht akzeptierenden Zustands
 - Sei $A \in REG$, $A = \mathcal{L}(M)$ für einen DFA M
 - Wir konstruieren eine TM, die bei Eingabe w den DFA simuliert und 1 ausgibt, falls der DFA in einem akzeptierenden Zustand endet, sonst
 - TM berechnet c_A
2. $WORT_2 \in REC \rightarrow$ CYK-Algorithmus
3. $WORT_3 \in REC$

Abschnitt 4.4: Das Wortproblem für kontextsensitive Sprachen

Bleibt zu zeigen: $CS \subsetneq REC$

→ Wir konstruieren eine Sprache die Entscheidbar ist, aber nicht kontextsensitiv

→ Diagonalisierung Es seien G_1, G_2, \dots alle kontextsensitiven Grammatiken und w_1, w_2, \dots alle Wörter über einem Alphabet Σ

	w_1	w_2	w_3	\dots
G_1	1	0	0	
G_2	0	1		
G_3	1	0	0	
G_4				0
\vdots				

→ Tabelleneintrag (i, j)

1 - falls $w_j \in \mathcal{L}(G_i)$

0 - sonst

→ Es sei $c = \{w_1 | w_i \notin \mathcal{L}(G_i)\}$ "alle Wörter in deren Spalte auf der die Hauptdiagonale 0 steht"

$c \in REC$, da $WORT_1 \in REC$

Es gibt $w_j \in \mathcal{L}(G_j) \Leftrightarrow w_j \notin c$ für alle $j \in \mathbb{N}^*$

Annahme: c ist kontextsensitiv

Dann gilt es $j \in \mathbb{N}$ mit $\mathcal{L}(G_j) = c \rightarrow$ Das ist ein Widerspruch

Satz 6.1.3

REC ist abgeschlossen bezüglich:

Komplement, Vereinigung, Durchschnitt und Produkt

Beweis: Es seien A, B entscheidbare Sprachen, $A, B \subseteq \Sigma^*$, $A, B \in REC$

TM T_A berechnet c_A , TM T_B berechnet c_B

2. konstruieren $T_{A \cup B}$, die $A \cup B$ berechnet

Eingabe: $w \in \Sigma^*$

- simuliere T_A auf Eingabe w
- falls T_A 1 ausgibt, gebe 1 aus
- falls T_A 0 ausgibt, simuliere T_B auf Eingabe w , Ausgabe von T_B

6.2 Semi-Entscheidbarkeit und Aufzählbarkeit

Sei $A \subseteq \Sigma^*$. Die Semicharakterist. Funktion von A ist wie folgt definiert

$$x_A(w) \begin{cases} 1 & \text{falls } w \in A \\ \text{md.} & \text{sonst} \end{cases}$$

Definition 6.2.1

Eine Menge $A \subseteq \Sigma^*$ heißt semientscheidbar gdw x_A berechenbar ist

Bemerkung: Aus Definition folgt: A ist semi-entscheidbar gdw. Es gibt TM, die Akzeptiert

ACHTUNG: Die Semi-charakteristische Funktion einer Sprache, die nicht von einer TM akzeptiert werden kann, ist nicht berechenbar.

Beispiel: für semientscheidbare

$A = \{(n, m) \in \mathbb{N}^2 \mid \bigvee_{k \geq 1} n^k + m^k \text{ prim}\}$ IN: $n, m \in \mathbb{N}$ Frage: Gibt es

$k \geq 1$ $n^k + m^k$ Prim?

$B = \{(x, y) \in \mathbb{N}^3 \mid \bigvee_{k \in \mathbb{N} \wedge k > 2} x^k + y^k = z^k\}$ IN: $(x, y, z) \in \mathbb{N}^3$ Frage: Gibt es

$n > 2$ $x^k + y^k = z^k$?

$C = \{n \in \mathbb{N} \mid n \text{ ist gerade und als Differenz zweier Primzahlen darstellbar}\}$

Definition 6.2.2

Eine enge $A \subseteq \Sigma^*$ heißt Rekursiv aufzahl gdw. entweder: -) $A = \emptyset$ oder

-) es gibt eine berechenbare, totale Funktion $\mathbb{N} \rightarrow \Sigma^*$ mit $w_f = A$

RE (rekursive enumerable)... Menge aller rekursiv aufzählbaren Mengen

Bemerkung:

- $A = \{f(0), f(1), \dots\}$ "f zählt A auf"

- $f(i) = f(j)$ für $i \neq j$ zulässig
- Entscheidung ob $w \in A$ ist nicht möglich
 - $w \in A$: w kommt in der Aufzählung nach endlich vielen Schritten vor
 - $w \notin A$: w kommt nicht in der Aufzählung vor, man kann nach endlichen vielen Schritten nicht sagen, dass $w \notin A$
- aufzählbare sind noch algorithmisch angebar
- rekursiv Aufzählbarkeit und Abzählbarkeit ist nicht das gleiche

A rekursiv aufzählbar $\Rightarrow A$ abzählbar

Aufzählbar ist die stärkere Bedingung, da f berechnbar sein muss

Für $A = \emptyset$ ist $\emptyset = w_f$ für $f = n. d.$

Satz 6.2.3

Eine Sprache $A \subseteq \Sigma^*$ ist rekursiv aufzählbar gdw. sie semi entscheidbar ist

" \Rightarrow :"

Sei A aufzählbar mittels f . Ein Semi-entscheidungsverfahren für A ist

Eingabe $w \in \Sigma^*$

for $i = 1, 2, 3, \dots$ do

if $f(i) = w$ then output 1

" \Leftarrow :" Wir geben ein Algorithmus an, der eine Aufzählende Funktion berechnet

$j = 0$

$t = 0$

repeat

erzeuge $\Sigma^{\leq t} = \{w_1, \dots, w_t\} = \Sigma^0 \cup \Sigma^1 \cup \dots \cup \Sigma^t$

for $m = 1$ to l

simuliere erst t Takte von T_A auf w_m

falls $x_A(w_m) = 1$ ausgegeben wird

gebe $f(j) = w_m$ aus

$$j = j + 1$$

$$t = t + 1$$

Behauptung $w_f = A$

" \subseteq :" Der Algorithmus gibt nur Wörter aus, für die T_A auf Eingabe w 1 ausgibt

" \supseteq :" $w \in A \rightarrow$ Es gibt z , so dass T_A angewendet auf w noch z Takten 1 ausgibt

\rightarrow Für ein $t \geq z$ wird obiger Algorithmus w ausgegeben

Folgerung 6.2.4

Eine Sprache ist rekursiv aufzählbar gdw. sie von einer TM akzeptiert werden kann

Satz 6.2.5

$$RE = \mathcal{L}_0$$

Beweis: Satz 4.3.3 und Folgerung 6.2.4

$$\mathcal{L}_0 = \{A : \bigvee_{TM M} L(M) = A\}$$

\rightarrow noch offen: $CS \subsetneq \mathcal{L}_0$, bisher gezeigt: $CS \subsetneq REC$, $RE = \mathcal{L}_0$

\rightarrow gzz. $REC \not\subseteq RE$

Satz 6.2.6 $REC \not\subseteq RE$

Beweis: 1.Fall: $A = \emptyset \rightarrow A \in RE$ nach def

2.Fall: $A \neq \emptyset$, $A \in REC$

\rightarrow es gibt eine TM M_1 , die c_A berechnet wir konstruieren TM M_2 , die χ_A berechnet

$\rightarrow M_1$ wird modifiziert: immer, wenn M_1 0 ausgibt, soll M_2 in eine Endlosschleife

gehen

\rightarrow Aus den Sätzen 6.2.5 ($REC \subseteq RE$) 6.2.5 ($RE = \mathcal{L}_0$) und 6.1.2 ($CS \not\subseteq REC$) ergibt sich der Beweis für Satz 4.1.2 ($CS \not\subseteq \mathcal{L}_0$)

Zusammenfassung

Folgende Aussagen sind äquivalenz: $\rightarrow A$ ist rekursiv aufzählbar

$\rightarrow A$ ist semi-entscheidbar

$\rightarrow A$ ist vom Typ-0

$\rightarrow A = \mathcal{L}(M)$ für eine TM M

$\rightarrow \chi_A$ ist Turingberechenbar

- A ist Definitionsbereich einer berechnbaren Funktionen
- A ist Wertebereich einer berechnbaren Funktion

Satz 6.2.7:

- RE ist abgeschlossen bezüglich:
- Durchschnitt
 - Vereinigung
 - Produkt

- ist RE abgeschlossen bezüglich: Komplement?

6.3 Beziehung zwischen REC und RE

Satz 6.3.1

$$\forall A \subseteq \Sigma^* : A \in RE \text{ und } \overline{A} \in RE \Leftrightarrow A \in REC$$

" \Leftarrow :" Ist A entscheidbar, so ist auch \overline{A} entscheidbar

→ Wegen $REC \subseteq RE$ folgt $A \in RE$ und $\overline{A} \in RE$

" \Rightarrow :" Sei M_A eine TM, die χ_A berechnet und $M_{\overline{A}}$ eine TM, die $\chi_{\overline{A}}$ Eingabe $w \in \Sigma^*$

→ Simuliere M_A und $M_{\overline{A}}$ schrittweise parallel auf Eingabe w

→ falls M_A 1 ausgibt, dann gebe 1 aus

→ falls $M_{\overline{A}}$ 1 ausgibt, dann geben 0 aus

→ diese Maschine hält immer an und liefert damit ein Entscheidungsverfahren für A

Satz 6.3.2

RE ist nicht abgeschlossen bezüglich des Komplement

→ Annahme: RE abgeschlossen bezüglich Komplement

→ dann folgt aus Satz 6.3.1: $RE = REC \Rightarrow$ Widerspruch

6.4 Kodierung von Turingmaschinen über {0,1}

$$M = (\Sigma, \Gamma, Z, \delta, Z_0, Z_E, \square), \Sigma = \{0, 1\}, \Gamma = a_0, \dots, a_k, Z = z_0, \dots, z_n$$

→ die Nummern der Symbole 0, 1, \square seien festgelegt, z.B.: $a_0 = \square, a_1 = 1, a_2 = 0$

→ Sei $\# \notin \Gamma \cup Z$

→ Codewort von M über $\{0, 1, \#\}$ → $c(M) = \#bin(k)\#bin(n)\#code(\delta)\#\#code(F)\#$

→ für jedes $(i, j) \in \{0, 1, \dots, n\} \times \{0, 1, \dots, k\}$ mit $\delta(z_i, a_j) = (z_l, a_m, X)$ sei:

→ $code(\delta(i, j)) = \# \# bin(i) \# bin(j) \# bin(l) \# bin(m) \# bin(x) \# \#$ mit $bin \begin{cases} 00 & X = L \\ 01 & X = M \\ 10 & X = N \end{cases}$

→ das Codewort für δ ergibt sich durch das hintereinanderschreiben aller $code(\delta(i, j))$ mit $\delta(z_i, a_j) \neq nd.$ in beliebiger Reihenfolge

→ Für $F = \{z_{i_1}, \dots, z_{i_l}\}$ ist: $code(F) = \# bin(i_1) \# bin(i_2) \# \dots \# bin(i_l) \#$

→ $c(M)$ Codewort über $\{0, 1, \#\}$

→ 00

→ $c(M)$ können wir $code(M) = \langle M \rangle$ zuordnen: $1 \rightarrow 01$

$\# \rightarrow 11$

Bemerkung: - $\langle M \rangle$ Kodierung der TM M

- aus $\langle M \rangle$ lässt sich M rekonstruieren (bis auf die Namen der Zustände und Bandsymbole)

- ebenso können Grammatiken, DFA's, NFA's usw. codiert werden

- nicht jedes $w \in \{0, 1\}^*$ ist code einer TM

Folgende Sprachen sind entscheidbar:

$$1. A_{DFA} = \left\{ \langle B, w \rangle \mid B \text{ ist DFA und } B \text{ akzeptiert } w \right\}$$

$$2. E_{DFA} = \left\{ \langle A \rangle \mid A \text{ ist DFA und } \mathcal{L}(A) \neq \emptyset \right\}$$

$$3. S_{REG} = \left\{ \langle G_1, G_2 \rangle \mid G_1, G_2 \text{ reguläre Grammatik und } \mathcal{L}(G_1) \cap \mathcal{L}(G_2) = \emptyset \right\}$$

$$4. EQ_{DFA} = \left\{ \langle A, B \rangle \mid A \text{ und } B \text{ sind DFA's und } \mathcal{L}(A) = \mathcal{L}(B) \right\}$$

$$5. WORT_2 = \left\{ \langle G, w \rangle \mid G \text{ ist k. f. G. und } G \text{ erzeugt } w \right\}$$

$$6. E_{CF} = \left\{ \langle G \rangle \mid G \text{ ist k. f. G. und } \mathcal{L}(G) = \emptyset \right\}$$



7. Wir konstruieren eine TM, die die Arbeit des DFA's nachvollzieht und entsprechend antwortet TM M :

- Eingabe $\langle B, w \rangle$
- Simuliere DFA B auf Eingabe w
- falls B akzeptiert gebe 1 aus, sonst 0

8. Markierungsalgorithmus für die Zustände

- Startzustand markieren
- alle Zustände markieren, die aus markierten Zustand erreichbar sind
- Wiederholen, bis keine neuen Zustände markiert werden
- Test, ob Endzustände markiert sind

9. DFA's A_1, A_2 konstruieren mit $\mathcal{L}(A_i) = \mathcal{L}(G_i)$

- konstruiere Produktautomaten A aus A_1 und A_2
- Teste, ob $\mathcal{L}(A) = \emptyset$ (mit 2)

10. ÜA

11. CYK

12. $E_{CF} = \{ \langle G \rangle \mid G \text{ ist kontextfreie Grammatik und } \mathcal{L}(G) = \emptyset \}$

- es kann eine TM konstruiert werden, die folgendes ausführt
- Markierungsalgorithmus: es werden alle Nichtterminale markiert aus denen ein Wort abgeleitet werden kann
- 1. Forme G in Chomsky-Normalform um
- 2. Markiere alle Nichtterminale aus Regeln der Form $A \rightarrow a$
- 3. Markiere alle Nichtterminale A aus Regeln der Form $A \rightarrow BC$, falls B und C
- 4. Wiederhole 3. bis keine neuen Nichtterminale markiert werden
- 5. Teste ob das Startsymbol S markiert ist

6.5 Das Halteproblem

6.5.1 Problemstellung und intuitive Argumentation

- Programmierer machen Fehler beim Programme schreiben
 - Diagramme gelangen in Endlosschleife und halten nicht an
- es wäre von Nutzen, wenn man Programme auf das Vorhandensein von Endlosschleife untersuchen kann, bevor sie ausgeführt werden
- Goldbachsche Vermutung:
 - Jede gerade Zahl größer als 2 lässt sich als Summe zweier Primzahlen darstellen
 $n := 4$
Wiederhole
teste, ob n als Summe zweier Primzahlen darstellen
 $\text{ja} \rightarrow n := n + 2$
 $\text{nein} \rightarrow \text{stopp}$
- letzter Satz von Fermat
 - Es gibt kein $k > 2$ für das es $(x, y, z) \in \mathbb{N}^3$ gibt mit $x^k + y^k = z^k$
| 1 | 2 | 3 | | | |
|---|---|---|---|---|---|
|1|1||3|→|4|
||↓|↗||↘|
|2|2||5||
|||↘|||
|3|6|||
- Fermatsche Primzahlen
 - Primzahlen der Form $2^k + 1$
 - $2^{2^m} + 1 \quad \backslash \quad 3, 5, 17, 257, 65537$
- Hält ein gegebener Algorithmus A bei gegebener Eingabe?
 - Zunächst intuitive Argumentation, dass das Halteproblem nicht entscheidbar

ist

→ A_0, A_1, A_2, \dots Auflistung aller Algorithmen, die als Eingabe eine natürliche Zahl haben

Halteproblemtabelle:

$\frac{\text{Eingabe}}{\text{Ausgabe}}$	0	1	2	3	4	5	6
---	---	---	---	---	---	---	---
A_0	J	N	J	J	N		
A_1	N	N	N	J	J	..	
A_2	N	J	J	J	J		
A_3	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:

$$t(i, j) = \begin{cases} J & A_i \text{ terminiert mit Eingabe } j \\ N & \text{sonst} \end{cases}$$

Annahme Das Halteproblem ist entscheidbar, d.h. es gibt einen Algorithmus A_H , der bei Eingabe (i, j) den Tabelleneintrag $t(i, j)$ berechnet

Algorithmus A: Eingabe $i \in \mathbb{N}$

berechne mit A_k $t(i, j)$

falls $t(i, i) = J$ gehe in Endlosschleife

else stop

- da alle Algorithmen in der Tabelle aufgeführt sind, gibt es $n \in \mathbb{N}$ mit $A_n = A$
- $t(n, n) = J$ heißt A_n stoppt auf Eingabe n , andererseits läuft $A = A_n$ für $t(n, n) = J$ endlos!
- für $t(n, n) = N$ ergibt sich ebenfalls ein Widerspruch
- somit gibt es Algorithmen A nicht und damit kann es auch A_H nicht geben

6.5.2 Formale Argumentation mittels TM

- Gibt es eine TM, die für jede beliebige TM entscheiden kann, ob sie mit beliebiger Eingabe enthält oder nicht?

Definition 6.5.2.1

Unter dem allgemeinen Halteproblem verstehen wir die Menge:

$$\rightarrow A_{TM} = \left\{ \langle M, w \rangle \mid M \text{ ist Turingmaschine und } M \text{ akzeptiert } w \right\}$$

Satz 6.5.2.2 $A_{TM} \notin REC$

Beweis: Annahme: Es gibt eine TM H , die A_{TM} entscheidet

$$\rightarrow \text{d.h.: } H(\langle M, w \rangle) = \begin{cases} 1 & \text{falls } M \text{ } w \text{ akzeptiert} \\ 0 & \text{falls } M \text{ } w \text{ nicht akzeptiert} \end{cases}$$

\rightarrow Dann gibt es auch folgende TM D , die H als Unterprogramm nutzt

$\rightarrow D$: Eingabe $\langle H \rangle$

1. Simuliere H auf Input $\langle M, \langle M \rangle \rangle$

2. if $H(\langle M, \langle M \rangle \rangle) = 1$ (d.h. M akzeptiert $\langle M \rangle$)

gehe in Endlosschleife

else gebe 1 aus

$$\rightarrow D(\langle M \rangle) = \begin{cases} 1 & \text{falls } H(\langle M, \langle M \rangle \rangle) = 0 \\ n.d. \text{ Endlosschleife} & \text{falls } H(\langle M, \langle M \rangle \rangle) = 1 \end{cases}$$

\rightarrow Die TM D wird nun auf ihren eigenen Code angewendet

$$D(\langle D \rangle) = \begin{cases} 1 & \text{falls } H(\langle D, \langle D \rangle \rangle) = 0, \text{ d.h. } D \text{ akzeptiert } \langle D \rangle \text{ nicht} \\ n.d. & \text{falls } H(\langle D, \langle D \rangle \rangle) = 1, \text{ d.h. } D \text{ gibt bei Eingabe } \langle D \rangle \text{ 1 aus} \end{cases}$$

\rightarrow Eine solche Maschine D gibt es nicht, also kann es auch die Maschine H nicht geben! \square

Satz 6.5.2.3 $REC \not\subseteq RE$

Beweis: $A_{TM} \in RE$

\rightarrow konstruiere TM, die $\chi_{A_{TM}}$ berechnet

Eingabe: $\langle M \rangle, w$

simuliere M auf w

falls M akzeptiert wird, gebe 1 aus

$\rightarrow A_{TM} \notin REC$ folgt $A_{TM} \in \setminus REC$

Satz 6.5.2.4

RE ist nicht abgeschlossen bezüglich Komplement

Annahme: RE ist abgeschlossen bezüglich Komplement

\rightarrow dann ist wegen Satz 6.3.1: $RE = REC$ Widerspruch

Satz 6.5.2.5

$\overline{A_{TM}} \notin RE$

Beweis: $A_{TM} \in RE$

→ falls auch $\overline{A_{TM}} \in RE$ folgt $A_{TM} \in REC$ Widerspruch

$A \in RE$ und $\overline{A} \in RE \Leftrightarrow A \in REC$

7. Kapitel - NP Vollständigkeit

→ bisher ging es um Machbarkeit, jetzt soll es schnell gehen

→ betrachten ab jetzt nur noch entscheidbare Probleme

Wiederholung: Entscheidungsprobleme (ja/nein Probleme) können als Mengen dargestellt werden

- Eulerkreis:
 - geg: Sei Graph $G = (V, E)$
 - Frage: Besitzt G einen Eulerkreis?
 - (gibt es einen Pfad in G , der jede Kante genau einmal benutzt und wieder am Startpunkt endet)
 - $EC = \{ \langle G \rangle \mid G \text{ hat einen Eulerkreis} \}$
 - → "gehört G zur Menge EC ?"
- Hamiltonkreis:
 - geg: Graph $G = (V, E)$
 - Frage: Besitzt G einen Hamiltonkreis?
 - (Permutation π der Knotenindizes $(v_{\pi(1)}, \dots, v_{\pi(n)})$, sodass $\forall i \in \{1, \dots, n-1\}$
 - $\{v_{\pi(i)}, v_{\pi(i+1)}\} \in E$ und $\{v_{\pi(n)}, v_{\pi(1)}\} \in E$... Pfad, der jeden Knoten genau einmal braucht und zum Startknoten zurückkehrt
- Clique:
 - geg: Graph $G = (V, E)$
 - Frage: Besitzt G eine Clique der Größe mindestens k ?
 - ($V' \subseteq V$ mit $|V'| \geq k$ und für alle $u, v \in V'$ mit $u \neq v$ gilt $\{u, v\} \in E$)
 -
 - $CLIQUE = \{ \langle G, K \rangle \mid G \text{ ist Graph und besitzt eine Clique der Größe mindestens } K \}$

- Independent Set:
 - geg: Graph $G = (V, E)$
 - Frage: Besitzt G eine Independent Set der Größe mindestens k ?
 - $(V' \subseteq V \text{ mit } |V'| \geq k \text{ und für alle } u, v \in V' \text{ mit } u \neq v \text{ gilt } \{u, v\} \notin E)$
 -
 - $INDEPENDENTSET = \{ \langle G, k \rangle \mid G \text{ ist Graph und besitzt einen Independent Set der Größe } k \}$
- Vertex-Cover:
 -
 - $VERTEXCOVER = \{ \langle G, k \rangle \mid G \text{ ist Graph und besitzt ein Vertex Cover der Größe } k \}$
 - $(V' \subseteq V) \text{ mit } |V'| \leq k, \text{ sodass für alle } \{u, v\} \in E \text{ gilt: } u \in V' \text{ oder } v \in V'$
- Traveling-Salesperson:
 - geg: $n \times n$ Matrix $M_{i,j}$ von Entfernungen zwischen n "Städten", $k \in \mathbb{N}$
 - Frage: Gibt es eine Permutation (Rundreise), sodass
 - $$\sum_{i=1}^{n-1} M_{\pi(i), \pi(i+1)} + M_{\pi(n), \pi(1)} \leq k$$

7.1 Die Klasse P

→ Wie misst man Zeit?

Definition 7.1.1

Es sei M eine deterministische TM, die bei jeder Eingabe hält. (Sie berechnet eine totale def. Funktion)

→ $time_M : \Sigma^* \rightarrow \mathbb{N}, time_M(w) = \text{Zahl der Takte (Konfigurationsübergänge) bis } M \text{ auf Eingabe } w \text{ hält}$

→ $Time_M = \mathbb{N} \rightarrow \mathbb{N}, Time_M(w) = \max\{time_M(w) : |w| = n\}$

Sei $t : \mathbb{N} \rightarrow \mathbb{R}$ eine Funktion:

→ Die Komplexitätsklasse $DTIME \langle t(n) \rangle$ ist

$$\left\{ A \subseteq \Sigma^* \mid \bigvee_{DTM M} (M \text{ berechnet } c_A \text{ und } \bigwedge_n time_M(n) \leq t(n)) \right\}$$

Definition 7.1.2

Eine Pol die Menge der Polynome der Form

$$p(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_2 n^2 + a_1 n + a_0 \text{ mit } n, a \in \mathbb{N} \text{ für alle } 0 \leq i \leq k$$

→ Die Komplexitätsklasse P (deterministische Polynomzeit) ist definiert als

$$P = \bigcup_{p \in Pol} DTIME(p(n))$$

- P ist die Menge aller Probleme, die sich in Polynomzeit lösen (entscheiden) lassen
- P ist die Klasse aller effizient lösbaren Probleme
- P ist unabhängig von zugrunde gelegten Maschinenmodell (hier TM)
- (die Überführung der Berechnungsmodelle ist in polynomiellen Zeitaufwand möglich)

Bsp:

Eulerkreis:

Naiver Algorithmus:

- prüfe jede Permutation der Kanten, ob sie einen Eulerkreis darstellt
- $t(m) \leq m! \leq \left(\frac{m}{2}\right)^{\frac{m}{2}} \notin Pol$
- *ABER*: G hat einen Eulerkreis gdw. G zusammenhängend ist und jeder Knoten hat gerade Valenz
- $EG = \left\{ \langle G \rangle \mid G \text{ ist eine Grammatik mit Eulerkreis} \right\} \in P$

Hamiltonkreis:

Naiver ALgorithmus: Prüfe jede Permutation der, ob sie einen Hamiltonkreis darstellt.

7.2 Die Klasse NP (nicht-deterministisch-polynomial)

Definition 7.2.1

Rechenzeit einer nichtdeterministischen TM

Sei N eine NTM mit $\mathcal{L}(N) \subseteq \Sigma^*$

- $time_N : \Sigma \rightarrow \mathbb{N}$
 - $$time_N(w) = \begin{cases} \min & \{ \text{Zahl der Konfigurierbarer übergänge (akzeptierende Berechnungen)} \\ n.d. & \text{sonst} \end{cases}$$
 - $time_N = \mathbb{N} \rightarrow \mathbb{N}$, $time_N(n) = \max\{time_N(w) \mid |w| = n \text{ und } w \in \mathcal{L}(N)\}$
 - $time_N = \mathbb{N} \rightarrow \mathbb{R}$, die Komplexitätsklasse
- $$NTIME(t(n)) = \{A \subseteq \Sigma^* \mid \exists NTM N : \mathcal{L}(N) = A \wedge \forall n : Time_N(n) \leq t(n)\}$$

Bemerkung:

- eine Eingabe wird in Zeit t akzeptiert, wenn der kürzeste akzeptierende Pfad höchstens t lang ist
- es kann unendlich lang nicht akzeptierende Pfade geben

Definition 7.2.2

Die Komplexitätsklasse NP (nicht Polynomialzeit) ist gerade:

$$NP = \bigcup_{p \in \text{Pol}} NTIME(p(n))$$

Frage: Wie sehen nichtdeterministische Algorithmen für unsere Eingangs genannten Probleme aus?

→ das Raten einer Lösung und ihr anschließendes Verifizieren ist ein nichtdeterministischer Algorithmus!

Beispiel:

Hamiltonkreis:

- Raten einer Knotenpermutation

- Testen, ob es eine Clique ist

⇒ das ist in Polynomialzeit möglich

Clique:

- Raten einer Menge von k Knoten

- Testen, ob es eine Clique ist

⇒ Indset, Vertex-Cover

→ all diese Probleme gehören zu der Klasse NP

- noch niemand hat einen P -Algorithmus für diese Probleme gefunden
 - NP enthält sehr viele interessante und schwer lösbare praktische Probleme
 - $P \subseteq NP$
- $P \neq NP$? Eine der größten offenen Fragen der Mathematik

7.3 NP-Vollständigkeit

Definition 7.3.1

Eine Sprache $A \subseteq \Sigma^*$ ist in polynomialzeit-reduzierbar auf eine Sprache $B \subseteq \Delta^*$, falls es eine total definierte und in polynomialer Zeit berechenbare Funktion: $f : \Sigma^* \rightarrow \Delta^*$ gibt, sodass für alle $x \in \Sigma^*$ gilt: $x \in A \Leftrightarrow f(x) \in B$

→ $A \leq_p B$

Definition 7.3.2

- Eine Sprache A heißt NP -schwer, falls für alle Sprachen $L \in NP$ gilt: $L \leq_p A$

- Eine Sprache A heißt NP-vollständig, falls A NP-schwer ist und $A \in NP$ gilt

Satz 7.3.3

Falls $A \leq_p B$ und $B \in P$, gilt $A \in P$

Beweis: Sei T_B TM, die B in Polynomialzeit entscheidet

Sei f die Reduktionfunktion und T_f TM, die f in Polynomialzeit berechnet

Wir konstruieren TM T , die A in Polynomialzeit entscheidet

T : Eingabe $w \in \Sigma^*$

Berechne $f(w)$ (T_f auf w anwenden)

Simuliere T_B auf $f(w)$

Gebe Ausgabe von T_B aus

Satz 7.3.4

Falls $A \leq_p B$ und $B \in NP$ gilt, folgt $A \in NP$

Beweis: Analog mit T_B ist nichtdeterministische Maschine und damit T auch

Satz 7.3.5

Beweis: " \Leftarrow :" Wenn A NP-Vollständig ist, gilt $A \in NP$ und dann wegen $NP = P$ auch $A \in P$

" \Rightarrow :" Sei $A \in P$ und L eine beliebige Sprache aus NP , da A NP-Vollständige ist, gilt:

$L \leq_p A$

\rightarrow damit gilt auch $L \in P$, da L beliebig gewählt war, folgt $P = NP$

Bemerkung:

- falls man für ein einziges NP vollständiges Problem zeigen kann, dass es in P liegt, gilt $P = NP$

- falls man für ein einziges NP vollständig Problem zeigen kann, dass es nicht in P liegt, dann gilt: $P \neq NP$

Satz 7.3.6

Falls A NP vollständig und $A \leq_p B$ folgt: B ist auch NP schwer

Beweis: Für alle $\mathcal{L} \in NP$ gilt $L \leq_p A$ und wegen $A \leq_p B$ auch $L \leq_p B$
 SAT... Erfüllbarkeitsproblem der Aussagenlogik
 geg: eine Formel F der Aussagenlogik

Frage: Ist F erfüllbar, d.h. gibt es eine Belegung β der Variablen, sodass $I_\beta(F) = 1$

$SAT = \{ \langle F \rangle \mid F \text{ ist erfüllbare Formel der Aussagenlogik} \}$

$((A \rightarrow B) \wedge (B \vee C)) \leftrightarrow ((A \wedge B) \leftrightarrow C) \rightarrow A$

Satz 7.3.7

SAT ist NP vollständig

→ ohne Beweis

$SAT \in NP$ Belegung raten und $I_\beta(F)$ bestimmen

→ zu zeigen: für alle $A \in NP$ gilt $A \leq_p SAT$

Idee: Arbeitsweise einer nichtdeterministischer TM durch eine Formel beschreiben

3-

$SAT = \{ \langle F \rangle \mid F \text{ ist erfüllbare Formel in konjunktiver Normalform mit höchstens 3 l} \}$

Man kann zeigen: $SAT \leq_p 3SAT$

Satz 7.3.7 Cook, Levin

SAT ist NP vollständig

→ ohne Beweis

$SAT \in NP$: Belegung raten und $I_\beta(F)$ bestimmen

→ zu zeigen: für alle $A \in NP$ gilt $A \leq_p SAT$

Idee: Arbeitsweise einer nichtdeterministischen TM durch eine Formel beschreiben

3 – $SAT = \{ \langle F \rangle \mid F \text{ erfüllbare Formel in konjunktiver Normalform mit höchstens 3 l} \}$

Man kann zeigen: $SAT \leq_p 3SAT$

Satz 3.7.8

INDSET ist NP vollständig

Beweis: $INDSET \in NP$: Knotenmenge raten und testen, ob es eine unabhängige Menge ist

Wir zeigen: $3SAT \leq_p INDSET$

Wir benötigen einen Algorithmus, einen (beliebigen) Input für $3SAT$ in einen Input $INDSET$ überführt

→ zu beliebiger Formel F ist 3 – KNF muss $G = (V, E)$, $k \in \mathbb{N}$ konstruiert werden, sodass F erfüllbar ist gdw. G eine unabhängige Menge der Größe k hat

$$V := \{(1, 1), (1, 2), \dots, (m, 2), \dots (m, 3)\}$$

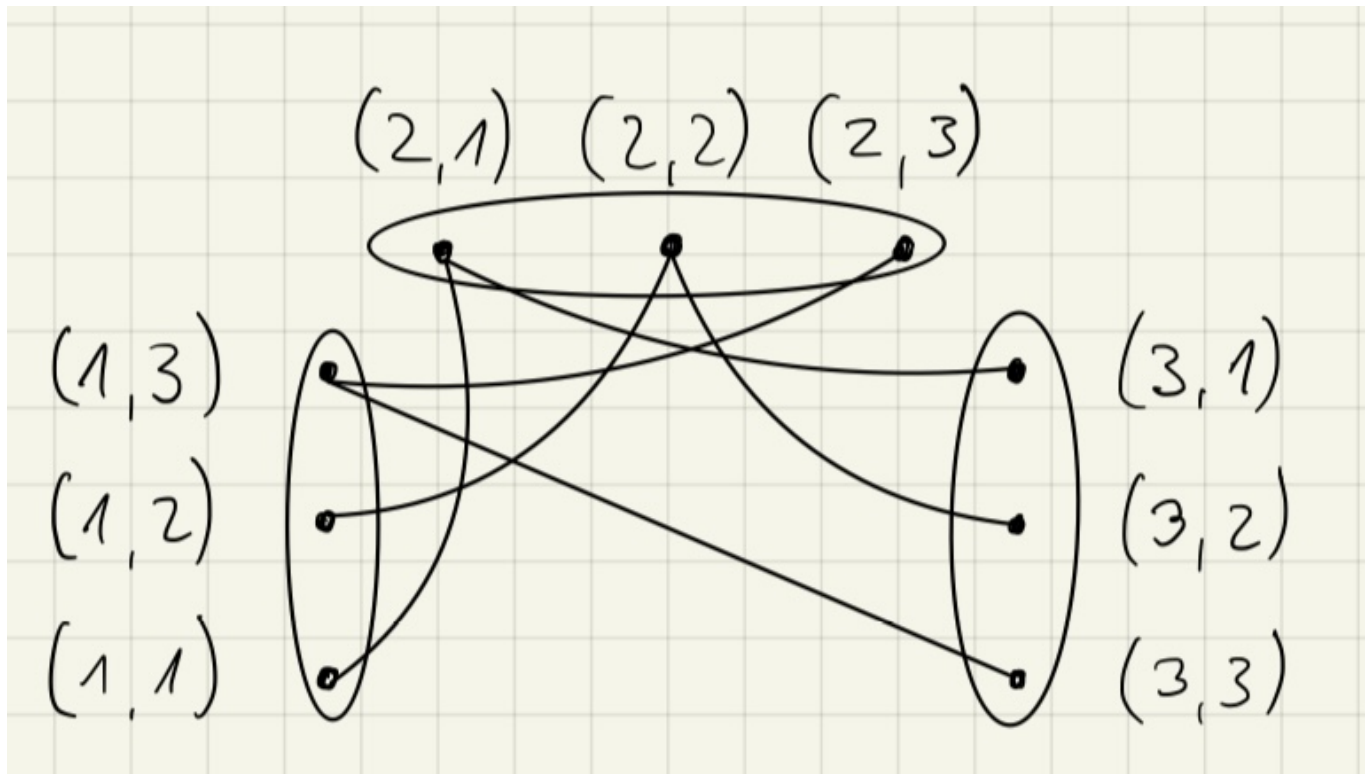
$$E := \{(i, j), (p, q) | i = p \text{ oder } z_{ij} = \neg z_{pq}\}$$

$$K := m$$

$x_1 = 1, x_2 = 0, x_3 = 1$ ist erfüllende Belegung

→ $(1, 1)(2, 2)(3, 3)$ sind paarweise nicht verbunden

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



$$\langle F \rangle \in SAT$$

$\Leftrightarrow F$ ist erfüllbar durch Belegung β

\Leftrightarrow es gibt in jeder Klausel ein Literal, dass unter β den Wert 1 erhält

z.B. $z_{1j_1}, z_{2j_2}, z_{3j_3}, \dots, z_{mj_m}$

\Leftrightarrow es gibt Literale $z_{1j_1}, z_{2j_2}, z_{3j_3}, \dots, z_{mj_m}$ die paarweise nicht komplementär sind

\Leftrightarrow es gibt Knoten $(1, j_1), (2, j_2), \dots, (m, j_m)$, die paarweise nicht verbunden sind

$\Leftrightarrow G$ hat eine unabhängige Menge der Größe $k \Leftrightarrow \langle G, k \rangle \in INDSET$

$F \rightarrow (G, k)$ ist in Polynomialzeit berechenbar \square

BOXCOVER

Geg.: n Punkte in der Ebene, $k \in \mathbb{N}$

Frage: Können die Punkte durch k achsenparallele Quadrate mit Seitenlänge 1 überdeckt werden?

Satz BOXCOVER ist NP-schwer

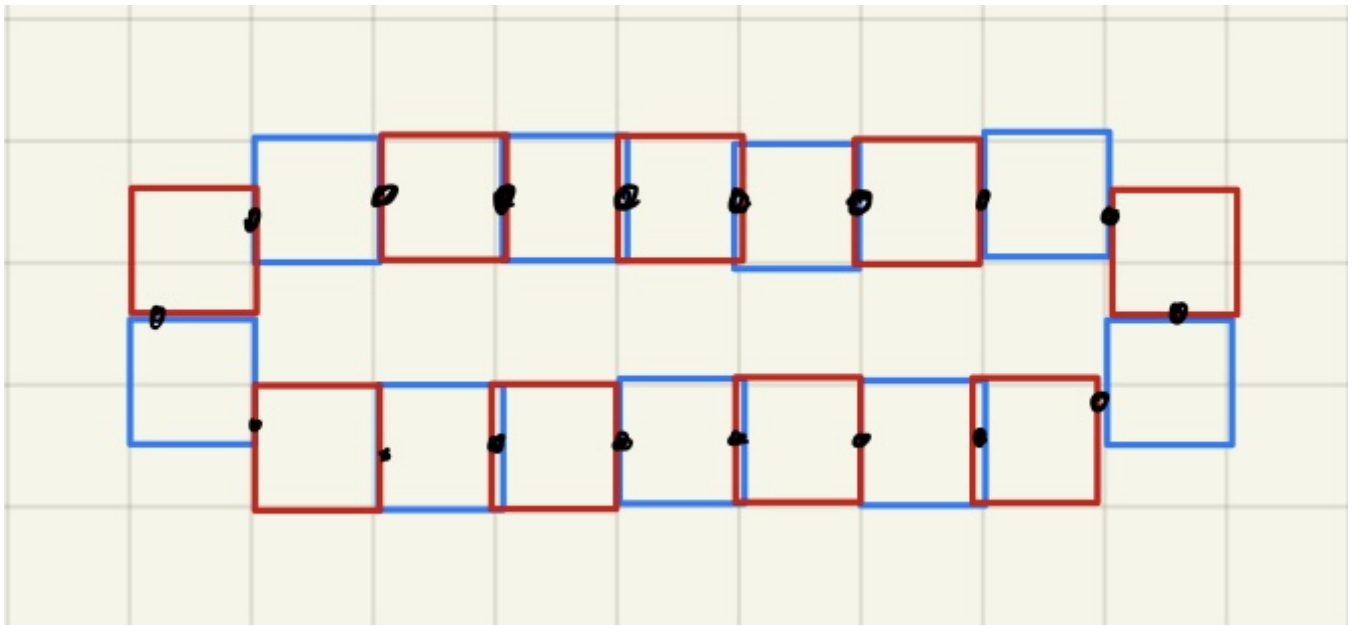
Beweis: $3SAT \leq_p BOXCOVER$

... konstruiere zu beliebiger konkreter Eingabe I für $3SAT$ eine konkrete Eingabe für $BOXCOVER$, so dass die k -Boxen ausreichen gdw. I erfüllbar ist

(1) Jede boolsche Variable wird durch eine Punktschleife simuliert, die auf genau 2 verschiedene

Weisen mit einer minimalen Anzahl von Quadraten

→ eine Überdeckung entspricht TRUE, die andere FALSE

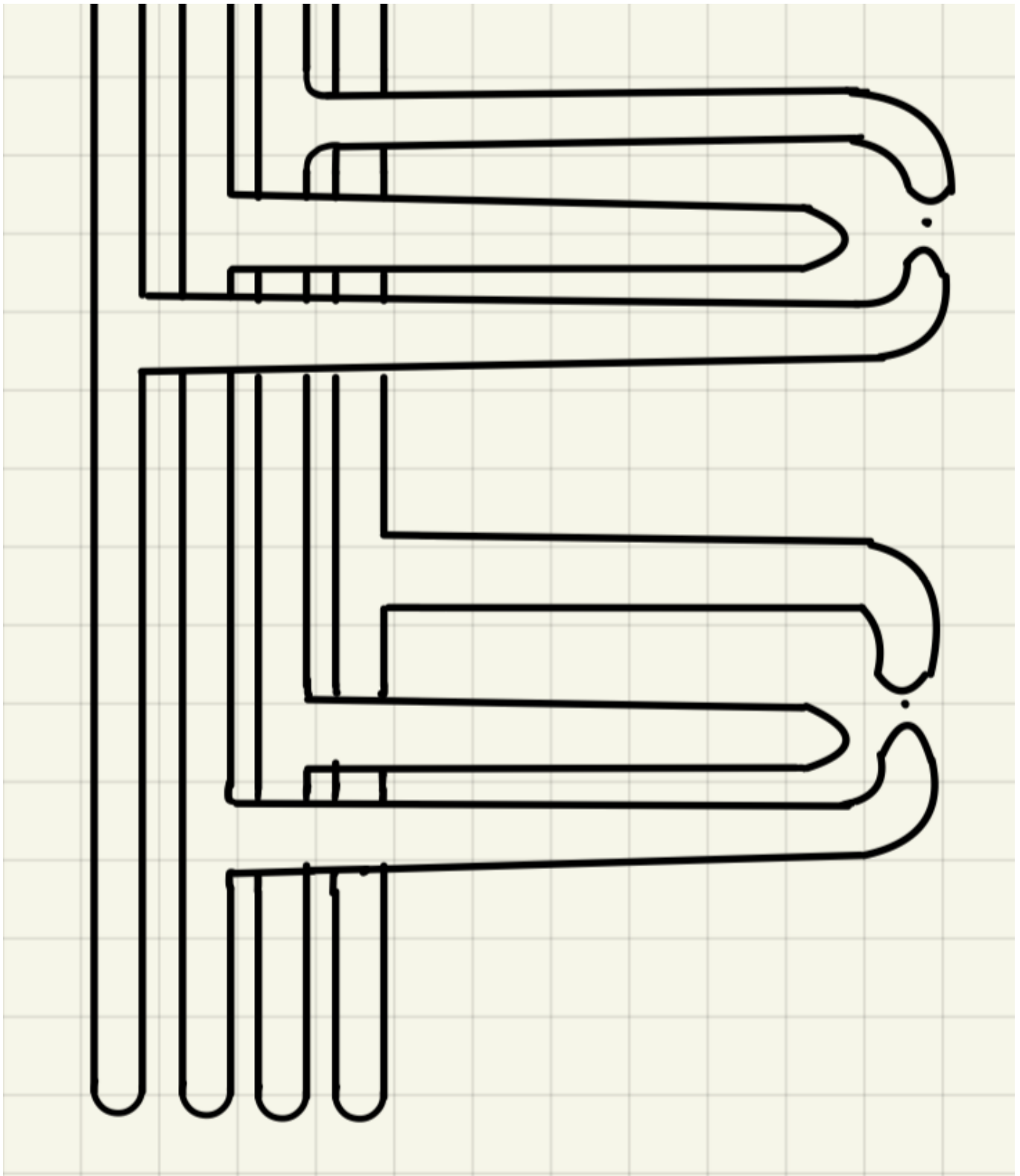


#

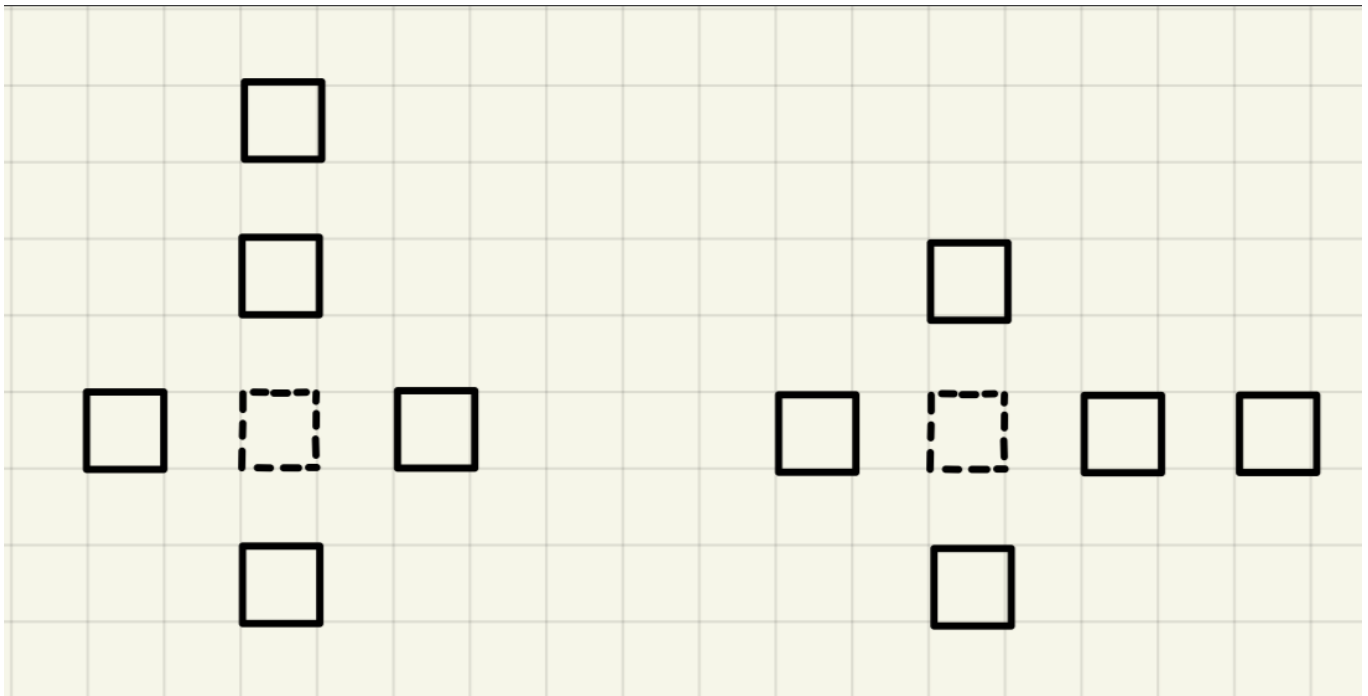
blaue und rote Quadrate

(2) Gesamte Konstruktion





(3) Kreuzungskomponente

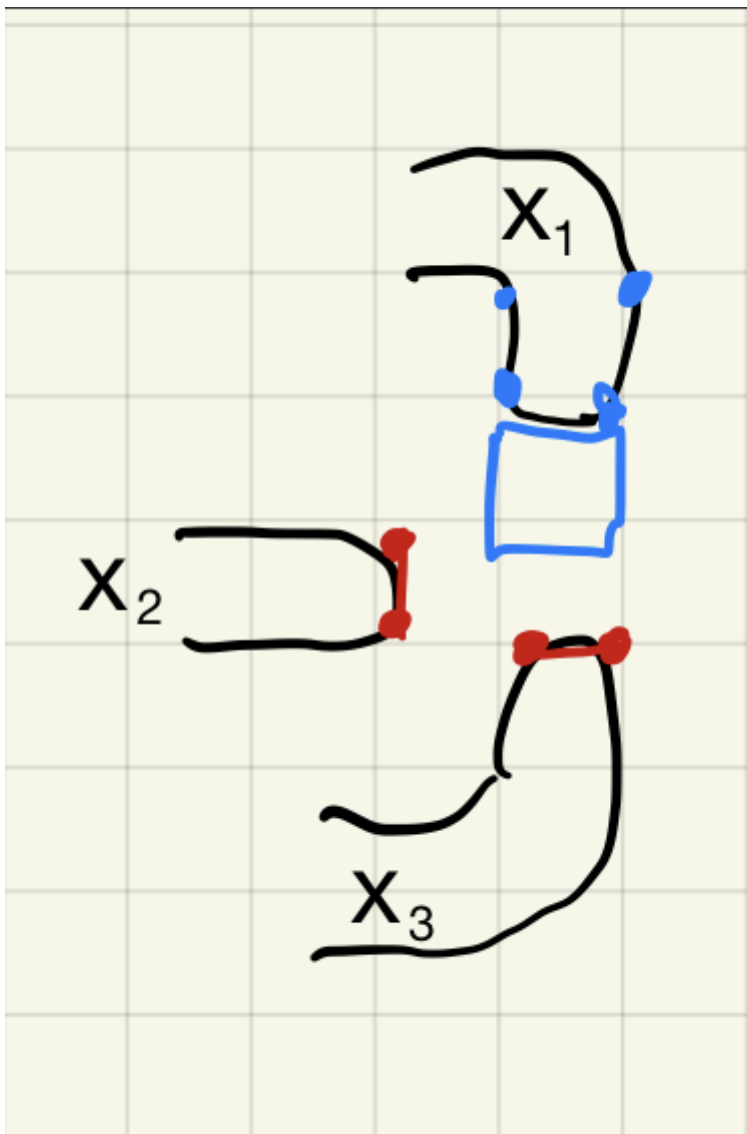


- Pro Kreuzungspunkt kann eine Box gespart werden
- Schleifen beeinflussen sich nicht
- N_c Anzahl der Kreuzungspunkte $k := \sum_{i=1}^n k_i - N_c$

(4) Klauselkomponente

$$x_1 \wedge \neg \neg x_2 \wedge \neg x_3$$

Schleifen für x_1 deckt im Falle der blauen (TRUE) Belegung den zusätzlichen Punkt ab.
 Schleifen für x_2 und x_3 im Falle der FALSE-Belegung



- Für jede Klausel werden die entsprechenden Variablenschleifen so zusammengefügt, dass für die positiven Literale die blauen (TRUE) Überdeckung den zusätzlichen Punkt überdeckt und für jede negativen die rote (FALSE)

Lemma

Sei I eine Eingabe für $3SAT$ und $P(I)$ die beschriebene Punktstruktur bestehend aus n Schleifen, N_c Kreuzungen und M Klauselkomponenten

Falls die einzelnen Schleifen jeweils mit k_i Boxen überdeckt werden können, kann $P(I)$ genau dann mit $k := \sum_{i=1}^n k_i - N_c$ Boxen überdeckt werden.