Geschrieben von Cora Zeitler, Maximilian Ackert

1. Kapitel Sprachen und Grammatiken

1.1 Grammatik

1.1.1 Alphabet und Wörter

Definition 1.1.1.1

Ein Alphabet $\sum$ ist eine endliche nicht leere Menge.
Die Elemente von $\sum$ heißen Buchstaben.
z.B.:
$\sum_{latein} := \{a, b, c, \ldots, x, y, z\}$
$\sum_{griech} := \{\alpha, \beta, \gamma, \ldots\}$
$\sum_{binär} := \{0, 1\}$
$\sum_{Ziff} := \{0, 1, 2, \ldots, 9\}$
$\sum_{Tastatur} := \{a, b, c, \ldots, z\}$
$\cup\{A, B, \ldots, Z\}$
$\cup\{\ddot{a}, \ldots, \ddot{A}, \ldots\}$
$\cup\{0, \ldots, 9\} \cup \{?, !, ., \} \cup \{\sqcup\}$

Definition 1.1.1.2

Es sei $\sum := \{a_1, \ldots, a_m\}$ ein Alphabet. Ein Wort w ist eine endliche Folge von Buchstaben aus $\sum$,
etwa $w = a_{i1}, a_{i2}, \ldots, a_{in}$ $a_{ij} \in \sum$. Die länge $|w|$ eines Wortes ist die Anzahl seiner Buchstaben.

λ sei das eindeutig bestimmtes Wort der länge 0.

Definition 1.1.1.3

Es sei $\sum$ ein Alphabet

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

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

Nebenrechnung:

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

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

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

$(\sum)^*$**... Menge aller endlichen Folgen über $\sum$, d.h. Menge aller Worter über $\sum$**

$(\sum)^i$**...Menge aller Wörter der länge i über $\sum$**

Bsp:

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

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

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

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

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

Operationen auf Wörter

Für zwei Wörter $u = x_1 x_2 x \ldots x_n$ und $u = y_1 y_2 \ldots y_k$ mit $x_{i2}, y_{i2} \in \sum$ ist $u \circ v = x_1 x_2 \ldots x_n y_1 y_2 \ldots y_k$ die Konkatinetion von u und v Schreibweise: $w = uv$ oder auch $w = u \circ v$ (wie oben gezeigt).

Für ein Wort $w = x_1 x_2 \ldots x_n,\ x_i \in \sum$ ist das Spiegelwort wie folgt definiert:
$Sp(w) = x_n x_{n-1} \ldots x_2 x_1 =_{df} w^R$

Potenz von Wörtern

Es sei $w = x_1 \ldots x_n,\ x_i \in \sum$ 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 $(\sum)^*$)

$T \subseteq (\sum)^* \backslash (\sum)^*$ **... Teilwortrelation**

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

"u ist Teilwort von v"

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

"a ist Präfix von b"

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

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

"a ist Suffix von b"

Bsp.: (BUMM, KABUMM)

Behauptung: T und P sind reflexiv, transitiv, asymetrisch

1.1.2 Formale Sprache

Definition 1.1.2.1

Jede $A \subset (\sum)^*$ heißt Formalsprache über $\sum$

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

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

$A_2 := \{w \in \{a, b\} w\ beginnt\ mit\ a\}$

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

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

$A \ldots$ 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

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

und somit $L_1, L_2 \in (\sum)^*$ mit $\sum = \sum_1 \sum_2$

Dann sind auch:
$L_1 \cup L_2$
$L_1 \cap L_2$
$L_1 \backslash L_2$
$L_1 = \sum_1 \backslash L_2 = \{w \in \sum_1^* | w \in L_1$
$L_1 \times L_2$

formale Sprachen

Definition 1.1.2.2

Es seien $L_1 \subseteq \sum_1^*, L_2 \sum_1^*$

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

die Verkettung (Konkatenation) der Sprachen L_1 und L_2

Definition 1.1.2.3

Es sei $L \subseteq \sum_1^*$

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

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

und

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

Bemerkung:

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

Bsp.:

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

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

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

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

$\sum = \{a\}$ $A = \{aa\}$

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

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

$A = \{ba, umpf\}$

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

$a\{a, b\}^* \cup \{a, b\}^* b$ ... 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 <u>nicht</u> terminale Symbole
3. $S \in N$ Startsymbole

4. $R \subseteq (E \cup N)^* N (\Sigma \cup N)^* \times (\Sigma \cup N)^* \ldots$ Regelmenge

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

Bsp:
$G = \{\{a, b\}, \{s, x, y, z\}, S, R_1\}$
$R_1 = \{S \to SX, S \to bY, Y \to aZ, Z \to b >, Z \to b\}$

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

$N = \{SATZ, SUBJEKT, PRÄDIKAT\}$
$S = SATZ$
$R = \{SATZ \to SUBJEKT\ PRÄDIKAT \ldots\}$

Definition 1.2.2

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

w' heißt unmittelbare Ableitung von w bezüglich G

$w \overset{Ableitbar}{\vdash_G} w' \leftrightarrow$

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

w' heißt die Ableitung bezüglich w

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

1. $w = w'$ <u>oder</u>
2. $\underset{n \in \mathbb{N} w_0}{V} \underset{w_1}{V} V \ldots \underset{w_n}{V}$ wobei gelten muss:
   $w_0 = w$
   $w_n = w'$
   $w_0 \vdash_G w_1$
   $w_1 \vdash_G w_2 \ldots$
   $w_{n+1} \vdash_G w_n$

Schreibweise

$w \vdash_G w'$ Ableitung der länge n

$\vdash_G, \vdash_G^*$ sind bniäre 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 $L = \{w \in \Sigma^* | S \vdash_G^* w\}$

heißt die von G erzeugte Sprache

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

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 $\rightarrow \mathscr{L}(G_1) = \{b(ab)^n | n \geq 1\}$

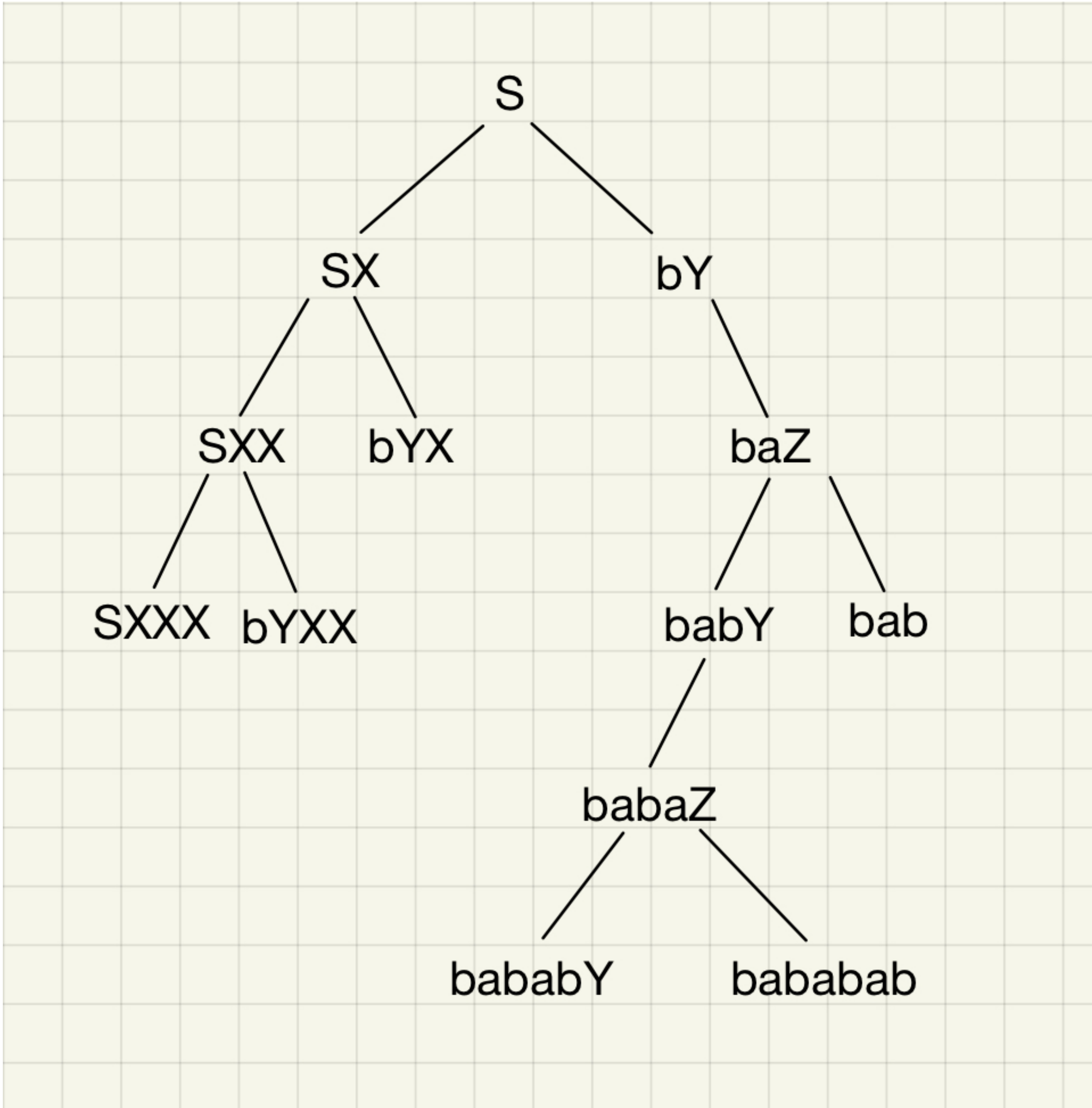
$B = \{w \in \{a, b\}^* | w \; endet \; auf \; a\}$

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

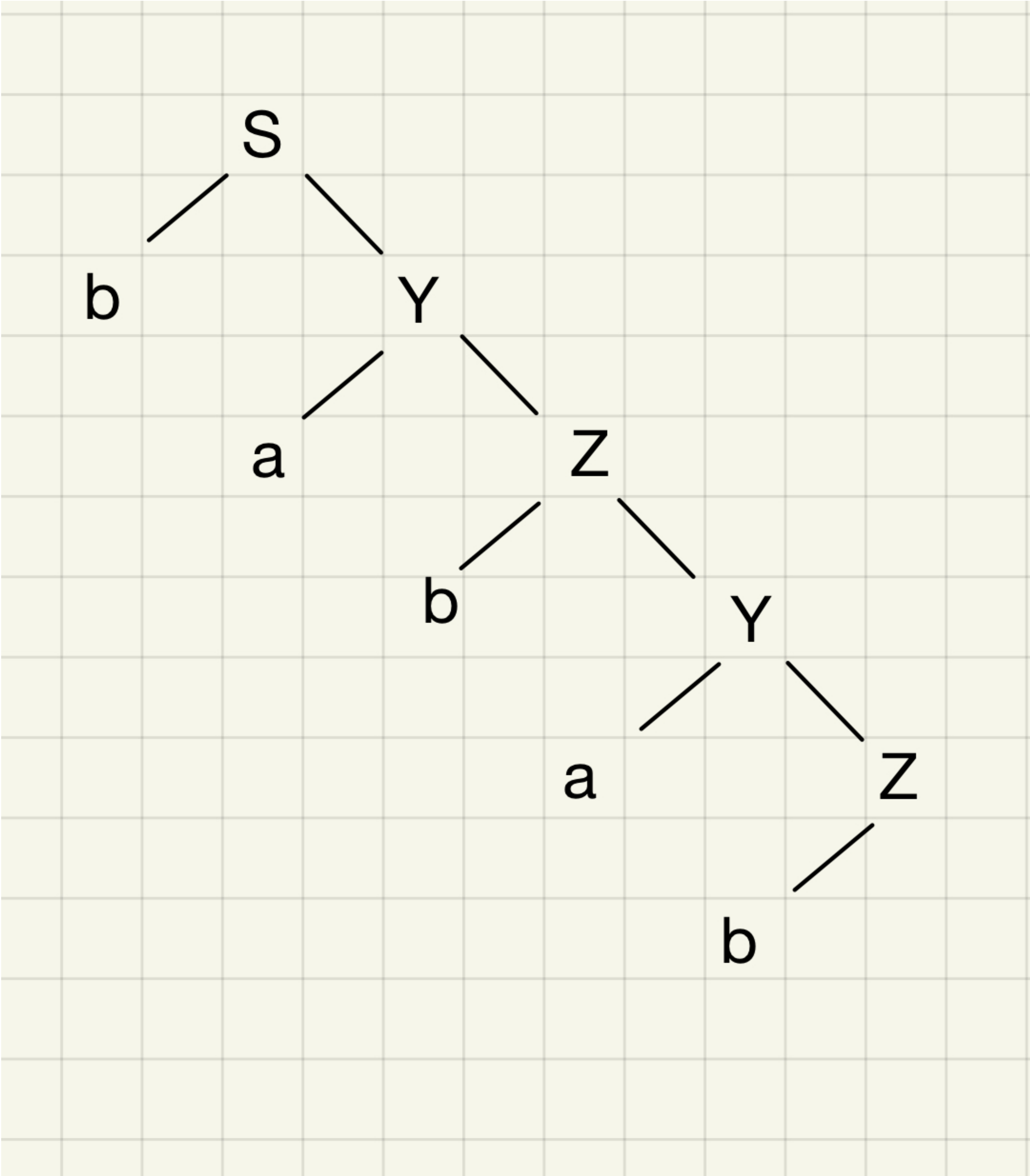
$R = \{S \rightarrow a | aS | bS\}$

Ableitungsbaum

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



- Wörter aus $\mathscr{L}(G_1)$ Sind Blätter
  Syntaxbaum für ein Wort nur möglich, wenn für alle Regeln $p \to q$ $p \in N$ gilt.
  Syntaxbaum am Beispiel "babab":



Blätter von links nach rechts ergeben w.

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

Bsp $G_1' = (\{a, b\}, \{s, y, z\}, S, R)$

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

$\to$ erzeugt ebenfalls $\mathscr{L}(G_1)$

Definition 1.2.4

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

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 $v \in \Sigma \circ N$

<u>Bsp</u>:

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

die Terminale **stehen** <u>nur</u> links

Definition 1.3.2

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

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

Sprachfamilien

$\mathscr{L}\ldots$ Klasse der von Grammatik erzeugbaren Sprachen
$\mathscr{L}_1\ldots$ CS (Context-sensitive) Klasse der Kontextfreien Sprachen
$\mathscr{L}_2\ldots$ CF (Context-free) Klasse der Kontextfreien Sprachen
$\mathscr{L}_3\ldots$ REG (Regual) Klasse der Regulären Sprachen

Satz 1.3.3

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

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 $5\text{-}Tupel$
$\rightarrow M = (\Sigma, Z, \delta, Z_0, Z_E)$ mit folgenden Eigenschaften

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

Satz 2.1.3

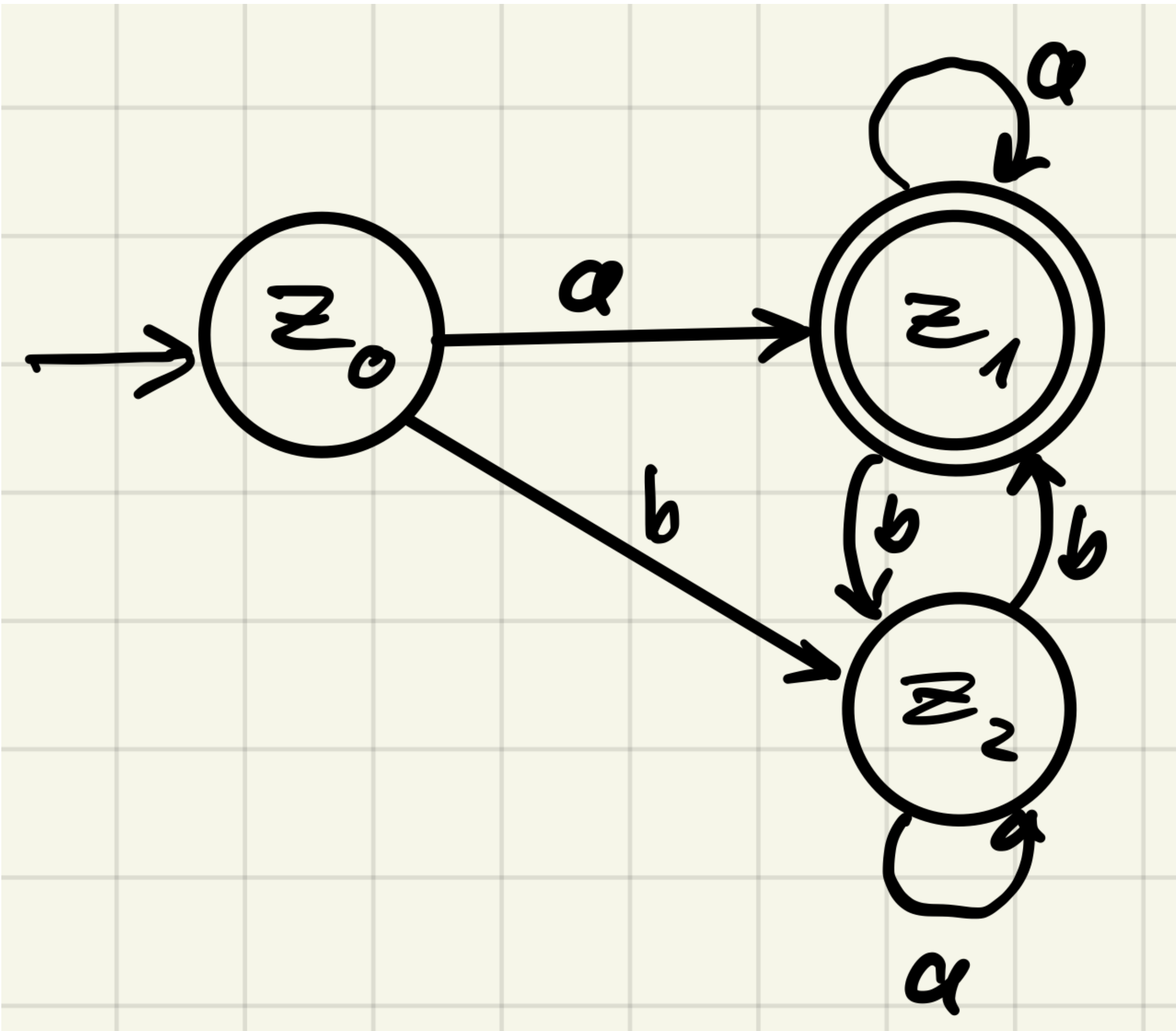
Jede Sprache die von einem DFA (DEA?) akzeptiert werden kann ist regulär.

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

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

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

Idee:



$Z_0 \to aZ_1 | bZ_2 | a$
$Z_1 \to aZ_1 | bZ_2 | a$
$Z_2 \to aZ_2 | bZ_1 | b$

$\delta(z_0, a) = Z_1$

$Z_0 \to aZ_1$
$Z_0 \to bZ_2$
$Z_0 \to a$

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

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

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

$w \in A\ \leftrightarrow w \in L(M)$
$\leftrightarrow$ es gibt eine Folge von Zuständen von M $Z_0, \ldots, Z_n$ 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 $Z_0, Z_1, \ldots, Z_n$ mit Z_0 als Startsymbol und
$Z_0 \vdash_G a_1Z_1 \vdash a_1a_2z_2 \vdash_G \ldots \vdash a_1a_2\ldots a_{n+1}Z_{n-1} \vdash a_1\ldots a_n$

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

2. Fall

folgt aus Fall 1 unter Berücksichtigung von Satz 1.3.4

$S \to 0S|1S|1X$
$X \to 1$

Definition 2.1.4

Es sei $M = (\Sigma, Z, \delta, Z_0, Z_E)$ ein DFA (DEA?) Dann mit
$R_M = \{(x, y) \in \Sigma^* \times \Sigma^*|\delta^*(Z_0, y)\}$

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

<u>Reflexivität</u>

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

<u>Symmetrie</u>

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

<u>Transitivität</u>

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 : "$\Rightarrow$"

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

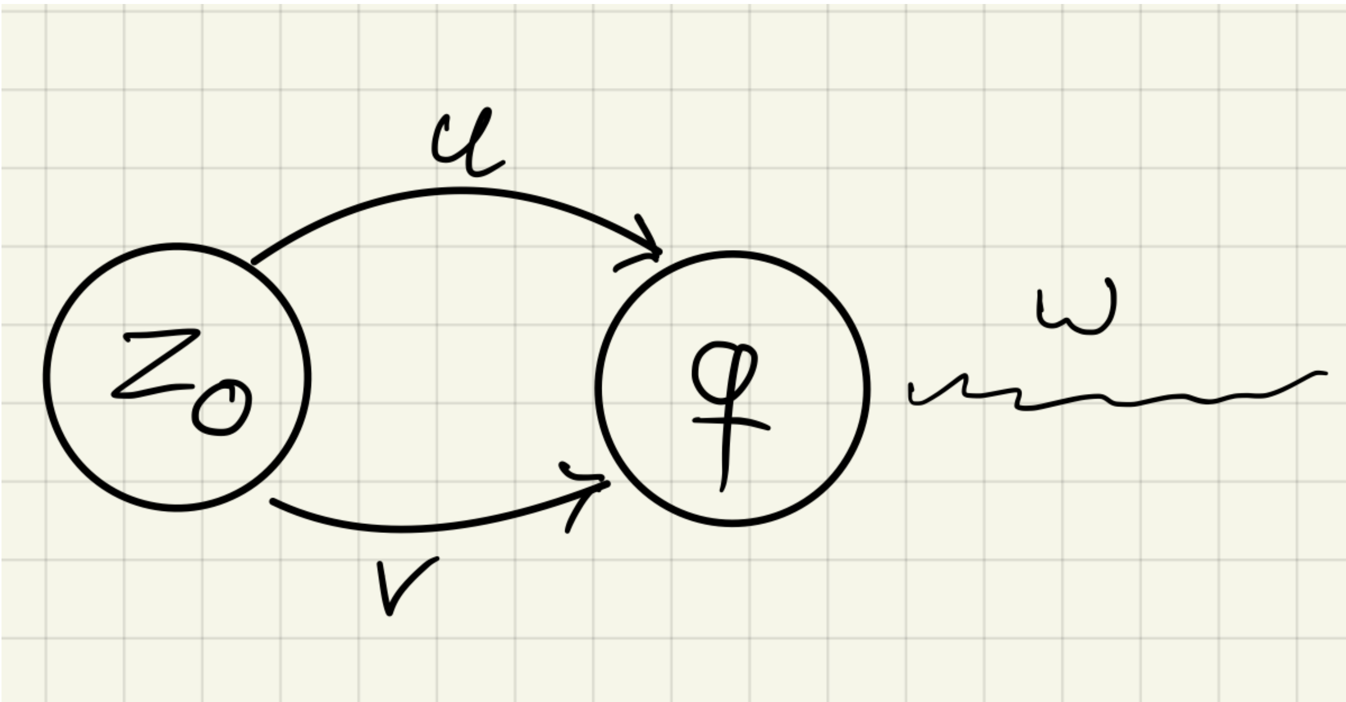
$\Rightarrow \bigwedge_{u \in \Sigma^*} \delta^*(Z_O, xu) = \delta^*(Z_O, yu)$
$\Rightarrow \bigwedge_{u \in \Sigma^*} (xu, yu) \in R_M$
$A \rightarrow B$
$\Leftarrow$"$Kontraposition$" $\neg A \rightarrow \neg B$

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

Das gilt für $u = \lambda$

Konstruktion deterministischer endlicher Automaten

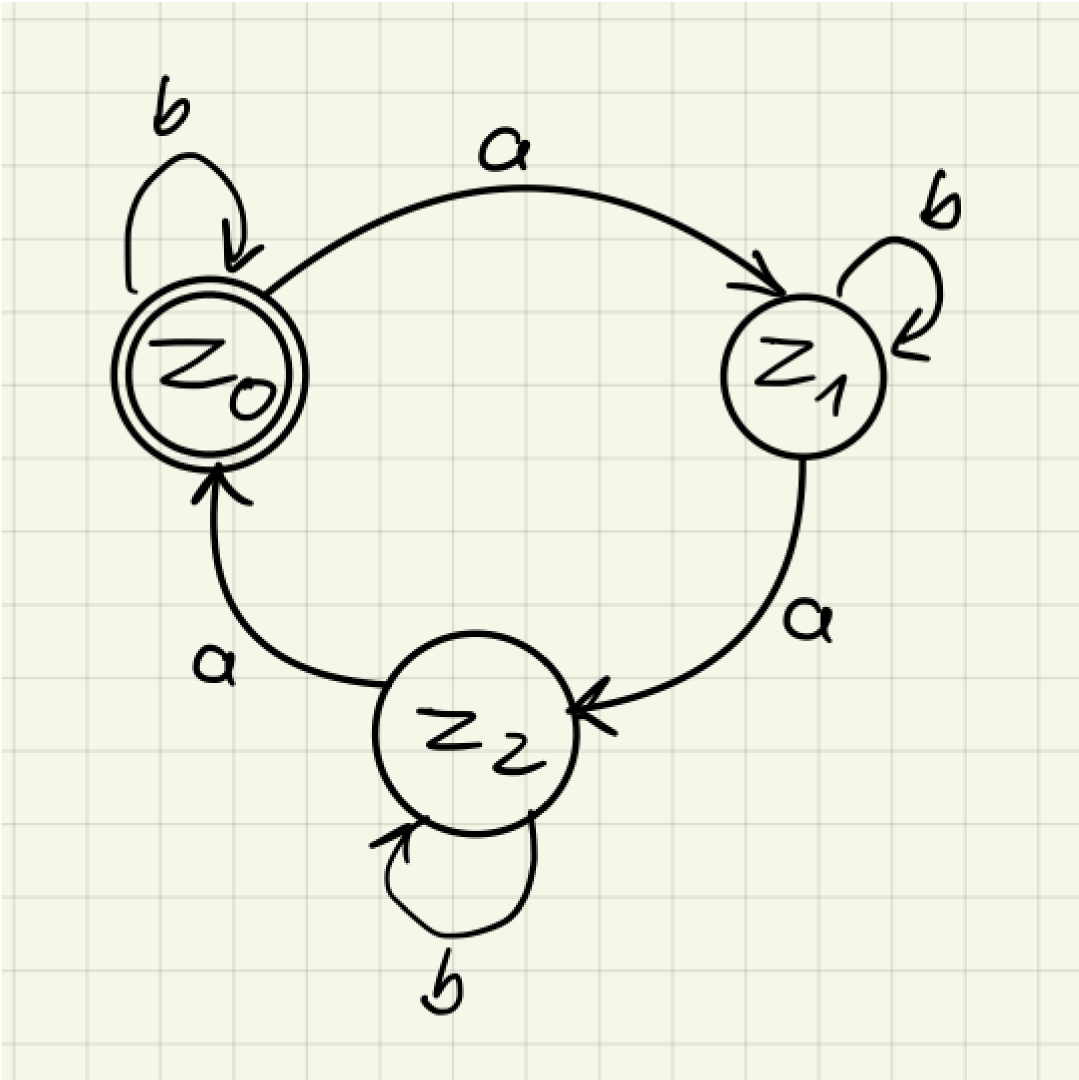
Seien $u, v \in \Sigma^*$ mit $(u,v) \in R_M$, d.h. $\delta^*(Z_O, z) = \delta^*(Z_O, v) = q$



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 Worteil endet $(\delta^*(Z_O, u))$. Die für die Zugehörigkeit bzw. "nicht"-Zugehörigkeit eines Wortes u zu L relevanten Informationen müssen in q gespeichert werden.

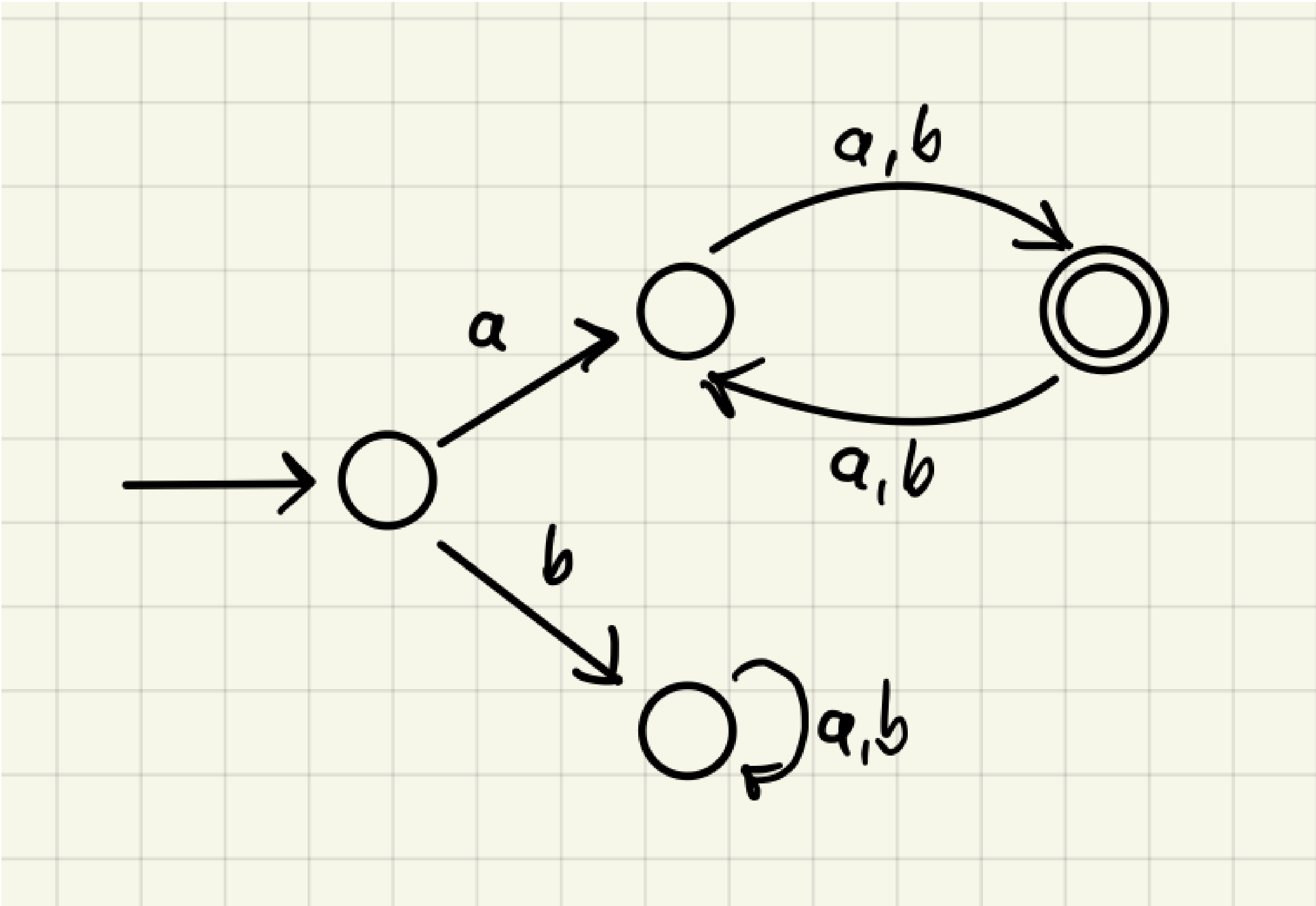
$aababbbba$

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



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

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

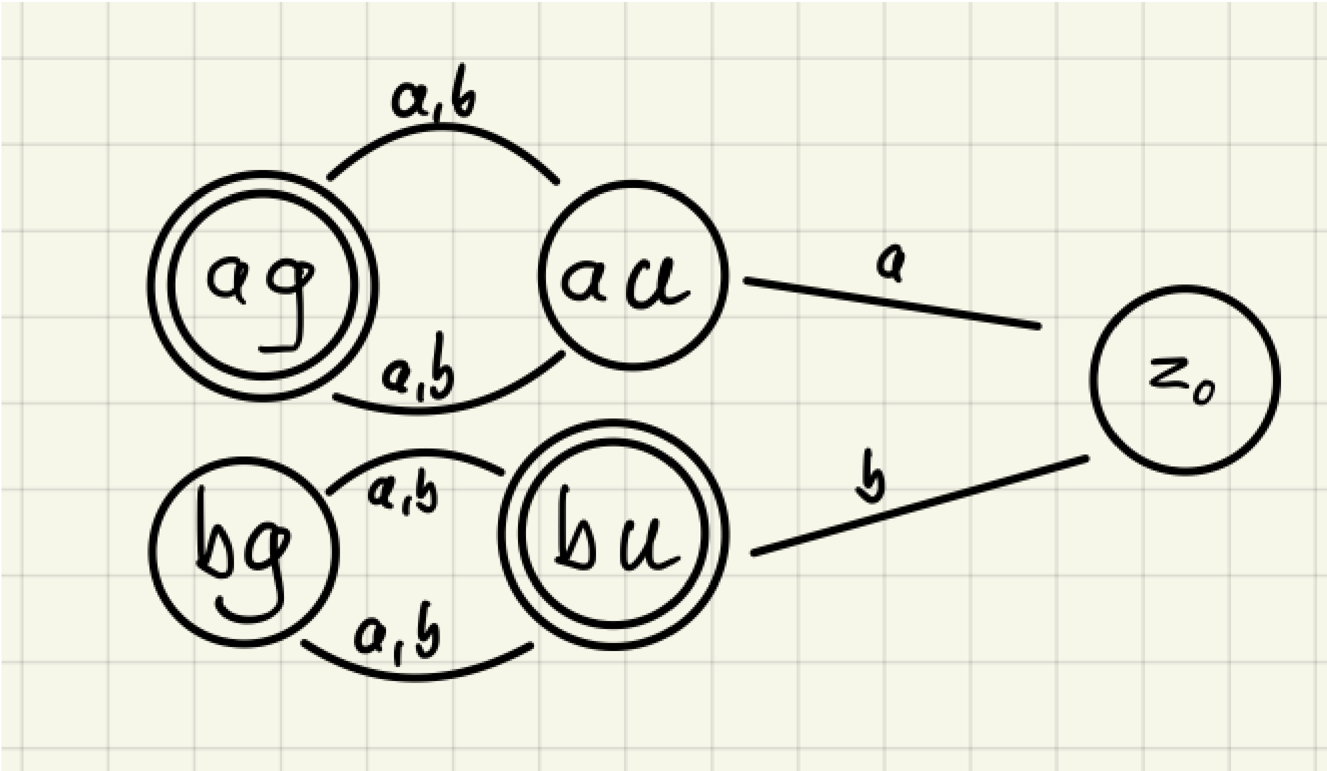


$L_3 = \{w \in \{a,b\}^* | w\ beginnt\ mit\ a\ \Leftrightarrow\ |w| \equiv_2 0\}$

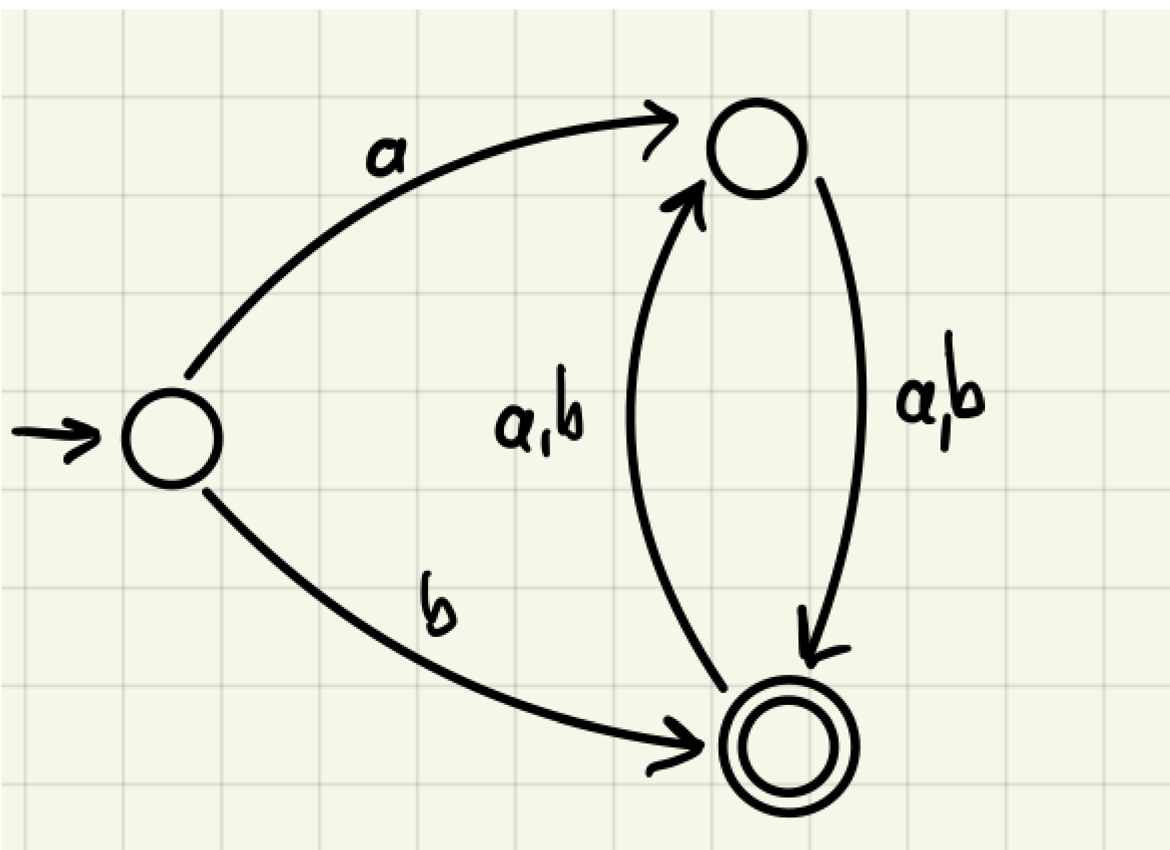
Beides gilt oder Beides gilt nicht !

<u>Bsp</u>: $aa \in L_3$, beginnt mit a und gerade länge$\Rightarrow 1 \Leftrightarrow 1 = 1$
$\lambda \notin L_3$,beginnt <u>nicht</u> mit a und gerade länge $\Rightarrow 0 \Leftrightarrow 1 = 0$
$b \in L_3$, beginnt <u>nicht</u> mit a und hat <u>nicht</u> gerade länge$\Rightarrow 0 \Leftrightarrow 0 = 1$
$baaaa \in L_3$, beginnt <u>nicht</u> mit a und hat <u>nicht</u> gerade länge$\Rightarrow 0 \Leftrightarrow 0 = 1$

Vorlesungsvariante:

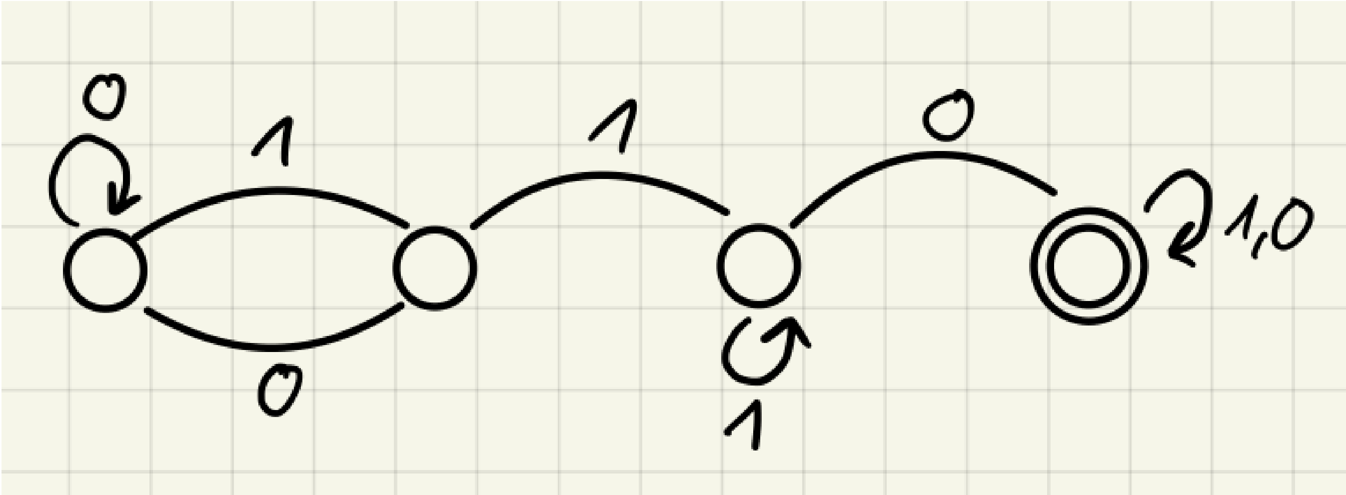


Wir haben eine optimierte Variante die richtig sein sollte!(Kein Gewähr):



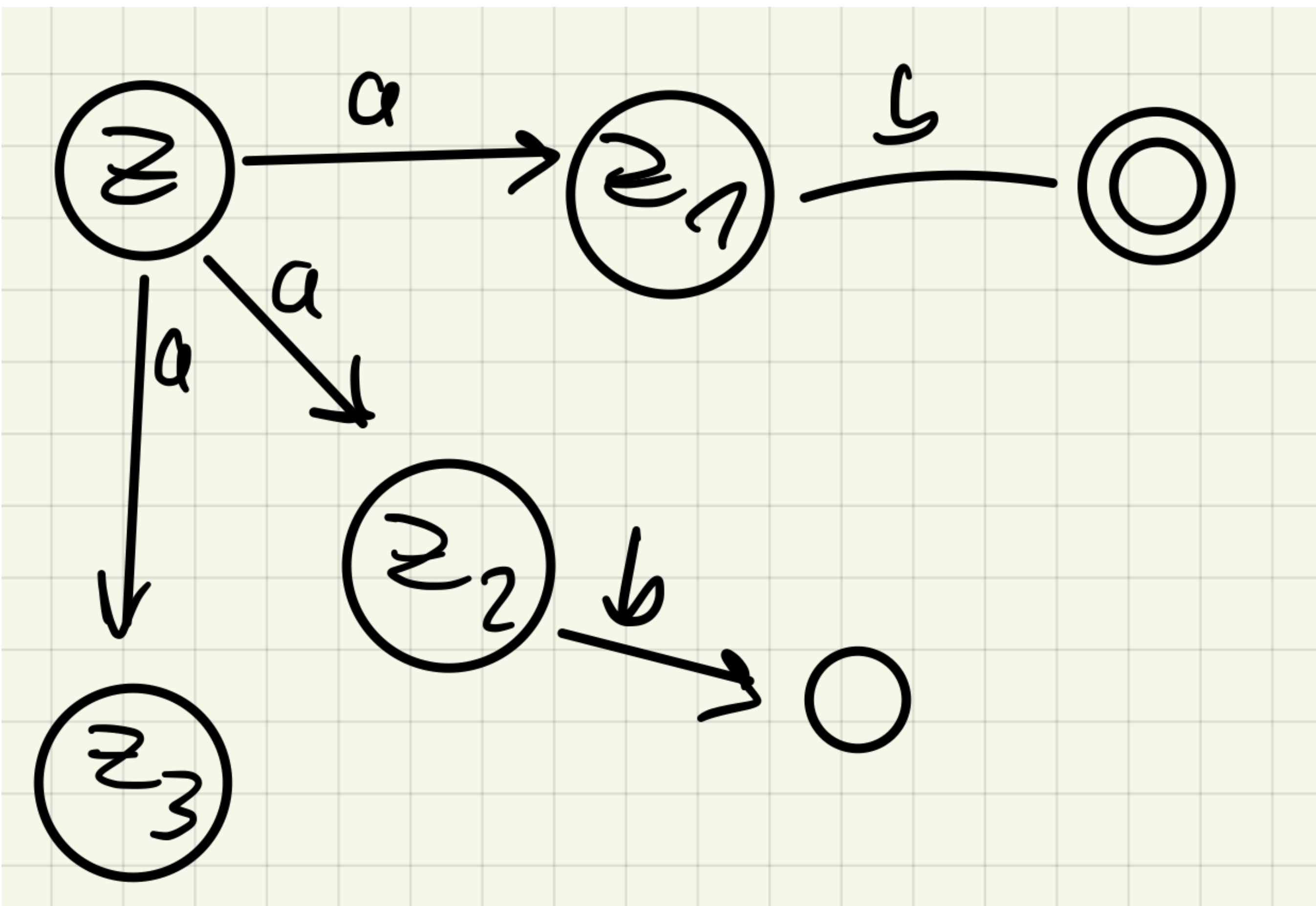
Letztes Beispiel:

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



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. $\delta: \ Z \times \Sigma \to P_{otenzmenge}(Z)$ ... Überführungsfunktion
4. $S \subseteq Z$ ... Menge Startzustände
5. $Z_E \subseteq Z$ ... Menge Endzustände

Definition 2.2.2

Die erweiterte Überfürhugnsfunktion:

$\delta^* : P(Z) \times \Sigma^* \to P(Z)$, sei wie folgt definiert: Für alle $\Sigma \subseteq Z, a \in \Sigma, w \in \Sigma^*$ ist
$\delta^*(\tilde{z}, \lambda) = \tilde{z}$
$\delta^*(\tilde{z}, aw) = \bigcup\limits_{z \in \tilde{Z}} \delta^*(\delta(z, a), w)$
$\delta(z, a)$ ... Menge von z nach Abarbeitung von a erreichabren Zuständen

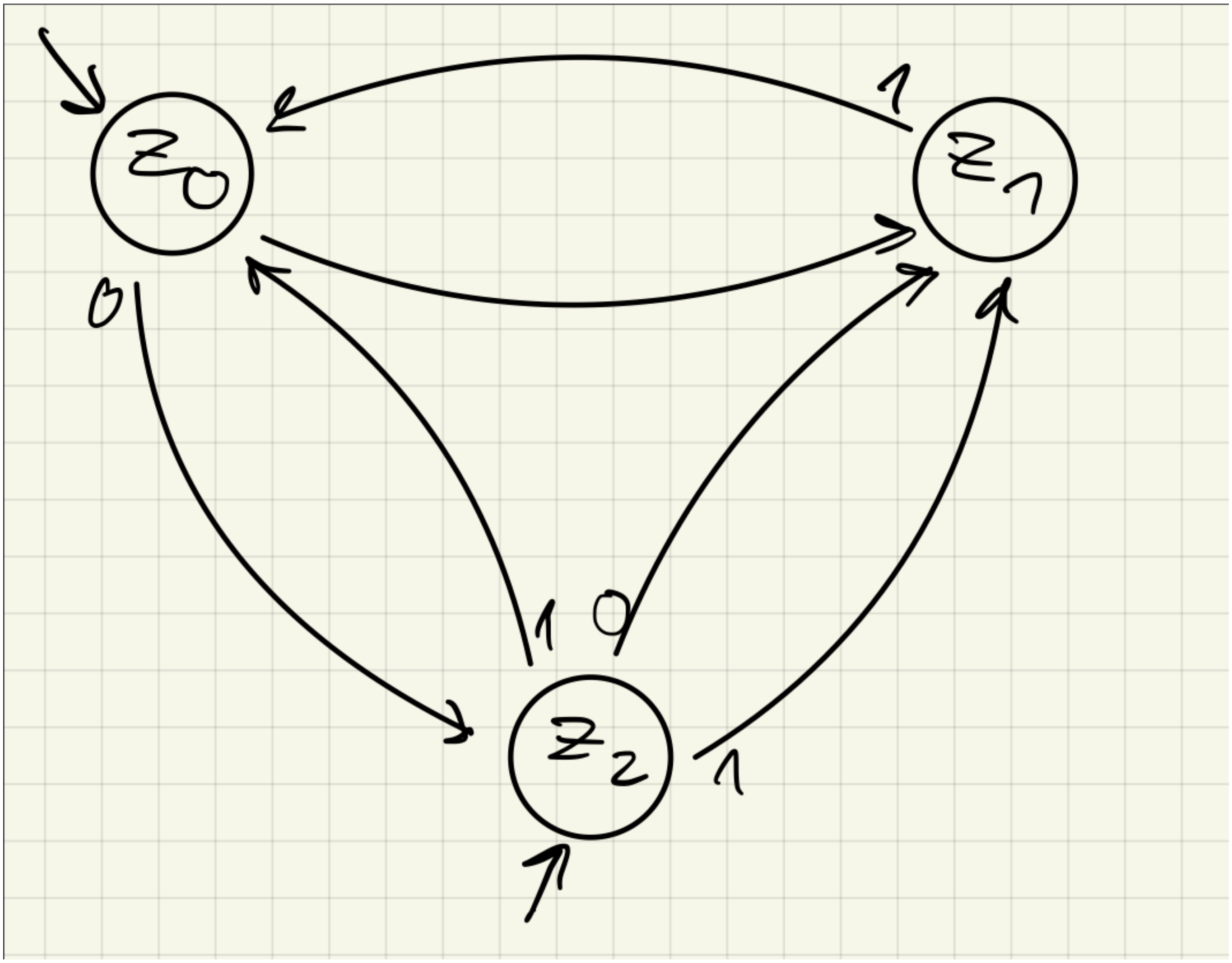
$\delta^*(\tilde{z}, w)$ ... Menge der Zustände die erreicht werden, wenn in $z \in \tilde{Z}$ gestartet wird und w abgearbeitet wird.

Definition 2.2.3

Die von einem NFA $M = \{\Sigma, Z, \delta, S, Z_E\}$ akzeptierte Sprache ist
$L(M) = \{w \in \Sigma^* | \delta^*(s, w) \cap Z_E \neq \emptyset\}$

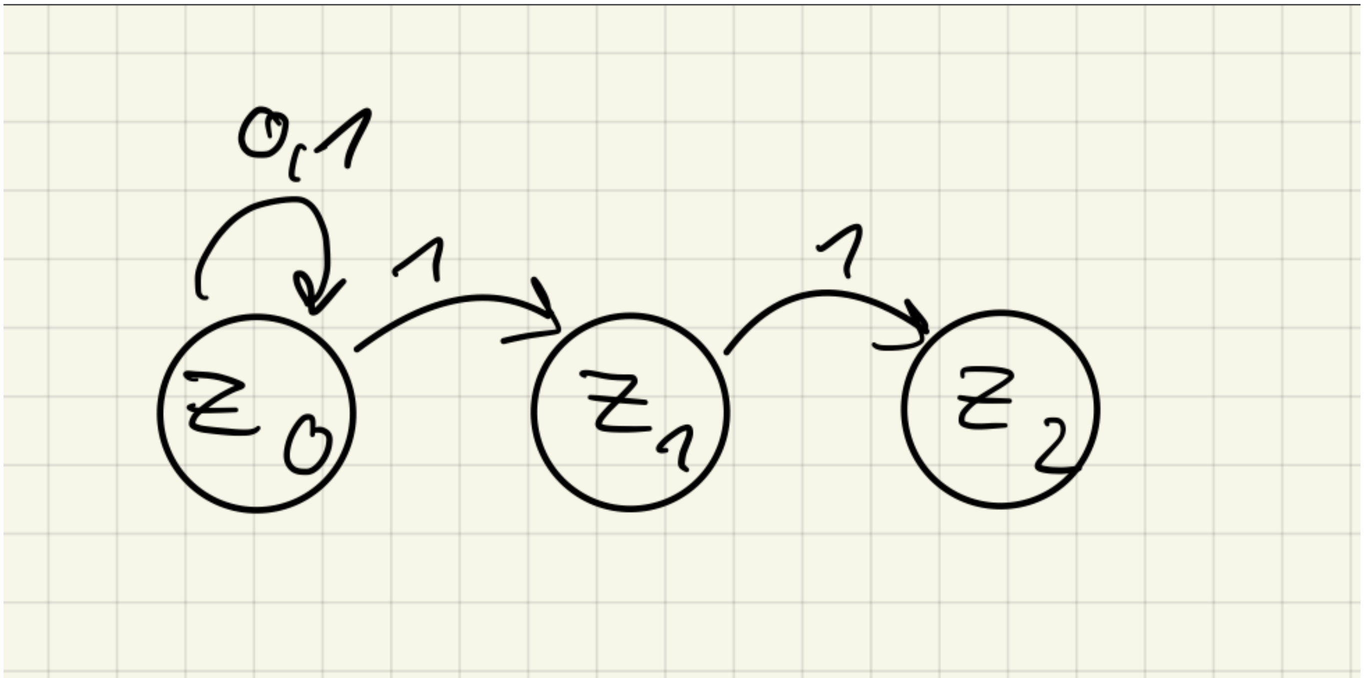
Bsp:

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



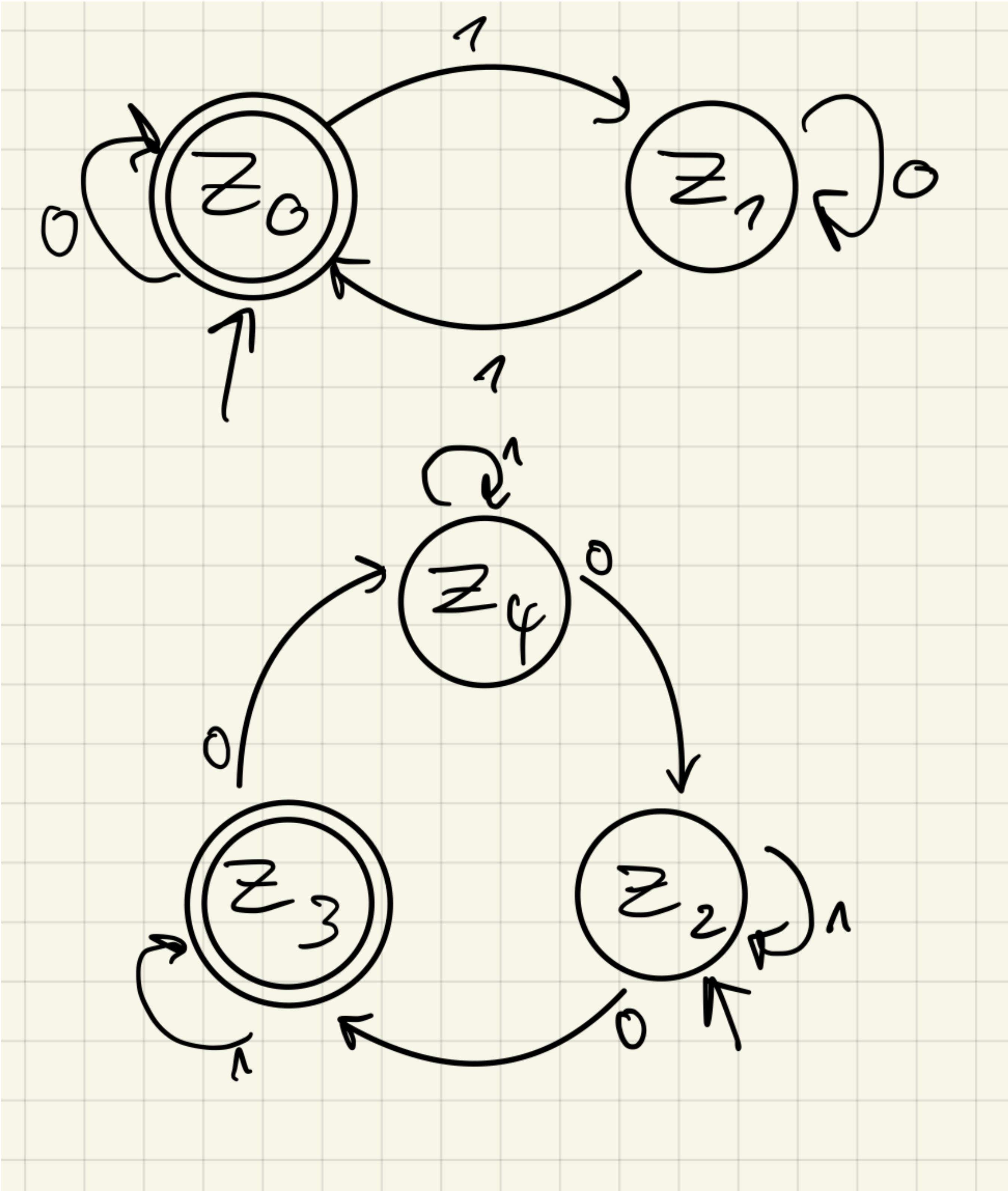
$\delta^*(\{z_0, z_2\}, 100)$
$= \delta^*(\underbrace{\delta(z_0, 1)}_{=\emptyset}, 00) \cup \delta^*(\delta(z_2, 1), 00)$
$= \delta^*(\emptyset, 0) \cup \delta^*(\{z_2, z_1\}, 00)$
$= \bigcup_{z \in \emptyset} \delta^*(\delta(z, 0), 0) \cup \delta^*(\delta(z_0, 0), 0) \cup \delta^*(\delta(z_1, 0), 0)$
$= \delta^*(\{z_1, z_2\}, 0) \cup \underbrace{\delta^*(\{z_2\}, 0)}_{=\emptyset}$
$= \delta^*(\delta(z_1, 0), \lambda) \cup \delta^*(\delta(z_2, 0), \lambda)$
$= \delta^*(\{z_2\}, \lambda) \cup \delta^*(\emptyset, \lambda) = \{z_2\}$
$\Rightarrow$ 100 wird <u>nicht</u> akzeptiert

Bsp:



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

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



101000 . . . wird akzeptiert
100 . . . wird <u>nicht</u> 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 $A = \mathscr{L}(G)$. Wir betrachten folgenden NFA $M = (\Sigma, Z, \delta, S, Z_E)$
$Z = N \cup \{F\} \ F \notin N$
$S' = \{S\}$

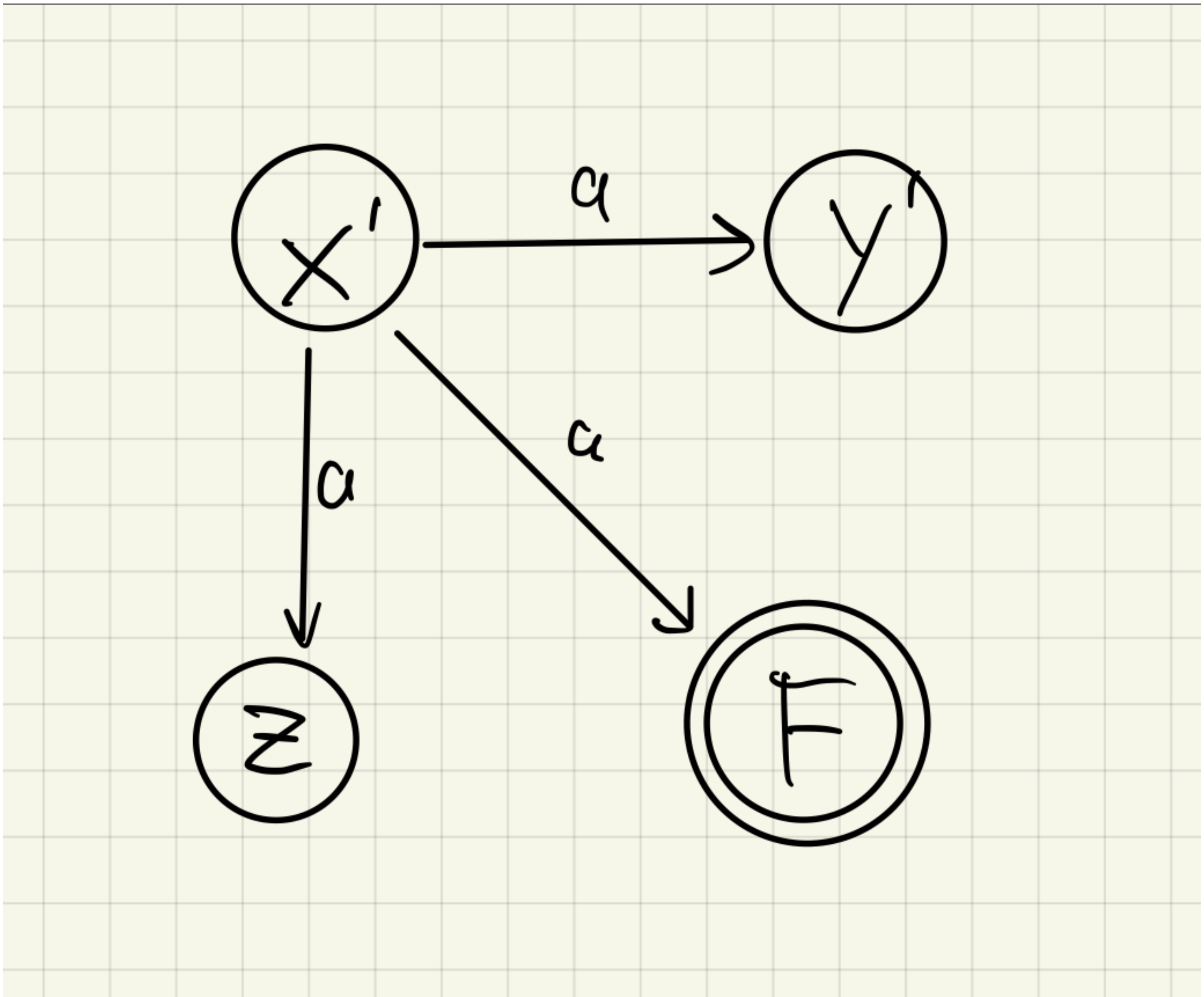
$$Z_E = \begin{cases} \{F\} & falls \ (S \to \lambda) \notin R \\ \{F, S'\} & falls \ (S \to \lambda) \in R \end{cases}$$

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

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

Idee in Grammatik

$X \to aY$
$X \to aZ$
$X \to a$



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

$\Leftrightarrow$ es gibt eine Folge von Nichtdeterminisitischen
$X_1, X_2, \ldots, X_{n-1} \in N$, so dass

$S \vdash_G a_1 X_1 \vdash_G a_1 a_2 X_2 \vdash_G \ldots \vdash_G a_1 a_2 \ldots a_{n-1} X_{n-1} \vdash_G a_1 \ldots a_n$
$\Leftrightarrow$ es gibt eine Folge von Zuständen

$X_1', X_2', \ldots, X_{n-1}' \in Z$ 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
$X_n' \in \delta(X_{n-1}', a_n)$
$\Leftrightarrow \; a_1 \ldots a_n \in L(M)$

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

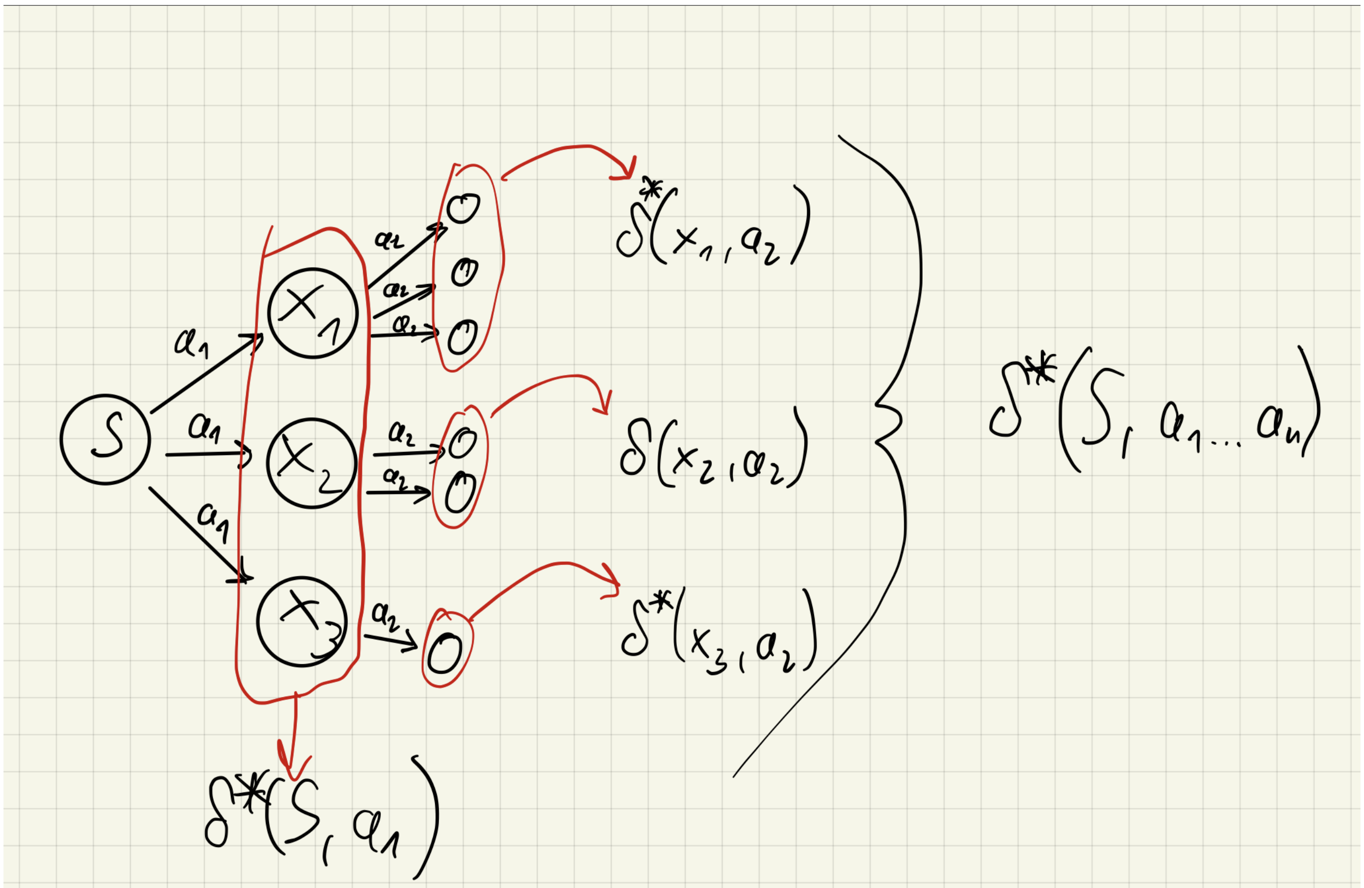
Satz 2.2.5

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

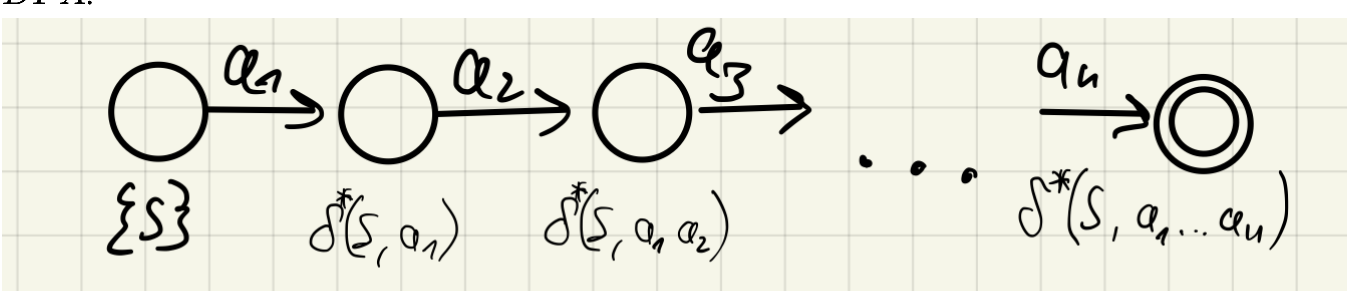
Beweis:

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

Idee: Abarbeitung von $w = a_1 ... a_n$
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 $Z' \in P(M)$ $Z' \in Z$

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

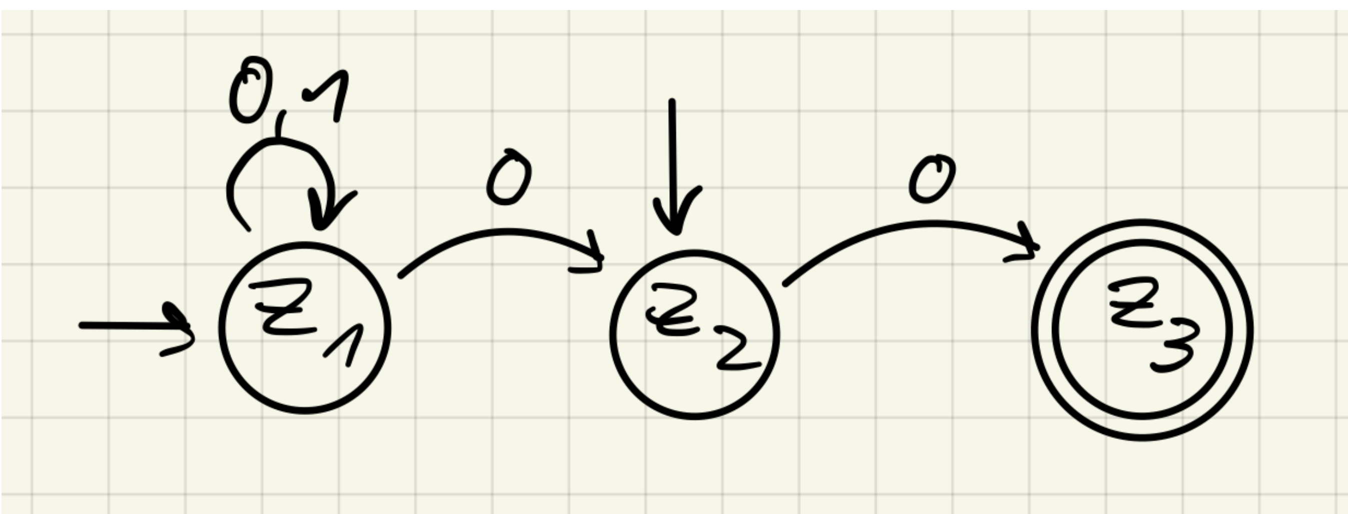
Für jedes Wort $w = a_1 ... a_n$ $a_1 \in \Sigma^*$ gilt

$w \in L(M) \Leftrightarrow \delta^*(S, w) \cap Z_E \neq \emptyset$
$\Leftrightarrow$ Es gibt eine Folge von Teilmengen $Z_1 ... Z_n$ von Z mit $\delta^*(S, a_1) = Z_1$
$\delta^*(Z_1, a_2) = Z_2 \ldots \delta(Z_{n-1}, a_n) = Z_n$ und $Z_n \cap Z_E \neq \emptyset$
$\Leftrightarrow$ es gibt eine Folge von Zuständen $Z_1, \ldots, Z_n$ von M' mit
$\delta'(S', a_1) = Z_1, \delta'(Z_2, a_2) = Z_2 ... \delta'(Z_{n-1}, a_n) = Z_n$ und $Z_n \cap Z_E \neq \emptyset$
$\Leftrightarrow \delta^*(S, a_1 ..., a_n) \in Z'_E$

<u>Bsp</u>:

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

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



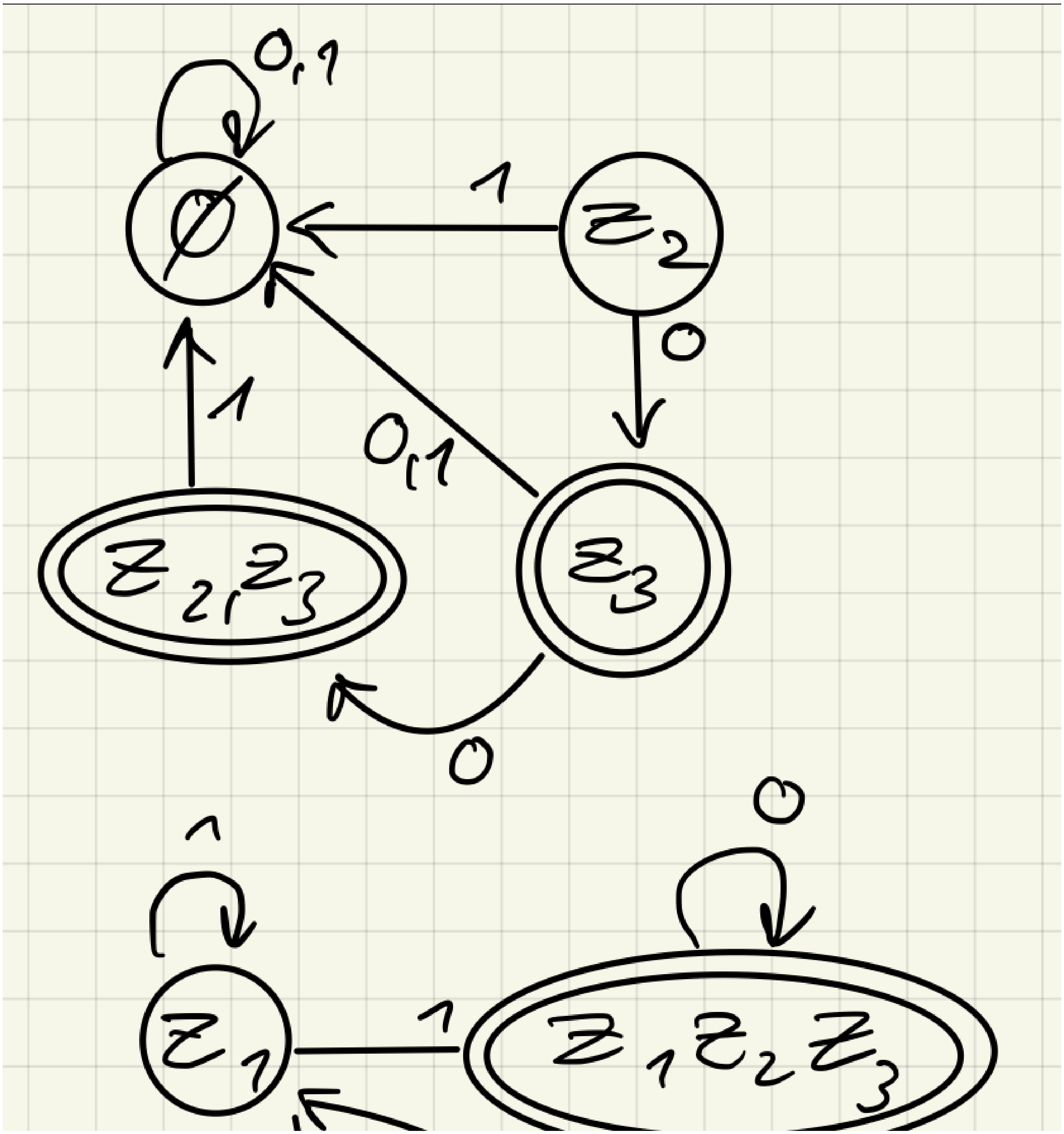
$DFA\ M' = (\{0, 1\}, Z', \delta', S', Z_E)$

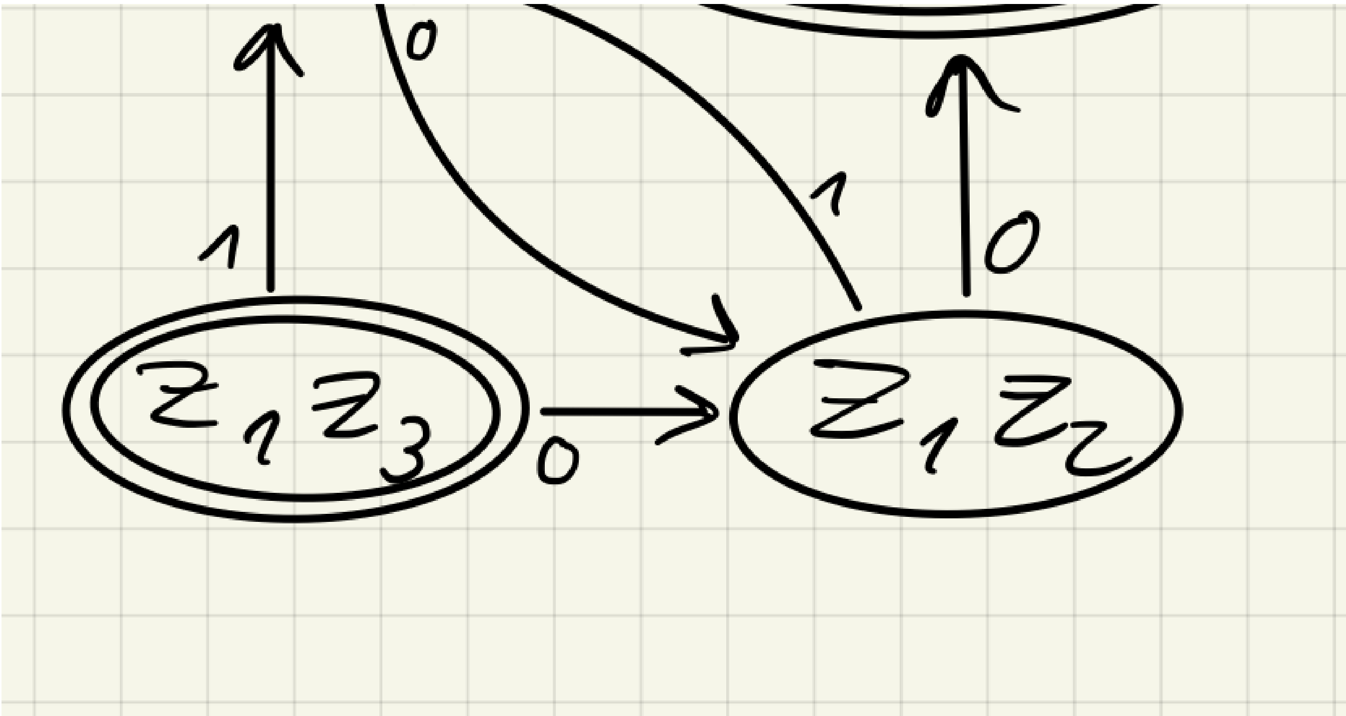
$S' = \{z_1, z_2\} = q_4$

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

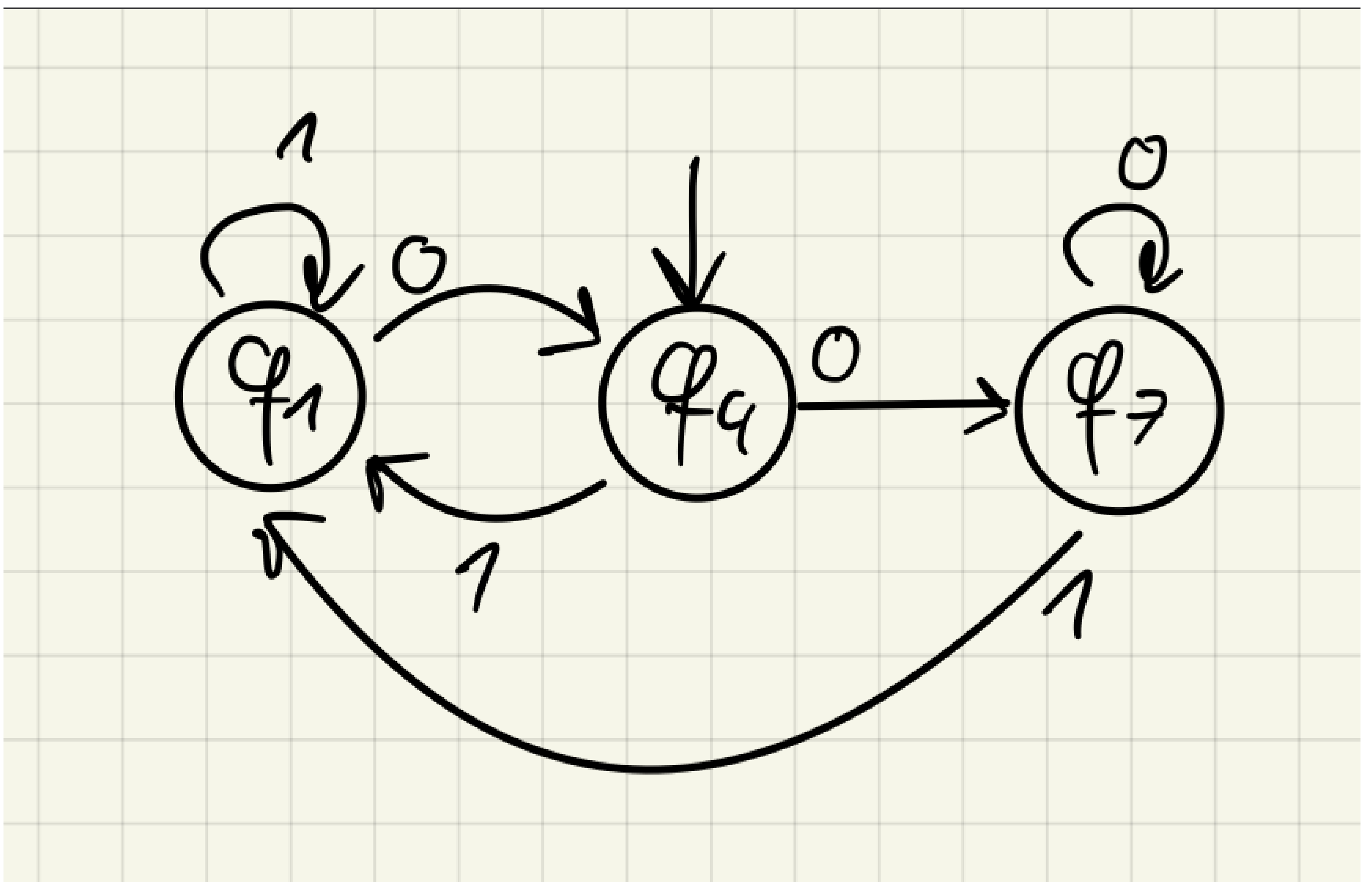
$$Z' = P(\{z_1, z_2, z_3\}) = \{\emptyset, \{z_1\}, \{z_2\}, \{z_3\}, \{z_1, z_2\}, \{z_1, z_3\}, \{z_2, z_3\}, \{z_1, z_2, z_3\}\}$$

	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 <u>alle</u> 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 regular <u>aber</u> Kontextfrei

Definition 2.3.2

Es sei γ ein regulärer Ausdruck über $\Sigma, \gamma \in RA(\Sigma)$
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. $$L(\gamma) = \begin{cases} L(\alpha) \cdot L(\beta) & , falls\ \gamma = \alpha\beta \\ L(\alpha) \cup L(\beta) & , falls\ \gamma = (\alpha + \beta) \\ L(\alpha) & , falls\ \gamma = (a)^* \end{cases}$$

   <u>Bsp</u>: $\Sigma = \{a, b\}$

$L\Big(\big((a+b)\big)^*\Big) = \{a, b\}^*$

$L\Big(a\big(a+b\big)\big)^*abba\big(a+b\big)\big)^*\Big) = \{a\}\{a,b\}^*\{abba\}\{a,b\}^*$

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

Lemma 2.3.3

Jede endliche Sprache kann durch einen regulären Ausdruck beschrieben werden.

Beobachtung:

Sei

$$\begin{aligned} &w_1 = a_{11}a_{12}\dots a_{1m_1} \\ &w_2 = a_{21}a_{22}\dots a_{1m_2} \\ &\vdots \\ &w_n = a_{n1}a_{n2}\dots a_{nm_n} \end{aligned}$$

Idee:

$$\begin{aligned} &\Big(((w_1+w_2)+w_3)\dots+w_n\Big) \\ &\gamma\Big(((a_{11}a_{12}\dots a_{1m_1}+a_{21}a_{22}\dots a_{2m_2}))+\dots+a_{n1}a_{n2}\dots a_{nm_n}\Big) \end{aligned}$$

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 $k \subseteq \{1, \dots, \}$

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

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

(keiner der erreichbaren Zwischenzustände hat Index der größe k)

$$k = 0 \quad und \quad i \ne j$$

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

$$k = 0 \quad und \quad i = j$$

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

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

Beobachtung:

Induktion über k:

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

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

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

$k \to k + 1$ Sei $i, j \in \{1, \ldots, n\}$

beliebig aber fest gewählt

Offenbar ist

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

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

für $L^k_{i,j}, L^k_{i,k+1}, L^k_{k+1,k+1}, L^k_{k+1,j}$

Diese gilt es nach I.V.:

Dann ist

$$L^{k+1}_{i,j} = L\Big((\alpha^k_{i,j}\alpha^k_{ik+1}(\alpha^k_{k+1,k+1})^*\alpha^k_{k+1,j})\Big)$$

Nun lässt sich $L(M)$ ausdrücken als

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

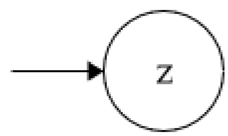
Sei $i_1, i_2, \ldots, i_m$ die Indizes der Endzustände so ist

$$L(M) = \left(\alpha^n_{1,i_1} + \left(\alpha^n_{1,i_1} + (\ldots(\alpha^m_{1,i_1})\ldots)\right)\right)$$

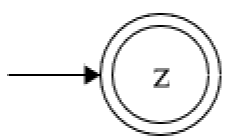
"$\Leftarrow$" Idee: Die Induktive Definition von $RA(\Sigma)|$ Kann mit NFA's noch vollzogen werden.

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

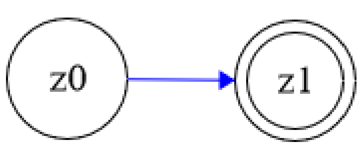
1.) Ist $\gamma = \emptyset$ $M = (\Sigma, \{z_0\}, \delta, \{z_0\}, \emptyset)$



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) $\gamma = \alpha + \beta$
$Z = Z_1 \cup Z_2$

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

b) $\gamma = \alpha \cdot \beta$
$Z = Z_1 \cup Z_2$

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

$Z_E = Z_{E2}$

Beweis Satz 2.3.4

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

Zu zeigen: es bigt einen NFA der L akzeptiert

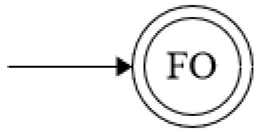
c) $\gamma = (\alpha)^*$ NFA $M = (\Sigma, Z, \delta, Z_0, Z_E)$

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

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

$\lambda \notin L(M_1)$,

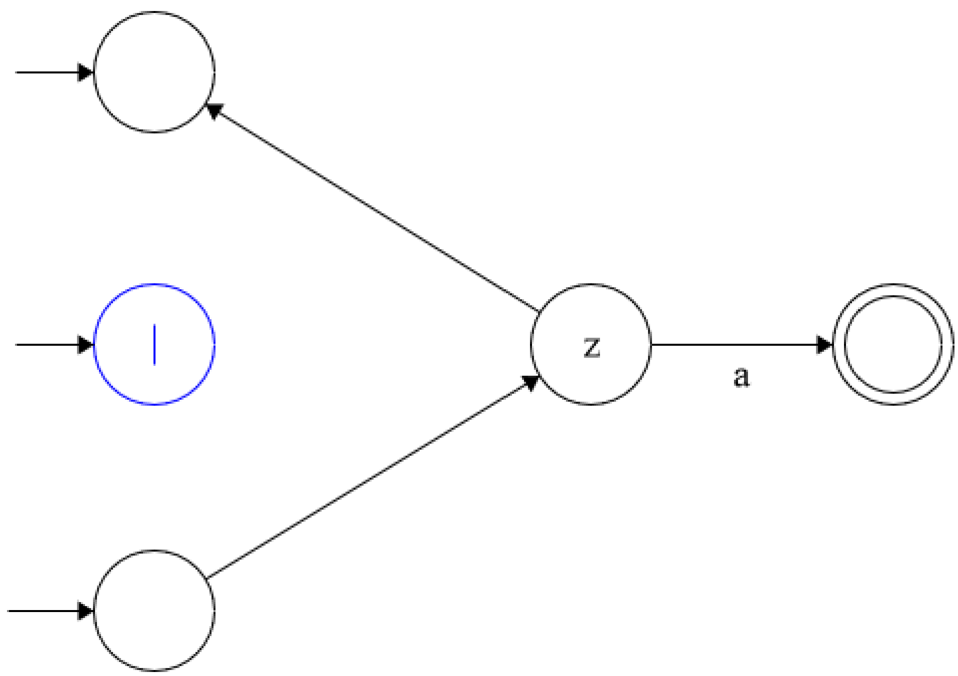
$M'_1 = (\Sigma, Z_1 \cup \{F0\}, \delta', Z_0 \cup \{F0\}, Z_E \cup \{F0\})$



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

$M_1' = \Sigma, Z', \delta'', Z_0', Z_E'$ mit

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



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.

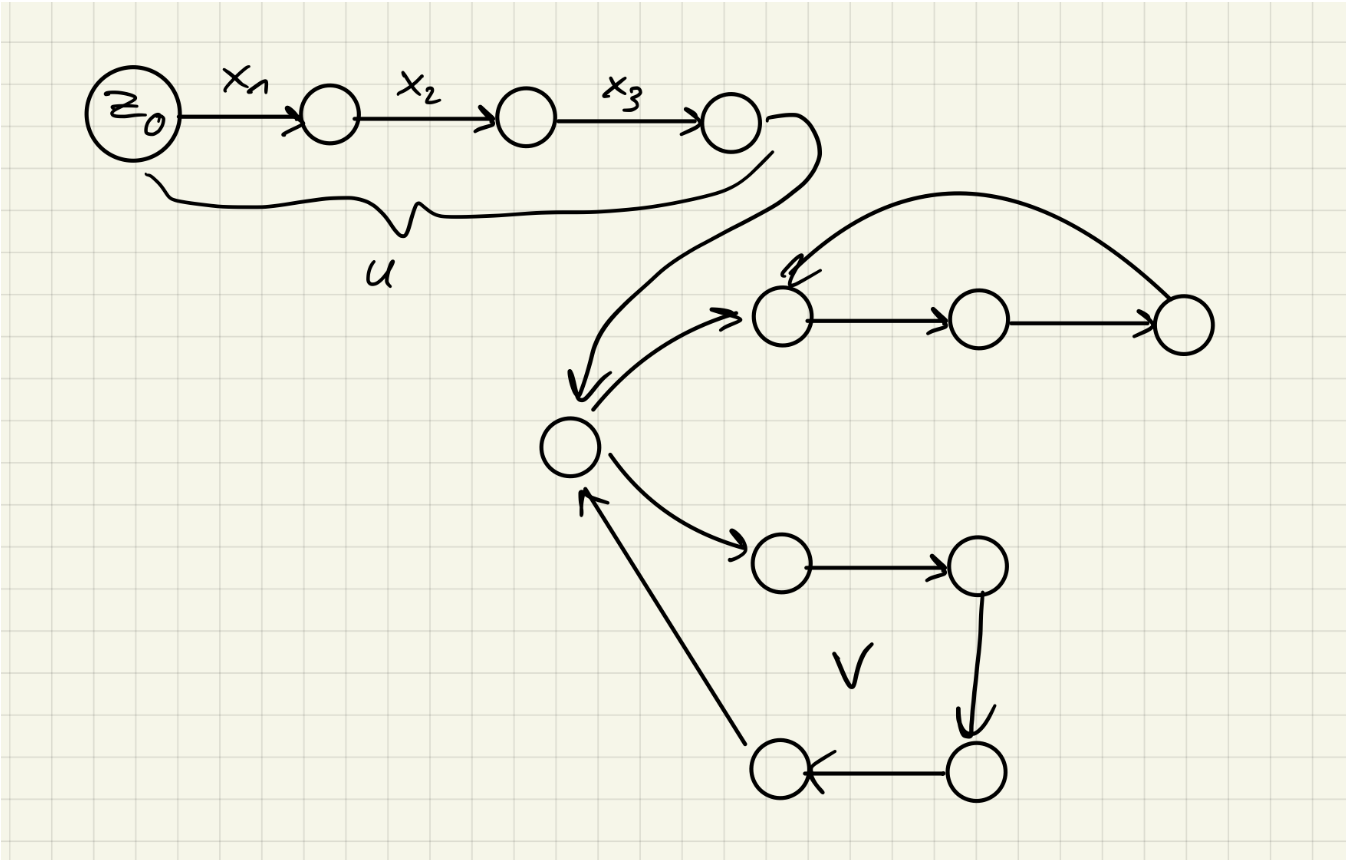
Satz 2.4.1 Pumping Lemma

Sei A eine reguläre Sprache. Dann gibt es ein $n \in \mathbb{N}$, so dass sich alle Wörter $x \in A$ $|x| \geq n$ 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 nciht 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

$$uvw \in A \Leftrightarrow uw \in A \Leftrightarrow uv^iw \in A$$

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

<u>Bsp</u>:

1.

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

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 $|v| \geq 1$, $|uv| \leq n$, $uv^iw \in A$ für alle $i > 0$. Wähle $x = a^nb^n$ $|v| = 2n \geq n$.

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.

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

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 $v = 0^k$ $1 \leq k \leq n$
$n^2 = |x| = |uvw| \leq |uv^2w \leq |uvw| + |v| = n^2 + k \leq^2 +n \leq n^2 + n + 1 = (n+1)^2$
$\Rightarrow uv^2 \notin B$. Damit ist $B \notin REG$

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

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

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

2.5 Minimalautomaten und Äquivalenzrelationen

Es kann jeder Sprache eine Äquivalenzrelation zugeordnet werden

Definition 2.5.1

Sei $A \subseteq \Sigma^*$
Dann ist $R_A = \{(x,y) \in \Sigma^* \times \Sigma^* | \bigwedge_{z \in \Sigma^*} (xz \in A \Leftrightarrow yz \in A)\}$
$\Leftrightarrow$ beim Anhängen beliebiger Wörter an x und y dann verhalten sie sich bezüglich Mitgliedschaft zu A gleich.

R_A ist Äquivalenzrelation:

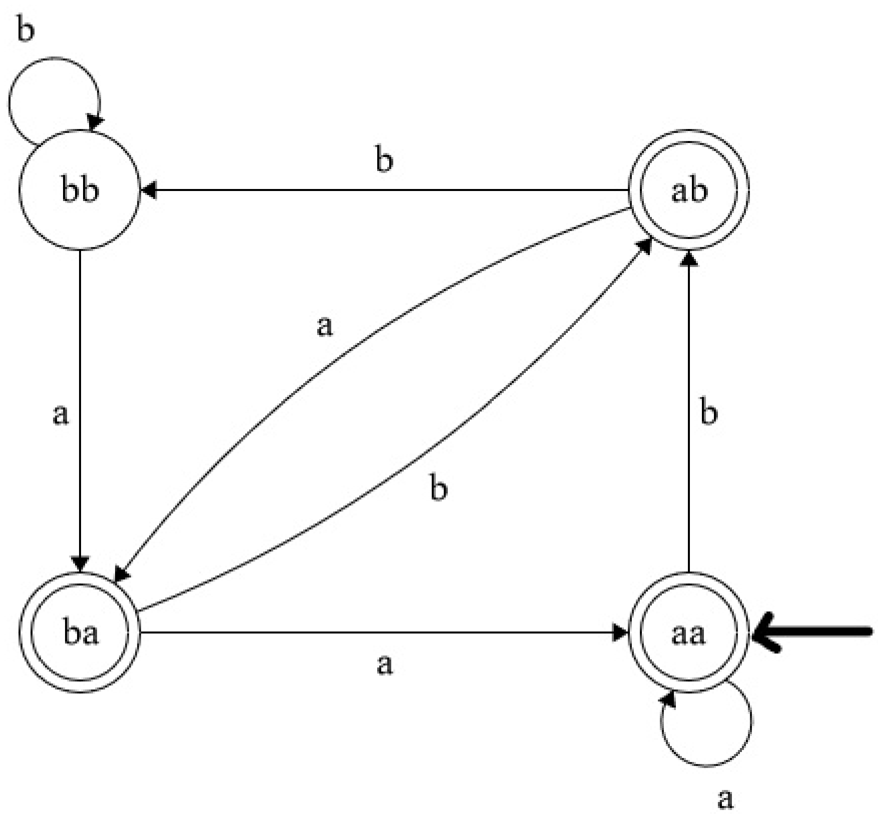
$$\bigwedge_{x\in\Sigma^*} (x,x) \in R_A$$
$$\bigwedge_{x,y\in\Sigma^*} (x,y) \in R_A \Rightarrow (y,x) \in R_A$$
$$\bigwedge_{x,y,u\in\Sigma^*} (x,y) \in R_A \Rightarrow (y,u) \in R_A \Rightarrow (x,u) \in R_A$$

Einschub/Erinnerung:

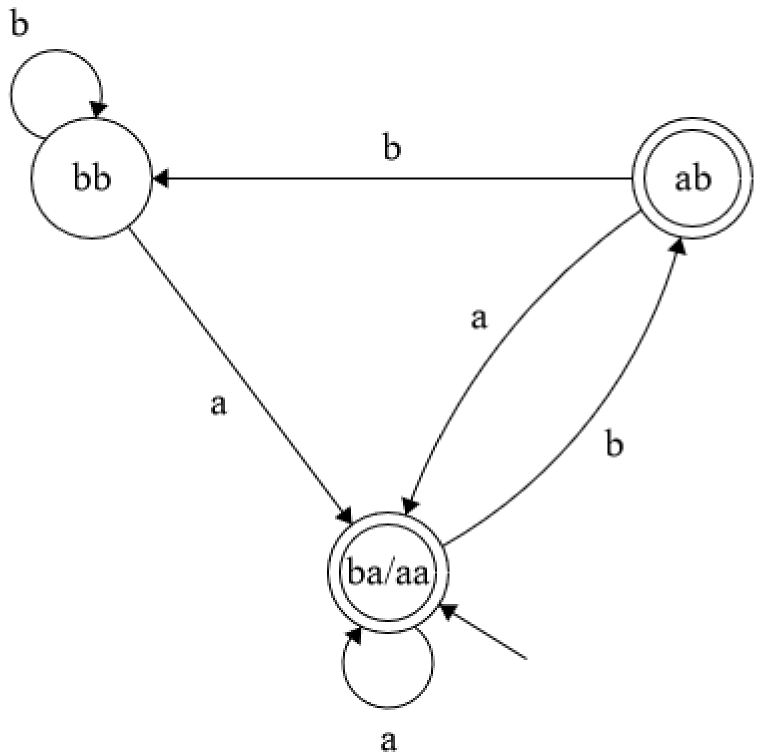
In 2.1 haben wir jedem DFA $M = \Sigma, Z, \delta, Z_0, Z_E$ eine Relation zu geordnet
$R_M = \{(x,y) \in \Sigma^* \times \Sigma^* | \delta^*(z_0) = \delta^*(z_0, y)\}$
R_M ist Äquivalenzrelation
M_M ist <u>rechtinvariant</u> bzgl. Verkettung

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

<u>Bsp</u>: endet <u>nicht</u> 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 Äuivalenzrelation 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.

$[x]_M = \{y \in \Sigma^*((x,y) \in R_M)\} = \{y \in \Sigma^* | \delta^*(z_0, y)\}$, 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 $(x,y) \in R \underset{rechtsinvariant}{\Rightarrow} \bigwedge_{z \in \Sigma^*} (xz, yz) \in R$

$\Rightarrow \bigwedge_{z \in \Sigma^*} xz \in [xz]_R$ 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 endlcih viele Index

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

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

$z = \{[x] | x \in A\}$ ... 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 Myhil Nerode kann sowohl als Beweis von Regularität als auch zum Beweis von Nichtregularität genutzt werden

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

Behauptung:

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

$K_2 = [a] = \{w \in \{a,b\}^* | \#_a(w) \equiv_3 1\}$
$K_3 = [aa] = \{w \in \{a,b\}^* | \#_a(w) \equiv_3 2\}$

$z.Z.: \; x, y \in K_i \; i = 1,2,3 \Rightarrow (x,y) \in R_{A_1}$
$K_1 \cup K_2 \cup K_3 = \{a,b\}^*$
K_i sind paarweise disjunkt
$\Rightarrow$ Index $R_{A_1} = 3 \rightarrow \; A_1$ ist regulär

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

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

z.Z.: $[a^i, a^j] \notin R_{A_2}$ für $i \neq j$
sei $z = b^i$
Dann gilt:

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

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

$\Rightarrow$ 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 unendlcihe viele Klassen gibt.

Bsp:

Für komplette Äquivalenzklassenzerlegung von Σ^* durch eine Sprache
$A = \{w \in \{a,b\}^* | \#_a(w) = \#_b(w)\}$
$[\lambda] = \{w \in \{a,b\}^* | \#_a(w) = \#_b(w)\}$
$[a^i] = \{w \in \{a,b\}^* | \#_a(w) - \#_b(w) = i\}$, für $i \geq 1$
$[a] = \{a, aba, aab, baa, aaabb, \dots\} \dots$ welche immer ein a mehr haben

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

2.6 Abschlusseigenschaften

Definition 2.6.1

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

SATZ 2.6.2

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

1. Vereinigung
2. Durschnitt
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 $M' = (\Sigma, Z, \delta, Z_0, Z \backslash Z_E)$ gerade $L(M') = \bar{A} = \Sigma^* \backslash A$

$2)$ 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 $A \backslash B = A \cap \bar{B} = A \cap \neg(B)$ 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

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

wobei $\delta((z_1, z_2), a) = (\delta_1(z_1, a), d_2(z_2, a))$

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

3. Kapitel Kontextfreie Sprachen

für alle Regeln außer $\lambda-$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

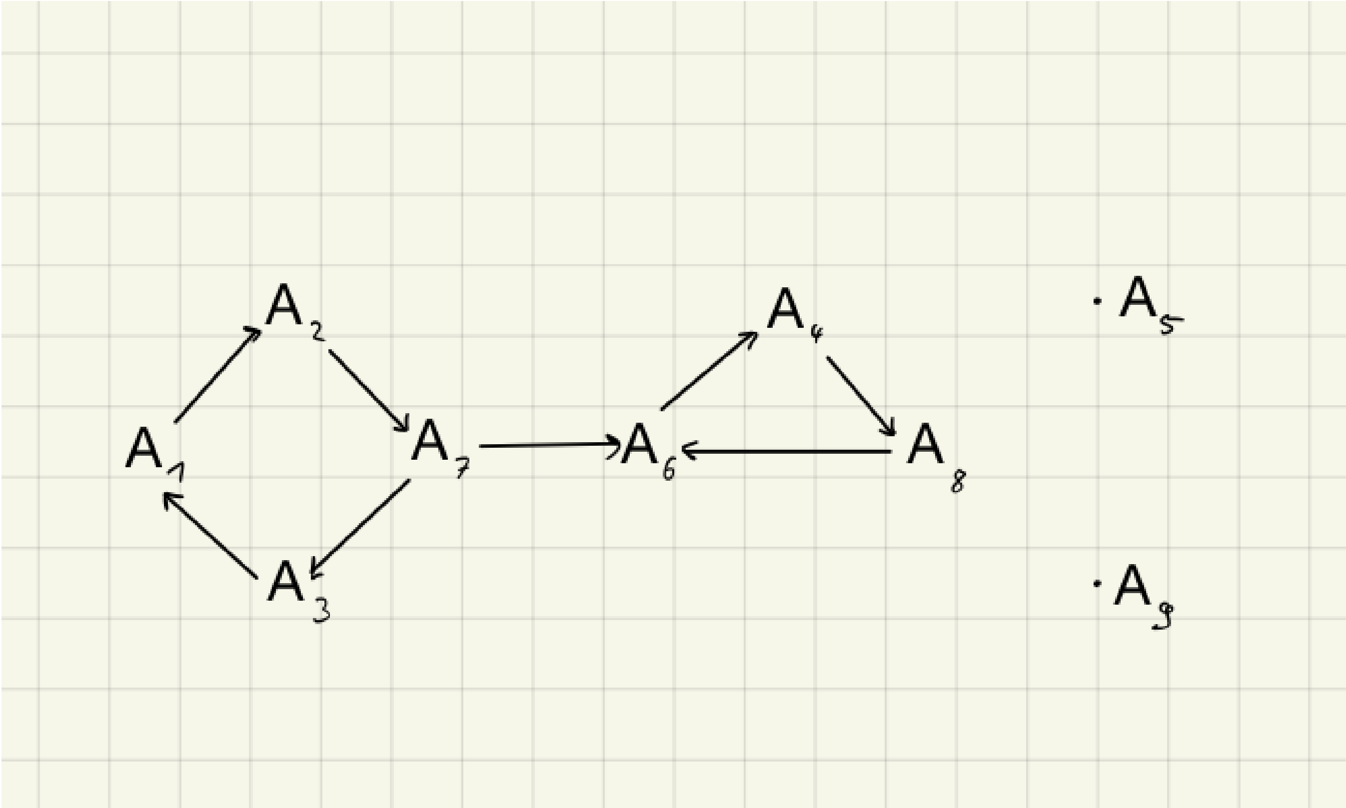
Definition 3.1.1

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 Knotemenge N.



1. Entfernen aller Zyklen

$$A_{i_1} \to A_{i_2}, A_{i_2} \to A_{i_3}, \ldots, A_{i_{k-1}} \to A_{i_k}, A_{i_k} \to A_{i_1}$$

$A_{i,j} \in N$

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

$A \to BC$ $A, B, C \in N$ $A \neq B, A \neq C$
$A \to a$ $A \in N, a \in \Sigma$
$S \to \lambda$ 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 $\mathscr{L}(G) = A$. 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 $C_a \to a\ ;\ a \in \Sigma$ werden eingefügt/hinzugefügt
4. Nun kann es Regeln der Form $A \to B_1 B_2$
   $B_m\ ;\ m > 2$ geben

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

Wir ersetzen $A \to B_1 D_1, D_1 \to B_2 D_2, D_3 \to B_3 D_3, \ldots, D_{m-2} \to B_m D_m$
Die nun entstehende Grammatik heiße G'. Sie ist in Chomsky-NF und es gilt $\mathscr{L}(G') = \mathscr{L}(G)$

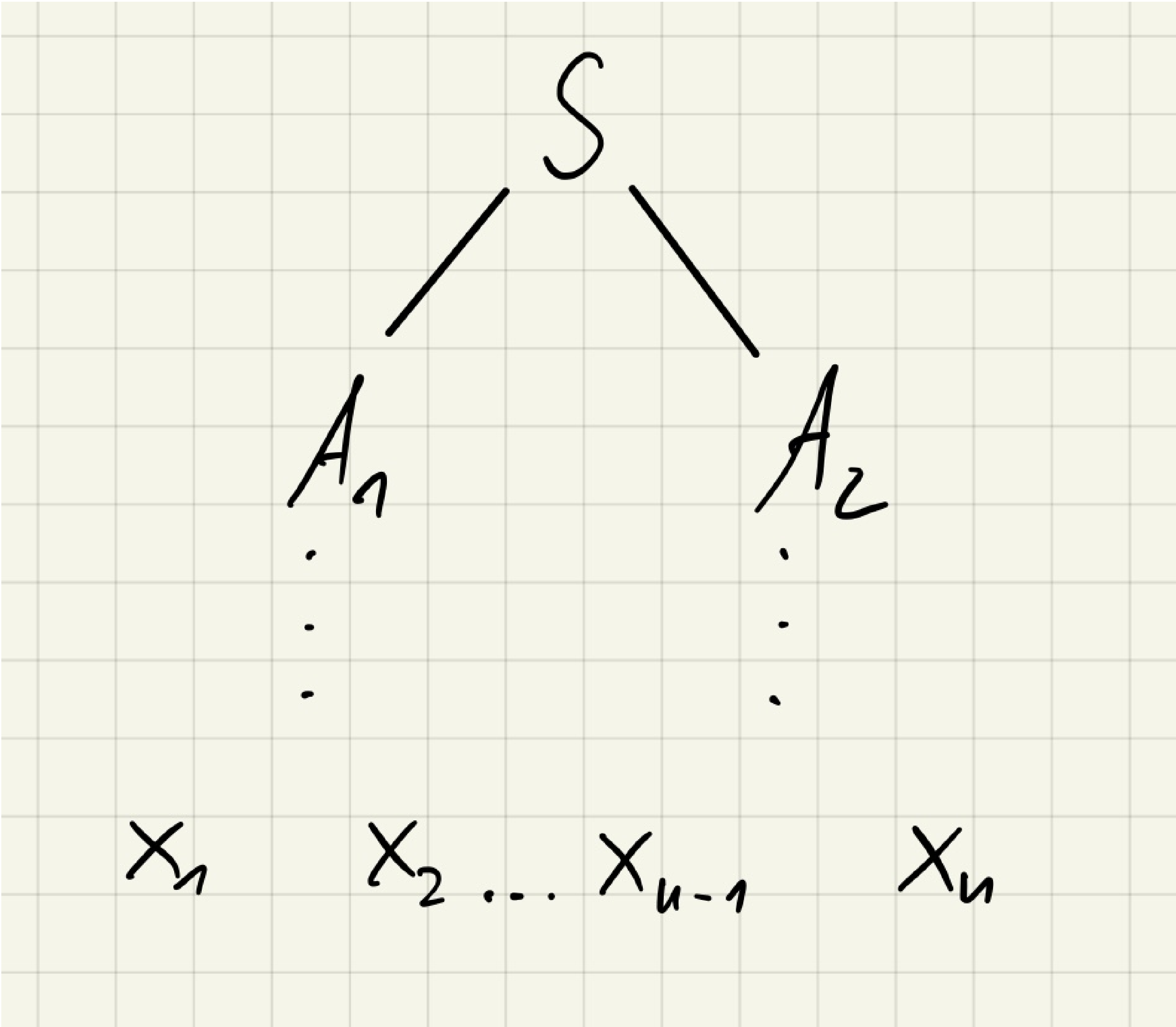
3.2 Pumping Lemma für Kontextfreie Sprachen

Satz 3.2.1

Sei $L \in \Sigma^*$ 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:

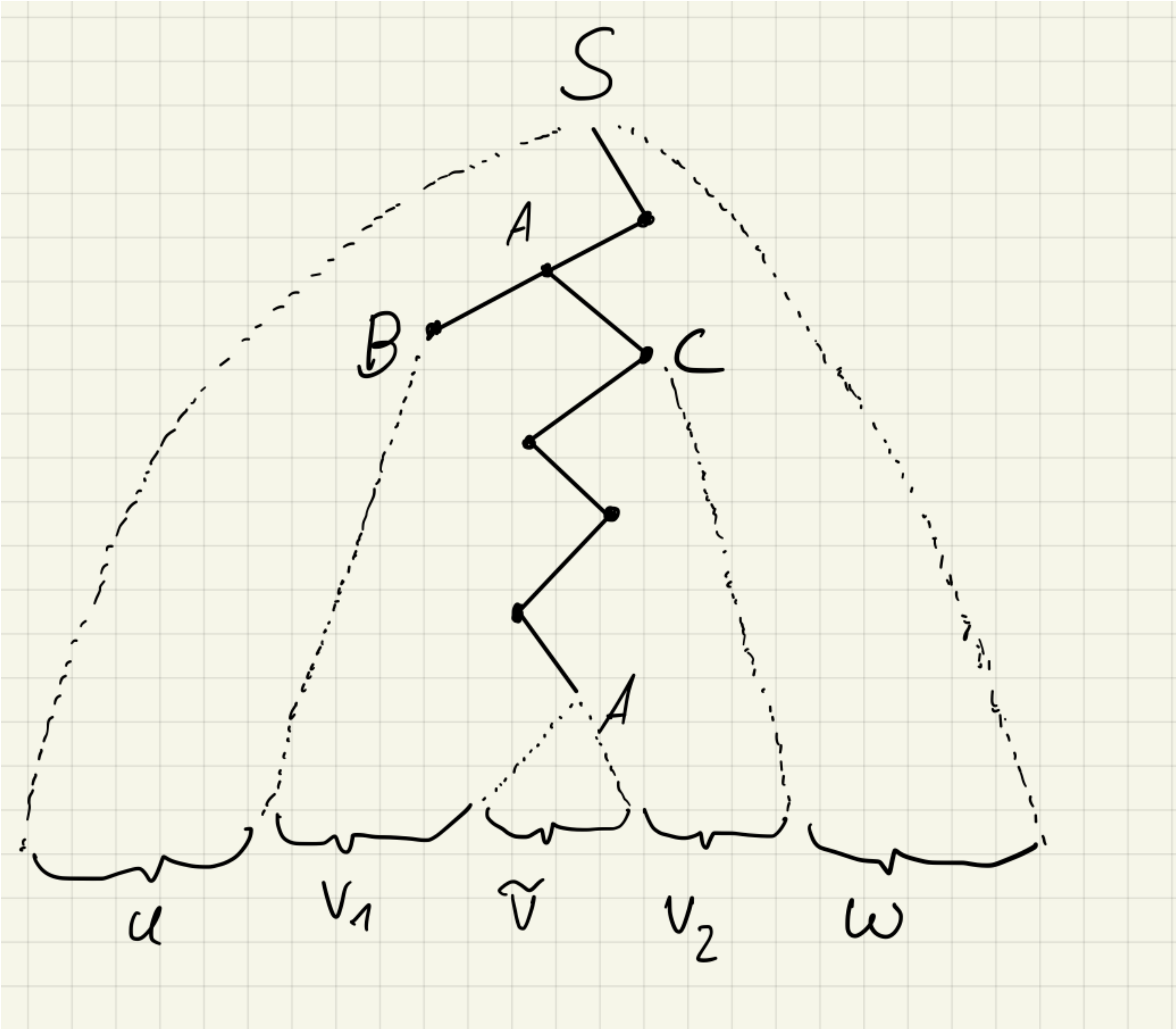


Dieser Baum hat $|x| \geq 2^{|N|}$ Blätter Folglich muss es einen Pfad geben der Mindestens länge $|N|$ hat.

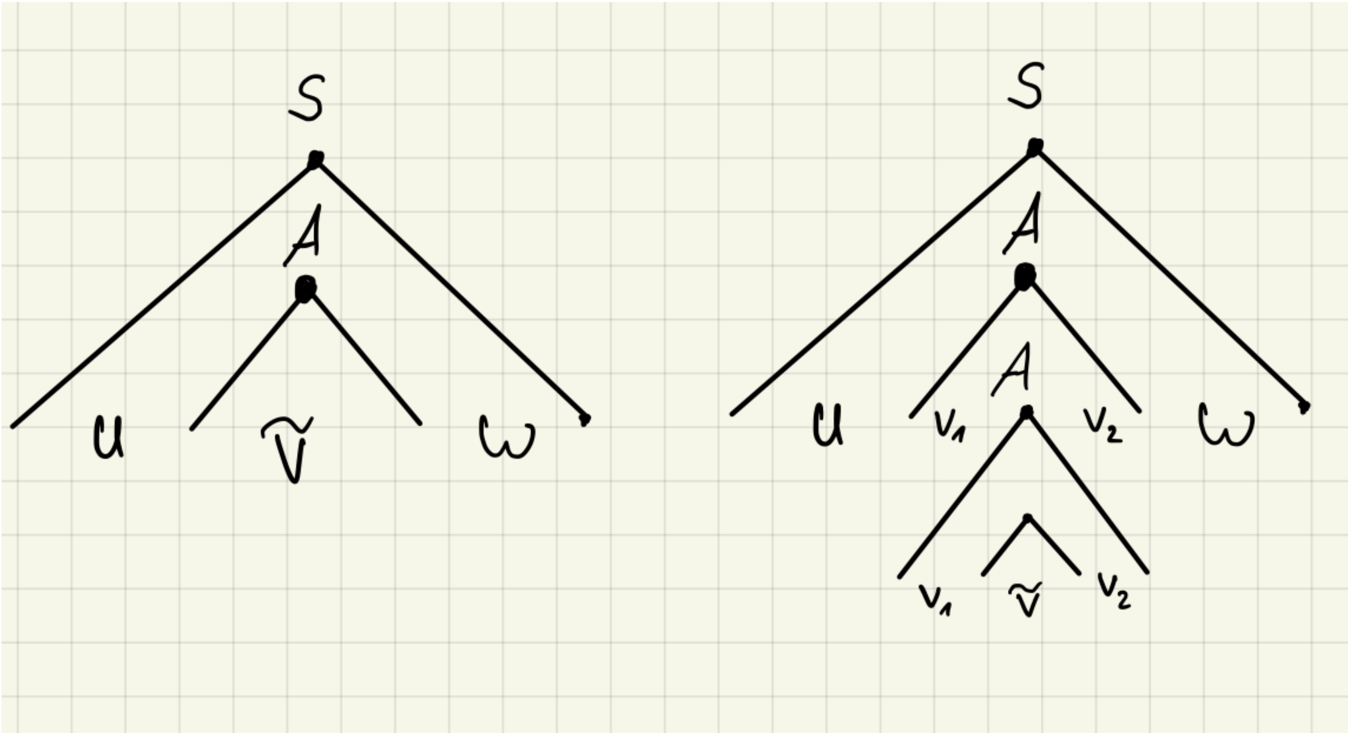
Lemma 3.2.2

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.
$K \to k + 1$.
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 $\to$ 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 doppelte 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 $uv_1^i\tilde{v}v_2^iw$ für $i \geq 0$

1. $L = \{a^nb^nc^n | n \geq 1\}$
   Annahme: $L \in CF$

Sei m die Pumpingzahl wir wählen $x = a^mb^mc^m$, $|x| = 3m \geq m$. Sei $x = uv_1\tilde{v}v_2w$ eine geeigente Zerlegung gemäß PL, dass heißt $v_1v_2 \geq 1$
$|uv_1v_2| \leq m$

$$\underbrace{aaa}_{u}\underbrace{a\ldots abb}_{v_1}\underbrace{b}_{\tilde{v}}\underbrace{b\ldots b}_{v_2}\underbrace{bccc\ldots cc}_{w}$$

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.

$\to |v_1\tilde{v}v_2| \leq m,\ |v_1v_2| \geq 1$
$\to v_1v_2 = 0^k\ 1 \leq k \leq m$

$$m^2 \leq |uv_1^2\tilde{v}v_2^2w| = |uv_1\tilde{v}v_2w| + |v_1v_2| = m^2 + |v_1v_2|$$

$$m^2 + |v_1v_2| \leq m^2 + m < (m+1)^2$$

$\to uv_1\tilde{v}v_2w \notin L$ (Widerspruch)
$\to L \notin CF$

3.3 Abschlusseigenschaften

Satz 3.3.1

CF ist eine abgeschlossen bezüglich

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

CF ist noch <u>nicht</u> abgeschlossen bezüglich

4. Durchschnitt
5. Komplement
6. Differenz

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

$G_1 = (\Sigma, N_1, S_1, R_1)$
$G_2 = (\Sigma, N_2, S_2, R_2)$ Kontextfrei

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

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

2. $\mathscr{L}((\Sigma, N_1 \cup N_2 \cup \{S\}, S, R_1 \cup R_2 \cup \{S \to S_1S_2\}))= A \cdot B$

3. $\mathscr{L}((\Sigma, N_1 \cup \{S\}, S, R_1 \cup R_2 \cup$
   $\{S \to S_1, S \to \lambda, S \to S_1, S \to S_1S_1\} \backslash \{S_1 \to \lambda\}))= A \cup B$

4. Wir betrachten zwei Sprachen:
   $L_1 = \{a^ib^jc^j | i, j > 0\}$
   $L_2 = \{a^ib^ic^j | i, j > 0\}$
   $L_1, L_2 \in CF$, denn ... ÜA

   aber $L_1 \cap L_2 = \{a^nb^nc^n | n > 0\} \notin CF$

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

6. $A \cap B = A \cup B \backslash \big((A \backslash B) \cup (B \backslash A)\big)$

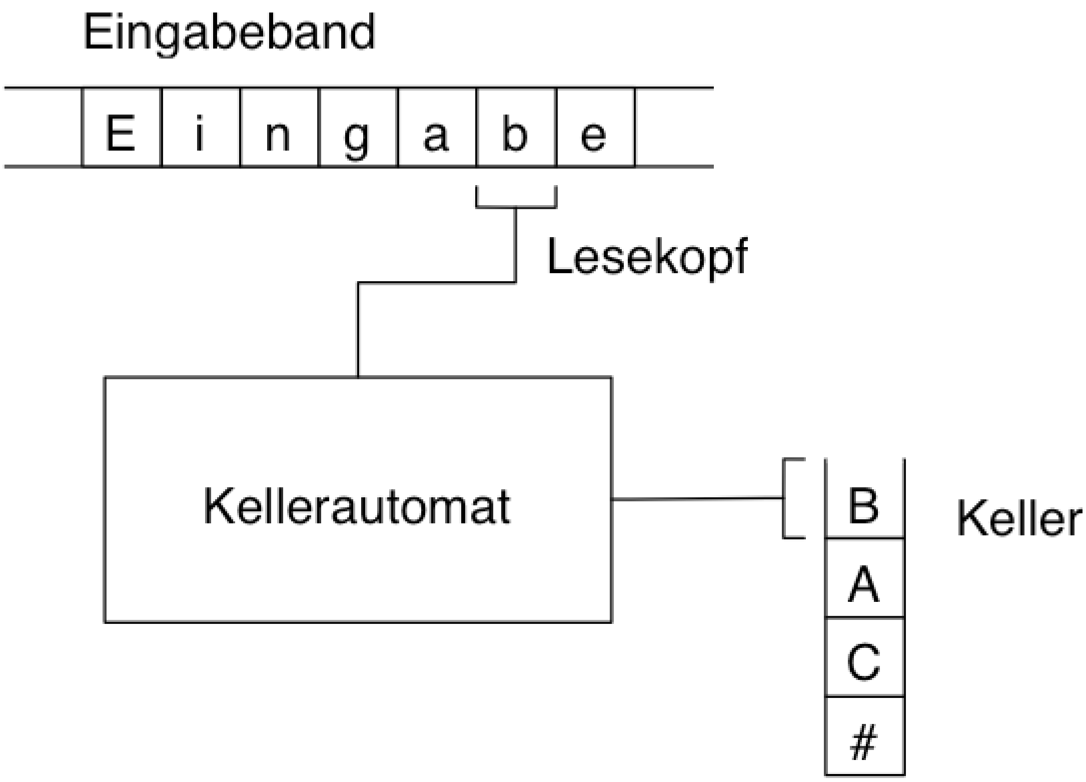
3.4 Nicht deterministischer Kellerautomat(Push-Down-Automaton)

$$\{a^nb^n | n \geq 0\} \in CF \ \downarrow \ \textit{Zähler wird gebraucht} \downarrow \textit{Stack}$$

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. $\delta : Z \times (\Sigma \cup \{\lambda\} \times \Gamma \to \gamma_{endl}(Z \times \Gamma^*)$
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. $\delta(z, a, A) = \{(_1, B_1, B_2), (z_2, \lambda), (z_3, BA)\}$
   $(z_1, B-1 .. . B_k) \to$ A wird durch $B_1 B_2 ... B_k$ ersetzt

$(z_1, \lambda) \to$ A wird entfernt (pop)
$(z_1, BA) \to$ neues Symbol B wird aufgesetzt (push)

2. $(\delta(z, \lambda, A) = \{(z_1, B_1 \ldots B_k), (z_2, \lambda), (z_3, BA)\}$
   $\downarrow$ ist auch erlaubt
   $\to$ Veränderung ohne das ein Symbol der Eingabe gelesen wird
3. nur ein Symbolzustand ist keine Einshränkung
   $\to \delta(z_0, \lambda, \square) = \{(z_0', \square), (z_0'', \square), (z_0''', \square)\}$
   $\to$ 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 $K = (z, u, v)$:

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

$\to$ Anfangskonfiguration: $(z_0, w, \square), w \in \Sigma^*$
$\to (z_1, \lambda, p), z_1 \in z_E, p \in \Gamma^*$

Definition 3.4.3

Wir definieren auf $Z \times \Sigma^* \times \Gamma^*$
(Menge der Konfigurationen) eine binäre Relation (Nachfolgekonfiguration) wie folgt:
$(z, u, v) \Vdash_M (z', u', v') \leftrightarrow_{df} (*)$

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

$\to$ Analog zu $\vdash_G^*$ definiert man $\vdash_M^\times$ als die reflexive und transitive Hülle von $\vdash_M$
$\to K \ \vdash_M^* K' \leftrightarrow_{df}$ es gibt endliche Konfigurationen $K_1, \ldots, K_n$ mit $K \vdash_M K_1$,
$K_1 \vdash_M K_2, \ldots, K_{n-1} \vdash_M K_n$ und $K_n \vdash_M K'$

Definition 3.4.4

Die vorne einem PDA akzeptierte Sprache ist definiert als:

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

<u>Beispiel</u>

1. $L = \{a^n b^n | n \geq 1\}$
   $M = \big(\{a, b\}, \{\Box, A\}, \{z_0, z_1, z_{ende}\}, \delta, z_0, \{z_{ende}\big)$
   $\delta)$ $(z_0, a, \Box) \to (z_0, A\Box)$
   $(z_0, a, A) \to (z_0, AA)$
   $(z_0, b, A) \to (z_1, \lambda)$
   $(z_1, b, A) \to (z_1, \lambda)$
   $(z_1, \lambda, \Box) \to (z_{ende}, \Box)$

2. $L = \{w\$w^R | w \in \{a, b\}^*\}$
   $M = \big(\{a, b, \$\}, \{A, B, \Box\}, \delta, z_0, \{z_{ende}\big)$
   $\delta)$
   $(z_0, a, A) \to (z_0, A\Box)$
   $(z_0, a, A) \to (z_0, AA)$
   $(z_0, a, B) \to (z_0, AB)$

   $(z_0, b, \Box) \to (z_0, B\Box)$
   $(z_0, b, A) \to (z_0, BA)$
   $(z_0, b, B) \to (z_0, BB)$

   $(z_0, \$, \Box) \to (z_1, \Box)$
   $(z_0, \$, A) \to (z_1, A)$
   $(z_0, \$, B) \to (z_1, B)$

   $(z_1, a, A) \to (z_1, \lambda)$
   $(z_1, b, B) \to (z_1, \lambda)$
   $(z_1, \lambda, \Box) \to (z_{ende}, \Box)$

Definition 3.4.5

Sei $M = \{\Sigma, \Gamma, Z, \delta, z_0, z_E\}$ ein PDA
Wir sagen M akzeptiert mit leerem Keller, wenn

$$A = \mathscr{L}(M) = \{w \in \Sigma^* | \bigvee_{z \in Z} (z_0, w, \Box) \Vdash_M^* (z, \lambda, \lambda)\}$$

Bemerkung: auf z_{ende} kann verzichtet werden

Satz 3.4.6

Eine Sprache A kann genau dann von einem PDA akzeptiert werden, wenn sie von einem PDA mit leeren Keller akzeptiert werden kann. (ohne Beweis)

Definition 3.4.7

Sei $G = (\Sigma, N, S, R)$ eine Ableitung $S \vdash_G^* w, S = p_1 \vdash_G p_2 \vdash_G p_3 \vdash_G \ldots \vdash_G p_n = w$
heißt Linksableitung $\leftrightarrow_{df}$ für alle $i = 1, \ldots, n-1$ gilt:

$$\bigvee_{u \in \Sigma^*} \bigvee_{A \in N} \bigvee_{q \in (\Sigma \cup N)^*} \bigvee_{v \in N^*} (p_i = uAv, p_{i+1} = uqv, (A \to q) \in R)$$

oder

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

Behauptung: Sei $G = (\Sigma, N, S, R)$ mit $\mathscr{L}(G) \in CF$
$w \in \mathscr{L}(G)$ gdw. Es gibt eine Linksableitung $S \vdash_M^* w$
Beweis: mit Hilfe von Syntaxbäumen

Satz 3.4.8

Eine Sprache ist genau dann kontextfrei, wenn sie von einem PDA akzeptiert werden kann
"$\to$:" Sei $G = (\Sigma, N, S, R)$ eine Konfiguration(kfG) mit $\mathscr{L}(G) = A$

- Wir geben ein PDA $M = \{\Sigma, \Gamma, Z, \delta, z_0, z_E\}$ an, der A mit leerem Keller akzeptiert, indem er die Ableitung von M im Keller simuliert
- Der Kellerinhalt repräsentiert die während der Ableitung entstehenden Zeichenketten aus $\Sigma \cup N$
  $\Gamma = \Sigma \cup N \cup \{\}$ ($\Box$ unteres Kellerzeichen)

$z = \{z\}, (z_E = \{z\})$
$\delta(z, \lambda, \square) = \{z, s\}$

1. $\delta(z, \lambda, A) = \{z, \alpha | A \to \alpha \in R, \alpha \in (\Sigma \cup N)^*\}$
2. $\delta(z, a, a) = \{(z, \lambda)\}$
   $\delta() = \emptyset$ Sonst

Bemerkung:
(1) Wenn das obere Kellersymbol ein Nichtterminal der Grammatik ist, wird ohne lesen eines Eingabesymbols das Nichtterminal im Keller entsprechend einer Regel aus R ersetzt.
(2) Ist das obere Kellerzeichen ein Terminal und stimmt mit dem nächsten Eingabezeichen überein, wird es vom Keller entfernt (pop)

<u>Beispiel</u>
$R = \{S \to aSb | ab\}$
$\to$ Es gilt nun für alle $w \in \Sigma^*$
$w \in A \Leftrightarrow w \in \mathscr{L}(G)$
$\Leftrightarrow$ Es gibt eine linksableitung der Form
$S \vdash_G \alpha, \ldots, \alpha_n \vdash_G w$
$\Leftrightarrow$ Es gibt eine Folge von Konfigurationen von M mit
$(z, w, \square) \vdash (z, w, \delta) \vdash \ldots \vdash (z, \lambda, \lambda)$
$\Leftrightarrow$ w wird von M mit leerem Keller akzeptiert

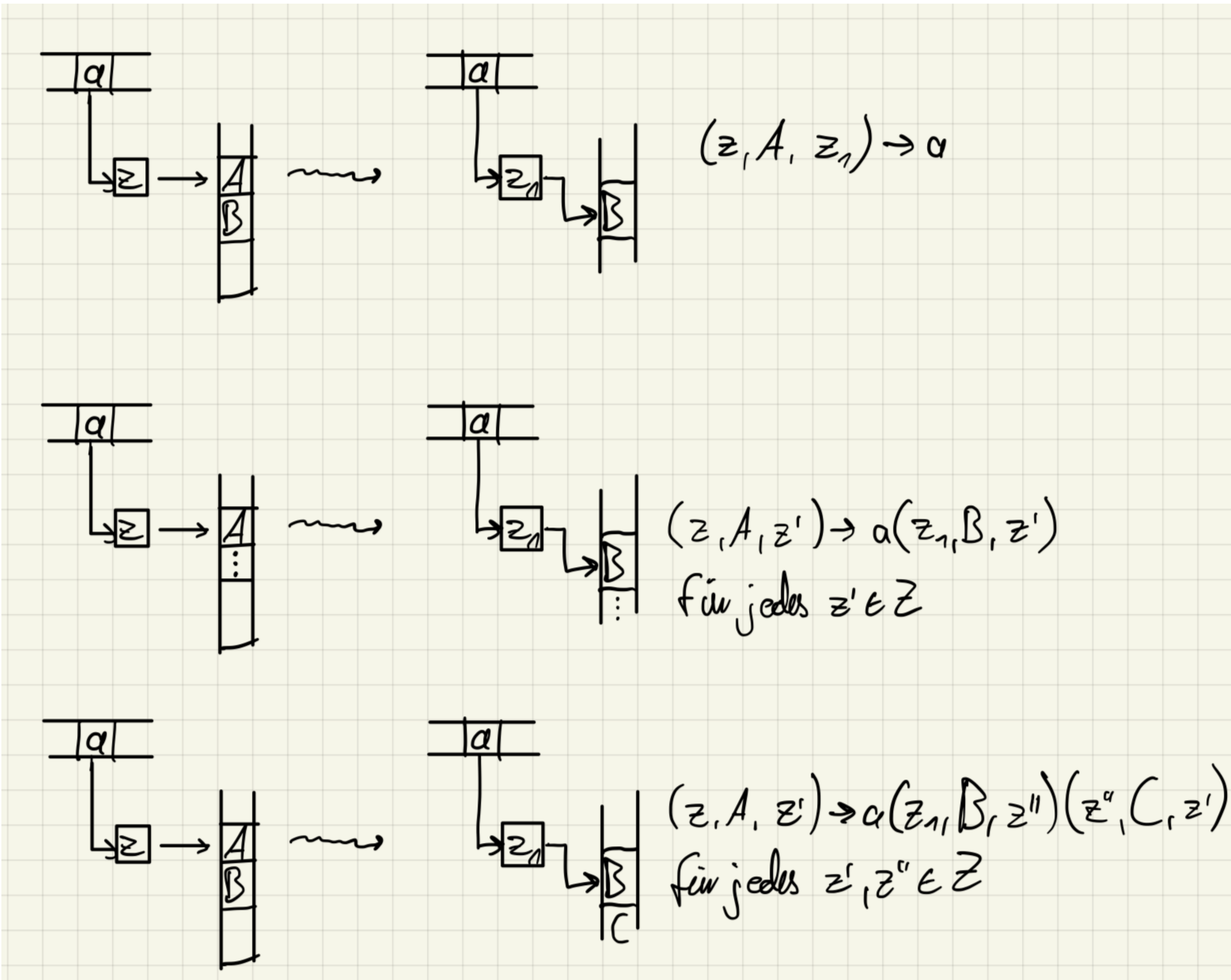
"$\underset{3.4.8}{\Leftarrow}$": Sei $A \subseteq \Sigma$ und $M = \{\Sigma, \Gamma, Z, \delta, z_0, z_E\}$ ein PDA mit

$$A = \left\{ W \in \Sigma^* | \bigvee_{z \in Z} \Big((z_0, w, \square) \vdash_M^* (z, \lambda, \lambda) \Big) \right\}$$

Diesen gibt es gemäß Satz 3.4.6
Wir nehmen an, dass für alle $z \in Z, a \in \Sigma, \gamma \in \Gamma$ gilt:
$(z', B_1, \ldots, B_k) \in \delta(z, a, \gamma) \to (z', B_1 \ldots B_k)$
$\to$ durch: $(z, a, \gamma) \to (z_1, B_{k-1}, B_k)$
$(z_1, \lambda, B_{k-1}) \to (z_1, B_{k-2}, B_{k-1})$
$(z_{k-2}, \lambda, B_2) \to (z', B_1, B_2)$ erreichen

- wir konstruieren nun eine Grammatik G, die die Rechenschritte von M durch linksableitung simuliert

Es sei $G = (\Sigma, N, S, R)$ mit $N = \{S\} \cup Z \times \Gamma \times Z$
$R = \{S \to (z_0, \square, z') | z' \in Z\}$
$\cup\{z, A, z_1 \to a | \big(z_1, \lambda \in \delta(z, a, A)\big)\}$
$\cup\{(z, A, z') \to a(z_1, B, z') | z' \in Z, (z_1, B) \in \delta(z, a, A))\}$
$\cup\{(z, A, z') \to a(z_1, B, z'')(z'', C, z') | z', z'' \in Z, (z_1, BC) \in \delta(z, a, A)\}$



-In z_1 wird B in C geschrieben, in Z'' wird B entfernt
-Nun ist C Topelement und steht als wäre es gerade geschrieben
<u>Behauptung:</u> Für alle $z, \tilde{z} \in N, A \in \Gamma$ nd alle $w \in \Sigma^*$ gilt:
$(z, a, \tilde{z}) \vdash_G^* \; w$ gdw $(z, w, A) \vdash_M^* (\tilde{z}, \lambda, \lambda)$

$*$) Offenbar gilt auch für alle $a \in \Sigma \cup \{\lambda\}$: $(z, A, \tilde{z}) \vdash_G \Leftrightarrow (z, A, \tilde{z}) \to a \in R$
$\Leftrightarrow (\tilde{z}) \in \delta(z, a, A)$
$\Leftrightarrow (z, a, A) \vdash_M (\tilde{z}, \lambda, \lambda)$
unmittelbar direkt aus der Definition der Grammatik

" $\underset{Beh.}{\Leftarrow}$:" Induktion über n. Zahl der Rechenschritte von M

I.A.: $n = 1$ (siehe($*$))
I.V.: Für alle $z, \tilde{z} \in Z, w \in \Sigma^*, A \in \Gamma$ gelte: $(z, w, A) \vdash_M^n (\tilde{z}, \lambda, \lambda) \to (z, A, \tilde{z}) \vdash_G^* w$
I.B.:Aussage gilt auch für $n+1$
I.S.: $n \to n+1$
Sei $n > 1$ und sei $(z, w, A) \vdash^{n+1} (z, \lambda, \lambda)$
$\to$ Folglich ist es ein $a \in \Sigma \cup \{\lambda\}$ und $y \in \Gamma^*$ mit $w = ay$ und
$(z, ay, A) \vdash_M (z_1, y, a) \vdash_M^* (\tilde{z}, \lambda, \lambda)$ für einen Zustand z_1
Für a gibt es 3 Möglichkeiten

1. Fall $a = \lambda \to$ nicht möglich, da (z_1, y, λ) kie Folgekonfiguration hat
2. $a = B, B \in \Gamma$
   $\to$ nach N ist dann $(z_1, B, \tilde{z}) \vdash_M^* y$ und außerdem ist $(z, A, \tilde{z}) \to a(z_1, B, \tilde{z})$ eine Regel in R
   $\to$ setzen wir die 2 Teile zsm. erhalten wir $(z, A, \tilde{z}) \vdash_b a(z_1, B, \tilde{z}) \vdash_G^* ay$
3. $a = BC; B, C \in \Gamma$
   $\to$ also $(z, ay, A) \vdash_M^{n_1} (z_2, y_2, C) \vdash_M^{n_2} (\tilde{z}, \lambda, \lambda)$
   $\to$ Dabei ist $n_1 + n_2 = n$ und y_2 ein Endwort von y $\left(\bigvee_{y_1 \in \Sigma^*} y = y_1 y_2 \right)$
   $\to$ Für y_1 gilt weiterhin: $(z_1, y_1, B) \vdash^m (z_2, \lambda, \lambda)$
   $\to$ Nach I.V. Ist somit: $(z_1, B, z_2) \vdash_G^* y_1$ und $(z, C, \tilde{z}) \vdash_G^* y_2$
   Außerdem gibt es in R die Regel: $(z, A, \tilde{z} \to a(z_1, B, z_2)(z_1, B, \tilde{z})$
   $\to$ Damit ist: $(z, A, \tilde{z}) \vdash_G a(z_1, B, z_2)(z_1, C, \tilde{z}) \vdash_G^* ay_2(z_2, C, \tilde{z})$
   $\vdash_G ay_1y_2 = w$ eine Ableitung in G $\square_{Behauptung}$

"$\Rightarrow$:"Induktion über k, Länge der linksableitung von w (ähnlich)
Aus der Behauptung folgt also: $w \in L(M) \Leftrightarrow (z_0, w, \square) \vdash_M^* (z, \lambda, \lambda)$ für ein $z \in Z$
$\Leftrightarrow_{Behauptung} (z_0, \square, z) \vdash_G^* w$ für ein $z \in Z$
$\Leftrightarrow S \vdash_G (z_0, \square, z) \vdash_G^* w$ für ein $z \in Z$
$\Leftrightarrow w \in \mathscr{L}(G)$

3.5 Deterministischer Kellererautomat für kontextfreie Sprachen

Definition 3.5.1

Ein Kellerautomat M heißt deterministisch (DPDA) falls für alle $z \in Z, a \in \Sigma$ und $A \in \Gamma$ gilt:

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

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

$\rightarrow CF_{det}\dots$ Menge der deterministisch kontextfreien Sprachen
Bsp.: $\{a^n b^n | n \geq 1\} \in CF_{det}$
$\{w\$w^R | w \in \{a,b\}^*\} \in CF_{det}$
$\{ww^R | w \in \{a,b\}^*\}\} \in CF_{det}$
Bemerkung: Nicht alle deterministisch kontextfreien Sprachen können durch einen DPDA mit leerem Keller akzeptiert werden.
DPDA's müssen mit Endzuand akzeptieren

Satz 3.5.2

Zu jedem DPDA M gibt es einen äquivalenten DPDA M', der für jede Einabe $w \in \Sigma$ steht's die gesamte Eingabe abarbeitet, also

$$\bigwedge_{w \in \Sigma^*} \bigvee_{z \in Z} \bigvee_{a \in \Gamma} (z_0, w, \Box) \vdash_M^* (z, \lambda, \alpha)$$

<u>Beweisidee:</u>
Sei $M = \{\Sigma, \Gamma, Z, \delta, z_0, z_E\}$ Ein DPDA mit $\mathscr{L}(M) = L$
Falls M eine Eingabe $w = x_1 \dots x_n$ nicht zünde liest, liegt einer der 3 folgenden Gründen vor:

1. M kommt in eine Konfiguration mit leerem Keller
2. M kommt in eine Konfiguration $(q, x_i, \dots, x_n, A\gamma)$ $i \leq n$ deswegen $\delta(q, x_i, A) = \emptyset$ keine Anweisung ausführbar ist
3. M kommt in eine Konfiguration $(q, x_i, \dots, x_n, A\gamma)$ $i \leq n$, so dass M ausgehend von (q, λ, A) eine unendliche Folge von λ-Sprüngen ausführt

$\rightarrow$ zu 1) neues Zeichen # wird auf den Keller platziert, s ist neuer Startzustand:
$(s, \lambda, \Box) \rightarrow (z_0, \Box, \#)$

zu 2) neuer Zustand $z_f \dots$ Fehler, $\Gamma = \Gamma \cup \{\#\}$
$(z, a, A) \rightarrow (z_f, A)$ für alle z, a, A mit $A = \#$ oder $\delta(z, a, A) = \delta(z, \lambda, A) = \emptyset$
$(z, a, A) \rightarrow (z_f, A)$ für alle $a \in \Sigma, A \cup \Gamma'$

zu 3)Auch Das ist zu beheben, wenn die λ-Sprünge umgeleitet werden

---

Erinnerung:

- CF ist abgeschlossen bezüglich:
  - Vereinigung
  - Produkt
  - Kleene-Abschluss $\to$ Nachthat: Schnit mit regulären Mengen
- CF ist <u>nicht</u> abgeschlossen bezüglich:
  - Komplement
  - Durchschnitt
  - Differenz

Satz 3.5.3

- CF_{det} ist abgeschlossen bezüglich:
  - 1. Komplement
  - 2. Schnitt mit regulären Mengen
- CF_{det} ist <u>nicht</u> abgeschlossen bezüglich:
  - 3. Durchschnitt
  - 4. Vereinigung
  - 5. Differenz
  - 6. Produkt
  - 7. Kleene-Abschluss

$\to$ zu 1) Idee: $A \in CF_{det}$ und $M = (\Sigma, \Gamma, Z, \delta, z_0, z_E)$
Sei ein DPDA mit $\mathscr{L}(M) = A$ #, der steht's die gesamte Eingabe abarbeitet
Aus M kann $\overline{M}$ konstruiert werden, der $\overline{A}$ akzeptiert

$\to$ Zu 2) Es sei $\mathscr{L} = \mathscr{L}(M_1)$ für einen PDA $M = \{\Sigma, \Gamma, Z_1, \delta_1, z_{01}, z_{E1}\}$
und sei Es sei $\mathscr{L} = \mathscr{L}(M_2)$ für einen DFA $M = \{\Sigma, Z, \delta_2, z_{02}, z_{E2}\}$
$\to$ wir konstruieren den Produktautomaten von M_1 und M_2. Das ein PDA der
$\mathscr{L}_1 \cap \mathscr{L}_2$ akzeptiert
$M = \{\Sigma, \Gamma, Z_1 \times Z_2, \delta_1, \{z_{01}, z_{02}\}, Z_{E1} \times Z_{E2}\}$

mit $\big((p',q'),\gamma\big) \in \delta\big((p,q),a,A\big)$ gdw. $(p',\gamma) \in \delta_1(p,a,A)$ und $\delta_2(q,a) = q'$ für
$A \in \Gamma, \gamma \in \Gamma^*$
und $\big((p',q'),\gamma\big) \in \delta\big((p,q),\lambda,A\big)$ gdw. $(p',\gamma) \in \delta_1(p,\lambda,A)$
$\rightarrow$ Man zeigt durch Induktion über k:
$\big((p,q),uv,\gamma\big) \vdash_M^k \big((p',q'),v,\beta\big)$ gdw. $(p,uv,\gamma) \vdash_{M_1}^k (p',v,\beta)$ und $(q,u) \vdash_M^k q'$
Bemerkung:
Falls M_1 deterministisch ist, ist auch M deterministisch, damit ist sowohl CF als auch CF_{det} abgeschlossen bezüglich Durchschnitt

zu 3)
$\mathscr{L}_1 = \{a^i b^j c^j | i,j > 0\} \in CF_{det}$
$\mathscr{L}_2 = \{a^i b^i c^j | i,j > 0\} \in CF_{det}$
$\rightarrow \mathscr{L}_1 \cap \mathscr{L}_2 = \{a^n b^n c^n | n > 0\} \notin CF_{det}$, also ebenfalls $\mathscr{L}_1 \cap \mathscr{L}_2 \notin CF_{det}$

zu 4)
$A \cap B = \overline{\overline{A} \cup \overline{B}}$
$\rightarrow$ da CF_{det} abgeschlossen bezüglich Komplement, wurde bei Abgeschlossenheit bezüglich "$\cup$"
Abgeschlossenheit bezüglich "$\cap$" folgen

zu 5)
$A \cap B = A \backslash \overline{B}$
zu 6)
$\mathscr{L}_1, \mathscr{L}_2$ wie oben
$L_3 = \#\mathscr{L}_1 \cup \mathscr{L}_2 \in CF_{det}$
$\rightarrow$ Das Vorhandensein von # (oder nicht) sagt uns, ob wir noch einem Wort aus der L_1 oder $\mathscr{L}_2$ zu suchen haben
$\rightarrow \{\#\}^* \in REG, \{\#\}^* \cdot \mathscr{L} \notin CF_{det}$ (intuitiv, bzz.)

zu 7) folgt aus 6)

Bemerkung: Die Abschlusseigenschaften ermöglichen uns z.Z., dass Sprachen nicht (deterministisch) kontextfrei sind

Bsp.:
$\mathscr{L} = \{w \in \{a,b\}^* | \#_a(w) = \#_b(w) = \#_c(w)\}$
$A = \{a^i b^j c^k | i,j,k \in \mathbb{N}\} \in REG$
$\mathscr{L} \cap A = \{a^n b^n c^n | n \in \mathbb{N}\} \notin CF$

$\rightarrow$ wäre $\mathscr{L}$ kontextfrei (CF) folgt wegen 2) $\mathscr{L} \cap A \in CF$
$\rightarrow$ Das ist ein Widerspruch, da $L \notin CF$

Satz 3.5.4

$REG \subsetneq CF$

Beweis: $a)$ $\{a^n b^n | n \geq 1\} \in CF_{det} \backslash REG$
$b)$ $CF_{det} \subsetneq CF$ wegen Komplementabgeschlossenheit

3.6 Das Wortproblem für kontextfreie Sprachen

geg.: kontextfreie Grammatik G und ein Wort $x \in \Sigma^*$

Frage: Ist $x \in \mathscr{L}(G)$

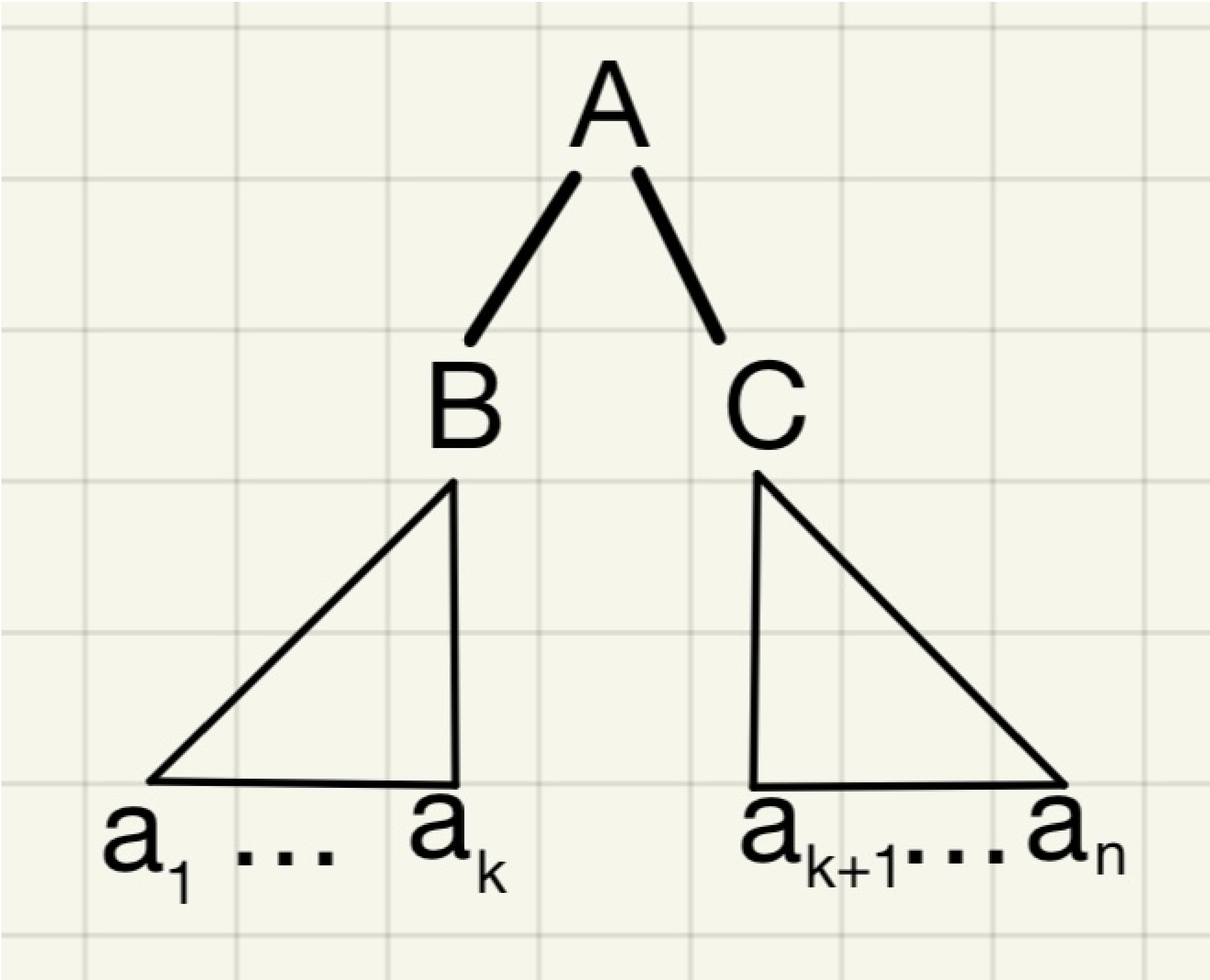
Gibt es einen Algorithmus, der bei Eingabe einer kontextfreien Grammatik $G = (\Sigma, N, S, R)$ und eines Wortes $x \in \Sigma^*$ in endlicher Zeit entscheidet, ob $x \in \mathscr{L}(G)$ oder $x \notin \mathscr{L}(G)$?

CYK-Algorithmus(Cocke, Younger, Kasumi)

- G ist in Chomsky Normalform gegeben
- Wenn $x = a, a \in \Sigma$ abgeleitet werden kann, dann nur durch $A \rightarrow a$ für ein $A \in N$

für $x = a_1 ... a_n$ $n \geq 2,$ $a \in \Sigma$ kann x aus $A \in N$ abgeleitet werden, wenn:

$\rightarrow$ eine Regel $A \rightarrow BC$ angewendet wurde und
$\rightarrow$ es gibt ein k $1 \leq k \leq n$ mit $B \vdash_G^* a_1 ... a_k$ und $C \vdash_a^* a_{k+1} ... a_n$
(Ein Anfangswort von x wird asu B abgeleitet und das entsprechende Endstück aus L)



k steht nicht fest, d.h. ess müssen alle Werte $1, 2, \ldots, n-1$ überprüft werden
$\rightarrow$ Dynamisches Programmieren

- alle Teilwörter von x werden (beginnend mit Länge 1) auf eventuelle Ableitbarkeit aus den Nichtterminalen untersucht
- diese Informationen werden in einer Tabelle abgelegt
- Wenn ein Teilwort der Länge n untersucht wird, stehen die Informationen für alle kurzen Teilwörter schon zur Verfügung

$\rightarrow$ Notation für $x = a_1 \ldots a_n$ bezeichnet $x_{i,j} = a_i a_{i+1} \ldots a_{i+j-1} \ldots x_{i,j}$ beginnt mit a_i und hat die Länge j $x_{1,k} = a_1 \ldots a_k$, $x_{k+1,n-k} = a_{k+1} \ldots a_n$

CYK-Algorithmus

Tabelle $T[1 \ldots n, 1 \ldots n]$, für $j = 1 \ldots n$ und $i = 1 \ldots n+j-1$ stehen in $T[i,j]$ alle Nichtterminale, aus denen $x_{i,j}$ abgeleitet werden kann

Eingabewort: $x = a_1 \ldots a_n \in \mathscr{L}(G)$ gdw. $S \in [1, n]$

for $i = 1$ to n do
    $T[i, 1] := \{A \in V | A \rightarrow a_i \in R\}$
end for
for $j = 2$ to n do
    for $i = 1$ to $n + 1 - j$ do
        $T[i, j] := \emptyset$
        for $k = 1$ to $j - 1$ do
            $T[i, j] := T[i, j] \cup \{A \in V | A \rightarrow BC \in R \wedge B \in T[i, k] \wedge C \in T[i + k, j - k]\}$
        end for
    end for
end for
if $S \in T[1, n]$
    print("x liegt in $L(G)$")
else
    print("x liegt nicht in $L(G)$")
end if

<u>Bsp:</u>
$\mathscr{L} = \{0^n 1 0^m | n, m \in \mathbb{N}, n > m\}$
$G = (\{0, 1\}, \{S, A\}, S, R)$ mit
$R = \{S \rightarrow 0S0 | A \; ; \; A \rightarrow 0A | 01\}$

1.

- G umformen zu G' in Chomsky-Normalform
- Kettenregel entfernen ($S \rightarrow A$)
  $S \rightarrow 0S0 | 0A | 01$
  $A \rightarrow 0A | 01$
- "lange Seiten" umformen $S \rightarrow 0S0$ ersetzen durch $S \rightarrow 0T$ und $T \rightarrow S0$

- Terminale aus der Regeln $p \to q$ mit $|q| \geq 2$ entfernen
  $R'' = \{S \to N_0T|N_0A|N_0N_1$
  $T \to SN_0$
  $A \to N_0A|N_0N_1$
  $N_0 \to 0$
  $N_1 \to 1\}$

Tabelle:

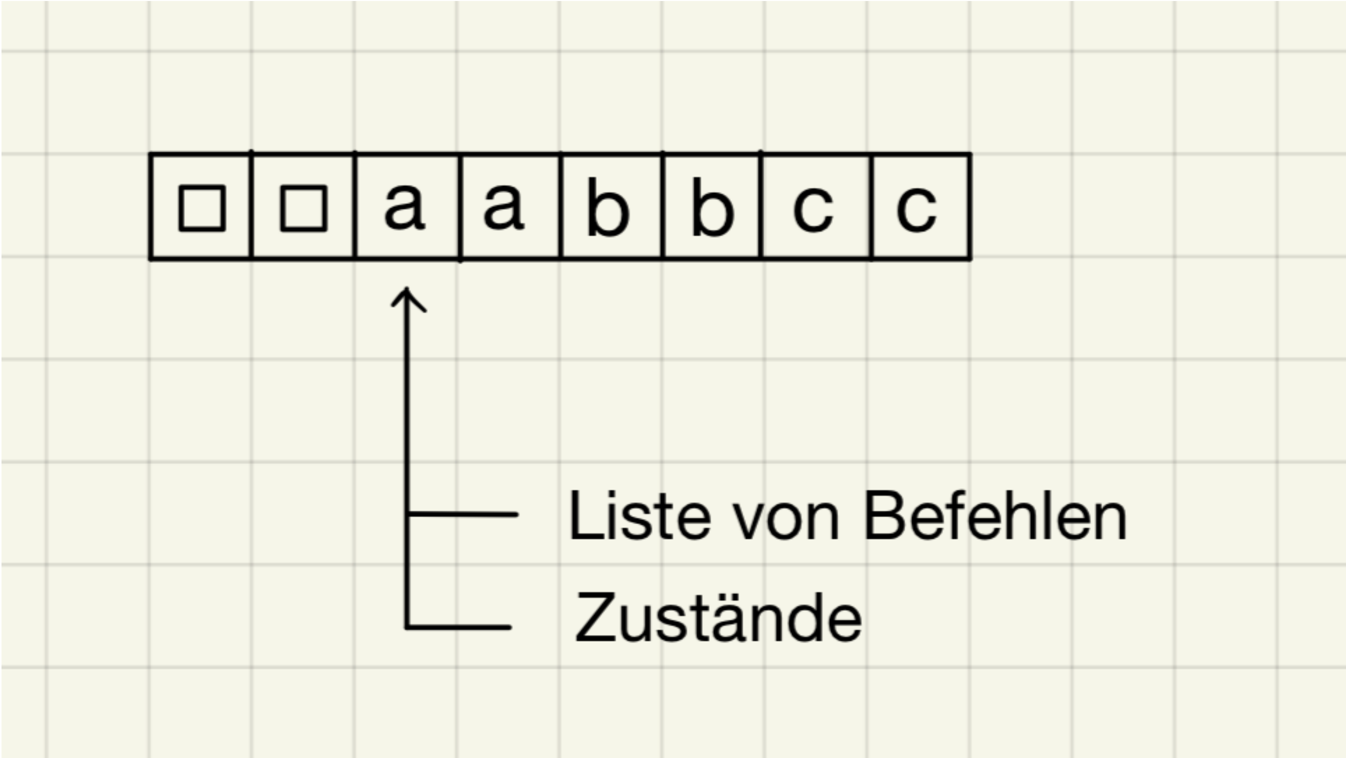
	0	0	0	1	0
1	N_0	N_0	N_0	N_1	N_0
2	$\emptyset$	$\emptyset$	A, S	$\emptyset$	
3	$\emptyset$	A, S	T		
4	A, S	S, T			
5	S				

4. Kapitel kontextsensitive und Typ 0 Sprachen

- kontextsensitiv oder Typ 1: nichtverkürzende Grammatik
  (Typ 0) $\mathscr{L}_0$: erzeugbar durch Grammatik
- Maschinenmodell: Wie könnte der Speicher erweitert werden?

4.1 Turingmaschinen

$\to$ Alan Turing 1936: Washeißt es etwas zu berechnen
$\to$ Ziel: mathematisch klar beschriebene Maschine, allgemein genug um jeden beliebigen Berechnungsprozess darzustellen



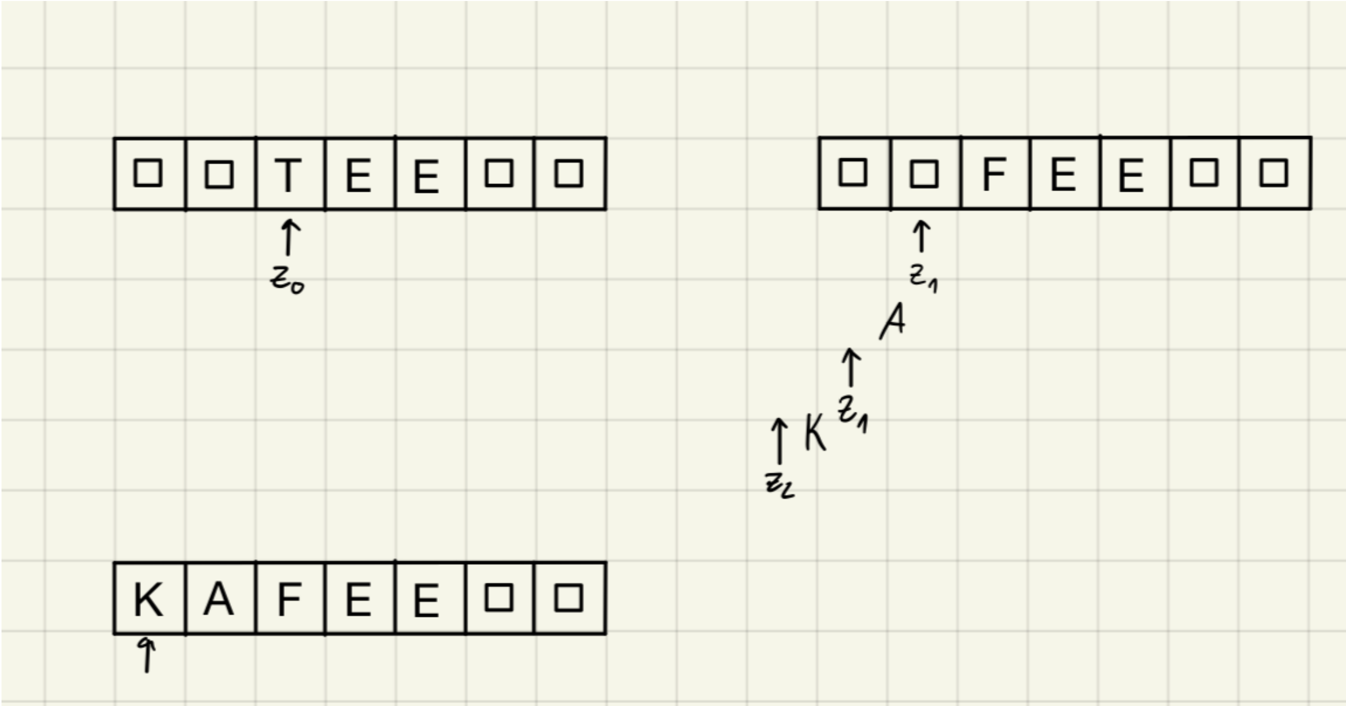
- Arbeitsspeicher ist ein Band
- eingeteilt in Zelle
- pro Zelle ein Buchstabe

<u>Arbeitsweise:</u>

- Alle Befehle in der Befehlsliste haben die Gestallt
  $(Zustand_{alt}, Symbol_{alt}) \rightarrow (Zustand_{neu}, Symbol_{neu}, Kopfbewegung)$
- TM im Zustand q liest ein a , so wird der Befehl mit
  $(a, a) \rightarrow (r, b, R)$
  $\rightarrow$ $\square$ wird durch b ersetzt, Zustand geändert
  $\rightarrow$ Kopf bewegt sich um eine Zelle nach rechts

<u>Bsp:</u>

$(z_0, T) \rightarrow (z_0, F, L)$
$(z_0, \square) \rightarrow (z_1, A, L)$
$(z_1, \square) \rightarrow (z_2, K, L)$
$(z_2, \square) \rightarrow (z_e, \square, R)$



Definition 4.1.1

Ein 7 - Tupel $(\Sigma, \Gamma, Z, \delta, z_0, z_E, \Box)$

1. $\sum$ endliche Menge
2. $\Gamma \supseteq \Sigma$ endliche Menge
3. Z endliche Menge
4. $\delta : Z \times \Gamma \rightarrow \gamma_{endl.}\{Z \times \Gamma \times \{L, R, N\}\}$
   DTM (deterministische) TM $\rightarrow Z \times \Gamma \times \{L, R, O\}$
5. $z_0 \in Z$
6. $z_E \in Z$
7. $\Box \in \Gamma \backslash \Sigma$ (Leersymbol)

-statt $\delta(q, a) \in \{(r, b, R), \dots\}$ schreiben wir auch $(q, a) \rightarrow (r, b, R)$
Konfiguration einer TM (Augenblickbeschreibung)
$\rightarrow$ Was steht auf dem Band ?
$\rightarrow$ Wo steht der Kopf ?
$\rightarrow$ In welchem Zustand befindet sich die TM ?

□	T	I	S	C	H	□	□

Konfiguration: TIzSCH

→ Konfiguration ist ein Wort aus $\Gamma^* z \Gamma^*$
→ Berechnung ist eine endliche Folge von Konfigurationen, die gemäß Überfürhungsfunktion auseinander hervorgehen

Bemerkung: In der Konfiguration stehht nur der Bandinhalt zwischen den Linken und Rechten $\Box$

Definition 4.1.2

Sei $M = (\Sigma, \Gamma, Z, \delta, Z_0, Z_E, \Box)$ eine TM.
Wir definieren eine binäre Relation $\vdash_M$ über $\Gamma^* \times \Gamma^*$ (Menge der Konfigurationen von M) wobei $K \vdash_M K'$ gerade bedeuten soll, dass K' in einem takt aus K hervorgeht (mittels einmaliger Anwendung von δ aus K gewonnen werden kann)

Sei $K = u\,\alpha\,z\,\beta\,v$ mit $u, v \in \Gamma^*; \alpha, \beta \in \Gamma^*, \alpha, \beta \in \Gamma, z \in Z$

$$K \vdash_M \begin{cases} u\,\alpha\,\gamma\,z'\,v & , falls\;\; z\beta \to z', \gamma, R \\ u\,z'\,\alpha\,\gamma\,v & , falls\;\; z\beta \to z', \gamma, L \\ u\,\alpha\,z'\,\gamma\,v & , falls\;\; z\beta \to z', \gamma, N \end{cases}$$

$\alpha\,z\,\beta \in \Gamma^* z \Gamma^*$
$\alpha\,z\,\beta \vdash_M \alpha'\,z'\,\beta'$
$\to \alpha'\,z'\,\beta'$ geht in einem Schritt weiter aus $\alpha\,z\,\beta$

- Schreibweise $K \nvdash \ldots$ K hat keine Folgekonfiguration
- Erweiterte Konfigurationsrelation: $\vdash_M^*$
  - $\to \;\; \vdash_M^*$ ist die reflexive und transitive Hülle von $\vdash_M$
  - $K_1 \vdash_M^* K_2 \ldots K_1$ geht in endlich vielen Schritten in K_2 über
  - $K_1 \vdash_M^* K_2$ gdw. $\bigvee_{n \in \mathbb{N}} \bigvee_{Q_1 \ldots Q_n} \quad K_1 \vdash Q_1 \wedge Q_1 \vdash Q_2, \ldots, Q_n \vdash K_2$
- $z_0\,w \ldots\ldots$ Startkonfiguration bei Eingabe $w = a_1 \ldots a_n$

	u_1	u_2	$\ldots$

↑
$z_0\;w = a_1 \ldots a_n$

Sei $w \in \Sigma^*$ eine Berechnung von m für w ist eine maximale (nciht verlängernde) Folge
$\to z_0 w \vdash \alpha_1 z_1 \beta_1 \vdash \ldots$ von Konfigurationen

Berechnungen - 3 mögliche Fälle

- Berechnungen ist akzeptierend, d.h. sie endet in einer Konfiguration $\alpha_n z_n \beta_n$,
- Berechnungen ist verwerfend, d.h. sie endet in einer Konfiguration $\alpha_n z_n \beta_n$, wobei $z_n \notin Z_E$ und $\delta(z_n, b) \neq \emptyset$. Dabei ist b das erste Zeichen von β_n
- Berechnung unendlich

Bei nicht (deterministisch) Turingmaschinen (NTM) kann es zu jedem Wort mehrere Berechnungen geben.

Berechnungsbäume

Berechnungsbäume können akzeptierende, verwerfende oder auch unendliche Berechnungen enthalten

Definition 4.1.3

Die von einer NTM akzeptierte Sprache ist:

$$\mathscr{L}(M) = \{w \in \Sigma^* | es\ gibt\ eine\ akzeptierende\ Berechnung\ von\ M\ für\ w\}$$

Bemerkung
$w \notin L(M)$ gdw. alle Berechnungen sind verwerfend oder unendlich

TM mit $\mathscr{L}(M) = \{a^n b^n c^n | n \geq 1\}$
$M = \big\{\{a, b, c\}, \{a, b, c, \$, \Box\}, \{z_0, z_1, z_2, \ldots, z_5, z_e\}, \delta, z_0, \{z_e\}, \Box\big\}$

	Zustand	Absicht
z_0	Startzustand	neuer Zyklus
z_1	ein a gemerkt	nächstes b suchen
z_2	ein a und ein b gemerkt	nächstes c suchen
z_3	ein a,b,c geilgt	rechten Rand suchen
z_4	rechter Rand erreicht	Test, ob alle c's getilgt
z_5	Test nicht erfolgreich	zum linken Rand und neuen Zyklus beginnen
z_6	alle c's getilgt	prüfe, ob alle a's, b's getilgt

□	a	a	a	b	b	b	c	c	□

$\uparrow_{z_0}$

$\rightarrow$ \$aa\$bb\$c□ $\Leftrightarrow$ \$\$a\$\$b\$\$□

	z_0		z_1		z_2		z_3		z_4	z_5	z_6
a	$z_1\$R$	$\rightarrow$	z_1aR							z_5aL	
			$\downarrow$								
b			$z_2R\$$	$\rightarrow$	z_2bR					z_5bL	
					$\downarrow$						
c					$z_3\$R$	$\rightarrow$	z_3cR		z_5cL	z_5cL	
\$	$z_0\$R$	$z_1\$R$	$z_2\$R$				$\downarrow$	$\nearrow$		$z_5\$L$	$z_0\$L$
□							$z_4\Box L$			$z_5\Box L$	$z_0\Box L$

$\rightarrow$ Ähnnlich bei $\{w\$w|w \in \{a,b\}^*\}$
$\{ww|w \in \{a,b\}^*\}$ geht auch, da wir nun auch dieMitte bestimmen können (Bsp: links+1,rechts+1,bis mitte)

Linear beschränkte Turingmaschinen (LBA)

LBA's sind spezielle Turingmaschine, die niemals den Bereich des Bandes verlassen auf dem die Eingabe steht.
Dazu zwäckmäßig, sich den rechten und linken Rand der Eingabe wie folgt zu markieren:

- linker Rand: im ersten Arbeitsschritt markieren
- rechter Rand: $\Sigma' \cup \Sigma \cup \{\hat{a}|a \in \Sigma\}$
  Vereinbarung: Die eigentliche Eingabe wird auf dem Band repräsentiert durch $a_1a_2\ldots\hat{a}_n$

Definition 4.2.1

Eine NTM heißt linear beschränkt (LBA), falls für alle Wörter $x = a_1\ldots a_n \in \Sigma^*$ und für alle Konfigurationen $\alpha z\beta$ mit $z_0\, a_1\ldots\hat{a}_n \vdash_M^* \alpha z\beta$ gilt: $|\alpha\beta| = n$

Satz 4.2.2

Eine Sprache ist genau dann kontextsensitiv, wenn sie von einem LBA akzeptiert werden kann.

"⇒:" Sei A kontextsensitiv, also $A = \mathscr{L}(G)$ mit $G = (\Sigma, N, S, R)$ vom Typ 1 (nicht verkürzend)
Wir beschreiben informal eine TM, die A akzeptiert

Bei Eingabe von $x = a_1 \ldots a_n$ wähle M nichtdeterministisch eine Regel $u \rightarrow v \in R$ aus
M sucht ein Vorkommen von v auf dem Band und ersetzt es durch u (Falls u kleiner als v, werden die restlichen buchstabene herangerutscht)

Die Maschine stoppt im Endzustand falls nur noch das Startsymbol S auf dem Band steht
$x \in L \Leftrightarrow$ es gibt eine Ableitung $S \vdash \ldots \vdash x$
$\Leftrightarrow$ es gibt eine Rechnung von M, die diese Ableitung in umgekehrte Richtung simuliert
$\Leftrightarrow x \in L(M)$

Da für alle $(u, v) \in R$ gilt $|u| \leq |v|$ ist M linear beschränkt

Satz 4.2.2

Eine Sprache ist genau dann kontextsensitiv, wenn sie von einem LBA akzeptiert werden kann
"⇐:"
Sei $A = \mathscr{L}(M)$ für einen LBA M
Wir beschreiben eine kontextsensitive Grammatik, deren Zeichenketten Konfigurationen zu M darstellen. Konfigurationen (ihre Beschreibungen) dürfen nicht länger als die Eingabe sein
Sei $\Delta = \Gamma \cup (Z \times \Gamma)\$ \rightarrow \ azbcd$ wird dargestellt als $a(z,b)cd$|a(z,b)cd|=4$\$ \rightarrow$ bezogen auf das erweirterte Alphhabet
$\delta(z, a) \ni (z', b, L) \rightarrow$ kann beschrieben werden durch kontextsensitive Regeln der Form:
$c(z, a) \rightarrow (z', c)\ b$ für alle $c \in \Gamma$

Diese Regelmenge sei R'
Falls $K \vdash_M^* K'$ gilt $\tilde{K} \vdash \tilde{K}'$ mittels Regelmenge R' wobei $\tilde{K}$ die oben angegebene Darstellung der Konfiguration ist

Beweisskizze $\longrightarrow$ Grammatik

Automat
$\downarrow$
Regelmenge $R \longrightarrow R'$ (die Ableitungen sind Konfigurationen des Automaten, sie vollziehen die
Arbeit des Automaten nach)
$\downarrow$
dazu muss das Eingabewort erst einmal auf dem Band stehen
$\downarrow$
Regeln (1),(2),(3)
erzeugen beliebiges Wort (auf das Band) in doppelter Ausführung
(1.Komponente, 2. Komponente)
$\downarrow \downarrow$
mit Regeln aus R' feststellen, aufheben für die Ausgabe
ob man zum Endzustand kommt
$\downarrow$ falls ja
alle ersten Komponenten löschen

Sei $G = (\Sigma, N, S, R)$ mit $N = \{S, A\} \cup (\Delta \times \Sigma)$
$R = \{S \to A(\hat{a}, a) | a \in \Sigma\}$ (1)
$\cup \{A \to A(a, a) | a \in \Sigma\}$ (2)
$\cup \{A \to \big((z_0, a), a\big) | a \in \Sigma\}$ (3)
$\cup \{\alpha_1, a)(\alpha_2, b) \to (\beta_1, a)(\beta_2, b) | \alpha_1\alpha_2 \to \beta_1\beta_2 \in R' \; a, b \in \Sigma\}$ (4)
$\cup \{\big((z, a), b\big) \to b | z \in Z_E, a \in \Gamma, b \in \Sigma\}$ (5)
$\cup \{(a, b) \to b | a \in \Gamma, b \in \Sigma\}$ (6)

$\to$ Durch (1),(2),(3) sind die Ableitungen der Form:
$S \vdash^* \big((z_0, a_1), a_1\big)(a_2, a_2) \dots (a_{n-1}, a_{n-1})(\hat{a}_n, a_n)$ möglich

1. Komponente: Startkonfiguratio
2. Komponente: Eingabewort

$c(z, a) \to (z', c) \; b$,$c \in \Gamma$

Auf den 1. Komponenten wird mit R'(Regelart (4)) die Rechnung von M simuliert, bis ein Endzustand erreicht ist
$\vdash^* (\gamma, a_1)\dots(\gamma_{k-1}, a_{k-1}), \big((z,\gamma_k), a_k\big), (\gamma_{k+1}, a_{k+1})\dots(\gamma_n, a_n)$ mit $z \in Z_E; \gamma_i \in \Gamma; a \in \Sigma$
$\rightarrow$ Danach mittels (5),(6) alle ersten Komponenten löschen
$\rightarrow$ Es bleibt $a_1 \dots a_n$ übrig $\square$

4.3 Grammatiken

$G = \big(\{a,b,c\}, \{S,A,B,C\}, S, R\big)$
$R = \{S \rightarrow aSBC | aBC$
$(2)\ CB \rightarrow BC$
$(3) \begin{cases} aB \rightarrow ab \\ bB \rightarrow bb \end{cases}$
$(4) \begin{cases} bC \rightarrow bc \\ cC \rightarrow cc \end{cases}$

S $\vdash aSBC$
$\vdash aaSBCBC$
$\vdash aaaSBCBCBC$
$\vdash aaaaBCBCBCBC$
$\vdash^5_{(2)} aaaaaBBBBCCCC$
$\vdash^4_{(3)} aaaabbbbCCCC$
$\vdash^4_{(4)} aaaabbbbcccc$

<u>Behauptung:</u>

$\mathscr{L}(G) = \{a^n b^n c^n | n \geq 1\}$
"$\subseteq$:" für jede abgeleitete Zeichenkette $z \in \{a,b,c,A,B,C,S\}^*$
gilt $\#_a(z) + \#_A(z) = \#_b(z) + \#_B(z) = \#_c(z) + \#_C(z)$
$\rightarrow$ für jedes abgeleitete Wort $w \in \{a,b\}^*$ gilt: $\#_a(z) = \#_b(z) = \#_c(z)$
Reihenfolge der Buchstaben:
- a's entstehen links und werden nie getauscht
- B's können nur zu Terminalen b werden (3), wenn sie über alle C#s nach links getauscht werden

"$\supseteq$:" Jedes $w = a^n b^n c^n, n \geq 1$ kann erzeugt werden
$(n-1)\times\ S \rightarrow aSBC\ a^{n-1}S(BC)^{n-1}$
$1\times\ S \rightarrow aSBC\ a^n(BC)^n$

$\frac{n(n-1)}{2} \times CB \to BC \; a^n B^n C^n$
$1\times \; aB \to ab \; a^n b B^{n-1} C^n$
$(n-1)\times \; bB \to bb \; a^n b^n C^n$
$1\times \; bC \to bc \; a^n b^n c C^{n-1}$
$(n-1)\times \; cC \to cc \; a^n b^n c^n$

$G = \big(\{a\}, \{S, L, R\}, S, R\big)$
$R = \{S \to a|aa|LSR$
(2) $Laa \to aaaaL$
(3) $LaaR \to aaaa\}$

$S \vdash LSR$
$\vdash LLSRR$
$\vdash LLLSRRR$
$\vdash LLLaaRRR$
$\vdash_{(3)} LLaaaaRR$
$\vdash_{(2)} LaaaaLaaRR = La^4La^2RR$
$\vdash_{(3)} La^8R$
$\vdash_{(2)} a^4La^6R$
$\vdash_{(2)} a^8La^4R$
$\vdash_{(2)} a^{12}La^2R$
$\vdash_{(3)} a^{16}$

$\mathscr{L}(G) = \{a^{2^n} | n \in \mathbb{N}\}$

Satz 4.3.1 CF $\subsetneq$ CS

Beweis: $\{a^n b^n c^n | n \geq 1\} \in \; CS \setminus CF$

Satz 4.3.2 CS $\subsetneq \mathscr{L}_0$

Beweis: $\nearrow$ später

Satz 4.3.3

Es gibt Sprachen, die von keiner Grammatik erzeugt werden können
Beweis: Es sei Σ ein Alphabet Σ^*, die Menge aller Wörter über Σ ist abzählbar

unendlich groß
$\rightarrow \mathcal{P}(\Sigma^*)$, die Menge aller Wörter über Σ, ist überabzählbar unendlich groß

Wie viele Grammatiken $G = (\Sigma, N, S, R)$ gibt es?
N und R sind endliche Mengen und lassen sich damit geeignet, als z.B:
$A_1, A_2, \dots, A_k; R_1, R_2, \dots, R_l$ beschreiben
$\rightarrow$ Damit ist G als Wort über einem geeignetem Alphabet beschreibbar
$\rightarrow \tilde{\Sigma}^* = \Sigma \cup \{A\} \cup \{0, 1, \dots, 9\} \cup \{\rightarrow, (,), , , ;, \dots\}$
$\tilde{\Sigma}^*$, die Menge aller Grammatiken mit Alphabet Σ ist also abzählbar unendlich groß

Das Wortproblem für kontextfreie Sprachen

geg: kontextsensitive Grammatik G und ein Wort $w \in \Sigma^*$

Frage: ist $w \in \mathscr{L}(G)$?
$\rightarrow$ Gibt eas einen Algorithmus, der bei Eingabe einer kontextsenssitiven Grammatik G und eines Worter $w \in \Sigma^*$ in endlicher Zeit $w \in \mathscr{L}(G)$ ausgibt? $\rightarrow$ Ja !

Idee: $|p| \leq |q|$ für $(p, q) \in \delta$
$\rightarrow$ für $w \in \mathscr{L}(G)$, $|w| = n$ und $S \vdash^* w$ folgt, dass alle $z \in (\Sigma \cup N)^*$, die während der Ableitung von w entsehen, höchsten Länge n haben
$\rightarrow$ Es gibt nur endlich viele $z \in (\Sigma \cup N)^*$ $|z| \leq n$
$\rightarrow$ durch Systematisches Probieren kann entschieden werden, ob $w \in \mathscr{L}(G)$ oder $w \notin \mathscr{L}(G)$

$T^n_m(x \in \{\Sigma \cup N\}^* \Big| \ |x| \leq n$ und x lässt sich in höchstens m Schritten aus S ableiten)
T^n_m $n \geq 1$ induktiv über m definiert
$T^n_0 = \{S\}$
$T^n_{m+1} = Abl_n(T^n_m)$ wobei $Abl_n(x) = X \cup \{x \in \{\Sigma \cup N\} \Big| \ |x| \leq n$ und $w' \vdash w$ für ein $w \in X\}$

Bemerkung: das gilt für Typ-1 Grammatiken, bei Typ-0 können die Zeichenketten der Länge n auch aus längeren Zeichenketten abgeleitet werden

$G = \{a, b, c\}, \{A, B, C, S, S, R\}$
$R = \{\ S \rightarrow aSBC|aBC$ $T^4_0 = \{S\}$
$aB \rightarrow ab$ $T^4_1 = \{S, aSBC, aBC\}$
$bB \rightarrow bb$ $T^4_2 = \{S, aSBC, aBC, abC\}$

$bC \to bc$ $T_3^4 = \{S, aSBC, aBC, abC, abc\}$
$cC \to cc$ $T_4^4 = T_3^4$
$CB \to BC\}$

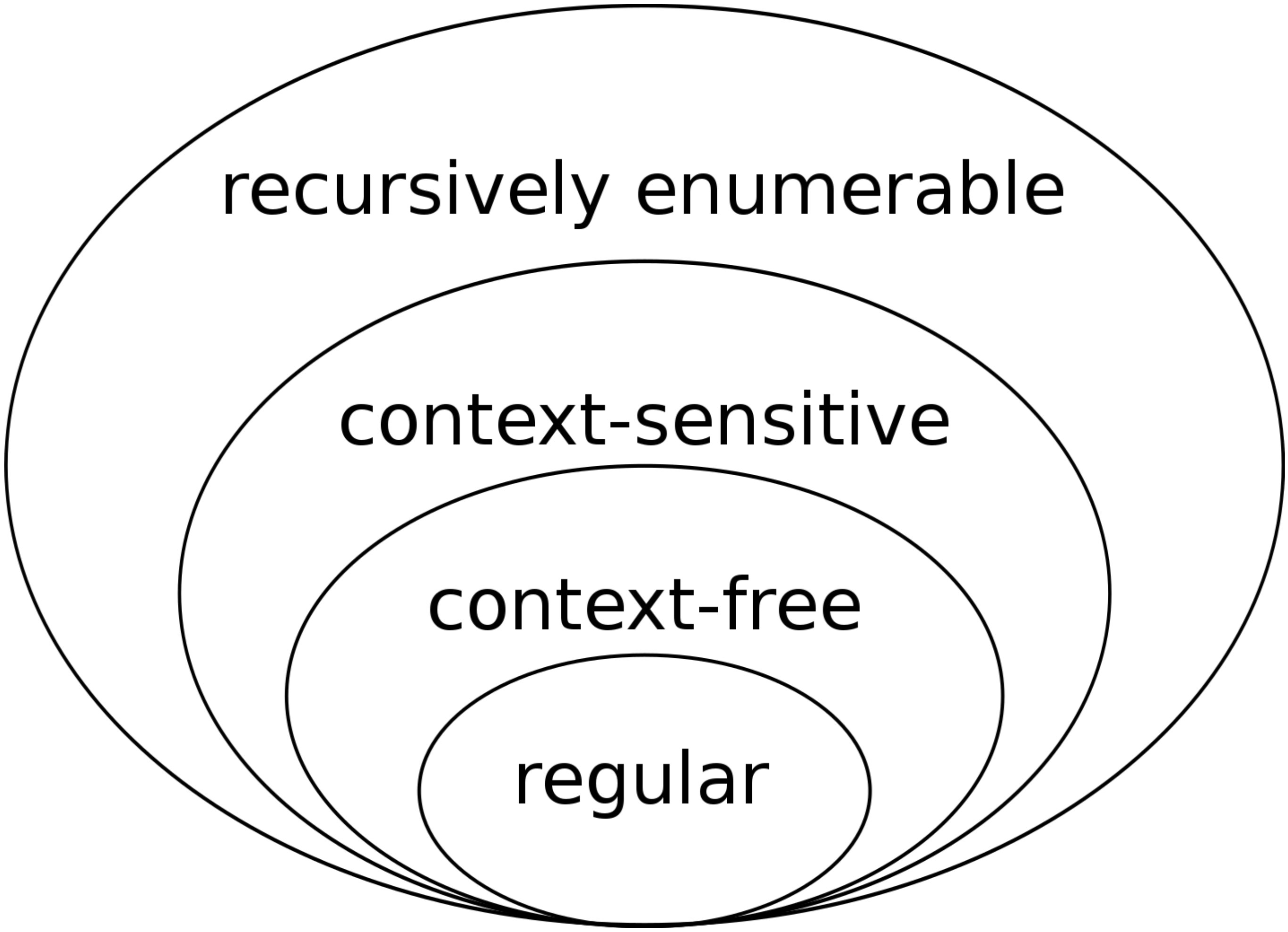
- Es gibt nur endlich viele Zeichenketten der Länge $\leq n$ in $(\Sigma \cup N)^*$
  - $\to \bigcup\limits_{m \geq 0} T_m^n$ ust endliche Menge
  - $\to$ Es gibt $m \geq 0$, so dass gilt: $T_m^n = T_{m+1}^n + \ldots +$ da $T_0^n \subseteq T_1^n$
- Falls w mit $|w| = n$ in $w \in \mathscr{L}(G)$ liegt, muss $w \in \bigcup\limits_{m \geq 0} T_m^n$, also $w \in T_m^n$ für ein m gelten
  Algorithmus

INPUT: $G = (\Sigma, N, S, R)$, kontextsensitiv $w \in (\Sigma \cup N)^*$ mit $|w| = n$

OUTPUT: $w \in \mathscr{L}(G)$ oder $w \notin \mathscr{L}(G)$

Berechne: $T_0^n, \ldots$ bist $T_{m+1}^n = T_m^n$ gilt $\to$ Teste ob $w \in T_m^n$: Ja?$w \in \mathscr{L}(G)$
Nein? $w \notin \mathscr{L}(G)$

Übersicht:



Quelle: https://de.wikipedia.org/wiki/Chomsky-Hierarchie#/media/Datei:Chomsky-Hierarchie.svg

- TM
- LBA
- PDA
- DFA/NFA

4.5 Abschlusseeigenschaften von CF

Satz 4.5.1: CS ist abgeschlossen bzgl:

1. Vereinigung
2. Produkt
3. Kleene-Abschluss
4. Komplement
5. Durschnitt

Beweis $L_1 \subseteq \Sigma_1^*; L_2 \subseteq \Sigma_2^*$
$G_1 = (\Sigma_1, N_1, S_1, R_1)$ mit $\mathscr{L}(G_1) = L_1$, $N_1 \cap N_2 = \emptyset$
$G_2 = (\Sigma_2, N_2, S_2, R_2)$ mit $\mathscr{L}(G_2) = L_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$

1. $G = (\Sigma, N_1 \cup N_2 \cup S, S, R)$

   $$R = \begin{cases} R_1 \cup R_2 \cup \{S \to S_1, S \to S_2\} & falls\ \lambda \notin L_1, \lambda \notin L_2 \\ R_1 \cup R_2 \cup \{S \to S_1, S \to S_2\} \cup \{S \to \lambda\} \backslash \{S_1 \to \lambda, S_2 \to \lambda\} & falls\ \lambda \in L_1\ oder\ \lambda \in L_2 \end{cases}$$

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

2. G_1, G_2 Normalformalgrammatiken
   $\to$ auf linken Seiten nur Nichtterminale, für jedes $a \in \Sigma$ A_a $A_a \to a$ und in allen Regeln a durch A_a ersetzt
   Bsp:
   $S \to S_1 S_2$ $G = (\Sigma, N_1 \cup N_2 \cup S, S, R)$
   $AB \to Ba$
   $R = R_1 \cup R_2 \cup \{S \to S_1, S \to S_2\} \cup \{S \to \lambda\} \backslash \{S_1 \to \lambda, S_2 \to \lambda\} falls\ \lambda \in L_1\ oder\ \lambda \in L_2$

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

   $(*)$ sonst könnte es in $R_1 \cup R_2$ Regeln $\alpha \to \beta$ 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 überraft

   $\mathscr{L}(G) = L_1 L_2$ und G ist kontextsensitiv
   $G = (\Sigma, N, S, R)$ $\mathscr{L}(G) = L$
   $G' = (\Sigma, N \cup \{S'\}, S', R \cup \{S' \to \lambda, S' \to SS'\})$
   $\mathscr{L}(G') = L^*$, aber G' ist nicht kontextsensitiv (S' kommt rechts vor)
   $\to G' = (\Sigma, N \cup \{S', S''\}, S'', R \cup \{S'' \to \lambda, S'' \to S', S' \to S, S' \to SS'\} \backslash \{S \to \lambda\})$

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? $\rightarrow f(n,m) = m+n$
$\rightarrow 99!$
$\rightarrow f(n) = n!$
$\rightarrow g(n) =$ 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 $x_{i1} \leq x_{i2} \leq \dots \leq x_{i1}$

5.2 Grundlagen

$\mathbb{N} = \{0, 1, 2, 3, \dots\}$
$\mathbb{N}^{\mathbb{N}} = \{f : f : \mathbb{N} \rightarrow \mathbb{N}\}$ "von $\mathbb{N}$ nach $\mathbb{N}$", $D_f = \mathbb{N}$
$\tilde{\mathbb{F}}_1 \mathbb{N}^{\mathbb{N}}$ $\mathbb{F}_1 = \{f \middle| f : \mathbb{N} \rightarrow \mathbb{N}\}$ "aus $\mathbb{N}$ in $\mathbb{N}$", $D_f \subseteq \mathbb{N}$
$\tilde{\mathbb{F}}_2 \mathbb{N}^{\mathbb{N} \times \mathbb{N}}$ $\mathbb{F}_2 = \{f \middle| f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}\}$
$\rightarrow$ Menge der totalen beliebigstelligen Fkt. über $\mathbb{N}$: $\tilde{\mathbb{F}} = \bigcup\limits_{i=1}^{\infty} \tilde{\mathbb{F}}_i$
$\rightarrow$ Menge der beliebigstelligen Funktion über $\mathbb{N}$: $\tilde{\mathbb{F}} = \bigcup^{\infty} \tilde{\mathbb{F}}_i$

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 $f_1(0)$ $f_1(1)$ $f_1(2)$ $f_1(3)$ $f_1(3)\dots$
f_2 $f_2(0)$ $f_2(1)$ $f_2(2)$ $f_2(3)$ $f_2(3)\dots$

f_3 $f_3(0)$ $f_3(1)$ $f_3(2)$ $f_3(3)$ $f_3(3)\dots$
f_4 $f_4(0)$ $f_4(1)$ $f_4(2)$ $f_4(3)$ $f_4(3)\dots$
$\vdots$
Wir konstruieren die Funktion g : seien $i, j \in \mathbb{N}$ $i \neq j$

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

$\rightarrow$ 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?

$\rightarrow$ Versuche:

$$f_1(n) = \begin{cases} 1 & falls\ Anfangsabschnitt\ der\ Dezimalbruchentwicklung\ von\ \pi\ ist \\ 0 & sonst \end{cases}$$

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

$$f_2(n) = \begin{cases} 1 & falls\ n\ irgendwo\ in\ der\ Dezimalbruchentwicklung\ von\ \pi \\ 0 & sonst \end{cases}$$

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

5.3 Turing Berechnbarkeit

bisher: Turingmaschine akzeptiert Sprachen
$\rightarrow$ Wir wollen den Begriff modifizieren, um das Berechnen von Funktionen zu erfassen

$\rightarrow$ Für Berechnbarkeit nutzen wir deterministische Turingmaschinen
$\rightarrow \delta \ldots$ 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
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 $\bigvee\limits_{z \in Z_E} \bigvee\limits_{\alpha, \beta \in \Gamma^*} z_0 w \vdash_M^* \alpha z \beta$	-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 $M = \left(\{0, 1, 2.., 9, \#\}, \Gamma, Z, \delta, z_0, Z_E, \Box\right)$, so dass für alle $(n_1, \ldots, n_i) \in \mathbb{N}^i$ gilt
1) $(n_1, \ldots, n_i) \in D_f$ gdw. $\bigvee\limits_{z \in Z_E} \bigvee\limits_{\alpha, \beta \in \Sigma^*} z_0 n_1 \# n_2 \# \ldots \# n_i \vdash_M^* \alpha z \beta$
2) Für alle $(n_1, \ldots, n_i) \in D_f$ gilt $f(n_1, \ldots, n_i) = m$ gdw.
$\bigvee\limits_{a \in \Gamma^*} \bigvee\limits_{z \in Z_E} z_0 n_1 \# n_2 \# \ldots \# n_i \vdash_M^* \alpha z m$

Definition 5.3.2

Eine Funktion $f: \Sigma^* \to \Delta^*$ heißt Turing berechenbar gdw.: Es gibt eine DTM $M = (\Sigma, \Gamma, Z, \delta, Z_0, Z_E, \square), \Delta \subseteq \Gamma \backslash setminus\{\square\}$: so dass für alle $w \in \Sigma^*$ gilt:
1) $w \in D_f \leftrightarrow \bigvee\limits_{z \in Z_E} \bigvee\limits_{\alpha,\beta \in \Sigma^*} z_0 w \vdash_M^* \alpha z \beta$
2) $f(w) = y \leftrightarrow \bigvee\limits_{z \in Z_E} \bigvee\limits_{\alpha,\beta \in \Sigma^*} z_0 w \vdash_M^* azy$

Bsp 1

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

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

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

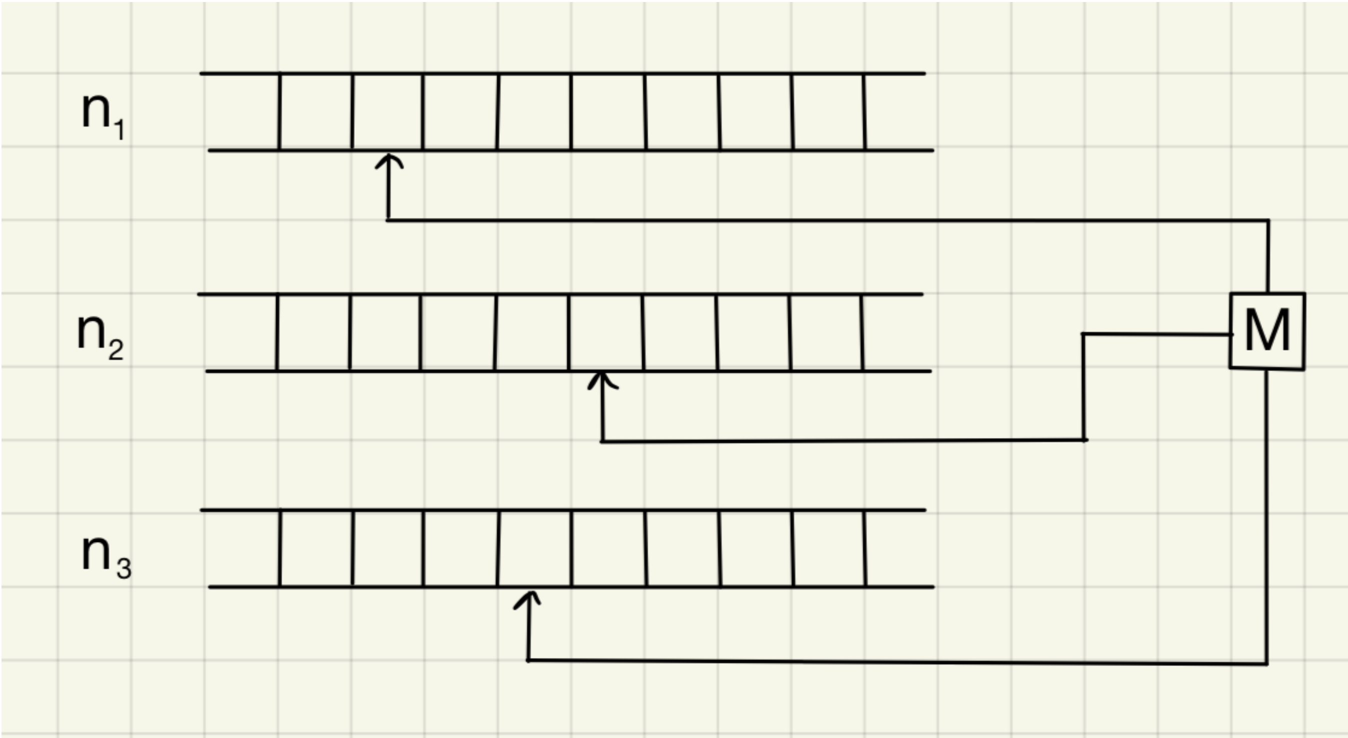
nirgends definierte Funktion $f(w) = n.d$

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

5.4 Andere Typen von Turingmaschinen

$\to$ Der bisher eingeführte Typ wird of als Standard TM bezeichnet
$\to$ Man kann die Definition auf verschiedene Weisen erweitern

1. mehrbändige Turingmaschine:



$\delta : Z \times \Gamma^k \to Z \times \Gamma \times \{L, R, N\}^k$
$k...$ Anzahl der Bänder
Befehle haben die Form $(k = 3)$
$zx_1x_2x_3 \to z'x_1'x_2'x_3' \quad \sigma_1\sigma_2\sigma_3$
$x_i...$ gelesenes Zeichen auf Band i
$x_i'...$ geschriebenes Zeichen auf Band i
$\sigma_i...$ Kopfbewgung auf Band i

$\to$ die Eingabe steht auf Band 1, die anderen Bänder sind leer
$\to$ nächster Schritt hängt vom aktuellen Zustand und den k-Bandinhalten ab und dem Lese-/Schreibkopf ab
$\to$ jedes der k-Symbole kann überschrieben werden
$\to$ jeder Kopf kann sich bewegen (unabhängig voneinander)

Konfiguration einer k-Band TM: $< \alpha_1, \alpha_2, ..., \alpha_k > z < \beta_1, \beta_2, ..., \beta_k >$
$\to k = 3:$ - 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

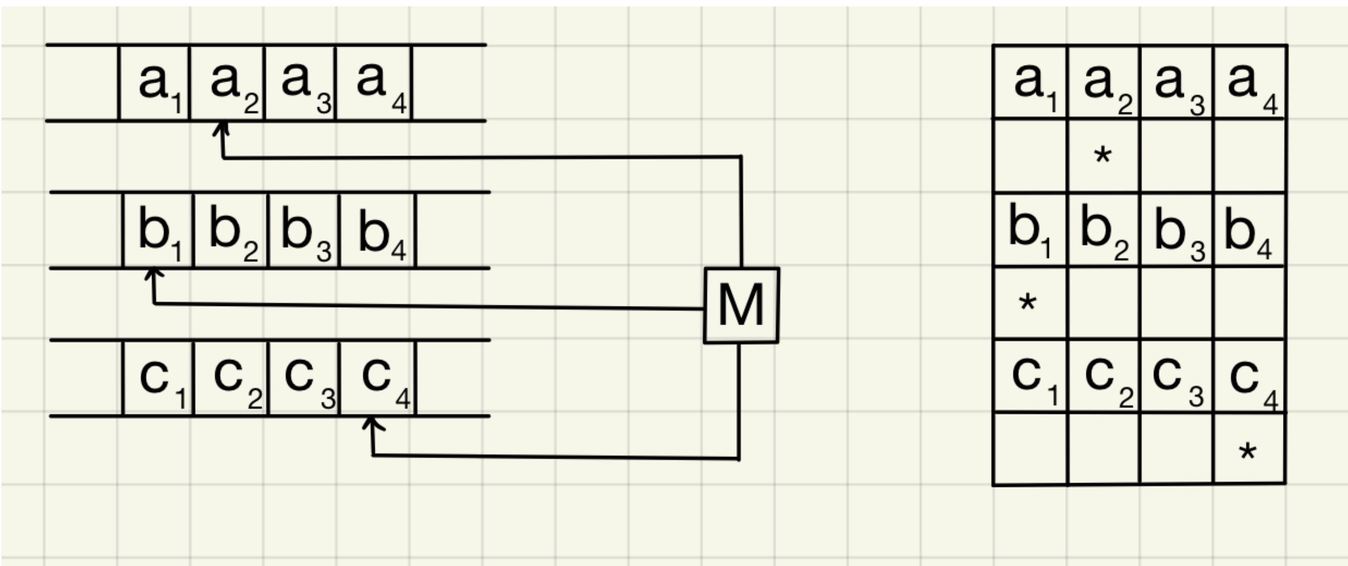
Satz 5.4.1

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.

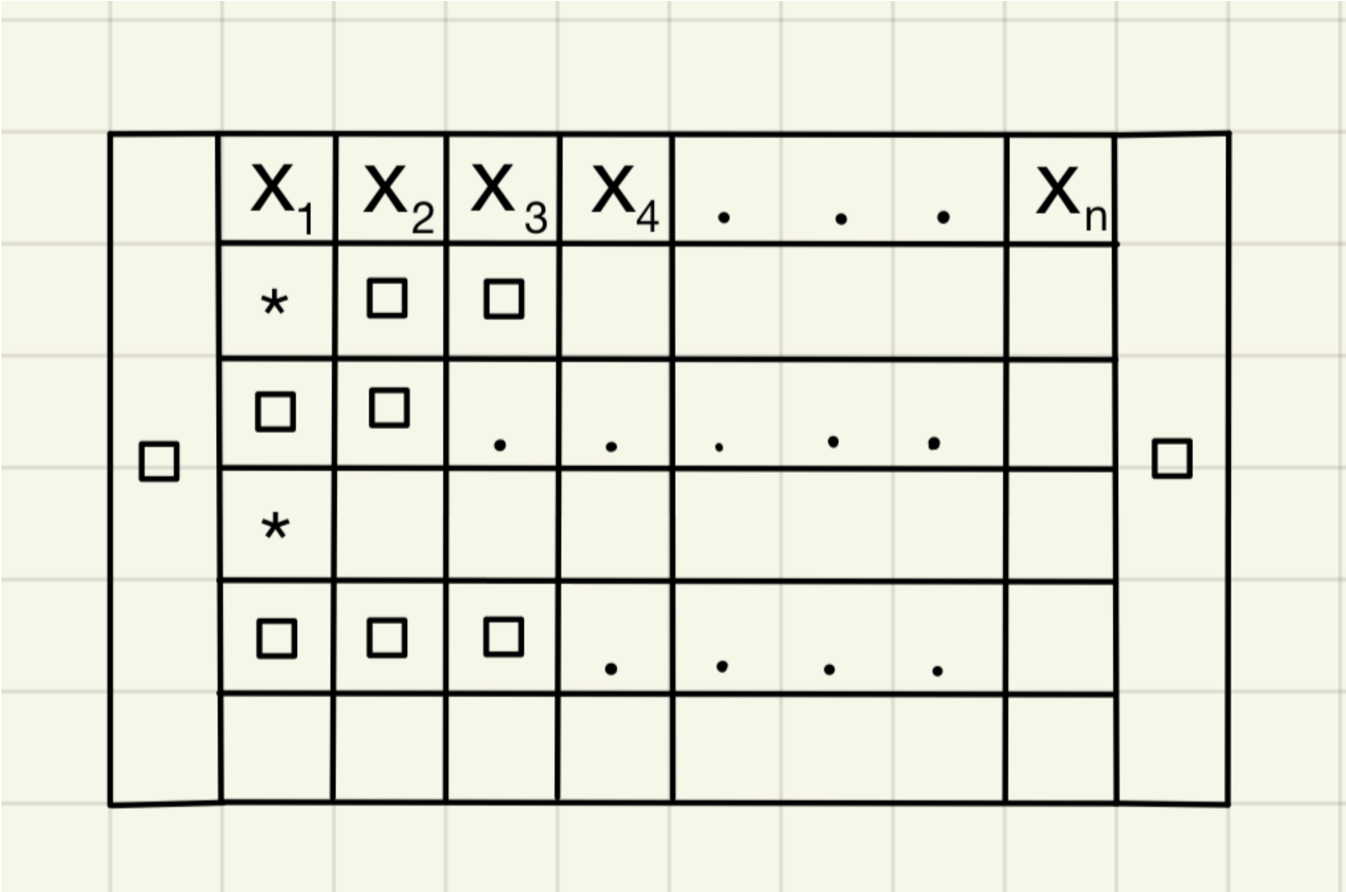
Wir konstuieren: $M' = (\Sigma, \Gamma', Z', \delta', z_0', Z_E', \Box)$

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



$\rightarrow$ in jeder Zelle des Bandes steht ein 6-Tupel:
Spur $2i - 1 \ldots$ Bandinhalt von Band i
Spur $2i \ldots$ Kopfposition von Kopf i

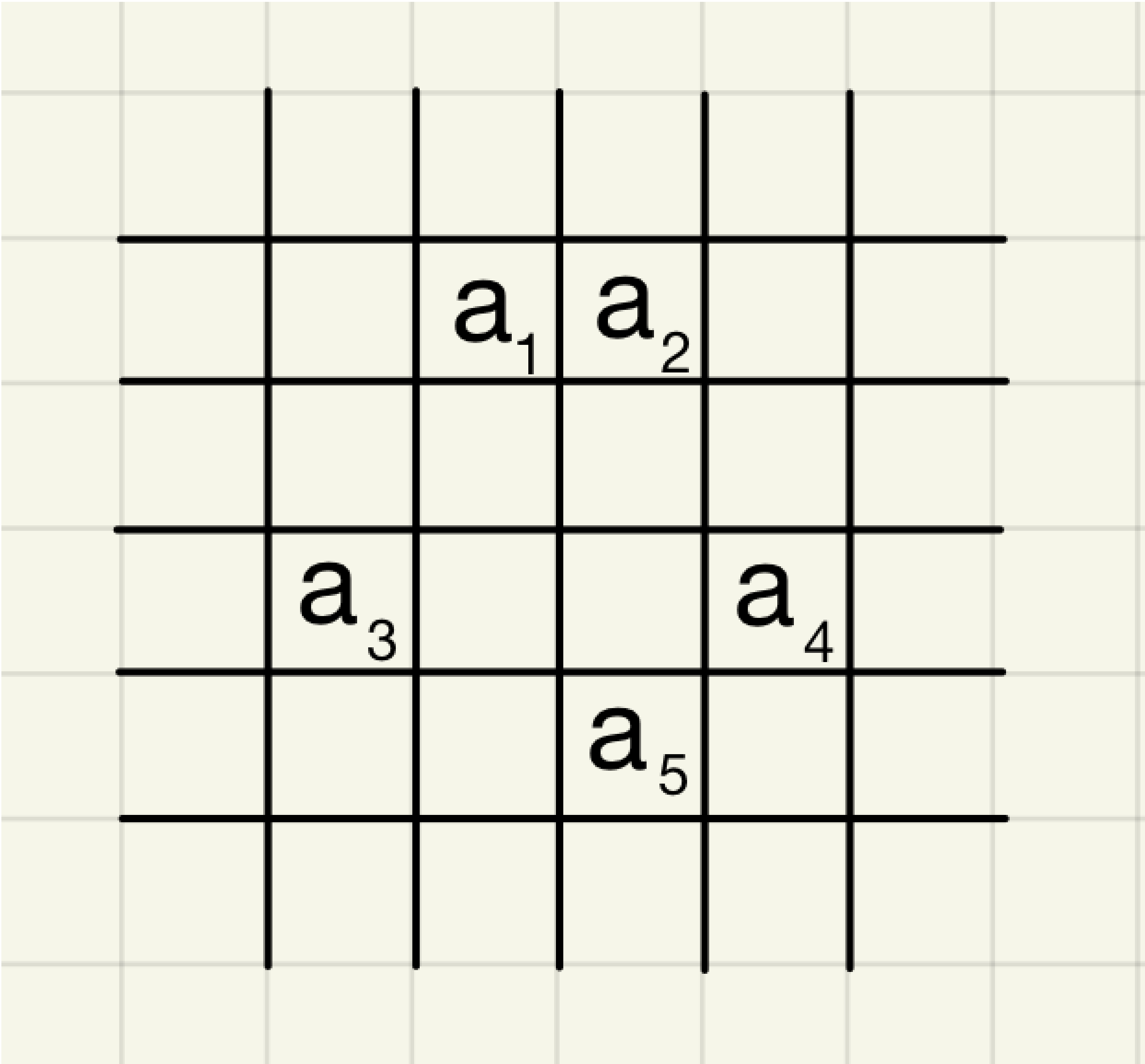
$\rightarrow$ Eingabe: $x_1, \ldots, x_n \in \Sigma^*$
$\rightarrow$ M' erzeugt Startkonfiguration in Spurendarstellung



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 $*$-$Markierungen$
- $M's$ 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 $\rightarrow$ $M's$ kann Überführungsregel aus δ anwenden
- auf dem Weg nach links über alle $*$-$Markierungen$ hinweng 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 $B = \{N, S, W, O, H\}$



$\to$ auch hier ist eine Simulation durch eine (Standard) TM möglich
$\to$ in jeder Zeile stehen endlich viele Symbole
$\to$ der Bandinhalt kann durch ein Recht eingegrenz werden
$\to$ Zeile für Zeile auf eindimensionales Band schreiben
$\to$ 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 - Berechenbarkeit 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
      $\rightarrow$ alle anderen Berechnungsmodelle lassen sich durch TM's simulieren
      $\downarrow$ dann gehen wir davon aus, dass es den intuitiven Begriff beschreibt
      (Beweis können wir es nicht)

These 5.5.1 Churche These

$\rightarrow$ 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

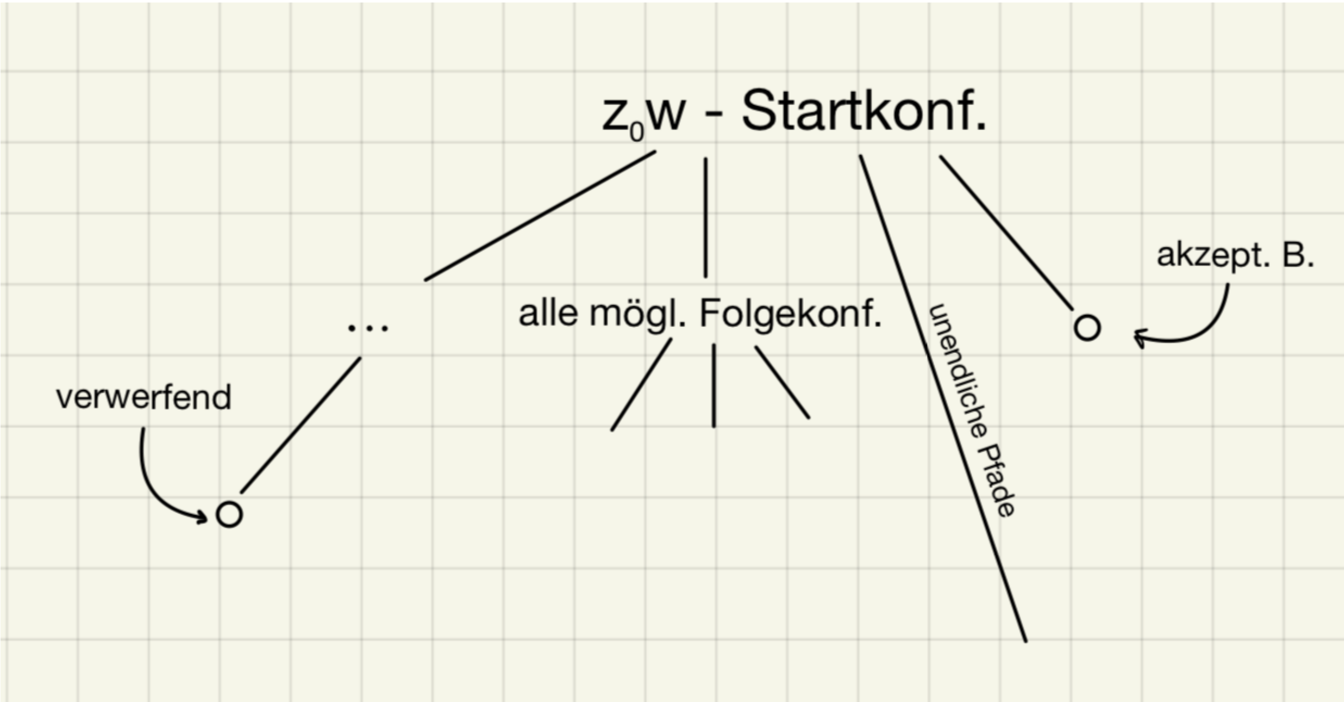
Jede von einer NTM akzeptierte Sprache kann auch von einer DTM akzeptiert werden

Berechnungsbaum einer NTM

Beweis

konstuiere eine DTM T', die eine NTM T simuliert
- DTM durchläuft den Berechungsbaum 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)(weiil es unendliche Pfade im Berechnungsbaum gibt)
  $\rightarrow \mathscr{L}(T) = \mathscr{L}(T')$



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:
$c_A(w) \begin{cases} 1 & w \in A \\ 0 & \notin A \end{cases}$
$\rightarrow$ die Funktion ist total

Definition 6.1.1

Eine Menge $A \subseteq \Sigma^*$ heißt entscheidbar gdw. c_A berechenbar ist REC (recusrive sets)
$\rightarrow$ Menge aller entscheidbaren Mengen

Bemerkung:

- c_A ist Turingberechenbar bedeutet es gibt DTM, die für jedes $w \in \Sigma^*$ anhalt, 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:
- $A = \{n \in \mathbb{N} | n\ ist\ Primzahl\}$ IN:$n \in \mathbb{N}$, Frage: Ist n eine Primzahl?
- $B = \{n \in \mathbb{N} | n\ und\ n+2\ sind\ Primzahlen\}$
- $C = WORT_2 = \{(G,w) | G\ kf\ und\ w \in \mathscr{L}(G)\}$ IN: $G\ kf, w \in \Sigma^*$, Frange: $w \in \mathscr{L}(G)$?
- $D = \{f : 2^k - 1\ ist\ Primzahl\}$
- $E = \{n \in \mathbb{N} | n\ ist\ gerade\ und\ als\ Summe\ zweier\ Primzahlen\ darstellbar\}$

Satz 6.1.2

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

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

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

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

1. $WORT_1 \in REC$
   $\rightarrow$ DFA's liefern nach Abarbeitung eines Wortes stets eine Antwort in Form eines akzeptierenden oder nicht akzeptierendes Zustands
   $\rightarrow$ Sei $A \in REG$, $A = \mathscr{L}(M)$ für einen DFA M
   $\rightarrow$ Wir konstuieren eine TM, die bei Eingabe w den DFA simuliert und 1 ausgibt, falls der DFA in einem akzeptierenden Zustand endet, sonst
   $\rightarrow$ 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: $CS \subsetneq REC$
$\rightarrow$ Wir konstuieren eine Sprache die Entscheidbar ist, aber nicht kontextsensitiv
$\rightarrow$ Diagonalisierung Es seien $G_1, G_2, \ldots$ alle kontextsensitiven Grammatiken und $w_1, w_2, \ldots$ alle Wörter über einem Alphabet Σ

	w_1	w_2	w_3	$\ldots$
G_1	1	0	0	
G_2	0	1		
G_3	1	0	0	
G_4				0
$\vdots$				

$\rightarrow$ Tabelleneintrag (i, j)
1 - falls $w_j \in \mathscr{L}(G)$
0 - sonst
$\rightarrow$ Es sei $c = \{w_1 | w_i \notin \mathscr{L}(G_i)\}$ "alle Wörter in deren Spalte auf der die Hauptdiagonale 0 sieht"

$c \in REC$, da $WORT_1 \in REC$
Es gibt $w_j \mathscr{L}(G_j) \Leftrightarrow w_j \notin C$ für alle $j \in \mathbb{N} (*)$
Annahme: c ist kontextsensitiv
Dann gilt es $j \in \mathbb{N}$ mit $\mathscr{L}(G_j) = c \rightarrow$ Das ist ein Widerspruch

Satz 6.1.3

REC ist abgeschlossen bezüglich:

Komplement, Vereinigung, Durschnitt 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. konstuieren $T_{A \cup B}$, 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

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

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
$A = \{(n,m) \in \mathbb{N}^2 | \bigvee_{k \geq 1} n^k + m^k\ prim\}$ IN: $n, m \in \mathbb{N}$ Frage: Gibt es
$k \geq 1\ n^k + m^k\ Prim?$
$B = \{(x,y) \in \mathbb{N}^3 | \bigvee_{k \in \mathbb{N} \wedge k > 2} x^k + y^k = z^k\}$ IN: $(x,y,z) \in \mathbb{N}^3$ Frage: Gibt es
$n > 2\ x^k + y^k = z^k?$
$C = \{n \in \mathbb{N} \big| n\ ist\ gerade\ und\ als\ Differenz\ zweier\ Primzahlen\ darstellbar\}$

Definition 6.2.2

Eine enge $A \subseteq \Sigma^*$ heißt Rekursiv aufzahl gdw. entweder: -) $A = \emptyset$ oder
-) es gibt eine berechenbare, totale Funktion $\mathbb{N} \to \Sigma^*$ mit $w_f = A$

RE (rekursive enumarable)... Menge aller rekursiv aufzahlbaren Mengen

Bemerkung:

- $A = \{f(0), f(1), \dots\}$ "f zahlt 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 angebbar

- rekursiv Aufzählbarkeit und Abzählbarkeit ist nicht das gleiche

  A rekursiv aufzählbar $\Rightarrow A$ abzahlbar
  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

"⇒:"

Sei A aufzählbar mittels f. Ein Semi-entscheidungsverfahren für A ist
Eingabe $w \in \Sigma^*$
for $i = 1, 2, 3, \ldots$ do
if $f(i) = w$ then output 1

"⇐:" Wir geben ein Algorithmus an, der eine Aufzählende Funktion berechnet
$j = 0$
$t = 0$
repeat
erzeuge $\Sigma^{\leq t} = \{w_1, \ldots, w_2\} = \Sigma^0 \cup \Sigma^1 \cup \ldots \cup \Sigma^k$
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
"$\supseteq$:" $w \in A \to$ Es gibt z, so dass T_A angewendet auf w noch z Takten 1 ausgibt
$\to$ 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

$RE = \mathscr{L}_0$
Beweis: Satz 4.3.3 und Folgerung 6.2.4
$\mathscr{L}_0 = \{A : \bigvee_{TM\ M} L(M) = A\}$
$\to$ noch offen: $CS \subsetneq \mathscr{L}_0$, bisher gezeigt: $CS \subsetneq REC, RE = \mathscr{L}_0$
$\to$ gzz. $REC \nsubseteq RE$

Satz 6.2.6 $REC \nsubseteq RE$

Beweis: <u>1.Fall:</u> $A = \emptyset \to A \in RE$ nach def
<u>2.Fall:</u> $A \neq \emptyset, A \in REC$
$\to$ es gibt eiine TM M_1, die c_A berechnet wir konstruieren TM M_2, die χ_A berechnet
$\to$ M_1 wird modifiziert: immer, wenn M_1 0 ausgibt,soll M_2 in eine Endlosschleife gehen
$\to$ Aus den Sätzen 6.2.5 ($REC \subseteq RE$) 6.2.5 ($RE = \mathscr{L}_0$) und 6.1.2 ($CS \nsubseteq REC$) ergibt sich der Beweis für Satz 4.1.2 ($CS \nsubseteq \mathscr{L}_0$)

<u>Zusammenfassung</u>

Folgende Aussagen sind äquivalenz: $\to$ A ist rekursiv aufzählbar
$\to$ A ist semi-entscheidbar
$\to$ A ist vom Typ-0
$\to$ $A = \mathscr{L}(M)$ für eine TM M
$\to$ χ_A ist Turingberechenbar

$\to$ A ist Definitionsbereich einer berechnbaren Funktionen
$\to$ A ist Wertebereich einer berechnabren Funktion

Satz 6.2.7:

RE ist abgeschlossen bezüglich: - Durschnitt
- Vereinigung
- Produkt

$\to$ ist RE abgeschlossen bezüglich: Komplement?

6.3 Beziehung zwischen REC und RE

Satz 6.3.1

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

"$\Leftarrow$:" Ist A entscheidbar, so ist auch $\overline{A}$ entscheidbar
$\to$ 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 $w \in \Sigma^*$
$\to$ Simuliere M_A und $M_{\overline{A}}$ schrittweise parallel auf Eingabe w
$\to$ falls M_A 1 ausgibt, dann gebe 1 aus
$\to$ falls $M_{\overline{A}}$ 1 ausgibt, dann geben 0 aus
$\to$ diese Maschine hält immer an und liefert damit ein Entscheidungsverfahren für A

Satz 6.3.2

RE ist nicht abgeschlossen bezüglich des Komplement
$\to$ <u>Annahme:</u> RE abgeschlossen bezüglich Komplement
$\to$ dann folgt aus Satz 6.3.1: $RE = REC \Rightarrow$ Widerspruch

6.4 Kodierung von Turingmaschinen über{0,1}

$M = (\Sigma, \Gamma, Z, \delta, Z_0, Z_E, \square)$, $\Sigma = \{0,1\}$, $\Gamma = a_0, \ldots, a_k$, $Z = z_0, \ldots, z_n$
$\to$ die Nummern der Symbole $0, 1, \square$ seien festgelegt, z.B.: $a_0 = \square, a_1 = 1, a_2 = 0$
$\to$ Sei $\# \notin \Gamma \cup Z$
$\to$ Codewort von M über $\{0,1,\#\} \to c(M) = \#bin(k)\#bin(n)\#code(\delta)\#\#code(F)\#$
$\to$ für jedes $(i,j) \in \{0,1,\ldots,n\} \times \{0,1,\ldots,k\}$ mit $\delta(z_i, a_j) = (z_l, a_m, X)$ sei:

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

$\rightarrow$ 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

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

$\rightarrow$ $c(M)$ Codewort über $\{0, 1, \#\}$

$\rightarrow$ $c(M)$ können wir $code(M) =< M >$ zuordnen: $\begin{matrix} 0 \rightarrow 00 \\ 1 \rightarrow 01 \\ \# \rightarrow 11 \end{matrix}$

Bemerkung: - $< M >$ Kodierung der TM M

- aus $< M >$ lässt sich M rekonstuieren (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. $A_{DFA} = \left\{ < B, w > \middle| B\ ist\ DFA\ und\ B\ akzeptiert\ w \right\}$

2. $E_{DFA} = \left\{ < A > \middle| A\ ist\ DFA\ und\ \mathscr{L}(A)\ \neq \emptyset \right\}$

3. $S_{REG} = \left\{ < G_1, G_2 > \middle| G_1, G_2\ reguläre\ Grammatik\ und\ \mathscr{L}(G_1) \cap \mathscr{L}(G_2) = \emptyset \right\}$

4. $EQ_{DFA} = \left\{ < A, B > \middle| A\ und\ B\ sind\ DFA's\ und\ \mathscr{L}(A) = \mathscr{L}(B) \right\}$

5. $WORT_2 = \left\{ < G, w > \middle| G\ ist\ k.f.G.\ und\ G\ erzeugt\ w \right\}$

6. $E_{CF} = \left\{ < G > \middle| G\ ist\ k.f.G.\ und\ \mathscr{L}(G) = \emptyset \right\}$

$$\Downarrow$$

7. Wir konstuieren eine TM, die die Arbeit des DFA's nachvollzieht und entsprechend antwortet TM M:
   $\rightarrow$ Eingabe $< B, w >$
   $\rightarrow$ Simuliere DFA B auf Eingabe w
   $\rightarrow$ falls B akzeptiert gebe 1 aus, sonst 0

8. Markierungsalgorithmus für die Zustände
   $\rightarrow$ Startzustand markieren
   $\rightarrow$ alle Zustände markieren, die aus markierten Zustand erreichbar sind
   $\rightarrow$ Wiederholen, bis keine neuen Zustände markiert werden
   $\rightarrow$ Test, ob Endzustände markiert sind

9. DFA's A_1, A_2 konstuieren mit $\mathscr{L}(A_i) = \mathscr{L}(G_i)$
   $\rightarrow$ konstruiere Produktautomaten A aus A_1 und A_2
   $\rightarrow$ Teste, ob $\mathscr{L}(A) = \emptyset$ (mit 2)

10. ÜA

11. CYK

12. $E_{CF} = \left\{ < G > \middle| G\ ist\ kontextfreie\ Grammatik\ und\ \mathscr{L}(G) = \emptyset \right\}$
    $\rightarrow$ es kann eine TM konstuiert werden, die folgenden ausführt
    $\rightarrow$ Markierungsalgorithmus: es werden alle Nichtterminale markiert aus denen ein Wort
    abgeleitet werden kann
    $\rightarrow$ 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 Endlosschelfein untersuchen kann, bevor sie ausgeführt werden

- Goldbasche Vermutung:
  - Jede gerade Zahl größer als 2 lässt sich als Summe zweier Primzahl darstellen
    $n := 4$
    Wiederhole
    teste, ob n als Summe zweier Primzahlen darstellen
    ja $\rightarrow n := n + 2$
    nein $\rightarrow$ 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	$\rightarrow$	4
	$\downarrow$	$\nearrow$		$\swarrow$	
2	2		5		
		$\swarrow$			
3	6				

- Fermatsche Primzahlen
  $\rightarrow$ Primzahlen der Form $2^k + 1$
  $\rightarrow 2^{2^m} + 1 \quad \setminus \quad 3, 5, 17, 257, 65537$

- Hält ein gegebener Algorithmus A bei gegebener Eingabe?
  $\rightarrow$ Zunächst intuitive Argumentation, dass das Halteproblem nicht entscheidbar ist
  $\rightarrow A_0, A_1, A_2, \ldots$ Auflistung aller Algorithmen, die als Eingabe eine natürliche Zahl haben

Halteproblemtabelle:

$\frac{Eingabe}{Ausgabe}$	0	1	2	3	4	5	6
A_0	J	N	J	J	N		
A_1	N	N	N	J	J	$\ldots$	
A_2	N	J	J	J			
A_3	$\vdots$						
$\vdots$							

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

<u>Annahme</u> 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 $_H$ 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 verstehn wir die Menge:
$\rightarrow A_{TM} = \left\{ < M, w > \middle| M \; ist \; Turingmaschine \; und \; M \; akzeptiert \; w \right\}$

Satz 6.5.2.2 $A_{TM} \notin REC$

Beweis: <u>Annahme:</u> Es gibt eine TM H, die A_{TM} entscheidet
$\rightarrow$d.h.: $H(< M, w >) = \begin{cases} 1 & falls \; M \; w \; akzeptiert \\ 0 & falls \; M \; w \; nicht \; akzeptiert \end{cases}$

$\rightarrow$ Dann gibt es auch folgende TM D, die H als Unterprogramm nutzt
$\rightarrow$ D: Eingabe $< H >$
1. Simuliere H auf Input$< M, < M >>$
2. if $H < M, < M >>= 1$ (d.h. M akzeptiert $< M >$)
gehe in Endlosschleife
else gebe 1 aus
$\rightarrow D(< M >) = \begin{cases} 1 & falls \; H(< M, < M >>) = 0 \\ n.d. \; Endlosschleife & falls \; H(< M, < M >>) = 1 \end{cases}$
$\rightarrow$ Die TM D wird nun auf ihren eigenen Code angewendet

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

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

Satz 6.5.2.3 $REC \nsubseteq RE$

Beweis: $A_{TM} \in RE$
$\rightarrow$ konstuiere TM, die $\chi_{A_{TM}}$ berechnet
Eingabe: $< M >, w$
simuliere M auf w
falls M akzeptiert wird, gebe 1 aus
$\rightarrow A_{TM} \notin REC$ folgt $A_{TM} \in \backslash REC$

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: $A_{TM} \in RE$
$\rightarrow$ 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

$\rightarrow$ bisher ging es um Machbarkeit, jetzt soll es schnell gehen
$\rightarrow$ 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)
  - $EC = \left\{ <G> \middle| G \; hat \; einen \; Eulerkreis \right\}$
  - $\rightarrow$ "gehört G zur Menge EC?"
- Hamiltonkreis:
  - geg: Graph $G = (V, E)$
  - Frage: Besitzt G einen Hamiltonkreis?
  - (Permutation π der Knotenindizees $(v_{\pi(1)}, \ldots, v_{\pi(n)})$, sodass $\forall i \in \{1, \ldots, n-1\}$
    - $\left\{v_{\pi(i)}, v_{\pi(i+1)} \in E\right\}$ und $\left\{\{v_{\pi(n)}, \{v_{\pi(1)}\right\} \in E \ldots$ 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 $\{u, v\} \in E)$
- $CLIQUE = \{ < G, K > | G\ ist\ Graph\ und\ besitzt\ eine\ Clique\ der\ Größe\ mindest$

- Independent Set:
  - geg: Graph $G = (V, E)$
  - Frage: Besitzt G eine Independent Set der Größe mindestens k?
  - $(V' \subseteq V\ mit\ |V'| \geq k\ und\ für\ alle\ u, v \in V'\ mit\ u \neq v\ gilt \{u, v\} \notin E)$
  - $INDEPENDENTSET = \left\{ < G, k > \middle| G\ ist\ Graph\ und\ besitzt\ einen\ Independe\right.$
- Vertex-Cover:
  - $VERTEXCOVER = \left\{ < G, k > \middle| G\ ist\ Graph\ und\ besitzt\ ein\ Vertex\ Cover\ der\right.$
  - $(V' \subseteq V)$ mit $|V'| \leq k$, sodass für alle $\{u, v\} \in E$ gilt: $u \in V'$ oder $v \in V'$
- Traveleing-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

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

Sei $t := \mathbb{N} \rightarrow \mathbb{R}$ eine Funktion:
$\rightarrow$ Die Komplexitätsklasse $DTIME < t(n) >$ ist
$\left\{ A \subseteq \Sigma^* \middle| \bigvee_{DTM\ M} (M\ berechnet\ c_A\ und\ \bigwedge_n time_M(n) \leq t(n)) \right\}$

Definition 7.1.2

Eine Pol die Menge der Polynome der Form
$p(n) = a_k n^k + a_{n-1} n^{k-1} + \ldots + a_2 n^2 + a_1 n + a_0$ mit $n, a \in \mathbb{N}$ für alle $0 \leq i \leq k$
$\rightarrow$ Die Komplexitätsklasse P (deterministische Polynomzeit) ist definiert als
$P = U_{p \in Pol} \ DTIME(p(n))$
$\rightarrow$ P ist die Menge aller Probleme, die sich in Polynomzeit lösen (entscheiden) lassen
$\rightarrow$ P ist die Klasse aller effizient lösbaren Probleme
$\rightarrow$ 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
$\rightarrow t(m) \leq m! \leq (\frac{m}{2})^{\frac{m}{2}} \notin Pol$
$\rightarrow$ $ABER$: G hat einen Eulerkreis gdw. G zusammenhängend ist und jeder Knoten hat gerde Valenz
$\rightarrow EG = \left\{ < G > \middle| G \; ist \; eine \; Grammatik \; mit \; Eulerkreis \right\} \in P$
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 $\mathscr{L}(N) \subseteq \Sigma^*$

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

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:
$NP = \bigcup_{p \in Pol} NTIME(p(n))$
Frage: Wie sehen nichtdeterministische Algorithmen für unsere Eingangs genannten Probleme aus?
$\rightarrow$ das Raten einer Lösung und ihr anschließendes Verifizieren ist ein nichtdeterministischer Algorithmus!

Beispiel:
Hamiltonkreis:
- Raten einer Knotenpermutation
- Testen, ob es eine Clique ist
$\Rightarrow$ das ist in Polynomialzeit möglich
Clique:
- Raten einer Menge von k Knoten
- Testen, ob es eine Clique ist
$\Rightarrow$ Indset, Vertex-Cover
$\rightarrow$ all diese Probleme gehören zu der Klasse NP

- noch niemand hat einen P. Algorithmus für diese Probleme gefunden
  $\rightarrow$ NP enthält sehr viele interessante und schwer lösbare praktische Probleme
  $\rightarrow P \subseteq NP$
- $P \neq NP$? Eine der größten offene 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 $x\Sigma^*$ gilt: $x \in A \Leftrightarrow f(x) \in B$
$\rightarrow A \leq_p B$

Definition 7.3.2

- Eine Sprache A heißt $NP\text{-}schwer$, falls für alle Sprachen $L \in NP$ gilt: $L \leq_p A$
- Eine Sprache A heißt NP-vollständig, falls A $NP\text{-}schwer$ 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 konstuieren 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 nichtdeterministisch 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 an 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 $\mathscr{L} \in NP$ 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 $I_\beta(F) = 1$
$SAT = \{< F > | F\ ist\ erfüllbare\ Formel\ der\ Aussagenlogik\}$
$\big((A \to B) \land (B \lor C)\big) \leftrightarrow \big((A \land B) \leftrightarrow C\big) \to A$

Satz 7.3.7

SAT ist NP vollständig
$\to$ ohne Beweis
$SAT \in NP$ Belegung raten und $I_\beta(F)$ bestimmen
$\to$ zu zeigen: für alle $A \in NP$ gilt $A \leq_p SAT$
Idee: Arbeitsweise einer nichtdeterministischer TM durch eine Formel beschreiben
$3 - SAT = \{< F > | F\ ist\ erfüllbare\ Formel\ in\ konjunktiver$
$\ Normalform\ mit\ höchstens\ 3\ liberalen\ pro\ Klausel\}$
Man kann zeigen: $SAT \leq_p 3SAT$

Satz 7.3.7 **Cook, Levin**

SAT ist NP vollständig
$\to$ ohne Beweis
$SAT \in NP$: Belegung raten und $I_\beta(F)$ bestimmen
$\to$ zu zeigen: für alle $A \in NP$ gilt $A \leq_p SAT$
Idee: Arbeitsweise einer nichtdeterministischen TM durch eine Formel beschreiben
$3 - SAT = \{< F > | F\ erfüllbare\ Formel\ in\ konjunktiver\ Normalform\ mit\ höchstens\ 3\ l$
Man kann zeigen: $SAT \leq_p 3SAT$

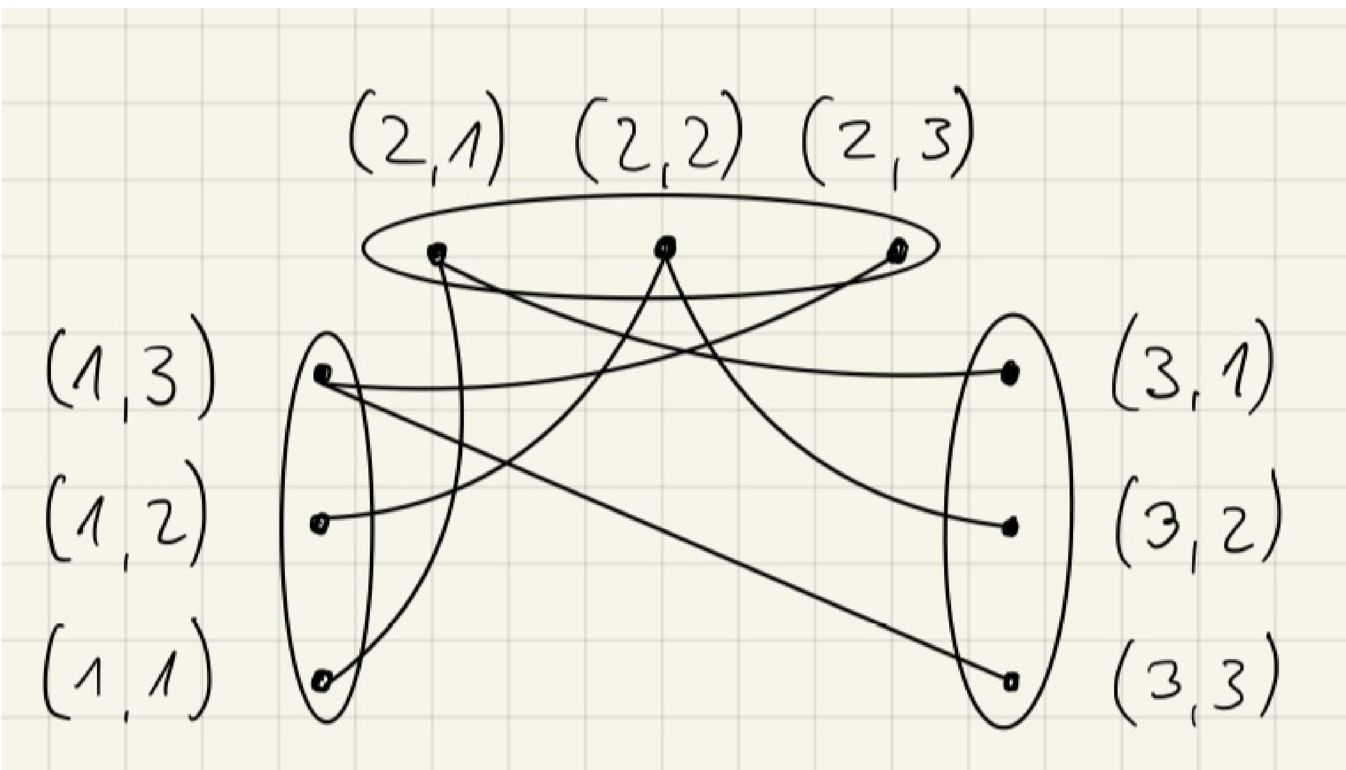
Satz 3.7.8

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 (belibiegen) Inpit für $3SAT$ in einen Input

$INDESET$ überführt
$\rightarrow$ 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

$V := \{(1,1),(1,2),\dots,(m,2),\dots(m,3)\}$
$E := \{(i,j),(p,q)|i=p \; oder \; z_{ij} = \neg z_{pq}\}$
$K := m$
$x_1 = 1, x_2 = 0, x_3 = 1$ ist erfüllende Belegung
$\rightarrow (1,1)(2,2)(3,3)$ sind paarweise nicht verbunden

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

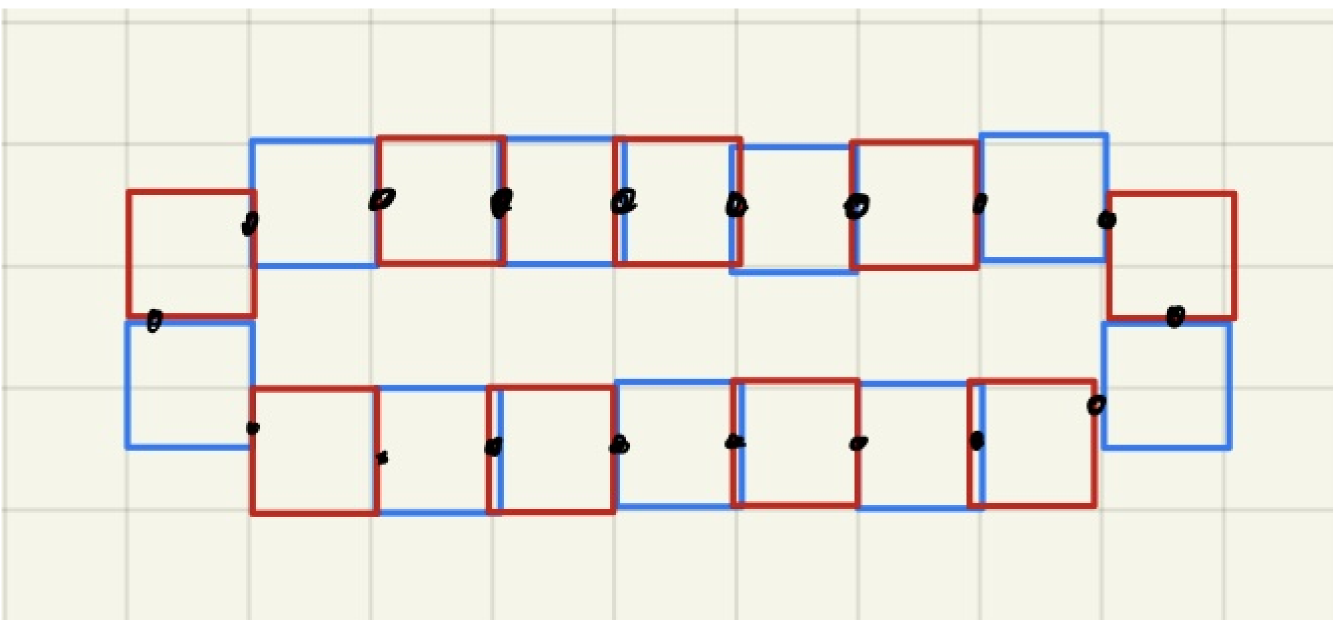


$< F > \; \in SAT$
$\Leftrightarrow$ F ist erfüllbar durch Belegung β
$\Leftrightarrow$ es gibt in jeder Klausel ein Literal, dass unter β den Wert 1 erhält
z.B. $z_{1_{j_1}}, z_{2_{j_2}}, z_{3_{j_3}}, \dots, z_{m_{jm}}$
$\Leftrightarrow$ es gibt Literale $z_{1_{j_1}}, z_{2_{j_2}}, z_{3_{j_3}}, \dots, z_{m_{jm}}$ 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 $k \Leftrightarrow < G, k > \in INDSET$
$F \rightarrow (G,k)$ ist in Polnomialzeit 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: $3SAT \leq_p BOXCOVER$
... konstuiere 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

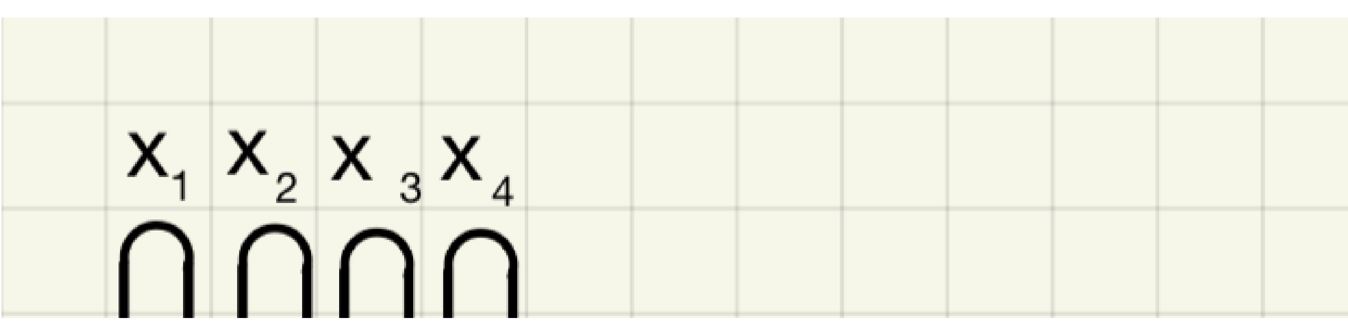
→ eine Überdeckung entspricht TRUE, die andere FALSE

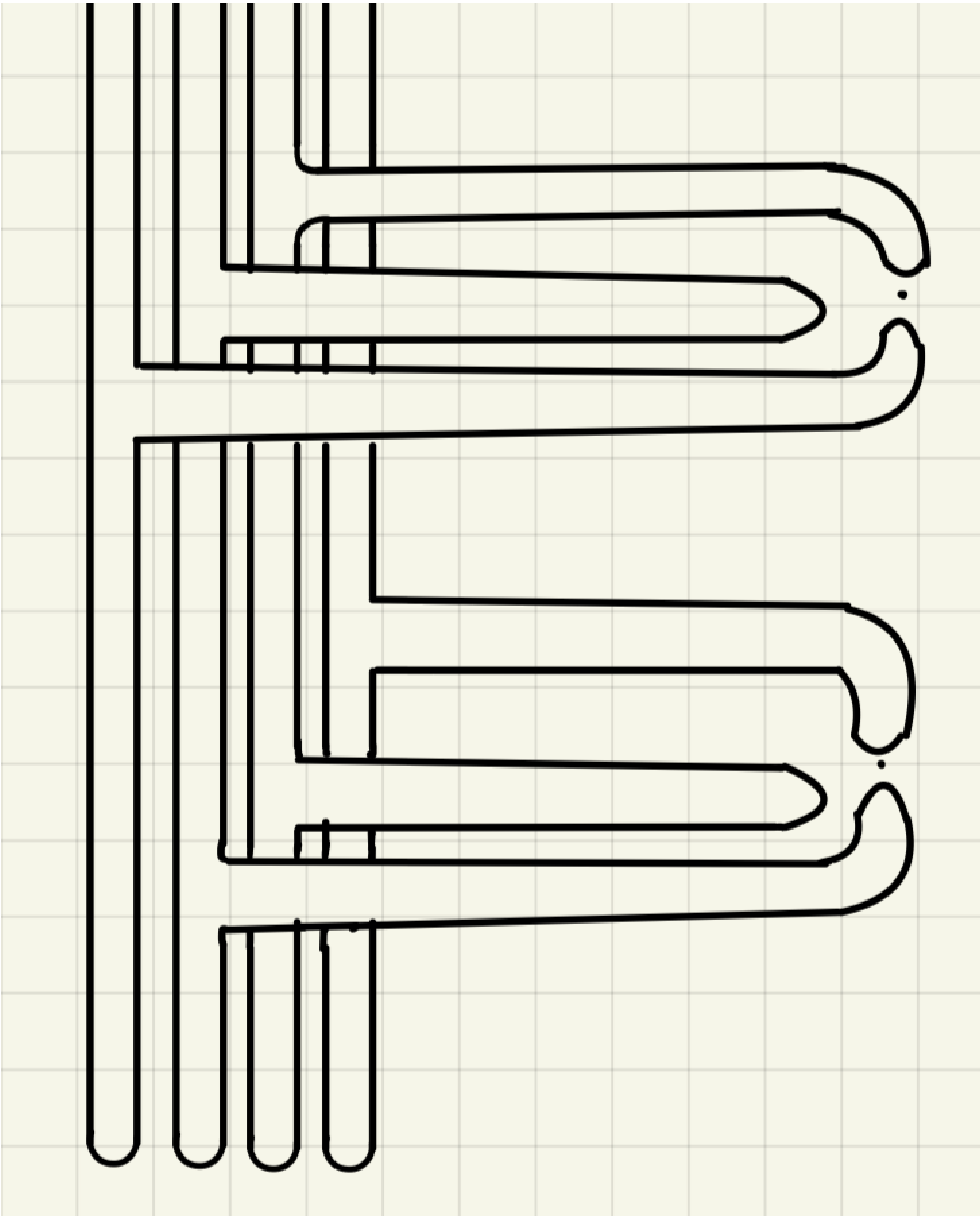


#

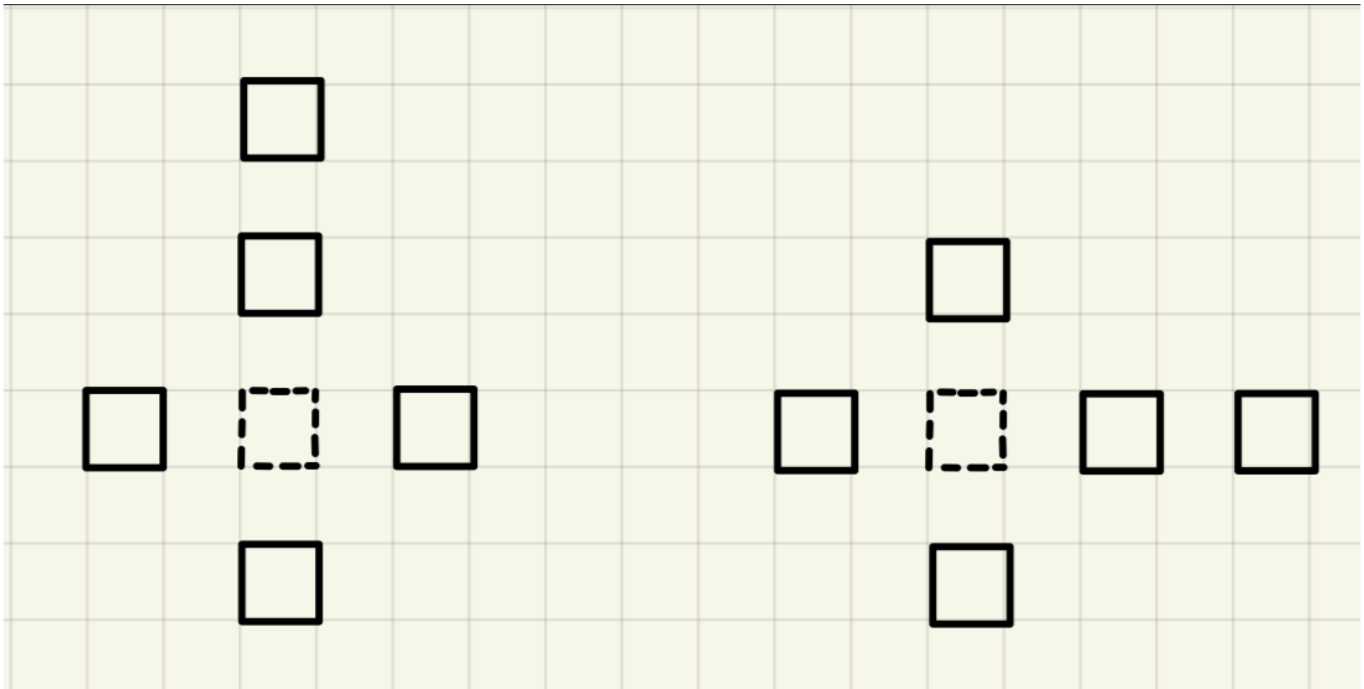
blaue und rote Quadrate

(2) Gesamte Konstruktion





(3) Kreuzungskomponente

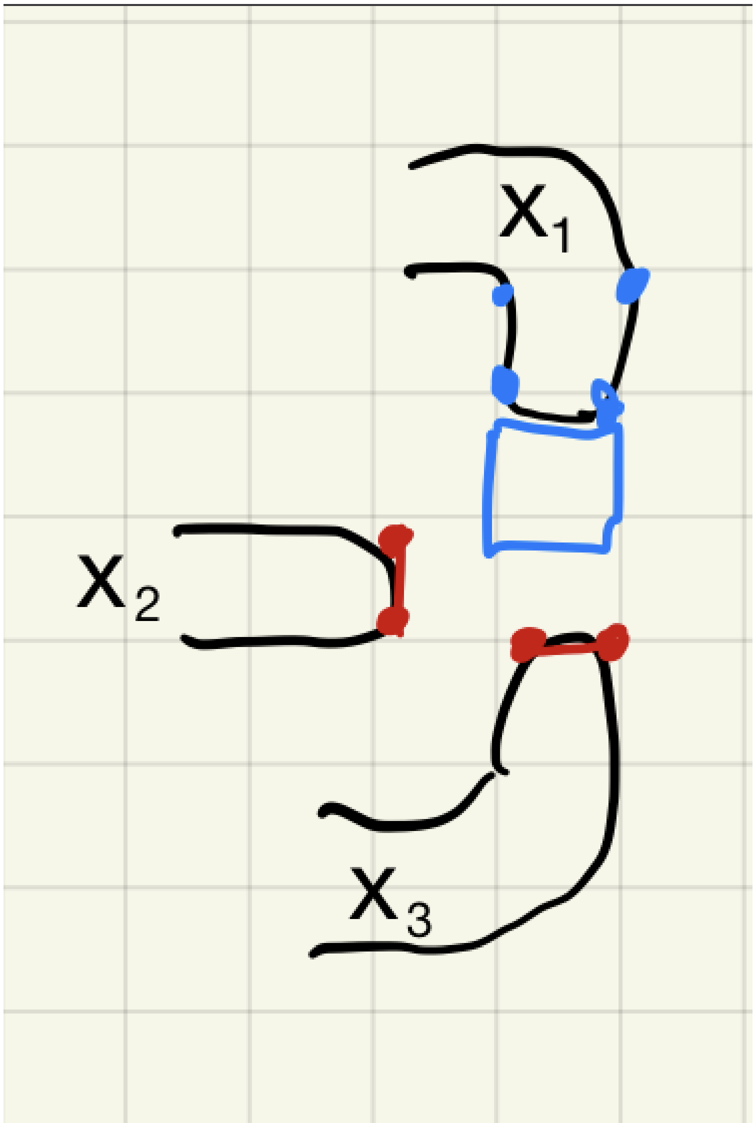


- Pro Kreuzungspunkt kann eine Box gespar werden
- Schleifen beeinflussen sich nicht
- N_c Anzahl der Kreuzungspunkte $k := \sum_{i=1}^{n} k_i - N_c$

(4) Klauselkomponente

$x_1 \vee \neg\neg x_2 \vee \neg x_3$

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 Punktkonstruktion 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.