Grundlagen der theoretischen Informatik

Dr. Harald Hempel

SS 2006

Inhaltsverzeichnis

Auflistung der Theoreme

Sätze

Definitionen und Festlegungen

Vorwort

Dieses Dokument wurde als Skript für die auf der Titelseite genannte Vorlesung erstellt und wird jetzt im Rahmen des Projekts „Vorlesungsskripte der Fakultät für Mathematik und Informatik“ weiter betreut. Das Dokument wurde nach bestem Wissen und Gewissen angefertigt. Dennoch garantiert weder der auf der Titelseite genannte Dozent, die Personen, die an dem Dokument mitgewirkt haben, noch die Mitglieder des Projekts für dessen Fehlerfreiheit. Für etwaige Fehler und dessen Folgen wird von keiner der genannten Personen eine Haftung übernommen. Es steht jeder Person frei, dieses Dokument zu lesen, zu verändern oder auf anderen Medien verfügbar zu machen, solange ein Verweis auf die Internetadresse des Projekts `http://uni-skripte.lug-jena.de/` *enthalten ist.*

Diese Ausgabe trägt die Versionsnummer 2606 und ist vom 6. Dezember 2009. Eine (mögliche) aktuellere Ausgabe ist auf der Webseite des Projekts verfügbar.

Jeder ist dazu aufgerufen, Verbesserungen, Erweiterungen und Fehlerkorrekturen für das Skript einzureichen bzw. zu melden oder diese selbst einzupflegen – einfach eine E-Mail an die Mailingliste `<uni-skripte@lug-jena.de>` *senden. Weitere Informationen sind unter der oben genannten Internetadresse verfügbar.*

Hiermit möchten wir allen Personen, die an diesem Skript mitgewirkt haben, vielmals danken:

- *Matti Bickel* `<kabel@ca0t.de>` *(2006)*

Teil I.

Formale Sprachen

1. Sprachen und Grammatiken

1.1. Grundlagen

Definition 1.1.1
Eine Menge Σ heißt **Alphabet** *gdw* sie endlich ist. (und nicht leer $\Sigma \neq \emptyset$)

Beispiel
$\Sigma_1 = \{a, b, c, \ldots, y, z\}$
$\Sigma_2 = \{\text{APFEL}, \text{FOR}, \text{IF}\}$

Elemente von Alphabeten heißen Buchstaben.

Definition 1.1.2
Sei Σ ein Alphabet. Ein **Wort** w über Σ ist eine endliche Folge von Buchstaben aus Σ.

Bemerkung
Wörter sind Aneinanderreihungen von Buchstaben: abba, FORFORFOR, ε (*leeres* Wort)

Definition 1.1.3
Seien w_1, w_2 Wörter über Σ, d. h. $w_1 = x_1 x_2 \ldots x_n$ und $w_2 = y_1 y_2 \ldots y_m$. Die Konkatenation von w_1, w_2 ($w_1 w_2$) ist definiert als $w_1 w_2 = x_1 \ldots x_n y_1 \ldots y_m$.
Die Länge des Wortes w_1, $|w_1|$, ist definiert als die Zahl der Buchstaben von $w_1 = n$.

Bemerkung
ε ist das eindeutig bestimmte Wort mit der Länge 0.

Definition 1.1.4
Sei Σ Alphabet. Der **Kleene-Abschluss** von Σ, Σ^* wird wie folgt definiert:

$$\Sigma^0 = \{\varepsilon\}$$
$$\Sigma^{i+1} = \{uv \colon u \in \Sigma \wedge v \in \Sigma^i\}$$

Bemerkung

$$\Sigma^1 = \{uv \colon u \in \Sigma \wedge v \in \Sigma^0\} = \Sigma$$
$$\Sigma^i = \text{ Menge der } i\text{-buchstabigen Wörter}$$
$$\Sigma^* = \text{ Menge aller Wörter über } \Sigma$$

Definition 1.1.5
Sei Σ Alphabet. A heißt (formale) **Sprache** über Σ *gdw* $A \subseteq \Sigma^*$

Bemerkung
$\emptyset$ Sprache über Σ_1
$\{abba, beatles, tokio\}$ Sprachen über Σ_1.

Notation: a Buchstabe: $a^0 = \varepsilon$, $a^i = \underbrace{aa \dots a}_{i-\text{mal}}$
Am Beispiel: $\{a^i b^i \colon i, j \in \mathbb{N}\}$ Sprache über Σ (unendliche Sprache)

Definition 1.1.6
Seien A und B zwei Sprachen über Σ.

$$A \cdot B = \{uv \colon u \in A \wedge v \in B\} \tag{1.1}$$

Bemerkung
$A \cap B$, $A \cup B$, $\bar{A}$, ...sind wie üblich definiert.
Gilt: $A \times B = A \cdot B$?
$A = \{a, aa\}$, $B = \{ab, b\}$
$A \times B = \{(a, ab), (a, b), (aa, ab), (aa, b)\}$
$A \cdot B = \{aab, ab, aaab\}$

1.2. Grammatiken

Sei A Sprache über Σ, A unendlich. Offenbar gibt es schnelle Algorithmen, die für eine solche Sprache ($A = \text{FOO}++$) für ein gegebenes Wort w entscheiden, ob $w \in A$ oder $w \notin A$.
Algorithmus läuft auf einem (endl.) Computer, wie kann A also repräsentiert werden?

1. Erzeugendensystem
2. Gleichungssysteme
3. wohlgeformte Ausdrücke

Grammatiken benötigen Bausteine:

1. Alphabet
2. Hilfssymbole
3. Anfang für Ersetzungen
4. Ersetzungsregeln

Definition 1.2.1
Eine **Grammatik** ist ein 4-Tupel: $G = (\Sigma, N, S, R)$ mit folgenden Bedingungen:

1. Σ endl. Menge
2. N endl. Menge, $\Sigma \cap N = \emptyset$
3. $S \in N$
4. R endl. Menge, $R \subseteq (N^* \backslash \{\varepsilon\}) \times (\Sigma \cup N)^*$

Bemerkung
Σ - Buchstaben, Terminalsymbole (Konvention: Kleinbuchstaben)
N - Hilfssymbole, Nichtterminalsymbole (Konvention: Großbuchstaben)
S - Startsymbol
R - Regelmenge $(p, q) \in R$, $p \in N^* \backslash \{\varepsilon\}$ ist eine Folge von Nichtterminalsymbolen, $q \in (\Sigma \cup N)^*$ eine Folge von Terminal und Nichtterminalsymbolen

Beispiel

$$G = (\{a, b\}, \{A, B\}, S, \{(S, AB), (A, aA), (B, Bb), (A, a), (B, b)\})$$
$$S \vdash AB \vdash aB \vdash ab$$
$$\mathfrak{L}(G) = \{a^i b^i : i, j \in \mathbb{N} \wedge i, j \geq 0\}$$

$\mathfrak{L}(G)$ enthält die Wörter über Σ, die aus dem Startsymbol durch Anwendung der Regeln abgeleitet werden können.

Bemerkung
$*$ ist Hüllenoperator:

1. $A \subseteq A^*$
2. $A \subseteq B \rightarrow A^* \subseteq B^*$
3. $(A^*)^* = A^*$

Definieren: $A^+ = A^* \backslash \{\varepsilon\}$

Bei Regeln stehen links nur Nichtterminalsymbole, aber mindestens eines. Rechts stehen beliebig lange Wörter über dem Alphabet $(\Sigma \cup N)^*$

Kommentar: allgemeine Regeldefinition: R endlich und $R \subseteq (\Sigma \cup N)^* \times (\Sigma \cup N)^*$. Man kann zeigen: es gibt zu jeder so definierten Regelmenge eine Regelmenge $N^+ \times (\Sigma \cup N)^*)$, die äquivalent ist.

Ist $(p.q) \in R$, so schreiben wir auch $p \rightarrow q$.

Definition 1.2.2
Seien w, w' Wörter über Σ, d. h. $w, w' \in (\Sigma \cup N)^*$ und $G = (\Sigma, N, S, R)$ eine Grammatik. w' heißt aus w ableitbar (in einem Schritt und Regeln aus G) *gdw* folgendes gilt:

$$\exists p_1, p_2, p, q \colon (p_1, p_2, q \in (\Sigma \cup N)^* \wedge p \in N^+ \wedge w = p_1 p p_2 \wedge w' = p_1 q p_2 \wedge (p, q) \in R)$$

Notation: $w \vdash_G w'$

Bemerkung
$\vdash_G$ ist eine binäre Relation über $(\Sigma \cup N)^*$
$\vdash_G^*$ ist reflexiv, transitive Hülle von $\vdash_G$

Definition 1.2.3
Seien w, w', G wie in Def. Definition 1.2.2. w' heißt aus w ableitbar (in bel. vielen Schritten mit Regeln aus G), $w \vdash_G^* w'$, *gdw*

$$\exists n \in \mathbb{N}, w_1, \ldots, w_n \in (\Sigma \cup N)^* \colon (w = w_1 \vdash_G w_2 \wedge w_2 \vdash_G w_3 \wedge \ldots \wedge w_{n-1} \vdash_G w_n = w')$$

Definition 1.2.4
Sei $G = (\Sigma, N, S, R)$ Grammatik. Die von G generierte (erzeugte) Sprache $\mathfrak{L}(G) = \{w \in \Sigma^* \mid S \vdash_G^* w\}$

Zum Beispiel sei $\mathfrak{L}(G) = \{a^i b^i \colon i, j \geq 1\}$. Dann sind $ab, aabb, a^{14}b^{16}, \ldots \in \mathfrak{L}(G)$ *aber* $BA, ba, \varepsilon, \ldots \notin \mathfrak{L}(G)$.

Beispiel
suchen Grammatik, die alle aussagenlogischen Aussagen mit den Variablen a, b, c generiert:

$$G = (\{a, b, c, (,), \wedge, \vee, \rightarrow, \leftrightarrow, \neg\}, R = \{S \rightarrow (A), S \rightarrow A, A \rightarrow B \wedge C, \ldots\}$$

Ableitungsbaum einer Grammatik G:
todo: Baumgrafik Baum ist i. A. unendlich groß; Blätter sind $\mathfrak{L}(G)$

Syntaxbaum des Wortes *aabb* bezüglich G:
todo: Baumgrafik Baum endlich, stellt Ableitung eines Wortes w, $S \vdash^* w$, dar; Reihenfolge der Anwendung der Regeln i. A. nicht mehr ersichtlich

Offenbar ist die von G generierte Sprache $\mathfrak{L}(G)$ eindeutig bestimmt. Umgekehrt kann es zu einer Sprache $A \subseteq \Sigma^*$ mehr als eine erzeugende Grammatik geben.

Beispiel
$\hat{G} = (\{a, b\}, \{F, X, Y\}, F, \{F \rightarrow XY, X \rightarrow aX, Y \rightarrow Yb, X \rightarrow a, Y \rightarrow b\})$
$\mathfrak{L}(G) = \mathfrak{L}(\hat{G}) = \{a^i b^i \colon i, j \geq 1\}$

Definition 1.2.5
Zwei Grammatiken G_1 und G_2 heißen äquivalent, *gdw* $\mathfrak{L}(G_1) = \mathfrak{L}(G_2)$
Notation: $G_1 \sim G_2$

Bemerkung
zu jeder Grammatik für die lediglich $R \subseteq (\Sigma \cup N)^* \times (\Sigma \cup N)^*$ gilt, gibt es eine äq. Grammatik für die $R \subseteq N^+ \times (\Sigma \cup N)^*$ gilt.

1.3. Die Chomsky-Hierarchie

Definition 1.3.1
Sei $G = (\Sigma, N, S, R)$ eine Grammatik gemäß Definition Definition 1.2.1.

1. G heißt auch Phrasenstrukturgrammatik bzw. **Typ-0-Grammatik**.
2. G heißt nicht verkürzend (**kontextsensitiv**) bzw. Typ-1-Grammatik
   gdw $\forall (p, q) \in R$: $|p| \leq |q|$
3. G heißt **kontextfrei** bzw. Typ-2-Grammatik
   gdw G Typ-1-Grammatik und $\forall (p, q) \in R$: $p \in N$
4. G heißt **regulär** bzw. Typ-3-Grammatik
   gdw G Typ-2-Grammatik und $\forall (p, q) \in R$: $q \in \Sigma \vee q \in \Sigma \cdot N$

Beispiel
G ist Typ-0,1,2-Grammatik. Gibt es $G' \sim G$ mit G' regulär?
ja: $G' = (\{a, b\}, \{S, A, B\}, S, \{S \to aA, A \to aA, A \to b, A \to bB, B \to bB, B \to b\})$

Definition 1.3.2
Eine Sprache $A \subseteq \Sigma^*$ heißt:
von einer Grammatik erzeugbar (Typ-0-Sprache)
kontextsensitiv (Typ-1-Sprache),
kontextfrei (Typ-2-Sprache),
regulär (Typ-3-Sprache), *gdw* sie von einer Sprache ihres Typs erzeugt werden kann.

Entsprechend definiert man folgende Teilmengen von $\mathbb{P}(\Sigma^*)$:
$\mathfrak{L}_0$ - Menge der Typ-0-Sprachen
CS- Menge der Typ-1-Sprachen
CF- Menge der Typ-2-Sprachen
REG- Menge der Typ-3-Sprachen

Beispiel: $\{a^i b^i : i, j \geq 1\} \in \text{REG}$

Satz 1.3.3
$\text{REG} \subseteq \text{CF} \subseteq CS \subseteq \mathfrak{L}_0$

Beweis:
folgt unmittelbar aus Definition Definition 1.1.1/Definition 1.1.2. ∎

Bemerkung
später zeigen: $\text{REG} \subsetneq \text{CF} \subsetneq \text{CS} \subsetneq \mathfrak{L}_0$

Problem: $\{\varepsilon\} \notin \text{CS}$, da jede erzeugende Grammatik nicht verkürzend ist ($|S| = 1$, $|\varepsilon| = 0$)
Lösung: Vereinbarung bzgl. ε:

1. $S \to \varepsilon$ ist als verkürzende Regel in Typ-(1,2,3)-Grammatiken erlaubt
2. Ist $S \to \varepsilon$ eine Regel der Grammatik, so kommt S in keiner rechten Seite einer Regel vor ($T \to aS$)

Satz 1.3.4
Zu jeder Grammatik $G = (\Sigma, N, S, R)$ vom Typ $\{1, 2, 3\}$ gibt es eine Grammatik, für die Folgendes gilt:

a) G' enthält $S' \to \varepsilon$ und G' erfüllt Sondervereinbarung: (S' kommt auf keiner rechten Seite vor)

b) $\mathfrak{L}(G') = \mathfrak{L}(G) \cup \{\varepsilon\}$

Beweis:
$G' = (\Sigma, N', S', R')$ def:

$$N' = N \cup S'$$
$$S' \notin \Sigma \cup N$$
$$R' = R \cup \{S' \to \varepsilon, S' \to S\}$$

funktioniert nur mit Typ-1,2-Grammatiken, für Typ-3-Grammatiken muss man anders vorgehen: R' = Menge aller Regeln aus R in denen S auf der linken Seite vorkommt, bei denen wir S durch S' ersetzen $\cup R$
Man sieht leicht $\mathfrak{L}(G') = \mathfrak{L}(G) \cup \{\varepsilon\}$ ∎

2. Reguläre Sprachen

Plan:

- Charakterisierung für REG
- Automatenmodell
- Beispiele: $\{w \in \{a,b\}^* : w \text{ beginnt mit } a\}$
- $\{a^i b^i : i \in [1, \infty)\} \notin \text{REG}$
- alle endlichen Sprachen

2.1. Endliche Automaten (deterministisch)

Grammatik - generiert Sprachen
Automat - akzeptiert bzw. entscheidet eine Sprache

Definition 2.1.1
Ein deterministischer endlicher Automat (**DFA**) M ist ein 5-Tupel $M = (\Sigma, Z, z_0, \delta, Z_E)$ mit folgenden Eigenschaften:

1. Σ ist endliche Menge (Eingabealphabet)
2. Z ist endliche Menge (Zustandsmenge)
3. $z_0 \in Z$
4. $\delta : \Sigma \times Z \mapsto Z$ (total definierte Überführungsfunktion)
5. $Z_E \subseteq Z$ (Endzustandsmenge)

Bemerkung
DFA: engl. definite finite automaton

Veranschaulichung: todo: Grafik mit Turingband und Programm δ
Startsituation:

- Kopf befindet sich auf 1. Symbol der Eingabe
- Zustand z_0

Arbeitsschritt:

- abhängig vom aktuellen zustand und dem aktuell gelesenen Eingabesymbol wird gemäß Überführungsfunktion ein neuer Zustand angenommen
- Lesekopf bewegt sich eine Zelle nach rechts

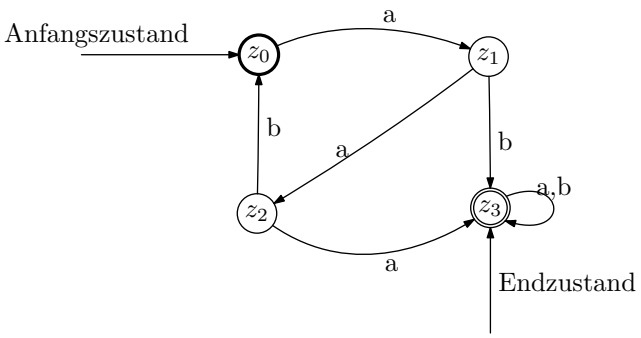
Endsituation:

- alle Eingabesymbole gelesen
- vorliegender Zustand in Z_E

DFAs lassen sich sehr elegant durch sogenannte Überführungsgraphen darstellen.

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

δ	z_0	z_1	z_2	z_3
a	z_1	z_2	z_3	z_3
b	z_0	z_3	z_0	z_3



Abarbeitung: von abba: $z_0 \to^a z_1 \to^b z_3 \to^b z_3 \to^a z_3$
abba wird akzeptiert, baba wird auch akzeptiert, *aber* alle Wörter $b^i : i \in \mathbb{N}$ werden nicht akzeptiert.

Die von einem DFA akzeptierte Sprache ist gerade die Menge aller Wörter, bei denen sich M nach Abarbeitung in einem Zustand aus Z_E befindet.

Definition 2.1.2
Sei $M = (\Sigma, Z, z_0, \delta, Z_E)$ ein DFA.

1. Die erweiterte Überführungsfunktion $\delta^* : \Sigma^* \times Z \to Z$ ist wie folgt definiert: für alle $a \in \Sigma, w \in \Sigma^*, z \in Z$ ist:

$$\delta^*(\varepsilon, z) = z$$
$$\delta^*(aw, z) = \delta^*(w, \delta(a, z))$$

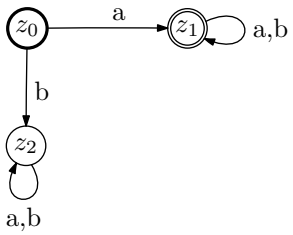
   $\delta^*(aw, z)$ ist der Zustand, der vorliegt, wenn M bei Start im Zustand z das Wort aw abarbeitet

2. Die von M akzeptierte Sprache $L(M)$ ist dann definiert als

$$L(M) = \{w \in \Sigma^* \colon \delta^*(w, z_0) \in Z_E\}$$

.

Beispiel
Baue DFA für $A = \{w \in \{a,b\}^* \colon w\text{beginntmit}a\}$



Satz 2.1.3
Jede Sprache, die von einem DFA akzeptiert wird, ist **regulär**.

Beweis:
Sei $A \subseteq \Sigma^*$ und sei $M = (\Sigma, Z, z_0, \delta, Z_E)$ DFA und $L(M) = A$. Wir geben nun eine reguläre Grammatik $G = (\Sigma, N, S, R)$ an, für die $\mathfrak{L}(G) = A$ gilt.
Setzen: $N = Z$, $S = z_0$.

1. Fall: $\varepsilon \notin A$: $R = \{z \to az' \colon \delta(a,z) = z'\} \cup \{z \to a \colon \delta(a,z) \in Z_E\}$
Offenbar gilt nun für jedes Wort $w = w_1 \dots w_n \in \Sigma^*$ $(w_i \in \Sigma)$:

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

2. Fall: Folgt unmittelbar aus Fall 1 unter Berücksichtigung von Satz Satz 1.3.4. ∎

2.2. Nichtdeterministische endliche Automaten

Unterscheidung (Nicht-)Determinismus:

Determinismus: gegenwärtiger Zustand bestimmt den zukünftigen Zustand eindeutig. Beispiel: übliche Algorithmen, DFA

Nichtdeterminismus: Die aktuelle Situation bestimmt die nachfolgende nicht eindeutig. Es gibt mehrere Folgesituationen. Beispiel: Grammatiken, NFA.

Nichtdeterministische endliche Automaten (NFA)

Idee:

- endlich viele gerichtete Kanten von einem Knoten (Zustand) ausgehend
- endlich viele Startzustände erlaubt

todo: Grafik möglicher NFA

Wort wird akzeptiert *gdw* es gibt einen Startzustand und ausgehend von diesem Startzustand gibt es einen Weg zu einem Endzustand, der das Wort abschließt.

Definition 2.2.1
Ein endlicher nichtdeterministischer Automat (**NFA**) $N = (\Sigma, Z, Z_0, \delta, Z_E)$ ist ein 5-Tupel mit folgenden Eigenschaften:

1. Σ Alphabet (Eingabealphabet)
2. Z endliche Zustandsmenge
3. $Z_0 \subseteq Z$ Startzustandsmenge
4. $\delta\colon \Sigma \times Z \to \mathbb{P}(Z)$ Überführungsfunktion
5. $Z_E \subseteq Z$ Endzustandsmenge

Ganz ähnlich zur Def. Definition 2.1.2 (erweiterte Überführungsfunktion für DFA) definiert man die erweiterte Überführungsfunktion des NFA N wie folgt:

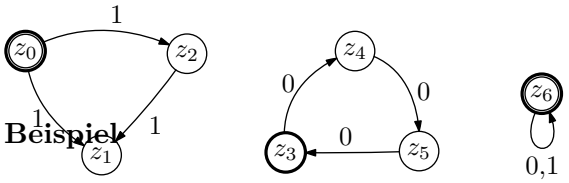
$$\delta^*\colon \Sigma^* \times \mathbb{P}(Z) \to \mathbb{P}(Z)$$

Für alle $\widetilde{Z} \subset Z$, $a \in \Sigma$, $w \in \Sigma^*$ ist δ^* so definiert:

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

Die von einem NFA N akzeptierte Sprache $L(N)$ ist definiert als

$$L(N) = \{w \in \Sigma^*\colon \delta^*(w, Z_0) \cap Z_E \neq \emptyset\}$$



$$\begin{aligned}\delta^*(1, z_0) &= \delta^*(\varepsilon, \delta(1, z_0)) \cup \delta^*(\varepsilon, \delta(1, z_3)) \cup \delta^*(\varepsilon, \delta(1, z_6)) \\ &= \delta^*(\varepsilon, \{z_1\}) \cup \delta^*(\varepsilon, \emptyset) \cup \delta(\varepsilon, \{z_6\}) \\ &= \{z_1\} \cup \emptyset \cup \{z_6\} \\ &= \{z_1, z_6\} \Rightarrow 1 \in L(\widetilde{N})\end{aligned}$$

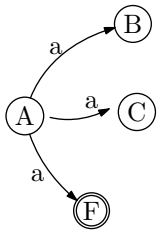
Offenbar ist $L(N) = \{0, 1\}^*$

Bemerkung
Jeder DFA ist auch NFA.

Eine „fast“ Umkehrung von Satz Satz 2.1.3:

Satz 2.2.2
Jede reguläre Sprache kann von einem NFA akzeptiert werden.

Beweis:
Sei $A \in \text{REG}$, $A \subseteq \Sigma^*$. Gemäß Definition Definition 1.3.2 gibt es eine reguläre Grammatik $G = (\Sigma, N, S, R)$ mit $\mathfrak{L}(G) = A$.
Idee: $G\colon A \to aB,\ A \to aC,\ A \to a$ NFA:



Wir betrachten folgenden NFA $N = (\Sigma, Z, Z_0, \delta, Z_E)$:

$$\begin{aligned}Z &=_{df} N \cup \{F\}, \quad F \notin N, \text{ Endzustand} \\ Z_0 &=_{df} \{S\}, \quad Z_E =_{df} \underbrace{\{S, F\}}_{\text{falls } S \to \varepsilon \in R} \vee \underbrace{\{F\}}_{\text{sonst}}\end{aligned}$$

Für $a \in \Sigma$ und $A \in N$ ist: $\delta(a, A) = \begin{cases} \{B \in N\colon A \to aB \in R\} \text{ falls } A \to a \notin R \\ \{B \in N\colon A \to aB \in R\} \cup \{F\} \text{ sonst} \end{cases}$

In allen anderen Fällen ist $\delta(a, A) = \emptyset$, insbesondere $\delta(a, F) = \emptyset$ für alle $a \in \Sigma$.

Offenbar gilt nun für alle $w \in \Sigma^*$, $w = a_1 a_2 \dots a_n,\ a_i \in \Sigma$:

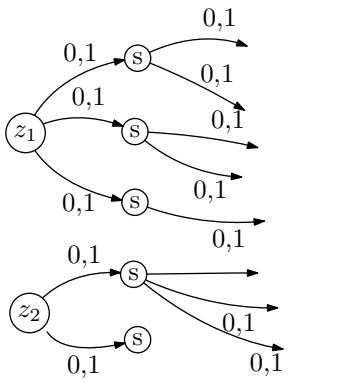
$$
\begin{aligned}
w \in A &\leftrightarrow w \in \mathfrak{L}(G) \\
&\leftrightarrow a_1 a_2 \dots a_n \in \mathfrak{L}(G) \\
&\leftrightarrow \text{ es gibt eine Folge von Nichtterminalen } A_1, A_2, \dots A_{n-1} \in N, \text{ so dass} \\
&\quad S \vdash a_1 A_1 \vdash a_1 a_2 A_2 \vdash \dots \vdash a_1 \dots a_{n-1} A_{n-1} \vdash a_1 \dots a_n \\
&\leftrightarrow \text{ es gibt eine Folge von Zuständen } A_1, \dots a_{n-1} \in Z \text{ mit} \\
&\quad A_1 \in \delta(a_1, S) \wedge \dots \wedge A_{n-1} \in \delta(a_{n-1}, A_{n-2}) \wedge F \in \delta(a_n, A_{n-1}) \\
&\leftrightarrow F \in \delta^*(a_1 \dots a_n, S) \\
&\leftrightarrow w \in L(N) \\
&\Rightarrow A = L(N)
\end{aligned}
$$

■

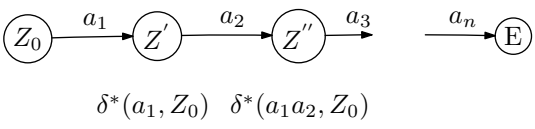
Satz 2.2.3
Jede Sprache, die von einem NFA akzeptiert wird, wird auch von einem DFA akzeptiert.

Beweis:
Sei $A \subset \Sigma^*$ und sei $N = (\Sigma, Z, Z_0, \delta, Z_E)$ mit $L(N) = A$. Wir wollen einen DFA M konstruieren, so dass $L(M) = A$.

Idee: NFA bei Abarbeitung $a_1 a_2 \dots a_n$



DFA: Zustände sind Mengen von Zuständen des NFA:



Wir definieren den DFA $M = (\Sigma, Z', z_0, \delta', Z'_E)$ so:

$$z' = \mathbb{P}(Z)$$
$$z_0 = Z_0$$
$$Z'_E = \{\widetilde{Z} \in \mathbb{P}(z) \colon \widetilde{Z} \cap Z_E \neq \emptyset\}$$

Für alle $a \in \Sigma$, $\widetilde{Z} \in \mathbb{P}(Z)$ ist $\delta'(a, \widetilde{Z}) = \bigcup_{z \in \widetilde{Z}} \delta(a, z) = (\delta^*(a, \widetilde{Z}))$

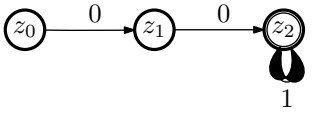
Für jedes Wort $w = a_1 a_2 \dots a_n \in \Sigma^*$ gilt:

$$\begin{aligned}
w \in L(N) &\leftrightarrow \delta^*(w, Z_0) \cap Z_E \neq \emptyset \\
&\leftrightarrow \text{ es gibt eine Folge von Teilmengen } Z_1, Z_2, \dots, Z_n \subseteq Z \text{ und} \\
&\quad\ \delta^*(a_1, Z_0) = Z_1 \wedge \dots \wedge \delta^*(a_n, Z_{n-1}) = Z_n \text{ und } Z_n \cap Z_E \neq \emptyset \\
&\leftrightarrow \text{ es gibt eine Folge von Zuständen } Z_0, Z_1, \dots, Z_n \text{ von } M, \\
&\quad\ \text{mit } \delta'(a_1, Z_0) = Z_1 \wedge \dots \wedge \delta'(a_n, Z_{n-1}) = Z_n \text{ und } Z_n \in Z_E \\
&\leftrightarrow \delta'^*(a_1 \dots a_n, Z_0) \in Z_E \\
&\leftrightarrow w \in L(M)
\end{aligned}$$

■

Beispiel
NFA $N = (\{0,1\}, \{z_0, z_1, z_2\}, \{z_0, z_1, z_2\}, \delta, \{z_2\})$

$$L(N) = \{0^i 1^j \colon i \in \{0,1,2\} \wedge j \in \mathbb{N}\}$$



DFA $M = (\{0,1\}, \mathbb{P}(\{z_0, z_1, z_2\}), \{z_0, z_1, z_2\}, \delta', \{\{z_2\}, \{z_1, z_2\}, \{z_0, z_2\}, \{z_0, z_1, z_2\}\})$.

todo: Hier fehlt der Graph, sowohl naiv als auch vereinfacht

Bemerkung

- nicht immer gibt es im konstruierten DFA nicht erreichbare (streichbare) Zustände, d. h. es gibt Sprachen bei denen NFAs mit n Zuständen auskommen, DFAs aber expotentiell viele (2^n) Zustände benötigen.
- wie klein (in Bezug auf Zustände) kann man DFAs machen ($\rightarrow$ später)

Folgerung 2.2.4
Sei $A \subseteq \Sigma^*$ eine Sprache. Die folgenden Aussagen sind äquivalent:

1. $A \in$ REG (es gibt reguläre Grammatik G und $\mathfrak{L}(G) = A$)
2. A kann von einem DFA akzeptiert werden

3. A kann von einem NFA akzeptiert werden

Beweis:
$2 \to 1$ s. Satz Satz 2.1.3
$1 \to 3$ s. Satz Satz 2.2.2
$3 \to 2$ s. Satz Satz 2.2.3 ■

Bemerkung
hat man eine reguläre Programmiersprache, so würde ein Syntaxprüfer der DFA für diese Sprache sein. Das heißt lineare Laufzeit für Syntaxtest (DFA prüft jedes Symbol einmal)

2.3. Reguläre Ausdrücke und Gleichungssysteme

- reguläre Ausdrücke: Beschreibungsvariante für reguläre Sprachen
- Gleichungssysteme: dienen zur Berechnung der von einem DFA/NFA erzeugten Sprache

Definition 2.3.1
Sei Σ ein Alphabet.

Die Menge der regulären Ausdrücke $RA(\Sigma)$ über Σ wird induktiv so definiert:

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

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

$$\begin{aligned} (((\varepsilon + \emptyset))^* + (a)^*)\,(b)^* \quad &\in RA(\Sigma) \\ ((a + b))^*\, b \quad &\in RA(\Sigma) \end{aligned}$$

Wir ordnen nun den regulären Ausdrücken Sprachen zu. Dabei gehen wir induktiv vor (wie in Definition Definition 2.3.1)

Definition 2.3.2
Sei Σ Alphabet und γ ein regulärer Ausdruck über Σ, d. h. $\gamma \in RA(\Sigma)$. Die mit γ assoziierte Sprache $L(\gamma)$ ist so definiert:

1. $L(\emptyset) = \emptyset$ und $L(\varepsilon) = \{\varepsilon\}$
2. Für jedes $a \in \Sigma$ ist $L(a) = \{a\}$

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

Beispiel
$L\left(((((\varepsilon + \emptyset))^*) + (a)^*\right)(b)^*) \Rightarrow \{a^i b^j : i, j \in \mathbb{N}\}$
$L(\varepsilon) = \{\varepsilon\}$, $L(\emptyset) = \emptyset$

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

Satz 2.3.3
Eine Sprache ist genau dann regulär, wenn es einen regulären Ausdruck γ gibt, so dass $L(\gamma) = A$.

Bemerkung
weiteres Mittel um schnell zeigen zu können, dass eine Sprache regulär ist

Gleichungssysteme

Ziel: Bestimmen der von einem NFA N akzeptierten Sprache $L(N)$.
Gegeben sei ein NFA $N = (\Sigma, Z, Z_0, \delta, Z_E)$ mit $Z = \{z_0, \ldots, z_n\}$
Gesucht ist $L(N)$.

Lösung: bilden lineares Gleichungssystem mit $n + 1$ Variablen $X_0, \ldots, X_n$ und $n + 1$ Gleichungen.
Idee: $L(N) = \{w \in \sigma^* :$ bei Start in einem Zustand aus Z_0 kann ich einen Zustand aus Z_E nach Abarbeitung von w erreichen$\}$.
Wir setzen gedanklich: X_i Menge aller Wörter, die ich bei Start im Zustand z_i akzeptieren kann.

Wir stellen für N folgendes Gleichungssystem auf:

1. Für jedes $z_i \in Z$ ist X_i eine Variable auf einer linken Seite einer Gleichung (X_i steht dort alleine)

2. Ist $z_j \in \delta(a, z_i)$ für $z_i, z_j \in Z$ und $a \in \Sigma$, so ist aX_j ein Summand auf der rechten Seite der Gleichung, deren linke Seite X_i ist.

3. Ist $z_i \in Z_E$, so ist $\emptyset^*$ ein Summand auf der rechten Seite der Gleichung $x_i = \ldots$

Offenbar steht x_i für die Menge aller Wörter, die der NFA akzeptiert, wenn er mit der Abarbeitung in z_i startet.

Streng genommen ist die RHS jeder Gleichung ein regulärer Ausdruck (mit Klammereinsparung) und den zusätzlichen Symbolen $X_0, X_1, \ldots, X_n$

$$L(N) = \bigcup_{z_i \in Z_0} X_i$$

Wie löst man ein derartiges Gleichungssystem?
Hat es stets eine eindeutige Lösung?
Wir werden im Folgenden für $X_i = a_1X_1 + \ldots + a_nX_n$ die Gleichung $X_i = \{a_1\}X_1 \cup \{a_2\}X_2 \cup \ldots \cup \{a_n\}X_n$ schreiben.

Lemma 2.3.4
Sind A und B reguläre Sprachen mit $\varepsilon \notin A$, so gibt es genau eine reguläre Sprache X, die die Gleichung $x = AX \cup B$ erfüllt.

Beweis:

1. Existenz von X: Offenbra ist $X = A^* \cdot B$ eine Lösung von $x = AX \cup B$

$$\begin{aligned} A(A^*B) \cup B &= (A(A_0 \cup A_1 \cup \ldots))B \cup B \\ &= \{\varepsilon\}(B) \cup (A^1 \cup A^2 \cup \ldots)B \\ &= A^0B \cup (A^1 \cup \ldots)B \\ &= (A^0 \cup A^1 \cup \ldots)B \\ &= A^*B \end{aligned}$$

2. Eindeutigkeit der Lösung: z. z. Ist X eine Lösung für $X = AX \cup B$, so gilt $X = A^*B$.
   Sei x eine solche Lösung.
   Dann gilt $AX \cup B \subseteq X \Rightarrow B \subseteq X \rightarrow AB \subseteq AX \subseteq AX \cup B \subseteq X$
   $\Rightarrow \underbrace{A(AB)}_{A^2B} \subseteq AX \subseteq AX \cup B \subseteq X$
   $\vdots$
   $\Rightarrow A^iB \subseteq X \qquad \forall i \in \mathbb{N}$
   $A^*B \subseteq X$

   n. z. z. $X \subseteq A^*B$
   Nachweis indirekt, Annahme: $A^*B \subsetneq X \Rightarrow X \backslash A^*B \neq \emptyset$

   Sei $y \in X$ ein Wort kürzestester Länge in $X \backslash A^*B$.

   $$\Rightarrow y \notin B \quad (y \in B \rightarrow y \in A^*B)$$

   Wegen $X = AX \cup B$ und $y \notin B$ muss $y \in AX$ gelten. Folglich gibt es Wörter u, v mit $u \in A$ und $v \in X$, so dass $y = uv$. Wegen $\varepsilon \notin A$ folgt $|u| \geq 1$.
   Damit gilt $|v| < |y|$.

   Da $v \in X$ und y ein kürzestestes Wort aus $X \backslash A^*B$ war, muss $v \in A^*B$ gelten.
   $\Rightarrow u \in A \wedge v \in A^*B \Rightarrow y = uv \in A^*B$

ist $u \in A$ und $v \in A^*B$, so ist $v \in A^iB$ für ein $i \in \mathbb{N}$ und sonst $uv \in A^{i+1}B$.
Widerspruch zu $y \in X \backslash A^*B$
$\Rightarrow$ Annahme $A^*B \subsetneq X$ war falsch
$\Rightarrow A^*B = X$ wegen $A^*B \subseteq X$ und $A^*B \nsubseteq X$
$\Rightarrow x = AX \cup B$ hat genau eine Lösung, diese ist $X = A^*B$.

Aufgrund der Abschlusseigenschaft von REG sieht man leicht dass $A^*B \in$ REG ist. (s. Abschnitt über Abschlusseigenschaften) ∎

Satz 2.3.5
Ein Gleichungssytem der Form

$$\begin{aligned} x_0 &= A_{00}X_0 \cup A_{01}X_1 \cup \ldots \cup A_{0n}X_n \cup B_0 \\ x_1 &= A_{10}X_0 \cup A_{11}X_1 \cup \ldots \cup A_{1n}X_n \cup B_1 \\ &\vdots \\ x_n &= A_{n0}X_0 \cup A_{n1}X_1 \cup \ldots \cup A_{nn}X_n \cup B_n \end{aligned}$$

mit regulären Sprachen A_{ij}, B_i, $0 \leq i, j \leq n$ hat eine eindeutige Lösung durch reguläre Sprachen $x_0, x_1, \ldots, x_n$, wenn keine der Sprachen A_{ij} das leere Wort enthalten.

Beweis:
Gibt es eine Gleichung mit $A_{ii} = \emptyset$ (d. h. keine Schlinge an z_i im NFA), so kann man x_i in allen Gleichungen durch die rechte Seite der Gleicung $x_i = \ldots$ ersetzen (erhalte damit nur noch n Gleichungen mit n Unbekannten).
Dabei sollte man natürlich nach dem Einsetzen entsprechend zusammenfassen. ($UX \cup VX = (U \cup V)X$)
Sind alle $A_{ii} \neq \emptyset (0 \leq i \leq n)$, so hat die erste Gleichung

$$x_0 = A_{00}X_0 \cup \underbrace{A_{01}X_1 \cup \ldots \cup B}_{B}$$

die Form aus Lemma Lemma 2.3.4: $x_0 = AX_0 \cup B$. Folglich ist $x_0 = A^*B$, d. h.

$$x_0 = A_{00}^*(A_{01}X_1 \cup A_{02}X_2 \cup \ldots \cup B_0)$$

Diesen Ausdruck für x_0 setzt man nun in alle anderen Gleichungen ein und wiederholt das Verfahren mit x_1. ∎

Beispiel
$A, B, C, D, E, F \in$ REG mit $\varepsilon \notin A, B, C, D, E$

$$\begin{aligned} X &= AX \cup BY \cup C \\ Y &= BX \cup EZ \qquad \rightarrow A_{ii} = \emptyset \\ Z &= DX \cup EZ \cup F \end{aligned} \tag{2.1}$$

Setzen Term für Y in die anderen rechten Seiten ein:

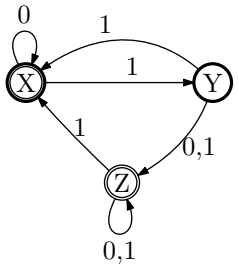
$$\begin{aligned} X &= AX \cup B(BX \cup EZ) \cup C \\ Z &= DX \cup EZ \cup F \\ X &= AX \cup B(BX \cup EZ) \cup C \\ &= (A \cup B^2)X \cup (BE)Z \cup C \end{aligned}$$

Lemma Lemma 2.3.4: $X = (A \cup B^2)X \cup [(BE)Z \cup C]$
Lösung: $X = (A \cup B^2)^* ((BE)Z \cup C)$
Ersetze in Z:

$$\begin{aligned} Z &= D((A \cup B)^*((BE)Z \cup C)) \cup EZ \cup F \\ &= (D(A \cup B^2)^*BE \cup E)Z \cup D(A \cup B^2)^*C \cup F \\ &= (D(A \cup B^2)^*BE \cup E)Z \cup D(A \cup B^2)^*C \cup F \end{aligned}$$

Lemma Lemma 2.3.4: $Z = (D(A \cup B^2)^*BE \cup E)^*(D(A \cup B^2)C \cup F)$
$X = (A \cup B^2)^*[BEZ \cup C]$
$Y =$ ähnlich, allerdings zu langer Ausdruck

Folgender NFA: N



Gleichungssysteme:

$$\begin{aligned} X &= \{0\}X \cup \{1\}Y \cup \emptyset^* \\ Y &= \{1\}X \cup \underbrace{\{0\}Z \cup \{1\}Z}_{\{0,1\}Z} \\ Z &= \{1\}X \cup \{0,1\}Z \cup \emptyset^* \end{aligned}$$

Dies ist ein Spezialfall vom Gleichungssystem Gleichung 2.1 mit

$$\begin{aligned} A = \{0\}, \quad B = \{1\}, \quad C = \emptyset^* \\ E = \{0,1\}, \quad D = \{1\}, \quad F = \emptyset^* \end{aligned}$$

setzt man jetzt entsprechend der eben bestimmten Lösung ein, erhält man eine Beschreibung der Sprachen X, Y, Z und damit auch eine Beschreibung von $L(N) = X \cup Y$.

2.4. Äquivalenzrelationen und Minimalautomaten

Wie kann man zeigen, dass eine Sprache nicht regulär ist?
(s. auch Pumping Lemma)

Definition 2.4.1
Sei $L \subseteq \Sigma^*$ eine Sprache. Die Relation R_L ist eine binäre Relation über Σ^*, die wie folgt definiert ist:

$$\forall x, y \in \Sigma^* \colon x R_L y \Leftrightarrow_{df} \forall z \in \Sigma^* \colon (xz \in L \leftrightarrow yz \in L)$$

Beispiel
$L = \emptyset$: $\quad R_L = \{(x, y) \colon \forall z \in \Sigma^* \colon (xz \in \emptyset \leftrightarrow yz \in \emptyset)\}$
$\varepsilon R_L \varepsilon, \quad \varepsilon R_L 0 : \forall z \colon \underbrace{\varepsilon z \in \emptyset}_{0} \leftrightarrow \underbrace{0z \in \emptyset}_{0}$

$\hat{L} = \{a^n b^n \colon n \in \mathbb{N}\}$
$aR_{\hat{L}}a$, $\not(aR_{\hat{L}}\varepsilon)$(da $ab \in \hat{L} \wedge \varepsilon b \notin \hat{L}$)
$aR_{\hat{L}}a^3b^2, \quad abbaR_{\hat{L}}babba, \ldots$

Beobachtung: Für alle $L \subseteq \Sigma^*$ ist R_L stets eine Äquivalenzrelation.

Uns interessiert nun die Anzahl der Äquivalenzklassen von R_L.

Beispiel
$L = \emptyset$ Äquivalenzklasse von R_L: $[\varepsilon] = [0] = [1] = \ldots = \{\varepsilon, 0, 1, 01, 10, \ldots\}$

$\hat{L} = \{a^n b^n \colon n \in \mathbb{N}\}$ Äquivalenzklassen:

$$\begin{aligned}
[\varepsilon] &= \{\varepsilon\} \\
[a^2 b] &= \{a^2 b, a^3 b^2, \ldots\} = \{a^{i+1} b^i \colon i \in \mathbb{N}\} \\
[b] &= \{b, bb, \ldots, abba, babba, \ldots\} \\
&\vdots \\
[a^k b] &= \{a^{i+k-1} b^i \colon i, k \in \mathbb{N}_+\}
\end{aligned}$$

unendlich viele Äquivalenzklassen!

Eine Äquivalenzrelation kann man statt für Sprachen auch für DFAs definieren:
Sei $M = (\Sigma, Z, z_0, \delta, Z_E)$ ein DFA. Wir definieren die Relation $R_M \subseteq \Sigma^* \times \Sigma^*$ so:

$$\forall x, y \in \Sigma^* \colon x R_M y \leftrightarrow_{df} \delta^*(x, z_0) = \delta^*(y, z_0)$$

Bemerkung: R_M ist auch eine Äquivalenzrelation.

Satz 2.4.2 (Myhill, Nerode)
Sei $L \subseteq \Sigma^*$ eine Sprache. Die folgenden Aussagen sind äquivalent:

1. L ist regulär
2. L ist die Vereinigung von Äquivalenzklassen einern rechtsinvarianten Äquivalenzrelation mit endlichem Index
3. R_L hat einen endlichen Index

Bemerkung

- eine Äquivalenzrelation R heißt **rechtsinvariant** (bzgl. Verkettung) *gdw* für alle $x, y \in \Sigma^*$ gilt:
$$x\,R\,y \leftrightarrow \forall z \in \Sigma^*\colon (xz\,R\,yz)$$
(R_M ist offenbar rechtsinvariant)
- Der Index einer Äquivalenzrelation R ist gerade die Zahl der Äquivalenzklassen von R.

Beweis:
Zeigen: $1 \to 2$, $2 \to 3$, $3 \to 1$

$(1 \to 2)$: Sei $A \subseteq \Sigma^*$ eine reguläre Sprache. Sei $M = (\Sigma, Z, z_0, \delta, Z_E)$ ein DFA mit $L(M) = A$. Wir betrachten R_M. Wir wissen R_M ist eine Äquivalenzrelation, sogar rechtsinvariant und R_M kann nur endlich viele Äquivalenzklassen haben (jedes $z \in Z$ definiert genau eine Äq.klasse über $\{x \in \Sigma^*\colon \delta^*(x, z_0) = z\}$)
Nun ist aber $L = L(M) = \{x \in \Sigma^*\colon \delta^*(x, z_0) \in Z_E\} = \bigcup_{z \in Z_E} \{\underbrace{x \in \Sigma^*\colon \delta^*(x, z_0) = z}_{\text{Äq.klasse über } R_M}\}$

$(2 \to 3)$: Sei R eine rechtinvariante Äquivalenzrelation mit endlichem Index und sei L die Vereinigung von Äquivalenzklassen von R. Wir werden zeigen, dass R eine Verfeinerung von R_L ist. Damit hat R_L höchstens so viele Äquivalenzklassen wie R, also endlich viele.
z. z.:
$$\forall x, y \in \Sigma^*\colon x\,R\,y \to x\,R_L\,y$$

Seien $x, y \in \Sigma^*$:

$$\begin{aligned} x\,R\,y &\to \forall z \in \Sigma^*\colon xz\,R\,yz \quad \text{(R rechtsinvariant)} \\ &\to \forall z \in \Sigma^*\colon xz \in [xz]_R \wedge yz \in [xz]_R \\ &\to \forall z \in \Sigma^*\colon xz \in L \leftrightarrow yz \in L \quad \text{(da L Vereinigung von Äquivalenzklassen von R)} \\ &\to x\,R_L\,y \end{aligned}$$

$(3 \to 1)$: R_L habe endlichen Index. Sei $[x]_{R_L} = \{y \in \Sigma^*\colon x\,R_L\,y\}$ für alle $x \in \Sigma^*$.
Wir geben nun einen DFA für L an: $M = (\Sigma, Z, z_0, \delta, Z_E)$ mit:

$$Z = \{[x]_{R_L} : x \in \Sigma^*\},\ z_0 = [\varepsilon]_{R_L}$$
$$Z_E = \{[x]_{R_L} : x \in L\}$$
$$\delta(a, [x]_{R_L}) = [xa]_{R_L} \text{ für alle } x \in \Sigma^*,\ a \in \Sigma$$

Es gilt nun $L = L(M)$, denn für alle $x \in \Sigma^*$ gilt:

$$\begin{aligned} x \in L &\leftrightarrow [x]_{R_L} \in Z_E \quad \text{(nach Definition von } Z_E \\ &\leftrightarrow \delta^*(x, [\varepsilon]_{R_L} && \leftrightarrow x \in L(M) \end{aligned}$$ ∎

Bemerkung
Der im Beweis von $3 \to 1$ konstruierte Automat heißt Äquivalenzklassenautomat. Er ist bzgl. der Zustandsanzahl ein kleinster DFA für die Sprache.

Beispiel

$$L = \{0^{2^n} : n \in \mathbb{N}\} \in \text{REG}? \quad L = \{0, 00, 0000, \ldots\}$$

Betrachten R_L. Hat R_L unendlich viele Äquivalenzklasen?
Idee: $0^{2^i} \neg R_L 0^{2^j}$ für $i \neq j$
Sei $i \neq j$ o. B. d. A. $i < j$: $\to 0^{2^i} 0^{2^i} = 0^{2^i + 2^i} = 0^{2^{i+1}} \in L$
aber: $0^{2^i} 0^{2^j} = 0^{2^i + 2^j} \notin L$ da $2^i < 2^j$ und $2^{j+1} = 2^j + 2^j$

2.5. Das Pumping-Lemma

Das Pumping-Lemma für reguläre Sprachen ist eine Aussage der Form

$$A \in \text{REG} \to \text{ „großer Ausdruck“}$$

Man kann daher das Pumping-Lemma *nicht* benutzen, um zu zeigen, dass eine Sprache regulär ist.

Das PL kann man lediglich anwenden, um zu zeigen, dass eine Sprache nicht regulär ist, indem man zeigt, dass sie „großen Ausdruck“ nicht erfüllt (Kontraposition).

Satz 2.5.1
Sei $A \subseteq \Sigma^*$ Sprache. Es gilt:

$$\begin{aligned} A \in \text{REG} \to &\exists n \in \mathbb{N}\, \forall x \in A, |x| \geq n\, \exists u, v, w \in \Sigma^* : \\ &(x = uvw \wedge |v| \geq 1 \wedge |uv| \leq n \wedge \forall i \in \mathbb{N} : uv^i w \in A) \end{aligned}$$

$\Rightarrow x = uvw$ Zerlegung von x
$\Rightarrow n$ nennt man Pumping-Zahl

Anwendung: zeigen $\{a^n b^n : n \in \mathbb{N}\} \notin \text{REG}$:

$\forall n \in \mathbb{N}\, \exists x \in A, |x| \geq n\, \forall u, v, w \in \Sigma^* : (x \neq uvw \vee |v| = 0 \vee |uv| > n \vee \exists i \in \mathbb{N} : uv^i w \in L)$

(auch möglich: indirekt, annehmen Sprache sei regulär, Gegenbeispiel finden)
Sei $n \in \mathbb{N}$. Betrachten $a^n b^n$: Seien $u, v, w \in \Sigma^*$ und sei nun $x = uvw \wedge |v| = 0 \wedge |uv| \leq n$. Wegen $x = a^n b^n$ und $|uv| \leq n$ folgt $uv \in \{a\}^*$. Wegen $|v| \geq 1$ folgt $v = a^k$ mit $k \geq 1$. Damit gilt:

$$uv^0 w = uw = a^{n-k} b^n \notin A$$

2.6. Abschlusseigenschaften

Definition 2.6.1
Sei C eine Familie von Sprachen und f eine Funktion, die Sprachen (event. mehrere) auf Sprachen abbildet (z. B. Vereinigung, Durchschnitt, Komplement). Die Familie C heißt abgeschlossen bzgl. f genau dann, wenn für alle $c_1, \ldots, c_n \in C$ gilt:

$$f(c_1, \ldots, c_n) \in C$$

Satz 2.6.2
REG ist abgeschlossen bzgl.

1. Verein
2. Durchschnitt
3. Komplement
4. Produkt
5. Kleeneabschluss $(.^*)$
6. Mengendifferenz
7. Reversal ($L^R = \{w^R : w \in L\}$, $w^R = w_n w_{n-1} \ldots w_1$ mit $w = w_1 \ldots w_n$)

Beweis:
1., 4. 5. über reguläre Ausdrücke (s. Sätze Satz 2.3.3/Definition 2.3.2)
zu 3.: Idee: bulgarisch akzeptieren, nehmen DFA mit $L(M) = A$ und invertieren die Endzustandsmenge, offenbar gilt: $L(M') = \overline{A}$
bis auf 7. zurückführen auf 1., 3. ∎

3. Kontextfreie Sprachen

Dies ist die nächstgrößte Klasse in der Chomsky-Hierarchie. Alle vernünftigen Programmiersprachen sind kontextfrei.

Satz 3.0.3
REG $\subsetneq$ CF

Beweis:
wissen bereits: REG $\subseteq$ CF, offenbar ist $\{a^n b^n : n \in \mathbb{N}\} \in$ CF\REG
Eine kontextfreie Grammatik für $\{a^n b^n : n \in \mathbb{N}\}$ sieht so aus:

$$G = (\{a, b\}, \{S, S'\}, S, \{S \to \varepsilon, S \to ab, S \to aS'b, S' \to ab, S' \to aS'b\})$$

wissen: $\{a^n b^n : n \in \mathbb{N}\} \notin$ REG ∎

3.1. Normalformen

Um leichter lesbare Grammatiken anzugeben, erlauben wir bei kontextfreien Grammatiken Regeln der Form $A \to \varepsilon$ für $A \in N$ und $A \neq S$

Definition 3.1.1
Eine Regel der Form $A \to \varepsilon$ heißt ε-Regel, $A \neq S$.
Eine Regel der Form $A \to B$, $A, B \in N$ heißt Kettenregel.

Eine Grammatik G heißt ε-frei *gdw* sie

1. keine ε-Regeln enthält und
2. nur die eine ε-enthaltende Regel $S \to \varepsilon$ enthält und S nicht auf der rechten Seite einer Regel vorkommt.

ε-Regeln dienen lediglich der Vereinfachung von Grammatiken.

Satz 3.1.2
Zu jeder kontextfreien Grammatik G (mit ε-Regeln) gibt es eine äquivalente ε-freie kontextfreie Grammatik.

Beweis:
Sei $G = (\Sigma, N, S, R)$ eine kontextfreie Grammatik mit ε-Regeln.
o. B. d. A. Sei $\varepsilon \notin \mathfrak{L}(G)$ (sonst hinzufügen nach Satz Satz 1.3.4)
Wir bestimmen $N_\varepsilon = \{A \in N : A \vdash^* \varepsilon \in R\}$ so

1. Ist $(A, \varepsilon) \in R$, so ist $A \in N_\varepsilon$
2. Ist $(A_1, \ldots, A_k) \in R$ und sind $k \geq 1$ und $A_1, \ldots, A_k \in N_\varepsilon$, so ist $A \in N_\varepsilon$

Wir entfernen alle ε-Regeln aus R und fügen jede Regel der Form $B \to uAv,\ A \in N_\varepsilon,\ B \in N,\ u, v \in (\Sigma \cup N)^*$ die Regel $B \to uv$ zu R hinzu.
Die resultierende Grammatik ist ε-frei und äquivalent zu G. ∎

Satz 3.1.3
Zu jeder kontextfreien Grammatik G gibt es eine kontextfreie Grammatik ohne Kettenregeln.

Beweis:
o. B. d. A. sei G eine ε-freie kontextfreie Grammatik. Die Kettenregeln von G induzieren einen gerichteten Graphen auf der Knotenmenge N. Bsp:

1. Eliminieren alle Kreise in diesem Graphen.
   Ein Kreis
   $$A_{i_1} \to A_{i_2}, A_{i_2} \to A_{i_3}, \ldots, A_{i_{n-1}} \to A_{i_n}, A_{i_n} \to A_{i_1}$$
   wird eliminiert, in dem wir alle diese Regeln aus R entfernen und in den verbleibenden Regeln $A_{i_j},\ 1 \leq j \leq n$ durch A_{i_1} ersetzen.
2. Wir benennen die verbleibenden Nichtterminale so um, dass die neuen Namen $\{B_1, \ldots, B_l\}$ sind und aus $B_i \to B_j$ stets $i < j$ folgt. (Regeln $B_i \to B_i$ können sofort entfernt werden)
3. Für $i = l-1, l-2, \ldots$ ersetzen wir die Regeln $B_i \to B_j$ so: sind die Regeln, die B_j auf der linken haben
   $$B_j \to u_1,\ B_j \to u_2,\ B_j \to u_m \quad u \in \{\Sigma \cup N\}^+$$
   so entferne $B_i \to B_j$ und füge $B_i \to u_1 | u_2 | \ldots | u_m$ hinzu. ∎

Definition 3.1.4
Eine kontextfreie Grammatik $G = (\Sigma, N, S, R)$ heißt Grammatik in **Chomksy-Normalform**, *gdw* jede Regel in R eine der folgenden Formen hat:

$A \to BC$	$A, B, C \in N,\ A \neq B,\ A \neq C$
$A \to a$	$A \in N,\ a \in \Sigma$
$S \to \varepsilon$	(in diesem Fall kommt S in keiner rechten Seite vor)

Bemerkung
Der Syntaxbaum von Chomksy-Normalformen ist „fast“ ein Binärbaum: Nachteil: meist viel mehr Regeln notwendig.

Satz 3.1.5
Zu jeder kontextfreien Grammatik $G = (\Sigma, N, S, R)$ gibt es eine äquivalente kontextfreie Grammatik in Chomksy-Normalform (CNF)

Beweis:
Sei G eine Grammatik wie oben. o. B. d. A. enthält R keine Kettenregeln und G ist ε-frei. Wir formen G zu G' um (G' wird in CNF sein)

1. Für jedes $a \in \Sigma$ führen wir ein neues Nichtterminal C_a ein: $\{C_a \colon a \in \Sigma\} \cap N = \emptyset$
2. In jeder Regel von R wird a durch C_a ersetzt
3. Alle Regeln $C_a \to a$ werden zu R hinzugefügt
4. Für jede Regel der Form

$$A \to B_1 B_2 \dots B_m \quad m > 2$$

die jetzt vorhanden ist, führen wir $m-1$ neue Nichtterminalsymbole ein, $D_1, \dots, D_{m-1}$. Die Regel $A \to B_1 \dots B_m$ wird durch $A \to B_1 D_1, D_1 \to B_2 D_2, \dots, D_{m-1} \to B_{m-1} B_m$ ersetzt.
Offenbar ist G' in CNF und äquivalent zu G. ∎

Es gibt eine weitere wichtige Normalform für kontextfreie Grammatiken.

Definition 3.1.6
Eine kontextfreie Grammatik heißt Grammatik in **Greibach-Normalform** *gdw* jede Regel eine der folgenden Formen hat:

$A \to aB_1 \dots B_m$	$A, B_1, \dots, B_m \in N,\ a \in \Sigma$
$A \to a$	$A \in N,\ a \in \Sigma$
$S \to \varepsilon$	und S kommt auf keiner rechten Seite vor

Satz 3.1.7
Zu jeder kontextfreien Grammatik gibt es eine Grammatik in Greibach-Normalform.

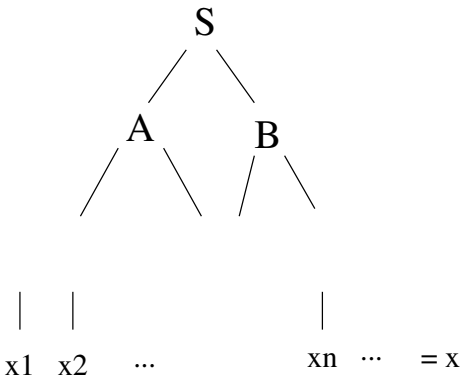
3.2. Das Pumping-Lemma für kontextfreie Sprachen

„unser wahrscheinlich einziges Mittel, um nicht vorhandene Kontextfreiheit von Sprachen nachzuweisen."

Satz 3.2.1
Sei $A \subseteq \Sigma^*$ eine Sprache. Es gilt:

$$\begin{aligned} A \in \mathrm{CF} \to & \exists n \in \mathbb{N}\ \forall x \in A, |x| \geq n \exists u, v_1, \widetilde{v}, v_2, w \in \Sigma^* \colon \\ & (x = u\, v_1\, \widetilde{v}\, v_2\, w \wedge |v_1 v_2| \geq 1 \wedge |v_1 \widetilde{v} v_2| \leq n \wedge \forall i \in \mathbb{N} \colon u\, v_1^i\, \widetilde{v}\, v_2^i\, w \in A) \end{aligned}$$

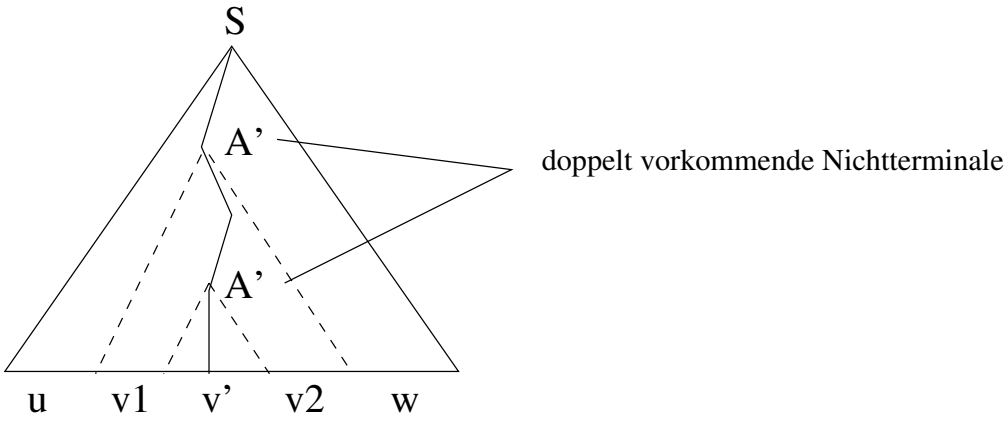
BEWEIS:
Sei $A \in \mathrm{CF}$ und sei $G = (\Sigma, N, S, R)$ eine kontextfreie Grammatik in Chomsky Normalform mit $\mathfrak{L}(G) = A$.
Wir wählen $n = 2^{|N|}$.
Sei $x \in A$ und $|x| \geq n$. Der Syntaxbaum von x bezüglich G sieht so aus:
(Binärbaum (nicht notwendigerweise vollständig) bis auf letzte Ebene)



Dieser Baum hat mindestens $2^{|N|}$ Blätter.

Lemma 3.2.2
In jedem Binärbaum (jeder Knoten hat 2 oder 0 Kinder) mit mindestens 2^k gibt es einen Pfad (von Wurzel bis Blatt) der Länge mindestens k. (Beweis später)

Wir betrachten den Syntaxbaum von x ohne die letzte Ebene. Hier gibt es immer noch $2^{|N|}$ Blätter.
$\Rightarrow$ Es gibt einen Pfad Wurzel zu Blatt mit mindestens $|N|$ Kanten.
$\Rightarrow$ Der Pfad berührt mindestens $|N| + 1$ Knoten im Baum
$\Rightarrow$ Es gibt ein Nichtterminal, welches auf dem Pfad zweimal vorkommt.



$\Rightarrow$ Obere Vorkommen von $\widetilde{A}$ ist höchstens $|N|$ von der Blattebene entfernt. Wir betrachten die Teilgebiete, die aus den beiden $\widetilde{A}$ abgeleitet werden: $u, v_1, \widetilde{v}, v_2, w$ (s. Abb.)
Offenbar ist $uv_1\widetilde{v}v_2w$ eine Zerlegung von x.

Da das obere $\widetilde{A}$ mittels einer Regel $\widetilde{A} \to BC$ umgeformt wird, und nur eines von B oder C auf p liegt und G ε-frei ist, folgt $|v_1v_2| \geq 1$.

Beweis: (Lemma Lemma 3.2.2)
induktiv über k:
IA: $k = 0$ klar.
IS: Aussage gelte für ein k. Betrachte Baum mit mindestens 2^{k+1} Blättern.
Einer der Teilbäume hat $\geq 2^{k+1}/2$ Blätter. Nach IV gibt es in ihm einen Pfad der Länge $k \to$ nimmt man zu diesem Pfad noch die Kante des Teilbaums zur Wurzel des Gesamtbaums hinzu erhält man einen Pfad der Länge $k + 1$. ∎

Anwendung des PL: $L = \{a^nb^nc^n : n \in \mathbb{N}\} \notin$ CF indirekt zeigen.

Satz 3.2.3
Jede kontextfreie Sprache über einem einbuchstabigen Alphabet ist regulär.

Bemerkung
wissen $\hat{L} = \{0^{(n^2)} : n \in \mathbb{N}\} \notin$ REG $\Rightarrow \hat{L} \notin$ CF.

3.3. Abschlusseigenschaften

Satz 3.3.1
Die Sprachklasse CFist abgeschlossen bezüglich:

1. Vereinigung
2. Produkt
3. Kleeneabschluss
4. Differenz und Schnitt mit regulären Mengen

Die Sprachklasse CFist nicht abgeschlossen bezüglich:

1. Durchschnitt
2. Komplement
3. Differenz

Beweis:
Durchschnitt: betrachten die Sprachen

$$L_1 = \{a^i b^i c^j : i, j \in \mathbb{N}\}$$
$$L_2 = \{a^j b^i c^i : i, j \in \mathbb{N}\}$$
$$L_1 \cap L_2 = L = \{a^i b^i c^i : i \in \mathbb{N}\} \notin \mathrm{CF}$$

Komplement: De-Morgan-Regeln
Differenz: Komplement und Schnitt ∎

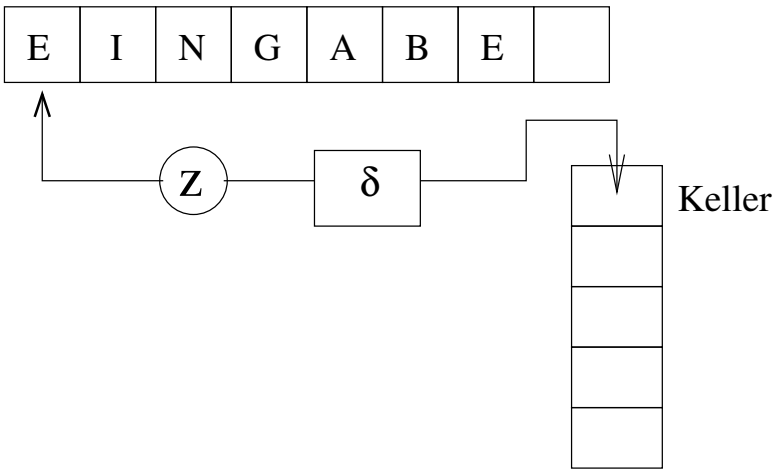
3.4. Kellerautomaten

Ziel: DFA mit Zähler (fifo, Stack)

Definition 3.4.1
Ein nichtdeterministischer **Kellerautomat** M (push-down-automaton, **PDA**) ist ein 6-Tupel $M = (\Sigma, \Gamma, Z, z_0, \delta, Z_E)$ mit folgenden Eigenschaften:

1. Σ ist ein Alphabet (Eingabealphabet)
2. Γ ist ein Alphabet (Kelleralphabet), $\square \in \Gamma$
3. Z ist endliche Menge (Zustandsmenge)
4. $z_0 \in Z$ ist Startzustand
5. $\delta\colon (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Z \to \mathbb{P}_{\text{fin}}(\Gamma^* \times Z)$ (Überführungsfunktion)
6. $Z_E \subseteq Z$

Vorstellung:



Eingabeband: nur lesen, d. h. jedes Symbol kann nur einmal gelesen werden
Keller: lesen und schreiben, Topelement des Stacks wird gelesen und durch ein $w \in \Gamma^*$ ersetzt. Neues Topelement ist oberster Buchstabe von Γ^*. Bei Arbeitsbeginn steht nur ein $\square$ im Keller.

Schreibweise: $\delta(a, A, z) = \{(B_1 \dots B_n, z_1), \dots, (\varepsilon, z_n)\}$

Definition 3.4.2
Sei $M = (\Sigma, \Gamma, Z, z_0, \delta, Z_E)$ ein PDA. Eine Konfiguration K von M ist ein Tripel $K \in \Sigma^* \times \Gamma^* \times Z$. Dabei ist für $K = (u, v, z)$ das Wort u der noch nicht gelesene Teil der Eingabe, das Wort v der aktuelle Kellerinhalt und z der aktuelle Zustand.

Bemerkung
Anfangskonfiguration $(w, \square, z_0)$, $w \in \Sigma^*$ ist Eingabe. Akzeptierende Endkonfiguration (ε, p, z) mit $p \in \Gamma^*$ und $z \in Z_E$.

Wir definieren auf $\Sigma^* \times \Gamma^* \times Z$, der Menge aller Konfigurationen eine binäre Relation wie folgt:

$$\begin{aligned}(u, v, w) \vdash_M (u', v', z') \Leftrightarrow & \exists x \in \Sigma \cup \{\varepsilon\}\ \exists y \in \Gamma \cup \{\varepsilon\}\ \exists y' \in \Gamma^*\ \exists r \in \Gamma^*: \\ & (u = xu' \wedge v = yr \wedge v' = y'r \wedge (y', z') \in \delta(x, y, z))\end{aligned}$$

Ganz analog zu $\vdash_G^*$ für Grammatiken G definiert man $\vdash_M^*$ als reflexive, transitive Hülle von $\vdash_M$.
Die von M akzeptierte Sprache $\mathfrak{L}(M)$ ist definiert als:

$$\mathfrak{L}(M) = \{w \in \Sigma*: \exists z \in Z_E\ \exists p \in \Gamma*: (w, \square, z_0) \vdash_M^* (\varepsilon, p, z)\}$$

Beispiel
PDA für $\{a^n b^n : n \in \mathbb{N}\}$
$M = (\{a, b\}, \{\square, |\}, \{z_0, z_1, z_+, z_-\}, z_0, \delta, \{z_+\})$

δ	z_0	z_1	z_+	z_-
$a, \square$	$(z_1, \|\square)$	$\emptyset$	$\emptyset$	$\emptyset$
$a, \|$	$\emptyset$	$(z_1, \|\|)$	$\emptyset$	$\emptyset$
$b, \square$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$b, \|$	$\emptyset$	(z_-, ε)	$\emptyset$	(z_-, ε)
$\varepsilon, \square$	$(z_+, \square)$	$\emptyset$	$\emptyset$	$(z_+, \square)$
$\varepsilon, \|$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Beispiel
PDA für $\{w\, w^R : w \in \{a, b\}^*\}$
z_l – lesen, z_v – vergleichen

δ	z_v	z_l
$a, \square$	$(a\square, z_l)$	
a, a		$\{(aa, z_l), (\varepsilon, z_v)\}$
a, b		$\{(ab, z_l)\}$
$b, \square$		
b, a		
b, b		

Bemerkung
In der Literatur auch: PDA akzeptiert mit leerem Keller (ist äquivalent zu unserem Akzeptanzbegriff) → selber beweisen und dann verwenden!

Satz 3.4.3
Eine Sprache ist genau dann kontextfrei, wenn sie von einem PDA akzeptiert wird.

3.5. Deterministisch kontextfreie Sprachen

wissen: „DFA = NFA“, der Nichtdeterminismus bei endlichen Automaten bringt nichts Neues.

Bedingungen:

1. $\delta\colon (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Z \to (\Gamma^* \times Z)$
2. $||\delta(a, A, Z)|| + ||\delta(\varepsilon, A, Z)|| \leq 1$

$\mathrm{CF}_{\mathrm{det}}$ ist die Menge aller Sprachen, die von deterministischen PDA akzeptiert werden.
Beobachtung: $\mathrm{REG} \subsetneq \mathrm{CF}_{\mathrm{det}} \subsetneq \mathrm{CF}$

Satz 3.5.1
$\mathrm{CF}_{\mathrm{det}}$ ist abgeschlossen bezüglich:

1. Komplement
2. Schnitt bezüglich regulären Mengen
3. keiner weiteren Operation

Folgerung 3.5.2

$$\mathrm{CF}_{\mathrm{det}} \subsetneq \mathrm{CF}$$

Beweis:
folgt aus den Abschlusseigenschaften ∎

Beweis: (von Satz Satz 3.5.1)
Zuerst *Durchschnitt*: Die im Beweis, dass CF bezüglich Schnitt nicht abgeschlossen ist, angegebenen Sprachen sind sogar in $\mathrm{CF}_{\mathrm{det}}$ und somit $L_1 \cap L_2 \notin \mathrm{CF}_{\mathrm{det}}$.

Dann *Vereinigung*: mit 1. und Durchschnitt

wir beweisen *Komplement*:

Lemma 3.5.3
zu jedem DPDA M' der für jede Eingabe $w \in \Sigma^*$ stets die gesamte Eingabe abarbeitet, also

$$\forall w \in \Sigma^* \; \exists z \in Z \; \exists \alpha \in \Gamma^* \colon (w, \square, z_0) \vdash_M^* (\varepsilon, \alpha, z)$$

Sei $A \subseteq \Sigma^*$ und $A \in \mathrm{CF}_{\mathrm{det}}$. Sei M ein DPDA mit $\mathfrak{L}(M) = A$; o. B. d. A. sei M ein DPDA, der die Eingabe vollständig liest (s. Lemma 3.5.3)
Wir definieren einen DPDA $\widetilde{M} = (\Sigma, \Gamma, Z \times \{+, -, f\}, \widetilde{z_0}, \widetilde{\delta}, Z \times \{f\})$ mit

$$\widetilde{z_0} = \begin{cases} (z_0, +) & \text{falls } z_0 \in Z_E \\ (z_0, -) & \text{falls } z_0 \notin Z_E \end{cases}$$

und ist $\delta(\varepsilon, A, z) = \{(\alpha, z')\}$ für $A \in \Gamma$, $\alpha \in \Gamma^*$, $z, z' \in Z$, so ist

$$\widetilde{\delta}(\varepsilon, A, (z, +)) = \{(\alpha, (z', +))\}$$

und

$$\widetilde{\delta}(\varepsilon, A, (z, -)) = \begin{cases} (\alpha, (z', +)) & \text{falls } z' \in Z_E \\ (\alpha, (z', -)) & \text{falls } z' \notin Z_E \end{cases}$$

Ist $\delta(a, A, z) = \{(\alpha, z')\}$ für $a \in \Sigma$, $A \in \Gamma$, $\alpha \in \Gamma^*$, $z, z' \in Z$, so ist

$$\widetilde{\delta}\left((\varepsilon, A, (z, -))\right) = \{(A, (z, f))\}$$

und

$$\widetilde{\delta}\left((a, A, (z, +))\right) = \begin{cases} \{(\alpha, (z', +))\} & \text{falls } z' \in Z_E \\ \{(\alpha, (z', -))\} & \text{falls } z' \notin Z_E \end{cases}$$

In allen anderen Fällen ist $\widetilde{\delta} = \emptyset$
Sei $w = a_1 a_2 \ldots a_n \in \mathfrak{L}(M) \;\Rightarrow$ es gibt eine Konfigurationenfolge von M

$$(w, \square, z) \vdash_M^* (a, A\beta, \widetilde{z}) \vdash_M (\varepsilon, \alpha\beta, z')$$

mit $a \in \Sigma \cup \{\varepsilon\}$, $A \in \Gamma$, $\alpha, \beta \in \Gamma^*$ und $z' \in Z_E$

Für jede Konfigurationenfolge von $\widetilde{M}$ gilt

1. falls $a = a_n$ bzw. $\delta(a_n, A, \widetilde{z}) = \{(\alpha, z')\}$ so endet die Konfigurationenfolge von $\widetilde{M}$ so:
$$\ldots \vdash_{\widetilde{M}} (\varepsilon, \alpha\beta, (z', +))$$

2. falls $a = \varepsilon$ bzw. $\delta(\varepsilon, A, \widetilde{z}) = \{(\alpha, z')\}$ so endet die Konfigurationenfolge von $\widetilde{M}$ so:

$$\ldots \vdash_{\widetilde{M}} (\varepsilon, \alpha\beta, (z', +))$$

$\Rightarrow w \notin \mathfrak{L}(M)$

Sei nun $w = a_1 \ldots a_n \notin \mathfrak{L}(M)$.
$\Rightarrow$ Für jede (endliche) Konfigurationenfolge von M (gibt es gemäß unserer Vorraussetzung) $(w, \square, z_0) \vdash_M^* (a, A\beta, \widetilde{z}) \vdash_M (\varepsilon, \alpha\beta, z')$ mit $a \in \Sigma \cup \{\varepsilon\}$, $A \in \Gamma$, $\alpha, \beta \in \Gamma^*$, $\widetilde{z} \in Z$ gilt $z' \notin Z_E$
$\Rightarrow$ Es gibt eine Konfigurationenfolge von $\widetilde{M}$

$$\ldots \vdash_{\widetilde{M}}^* (\varepsilon, \alpha\beta, (z', -)) \vdash_{\widetilde{M}} (\varepsilon, \alpha\beta, (z', f))$$

■

3.6. Das Wortproblem für kontextfreie Sprachen

Wortproblem:
gegeben: Sprache $L \subseteq \Sigma^*$ in Form einer Grammatik oder eines Automaten
gesucht: $w \in L$?

Wortproblem für reguläre Sprachen: Reguläre Sprache sei in Form eines DFA gegeben, so haben wir einen Algorithmus mit $O(n)$-Komplexität (n = die Anzahl der Buchstaben in w).

Sprachen werden uns in Form von CNF-Grammatiken gegeben. Sei $G = (\Sigma, N, S, R)$ eine solche Grammatik. Sei $w = a_1 \ldots a_n \in \Sigma^*$ mit $a_i \in \Sigma$ für $i = 1, \ldots, n$.
Frage: $w \in \mathfrak{L}(G)$?
naiver Ansatz: Breitensuche im Ableitungsbaum (expotentielle Laufzeit)
besser als naiv: dynamische Programmierung.
Wir betrachten folgende Mengen:

$$N[i, j] = \{A \in N \colon A \vdash_G^* a_i a_{i+1} \ldots a_{i+j-1}\}$$

mit $1 \leq i \leq n$ und $1 \leq j \leq n - i + 1$.
Offenbar ist $w \in \mathfrak{L}(G) \leftrightarrow S \in N[1, n]$
Außerdem gilt, da G in CNF ist:

$$\begin{aligned} N[i, 1] &= \{A \in N \colon A \vdash_G^* a_i\} = \{A \in N \colon A \vdash_G a_i\} \\ &= \{A \in N \colon (A \to a_i) \in R)\} \end{aligned}$$

$\Rightarrow$ die $N[i,1]$ sind einfach zu bestimmen. Für $j \geq 2$ werden die $N[i,j]$ mit dynamischer Programmierung bestimmt: Sei $j \geq 2$. Offenbar gilt $A \in N[i,j]$, gdw.

$$\begin{gathered} A \to BC \ \in R \text{ mit einem } B, C \in N \text{ und} \\ B \in N[i,l] \text{ und } C \in N[i+l, j-l] \text{ für ein } l \\ (B \vdash_G^* a_i \dots a_{i+l-1}) \qquad (C \vdash_G^* a_{i+l} \dots a_{i+j-1}) \end{gathered}$$

$\Rightarrow$ rekursive Beschreibung des $N[i,j]$:

$$N[i,j] = \bigcup_{l=1}^{j-1} \{A \in N \colon \exists B, C \in N \colon ((A \to BC) \in R \wedge B \in N[i,l] \wedge C \in N[l-1, j-l])\}$$

Das liefert im Wesentlichen den Algorithmus von Cocke, Younger, Kasami - **CYK-Algorithmus**

$N[1,n]$				
$N[1,n-1]$	$N[2,n-1]$			
$\vdots$	$\vdots$	$\ddots$		
$N[1,2]$	$N[2,2]$	$\dots$	$N[n-1,2]$	
$N[1,1]$	$N[2,1]$	$\dots$	$\dots$	$N[n,1]$
a_1	a_2	$\dots$	$\dots$	a_n

Algorithm 1 CYK-Algorithmus

Require: $w = a_1 \dots a_n$
  for $i = 1, \dots, n$ **do**
    $N[i,1] = \{A \in N \colon (A \to a_i) \in R\}$
  end for
  for $j = 2, \dots, n$ **do**
    for $i = 1, \dots, n+1-j$ **do**
      $N[i,j] = \emptyset$
      for $l = 1, \dots, j-1$ **do**
        $N[i,j] = N[i,j] \cup \{A \in N \colon \exists B, C \in N \colon$
          $(A \to BC) \in R \wedge B \in N[i,l] \wedge C \in N[l+i, j-l]\}$
      end for
    end for
  end for
  if $S \in N[1,n]$ **then**
    return $w \in \mathfrak{L}(G)$
  else
    return $w \notin \mathfrak{L}(G)$
  end if

Analyse:
Zeit: $O(n^3)$ (falls G nicht in CNF gegeben + entsprechender Zeit für Umformung)
Platz: $O(n^2)$

Beispiel
Betrachten einfaches Beispiel $w = aabb \in L$ mit

$$L = \{a^n b^n : n \geq 1\} \qquad G = (\{a, b\}, \{S, A, B, C\}, S, R)$$

mit

$$R = \{S \to AB|AC,\ C \to SB,\ A \to a,\ B \to b\}$$

mit $\mathfrak{L}(G) = L$ und G in CNF.

$\{S\}$			
$\emptyset$	$\{C\}$		
$\emptyset$	$\{S\}$	$\emptyset$	
$\{A\}$	$\{A\}$	$\{B\}$	$\{B\}$
a	a	b	b

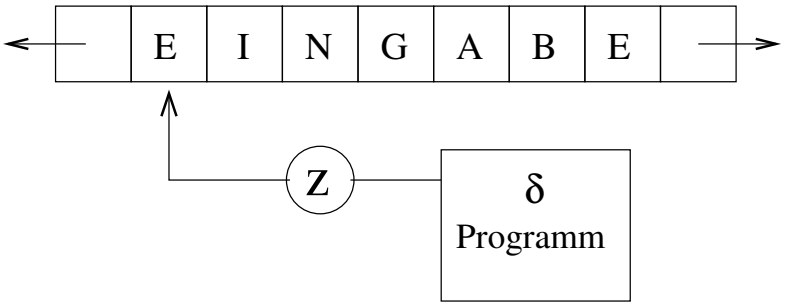
4. Kontexsensitive und $\mathfrak{L}_0$-Sprachen

kontextsensitiv: nicht verkürzende Grammatik (Ausnahme $S \to \varepsilon$)
$\mathfrak{L}_0$-Sprachen: (Normalform-)Grammatiken

4.1. Turingmaschinen

Entwickelt von Alan Turing 1936.

Anschaulich:



Arbeitsweise: abhängig vom gelesenen Symbol und dem aktuellen Zustand wird ein neues Symbol geschrieben, der Zustand verändert und der Kopf bewegt.

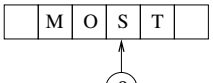
Definition 4.1.1
Ein 7-Tupel $M = (\Sigma, \Gamma, Z, z_0, \delta, Z_E, \square)$ heißt **Turingmaschine** gdw. folgendes gilt:

1. Σ Eingabealphabet
2. Γ Arbeitsalphabet, $\square \in \Gamma$, $\Sigma \subseteq \Gamma$
3. Z ist Zustandsmenge
4. $z_0 \in Z$ Startzustand
5. $\delta\colon \Gamma \times Z \to \mathbb{P}_{\text{fin}}(\Gamma \times Z \times \{L, R, 0\})$
6. $Z_E \subseteq Z$ Endzustandsmenge
7. $\square \in \Gamma$ Leersymbol

Bemerkung
statt $\delta(z,a) = \{(b,\hat{z},L)\}$ schreiben wir $(a,z) \to (b,\hat{z},L)$

Um Berechnungen von TMs beschreiben zu können, benötigen wir den Begriff der Konfigurationen. Dazu benötigen wir folgende Informationen:

- Was steht auf dem Band
- Wie lautet der aktuelle Zustand
- Wo steht der Kopf



Kodierung: $\Rightarrow$ MOβST (ein Wort aus $\Gamma^*\dot{Z}\dot{\Gamma}^*$)
Eine Berechnung einer TM M ist damit eine Folge von Konfigurationen mit Startkonfiguration z_0w.

Definition 4.1.2
Sei M eine TM. Wir definieren eine binäre Relation $\vdash_M$ über $\Gamma^*\dot{Z}\dot{\Gamma}^*$, wobei $K \vdash_M K'$ gerade bedeuten soll, dass K' in einem Takt aus K hervorgeht.
Sei $K = \alpha uzv\beta$ mit $\alpha,\beta \in \Gamma^*$, $u,v \in \Gamma$, $z \in Z$. Dann ist

$$K' = \begin{cases} \alpha u\hat{v}z'\beta & (v,z) \to (\hat{v},z',R) \\ \alpha z'u\hat{v}\beta & (v,z) \to (\hat{v},z',L) \\ \alpha uz'\hat{v}\beta & (v,z) \to (\hat{v},z',0) \end{cases}$$

Wir betrachten die reflexive und transitive Hülle $\vdash_M^*$ von $\vdash_M$

Die von M akzeptierte Sprache $\mathfrak{L}(M)$ ist definiert als:

$$\mathfrak{L}(M) = \{w \in \Sigma^* \colon \exists z \in Z_E \; \exists \alpha,\beta \in \Gamma^* \colon z_0w \vdash_M^* \alpha z\beta\}$$

Beispiel
Eine TM für $L = \{a^nb^nc^n \colon n \geq 1\}$:

$$M = \{\{a,b,c\},\{a,b,c,\$,\square\},\{z_0,\ldots,z_5,z_E\},\delta,z_0,\{z_E\},\square\}$$

	Bedeutung	Absicht
z_0	Startzustand	Starte neuen (a,b,c)-Zyklus
z_1	a gemerkt	b suchen
z_2	a und b gemerkt	c suchen
z_3	a,b und c gefunden u. ersetzt	rechten Rand suchen
z_4	rechten Rand erreicht	Rücklauf und Test, ob alle a,b,c ersetzt
z_5	Test nicht erfolgreich	zum linken Rand laufen, neuen Zyklus starten
z_E	alles gut	Akzeptieren

	z_0	z_1	z_2	z_3	z_4	z_5	z_E
a	$(\$, z_1, R)$	(a, z_1, R)				(a, z_5, L)	
b		$(\$, z_2, R)$	(b, z_2, R)			(b, z_5, L)	
c			$(\$, z_3, R)$	(c, z_3, R)	(c, z_5, L)	(c, z_5, L)	
$\$$	$(\$, z_0, R)$		$(\square, z_4, L)$	$(\$, z_4, L)$	$(\$, z_5, L)$		
$\square$					$(\square, z_E, 0)$	$(\square, z_0, R)$	

Bemerkung
Akzeptanzkriterien:

PDA: 1. akzeptieren mit Endzustand

2. akzeptieren mit leerem Keller

TM: 1. akzeptieren mit Endzustand

2. akzeptieren gdw. Berechnung anhält (keine Folgekonfiguration vorhanden)

4.2. Linear beschränkte Turingmaschinen (LBA)

LBA sind TM, die den Bereich der Eingabe bei jeder Berechnung niemals verlassen. Dafür ist es zweckmäßig, dass linker und rechter Rand der Eingabe markiert sind.
Dazu: Verdoppeln das Eingabealphabet $\Sigma = \Sigma \cup \hat{\Sigma}$ mit $\hat{\Sigma} = \{\hat{a} \colon a \in \Sigma\}$. Statt Eingabe $x_1 \dots x_n$ arbeitet LBA mit Eingabe $\hat{x_1} x_2 \dots x_{n-1} \hat{x_n}$.

Definition 4.2.1
Eine TM $M = (\Sigma, \Gamma, Z, z_0, \delta, Z_E, \square)$ heißt **linear beschränkter Automat (LBA)**, falls für alle Wörter $x = x_1 \dots x_n \in \Sigma^*$ und für alle Konfigurationen $\alpha z \beta$ von M mit

$$z_0 x \vdash_M^* \alpha z \beta \text{ gilt: } |\alpha\beta| = n$$

und kein $\square$ jemals durch ein $u \in \Gamma$ überschrieben wird.

Die von einem LBA M akzeptierte Sprache $\mathfrak{L}(M)$ ist

$$\mathfrak{L}(M) = \{w = a_1 \dots a_n \in \Sigma^* \colon \exists z \in Z_E \ \exists \alpha, \beta \in \Gamma^* \colon z_0 \hat{a_1} a_2 \dots a_{n-1} \hat{a_n} \vdash_M^* \alpha z \beta\}$$

Satz 4.2.2
Eine Sprache ist genau dann kontextsensitiv, wenn sie von einem LBA akzeptiert werden kann.

Satz 4.2.3
Es gilt CF $\subsetneq$ CS

Beweis:
CF $\subseteq$ CS bekannt, aber $\{a^n b^n c^n : n \in \mathbb{N}\} \in \text{CS} \setminus \text{CF}$ ∎

Bemerkung
Können deterministische LBAs genausoviel wie nichtdeterministische LBAs?

Ja, ähnlich wie bei NFA und DFA kann man die Zustandsmenge des deterministischen LBA mit einer Potenzmengenkonstruktion erweitern und die Regelmenge entsprechend anpassen. (in allen Beispielen werden nur Überführungen von $(Z \times \Sigma) \to (\Sigma \times Z \times \{L, R, 0\})$ verwendet, die Definition lässt aber $(Z \times \Sigma) \to \mathfrak{P}(\Sigma \times Z \times \{L, R, 0\})$ zu.

4.3. $\mathfrak{L}_0$-Sprachen

Satz 4.3.1
Eine Sprache gehört genau dann zu $\mathfrak{L}_0$, wenn sie von einer (nichtdeterministischen) TM akzeptiert werden kann.

Satz 4.3.2
CS $\subsetneq \mathfrak{L}_0$, Beweisidee: Diagonalisierung, Beispiele.

Satz 4.3.3
Es gibt Sprachen, die von keiner Grammatik generiert werden können.

Beweis: (mit Zählargument)
Sei Σ unser Alphabet.

1. Wie viele Grammatiken $G = (\Sigma, N, S, R)$ gibt es?
   Grammatiken können als Wörter über einem geeigneten Alphabet Γ notiert werden:

   $$\Gamma = \Sigma \cup \{(,\ ),\ ,\ \triangle, \nabla(\text{für die Nichtterminale}),\ S,\ \to\}$$

   Es gibt nur abzählbar unendlich viele Grammatiken.

2. Wie viele Sprachen gibt es? (Wie groß ist $\mathbb{P}(\Sigma^*)$?)

   $$|\mathbb{P}(\Sigma^*)| = \aleph_1 \qquad \text{(überabzählbar unendlich)}$$

$\Rightarrow$ es gibt mehr Sprachen als Grammatiken, also auch Sprachen, die von keiner Grammatik erzeugt werden. ∎

Teil II.

Berechenbarkeit

5. Churchsche These

5.1. Einführung

Was bedeutet Berechenbarkeit?
Welche Funktionen sind (nicht) berechenbar?
Ideen: Halteproblem, Wahrheitswertberechnungen, chaotische Funktionen (lassen sich nicht mit endlich vielen Zeichen beschreiben)

Wir benötigen ein Berechenbarkeitsmodell:
Ein intuitiver Berechenbarkeitsbegriff hängt vom Menschen ab. Wenn wir Berechenbarkeit formal definieren wollen, müssen wir ein zugrundeliegendes Berechenbarkeitsmodell definieren.
Zum Beispiel: Papier-und-Stift-B., Maple-B., C++-B.

5.2. Grundlagen

$\mathbb{N} = \{0, 1, 2, \ldots\}$
$\mathbb{N}^{\mathbb{N}} = \widetilde{\mathbb{F}_1}$ – Menge aller einstelligen (totalen) zahlentheoretischer Funktionen

$$\widetilde{\mathbb{F}_1} = \mathbb{N}^{\mathbb{N}} \qquad \mathbb{F}_1 = \{f \mid f \colon \mathbb{N} \to \mathbb{N}\}$$
$$\widetilde{\mathbb{F}_2} = \mathbb{N}^{\mathbb{N}\times\mathbb{N}} \qquad \mathbb{F}_2 = \{f \mid f \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}\}$$
$$\widetilde{\mathbb{F}} = \bigcup_{i\in\mathbb{N}} \widetilde{\mathbb{F}_i} \qquad \mathbb{F} = \bigcup_{i\in\mathbb{N}} \mathbb{F}_i$$

$\widetilde{F}$ ist also die Menge aller beliebigstelligen, totalen Funktionen und F die Menge aller beliebigstelligen, partiellen Funktionen.

5.3. Turing-Berechenbarkeit

Definition 5.3.1
Eine Funktion $f \in \mathbb{F}_i$ heißt Turing-berechenbar, genau dann wenn es eine TM $M = (\{0, 1, \ldots, 9, \#\}, \Gamma, Z, z_0, \delta, Z_E, \square)$ gibt, so dass für alle $(n_1, \ldots, n_i) \in \mathbb{N}^i$ gilt:

1. $(n_1, \ldots, n_i) \in D_f \leftrightarrow M$ erreicht bei Eingabe von $n_1\#n_2\#\ldots\#n_i$ einen Zustand aus Z_E

2. $(n_1, \ldots, n_i) \in D_f \leftrightarrow$ es gibt ein $z \in Z_E$ und $x \in \Gamma^*$, so dass $z_0 n_1 \# n_2 \# \ldots \# n_i \vdash_M^* \alpha z m$, wobei $m = f(n_1, \ldots, n_i)$ ($\alpha z m$ liegt vor bei erstmaligem Erreichen eines Zustandes $z \in Z_E$)

Bemerkung
Die TM transformiert das Eingabewort in endlich vielen Schritten so, dass der „Zeiger“ (Lesekopf) der TM am Beginn des Rückgabewertes steht und rechts vom Rückgabewert nur noch Leersymbole stehen (links vom Lesekopf kann alles mögliche stehen)

Beispiel
Eine TM für $f(x) = x + 1$: $M = (\{|\}, \{|, \square\}, \{z_0, z\}, z_0, \delta, \{z\}, \square)$

δ	z_0	z
|	$(\|, z_0, L)$	
$\square$	$(\|, z, 0)$	

Betrachten folgende Funktion:

$$\chi_A(x) = \begin{cases} 1 & \text{falls } x \in A \\ \text{n. d.} & \text{sonst} \end{cases}$$

χ_A ist Turing-berechenbar: modifizieren M so zu M':

- $M'(x)$ simuliert $M(x)$
- gerät $M(x)$ in einen Endzustand und ist $\alpha z \beta$, $z \in Z_E$ die Konfiguration von M, so ersetzt M' β durch 1 und geht in einen Endzustand $z' \in Z_E'$

$\Rightarrow$ partiell charakteristische χ_A für $A \in \mathfrak{L}_0$ sind turing-berechenbar.

Wie steht es mit der charakteristischen Funktion?
Ist $A \subseteq \Sigma^*$ so ist c_A die charakteristische Funktion von A,

$$c_A = \begin{cases} 1 & \text{für } x \in A \\ 0 & \text{für } x \notin A \end{cases}$$

Ist c_A turing-berechenbar? (später: ganz sicher nicht!)

5.4. Andere Typen von Turing-Maschinen

Bisher hatten wir *nichtdeterministische* und *deterministische* Turing-Maschinen.
Was können wir noch verändern?

- mehrdimensionale Bänder
- mehrere Köpfe

- zusätzliche Bänder
- Bänder mit speziellen Funktionen (Eingabeband, Ausgabeband, Rechenband,...)

Satz 5.4.1
Alle genannten Typen von Turingmaschinen (und Kombinationen davon) sind im Hinblick auf ihre Berechnungskraft äquivalent.

Bemerkung
wichtig ist die *endliche* Beschreibung!

5.5. Die Churchsche These

Satz 5.5.1 (Churchsche These)
Die durch die formale Definition der Turing-Berechenbarkeit erfasste Klasse von Funktionen stimmt genau mit der Klasse der im primitiven Sinne berechenbaren Funktionen überein.

Bemerkung
Kein Beweis, lediglich Plausibilitätsbetrachtung. Beachte: die These trifft keine Effizienzaussage, es zählt nur die reine Berechenbarkeit.

6. Primitv rekursive und partiell rekursive Funktionen

Idee:

- starten mit ganz wenigen, einfachen Funktionen
- definieren ein paar einfache Operationen, die aus Funktionen neue Funktionen machen

6.1. Primitiv rekursive Funktionen

Definition 6.1.1
Grundfunktionen (E)

1. Nachfolgerfunktion: $\text{succ}\colon \mathbb{N} \mapsto \mathbb{N}\colon \text{succ}(n) = n + 1$
2. konstante Funktion: $C_k^m\colon \mathbb{N}^m \mapsto \mathbb{N}\colon C(m_1, \ldots, m_m) = k$
3. Projektionsfunktionen: $id_l^m\colon \mathbb{N}^m \mapsto \mathbb{N}\colon id_l^m(m_1, \ldots, m_m) = m_l$

Die genannten Funktionen bilden die Menge E der Grundfunktionen.

Jetzt die Operationen:

Definition 6.1.2
Substitution: $\text{SUB}_i^m\colon \mathbb{F}_i \times \underbrace{(\mathbb{F}_m)^i}_{i\times\mathbb{F}_m\text{-stellige Funktionen}}$
Ist $g \in \mathbb{F}_i$ und sind $h_1, \ldots, h_i \in \mathbb{F}_m$, so ist die Funktion $\text{SUB}_i^m(g, h_1, \ldots, h_i)$ definiert durch:

$$\text{SUB}_i^m(g, h_1, \ldots, h_i)(n_1, \ldots, n_m) = g(h_1(n_1, \ldots, n_m), \ldots, h_i(n_1, \ldots, n_m))$$

Primitive Rekursion: $\text{PR}^{m+1}\colon \mathbb{F}_m \times \mathbb{F}_{m+2} \mapsto \mathbb{F}_{m+1}$
Sind $g \in \mathbb{F}_m$ und $h \in \mathbb{F}_{m+2}$, so ist die Funktion $f = \text{PR}^{m+1}(g, h)$ definiert durch:

$$\begin{aligned} f(0, n_1, \ldots, n_m) &= g(n_1, \ldots, n_m) \\ f(n+1, n_1, \ldots, n_m) &= h(n, n_1, \ldots, n_m, f(n, n_1, \ldots, n_m)) \end{aligned}$$

Beispiel

$$\mathrm{SUB}_1^1(\mathrm{succ}, \mathrm{succ})(n) = n + 2$$

ähnlich $n + k$: $n + 5 = \mathrm{SUB}(\mathrm{succ}, \mathrm{SUB}_1^1(\mathrm{succ}, \mathrm{SUB}_1^1(\mathrm{succ}, \mathrm{SUB}_1^1(succ, succ))))$

$\mathrm{PR}^3(\mathrm{succ}, id_3^3) = f$

$$\begin{aligned} f(0, m) &= \mathrm{succ}(m) = m + 1 \\ f(n + 1, m) &= id_3^3(n, m, f(n, m)) = f(n, m) \\ f(n, m) &= m + 1 \end{aligned}$$

Addition: $add(n, m) = n + m$, $add(0, m) = m$, $add(n + 1, m) = add(n, m) + 1$
Damit gilt für $add = \mathrm{PR}(g, h)$:

$$\begin{aligned} g(m) = m \mapsto g &= id_1^1 \\ h(n, m, add(n, m)) &= \mathrm{succ}(id_3^3(n, m, add(n, m))) \\ h &= \mathrm{SUB}_1^3(\mathrm{succ}, id_3^3) \end{aligned}$$

Damit ist $add = \mathrm{PR}(id_1^1, \mathrm{SUB}_1^3(\mathrm{succ}, id_3^3))$

Definition 6.1.3
Die Klasse $\mathbb{P}\mathrm{r}$ der primitv rekursiven Funktionen ist definiert als

$$\mathbb{P}\mathrm{r} = \Gamma_{\{\mathrm{SUB,PR}\}}(\{\mathrm{succ}\} \cup \{c_k^m : m, k \in \mathbb{N}\} \cup \{id_l^m : 1 \leq l \leq m\})$$

der Menge aller Funktionen, die sich durch endlichmalige Anwendung von SUB und PR aus den Grundfunktionen gewinnen lassen.

Bemerkung
$\Gamma_{\{\mathrm{SUB,PR}\}}(E)$ ist algebraische Hülle auf E.

6.2. Beispiele für primitiv rekursive Funktionen

Multiplikation:
mult: $\mathbb{N}^2 \mapsto \mathbb{N}$ mit $\mathrm{mult}(n, m) = n \cdot m$

$$\begin{aligned} \mathrm{mult}(0, m) &= 0 = g(m) \\ \mathrm{mult}(n + 1, m) &= \mathrm{mult}(n, m) + m = h(n, m, \mathrm{mult}(n, m)) \end{aligned}$$

Damit gilt für $\mathrm{mult} = \mathrm{PR}(g, h)$:

$$\begin{aligned} g &= C_0^1 \\ h &= \mathrm{SUB}_2^3(\mathrm{add}, id_2^3, id_3^3) \end{aligned}$$

$\Rightarrow \text{mult} = \text{PR}(C_0^1, \text{SUB}_2^3(\text{add}, id_2^3, id_3^3))$

Potenz:
pot: $\mathbb{N}^2 \mapsto \mathbb{N}$ $\text{pot}(n, m) = m^n$

$$\text{pot}(0, m) = 1 = g(m) \rightarrow g = C_1^1$$
$$\text{pot}(n+1, m) = m \cdot \text{pot}(n, m) = h(n, m, \text{pot}(n, m))$$
$$\Rightarrow h = \text{SUB}_2^3(\text{mult}, id_2^3, id_3^3)$$

$\hat{\text{pot}}$: $\mathbb{N}^2 \mapsto \mathbb{N}$ $\hat{\text{pot}}(n, m) = n^m$

$$\hat{\text{pot}} = \text{SUB}_2^2(\text{pot}, id_2^2, id_1^2)$$

modifizierter Vorgänger:
pred: $\mathbb{N} \mapsto \mathbb{N}$ $\text{pred}(n) = \begin{cases} 0 & \text{für } n = 0 \\ n-1 & \text{für } n \neq 0 \end{cases}$

$$\text{pred}(0) = 0 \ (g \text{ müsste nullstellige Funktion } C_0^0 \text{ sein})$$
$$\text{pred}(n+1) = 1 + \text{pred}(n)$$
$$h(n, \text{pred}(n)) \rightarrow h = \text{SUB}_1^2(\text{succ}, id_2^2)$$

Umweg über eine zweistellige Vorgängerfunktion:

$$\text{pred}_2(n, m) = \begin{cases} 0 & \text{falls } n = 0 \\ n-1 & \text{sonst} \end{cases}$$

$$\text{pred}_2(0, m) = 0 \qquad \rightarrow g = C_0^1$$
$$\text{pred}_2(n+1, m) = 1 + \text{pred}_2(n, m) \qquad \rightarrow h(n, m, \text{pred}_2(n, m))$$
$$h = \text{SUB}_1^3(\text{succ}, id_3^3)$$
$$\text{pred}_2 = \text{PR}(C_0^1, \text{SUB}_1^3(\text{succ}, id_3^3))$$

Daraus gewinnt man durch einfache Substitution die Vorgängerfunktion:

$$\text{pred}(n) = \text{pred}(n, n) \qquad \text{pred} = \text{SUB}_2^1(\text{pred}_2, id_1^1, id_1^1)$$

Weitere Beispiele könnten sein: ganzzahlige Division, Rest, Teiler, Primzahltest, Primzahlanzahl, Primfaktorzerlegung,...

6.3. Weitere Rekursionsarten

Werteverlaufsrekursion

Beispiel
Fibonaccizahlen F_i:

$$F_0 = F_1 = 1$$
$$F_{n+1} = F_n + F_{n-1}$$

Die Funktion geht durch Werteverlaufsrekursion aus den Funktionen $g_0, g_1, g_2, \ldots, g_k, h$ hervor, genau dann, wenn gilt:

$$\begin{aligned} f(0, n_1, \ldots, n_m) &= g_0(n_1, \ldots, n_m) \\ f(1, n_1, \ldots, n_m) &= g_1(n_1, \ldots, n_m) \\ &\vdots \\ f(k, n_1, \ldots, n_m) &= g_k(n_1, \ldots, n_m) \end{aligned}$$

$$\begin{aligned} f(n+1, n_1, \ldots, n_m) = h(&n, n_1, \ldots, n_m, \\ &f(n, n_1, \ldots, n_m), f(n-1, n_1, \ldots, n_m), \ldots, f(n-l, n_1, \ldots, n_m)) \\ &n+1 > k, \quad l \leq k \end{aligned}$$

Satz 6.3.1
Die Klasse Pr ist abgeschlossen bezüglich Werteverlaufsrekursion.

Beweis:
Sieen $g_0, \ldots, g_n, h \in \text{Pr}$ und sei f durch Werteverlaufsrekursion basierend auf $g_0, \ldots, g_k, h$ definiert
z. z.: $f \in \text{Pr}$ (o. B. d. A. f einstellig)
Wir wählen eine clevere Codierung für die Funktionswerte $f(0), f(1), \ldots, f(n)$:

$$\varphi(n) = \prod_{i=0} pz(i)^{f(i)}$$

$pz(i)$ steht hier für die Funktion, die die i-te Primzahl errechnet ($p(0) = 2, p(1) = 3, \ldots$).
(Idee: $f(n)$ kann leicht aus $\varphi(n)$ gewonnen werden und $\varphi \in \text{Pr}$)

Behauptung: $\varphi \in \text{Pr}$

Beweis: $\varphi(0) = 2^{f(0)}$ bzw. $\varphi(0) = 2^{a_0}$ mit $a_0 = f(0)$
$\varphi(n+1) = \varphi(n) \cdot pz(n+1)^{f(n+1)} = \varphi(n) \cdot pz(n+1)^{g(n,\varphi(n))}$
wobei g so beschaffen sein soll, dass $f(n+1) = g(n, \varphi(n))$.
$\Rightarrow \varphi$ lässt sich mittels primitver Rekursion und $pz, g, C^1_{a_0}$ beschreiben.
$\Rightarrow \varphi \in \text{Pr}$ ∎

Bemerkung
$g \in \text{Pr}$? g stellt basierend auf $\varphi(n)$ und n den Wert $f(n+1)$ zur Verfügung.

$$g(n, \varphi(n)) \stackrel{!}{=} f(n+1) = h(n, f(n), f(n-1), \ldots, f(f(n-l)))$$

$\Rightarrow g(n, \varphi(n)) = h(n, exp(pz(n), \varphi(n)), exp(pz(n-1), \varphi(n)), \ldots, exp(pz(n-l), \varphi(n)))$
g lässt sich mittels SUB und exp (Exponent von $pz(a)$ in der Primfaktorzerlegung von n), pz, div und Vorgängerfunktion ausdrücken.

Simultanrekursion

$f_1(0) = k_1 \qquad f_2(0) = k_2$
$f_1(n+1) = h_1(n, f_1(n), f_2(n)) \qquad f_2(n+1) = h_2(n, f_1, (n), f_2(n))$

Beispiel: $f(2n) = f_1(n)$ und $f(2n+1) = f_2(n)$

Satz 6.3.2
$\mathbb{P}\text{r}$ ist abgeschlossen bezüglich simultaner Rekursion.

Beweis:
Seien $h_1, h_2 \in \mathbb{P}\text{r}$ und $k_1, k_2 \in \mathbb{N}$
Sei $f_1(0) = k_1$ und $f_2(0) = k_2$ und $f_1(n+1) = h_1(\ldots)$ und $f_2(n+1) = h_2(\ldots)$ wie oben.

$$f(n) = \begin{cases} f_1(\frac{n}{2}) & n \text{ gerade} \\ f_2(\frac{n-1}{2}) & n \text{ ungerade} \end{cases}$$

$\Rightarrow f(0) = k_1$ und $f(1) = k_2$

$$f(2n+2) = f_1(n+1) = h_1(n, f_1(n), f_2(n))$$
$$f(2n+3) = f_2(n+1) = h_2(n, f_1(n), f_2(n))$$

bzw.

$$f(n+1) = \begin{cases} h_2(\frac{n-2}{2}, f(n-2), f(n-1)) & n \text{ gerade} \\ h_1(\frac{n-1}{2}, f(n-1), f(n)) & n \text{ ungerade} \end{cases}$$

Zudem ist

$$n \text{ gerade} \leftrightarrow \overline{\text{sgn}}(\text{mod}(n, 2)) = 1$$
$$n \text{ ungerade} \leftrightarrow \overline{\text{sgn}}(\text{mod}(n, 2)) = 1$$

$\rightarrow$ f ist durch Werteverlaufsrekursion und Fallunterscheidung definiert $\Rightarrow$ $f \in \mathbb{P}\text{r}$. ∎

Gibt es Funktionen, die man vernünftigerweise als berechenbar ansehen sollte, die aber nicht in $\mathbb{P}\mathrm{r}$ liegen?
Antwort: *ja*: alle Funktionen in $\mathbb{P}\mathrm{r}$ sind total. Allerdings ist die Funktion $nd\colon \mathbb{N} \mapsto \mathbb{N}$ mit $nd(n) = \text{n. d.}$ für alle n als berechenbar anzusehen.

Gibt es auch totale Funktionen, die zwar vernünftigerweise als berechenbar gelten, aber nicht in $\mathbb{P}\mathrm{r}$ liegen?
Antwort: *ja*.

6.4. Allgemein rekursive und partiell rekursive Funktionen

Wir überlegen zunächst, dass D. Hilbert mit seiner Aussage, es gebe keine berechenbaren, totalen Funktionen außerhalb von $\mathbb{P}\mathrm{r}$, irrte.

nicht konstruktiv:
jede Funktion in $\mathbb{P}\mathrm{r}$ lässt sich als Wort über einem geeigneten Alphabet Γ darstellen. Man kann alle Wörter über Γ nummerieren. Man kann weiter alle Wörter über Γ nummerieren, die zu Funktionen in $\mathbb{P}\mathrm{r}$ gehören. Also kann man Funktionen in $\mathbb{P}\mathrm{r}$ mittels ihrer Wörter nummerieren.
Jetzt definieren wir eine Funktion mittels Diagonalisierung:

$$\begin{array}{llll} N & 0 & 1 & \dots \\ f_0 & & & \\ f_1 & & & \\ \vdots & & & \end{array}$$

$f(n) = f_n(n) + 1$ - offenbar ist die Funktion einstellig total, aber nicht $\in \mathbb{P}\mathrm{r}$, jedoch sehr wohl XFOO++ berechenbar.

Definition 6.4.1
Die Peter-Funktion (auch **Ackermann-Funktion**)

$$P\colon \mathbb{N}^2 \mapsto \mathbb{N}$$

ist definiert durch

$$\begin{aligned} P(0, x) &= y + 1 \\ P(x + 1, 0) &= P(x, 1) \\ P(x + 1, y + 1) &= P(x, P(x + 1, y)) \end{aligned}$$

Bemerkung
Vernünftigerweise ist diese Funktion berechenbar, aber nicht primitiv rekursiv.

Beispiel

$$P(0,y) = y+1$$
$$P(1,0) = P(0,1) = 2 \qquad \rightarrow\ P(1,y) = y+2$$
$$P(1,y+1) = P(0,P(1,y)) = P(1,y)+1$$
$$P(2,0) = P(1,1) = 3$$
$$P(2,y+1) = P(1,P(2,y)) = P(2,y)+2 \qquad \rightarrow\ P(3,y) = 2y+3$$
$$P(3,0) = P(2,1) = 5$$
$$P(3,y+1) = P(2,P(3,y)) = 2 \cdot P(3,y)+3 \qquad \rightarrow\ P(3,y) = 2^{y+3}-3$$
$$P(4,0) = P(3,1) = 2^{1+3}-3 = 13$$
$$P(4,y+1) = P(3,P(4,y)) = 2^{P(4,y)+3}-3 \qquad \rightarrow\ P(4,y) = 2^{\underbrace{\cdot^{\cdot^{\cdot}2^{16}}}_{y\text{-mal}}}-3$$
$$P(4,1) = 2^{13+3}-3 = 65.533$$
$$P(4,2) = 2^{2^{16}}-3 =? \leftarrow \text{ Zahl mit 21.000 Dezimalstellen}$$

Satz 6.4.2
$P \notin \mathbb{P}\mathrm{r}$

Wie könnte man $\mathbb{P}\mathrm{r}$ erweitern, um alle berechenbaren Funktionen beschreiben zu können?

Definition 6.4.3
Der μ-Operator

Ist $g \in \mathbb{F}_{m+1}$ so ist $f = \mu g$ (ausführlich schreibt man $f(n_1,\ldots,n_m) = \mu g(g(n_1,\ldots,n_m,y) = 0)$) genau dann wenn

1. $f(n_1,\ldots,n_m) = \min y \in \mathbb{N}\colon g(n_1,\ldots,n_m,y) = 0$ sofern dieses Minimum existiert und für alle $m < \min y \in \mathbb{N}\colon g(n_1,\ldots,n_m,y) = 0$ der Funktionswert $g(n_1,\ldots,n_m,m)$ definiert ist
2. In allen anderen Fällen ist f nicht definiert

Bemerkung
Das entspricht einer while-Schleife

while $g(n_1,\ldots,n_m,x) \neq 0$ **do**
  $x := x+1$;
end while
OUTPUT x

Beispiel: die nirgends definierte Funktion $g(n,m) = C_1^2(n,m)$

Definition 6.4.4
Die Klasse der **partiell rekursiven Funktionen** $\mathbb{P}$ ist definiert durch

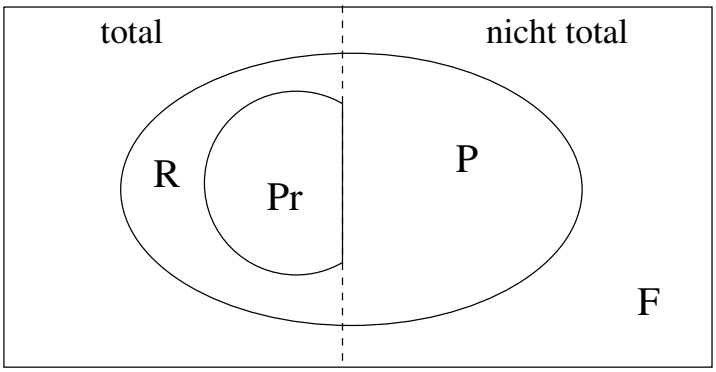
$$\mathbb{P} = \Gamma_{\{\mathrm{SUB},\mathrm{PR},\mu\}}(E)$$

Bemerkung
Man kann zeigen: $\mathbb{P} = \Gamma_{\{\mu\}}(\mathbb{P}\mathrm{r})$

Definition 6.4.5
Die Klasse der **allgemein rekursiven Funktionen** $\mathbb{R}$ ist definiert als Menge der totalen Funktionen in $\mathbb{P}$.

Satz 6.4.6
Es gilt:

$$\emptyset \subsetneq \mathbb{P}\mathrm{r} \subsetneq \mathbb{R} \subsetneq \mathbb{P} \subsetneq \mathbb{F}$$



Wir wollen uns mit dem Zusammenhang zwischen $\mathbb{P}$ und der Menge der Turing-berechenbaren Funktionen befassen.
„$\mathbb{P} \subseteq$ TM-berechenbar“ zeigt, dass Bestandteile von $f \in \mathbb{P}$ von TM simuliert werden können.

Index