Theoretische Informatik

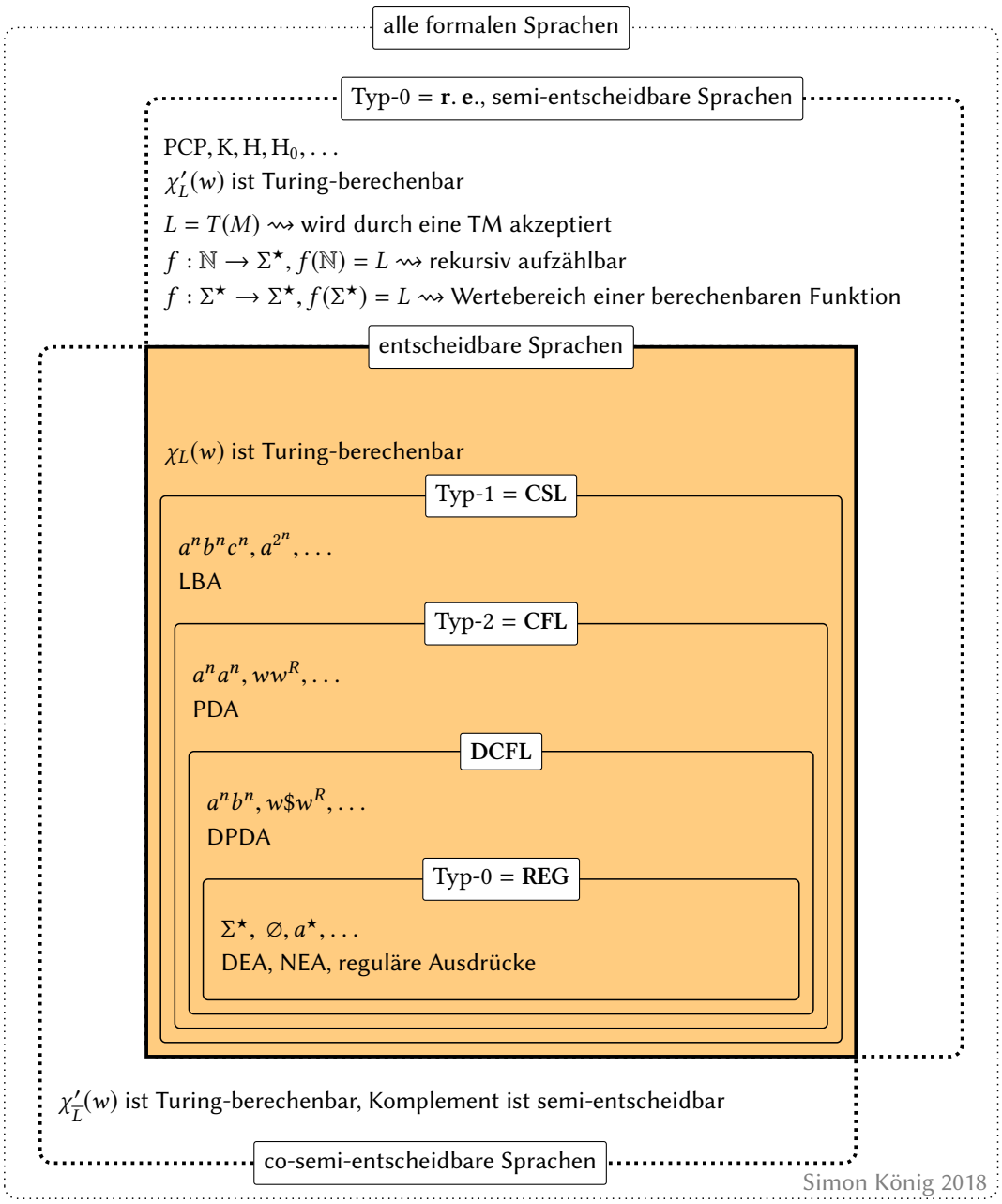
Theoretische Informatik 1 und 2, Kurzzusammenfassung

Simon König

Inhaltsverzeichnis

1: Überblick

Ein kurzer Gesamtüberblick über den Zusammenhang von formalen Sprachen zur Komplexität und Berechenbarkeitstheorie:



1

Kapitel 1: Grundlagen

1: Schreibweisen und Definitionen

1.1 Formale Sprachen

Definition 1.1: Alphabet

Als Alphabet bezeichnen wir eine endliche, nichtleere Menge, deren Elemente Buchstaben genannt werden.
Dieses wird üblich mit Σ bezeichnet.

Definition 1.2: Freies Monoid über Σ

Ein Monoid ist eine Menge mit einer assoziativen Verknüpfung und einem neutralen Element.
Die Menge aller endlichen Zeichenketten, die sich aus Elementen von Σ bilden lassen bilden mit der *Konkatenation* ein Monoid $\Sigma^\star$.
Das leere Wort (ϵ) bildet das neutrale Element.

Definition 1.3: Grammatik

Eine Grammatik ist ein Quadrupel:

$$G = (V, \Sigma, P, S)$$

V Eine Menge an Zeichen, den Variablen

Σ Das Alphabet, $V \cap \Sigma = \varnothing$

P Die Produktionsmenge, $P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^\star$

S Das Startsymbol oder die Startvariable, $S \in V$

1.2 Turingmaschinen und deren Kodierungen

Für Kodierungen von Turingmaschinen wird stets ein Wort $w \in \{0,1\}^\star$ verwendet. Die zugehörige Turingmaschine der Kodierung wird als M_w bezeichnet. In der Literatur ist oft auch üblich für die Maschine nur M zu verwenden, die zugehörige Kodierung ist dann $\langle M \rangle \in \Sigma^\star$.
Die zugrundeliegende Gödelisierung darf keinen Einfluss auf die Arbeitsweise der Maschine haben.
Es wird festgelegt, dass jedes Wort über dem Alphabet $\{0,1\}$ einer Turingmaschine zugeordnet wird,

dabei können mehrere Kodierungen auf die gleiche Turingmaschine abgebildet werden.

1.3 Grundlagen der Aussagenlogik

1.3.1 Syntax der Aussagenlogik

- Atomare Formeln: A_i mit $i \in \mathbb{N}$
- F und G Formeln $\rightarrow (F \wedge G), (F \vee G)$ und $\neg F$ auch Formeln

Aus dieser grundlegenden Syntaxdefinition lassen sich die in der Mathematik sonst auch üblichen Verknüpfungen als Abkürzungen darstellen

- Implikation: $F_i \Rightarrow F_j \equiv (\neg F_i \vee F_j)$
- Äquivalenz: $F_i \Leftrightarrow F_j \equiv ((F_i \wedge F_j) \vee (\neg F_i \wedge \neg F_j))$
- n-faches Und: $\bigwedge_{i=1}^{n} F_i \equiv (\ldots((F_1 \wedge F_2) \wedge F_3)\ldots \wedge F_n)$
- n-faches Oder: $\bigvee_{i=1}^{n} F_i \equiv (\ldots((F_1 \vee F_2) \vee F_3)\ldots \vee F_n)$

1.3.2 Semantik der Aussagenlogik

$D \subseteq \{A_1, A_2, \ldots\}$ eine Teilmenge der Menge aller Variablen (atomare Formeln).
Eine Abbildung $\mathcal{A} : D \rightarrow \{0, 1\}$ heißt Belegung. Durch diese Abbildung erhält jede atomare Formel einen Wert. Der Wert von zusammengesetzten aussagenlogischen Formeln berechnet sich wie üblich über die induktive Definition der Semantik, dies wird hier nicht näher ausgeführt.

- Eine Belegung $\mathcal{A}$ ist **passend** zu einer Formel F, falls alle in F vorkommenden atomaren Variablen zum Definitionsbereich von $\mathcal{A}$ gehören.
- Eine Belegung $\mathcal{A}$ ist **Modell** für eine Formel, falls $\mathcal{A}$ zu F passend ist und $\mathcal{A}(F) = 1$ gilt.
  Man schreibt dann $\mathcal{A} \vDash F$.
- F ist **erfüllbar**, falls ein Modell für F existiert.
- F ist **gültig**, falls alle passenden Belegungen Modelle sind. $\rightarrow F$ nennt man dann eine **Tautologie**.
  Das Komplement einer Tautologie ist unerfüllbar.
- Zwei Formeln F und G heißen **semantisch äquivalent**, wenn alle zu beiden passenden Belegungen $\mathcal{A}$ gilt: $\mathcal{A}(F) = \mathcal{A}(G)$. Man schreibt dann $F \equiv G$.

Wichtige Äquivalenzen sind die folgenden:

- Itempotenz: $F \equiv (F \wedge F) \equiv (F \vee F)$
- Kommutativität: $(F \wedge G) \equiv (G \wedge F), (F \vee G) \equiv (G \vee F)$
- Assoziativität: $((F \wedge G) \wedge H) \equiv (F \wedge (G \wedge H)), ((F \vee G) \vee H) \equiv (F \vee (G \vee H))$
- Absorption: $(F \wedge (F \vee G)) \equiv F \equiv (F \vee (F \wedge G))$
- Distributivität: $(F \wedge (G \vee H)) \equiv ((F \wedge G) \vee (F \wedge H)), (F \vee (G \wedge H)) \equiv ((F \vee G) \wedge (F \vee H))$
- Doppelnegation: $\neg\neg F \equiv F$
- deMorgan-Regeln: $\neg(F \wedge G) \equiv (\neg F \vee \neg G), \neg(F \vee G) \equiv (\neg F \wedge \neg G)$

1.3.3 Prädikatenlogik

2

Kapitel 2: Formale Sprachen und Automatentheorie

1: Einstieg

In der Theorie um formale Sprachen geht es grundsätzlich darum, Probleme durch formale Sprachen beschreiben zu können. Diese Probleme werden dann in Klassen eingeordnet und gegeneinander abgrenzt. Wichtig ist zu verstehen, wie man eine formale Sprache beschreiben kann, zum Beispiel auch mithilfe der Automatentheorie und genauso wo die Unterschiede zwischen den einzelnen Klassen liegen.

2: Endliche Sprachen FIN

$$\mathbf{FIN} \subset \mathbf{REG} \subset \mathbf{DCFL} \subset \mathbf{CFL} \subset \mathbf{CSL} \subset \mathbf{REC} \subset \mathbf{r.e.}$$

Alle endlichen Sprachen sind regulär.

2.1 Beispiele

- $L_1 = \varnothing$ (leere Sprache ist endlich)
- $L_2 = \Sigma$ (nur die Buchstaben)
- $L_3 = \{aaa, baba\}$ (z.B. nur zwei Wörter)

3: Reguläre Sprachen

Typ-3 = **REG** ⊂ **DCFL** ⊂ **CFL** ⊂ **CSL** ⊂ **REC** ⊂ **r. e.**

3.1 Automatenmodell DEA

Ein deterministischer endlicher Automat ist ein 5-Tupel

$$M = (Z, \Sigma, \delta, z_0, E)$$

Z endliche Zustandsmenge

Σ Eingabealphabet

δ Überführungsfunktion $\delta : Z \times \Sigma \to Z$

z_0 Startzustand, $z_0 \in Z$

E Endzustandsmenge, $E \subseteq Z$

Es lässt sich außerdem eine erweiterte Funktion $\hat{\delta}$ definieren:

$$\hat{\delta} : Z \times \Sigma^\star \to Z$$

Mit den folgenden Eigenschaften:

$$\hat{\delta}(z, \epsilon) = z$$

$$\hat{\delta}(z, ax) = \hat{\delta}(\delta(z, a), x)$$

Die von einem deterministischen Automaten M akzeptierte Sprache ist

$$T(M) = \left\{ w \in \Sigma^\star \,\middle|\, \hat{\delta}(z_0, w) \in E \right\}$$

3.2 Automatenmodell NEA

Ein nichtdeterministischer endlicher Automat ist ein 5-Tupel

$$M = (Z, \Sigma, \delta, S, E)$$

Z endliche Zustandsmenge

Σ Eingabealphabet

δ Überführungsfunktion $\delta : Z \times \Sigma \to \mathrm{Pot}(Z)$

S Startzustandsmenge, $S \subseteq Z$

E Endzustandsmenge, $E \subseteq Z$

Der NEA ist formal stärker als der DEA, sie akzeptieren jedoch beide die selbe Sprachklasse.
Die akzeptierte Sprache eines nichtdeterministischen endlichen Automaten ist:

$$T(M) = \left\{ w \in \Sigma^\star \,\middle|\, \hat{\delta}(S, w) \cap E \neq \varnothing \right\}$$

3.3 Reguläre Ausdrücke

Die regulären Sprachen lassen sich zusätzlich zu den zwei Automatenmodellen auch durch sog. reguläre Ausdrücke beschreiben. Eine Definition für die Syntax der regulären Ausdrücke ist:

- $\varnothing$ und ϵ sind reguläre Ausdrücke.
- a ist ein regulärer Ausdruck für alle $a \in \Sigma$
- Wenn α und β eguläre Ausdrücke sind, dann sind auch $\alpha\beta$, $(\alpha|\beta)$ und $(\alpha)^\star$ reguläre Ausdrücke.

Die Semantik der regulären Ausdrücke ist ebenso induktiv bestimmt:

- $L(\varnothing) = \varnothing$ und $L(\epsilon) = \{\epsilon\}$
- $L(a) = \{a\}$ für jedes $a \in \Sigma$
- $L(\alpha\beta) = L(\alpha)L(\beta)$, $L(\alpha|\beta) = L(\alpha) \cup L(\beta)$, $L((\alpha)^\star) = L(\alpha)^\star$

3.4 Sätze zu den regulären Sprachen:

- Typ-3 Sprachen können *nicht* inhärent mehrdeutig sein, da sich zu jeder Sprache ein Minimalautomat bilden lässt.
- Die Klasse der Typ-3 Sprachen ist unter allen boole'schen Operationen, Sternoperation und der Konkatenation abgeschlossen.
- Für reguläre Sprachen ist das Wortproblem (in Linearzeit), das Leerheitsproblem, das Äquivalenzproblem sowie das Schnittproblem entscheidbar.
- Alle Typ-2 Sprachen über einem einelementigen Alphabet sind bereits regulär.

3.4.1 Pumping-Lemma für Typ-3

Für jede reguläre Sprache L gibt es ein $n \in \mathbb{N}$, so dass für jedes $x \in L$ mit $|x| \geq n$ eine Zerlegung in drei Teile exisitert: $x = uvw$, so dass die drei Bedingungen erfüllt sind:

- $|v| \geq 1$
- $|uv| \leq n$
- $\forall i \in \mathbb{N} : uv^i w \in L$ („Pump-Bedingung“)

Gilt die Negation dieser Aussage, also

$$\forall n \in \mathbb{N} : \exists x \in L, |x| \geq n : \forall u, v, w \in \Sigma^\star, x = uvw, |v| \geq 1, |uv| \leq n : \exists i \in \mathbb{N} : uv^i w \notin L$$

so ist L nicht regulär!

ABER: Das Pumping-Lemma gibt keine Charakterisierung der Typ-3 Sprachen an! Es gibt auch Sprachen, die nicht Typ-3 sind, die Aussage des Lemmas aber trotzdem erfüllen!
Das Pumping-Lemma gibt also nur eine Möglichkeit, zu Beweisen, dass eine Sprache *nicht* regulär ist! (Siehe Unterabschnitt 3.6.1)

3.4.2 Myhill-Nerode-Äquivalenz

Mit der Myhill-Nerode-Äquivalenz ist es möglich nachzuweisen, ob eine Sprache regulär ist.

$$xR_L y \iff \left[\forall w \in \Sigma^\star : xw \in L \Leftrightarrow yw \in L\right]$$

Bzw. anhand eines DEA (dies führt zu einer Verfeinerung von R_L)

$$xR_M y \iff [\delta(z_0, x) = \delta(z_0, y)]$$

Es gilt:

$$xR_M y \Rightarrow \forall w \in \Sigma^\star : \delta(z_0, xw) = \delta(z_0, yw) \Rightarrow xR_L y$$

Die Sprache $L \subseteq \Sigma^\star$ ist genau dann regulär, wenn der Index der Myhill-Nerode-Äquivalenz $R_L$endlich ist.

3.4.3 Erkennung durch Monoide – Syntaktisches Monoid

Sei $L \subseteq \Sigma^\star$ eine formale Sprache und M ein Monoid.
M erkennt L, wenn eine Teilmenge $A \subseteq M$ und ein Homomorphismus $\varphi : \Sigma^\star \to M$ existiert, so dass gilt:

$$L = \varphi^{-1}(A) \qquad \text{d.h. } w \in L \Leftrightarrow \varphi(w) \in A$$
$$L = \varphi^{-1}(\varphi(L)) \qquad \text{d.h. } w \in L \Leftrightarrow \varphi(w) \in \varphi(L)$$

Weiter kann man für eine konkrete Sprache L die *syntaktische Kongruenz* definieren:

$$w_1 \equiv_L w_2 \iff [\forall x, y \in \Sigma^\star : xw_1y \in L \Leftrightarrow xw_2y \in L]$$

Basierend auf dieser Kongruenz definieren wir das Quotientenmonoid der Kongruenz, dessen Elemente die Äquivalenzklassen sind.
Das Quotientenmonoid oder auch syntaktisches Monoid bezüglich der syntaktischen Kongruenz wird mit

$$\text{Synt}(L) := \left(\Sigma^\star / \equiv_L\right)$$

bezeichnet.
Für jede Sprache L gibt es ein syntaktisches Monoid das L mit dem Homomorphismus

$$\varphi : L \to \text{Synt}(L), w \mapsto [w]$$

erkennt. Ist $|\text{Synt}(L)|$ endlich, so ist L regulär bzw. erkennbar.

3.4.4 Wohldefiniertheit des Produkts der Äquivalenzklassen

$$[u_1]_{\equiv_L} \cdot [u_2]_{\equiv_L} \stackrel{?}{=} [u_1u_2]_{\equiv_L} \quad \text{wohldefiniert?}$$

Es gilt:

$$u_1, \widetilde{u_1} \in [u_1]_{\equiv_L} \iff u_1 \equiv_L \widetilde{u_1} \iff \left[\forall x, y \in \Sigma^\star : xu_1y \in L \Leftrightarrow x\widetilde{u_1}y \in L\right] \tag{3.1}$$
$$u_2, \widetilde{u_2} \in [u_2]_{\equiv_L} \iff u_2 \equiv_L \widetilde{u_2} \iff \left[\forall x, y \in \Sigma^\star : xu_2y \in L \Leftrightarrow x\widetilde{u_2}y \in L\right] \tag{3.2}$$

Setzt man nun in Gleichung 3.1 für $y = u_2y'$ ein:

$$u_1, \widetilde{u_1} \in [u_1]_{\equiv_L} \iff u_1 \equiv_L \widetilde{u_1} \iff \left[\forall x, y' \in \Sigma^\star : xu_1u_2y' \in L \Leftrightarrow x\widetilde{u_1}u_2y' \in L\right]$$

Äquivalent gilt das selbe für $\widetilde{u_2}$. Ebenso gilt das gleiche für Gleichung 3.2 mit $x = x'u_1$:

$$u_2, \widetilde{u_2} \in [u_2]_{\equiv_L} \Longleftrightarrow u_2 \equiv_L \widetilde{u_2} \Longleftrightarrow \left[\forall x', y \in \Sigma^\star : x'u_1u_2y \in L \Leftrightarrow x'u_1\widetilde{u_2}y \in L\right]$$

Äquivalent gilt das selbe für $\widetilde{u_1}$. Setzt man nun alle Gleichungen zusammen, erhält man:

$$u_1u_2 \equiv_L \widetilde{u_1}\widetilde{u_2} \Longleftrightarrow \left[\forall x, y \in \Sigma^\star : xu_1u_2y \in L \Leftrightarrow x\widetilde{u_1}\widetilde{u_2}y \in L\right]$$

Dies gilt wie oben gezeigt für alle Kombinationen $u_1u_2, u_1\widetilde{u_2}, \widetilde{u_1}u_2$ und $\widetilde{u_1}\widetilde{u_2}$. Damit erzeugt diese Äquivalenz die Klasse:

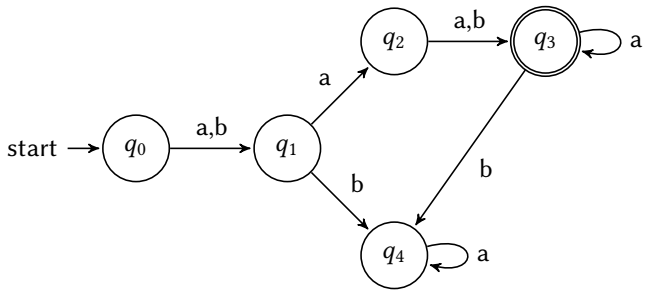
$$[u_1u_2]_{\equiv_L} \text{ mit } u_1u_2, u_1\widetilde{u_2}, \widetilde{u_1}u_2, \widetilde{u_1}\widetilde{u_2} \in [u_1u_2]_{\equiv_L}$$

3.5 Konstruktionsalgorithmus für Minimal-DEAs

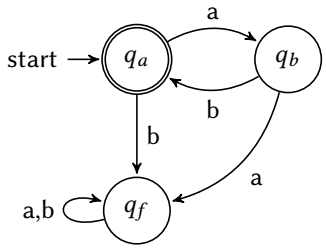
Mit dem Beweis zur Myhill-Nerode-Äquivalenz wird ein Automat definiert, dieser ist isomorph zum Minimalautomaten. Der Index der Myhill-Nerode-Äquivalenz ist genau die Anzahl der Zustände des Minimalautomaten.
Man kann mit einem einfachen Algorithmus aus einem beliebigen DEA den Minimalautomaten erzeugen: Wir ermitteln algorithmisch, welche Zustände nicht äquivalent sind und verschmelzen die übrig bleibenden. Nicht äquivalent sind Zustände, bei denen bei Eingabe eines Worts vom einen aus ein Endzustand erreicht wird, vom anderen jedoch nicht.
So sind im ersten Schritt Zustandspaare aus Endzustand und Nichtendzustand nicht äquivalent und werden markiert.

3.6 Beispiele:

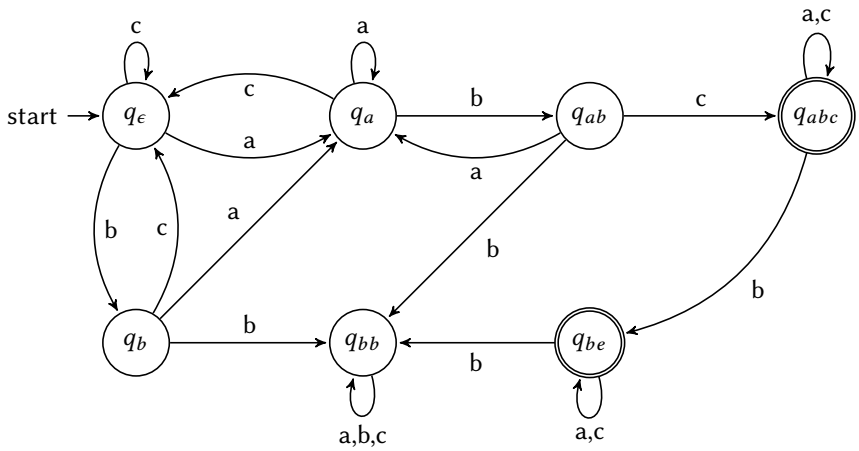
- $L_1 = \Sigma^\star$
- $L_2 = L\left((a|b)a(a|b)a^\star\right)$



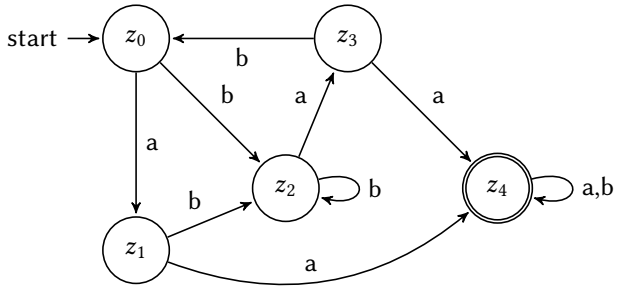
- Automat M mit $L_3 = T(M) = \{(ab)^n \mid n \in \mathbb{N}_0\}$:



- Automat erkennt die Sprache
  $L_3 = \{w \in \{a,b,c\}^\star \mid w \text{ enthält das Teilwort } abc \text{ aber nicht das Teilwort } ab\}$:



Konstruktion für den Minimalautomaten



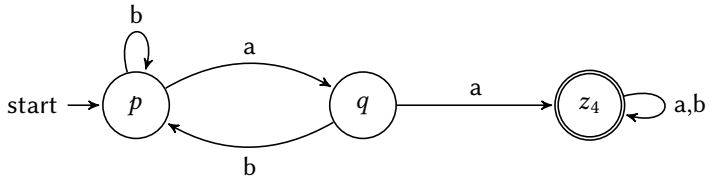
Die Paare $\{z_i, z_4\}$ mit $i = 0, 1, 2, 3$ werden markiert:

z_1				
z_2				
z_3				
z_4	ϵ	ϵ	ϵ	ϵ
	z_0	z_1	z_2	z_3

Durch Testen der Zustandspaare erhält man die untenstehende Tabelle. Hierbei wurden Zeugen für die Inäquivalenz eingetragen.

z_1	a			
z_2		a		
z_3	a		a	
z_4	ϵ	ϵ	ϵ	ϵ
	z_0	z_1	z_2	z_3

Damit lassen sich Zustände zusammenfassen: $p = \{z_0, z_2\}, q = \{z_1, z_3\}$. Der Minimalautomat ist also:



$$T(M) = \left\{waaw' \,\middle|\, w, w' \in \{a,b\}^\star\right\}$$

3.6.1 Pumping-Lemma für Typ-3

Für die Sprache $L = \{a^n b^n \mid n \geq 1\}$:

Zunächst wählt man ein Wort x, mit $|x| \geq n$:

$$x = a^n b^n \in L, \quad |a^n b^n| = 2n > n$$

Nun zu einer beliebigen Zerlegung $x = uvw$, für die die Bedingungen gelten:

$$x = uvw = a^n b^n$$
$$u = a^{n-l-k} \quad v = a^l \quad w = a^k b^n \text{ mit } l \geq 1$$
$$x = (a^{n-l-k})(a^l)(a^k b^n)$$

Pumpt man nun v mit $v = 0$:

$$x = (a^{n-l-k})(a^l)^i(a^k b^n) = (a^{n-l-k})(a^l)^0(a^k b^n)$$
$$x = (a^{n-l-k})(a^k b^n) = a^{n-l} b^n \notin L, \text{ da } l \geq 1$$

3.6.2 Myhill-Nerode-Äquivalenz

Für die Sprache $L = \left\{w \in \{a,b\}^\star \,\middle|\, w \text{ enthält das Teilwort } abb\right\}$:

Finden der Äquivalenzklassen:

$$[\epsilon] = \{\epsilon, b, bb, \ldots\} = \{b^n \mid n \in N_0\}$$
$$[a] = \left\{b^n a^m \,\middle|\, n \in \mathbb{N}_0, m \in \mathbb{N}^+\right\}$$
$$[ab] = \left\{w \in \{a,b\}^\star \,\middle|\, w \text{ enthält das Teilwort } ab \text{ aber nicht das Teilwort } abb\right\}$$
$$[abb] = \left\{wabbw' \,\middle|\, w, w' \in \{a,b\}^\star\right\}$$

3.6.3 Beweis durch Abschlusseigenschaften

Zu zeigen: $L = \{b^n a^m \mid n, m \in \mathbb{N}_0, n \neq m\} \notin \mathbf{REG}$

Annahme:

$$L \in \mathbf{REG}$$

Mit dem Abschluss gegen Komplement folgt

$$\overline{L} \in \mathbf{REG}$$

Wegen Abschluss gegen Schnitt folgt dann auch

$$\overline{L} \cap L(b^\star a^\star) \in \mathbf{REG}$$

Jedoch gilt

$$\overline{L} \cap L(b^\star a^\star) = \{b^n a^n \mid n \in \mathbb{N}_0\} \in \mathbf{DCFL} \supsetneq \mathbf{REG}$$

Widerspruch! $L \notin \mathbf{REG}$

4: Deterministisch kontextfreie Sprachen

$\mathbf{DCFL} \subset \mathbf{CFL} \subset \mathbf{CSL} \subset \mathbf{REC} \subset \mathbf{r.e.}$

4.1 Automatenmodell DPDA

Der deterministische Kellerautomat ist ähnlich definiert wie ein der nichtdeterministische (Siehe Abschnitt 5.1).
Der Unterschied zum PDA liegt dabei, dass beim DPDA in jeder Situation nur ein Übergang möglich sein darf,

$$\forall z \in Z, a \in \Sigma, A \in \Gamma : |\delta(z, a, A)| + |\delta(z, \epsilon, A)| \leq 1$$

und der DPDA akzeptiert nicht durch leeren Keller sondern durch Endzustände.

Ein deterministischer Kellerautomat ist ein 7-Tupel

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \#, E)$$

Z endliche Zustandsmenge

Σ Eingabealphabet

Γ Kelleralphabet

δ Überführungsfunktion $\delta : Z \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to Z \times \Gamma^\star$

z_0 Startzustand, $z_0 \in Z$

$\#$ Keller-Bottom-Symbol $\# \in \Gamma \setminus \{\Sigma\}$

E Endzustandsmenge $E \subseteq Z$

Akzeptierte Sprache eines deterministischen PDA:

$$N(M) := \left\{ w \in \Sigma^\star \,\middle|\, \exists e \in E, V \in \Gamma^\star : (z_0, w, \#) \vdash^\star (e, \epsilon, V) \right\}$$

dies wird auch als *Akzeptieren durch Endzustand* bezeichnet.
Beide Akzeptanzarten, (durch Endzustand und leeren Keller, siehe Abschnitt 5.1) sind äquivalent.

4.2 Sätze zu DCFL

- Eine deterministisch kontextfreie Sprache geschnitten mit einer regulären Sprache ist wieder eine deterministisch kontextfreie Sprache.

  $$L_1 \in \mathbf{DCFL}, L_2 \in \mathbf{REG} \Rightarrow L_1 \cap L_2 \in \mathbf{DCFL}$$

- Die Klasse der deterministisch kontextfreien Sprachen ist nur abgeschlossen unter Komplement.

- Das Leerheitsproblem (Markieren von produktiven Variablen), das Wortproblem (in Linearzeit mit Kellerautomat) und das Äquivalenzproblem sind entscheidbar.

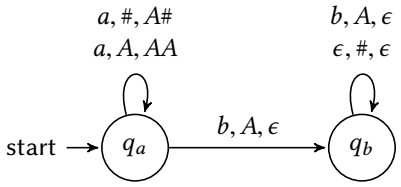
  Zum Äquivalenzproblem:

  $$\begin{aligned} L = L' &\Leftrightarrow L \subseteq L' \wedge L' \subseteq L \\ &\Leftrightarrow L \cap \overline{L'} = \varnothing \wedge L' \cap \overline{L} = \varnothing \end{aligned}$$

  entscheidbar, da Abschluss unter Komplement und Leerheitsproblem entscheidbar.

4.3 Beispiele

- $L_1 = \left\{ w\$w^R \,\middle|\, w \in \Sigma^\star \right\}$ (markierte Palindrome)
- Deterministischer Kellerautomat, der die Sprache $L_2 = \{a^n b^n \mid n \geq 1\}$ akzeptiert:



Konfigurationsübergänge bei Eingabewort *aaabb*:

$$\begin{aligned} (z_a, aaabb, \#) &\vdash (z_a, aabb, A\#) \\ &\vdash (z_a, abb, A\#) \\ &\vdash (z_a, bb, AAA\#) \\ &\vdash (z_a, b, AA\#) \\ &\vdash (z_a, \epsilon, A\#) \\ &\rightsquigarrow aaabb \notin N(M) = L_2 \end{aligned}$$

5: Kontextfreie Sprachen

Typ-2 = **CFL** ⊂ **CSL** ⊂ **REC** ⊂ **r. e.**

5.1 Automatenmodell PDA

Der Push DownAutomat (Kellerautomat) ist ähnlich definiert wie ein nichtdeterministischer endlicher Automat (Siehe Abschnitt 3.2).
Der PDA jedoch hat einen entscheidenden Unterschied, er hat einen sogenannten *Kellerspeicher*, in dem Informationen zwischengespeichert werden können.

Ein Kellerautomat ist ein 6-Tupel

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$$

Z endliche Zustandsmenge

Σ Eingabealphabet

Γ Kelleralphabet

δ Überführungsfunktion $\delta : Z \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to \mathrm{Pot}_e(Z \times \Gamma^\star)$

z_0 Startzustand, $z_0 \in Z$

$\#$ Keller-Bottom-Symbol $\# \in \Gamma \setminus \{\Sigma\}$

Akzeptierte Sprache eines nichtdeterministischen PDA

$$N(M) \coloneqq \left\{ w \in \Sigma^\star \,\middle|\, \exists z \in Z : (z_0, w, \#) \vdash^\star (z, \epsilon, \epsilon) \right\}$$

dies wird auch als *Akzeptieren durch leeren Keller* bezeichnet.
Insbesondere beim deterministischen Kellerautomaten gibt es eine zweite Definition der akzeptierten Sprache, beide Definitionen sind äquivalent (Siehe Abschnitt 4.1).

Konfiguration des PDA Eine Konfiguration ist jedes Element k aus der Menge $Z \times \Sigma^\star \times \Gamma^\star$
Einen Konfigurationsübergang stellt man durch $\vdash$ dar.
Die aktuelle Konfiguration sei $(z, a_1 a_2 \dots a_n, A_1 \dots A_m)$, möglich sind jetzt die Übergänge aus $\delta(z, a_1, A_1)$ und $\delta(z, \epsilon, A_1)$.
Ein möglicher Übergang ist also für $(z', B_1 \dots B_k) \in \delta(z, a_1, A_1)$:

$$(z, a_1 a_2 \dots a_n, A_1 \dots A_m) \vdash (z', a_2 \dots a_n, B_1 \dots B_k A_2 \dots A_m)$$

5.2 Sätze zu CFL

- Alle Typ-2 Sprachen über einem einelementigen Alphabet sind bereits regulär.
- Die Kontextfreien Sprachen sind gegen Substitution abgeschlossen.

- Die Klasse der kontextfreien Sprachen ist abgeschlossen unter Sternoperation, Vereinigung und Konkatenation.

  Zur Vereinigung: $G_1 = (V_1, \Sigma, P_1, S_1), G_2 = (V_2, \Sigma, P_2, S_2), V_1 \cap V_2 = \varnothing, S \notin V_1 \cup V_2$.

  $$G = (V_1 \cup V_2, \Sigma, P_1 \cup P_2 \cup \{(S, S_1), (S, S_2)\}, S) \rightsquigarrow L(G) = L(G_1) \cup L(G_2)$$

- Das Wortproblem ($O(n^3)$) sowie das Leerheits- und Endlichkeitsproblem ist entscheidbar (Siehe Unterabschnitt 5.2.2).
- Die Klasse der Typ-2 Sprachen ist identisch zu der durch EBNF beschreibbaren Sprachen.

5.2.1 Pumping-Lemma für Typ-2

Sei $L \subseteq \Sigma^\star$ eine kontextfreie Sprache, dann gibt es eine Zahl n so, dass für alle $z \in L$ mit $|z| \geq n$ eine Zerlegung mit $z = uvwxy$ in $u, v, w, x, y \in \Sigma^\star$ exisitert für die die drei Bedingungen erfüllt sind:

- $|vx| \geq 1$
- $|vwx| \leq n$
- $\forall i \in \mathbb{N} : uv^i wx^i y \in L$

5.2.2 CYK-Algorithmus zur Lösung des Wortproblems für Typ-2

Mit dem CYK-Algorithmus ist das Wortproblem für Typ-2 in $O(n^3)$ entscheidbar.
Hierfür werden alle Ableitungsmöglichkeiten in einer Tabelle geordnet dargestellt. Ist am Ende die Startvariable als Startknoten für die Ableitung möglich, so ist das Wort in L.

5.3 Chomsky-Normalform

Eine Typ-2 Grammatik (V, Σ, P, S) ist in Chomsky-Normalform (CNF), wenn gilt:

$$\forall (u, v) \in P : v \in V^2 \cup \Sigma$$

Zu jeder Typ-2 Grammatik existiert eine Grammatik G' in CNF, für die gilt $L(G) = L(G')$!
Für alle Ableitungen in CNF gilt: Die Ableitung eines Wortes der Länge n benötigt genau $2n - 1$ Schritte!

5.3.1 Umformungsalgorithmus

1. Zunächst wollen wir erreichen, dass folgendes gilt: $(u, v) \in P \Rightarrow (|v| > 1 \vee v \in \Sigma)$

   (a) **Ringableitungen entfernen:**
   Eine Ringableitung liegt vor, wenn es Variablen $V_1, \ldots V_r$ gibt, die sich im Kreis in einander ableiten lassen, d.h. es gibt Regeln $V_i \to V_{i+1}$ und $V_r \to V_1$.
   Um dies loszuwerden, werden alle Variablen V_i durch eine neue Variable V ersetzt. Überflüssige Regeln wie $V \to V$ werden gelöscht.

   (b) **Variablen anordnen:**
   Man legt eine Ordnung der Variablen fest: $V = \{A_1, A_2, \ldots, A_n\}$, hierfür muss gelten:

   $$A_i \to A_j \in P \Leftrightarrow i < j$$

   Falls dies nicht gilt, müssen Abkürzungen verwendet werden, also alle Produktionen von A_j werden eingesetzt:

   $$P = (P \setminus \{A_i \to A_j\}) \cup \{(A_i, w) \mid (A_j, w) \in P\}$$

2. Jetzt gilt für jede Regel $(u, v) \in P$ entweder $v \in \Sigma$ oder $|v| \geq 2$.

   Für letztere Regeln werden nun **Pseudoterminale** eingeführt. Es werden neue Variablen und Produktionen für jedes Terminalsymbol hinzugefügt, z.B. $V_a \to a$.

3. **Letzter Schritt:** Alle rechten Seiten mit $|v| > 2$ müssen nun noch auf Länge 2 gekürzt werden. Hierfür werden wiederum neue Variablen eingefügt:

   $$A \to C_1C_2C_3$$

   Wird gekürzt zu

   $$A \to C_1D_1$$
   $$D_1 \to C_2C_3$$

5.4 Greibach-Normalform

Eine Typ-2 Grammatik (V, Σ, P, S) ist in Greibach-Normalform (GNF), wenn gilt:

$$\forall (u, v) \in P : v \in \Sigma V^\star$$

Zu jeder Typ-2 Grammatik existiert eine Grammatik G' in GNF, für die gilt $L(G) = L(G')$!

5.4.1 Umformungsalgorithmus

1. **Der erste Algorithmus:**

   Der erste Algorithmus hat zum Ziel, dass alle Produktionen einer bestimmten Ordnung unterliegen. Hierfür werden zunächst die Variablen angeordnet:

   $$V = \{A_1, A_2, \ldots, A_m\}$$

   Nun sollen nur Regeln $A_i \to A_j\alpha$ in P sein, wenn $i < j$ gilt.

   Um dies zu erreichen wird solange Eingesetzt bis keine Regeln dieser Form vorliegen.

   Für alle $A_i \to A_j\alpha \in P$ mit $i > j$ werden alle Produktionsregeln

   $$A_j \to \beta_1 | \ldots | \beta_r$$

   Eingesetzt zu

   $$A_i \to \beta_1\alpha | \ldots | \beta_r\alpha$$

   Und schließlich die Regel $A_i \to A_j\alpha$ aus P gestrichen.

2. **Beseitigung von Linksrekursion:**

   Alle Produktionsregeln sind von der Form:

   $$A \to A\alpha_1 | \ldots | A\alpha_k | \beta_1 | \ldots | \beta_l$$

   Diese können durch diese $2k + 2l$ Regeln ersetzt werden:

   $$A \to \beta_1 | \ldots | \beta_l$$
   $$A \to \beta_1 B | \ldots | \beta_l B$$
   $$B \to \alpha_1 | \ldots | \alpha_k$$
   $$B \to \alpha_1 B | \ldots | \alpha_k B$$

   Nun sind keinerlei Linksrekursionen mehr vorhanden!

3. **Der zweite Algorithmus:**

   Der zweite Algorithmus hat zum Ziel, dass alle rechten Seiten mit einem Terminalsymbol beginnen.

   Da die Regeln mit A_m auf der linken Seite nur in ein Terminal übergehen können, da sie am Ende der Variablenanordnung stehen müssen wir nur alle A_m-Produktionen bei den A_{m-1}-Regeln einsetzen und so weiter.

4. **Der letzte Schritt:**

   Als letztes müssen wir überprüfen ob die B-Regeln aus der Beseitigung der Linksrekursionen die gewünschte Form haben. Da aber alle B-Produktionen entweder mit einem A_i oder einem Terminal beginnen, muss wieder nur eingesetzt werden.

   Schließlich müssen die Terminalsymbole, die in allen Produktionen weiter hinten auftreten durch Pseudoterminale ersetzt werden.

5.5 Beispiele

- $L_1 = \left\{ww^R \,\middle|\, w \in \Sigma^\star\right\}$ (unmarkierte Palindrome)
- Korrekt geklammerte arithmetische Ausdrücke (Dyck-Wörter)

CYK-Algorithmus

Produktionsregeln der Grammatik (CNF):

$$S \to AX|YB, A \to XA|AB|a, B \to XY|BB, X \to YA|a, Y \to XX|b$$

Eingabewort: *abbaab*

Länge	a	b	b	a	a	b
1	$\{A,X\}$	$\{Y\}$	$\{Y\}$	$\{A,X\}$	$\{A,X\}$	$\{Y\}$
2	$\{B\}$	$\varnothing$	$\{X\}$	$\{A,S,Y\}$	$\{B\}$	
3	$\varnothing$	$\varnothing$	$\{A,Y,X\}$	$\{A\}$		
4	$\varnothing$	$\{X\}$	$\{B,X\}$			
5	$\{S,Y\}$	$\{B,S\}$				
6	$\{A,B\}$					

Da die unterste Zelle nun $\{A,B\}$ enthält, ist $w = abbaab \notin L$

5.5.1 Pumping-Lemma für Typ-2

- Für die Sprache $L = \{a^n b^n c^n \mid n \geq 1\}$:

  Zunächst wählt man ein Wort z, mit $|z| \geq n$:

  $$x = a^n b^n c^n \in L, \quad |a^n b^n c^n| = 3n > n$$

  Nun zu einer beliebigen Zerlegung $z = uvwxy$, für die die Bedingungen gelten. Da $|vwx| \leq n$, können v und x nur maximal zwei unterschiedliche Buchstaben beinhalten, niemals jedoch a, b und c.

  Damit kann $uv^i wx^i y$ nicht in L sein für ein $i \neq 1$. □

- Für die Sprache $L = \{a^n \mid n \text{ ist eine Quadratzahl}\}$:

  Zunächst wählt man ein Wort x, mit $|x| \geq n$:

  $$z = a^{n^2} \in L$$

Bei jeder Zerlegung $z = uvwxy$ sind nur as in den Teilwörtern v und x, die gepumpt werden.

Betrachtet man die Länge $|vx| = r$, so gilt:

$$|uv^i wx^i y| = n^2 + r(i-1)$$

Insbesondere gilt für das Wort

$$z' = uv^2 wx^2 y = a^s$$
$$|z'| = n^2 + r = s$$

Das hieße, $n^2 + r$ müsste eine Quadratzahl sein damit x' wiederum in L läge. Für r gilt aber die Ausgangsbedinung des Pumping-Lemmas!

$$|vwx| = r + |w| \leq n$$

s kann damit unmöglich eine Quadratzahl sein. □

6: Kontextsensitive Sprachen

Typ-1 = **CSL** $\subset$ **REC** $\subset$ **r. e.**

6.1 Automatenmodell Turingmaschine

Eine Turingmaschine ist ein 7-Tupel

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$$

Z endliche Zustandsmenge

Σ Eingabealphabet $\Sigma \subseteq \Gamma$

Γ Bandalphabet

δ Überführungsfunktion $\delta : Z \times \Gamma \to Z \times \Gamma \times \{L, R, N\}$

z_0 Startzustand, $z_0 \in Z$

$\square$ Leersymbol, $\square \in \Gamma \setminus \Sigma$

E Endzustandsmenge $E \subseteq Z$

Konfiguration Eine Konfiguration k einer Turingmaschine definieren wir als ein Element der Menge

$$k \in \Gamma^\star Z \Gamma^\star$$

Das bedeutet, dass sich die Turingmaschine in der Konfiguration $k = \alpha z \beta$ im Zustand z befindet und der Schreib- Lesekopf das erste Symbol von β liest.

Die akzeptierte Sprache einer Turingmaschine Die akzeptierte Sprache einer Turingmaschine ist definiert als

$$T(M) := \left\{ w \in \Sigma^\star \,\middle|\, \exists \alpha, \beta \in \Gamma^\star, e \in E : z_o w \vdash^\star \alpha e \beta \right\}$$

Wenn $w \notin T(M)$ können drei verschiedene Dinge geschehen:

1. Die Turingmaschine hält an und ist nicht in einem Endzustand.
2. Die Turingmaschine läuft weiter und kommt irgendwann in eine Schleife
   $z_o w \vdash^\star \alpha z \beta \vdash^\star \alpha' z' \beta' \vdash^\star \alpha z \beta \vdash \cdots$
3. Die Turingmaschine rechnet endlos, ohne in eine Schleife zu kommen (Halteproblem).

6.1.1 Einschränkung LBA

Ein linear beschränkter Automat ist eine Turingmaschine, die bei der Verarbeitung der Eingabe niemals den Platz der Eingabe auf dem Arbeitsband verlässt.
Um das zu erreichen muss das letze Symbol der Eingabe markiert sein um den rechten Rand der Eingabe erkennbar zu machen, deshalb definieren wir:

$$\Sigma' = \Sigma \cup \{\hat{a} \mid a \in \Sigma\}$$

Eine nichtdeterministische Turingmaschine nennen wir einen linear beschränkten Automaten, wenn gilt:

$$\forall a_1 a_2 \dots a_{n-1}\hat{a}_n \in \Sigma^+, \alpha, \beta \in \Gamma^\star, z \in Z \text{ mit } z_0 a_1 a_2 \dots a_{n-1}\hat{a}_n \vdash^\star \alpha z \beta : |\alpha\beta| \leq n$$

Für die akzeptierte Sprache gilt dann:

$$T(M) \coloneqq \left\{a_1 \dots a_n \in \Sigma^\star \,\middle|\, \exists \alpha, \beta \in \Gamma^\star, e \in E : z_o a_1 \dots \hat{a}_n \vdash^\star \alpha e \beta\right\}$$

Die Klasse der durch LBAs erkannten Sprachen ist gleich der Typ-1 Sprachen!
Dabei ist die Frage ob nichtdeterministische und deterministische LBAs gleich mächtig sind noch offen.

6.2 Sätze zu CSL

- Die Klasse der kontextsensitiven Sprachen ist abgeschlossen unter allen Operationen!
- Für die kontextsensitiven Sprachen und alle höheren Sprachklassen gibt es die sog. Epsilon-Sonderregel, die oft verwendet wird um trotz der Eigenschaft nichtverkürzend ein Wort der Länge Null zu erreichen:

  $$(S, \epsilon) \in P \text{ aber } \forall (u, v) \in P : S \notin v$$

6.2.1 Algorithmus zur Entscheidbarkeit des Wortproblems

Die Funktion $\text{Abl}_n(X)$ wird iteriert angewendet, bis sich entweder X nicht mehr ändert ($w \notin L(G)$) oder das gesuchte Wort w in X enthalten ist ($w \in L(G)$).
Dabei ist n die Länge des gesuchten Worts w, also $|w|$.
Die Funktion $\text{Abl}_n(X)$ ist für eine Grammatik G wie folgt definiert:

$$\text{Abl}_n(X) \coloneqq X \cup \left\{w \in (V \cup \Sigma)^\star \,\middle|\, |w| \leq n \wedge \exists y \in X : y \Rightarrow_G w\right\}$$

6.3 Kuroda-Normalform

Eine Typ-1 Grammatik (V, Σ, P, S) ist in Kuroda-Normalform wenn alle Regeln von einem der vier Typen sind:

- $A \to a$
- $A \to B$
- $A \to BC$
- $AB \to CD$

Wobei $A, B, C, D \in V$ und $a \in \Sigma$ ist.
Zu jeder Typ-1 Grammatik existiert eine äquivalente Grammatik in Kuroda-Normalform

6.3.1 Umformungsalgorithmus

1. **Pseudoterminale:** Zunächst alle Terminalsymbole, die nicht alleine auf einer rechten Seite stehen durch Pseudoterminale ersetzen.

   Jetzt stören noch die Regeln $A_1 \dots A_n \to B_1 \dots B_m$ mit $1 \leq n \leq m, m > 2$

2. Für Regeln der Form $A \to B_1 \dots B_m, m > 2$ können wir die gleiche Methode wie bei den Typ-2 Normalformen anwenden (Siehe Unterabschnitt 5.3.1).

3. Regeln der Form $A_1 \dots A_n \to B_1 \dots B_m$ mit $2 \leq n \leq m, m > 2$ ersetzen wir durch:

   $$A_1 A_2 \to B_1 D_1$$
   $$D_1 A_2 A_1 \dots A_n \to B_2 \dots B_m$$

6.4 Beispiele

- $L_1 = \{a^n b^n c^n \mid n \geq 1\}$ (drei und mehr gleiche Exponenten)
- $L_2 = \left\{ a^{n^2} \,\middle|\, n \in N^+ \right\}$

3

Kapitel 3: Berechenbarkeitstheorie und Komplexität

1: Rekursiv aufzählbare Sprachen

Typ-0 = **r**. **e**.

Eine Turingmaschine M *akzeptiert* die Sprache L, wenn sie nach endlicher Zeit hält. Bei Eingaben, die nicht zu L gehören, rechnet sie unendlich lang weiter (dieses Verhalten führt später auf das Halteproblem). Solche Sprachen L gehören dann zu **r**. **e**., sie sind *semi-entscheidbar*.

1.1 Sätze zu r. e.

- Die Klasse der Typ-0 Sprachen ist abgeschlossen unter Sternopertaion, Vereinigung, Schnitt und Konkatenation.
- Die Klasse der durch Turingmaschinen erkennbaren/akzeptierten Sprachen ist gleich der Klasse der rekursiv aufzählbaren.

2: Entscheidbare Sprachen

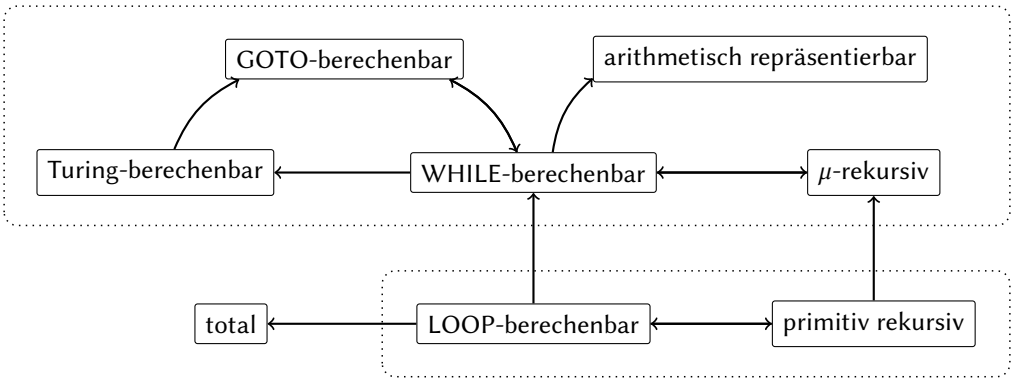
REC $\subset$ **r. e.**

Eine Turingmaschine *entscheidet* die Sprache L, wenn sie für jede Eingabe nach endlicher Zeit stoppt und JA oder NEIN ausgibt, je nachdem ob $w \in L$ gilt oder nicht. Diese Sprachen L gehören dann zur Klasse der entscheidbaren Sprachen **REC** (rekursive Sprachen).

3: Berechenbarkeitstheorie

ACHTUNG: Dieser Abschnitt ist noch nicht abgeschlossen und enthält möglicherweise Fehler!

3.1 Berechenbarkeiten



3.1.1 LOOP-Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt LOOP-berechenbar, falls es ein LOOP-Program P gibt, das gestartet auf der Eingabe $n_1, n_2, \ldots, n_k$ in den Variablen $x_1, x_2, \ldots, x_n$ nach endlich vielen Schritten hält und die Variable x_0 den Wert $f(n_1, \ldots, n_k)$ beinhaltet.

Erlaubte Basisanweisungen

- $x_i := x_j + c$ bzw. $x_i := x_j - c$ mit $c \in \mathbb{N}$
- `LOOP` x_i `DO` P `END`
- Hintereinanderausführung von LOOP-Programmen

Simulierbare Makros

- Wertzuweisungen $x_i := x_j$ und $x_i := c$
- `IF` $x_i > c$ `THEN` P `END`
- Alle üblichen arithmetischen Operationen, auch Modulrechnung

3.1.2 WHILE-Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt WHILE-berechenbar, falls es ein WHILE-Program P gibt, das gestartet auf der Eingabe $n_1, n_2, \ldots, n_k$ in den Variablen $x_1, x_2, \ldots, x_n$ nach endlich vielen Schritten hält (falls das Ergebnis definiert ist) und die Variable x_0 den Wert $f(n_1, \ldots, n_k)$ beinhaltet. Ist $f(n_1, \ldots, n_k)$ undefiniert, so hält P nicht.

1. Jedes GOTO-Programm kann durch ein WHILE-Programm mit nur einer einzigen WHILE-Schleife simuliert werden.
2. Jede WHILE-Berechenbare Funktion kann durch ein WHILE-Programm mit nur einer einzigen WHILE-Schleife berechnet werden (Kleene'sches Normalform-Theorem)

Erlaubte Anweisungen

- Alle Anweisungen von LOOP-Programmen
- `WHILE` $x_i \neq 0$ `DO` P `END`

3.1.3 GOTO-Berechenbarkeit

Erlaubte Anweisungen

- Berechnungen und Zuweisungen: $x_i := x_j \pm c$
- Marken: M_i
- `GOTO` M_i
- `HALT`
- `IF` $x_i = c$ `THEN GOTO` M_j

Simulierbare Makros

- Die `WHILE`-Schleife

3.1.4 Turing-Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt Turing-berechenbar, falls eine deterministische Turingmaschine existiert, die $f(n_1, \ldots, n_k) = m$ berechnet indem, sie gestartet auf dem k-Tupel $(n_1, \ldots, n_k)$ nach endlich vielen Berechnungsschritten einen Endzustand erreicht und dann m auf dem Band steht. Falls das Ergebnis für die Eingabe undefiniert ist, terminiert die Maschine nie.

3.1.5 Primitive Rekursion

Eine Funktion ist genau dann primitiv rekursiv, wenn sie LOOP-berechenbar ist.

- Konstante Funktionen sind primitiv rekursiv
- Projektionen sind primitiv rekursiv
- Die Nachfolgerfunktion $s : \mathbb{N} \to \mathbb{N}, n \mapsto n + 1$ ist primitiv rekursiv
- Verkettungen von primitiv rekursiven Funktionen sind primitiv rekursiv
- Funktionen, die durch primitive Rekursion aus primitiv rekursiven Funktionen entstehen, sind primitiv rekursiv:

$$f(0, x_1, \ldots, x_k) = g(x_1, \ldots, x_k) \qquad \text{(Startwert)}$$
$$f(n+1, x_1, \ldots, x_k) = h(f(n, x_1, \ldots, x_k), n, x_1, \ldots, x_k) \qquad \text{(Rekursionsschritt)}$$

Das heißt, g, h primitiv rekursiv $\Rightarrow f$ primitiv rekursiv.

3.1.6 μ-Rekursion

$$\mu f(x_1, \ldots, x_k) = \min \{n \in \mathbb{N} \mid f(n, x_1, \ldots, x_k) = 0 \wedge \forall m < n : f(m, x_1, \ldots, x_k) > 0\}$$

Der μ-Operator liefert den kleinsten Eingabewert n der ersten Variable, bei der die Funktion 0 ausgibt. Dabei muss insbesondere f für alle Werte kleiner als n definiert sein, da sonst eine Berechnung nicht möglich wäre.

Satz von Kleene: Eine μ-rekursive Funktion $f : \mathbb{N}^n \to \mathbb{N}$ lässt sich immer durch zwei $n+1$-stellige primitiv rekursive Funktionen darstellen:

$$f(x_1, \ldots, x_n) = p(x_1, \ldots, x_n, \mu q(x_1, \ldots, x_n))$$

3.1.7 Arithmetische Repräsentierbarkeit

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt arithmetisch repräsentierbar, falls es eine $(k+1)$ stellige arithmetische Formel F gibt, so dass gilt

$$F(n_1, \ldots, n_k, m) \Leftrightarrow f(n_1, \ldots, n_k) = m$$

Hierfür besteht eine arithmetische Formel zunächst aus Termen:

- Alle $n \in \mathbb{N}$ sind Terme
- Für alle $i \in \mathbb{N}$ ist x_i ein Term
- Verknüpfung zweier Terme mit $+$ oder $*$ sind ebenfalls Terme

Aus den Termen werden Formeln gebildet

- Gleichheit zweier Terme $(t_1 = t_2)$ ist eine Formel
- Für F, G Formeln sind auch $\neg F, (F \wedge G)$ und $(F \vee G)$ Formeln
- Für F eine Formel und $i \in \mathbb{N}$ sind auch $\forall x_i F$ und $\exists x_i F$ Formeln

Gödel'scher Unvollständigkeitssatz Jedes Beweissystem, das nur wahre Formeln der Arithmetik beweist muss unvollständig sein.

4: Entscheidbarkeitstheorie

ACHTUNG: Dieser Abschnitt ist noch nicht abgeschlossen und enthält möglicherweise Fehler!
Die Berechenbarkeitstheorie spielt sich innerhalb der Entscheidbarkeitstheorie ab. Wenn eine Sprache entscheidbar ist, macht es Sinn ihre Komplexität zu bestimmen. Andersherum heißt das, dass jede Sprache, die in einer der Komplexitätsklassen enthalten ist entscheidbar ist (also insbesondere semi- und co-semi-entscheidbar).

4.1 Definitionen

4.1.1 Charakteristische Funktionen

Für jede Menge A existiert die sogenannte *charakteristische Funktion* $\chi_A(w)$. Diese Funktion entscheidet für jedes Wort w aus einer festgelegten Grundmenge, ob $w \in A$ gilt.

$$\chi_A(w) = \begin{cases} 1, \text{ falls } w \in A \\ 0, \text{ sonst} \end{cases}$$

Ebenso lässt sich die semi-charakteristische Funktion $\chi'_A(w)$ definieren

$$\chi'_A(w) = \begin{cases} 1, \text{ falls } w \in A \\ \text{undefiniert, sonst} \end{cases}$$

Eine Menge heißt entscheidbar, wenn ihre zugehörige charakteristische Funktion berechenbar ist. Eine Menge heißt semi-entscheidbar, falls die semi-charakteristische Funktion berechenbar ist. Das Komplement einer Mengen, deren semi-charakteristische Funktion berechenbar ist, heißt co-semi-entscheidbar. Ist eine Menge entscheidbar, so ist es ihr Komplement trivialerweise auch.

4.1.2 Rekursive Aufzählbarkeit

Eine Sprache A ist rekursiv aufzählbar, wenn eine totale, berechenbare Funktion $c : \mathbb{N} \to \Sigma^\star$ existiert sodass $A = \{c(n) \mid n \in \mathbb{N}\}$ gilt. Rekursive Aufzählbarkeit ist äquivalent zu Semi-Entscheidbarkeit.

4.1.3 Entscheidbarkeitsprobleme

Hierbei seien die Gödelisierungen der Maschinen in einer geeigneten Art und Weise kodiert. Beispielsweise $w \in \{0,1\}^\star$ für eine Maschine M_w.

Spezielles Halteproblem	$K = \{w \mid M_w \text{ hält auf Eingabe } w\}$	semi-entscheidbar
Allgemeines Halteproblem	$H = \{w\#x \mid M_w \text{ hält auf Eingabe } x\}$	semi-entscheidbar
Halteproblem auf leerem Band	$H_0 = \{w \mid M_w \text{ hält auf Eingabe } \epsilon\}$	semi-entscheidbar
PCP	Postsches Korrespondenzproblem	semi-entscheidbar
MPCP	Für alle Lösungen gilt $i_1 = 1$	semi-entscheidbar
WA	Menge aller wahren arithmetischen Formeln	unentscheidbar
$\overline{\text{WA}}$	Menge aller falschen arithm. Formeln	unentscheidbar

Satz von Rice Sei $\mathcal{R}$ die Klasse der Turing-berechenbaren Funktionen und $\mathcal{S}$ eine nichttriviale Teilmenge von $\mathcal{R}$. Dann ist die Menge $C(\mathcal{S}) = \{w \mid M_w \text{ berechnet eine Funktion aus } \mathcal{S}\}$ unentscheidbar.

Probleme in der Theorie der formalen Sprachen

- Für deterministisch kontextfreie Sprachen $L, K \in \mathbf{DCFL}$ sind die Fragen $L \cap K = \emptyset$, $|L \cap K| < \infty$, $L \cap K \in \mathbf{CFL}$ sowie $L \subseteq K$ unentscheidbar.
- Für eine kontextfreie Grammatik sind die Frage nach Mehrdeutigkeit, $\overline{L(G)} \in \mathbf{CFL}$, $L(G) \in \mathbf{REG}$ und $L(G) \in \mathbf{DCFL}$ alle unentscheidbar.
- Für eine kontextsensitive-Grammatik ist die Leerheit sowie die Endlichkeit der erzeugten Sprache unentscheidbar.

4.2 Reduktion von Problemen

Um die prinzipielle Lösbarkeit zweier Probleme zu betrachten, verwendet man Reduktionen:
Seien $A \subseteq \Sigma^\star$ und $B \subseteq \Gamma^\star$ zwei Probleme. Kann man eine *totale und berechenbare Funktion* $f : \Sigma^\star \to \Gamma^\star$ finden, so dass gilt

$$x \in A \Leftrightarrow f(x) \in B,$$

so sagt man, A ist auf B (many-one-)reduzierbar. Man schreibt dann auch $A \leq B$.

1. $A \leq B$ und B entscheidbar $\Rightarrow$ A entscheidbar.
2. $A \leq B$ und B semi-entscheidbar $\Rightarrow$ A semi-entscheidbar.

Bei der Reduktion übertragen sich immer fehlende Eigenschaften von links nach rechts, d.h. reduziert man ein semi-entscheidbares, aber nicht co-semi-entscheidbares Problem A auf ein Problem B (d.h. $A \leq B$), so ist B ebenfalls nicht co-semi-entscheidbar. Über die Semi-Entscheidbarkeit von B wird keine Aussage gemacht.
Reduziert man allerdings ein unbekanntes Problem A auf ein semi-entscheidbares Problem B (d.h. $A \leq B$), so hat man die Semi-Entscheidbarkeit von A gezeigt. Es übertragen sich also „positive Eigenschaften“ von rechts nach links.
Letzterer Sachverhalt wird klar, wenn man sich überlegt, dass man mit der Reduktion eine Funktion gefunden hat, die eine Fragestellung aus A in eine aus B überträgt. Man kann also das Entscheidungsproblem für A zunächst in B übersetzen und schließlich mit dem Entscheidungsalgorithmus für B entscheiden.

5: Komplexität

ACHTUNG: Dieser Abschnitt ist noch nicht abgeschlossen und enthält möglicherweise Fehler!
Die Frage in der Komplexitätstheorie ist, wie viel Aufwand benötigt wird, ein Problem zu lösen. Die theoretische Lösbarkeit wird in der Entscheidbarkeits- bzw. Berechenbarkeitstheorie behandelt.
Die Komplexitätstheorie befasst sich mit oberen und unteren Schranken an den Ressourcenaufwand zur Problemlösung. Wichtig sind hierbei auch inhärente Komplexitäten, also eine Beschränkung nach oben und unten.

5.1 Komplexitätsklassen

Diese Komplexitätsklassen liegen alle in der Menge der entscheidbaren Sprachen. Alle folgenden Klassen sind also Teilmengen von **REC**.
Für alle Komplexitätsklassen C ist zu unterscheiden:

$$\operatorname{co} C = \left\{ L \in \Sigma^\star \,\middle|\, \overline{L} \in C \right\} \qquad \overline{C} = \left\{ L \in \Sigma^\star \,\middle|\, L \notin C \right\}$$

5.1.1 Zeitklassen

Die Funktion $\text{time}_M : \Sigma^\star \to \mathbb{N}$ ordnet einer Eingabe die Anzahl Schritte zu, die eine deterministische Maschine M auf ihr ausführt.
Die Zeitklasse $\text{DTIME}(f(n))$ enthält alle Probleme, die sich durch eine deterministische Turingmaschine in einem Zeitaufwand kleiner als f lösen lassen. Das heißt, f ist eine obere Schranke für die Komplexität der Probleme in $\text{TIME}(f)$.

Die Funktion $\text{ntime}_M : \Sigma^\star \to \mathbb{N}$ ordnet einer Eingabe die Länge des kürzesten akzeptierenden Pfades der nichtdeterministischen Maschine M zu. Falls die Maschine nicht akzeptiert, ist $\text{ntime}_M(w) = 0$.
Die Zeitklasse $\text{NTIME}(f(n))$ enthält alle Probleme, die sich durch eine nichtdeterministische Turingmaschine in einem Zeitaufwand kleiner als f lösen lassen.

$$\mathbf{P} = \bigcup_{k \geq 1} \text{DTIME}(n^k)$$
$$\mathbf{NP} = \bigcup_{k \geq 1} \text{NTIME}(n^k)$$

5.1.2 Platzklassen

Analog zu den Zeitklassen sind Platzklassen definiert, die den maximal belegten Platz für eine Berechnung beschränken. Die Funktion $\text{space}_M : \Sigma^\star \to \mathbb{N}$ ordnet dabei jeder Eingabe auf einer deterministischen Turingmaschine die Anzahl der maximal verwendeten Bandelemente (auf einem Arbeitsband) zu.
So ist die Platzklasse $\text{SPACE}(f(n))$ definiert als die Klasse aller Probleme für die f eine Platzbeschränkung für alle Eingaben ist.
Analog wie bei Zeitklassen die Unterscheidung deterministische-nichtdeterministische Platzklassen.

$$\begin{aligned}
\mathbf{L} &= \mathrm{DSPACE}(\log n) \\
\mathbf{NL} &= \mathrm{NSPACE}(\log n) \\
\mathbf{PSPACE} &= \bigcup_{k\geq 1} \mathrm{DSPACE}(n^k) = \bigcup_{k\geq 1} \mathrm{NSPACE}(n^k)
\end{aligned}$$

5.2 Beziehungen zwischen den Klassen

Für ein beliebiges $f : \mathbb{N}_0 \to \mathbb{N}_0$, falls nicht näher angegeben

5.2.1 Äquivalenzen von Klassen

- In den Platzklassen spielt die O-Notation keine Rolle, Konstanten können wegen Bandreduktion und Bandkompression vernachlässigt werden

  $$\mathrm{DSPACE}(O(f)) = \mathrm{DSPACE}_{\text{1-Band}}(f)$$
  $$\mathrm{NSPACE}(O(f)) = \mathrm{NSPACE}_{\text{1-Band}}(f)$$

- In nichtdeterministischen Zeitklassen spielt die O-Notation ebenfalls keine Rolle

  $$\mathrm{NTIME}(O(f)) = \mathrm{NTIME}(f)$$

- Bei deterministischen Zeitklassen gilt i. A. $\mathrm{DTIME}(O(f)) \neq \mathrm{DTIME}(f)$, nur für größer als lineare Funktionen gilt Gleichheit d.h.

  $$\mathrm{DTIME}(O(f)) = \mathrm{DTIME}(f) \text{ mit } f(n) \geq (1+\epsilon)\cdot n \text{ für ein } \epsilon > 0$$

- **Satz von Immerman und Szelepcsenyi**: Falls $f \in \Omega(\log(n))$, gilt:

  $$\mathrm{NSPACE}(f) = \mathrm{coNSPACE}(f)$$

- Alle deterministischen Zeit- und Platzklassen sind gegen Komplement abgeschlossen:

  $$\mathrm{DSPACE}(f) = \mathrm{coDSPACE}(f)$$
  $$\mathrm{DTIME}(f) = \mathrm{coDTIME}(f)$$

5.2.2 Hierarchien und Teilmengen

- Für alle $f(n) \geq n$ gilt für die Zeitklassen

  $$\mathrm{DTIME}(f) \subseteq \mathrm{NTIME}(f) \subseteq \mathrm{DSPACE}(f)$$

- Und für alle $f(n) \geq \log n$ gilt

  $$\mathrm{DSPACE}(f) \subseteq \mathrm{NSPACE}(f) \subseteq \mathrm{DTIME}(2^{O(f)})$$

- **Satz von Savitch**: Sei $s \in \Omega(\log(n))$, dann gilt

  $$\mathrm{NSPACE}(s) \subseteq \mathrm{DSPACE}(s^2)$$

- **Platzhierarchiesatz**: Sei $s_1 \notin \Omega(s_2)$ und $s_2 \in \Omega(\log(n))$, dann gilt

  $$\text{DSPACE}(s_2) \setminus \text{DSPACE}(s_1) \neq \emptyset$$
  $$\Rightarrow \text{DSPACE}(s_1) \subsetneq \text{DSPACE}(s_2)$$

- **Zeithierarchiesatz**: Sei $t_1 \log(t_1) \notin \Omega(t_2)$ und $t_2 \in \Omega(n \log(n))$, dann gilt

  $$\text{DTIME}(t_2) \setminus \text{DTIME}(t_1) \neq \emptyset$$
  $$\Rightarrow \text{DTIME}(t_1) \subsetneq \text{DTIME}(t_2)$$

- **Satz von Hennie und Stearns**: Falls $\epsilon > 0$, $f(n) \geq (1+\epsilon)n$, dann gilt

  $$\text{DTIME}(f) \subseteq \text{DTIME}_{\text{2-Band}}(f \log f)$$

5.2.3 Weitere Sätze

- **Lückensatz von Borodin**: Für jede totale berechenbare Funktion $r(n) \geq n$ existiert effektiv eine totale berechenbare Funktion $s(n) \geq n+1$ mit

  $$\text{DTIME}(s(n)) = \text{DTIME}(r(s(n)))$$

5.3 Polynomialzeitreduktion

Reduktionen können nicht nur verwendet werden um Entscheidbarkeiten festzustellen, sondern helfen genauso bei der Einstufung von Problemen in Komplexitätsklassen. So kann man durch Beschränken der Reduktionsfunktion beispielsweise eine Lösbarkeit in einer bestimmen Zeit zeigen.
Ähnlich wie in der Berechenbarkeitstheorie kann man zur Analyse der Komplexität von Problemen Reduktionen zwischen diesen entwickeln. Hier ist das Ergebnis allerdings sogar eine Beschränkung des Aufwands nach oben oder unten.
Allerdings ist die bereits vorgestellte many-one-Reduktion (ohne Beschränkungen) zu grob um sinnvolle Reduktionen zwischen Problemen mit Zeitbeschränkung durchzuführen. (Ein Problem, das in Polynomialzeit lösbar ist, könnte durch so eine Reduktionsfunktion gelöst werden. So kann man eigentlich alle berechenbaren Probleme aufeinander reduzieren, diese Aussage ist jedoch für eine Komplexitätsbetrachtung uninteressant.)
Deshalb gibt es die sog. Polynomialzeitreduktion. Hier ist eine Beschränkung an die Reduktionsfunktion gesetzt, nämlich muss diese zusätzlich in Polynomialzeit berechenbar sein. Man schreibt $A \leq_p B$.

1. $A \leq_p B \wedge B \in \mathbf{P} \Rightarrow A \in \mathbf{P}$
2. $A \leq_p B \wedge B \in \mathbf{NP} \Rightarrow A \in \mathbf{NP}$

5.3.1 Härte und Vollständigkeit

NP-Härte Eine Sprache ist **NP**-hart falls sich alle Probleme aus **NP** in Polynomialzeit darauf reduzieren lassen. Das heißt dieses Problem ist mindestens so schwierig zu lösen wie ein Problem aus **NP**. Es gilt: A **NP**-hart und $A \leq_p B \Rightarrow B$ **NP**-hart.

NP-Vollständigkeit Ist eine Sprache **NP**-hart und selbst in **NP** enthalten, so nennt man sie **NP**-vollständig. Diese Sprachen gehören zu den schwierigsten Sprachen aus **NP**. Kann man von einer **NP**-vollständigen Sprache zeigen, dass sie sogar in **P** liegt, so hat man **P**=**NP** gezeigt.
Einige **NP**-vollständige Probleme:

SAT	$\{w \mid w$ kodiert eine erfüllbare Formel$\}$
3KNF-SAT	Ist eine Formel in KNF mit max. 3 Literalen pro Klausel erfüllbar?
CLIQUE	Enthält ein Graph eine Clique der Größe k?
FÄRBBARKEIT	Gibt es eine Knotenfärbung mit k Farben?

PSPACE-Vollständigkeit
Die quantifizierten booleschen Formeln (QBF) stellen ein bekanntes **PSPACE**-vollständiges Problem dar

$$\textbf{QBF} = \{F \mid F \text{ ist eine geschlossene QBF-Formel, die sich zu TRUE auswerten lässt }\}$$

5.3.2 Logspace-Reduktion

Logspace-Transducer Ein Logspace-Transducer ist eine deterministische Turingmaschine mit einem read-only Eingabeband und einem logarithmisch beschränkten Arbeitsband, sowie einem write-only Ausgabeband auf dem der Schreib-/ Lesekopf nicht nach links bewegt werden kann.
Eine Funktion heißt logspace-berechenbar, wenn ein Logspace-Transducer existiert, der sie berechnet.
Auf Basis dieser Definitionen gibt es nun als Verfeinerung der Polynomialzeitreduktion die Logspace-Reduktion:
Seien $A, B \subseteq \Sigma^\star$ Sprachen. A heißt auf B *logspace-reduzierbar*, falls eine logspace-berechenbare Funktion $f : \Sigma^\star \to \Sigma^\star$ existiert, so dass

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

Man schreibt dann $A \leq_{\log} B$.
So kann man nun Begriffe wie **NL**- bzw. **P**-Härte und -Vollständigkeit sinnvoll (und analog zur **NP**-Vollständigkeit) definieren.
Einige Probleme, die **NL**-vollständig bezüglich Logspace-Reduktion sind:

GAP	Grapherreichbarkeitsproblem
2-SAT	erfüllbare boole'sche Formeln in 2KNF-Darstellung
2-NSAT	nicht erfüllbare Formeln in 2KNF-Darstellung

Einige Probleme, die **P**-vollständig bezüglich Logspace-Reduktion sind:

CFE	$L_{cfe} = \{w \mid w \text{ ist Kodierung einer Typ-2 Grammatik } G \text{ mit } L(G) = \varnothing\}$
CVP	Circuit Value Problem (Ausgabe eines Schaltnetzes)
MCVP	Wie **CVP**, allerdings ohne Inverter (Negationen)

5.3.3 Weitere Reduktionen

Man kann beliebige Reduktionen definieren indem man bestimmte Einschränkungen an die Reduktionsfunktion vorgibt. Bekannte Reduktionen sind die Logplatz-Reduktion, die Polynomialzeitreduktion, die allgemeine many-one-Reduktion sowie die Turing-Reduktion $\leq_T$. Diese sind jeweils Verfeinerungen voneinander

$$A \leq_{\log} B \Rightarrow A \leq_p B \Rightarrow A \leq B \Rightarrow A \leq_T B.$$

(Bei der Turing-Reduktion wird die Übersetzungsfunktion mit einer Orakel-Turingmaschine durchgeführt.)

5.4 Translationtechnik

Die Translationssätze werden verwendet, Separationen von größeren zu kleineren Klassen bzw. Gleichheiten oder Inklusionen von kleineren zu größeren Klassen zu übertragen. Die durch Padding mit f aufgebläte Sprache ist $Pad_f(L) := \left\{w\$^{f(|w|)-|w|} \,\middle|\, w \in L\right\}$, wobei $\{\$\} \cap \Sigma = \varnothing$.

1. Für zwei Funktionen $f(n), g(n) \geq n$ gilt der **Translationssatz für Zeitklassen**:

$$Pad_f(L) \in \text{DTIME}(O(g)) \Leftrightarrow L \in \text{DTIME}(O(g \circ f))$$
$$Pad_f(L) \in \text{NTIME}(O(g)) \Leftrightarrow L \in \text{NTIME}(O(g \circ f))$$

2. Und analog für $g \in \Omega(\log)$ und $f(n) \geq n$ der **Translationssatz für Platzklassen**:

$$Pad_f(L) \in \text{DSPACE}(\mathcal{O}(g)) \Leftrightarrow L \in \text{DSPACE}(\mathcal{O}(g \circ f))$$
$$Pad_f(L) \in \text{NSPACE}(\mathcal{O}(g)) \Leftrightarrow L \in \text{NSPACE}(\mathcal{O}(g \circ f))$$