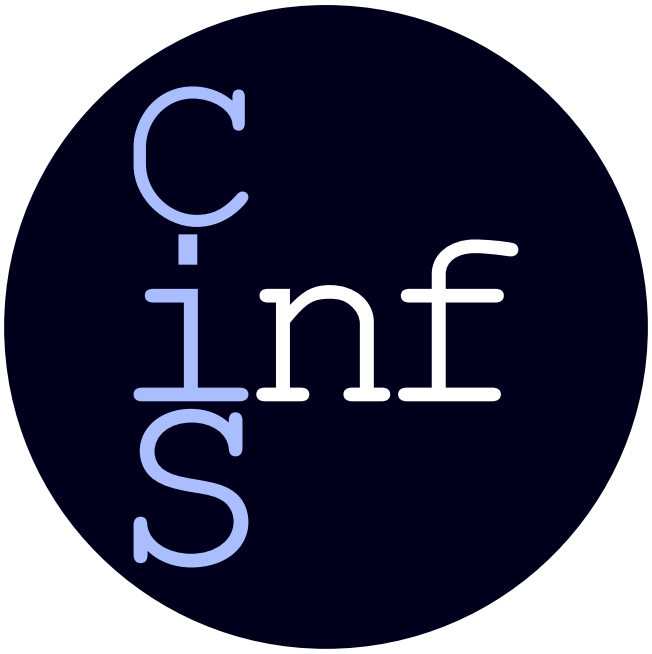


Einführung in die theoretische Informatik

Cheatsheet SoSe2020

Franke, Kniep

12. Mai 2021

Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Inhaltsverzeichnis

0. Disclaimer

Hello there!
Dies ist eine von Studierenden erstellte Zusammenfassung der Inhalte des Moduls *Einführung in die Theoretische Informatik*, wie angeboten an der Exzellenzuniversität Hamburg im Sommersemester des Jahres 2020.
Wir übernehmen weder Verantwortung für die Korrektheit, noch die Vollständigkeit der Inhalte dieses Dokumentes, es stellt keinen Ersatz zur Vorlesung oder Fachliteratur des Moduls dar, wir haben aber immer Recht.
Die aktuellste Version dieses Dokumentes ist über folgende Seite[1] erreichbar.

Dieses Dokument verwendet als Quellen, neben dem berühmten CiS-Hivemind:

- Introduction to the Theory of Computation (Third Edition), Michael Sipser
- Logik für Informatiker (5. Auflage), Uwe Schöning
- Folien (2020), Dr. Daniel Moldt
- ETI[2] Repetitoriumsfolien (2020), Dimitri Popov
- FGI[3] 1 Tutoriumsfolien (2020), Marcus Soll

Für die Automatenabbildungen (Tikz-Figures) wurde die von uns modifizierte Seite[4], welche unter anderem den Export nach LaTeX 2_ε ermöglicht, genutzt.

Dieses Dokument darf zur Selbstbildung und Persönlichkeitsentwicklung genutzt werden, bei Vervielfältigung, Bearbeitung, Übersetzung oder Mikroverfilmung der untenstehenden Punkte ohne eindeutige Zustimmung unsererseits[5] gilt allerdings trotzdem:

"Es gibt da so rechtliche Dinge." - Daniel Moldt, 2020

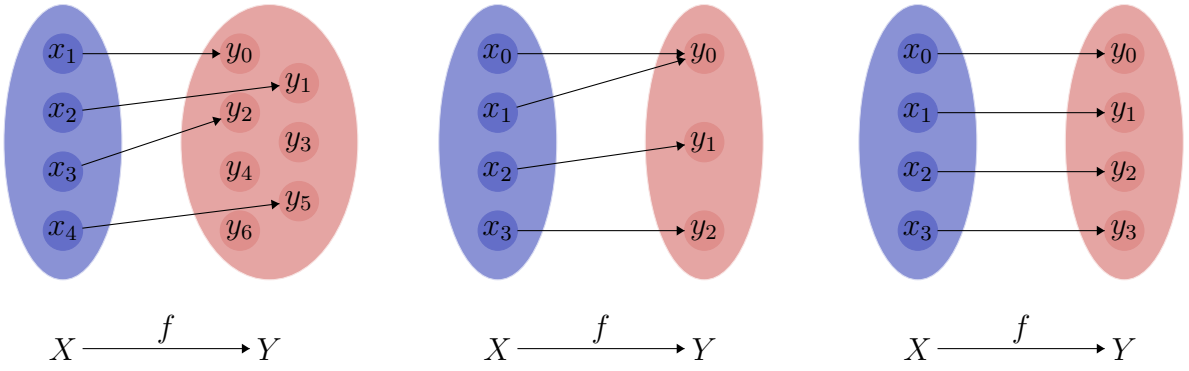
[1] https://cis-exzellenz.de/files/eti-cheatsheet
[2] Einführung in die Theoretische Informatik
[3] Formale Grundlagen der Informatik
[4] https://cis-exzellenz.de/fsm
[5] kontakt@traumweh.tk

1. Vorwissen

1.1 Funktionen und Mengen

Injektivität	Jedes Bild hat maximal ein Urbild
	$\forall b \in f(A) \quad \exists! a \in A : f(a) = b$
	$\forall x, y \in A : f(x) = f(y) \Rightarrow x = y$
Surjektivität	Jedes Bild hat mindestens ein Urbild
	$\forall b \in B,\ a \in A : f(a) = b$
Bijektivität	Injektivität + Surjektivität

Abbildung 1.1: $X \longrightarrow Y, \quad f(X) = Y$



(a) Injektivität (b) Surjektivität (c) Bijektivität

1.1.1 Abzählbarkeit

Eine Menge A ist abzählbar, wenn sie entweder endlich ist oder die gleiche Mächtigkeit wie $\mathbb{N}$ hat. A hat die selbe Mächtigkeit wie $\mathbb{N}$, wenn es eine Bijektion f gibt mit: $f : A \to \mathbb{N}$.

1.2 Graph

$$G = (V, E)$$

Symbol	Erklärung
G	**Graph** G
V	**Menge** der Knoten
E	**Menge** aller Knotenpaar**tupel**

E darf keinen Knoten enthalten, der nicht in V vorkommt und für jeden Knoten in V muss mindestens ein **Knotenpaar** in E existieren.

Ein Graph ist **zusammenhängend** wenn jedes Knotenpaar durch einen Pfad verbunden ist. Dabei ist es egal, ob der Pfad gerichtet oder ungerichtet ist.
Ein **gerichteter Graph** kann zudem **streng zusammenhängend** sein, wenn (mindestens) ein gerichteter Pfad alle Knotenpaare verbindet.
Beim E eines **ungerichteten** Graphen ist die Reihenfolge in den **Knotenpaartupeln** nicht relevant, da diese nicht gerichtet sind.

Ein **Subgraph** $S = (V_S, E_S)$ des Graphen G besitzt ein $V_S \subseteq V$, wobei auch hier gilt, dass alle Knoten in V_S durch mindestens ein Knotenpaar in $E_S \subseteq E$ verbunden sein müssen und es darf in E_S kein Knoten vorkommen, der nicht in V_S vorkommt.

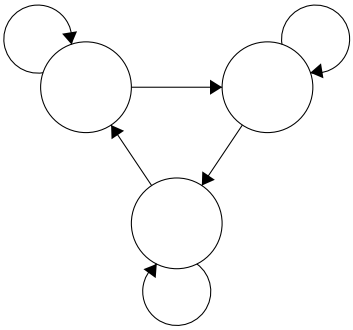
1.2.1 Pfade

Ein **Pfad** ist eine **Sequenz** an **Knoten**, welche durch **Kanten** verbunden sind. Dabei wird in einem **einfachen Pfad** kein Knoten mehr als einmal durchschritten.
Ein Pfad von x nach y wird geschrieben als Tupel (x, y).

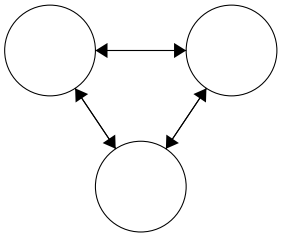
1.2.2 Relationen

Reflexivität	$\forall x \in A : xRx$	Jeder Knoten x hat ein (x, x)
Symmetrie	$\forall x, y : xRy \to yRx$	Zu jedem (x, y) gibt es ein (y, x)
Asymmetrie	$\forall x, y : xRy \to \neg(yRx)$	Zu keinem (x, y) gibt es ein (y, x)
Antisymmetrie	$\forall x, y : xRy \land yRx \to x = y$	Reflexive Symmetrie
Transitivität	$\forall x, y, z : xRy \land yRz \to xRz$	Zu jedem $(x, y), (y, z)$ gibt es ein (x, z)

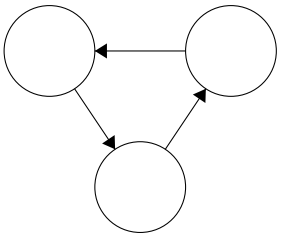
Beispiele von Relationen:



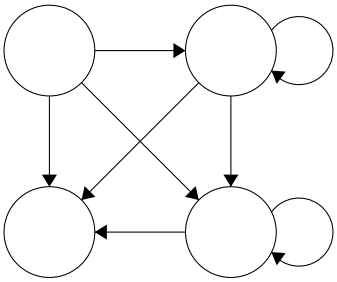
(a) Reflexiv, Antisymmetrisch



(b) Symmetrisch



(a) Asymmetrisch



(b) Transitiv

2. Chomsky Hierarchie

Sprachfamilie			Automaten	Grammatik		$\cup$	$\cap$	$\overline{\ }$
Endliche Mengen						+	+	-
Regulär	$\mathcal{R}eg$	$\mathcal{L}_3$	FSA	Typ-3	rechtslinear	+	+	+
Determ. Kontextfrei	$\mathcal{DC}f$		DPDA		$LR(k),\ k \geq 1$	-	-	+
Kontextfrei	$\mathcal{C}f$	$\mathcal{L}_2$	PDA	Typ-2	kontextfrei	+	-	-
Kontextsensitiv	$\mathcal{C}s$	$\mathcal{L}_1$	NLBA	Typ-1	monoton	+	+	+
Entscheidbar[1]	$\mathcal{R}ec$					+	+	+
Erkennbar[2]	$\mathcal{R}e$	$\mathcal{L}_0$	TM	Typ-0		+	+	-
Abzählbare Mengen								

[1] recursive language
[2] recursively enumerable language

3. Finite State Automata (FSA)

Ein FSA kann entweder deterministisch (DFA) oder nichtdeterministisch (NFA) sein, diese sind gleich **mächtig** und sogar **äquivalent** zu einander (s. Abschnitt 3.3), da sie in die jeweils andere Form überführt werden können.
Zwei FSA M_1 und M_2 sind äquivalent, genau dann wenn sie die gleiche Sprache akzeptieren, also wenn gilt: $L(M_1) = L(M_2)$ (s. Abschnitt 6.1.3)
Sei M ein FSA so bezeichnet $L(M)$ die von M **erkannte Sprache** (die Menge von Wörtern die M erkennt). Die Sprache die ein FSA erkennt ist eine reguläre Sprache (s. Abschnitt 6.3).
Ein FSA ist **vollständig**, wenn jede Eingabe gelesen werden kann bzw. zu jedem $(q, x) \in Q \times \Sigma$ ein $q' = \delta(q, x)$ existiert.

3.1 Deterministic Finite Automaton (DFA)

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

Symbol	Erklärung
A	**DFA**
Q	Endliche **Menge** aller Zustände
Σ	Eingabealphabet
δ	Übergangsfunktion $Q \times \Sigma \to Q$
q_0	Startzustand
$F \subseteq Q$	**Menge** der Endzustände

Ein DFA ist **initial zusammenhängend**, wenn jeder Zustand vom Startzustand aus erreichbar ist. Außerdem ist ein DFA nach Sipser immer vollständig.

3.1.1 Akzeptiertes Wort

Ein Wort w wird von einem DFA **akzeptiert**, wenn es bis zum Ende gelesen wird und sich der DFA in einem Endzustand befindet. Die akzeptierte/erkannte Sprache des DFA F ist:

$$L(F) = A = \{w \mid F \text{ akzeptiert } w\}$$

3.2 Nondeterministic Finite Automaton (NFA)

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

Symbol	Erklärung
A	**NFA**
Q	Endliche **Menge** aller Zustände
Σ	Eingabealphabet
δ	Übergangsfunktion $Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$
q_0	Startzustand
$F \subseteq Q$	**Menge** der Endzustände

3.2.1 Akzeptiertes Wort

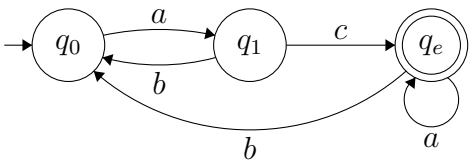
Ein Wort w wird von einem NFA **akzeptiert**, wenn es irgendeine Rechnung gibt, in der sich der NFA in einem Endzustand befindet, nach dem er das Wort komplett gelesen hat.

3.2.2 Generalized Nondeterministic Finite Automaton (GNFA)

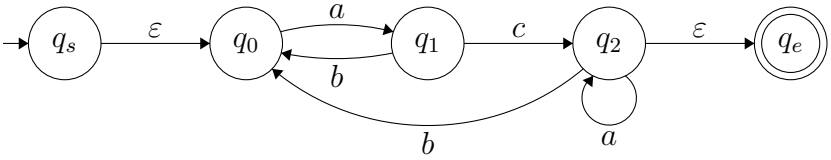
- Kanten eines GNFA können Reguläre Ausdrücke sein.
- Ein GNFA kann ganze Blöcke (Reguläre Ausrücke) aus dem Input lesen und nicht nur einzelene Symbole.
- Der Startzustand hat ausgehende Kanten zu jedem anderen Zustand, aber keine eingehenden.
- Es gibt nur einen Endzustand, dieser hat eingehende Kanten von jedem Zustand, aber keine ausgehenden.
- Startzustand und Endzustand sind nicht der gleiche Zustand.
- Für jeden Zustand außer Start- und Endzustand gilt außerdem, dass eine Kante zu jedem Zustand, inklusive sich selbst, existieren muss.

DFA zu Regulärem Ausdruck: Um von einem DFA (x-Zustände) zu einem Regulären Ausdruck zu gelangen, sind (nach Lemma 1.60) folgende Schritte notwendig:

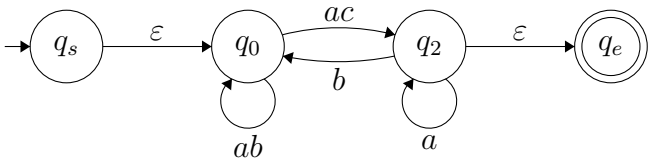
- Konstruktion eines äquivalenten GNFA mit x+2-Zuständen
- Solange einen Zustand wegnehmen, bis nur noch zwei Zustände (eine Kante) übrig sind
- Ablesen des Regulären Ausdrucken



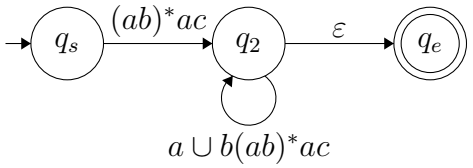
(a) NFA M



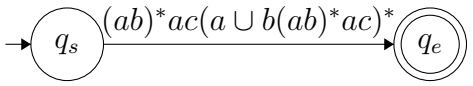
(b) Neuer Start- und Endzustand



(c) Entfernen von q_1



(d) Entfernen von q_0



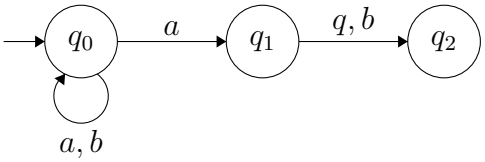
(e) Entfernen von q_2

Der letzte Graph ist nun ein GNFA mit zwei Zuständen und einer einzigen Kante, welche einen Regulären Ausdruck als Kantenbeschriftung hat.
(Bei (b), (c) und (d) wurden alle $\emptyset$-Kanten aus Gründen der Übersichtlichkeit weggelassen.)

3.3 Potenzautomat (NFA zu DFA)

Symbol	Erklärung
N	**NFA**, 5-Tupel: $(Q, \Sigma, \delta, q_0, F)$
Q	Endliche **Menge** aller Zustände
Σ	Eingabealphabet
δ	Übergangsfunktion $Q \times \Sigma \to Q$
q_0	Startzustand
$F \subseteq Q$	**Menge** der Endzustände
M	**Potenzautomat**, 5-Tupel: $(Q', \Sigma', \delta', q_0', F')$
Q'	$\mathcal{P}(Q), \quad \lvert Q' \rvert \leq 2 \lvert Q \rvert$
Σ'	Σ
$\delta'(R, x)$	$\bigcup_{q \in R} \delta(q, x), \ R \in Q'$
q_0'	$\{q \mid \delta(q_0, \varepsilon^*) = q, \ q \in Q\}$
F'	$\{R \in Q' \mid R \cap F \neq \emptyset\}$
	Alle Zustandsmengen $R \in Q'$, welche $q \in F$ enthalten.

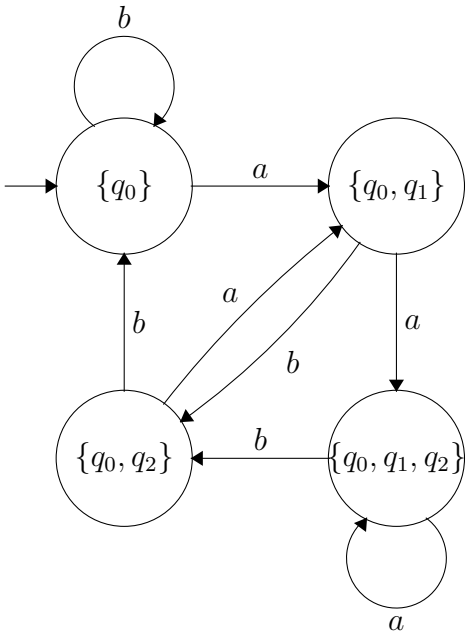
Beispiel NFA:



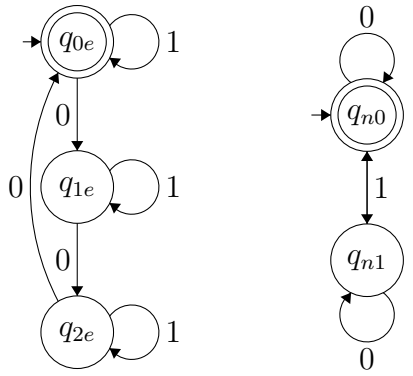
Fertiger Potenzautomat/DFA:

$$\begin{aligned} \delta'(\{q_0\}, a) &= \{q_0, q_1\} \\ \delta'(\{q_0, q_1\}, a) &= \delta(q_0, a) \cup \delta(q_1, a) \\ &= \{q_0, q_1\} \cup \{q_2\} \\ &= \{q_0, q_1, q_2\} \end{aligned}$$

$$\begin{aligned} R &= \{q_0, q_1\} \\ \delta(q_0, a) &= \{q_0, q_1\} \\ \delta(q_1, a) &= \{q_2\} \end{aligned}$$



3.4 Mengenoperationen auf Sprachen von FSA



(a) A (b) B

$$L(A) = \{w \in \Sigma^* \mid |w|_0 \mod 3 = 0\}$$
$$L(B) = \{w \in \Sigma^* \mid |w|_1 \mod 2 = 0\}$$

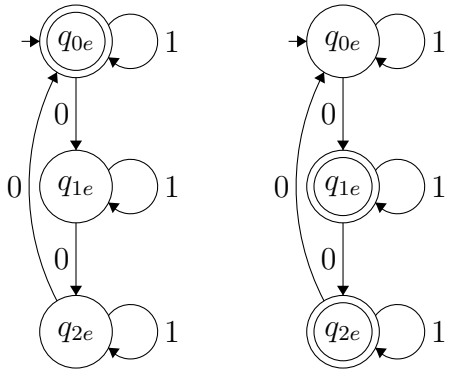
3.4.1 Komplementbildung

Ein Automat $\overline{A}$, der das Komplement $\overline{L(A)}$ der Sprache $L(A)$ erkennt, wird durch folgendes 5-Tupel repräsentiert:

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

wobei Q', Σ', δ' sowie q_0' identisch zu den Gegenstücken aus A sind und $F' = \{Q \setminus F\}$.

Abbildung 3.4



(a) A (b) $\overline{A}$

3.4.2 Vereinigung

Um einen NFA zu konstruieren, der die Vereinigung der Sprachen von den DFA A und B erkennt, genügt es einen neuen Stratzustand (hier q_{en}) einzufügen mit ε-Kanten zu den ursprünglichen Startzuständen (q_{n0} und q_{0e}).

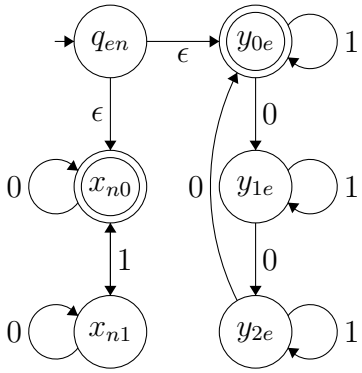


Abbildung 3.5: NFA von $L(A) \cup L(B)$

Um einen DFA zu konstruieren, der die Vereinigung erkennt, braucht man etwas mehr Aufwand:

- Die **Zustandsmenge** Q' ist wie folgt definiert: $\{(r_1, r_2) \mid r_1 \in Q_1 \wedge r_2 \in Q_2\}$, also alle Tupel, wobei r_1 aus der Zustandsmenge von dem einen Automaten kommt (in unserem Fall von dem Automaten A) und r_2 aus der Zustandsmenge von dem anderen (also bei uns aus B).

- Das Alphabet Σ' ist definiert als $\Sigma_1 \cup \Sigma_2$. Also in unserem Beispiel $\{0,1\} \cup \{0,1\} = \{0,1\}$.

- Die Übergangsfunktion ist wie folgt definiert:

$$\delta'\Big((r_1, r_2), a\Big) = \Big(\delta_1(r_1, a), \delta_2(r_2, a)\Big)$$

  für alle Tupel $(r_1, r_2) \in Q'$ und für alle $a \in \Sigma'$.

  Nehmen wir beispielsweise die Übergangsfunktion von q_10 aus unserem Beispiel: Die $_1$ steht für den Zustand q_1 von dem Automaten A und die 0 steht für den Zustand q_0 aus dem Automaten B.

  Befinden wir uns also im Automaten A bei dem Zustand q_1 und lesen eine 0, würden wir in den Zustand q_2 kommen. Somit ändert sich die $_1$ aus unser Zustand q_{10} zu einer $_2$. Befinden wir uns dagegen im Automaten B bei dem Zustand q_0 und lesen eine 0, so bleiben wir bei dem Zustand q_0. Daraus ergibt sich die folgende Übergangsfunktion für q_{10}: $\delta'(q_{10}, 0) = q_{20}$

- q_0 wird durch das Tupel (q_x, q_y) gebildet, wobei q_x der Startzustand von A ist und q_y der Startzustand von B. Dem entsprechend ist der Startzustand von unserem Automaten q_{00}

- Die Menge der Endzustände ist wie folgt definiert: $F' = (F_1 \times Q_2) \cup (Q_1 \times F_2)$, also die Zustände der Tupel, bei denen mindestens ein Element des Tupels aus F_1 oder F_2 kommt. Das bedeutet in unserem Fall: $F' = \{q_{00}, q_{01}, q_{10}, q_{20}\}$

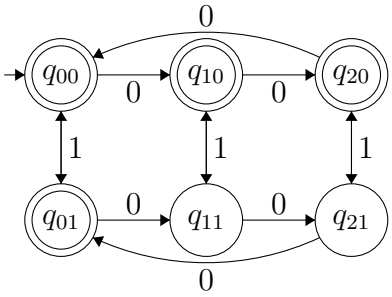


Abbildung 3.6: DFA von $L(A) \cup L(B)$

3.4.3 Schnitt

Um einen DFA zu konstruieren, der die Schnittmenge zweier Automaten (A und B) erkennt, müssen nur die Endzustände vom DFA zur Vereinigung angepasst werden. Die Endzustände bestehen dann aus folgender Menge: $F = (F_1 \times Q_2) \cap (Q_1 \times F_2)$, also die Menge der Tupel, bei denen das eine Element aus F_A und das andere Element aus F_B kommt. In unserem Beispiel ist das nur der Zustand q_{00}

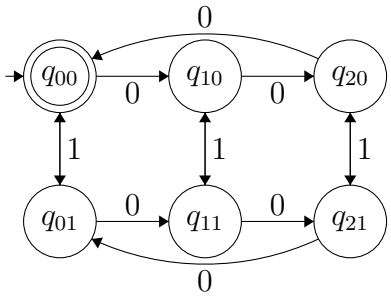
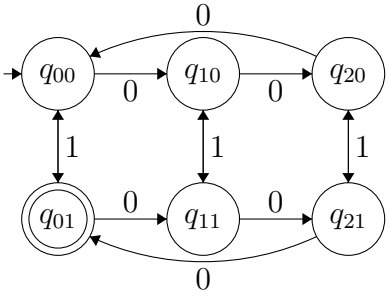


Abbildung 3.7: DFA von $L(A) \cap L(B)$

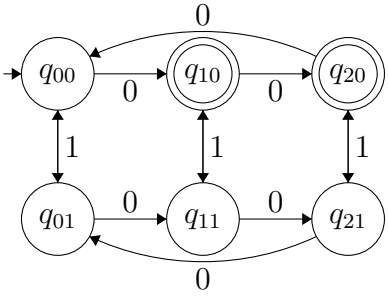
3.4.4 Differenz

Die Differenz zweier Mengen M_1 und M_2, lässt sich auch durch $M_1 \cap \overline{M_2}$ ausdrücken, möchte man also einen Automaten konstruieren, der die Differenz der Sprachen der DFA A und B erkennt, bildet man das Komplement des Automaten B, vertauscht also die Menge der Endzustände und der restlichen Zustände. Im Anschluss wird dann nur noch der Automat, der den Schnitt von A und $\overline{B}$ erkennt, wie bei dem Abschnitt 3.4.3 (Schnitt) konstruiert.

Abbildung 3.8



(a) DFA von $L(A) \setminus L(B)$



(b) DFA von $L(B) \setminus L(A)$

4. Pushdown Automaton / Kellerautomat (PDA)

PDAs haben einen zusätzlichen Stack-Speicher. (N)PDAs und DPDAs akzeptieren nicht die gleich Sprachfamilie.

4.1 Nondeterministic Pushdown Automaton (PDA)

Symbol	Erklärung
P	**PDA**
Q	Endliche **Menge** aller Zustände
Σ	**Eingabealphabet** (ohne ε)
Γ	**Kelleralphabet / Stapelalphabet**
δ	Übergangsfunktion $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$
q_0	Startzustand
$F \subseteq Q$	**Menge** der Endzustände

Ein PDA akzeptiert einen Eingabestring, wenn er sich am Ende des Lesens in einem Endzustand befindet, nach Sipser muss der Keller zum Akzeptieren nicht leer sein, aber es gibt auch Varianten bei denen das anders ist. Nach Sipser-Notation gibt es dafür das Symbol \$, welches am Anfang ganz unten auf den Stapel abgelegt wird und beim erneuten Einlesen signalisiert, dass der Stapel leer sein muss.

Kantenbeschriftung: $a, b \to c$

- a: Das Symbol, welches vom Eingabeband gelesen wird.
- b: Das Symbol, welches von Stack (Keller) gelesen (und gelöscht) wird.
- c: Das Symbol, welches auf den Stack (Keller) geschrieben wird.

Beispiel: PDA, der $\{0^n 1^n \mid n \geq 0\}$ erkennt

$Q = \{q_1, q_2, q_3, q_4\}$

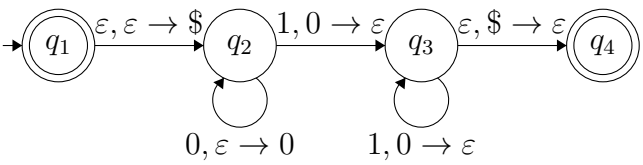
$\Sigma = \{0, 1\}$

$\Gamma = \{0, \$\}$

$F = \{q_1, q_4\}$

$\delta =$

Input	0			1			ε		
Stack	0	\$	ε	0	\$	ε	0	\$	ε
q_1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{(q_2, \$)\}$
q_2	$\emptyset$	$\emptyset$	$\{(q_3, \varepsilon)\}$	$\{(q_3, \emptyset)\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\emptyset$	$\{(q_3, \varepsilon)\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{(q_4, \varepsilon)\}$	$\emptyset$
q_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



4.2 Deterministic Pushdown Automaton (DPDA)

Symbol	Erklärung
P	**PDA**
Q	Endliche **Menge** aller Zustände
Σ	**Eingabealphabet** (ohne ε)
Γ	**Kelleralphabet / Stapelalphabet**
δ	Übergangsfunktion $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to (Q \times \Gamma_\varepsilon) \cup \{\emptyset\}$
q_0	Startzustand
$F \subseteq Q$	**Menge** der Endzustände

Zudem muss folgendes gelten: $\forall q \in Q, a \in \Sigma, x \in \Gamma$ muss genau eine der folgenen Übergangsfunktion ungleich der leeren Menge sein:

$$\delta(q, a, x), \delta(q, a, \varepsilon), \delta(q, \varepsilon, x), \delta(q, \varepsilon, \varepsilon)$$

Wenn ein DPDA keine "legale" Möglichkeit mehr hat, also keine der gerade genannten Übergangsfunktionen ungleich der leeren Menge sind, lehnt der DPDA die Eingabe ab ohne bis zum Ende zu lesen.

Ein DPDA akzeptiert den Input, wenn er ihn bis zum Ende liest und sich dann in einem Accept-Zustand befindet, in allen anderen Fällen lehnt er den Inputstring ab, (also, wenn sich der DPDA nach dem Lesen nicht in einem Accept-Zustand befindet, wenn er keine legale Möglichkeit mehr hat oder wenn er loopt).

Für jeden DPDA gibt es einen DPDA, der immer die gesamte Eingabe liest.

5. Turing Mashine (TM)

$$TM = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$

Symbol	Erklärung
TM	**Turing Machine**
Q	Endliche **Menge** aller Zustände
Σ	Eingabealphabet ohne $\sqcup$
Γ	Das Bandalphabet, mit $\sqcup \in \Gamma$ und $\Sigma \in \Gamma$
δ	Übergangsfunktion $Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
q_0	Startzustand
q_{accept}	**Accept-Zustand**
q_{reject}	**Reject-Zustand**

Eine Turing Machine (TM) hat ein Band, welches (im ursprünglichen Konzept) nach rechts unendlich, nach links endlich ist, es gibt aber auch Konzepte, bei denen das Band in beide Richtungen unendlich ist. Der "unbeschriebene" Teil des Bands ist mit $\sqcup$-Blankcharacters beschriftet. Der Lesekopf der TM startet bei dem sich am weitesten links befindenen Symbol, wenn der Lesekopf der TM übers linke Bandende hinausgeht, bleibt der Kopf technisch gesehen an der Stelle, aber die TM hält nicht.
Es gibt auch Variationen der TM, welche mehrere Bänder hat, diese lassen sich in eine klassische TM umformen.
Wenn die TM durch den Accept-Zustand geht, ohne vorher durch den Reject-Zustand zu gehen, wird die Eingabe akzeptiert. Andersrum gilt, geht sie zuerst durch einen Reject-Zustand, lehnt sie ab. Geht sie zu keinem Zeitpunkt durch einen Accept- oder Reject-Zustand, loopt die TM.

5.1 Abzählbar/Aufzählbar/Entscheidbare Mengen

Eine Menge ist **abzählbar**, wenn sie endlich ist oder es eine Bijektion aus den natürlichen Zahlen auf die Menge gibt.

Eine Menge ist **aufzählbar**, wenn sie abzählbar ist und die Elemente der Menge in endlicher Zeit durch eine TM als solche identifizierbar sind

Eine Menge ist **erkennbar**, wenn sie und ihr Komplement aufzählbar sind.

5.2 A_{TM}

A_{TM} (Sipser Third Edition, Theorem 4.11) ist eine Sprache, welche in der Lage sein soll, zu überprüfen, ob eine TM über einem Input akzeptiert oder ablehnt. Sie ist definiert als:

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ ist eine TM und } M \text{ akzeptiert } w\}$$

A_{TM} ist nicht entscheidbar, aber Turing-erkennbar; dies kann man zeigen mithilfe der **universellen TM** U, welche wie folgt definiert ist:

Über dem Input $\langle M, w\rangle$, wobei M eine TM und w ein String sind:
1. Simuliere M über dem Input w.
2. Falls M jemals einen Accept-Zustand betritt, akzeptiere; falls M jemals einen Reject-Zustand betritt, lehne ab.

U ist in der Lage jede andere TM durch ihre Beschriebung zu simulieren. Allerdings loopt sie über dem Input $\langle M, w\rangle$, wenn M über w loopt, daher kann sie A_{TM} nicht entscheiden, aber erkennen.

Beweis durch Widerspruch: Um zu zeigen, dass A_{TM} nicht entscheidbar ist, gehen wir zunächst vom Gegenteil aus und sagen $H(\langle M, w\rangle)$ ist ein Decider für A_{TM}, welcher akzeptiert, wenn M w akzeptiert und andernfalls ablehnt. Wenn wir nun eine TM D konstruieren, welche H als Subroutine nutzt:

Über dem Input $\langle M\rangle$, wobei M eine TM ist:
1. Führe H über dem Input $\langle M, \langle M\rangle\rangle$ aus.
2. Gib das Gegenteil von dem aus, was H ausgibt.

Wenn man D nun auf sich selbst anwendet, würde gelten: D lehnt $\langle D\rangle$ ab, wenn D $\langle D\rangle$ akzeptiert. Dies ist ein klarer Widerspruch, weshalb weder D noch H existieren können und somit A_{TM} nicht entscheidbar ist.

5.3 Halteproblem

Das Halteproblem (Sipser Third Edition, Theorem 5.1), herauszufinden ob eine TM über einem Input hält, ist unentscheidbar. Um dies zu zeigen, definiert man die TM:

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ ist eine TM und } M \text{ hält über dem Input } w\}$$

Beweis durch Widerspruch: Man geht wie bei A_{TM} zunächst davon aus, dass die Sprache $HALT_{TM}$ entscheidbar ist. Wir nutzen dies um zu zeigen, dass A_{TM} entscheidbar ist. Dies ist ein Widerspruch zur tatsächlichen Unentscheidbarkeit von A_{TM}.

Annahme: Es gibt eine TM R, welche $HALT_{TM}$ entscheidet. Diese konstruiert eine TM $S(\langle M, w\rangle)$, welche A_{TM} entscheidet. Da S allerdings loopt, wenn M loopt, würde diese Simulation nicht halten und S könnte kein Decider sein.
Stattdessen soll R testen ob M auf w hält. Deutet R an, dass M nicht auf w hält, soll R ablehnen, weil $\langle M, w\rangle$ nicht in A_{TM} sein kann. Andernfalls kann die Simulation durchgeführt werden.

Würde R nun existieren, gäbe es einen Decider für A_{TM}, was ein Widerspruch zur Unentscheidbarkeit von A_{TM} ist, weshalb $HALT_{TM}$ unentscheidbar sein muss.

5.4 Reduktion

$A \leq_m B$ wird als: "A wird auf B reduziert" gelesen und meint, dass das Problem A auf ein anderes Problem B reduziert werden kann, sodass wenn B lösbar ist, auch A lösbar ist.

Es soll gezeigt werden, dass jede Instanz aus dem Problem A in einen Spezialfall des Problems B überführt werden kann. Können wir nun Problem B lösen, so können wir dadurch auch Problem A lösen.

Seien A und B Sprachen sind mit $A \leq_m B$ und sei $f : \Sigma^* \to \Sigma^*$ die zugehörige Reduktion, dann gilt auch auch:

- $f(A) \cap \overline{B} = \emptyset \longleftrightarrow f(A) \subseteq B$
- Reduktionen sind transitiv, also gilt: $((A \leq_m B) \wedge (B \leq_m C)) \to (A \leq_m C)$
- $\overline{A} \leq_m \overline{B}$
- Wenn B entscheidbar ist, ist auch A entscheidbar
- Wenn B erkennbar ist, ist auch A erkennbar

6. Sprachen und Grammatiken

Eine Sprache ist eine, nicht zwangsweise endliche, Menge an Wörtern.

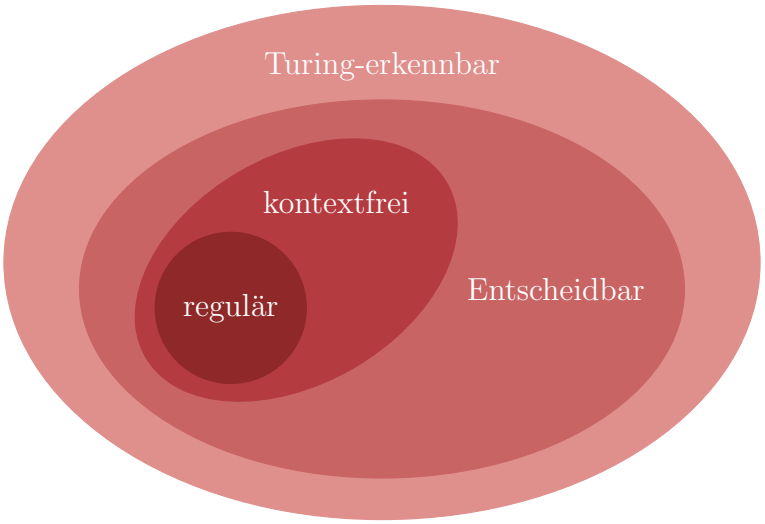


Tabelle 6.1: Abgeschlossenheit

Sprache	$\cup$	$\cap$	$\circ$	*	$\overline{}$
Regulär	⊠	⊠	⊠	⊠	⊠
CFL	⊠	□	⊠	⊠	□
DCFL	□	□	□[1]	□[1]	⊠
Entscheidbar	⊠	⊠	⊠	⊠	⊠
Erkennbar	⊠	⊠	⊠	⊠[1]	□

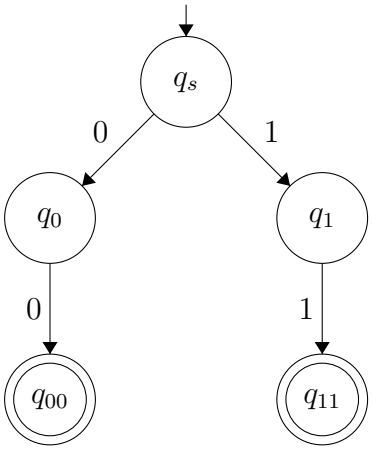
Kontextfreie Sprachen geschnitten mit regulären Sprachen sind kontextfrei.

[1] Nicht im Rahmen des Moduls bewiesen!

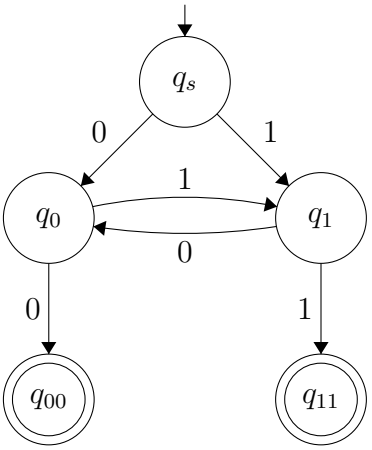
6.1 Beweise

6.1.1 Beweis durch Konstruktion

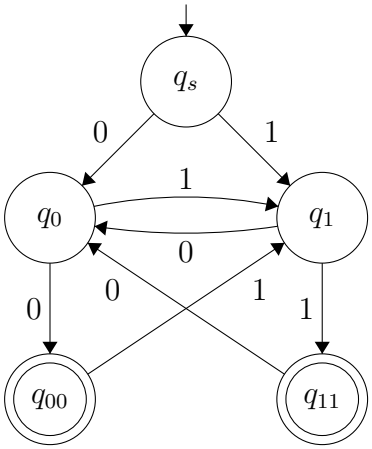
Abbildung 6.1: Beweis: $L(A) := \{w | w \text{ endet entweder auf } 00 \text{ oder auf } 11\}$ ist regulär



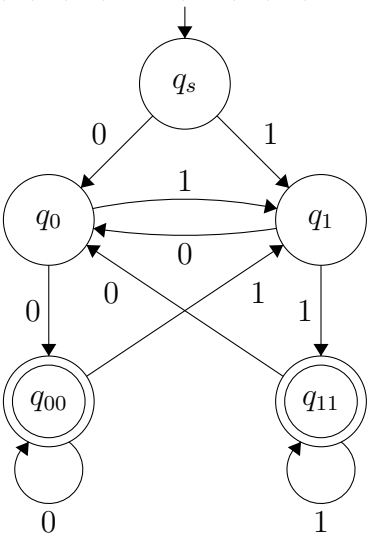
(a) $(00) \cup (11)$



(b) $(01)^+(1 \cup 00) \cup (10)^+(0 \cup 11)$

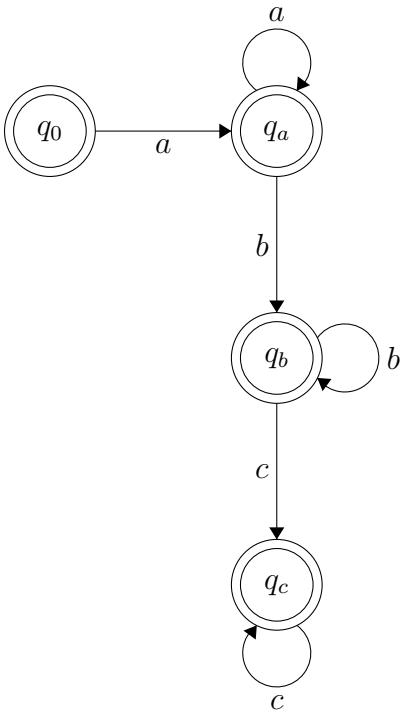


(c) $\ldots((001) \cup (110))\ldots$

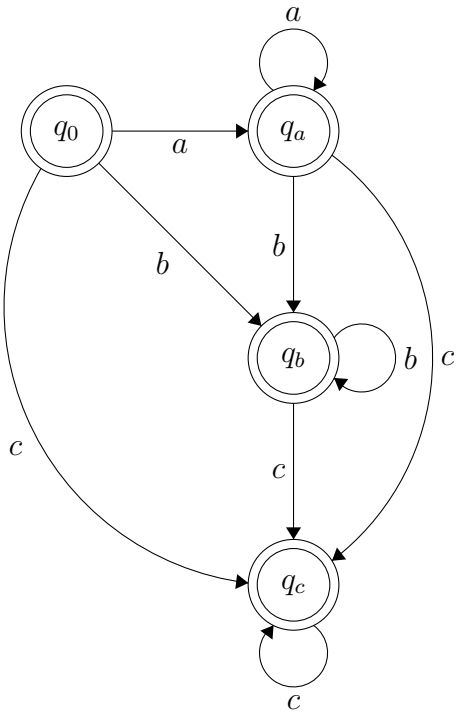


(d) $\ldots(000^+ \cup 111^+)\ldots$

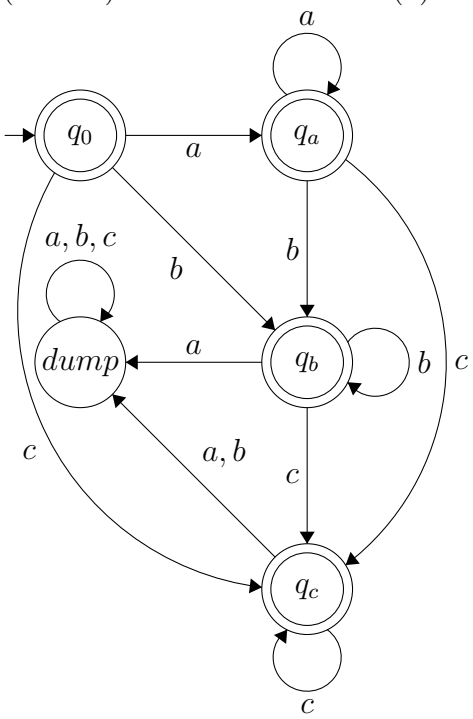
Abbildung 6.2: Beweis: $L(B) := a^*b^*c^*$ ist regulär



(a) Fall: $\varepsilon \cup (a^+b^*) \cup (a^+b^+c^*)$



(b) Fall: $(b^*c^+) \cup (a^+c^+)$



(c) Fall: Dump

6.1.2 Beweis durch Widerspruch

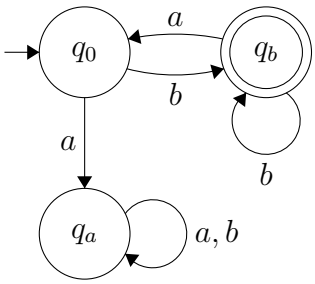


Abbildung 6.3: DFA A

Beweis durch Widerspruch: Die Sprache $L = b^+(ab)^*b^+$ ist nicht äquivalent zu der Sprache $L(A)$ des Automaten A:

Angenommen, die Sprache L sei äquivalent zu der Sprache $L(A)$, so müsste jedes Wort, welches der Automat A akzeptiert in L ($L(A) \subseteq L$) sein.

$$
\begin{aligned}
&q_0babbabb\\
&bq_babbabb\\
&baq_0bbabb\\
&babq_bbabb\\
&babbq_babb\\
&babbaq_0bb\\
&babbabq_bb\\
&babbabbq_b
\end{aligned}
$$

Das Wort $w = babbabb$ ist in der Sprache $L(A)$, da es vom Automaten zu Ende gelesen werden kann und dieser sich am Ende im Endzustand q_b befindet.
Dieses Wort ist aber nicht Teil der Sprache L:

b^+	$(ab)^*$	b^+
b	ab	$babb$
b	$abbab$	b

6.1.3 Beweis zur Äquivalenz von Sprachen

Allgemein: Es sind zwei Richtungen zu zeigen:
$L \subseteq L(A)$

- Alle Wörter die in der Sprache L sind werden akzeptiert.
- Alle Wörter die nicht akzeptiert werden sind nicht in L.

$L(A) \subseteq L$

- Alle Wörter die akzeptiert werden sind in L.
- Alle Wörter die nicht in L sind werden von A nicht akzeptiert.

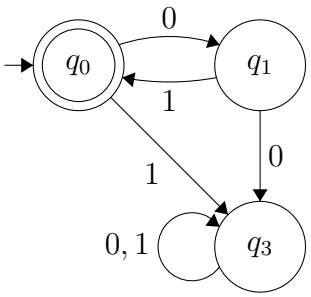


Abbildung 6.4: DFA A

$L = \{w \in \{0,1\}^* \mid 0 \text{ und } 1 \text{ wechseln sich ab, beginnend bei } 0\}$

$L \subseteq L(A)$**:**
Sei das Wort $w \in L$. Damit das Wort akzeptiert wird muss sich der Automat nach dem vollständigen einlesen des Wortes im Zustand q_0 oder q_1 befinden. Damit dies geschieht darf der Automat nie in den Zustand q_2 kommen, da er von dort nicht mehr rauskommt. Folglich kann das Wort höchstens zwischen q_0 und q_1 wechseln. Dies bedeutet, dass das Wort abwechselnd eine 0 und eine 1 haben muss, beginnend mit einer 0. Folglich muss $w \in L(A)$ gelten.

$L(A) \subseteq L$**:**
Sei das Wort $w \in L(A)$. Dann muss es abwechselnd aus 0 und 1 bestehen, beginnend mit einer 0. Folglich kann der Automat A nur zwischen q_0 und q_1 wechseln. Da beide Zustände akzeptieren, wird auch das Wort akzeptiert und es gilt $w \in L$.

Da nun bewiesen ist, dass $L \subseteq L(A)$ und $L(A) \subseteq L$, gilt $L = L(A)$.

6.2 Erkannte (akzeptierte) Sprachen

Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DFA.
Sei $w = w_1 w_2 \dots w_n$ ein String mit $w_i \in \Sigma$.
Dann **erkennt** M w, falls es eine Folge von Zuständen $r_0, r_1, \dots, r_n$ in Q gibt, für die gilt:

- $r_0 = q_0$
- $\delta(r_i, w_{i+1}) = r_{i+1}$, für $i = 0, 1, \dots, n-1$
- $r_n \in F$

M erkennt die Sprache A, falls $A = \{w \mid M \text{ erkennt } w\}$.
A wird auch als $L(M)$ geschrieben.

6.3 Reguläre Sprachen

Eine Sprache ist regulär, wenn sie von einem endlichen Automaten erkannt wird, sowie jede endliche Sprache. Die Familie der regulären Sprachen wird mit REG bezeichnet.

Zu jeder regulären Sprache L gibt es:

- einen DFA A mit $L(A) = L$
- einen NFA B mit $L(B) = L$
- einen NFA mit ε-Kanten C mit $L(C) = L$
- einen regulären Ausdruck D, der L beschreibt ($M_D = L$)

6.3.1 Beweis (Pumping Lemma für reguläre Sprachen)

Automaten, welche unendliche Mengen akzeptieren, müssen Schleifen durchlaufen, da die Zustandsmenge begrenzt ist. Eine Teilmenge der Zustände muss also bei bestimmten Wörtern mehrfach besucht werden. Dies gilt für alle regulären Mengen. Das Pumping Lemma kann nur benutzt werden um zu zeigen, dass eine Sprache nicht regulär ist:

Pumping Lemma ist	erfüllt	nicht erfüllt
Sprache ist regulär	☒	☐
Sprache ist nicht regulär	☒	☒

Wenn eine Sprache A regulär ist, dann gibt es eine Pumping Länge p, sodass jedes Wort $s \in A$ mit $|s| \geq p$ in die Form $s = xyz$ zerlegt werden kann, wobei

- $xy^i z \in A \; \forall i \in \mathbb{N} \cup \{0\}$
- $|y| > 0$
- $|xy| \leq p$

Um zu zeigen, dass eine Sprache A nicht regulär ist, genügt es zu zeigen, dass für alle Zerlegungen xyz eines Wortes $s \in A$ das Pumping Lemma nicht erfüllt ist.

Beispiel: $L = \{a^n b^m \mid n < m\}$
Angenommen L sei regulär, so müsste jedes Wort aus L die Bedingungen des Pumping Lemmas erfüllen.
Sei p die Pumping Length und $s \in L$ ein Wort aus der Sprache; wir betrachten das Wort $s = a^p b^{p+1}$. Das Pumping Lemma muss für alle Zerlegungen $s = xyz$ erfüllt sein:

1 | $x = a^{p-j-k}, y = a^j, z = a^k b^{p+1} \quad j > 1, j + k < p$
Mit $i = 2$ wird $|xy^i z|_a \not< |xy^i z|_b$ und das Lemma verletzt:
$n = p - j - k + j * i + k < p + 1 = m$
$n = p + j \not< p + 1 = m, \ j \geq 1$

2 | $x = a^{p-j}, y = a^j b^k, z = b^{p-k+1} \quad j > 1 \vee k > 1, \ j < p, \ k < p + 1$
Wird y mit $i \geq 2$ gepumped, wird die Reihenfolge verletzt.

3 | $x = a^p b^{p-j-k+1}, y = b^j, z = b^k \quad j > 1, \ j + k < p + 1$
Da y nicht leer sein darf und $|x| \geq p$, wird $|xy| > p$ und somit die dritte Bedingung des Pumping Lemmas verletzt.

Da alle möglichen Zerlegungen des Wortes das Pumping Lemma verletzen, kann L nicht regulär sein.

6.4 Reguläre Ausdrücke

Induktive Definition: R ist ein regulärer Ausdruck, wenn R einem der folgenden Ausdrücke entspricht:

1. $a \in \Sigma$,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, mit R_1 und R_2 sind reguläre Ausdrücke,
5. $(R_1 \circ R_2)$, mit R_1 und R_2 sind reguläre Ausdrücke, oder
6. (R_1^*), mit R_1 ist ein regulärer Ausdruck.

Abschlussklausel: Nur die so erzeugten Ausdrücke sind reguläre Ausdrücke.
Gebe einen regulären Ausdruck für:

$$L_1 = \{w \in \{0,1\} \mid 0 \text{ und } 1 \text{ wechseln sich ab, beginnend bei } 0\}$$

Regulärer Ausdruck:

$$0((10)^* \cup (10)^* 1)$$

Gebe einen regulären Ausdruck für:

$$L_2 = \{w \in \{0,1\} \mid 0 \text{ und } 1 \text{ wechseln sich ab, beginnend bei } 0\}$$

Regulärer Ausdruck:

$$a^* b^* c^*$$

6.5 Kontextfreie Sprachen

Eine Sprache ist kontextfrei, wenn sie von einem PDA erkannt oder es eine CFG gibt, die sie erkennt. Wenn sie kontextfrei ist, wird sie von einem PDA erkannt.
Jede reguläre Sprache ist auch kontextfrei und wird von einem PDA erkannt, wobei der Stack ignoriert wird.

6.5.1 Beweis (Pumping Lemma für kontextfreie Sprachen)

Das Pumping Lemma kann nur benutzt werden um zu zeigen, dass eine Sprache nicht kontextfrei ist:

Pumping Lemma ist	erfüllt	nicht erfüllt
Sprache ist kontextfrei	⊠	□
Sprache ist nicht kontextfrei	⊠	⊠

Wenn eine Sprache A kontextfrei ist, dann gibt es eine Pumping Länge p, sodass jedes Wort $s \in A$ mit $|s| \geq p$ in die Form $s = uvxyz$ zerlegt werden kann, wobei

- $uv^ixy^iz \in A \; \forall i \in \mathbb{N} \cup \{0\}$
- $|vy| > 0$
- $|vxy| \leq p$

Um zu zeigen, dass eine Sprache A nicht kontextfrei ist, genügt es zu zeigen, dass für alle Zerlegungen $uvxyz$ eines Wortes $s \in A$ das Pumping Lemma nicht erfüllt ist.

Beispiel: $L = \{(ab)^n(ca)^n(bc)^n \mid n \geq 0\}$
Angenommen L sei kontextfrei, so müsste jedes Wort aus L die Bedingungen des Pumping Lemmas erfüllen.
Sei p die Pumping Length und $s \in L$ ein Wort aus der Sprache; wir betrachten das Wort $s = (ab)^p(ca)^p(bc)^p$. Das Pumping Lemma muss für alle Zerlegungen $s = uvxyz$ erfüllt sein:

$v = (ab)^j, y = (ab)^k, \quad j > 1 \vee k > 1, j + k < p$

- dann gilt für $v^ixy^i, i = 2$: $(ab)^{p-j-k+2j+2k}(ca)^p(bc)^p = (ab)^{p+j+k}(ca)^p(bc)^p$
- das ist nicht in L, da $p + j > p$

- dies gilt analog für $v = (ca)^j, y = (ca)^k, \quad j > 1 \vee k > 1, j + k < p$
- und $v = (bc)^j, y = (bc)^k, \quad j > 1 \vee k > 1, j + k < p$

$v = (ab)^j, y = (bc)^k, \quad j, k > 1$

- dann müsste $x = (ca)^p$ und damit $|vxy| = p + j + k \geq p, \quad j, k > 1$

- außerdem gilt dann für $v^i xy^i, i = 2$:
  $(ab)^{p-j+2j}(ca)^p(bc)^{p-k+2k} = (ab)^{p+j}(ca)^p(bc)^{p+k}$
- das ist nicht in L, da $p + j > p < p + k$

$u = (ab)^{p-j-k}, v = (ab)^j, x = (ab)^k(ca)^l, y = (ca)^m, z = (ca)^{p-m-l}(bc)^p,$
$j, m \geq 1, k, l \geq 0$

- dann gilt für $uv^i xy^i z,\ i = 2$:
  $(ab)^{p-j-k+2j+k}(ca)^{p-m-l+2m+l}(bc)^p = (ab)^{p+j}(ca)^{p+m}(bc)^p$
- das ist nicht in L, da $p + j > p < p + m$

- dies gilt analog für
  $u = (ab)^p(ca)^{p-j-k}, v = (ca)^j, x = (ca)^k(bc)^l, y = (bc)^m, z = (bc)^{p-l-m},$
  $j, m \geq 1, k, l \geq 0$

Da $|vy| > 0$, gilt für alle weiteren Zerlegungen:

- Falls v oder y eine Kombination aus $(ab), (ca), (bc)$ enthält,
- wird beim Pumpen die Reihenfolge und somit das Pumping Lemma verletzt.

Da alle möglichen Zerlegungen des Wortes das Pumping Lemma verletzen, kann L nicht kontextfrei sein.

6.6 Kontextfreie Grammatiken

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

V	Das endliche Alphabet der **Variablen/Nonterminale**
Σ	Das endliche Alphabet der **Terminale** mit $\Sigma \cap V = \emptyset$
R	**Produktionsregeln**
S	**Startsymbol** (Startvariable)

Sprache einer Kontextfreien Grammatik (CFG) G:

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w\}$$

6.6.1 Ableitung

Einschrittige Ableitungen: Um die einschrittige Ableitung des Wortes v aus dem Wort u, mittels einer Regel aus der Grammatik G, zu notieren wird wie folgendes geschrieben:

$$u \underset{G}{\Rightarrow} v$$

Wenn der Kontext klar ist kann das G auch weggelassen werden. Für **mehrschrittige Ableitungen** wird $\overset{*}{\underset{G}{\Rightarrow}}$ verwendet.

6.6.2 Mehrdeutigkeit

Eine Grammatik ist mehrdeutig (ambiguous), wenn es mehrere Syntaxbäume (Parse Trees) gibt, um das gleiche Wort zu erzeugen.

6.6.3 CFL zu CFG

L_{CF}	$\{a^n b^m \mid 0 \leq n < m\}$
$S \to aSb \mid \varepsilon$	$\{a^n b^n \mid 0 \leq n\}$
$S \to aSb \mid Sb \mid b$	$\{a^n b^m \mid 0 \leq n < m\}$

L_{CF}	$\{a^{2n} b^n \mid 0 \leq n\}$
$S \to aaSb \mid \varepsilon$	$\{a^{2n} b^n \mid 0 \leq n\}$

6.6.4 Chomsky Normalform

Eine kontextfreie Grammatik in Chomsky Normalform hat die Form

$$\begin{aligned} A &\to BC \\ A &\to a \end{aligned}$$

Jedes Nonterminal kann also entweder auf zwei Nonterminale oder auf ein Terminal auflösen. Außerdem darf nur die Startvariable S auf ε auflösen.
Anmerkung: Jede CFG kann in die Chomsky Normalform überführt werden.
Beispiel: CFG in Chomsky Normalform

$$\begin{aligned} G &= (V, \Sigma, R, S) \\ V &= \{S, A, B\} \\ \Sigma &= \{0, 1\} \\ R &= \end{aligned}$$

$$\begin{aligned} S &\to ABA \mid ABB \mid A \mid B1 \\ B &\to 0 \mid 1 \mid A \\ A &\to 0 \mid 1 \mid \varepsilon \end{aligned}$$

Einfügen von S_0:

$$\begin{aligned} S_0 &\to S \\ S &\to ABA \mid ABB \mid A \mid B1 \\ B &\to 0 \mid 1 \mid A \\ A &\to 0 \mid 1 \mid \varepsilon \end{aligned}$$

Entfernen von $A \to \varepsilon$:

$$
\begin{aligned}
S_0 &\to S \\
S &\to ABA \mid ABB \mid A \mid B1 \mid AB \mid BA \mid B \mid BB \mid \varepsilon \\
B &\to 0 \mid 1 \mid A \mid \varepsilon \\
A &\to 0 \mid 1
\end{aligned}
$$

Entfernen von $B \to \varepsilon$:

$$
\begin{aligned}
S_0 &\to S \\
S &\to ABA \mid ABB \mid A \mid B1 \mid AB \mid BA \mid B \mid BB \mid \varepsilon \mid AA \mid 1 \\
B &\to 0 \mid 1 \mid A \\
A &\to 0 \mid 1
\end{aligned}
$$

Vereinfachen, da quasi $B \to A|A$:

$$
\begin{aligned}
S_0 &\to S \\
S &\to AAA \mid AA \mid A \mid A1 \mid 1 \mid \varepsilon \\
A &\to 0 \mid 1
\end{aligned}
$$

Jedes Nonterminal auf genau zwei Nonterminale oder ein Terminal:

$$
\begin{aligned}
S_0 &\to S \\
S &\to AA_2 \mid AA \mid 0 \mid AA_1 \mid 1 \mid \varepsilon \\
A &\to 0 \mid 1 \\
A_1 &\to 1 \\
A_2 &\to AA
\end{aligned}
$$

Ersetzen von S_0 mit S, ε nur noch in der Startvariable:

$$
\begin{aligned}
S &\to AA_2 \mid AA \mid 0 \mid AA_1 \mid 1 \mid \varepsilon \\
A &\to 0 \mid 1 \\
A_1 &\to 1 \\
A_2 &\to AA
\end{aligned}
$$

6.7 Deterministische Kontextfreie Sprachen

Eine **Deterministische Kontextfreie Sprache** (DCFL) ist eine Sprache die von einem DPDA erkannt wird.
Die Familie der DCFL ist abgeschlossen gegenüber der Komplementbildung.
A ist eine DCFL gdw. $A \dashv$ eine DCFL ist. Das dient dazu, dass eine Grammatik das Ende des Inputs voraussetzen kann (wie ein $-Zeichen in einer Regex gängiger Programmiersprachen). Die Sprachen $A \dashv$ heißen **endmarkierte Sprachen**.

6.8 Deterministische Kontextfreie Grammatiken

Bei CFGs wird aus einer (Start-)Variable (in mehreren Zwischenschritten) ein Wort abgeleitet. Das umgekehrte Verfahren (aus einem Wort den Baum zur Startvariable aufbauen) heißt **Reduktion**. Bei jedem **Reduktionsschritt** wird ein String (der **Reduktionsstring**) ersetzt durch eine Variable.

Schreibweise: $abbcdde \rightarrowtail aBcdde \rightarrowtail aBcDe \rightarrowtail S$ oder kürzer $abbcdde \overset{*}{\rightarrowtail} S$ (äquivalent zu $S \overset{*}{\Rightarrow} abbcdde$)

In einer **linkesten Reduktion** wird ein String erst reduziert, wenn alle reduzierbaren Strings die ganz zu seiner Linken liegen reduziert sind. Das ist das Gegenstück zu einer rechtesten Ableitung.

Ein String der in einer linkesten Reduktion eines Wortes $w \in L(G)$ vorkommt, heißt ein **gültiger String**.

Ein **Handle** eines gültigen Strings ist ein Reduktionsstring gemeinsam mit der Regel des dazugehörigen Reduktionsschritts. Wenn G eine nicht doppeldeutige Grammatik ist, hat jeder gültige String einen eindeutigen Handle.

Ein Handle eines gültigen Strings ist genau dann ein **Forced Handle**, wenn er der eindeutige Handle von jedem gültigen String mit dem Anfang ist, unabhängig von was hinter dem Handle noch steht.

Eine **deterministische kontextfreie Grammatik** (DCFG) ist eine CFG, in der jeder gültige String einen Forced Handle hat.

DCFGs sind **für endmarkierte Sprachen** äquivalent zu DPDAs.

6.9 Turing-erkennbare Sprachen

wenn U	U			$\overline{U}$		
	erkennbar	entscheidbar	endlich	erkennbar	entscheidbar	endlich
entscheidbar	☒	☒	-	☒	☒	-
erkennbar	☒	-	-	-	-	-
unentscheidbar	-	☐	☐	-	☐	☐
unerkennbar	☐	-	☐	-	-	☐
endlich	☒	☒	☒	☒	☒	☐
unendlich	-	-	☐	-	-	-

6.10 Turing-entscheidbare Sprachen

Wenn eine TM auf jeder Eingabe einer Sprache hält (accept oder reject), dann ist die Sprache entscheidbar und die TM ein Decider für diese Sprache.

Die Entscheidbarkeit eines Problems ist zeigbar durch die Konstruktion einer TM, welche das Problem entscheidet oder durch Beweis der Äquivalenz zu einem anderen, entscheidbaren Problem.

7. Operatoren

7.1 Konkatenation

$$\{a, bc\} \cdot \{de, fg\} = \{ade, afg, bcde, bcfg\}$$
$$R = \{a, b\}$$
$$R^2 = R \cdot R = \{aa, ab, ba, bb\}$$

7.2 Mächtigkeit

Symbole	$x, y \in \Sigma$
Wort	$w \in \Sigma^*$ $w = xxyxxxy$
Mächtigkeit von w	$\lvert w \rvert = 7$
Häufigkeit von x in w	$\lvert w \rvert_x = 5$

7.3 Reguläre Mengenoperationen

Operation	Definition
Vereinigung	$A \cup B = \{x \mid x \in A \text{ oder } x \in B\}$
Durchschnitt	$A \cap B = \{x \mid x \in A \text{ und } x \in B\}$
Konkatenation	$A \circ B = \{xy \mid x \in A \text{ und } y \in B\}$
Stern	$A^* = \{x_1 x_2 \dots x_k \mid k \geq 0 \text{ und alle } x_i \in A\}$

Tabelle 7.1: Operatoren, denen gegenüber regulärer Sprachen abgeschlossen sind

8. Logik

8.1 Syntax

Mit $\mathbf{AS_{AL}}$ sei die Menge der Aussagensymbole der Aussagenlogik bezeichnet. Also die Menge der Symbole, die für einzelne Aussagen stehen, üblicherweise notiert mit A_1, A_2, A_3, etc. oder mit A, B, C, etc. Solche Symbole (bzw. die Aussagen dahinter) werden auch als Atome bezeichnet.
$\mathcal{L}_{\mathbf{AL}}$, bezeichnet die Menge der Formeln der Aussagenlogik.

Induktive Definition: Nur Formeln, die durch den folgenden induktiven Prozess definiert sind, sind Formeln.

1. Alle atomaren Formeln sind Formeln.
2. Für alle Formeln F und G sind $(F \wedge G), (F \vee G), (F \to G)$ und $F \Leftrightarrow G$ Formeln.
3. Für jede Formel F ist $\neg F$ eine Formel.

Tabelle 8.1: Junktoren in Bindungsreihenfolge

F	Entweder ein **Atom** oder eine **Formel**
$\neg F$	**Negierung** von F
$(F \wedge G)$	Logisches **Und**, Konjunktion
$(F \vee G)$	Logisches **Oder**, Disjunktion
$(F \to G)$	**Implikation**, Abkürzung für $(\neg F \vee G)$
$(F \leftrightarrow G)$	**Äquivalenz**, Abkürzung für $((F \wedge G) \vee (\neg F \wedge \neg G))$

8.1.1 Strukturelle Definition

Tabelle 8.2: Strukturelle Definitionen von Funktionen zur Syntax

$length(F)$	Anzahl an Symbole einer Formel $length(A) = 1$ $length(\neg F) = length(F) + 1$ $length((F \wedge G)) = length(F) + length(G) + 3$ $length((F \vee G)) = length(F) + length(G) + 3$
$degree(F)$	Anzahl an Junktoren einer Formel $degree(A) = 0$ $degree(\neg F) = degree(F) + 1$ $degree((F \wedge G)) = degree(F) + degree(G) + 1$ $degree((F \vee G)) = degree(F) + degree(G) + 1$
$depth(F)$	Anzahl an Junktoren des längsten Zweiges (Tiefe der Formel) $depth(A) = 0$ $depth(\neg F) = depth(F) + 1$ $depth((F \wedge G)) = \max\{depth(F), depth(G)\} + 1$ $depth((F \vee G)) = \max\{depth(F), depth(G)\} + 1$
$paren(F)$	Anzahl an Klammern einer Formel $paren(A) = 0$ $paren(\neg F) = paren(F)$ $paren((F \wedge G)) = paren(F) + paren(G) + 2$ $paren((F \vee G)) = paren(F) + paren(G) + 2$
wobei	$A \in AS_{AL}$ sowie $F, G \in L_{AL}$

8.1.2 Strukturelle Induktion

Allgemein: Um eine Behauptung $B(F)$ für jede Formel $F \in \mathcal{L}_{AL}$ zu zeigen, genügt es:

IA	Zu zeigen, dass $B(F)$ für jede atomare Formel F gilt.
IV	Anzunehmen, dass $B(F)$ und $B(G)$ für zwei Formeln F und G gilt.
IS	Zu zeigen, dass unter der Induktionsannahme nun auch $B(\neg F)$, $B((F \vee G))$ und $B((F \wedge G))$ (sowie $B((F \rightarrow G))$ und $B((F \leftrightarrow G))$) gelten.

Beispiel: Beweise, dass $paren(F) \leq 2 \cdot degree(F)$ gilt.

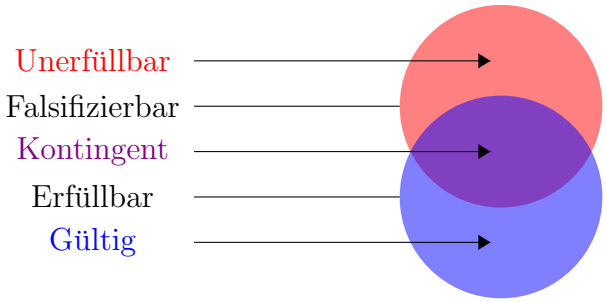
IA	Für $A \in AS_{AL}$:	$paren(A) = 0 \leq 0 = 2 \cdot degree(A)$
IV	Gelte für $F, G \in L_{AL}$:	$paren(F) \leq 2 \cdot degree(F)$ $paren(G) \leq 2 \cdot degree(F)$
IS	$paran(\neg F)$	$= paran(F)$ $\leq_{IA} 2 \cdot degree(F)$ $\leq_{IA} 2 \cdot (degree(F) + 1)$ $\leq 2 \cdot degree(\neg F)$
	$paren((F \wedge G))$ (analog für $\vee$)	$= paren(F) + paren(G) + 2$ $\leq_{IA} 2 \cdot degree(F) + paren(G) + 2$ $\leq_{IA} 2 \cdot degree(F) + 2 \cdot degree(G) + 2$ $= 2 \cdot (degree(F) + degree(G) + 1)$ $= 2 \cdot degree((F \wedge G))$

8.2 Semantik

Sei F eine Formel, so ist eine zu F passende **Belegung**, eine Belegung die jedem Atom in F einen Wahrheitswert zuordnet.
Ein **Modell** M ist eine zu F passende Belegung, unter der F den Wahrheitswert 1 zugeordnet bekommt, also wenn unter der Belegung $\mathcal{A}(F) = 1$ gilt. Man sagt dann auch M ist ein Modell für F oder eine erfüllende Belegung für F.

Begriff	**Bedeutung**	**Beispiel**
Unerfüllbar	**Kontradiktion**, immer falsch	$F = (\neg B \wedge B)$
Falsifizierbar	mindestens einmal falsch	$F = (B \to C)$
Kontingent	mindestens einmal wahr und einmal falsch	$F = (B \to C)$
Erfüllbar	mindestens einmal wahr	$F = (B \to C)$
Gültig	**Tautologie**, immer wahr	$F = (\neg B \vee B)$

	F ist eine Tautologie	
gdw.	jede Belegung ein Modell für F ist	(Def. der Gültigkeit)
gdw.	$\mathcal{A}(F) = 1$ für jede Belegung $\mathcal{A}$ ist	(Def. eines Modells)
gdw.	$\mathcal{A}(\neg F) = 0$ für jede Belegung $\mathcal{A}$ ist	(Eigenschaft von $\neg$)
gdw.	keine Belegung ein Modell von $\neg F$ ist	(Def. eines Modells)
gdw.	$\neg F$ unerfüllbar ist	(Def. der Unerfüllbarkeit)



8.2.1 Folgerungen

Eine Formel G folgt aus einer Formel F, wenn alle Modelle von F auch Modelle von G sind. Dementsprechend ist $A \wedge B$ ein Modell für $A \vee B$ (man schreibt $A \wedge B \models A \vee B$), denn immer wenn $A \wedge B$ gleich 1 ist, ist auch $A \vee B$ gleich 1:

A	B	$A \wedge B$	$A \vee B$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Außerdem gilt auch: Wenn $F \models G$, dann $\neg G \models \neg F$:

F	G	$\neg G$	$\neg F$
0	0	1	1
0	1	0	1
1	0	1	0
1	1	0	0

Die grünen Zeilen decken alle Fälle ab, die durch die Aussage $F \models G$ möglich sind. Somit gilt $F \models G$ gdw. $\neg G \models \neg F$.

Sei M eine Formelmenge und A eine Belegung:

$M \models F$	F folgt aus der Formelmenge M
$A \models F$	Die Belgung A erfüllt F
$\models F$	F ist eine Tautologie
$F \models$	F ist eine Kontradiktion

$M \models F$ gdw. $M \cup \{\neg F\} \models$, dies kann mit diversen Methoden zum Nachweiß von Unerfüllbarkeit gezeigt werden.

- Wenn $F \equiv G$ und $\models G$ gilt, dann gilt auch $\models F$.
- Wenn $F \equiv G$ und $G \models$ gilt, dann gilt auch $F \models$.
- Wenn $\models F$ und $\models G$ gilt, dann gilt auch $F \equiv G$.
- Wenn $F \models$ und $G \models$ gilt, dann gilt auch $F \equiv G$.
- Wenn $F \models G$ und $G \models F$ gilt, genau dann gilt auch $F \equiv G$.
- Wenn $\models F \leftrightarrow G$ gilt, genau dann gilt auch $F \equiv G$.
- Wenn $\models F \rightarrow G$ gilt, genau dann gilt auch $F \models G$.
- Wenn $F \wedge \neg G \models$ gilt, genau dann gilt auch $F \models G$.
- Wenn $M \models F \rightarrow G$ gilt, genau dann gilt auch $M \cup \{F\} \models G$.
- Wenn $M \cup \{\neg G\}$ unerfüllbar ist, genau dann gilt auch $M \models G$.

8.3 Äquivalenz

Wenn eine Formel F die gleiche Syntax hat wie eine andere Formel G, dann sind diese syntaktisch äquivalent:

$$F = G$$

Zwei Formeln F und G sind semantisch äquivalent (oder äquivalent), wenn die Wahrheitswerte beider identisch sind. Also zwei Formeln F und G sind äquivalent, gdw. $\mathcal{A}(F) = \mathcal{A}(G)$ für jede Belegung $\mathcal{A}$ gilt. Man schreibt auch:

$$F \equiv G$$

Wenn eine Formel per Definition äquivalent zu einer anderen ist, dann sind diese automatisch sowohl syntaktisch, als auch semantisch äquivalent zueinander:

$$F \to G := \neg F \vee G$$
$$F \to G = \neg F \vee G$$
$$F \to G \equiv \neg F \vee G$$

Tabelle 8.3: Äquivalenzregeln

$(F \wedge F)$	$\equiv$	F	Idempotenz
$(F \vee F)$	$\equiv$	F	
$(F \wedge G)$	$\equiv$	$(G \wedge F)$	Kommutativität
$(F \vee G)$	$\equiv$	$(G \vee F)$	
$((F \wedge G) \wedge H)$	$\equiv$	$(F \wedge (G \wedge H))$	Assoziativität
$((F \vee G) \vee H)$	$\equiv$	$(F \vee (G \vee H))$	
$(F \wedge (F \vee G))$	$\equiv$	F	Absorption
$(F \vee (F \wedge G))$	$\equiv$	F	
$(F \wedge (G \vee H))$	$\equiv$	$((F \wedge G) \vee (F \wedge H))$	Distributivität
$(F \vee (G \wedge H))$	$\equiv$	$((F \vee G) \wedge (F \vee H))$	
$\neg\neg F$	$\equiv$	F	Doppelnegation
$(F \leftrightarrow G)$	$\equiv$	$(F \to G) \wedge (G \to F)$	Elimination von $\leftrightarrow$
$(F \leftrightarrow G)$	$\equiv$	$(F \wedge G) \vee (\neg F \wedge \neg G)$	
$(F \to G)$	$\equiv$	$(\neg F \vee G)$	Elimination von $\to$
$\neg(F \wedge G)$	$\equiv$	$(\neg F \vee \neg G)$	deMorgan
$\neg(F \vee G)$	$\equiv$	$(\neg F \wedge \neg G)$	
$(F \wedge \top)$	$\equiv$	F	Tautologieregeln
$(F \vee \top)$	$\equiv$	$\top$	
$(F \wedge \bot)$	$\equiv$	$\bot$	Unerfüllbarkeitsregeln
$(F \vee \bot)$	$\equiv$	F	
$(F \wedge \neg F)$	$\equiv$	$\bot$	Komplement
$(F \vee \neg F)$	$\equiv$	$\top$	

8.4 Normalformen

Ein **Literal** ist entweder ein Atom (ein **positives** Literal) oder die Negation eines Atoms (ein **negatives** Literal). Zwei Literale heißen **komplementär**, wenn das eine das positive und das andere das negative Literal von dem selben Atom ist. Beispielsweise wären A und nicht $\neg A$ komplementäre Literale

Eine **Klausel** ist eine Menge an Literalen, die entweder ausschließlich durch Konjunktionen ($\wedge$) oder ausschließlich durch Disjunktionen ($\vee$) verbunden ist. Eine Klausel von Konjunktionen wird auch als **duale Klausel** bezeichnet.

Zu jeder Formel F aus $\mathcal{L}_{AL}$ gibt es mindestens eine Konjunktive Normalform (K) und mindestens eine Disjunktive Normalform (D) mit: $F \equiv K \equiv D$

8.4.1 Konjunktive Normalform

Eine Formel F ist in konjunktiver Normalform, falls sie eine Konjunktion von Disjunktionen von Literalen ist. Dabei sind die Disjunktionen die Klauseln.

Beispiel:

$$(A \vee B) \wedge \neg B \wedge (C \vee \neg A \vee D) \wedge D$$
$$A \wedge B$$
$$A \vee B$$

Eine KNF ist eine Tautologie, genau dann wenn alle ihre Klauseln eine Tautologie sind. Dies ist der Fall, gdw. in allen Klauseln mindestens ein paar komplementärer Literale vorkommt.

8.4.2 Resolution

Resolution ist ein Weg um zu zeigen, dass eine Formel (un)erfüllbar ist, sie lässt sich aber nur auf eine Formel in KNF anwenden.

$$\begin{aligned} Res(F) &:= F \cup \{R \mid R \text{ ist Resolvente zweier Klauseln aus F}\} \\ Res^0(F) &:= F \\ Res^{n+1}(F) &:= Res(Res^n(F)) \\ Res^*(F) &:= \bigcup_{i \geq 0} Res^i(F) \end{aligned}$$

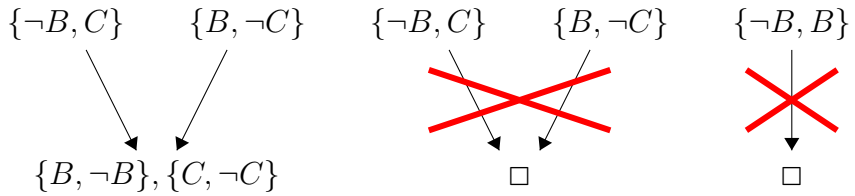
Wenn bei der Resolution an irgendeinem Punkt die leere Klausel ($\square$) rauskommt, ist die Formel **unerfüllbar**, kommen nach einem Resolutionsschritt keine weiteren Resolventen mehr dazu, ist die Formel **erfüllbar**.

Korrektheit: Wenn $\square \in Res^*(F)$, dann ist F unerfüllbar.

Vollständigkeit: Wenn F unerfüllbar ist, dann ist $\square \in Res^*(F)$.

Resolutionslemma:
Sei F eine Formel in KNF und R eine Resolvente zweier Klauseln K_1 und K_2 in F. Dann sind F und $F \cup \{R\}$ äquivalent.

Pro Resolution zweier Resolventen ist es nur möglich, ein Paar komplementerer Literale zu resolvieren:



Beispiel zum Aufstellen der 'nullten' Resolution

$$F = \underbrace{(A \wedge B) \vee (C \wedge D)}_{DNF} \equiv \underbrace{(A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)}_{KNF}$$

$$Res^0(F) = \{\{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}\}$$

Beispiel zum Zeigen der Erfüllbarkeit

$$
\begin{aligned}
G &= \neg B \wedge (A \vee B \vee C) \wedge (\neg A \vee C) \wedge (\neg A \vee \neg B) \\
Res^0(G) &= \{\{\neg B\}, \{A, B, C\}, \{\neg A, C\}, \{\neg A, \neg B\}\} \\
Res^1(G) &= \{\{\neg B\}, \{A, B, C\}, \{\neg A, C\}, \{\neg A, \neg B\}, \{A, C\}, \{B, C\}, \{A, \neg A, C\}, \\
&\quad \{B, \neg B, C\}\} \\
Res^2(G) &= \{\{\neg B\}, \{A, B, C\}, \{\neg A, C\}, \{\neg A, \neg B\}, \{A, C\}, \{B, C\}, \{A, \neg A, C\}, \\
&\quad \{B, \neg B, C\}, \{C\}, \{\neg B, C\}, \{\neg A, \neg B, C\}\} \\
Res^3(G) &= \{\{\neg B\}, \{A, B, C\}, \{\neg A, C\}, \{\neg A, \neg B\}, \{A, C\}, \{B, C\}, \{A, \neg A, C\}, \\
&\quad \{B, \neg B, C\}, \{C\}, \{\neg B, C\}, \{\neg A, \neg B, C\}\} \\
\ldots &\quad \text{Keine weiteren Resolventen, daher ist } G \text{ erfüllbar.}
\end{aligned}
$$

Beispiel zum Zeigen der Unerfüllbarkeit

$$H = \{\{D, \neg E\}, \{\neg A\}, \{\neg B, \neg C\}, \{A, C, E\}, \{A, \neg C, D\}, \{B, \neg D\}, \{C, \neg D\}\}$$

$K_1 = \{D, \neg E\}$	Klausel aus H
$K_2 = \{A, C, E\}$	Klausel aus H
$K_3 = \{A, C, D\}$	Resolvent aus K_1, K_2
$K_4 = \{A, \neg C, D\}$	Klausel aus H
$K_5 = \{A, D\}$	Resolvent aus K_3, K_4
$K_6 = \{\neg B, \neg C\}$	Klausel aus H
$K_7 = \{B, \neg D\}$	Klausel aus H
$K_8 = \{\neg C, \neg D\}$	Resolvent aus K_6, K_7
$K_9 = \{C, \neg D\}$	Klausel aus H
$K_{10} = \{\neg D\}$	Resolvent aus K_8, K_9
$K_{11} = \{A\}$	Resolvent aus K_5, K_{10}
$K_{12} = \{\neg A\}$	Klausel aus H
$K_{13} = \square$	Resolvent aus K_{11}, K_{12}

8.4.3 Disjunktive Normalform

Eine Formel F ist in disjunktiven Normalform, falls sie eine Disjunktion von Konjunktionen von Literalen ist. D.h. alle Klauseln sind ausschließlich durch Disjunktionen verbunden, wobei die Literale in den Klauseln ausschließlich durch Konjunktionen verbunden sind.

Beispiel:

$$(A \wedge B) \vee \neg B \vee (C \wedge \neg A \wedge D) \vee D$$
$$A \wedge B$$
$$A \vee B$$

Eine DNF ist eine Kontradiktion gdw. alle ihre (dualen) Klauseln unerfüllbar sind. Dies ist der Fall, gdw. in allen Klauseln mindestens ein Paar komplementärer Literale vorkommt.

8.4.4 Umformungen

Formen sie die Formel $\neg(A \to B) \vee C$ in KNF und DNF um.

Wahrheitstafelmethode:

A	B	C	$(A \to B)$	$\neg(A \to B)$	$\neg(A \to B) \vee C$	Klausel
0	0	0	1	0	0	$(A \vee B \vee C)$
0	0	1	1	0	1	$(\neg A \wedge \neg B \wedge C)$
0	1	0	1	0	0	$(A \vee \neg B \vee C)$
0	1	1	1	0	1	$(\neg A \wedge B \wedge C)$
1	0	0	0	1	1	$(A \wedge \neg B \wedge \neg C)$
1	0	1	0	1	1	$(A \wedge \neg B \wedge C)$
1	1	0	1	0	0	$(\neg A \vee \neg B \vee C)$
1	1	1	1	0	1	$(A \wedge B \wedge C)$

Resultat:
KNF: $(A \vee B \vee C) \wedge (A \vee \neg B \vee C) \wedge (\neg A \vee \neg B \vee C)$
DNF: $(\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge C) \vee (A \wedge \neg B \wedge \neg C) \vee (A \wedge \neg B \wedge C) \vee (A \wedge B \wedge C)$

Äquivalenzumformung

$$\begin{aligned}
\neg(A \to B) \vee C &\equiv \neg(\neg A \vee B) \vee C && \text{Definition der Implikation} \\
&\equiv \underbrace{(A \wedge \neg B) \vee C}_{DNF} && \text{deMorgan} \\
&\equiv \underbrace{(A \vee C) \wedge (\neg B \vee C)}_{KNF} && \text{Distributivität}
\end{aligned}$$

8.4.5 Hornformeln

Hornformeln sind eine besondere Form der KNF, bei der die einzelnen Klauseln maximal ein positives Literal enthalten dürfen. Hornformeln können anschaulicher als Konjunktionen von Implikationen geschrieben werden.

Beispiel:

$$\begin{aligned} & A \wedge (\neg B \vee C \vee \neg D) \wedge \neg C \\ \equiv\ & (1 \to A) \wedge ((B \wedge D) \to C) \wedge (C \to 0) \\ \equiv\ & (\top \to A) \wedge ((B \wedge D) \to C) \wedge (C \to \bot) \end{aligned}$$

Hierbei stehen 1 bzw. $\top$ für eine beliebige Tautologie (gültige Formel) und 0 bzw. $\bot$ für eine beliebige Kontradiktion (unerfüllbare Formel).

Markierungsalgorithmus

Eingabe: eine Hornfomel F,

1	Falls es in F eine Teilformel der Form $(1 \to A)$ gibt, versehe jedes Vorkommen der atomaren Formel A in F mit einer Markierung;	
2	**while**	es gibt eine Teilformel G der Form $(A_1 \wedge \cdots \wedge A_n \to B)$, wobei A_1 bis A_n bereits markiert oder gleich 1 sind und B unmarkiert ist.
	do	**if** G hat die genannte Form **then** markiere jedes Vorkommen von B **else** gib "unerfüllbar" aus und stoppe;
3	**if**	Es gibt eine Teilformel H der Form $A_1 \wedge \cdots \wedge A_n \to 0$ wobei A_1 bis A_n bereits markiert sind
	then	gib "unerfüllbar" aus und stoppe;
	else	gib "erfüllbar" aus und stoppe. (Die erfüllende Belegung wird hierbei durch die Markierung gegeben: $\mathcal{A}(A_i) = 1$ gdw. A_i hat eine Markierung.)

$$\begin{aligned} & A \wedge (\neg A \vee \neg B \vee C) \wedge (\neg A \vee B) \\ \equiv\ & (1 \to A) \wedge (A \wedge B \to C) \wedge (A \to B) \\ \equiv\ & (1 \to A) \wedge (A \wedge B \to C) \wedge (A \to B) \\ \equiv\ & (1 \to A) \wedge (A \wedge B \to C) \wedge (A \to B) \\ \equiv\ & (1 \to A) \wedge (A \wedge B \to C) \wedge (A \to B) \end{aligned}$$

Eine Belegung, die die Formel erfüllt, wäre nach dem Markierungsalgorithmus:

$$\mathcal{A}(A) = 1, \mathcal{A}(B) = 1, \mathcal{A}(C) = 1$$

9. Glossar

Bezeichnung	Beschreibung
$\mathcal{L}_{AL}$	Menge aller Formeln der Aussagenlogik
ε-Produktion	Ein Produktion (Regel) mit $A \to \varepsilon$ wird als ε-Produktion bezeichnet
ε-frei	Eine Kontextfreie Grammatik, die keine ε-Produktionen besitzt wird als ε-frei bezeichnet
AS_{AL}	Menge aller Aussagensymbole (Atome) der Aussagenlogik
Alphabet	Endliche, nicht leere Menge an Objekten, genannt Symbole
Äquivalenzrelation	Eine binäre Relation die reflexiv, symmetrisch und transitiv ist
Argument	Eine Eingabe einer Funktion
Baum	Ein zusammenhängender Graph ohne einfache Zyklen
Bildbereich	Die Menge aus der die Ausgaben einer Funktion gewählt werden
Binäre Relation	Eine Relation dessen Urbildmenge eine Menge aus Paaren ist
Bool'sche Operation	Eine Operation auf bool'schen Werten
Bool'scher Wert	Die Werte TRUE und FALSE, häufig auch durch 0 und 1 repräsentiert
CFG	Context Free Grammar
DFA	Deterministic Finite Automata
Disjunktion	Bool'sche ODER Operation
DNF/Disjunktive Normalform	Eine Disjunktion von Konjunktionen von Literalen
Ecke	Ein Punkt in einem Graphen
Eigenschaft	Ein Prädikat
Einfacher Pfad	Ein Pfad ohne Wiederholung
Element	Ein Objekt in einer Menge
Erfüllbar	Eine Aussage dessen Wahrheitswert bei wenigstens einer Belegung 1 ist
Falsifizierbar	Es gibt eine Belegung, so dass der Wahrheitswert der Aussage 0 ist

Bezeichnung	**Beschreibung**
Forced Handle	der eindeutige Handle von jedem gültigen String mit dem Anfang, unabhängig von was hinter dem Handle noch steht
Funktion	Eine Operation die Eingaben in Ausgaben übersetzt
Geordnetes Paar	Eine Liste aus zwei Elementen (ein 2-Tupel)
Gerichteter Graph	Eine Sammlung von Punkten und Pfeilen die Paare von Punkten verbinden
Graph	Eine Sammlung von Punkten und Linien die Paare von Punkten verbinden
Handle	ein Reduktionsstring gemeinsam mit der Regel des dazugehörigen Reduktionsschritts (nur definiert für gültige Strings)
Hornformel	Eine Formel in KNF, wobei jede Klausel maximal ein positives Literal enthält
Junktoren	Operatoren der Junktorenlogik: Negation ($\neg$), Konjunktion ($\wedge$), Disjunktion ($\vee$), Implikation ($\rightarrow$, Äquivalenz/Biimplikation ($\leftrightarrow$)
Klausel	Eine Disjunktion oder Konjunktion von Literalen
KNF/Konjunktive Normalform	Eine Konjunktion von disjunktiven Klauseln
Knoten	Ein Punkt in einem Graphen
Kontingent	Eine Aussage dessen Wahrheitswert bei wenigstens einer Belegung 1 und wenigstens einer Belegung 0 ist
Kontradiktion/unerfüllbar	Eine Aussage dessen Wahrheitswert unabhängig von der Belegung 0 ist
k-Tupel	Eine Liste von k Objekten
Kante	Eine Linie in einem Graphen
Kartesisches Produkt	Eine Operation auf Mengen die eine Menge aus allen Tupeln der Elemente der Mengen erzeugt
Komplement	Eine Operation auf eine Menge die eine Menge aus allen nicht vorhandenen Elementen erzeugt
Konjunktion	Bool'sche UND Operation
Konkatenation	Eine Operation die Wörter aneinanderreiht
Leere Menge $\emptyset$	Ein Objekt in einer Menge
Leeres Wort ε	Ein Wort mit der Länge null
Literal	Atom oder dessen Negierung
Menge	Eine Gruppe von Objekten

Bezeichnung	**Beschreibung**
NFA	Nondeterministic Finite Automata
PDA	PushDown Automata
Pfad	Eine Sequenz von Knoten in einem Graphen die durch Kanten verbunden sind
Prädikat	Eine Funktion mit dem Bildbereich {TRUE, FALSE}
Relation	Ein Prädikat, typischerweise wenn die Urbildmenge eine Menge an k-Tupeln ist
Schnitt	Eine Operation auf Mengen die eine Menge aus allen gemeinsamen Elementen erzeugt
Sequenz	Eine Liste von Objekten
Sprache	Eine Menge an Wörtern
Symbol	Ein Element eines Alphabets
Tautologie/Gültig	Eine Aussage dessen Wahrheitwert unabhängig von der Belegung 1 ist
Ungeordnetes Paar	Eine Menge mit zwei Elementen
Urbildmenge	Die Menge aller möglichen Eingaben einer Funktion, auch: Quellmenge
Vereinigung	Eine Operation auf Mengen die alle Elemente in eine Menge kombiniert
Wort	Eine endliche Liste an Symbolen aus einem Alphabet
Zusammenhängender Graph	Ein Graph mit Pfaden die alle Knoten paarweise verbinden
Zyklus	Ein Pfad der im selben Knoten startet und endet