

ETI Zusammenfassung

Hier sollte alles drinstehen, was in ETI (SoSe 2021) vorkam und Klausurrelevant ist.

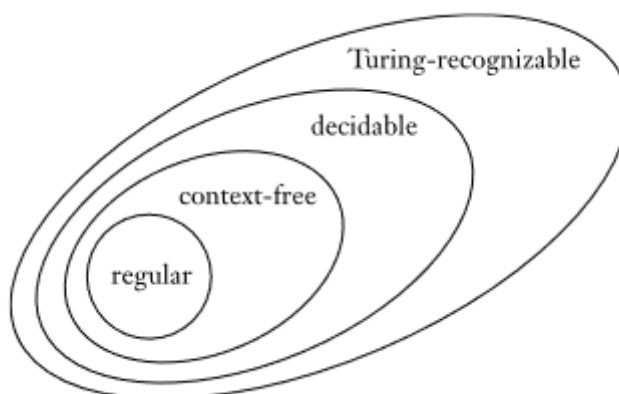
Inhaltsverzeichnis

Chomsky Sprachhierarchie	3
Sprachen	4
Mengenrechnung	4
Reguläre Ausdrücke.....	4
Leere Wörter + Sprachen	4
Abgeschlossenheit.....	4
[Typ 3] Reguläre Sprachen.....	5
DFA	5
NFA	5
Abgeschlossenheit DFA und NFA	6
Umwandlung NFA zu DFA.....	7
GNFA.....	7
Pumping-Lemma für reguläre Sprachen	8
[Typ 2] Kontextfreie Sprachen	9
Kontextfreie Grammatiken.....	9
Chomsky Normalform	9
PDA	10
Pumping-Lemma für kontextfreie Sprachen	10
Abgeschlossenheit Kontextfreier Sprachen	11
Deterministische Kontextfreie Sprachen	11
DPDA.....	11
Abgeschlossenheit von DCFL.....	11
Endmarkierte Wörter	11

Logik	12
Logische Verknüpfungen	12
Belegungen	12
Gültigkeit, Erfüllbarkeit, Unerfüllbarkeit	13
Äquivalenz von Formeln	13
Formelsätze	13
Normalformen	14
Hornformeln	14
Resolution	15
 [Typ 0] Turing Maschinen	 16
Turing Maschinen	16
Erkennbare / Entscheidbare Sprachen	17
Turingmaschinenvarianten	17
Abgeschlossenheit von TM	18
Algorithmen	18
 Entscheidbarkeit	 19
Entscheidbarkeit regulärer Sprachen [Typ 3]	19
Entscheidbarkeit kontextfreier Sprachen [Typ 3]	20





Chomsky Sprachhierarchie

Sprachfamilie	Automaten	Grammatik	Beispiel	\cup	\cap	$\bar{\cdot}$
endl. Mengen	–	–	$\{a, ab, abb\}$	+	+	–
$\mathcal{L}_3 = \mathcal{R}eg$	DFA=NFA	Typ-3 = rechts-lineare G.	$\{a\}^*\{b\}^*$	+	+	+
$det\mathcal{C}f$	DPDA	$LR(k)$, $k \geq 1$	$\{a^n b^n \mid n \geq 0\}$ $\{w c w^{rev} \mid w \in \{a, b\}^*\}$	–	–	+
$\mathcal{L}_2 = \mathcal{C}f$	PDA	Typ-2 = kontextfreie G.	$\{w w^{rev} \mid w \in \{a, b\}^*\}$	+	–	–
$\mathcal{L}_1 = \mathcal{C}s$	NLBA	Typ-1 = monotone G.	$\{a^n b^n c^n \mid n \geq 0\}$	+	+	+
$\mathcal{R}ec$	–	–	L_e	+	+	+
$\mathcal{L}_0 = \mathcal{R}e$	DTM = NTM	Typ-0	$H, (G^* \setminus L_d)$	+	+	–
abzählbare Mengen	–	–	L_d, \mathbb{N}			



Sprachen

Mengenrechnung

$A \cup B$		<i>Vereinigung</i>	Alle Elemente, die entweder in A oder B sind.
$A \cap B$		<i>Schnittmenge</i>	Alle Elemente, die in A und B sind.
$A \setminus B$		<i>Differenz</i>	Alle Elemente, die in A und nicht in B sind.
\bar{A}		<i>Komplement</i>	Alle Elemente, die nicht in A sind.

Reguläre Ausdrücke

Reguläre Ausdrücke beschreiben Sprachen (Mengen von Wörtern). Sie bauen auf den Mengenoperationen auf und fügen weitere nichtmathematische Operationen hinzu.

a^*	ϵ, a, aa, \dots	<i>Stern</i>	Beliebig viele a's, ϵ ist erlaubt.
a^+	a, aa, \dots	<i>Plus</i>	Mehr als ein a, ϵ ist <u>nicht</u> erlaubt.
$a \circ b$	ab	<i>Konkatenation</i>	a gefolgt auf b.

Viele Sprachen lassen sich als reg. Ausdrücke beschreiben.

$\{aa\}^*$ Eine grade Anzahl von a's

$\{b, ab\}^*$ Auf jedes a folgt ein b

Leere Wörter + Sprachen

ϵ	<i>Das leere Wort</i>	Ein Wort ohne Zeichen (Länge 0)
\emptyset	<i>Die leere Sprache</i>	Eine Sprache ohne Wörter (oder allgemein, eine leere Menge).

Wichtig! ϵ ist nicht das gleiche wie \emptyset

Sei R eine reguläre Sprache:

$$R \cup \emptyset = R$$

$$R \cup \epsilon \text{ ist nicht unbedingt } R$$

$$R \circ \emptyset \text{ ist nicht unbedingt } R$$

$$R \circ \epsilon = R$$

Abgeschlossenheit

Falls nach einer Sprachoperation eine Sprache immer noch in einer bestimmten Sprachklasse liegt, so sagt man, dass die Sprachklasse auf der Operation **abgeschlossen** ist.

Bsp.: Reguläre Sprachen sind auf dem Komplement abgeschlossen. Nimmt man das Komplement einer regulären Sprache, erhält man wieder eine reguläre Sprache.

[Typ 3] Reguläre Sprachen

DFA's und NFA's sind gleichwertige Automatenmodelle, die beide reguläre Sprachen beschreiben. Eine Sprache ist außerdem regulär, wenn ein DFA/NFA existiert, der sie akzeptiert.

Die Sprachklasse der DFA/NFA sind die regulären Sprachen **REG**.

DFA

Ein sehr simpler Automat, formell ein 5-Tupel (runde Klammern!).

$$\text{DFA} = (Q, \Sigma, \delta, q_0, F)$$

Q	<i>Zustandsmenge</i>	Alle Zustände des Automaten
Σ	<i>Eingabesymbole</i>	Alle Symbole, die der Automat versteht
δ	<i>Übergangsfunktion</i>	$\delta: Q \times \Sigma \rightarrow Q$ Gibt neuen Zustand wieder.
q_0	<i>Startzustand</i>	Zustand, in dem der Automat startet.
F	<i>Endzustände</i>	Zustände, in denen der Automat akzeptiert.

Beispiel:

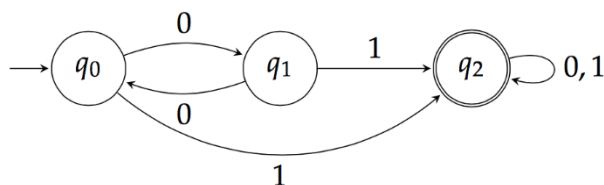
$$Q = \{q_0, q_1, q_2\}$$

$$\Sigma = \{1, 2\}$$

$$\delta: \begin{array}{l} q_0 \times 0 \rightarrow q_1, \\ q_0 \times 1 \rightarrow q_2, \\ q_1 \times 0 \rightarrow q_0, \\ q_1 \times 1 \rightarrow q_2, \\ q_2 \times 0 \rightarrow q_2, \\ q_2 \times 1 \rightarrow q_2 \end{array}$$

$$q_0 = q_0$$

$$F = \{q_2\}$$



NFA

Ein nichtdeterministischer endlicher Automat ist ein endlicher Automat, bei dem es für den Zustandsübergang mehrere gleichwertige Möglichkeiten gibt. Dabei wird eine Menge von Automaten simuliert.

Formell ist ein NFA ein 5-Tupel, ähnlich zum DFA.

Q	<i>Zustandsmenge</i>	Alle Zustände des Automaten
Σ	<i>Eingabesymbole</i>	Alle Symbole, die der Automat versteht
δ	<i>Übergangsfunktion</i>	$\delta: Q \times \Sigma \rightarrow P(Q)$ Gibt neue Zustände wieder.
q_0	<i>Startzustand</i>	Zustand, in dem der Automat startet.
F	<i>Endzustände</i>	Zustände, in denen der Automat akzeptiert.

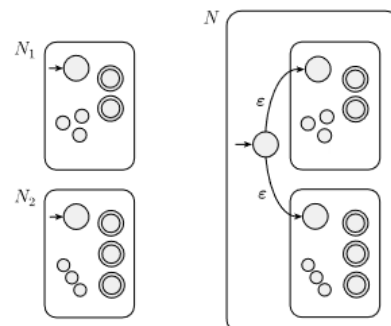
Abgeschlossenheit DFA und NFA

Reguläre Sprachen sind gegenüber \cup Vereinigung, \cap Schnittmenge, \bar{A} Komplement, \circ Konkatenation und $*$ Stern abgeschlossen.

Seien A und B DFA:

$A \cup B$

Simuliere A und B und akzeptiere, falls A oder B in einem Akzeptanzzustand ist.



$A \cap B$

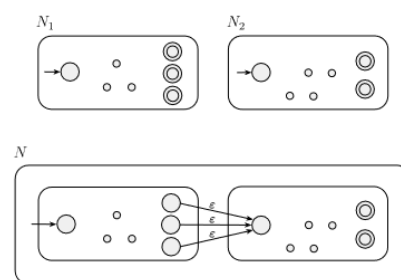
Simuliere A und B und akzeptiere, falls A und B in einem Akzeptanzzustand sind.

\bar{A}

Alle Nichtakzeptanzzustände werden Akzeptanzzustände und umgekehrt.

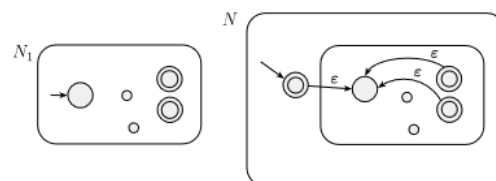
$A \circ B$

Simuliere A bis zu einem Akzeptanzzustand, danach B.



A^*

Füge einen extra Startzustand, der gleichzeitig ein Endzustand ist hinzu. Alle Endzustände des Automaten führen über eine Epsilonkante zum ehemaligen Startzustand.



Umwandlung NFA zu DFA

NFAs können in äquivalente DFAs umgewandelt werden. Dabei wird zuerst die Potenzmenge der Zustandsmenge Q des NFAs berechnet. Jedes Element der Potenzmenge wird zu einem Zustand des DFAs, darunter auch die leere Menge.

A ist ein NFA, B ein äquivalenter DFA.

$$Q_A = \{q_0, q_1, q_2\}$$

$$P(Q_A) = Q_B = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$$

Das Eingabealphabet Σ bleibt gleich. Die Zustandsübergangsfunktion δ muss angepasst werden. Der Startzustand q_0 ist die Menge mit nur dem Startzustand von A. Die Endzustandsmenge F ist die Menge aller Zustände, die mindestens einen Endzustand von A enthalten.

GNFA

GNFAs bieten die Möglichkeit, direkt den regulären Ausdruck eines Automaten zu bestimmen. Dabei können an den Kanten von GNFA ganze reguläre Ausdrücke stehen.

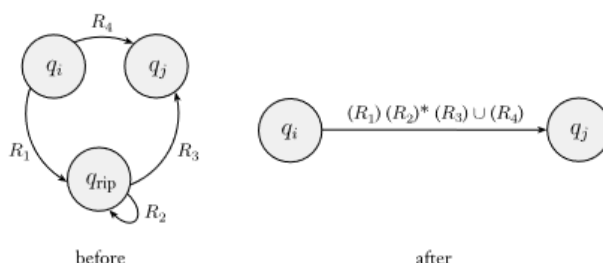
Formell ist ein GNFA ein 5-Tupel:

$$\text{GNFA} = (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$$

Q	<i>Zustandsmenge</i>	Alle Zustände des Automaten
Σ	<i>Eingabesymbole</i>	Alle Symbole, die der Automat versteht
δ	<i>Übergangsfunktion</i>	$\delta: (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \rightarrow R$ Nimmt zwei Zustände und gibt den regulären Ausdruck wieder, um vom ersten zum zweiten Zustand zu kommen.
q_{start}	<i>Startzustand</i>	Zustand, in dem der Automat startet.
q_{accept}	<i>Endzustand</i>	Zustand, in dem der Automat akzeptiert.

So wandelt man einen DFA in einen GNFA um, um später den regulären Ausdruck abzulesen:

1. Füge einen neuen Anfangszustand q_{start} und Endzustand q_{accept} hinzu.
2. q_{start} hat Pfeile die zu jedem anderen Zustand gehen, q_{accept} hat Pfeile, die von jedem Zustand auf ihn zeigen. Jeder andere Zustand hat Pfeile zu jedem anderen „normalen“ Zustand und sich selbst. Alle neuen Pfeile haben ϵ als Beschriftung.
3. Entferne einen „normalen“ Zustand (oft q_{rip} genannt) und passe alle anderen regulären Ausdrücke an den Pfeilen an.
4. Wiederhole 3. bis nur noch q_{start} und q_{accept} übrig sind. Der reguläre Ausdruck des Automaten kann am Übergang abgelesen werden.



Dabei hat der DFA am Anfang n Zustände, bei Schritt 1. $n+2$ Zustände, verliert bei 3. jeweils einen Zustand, und endet in 4. mit 2 Zuständen.

Pumping-Lemma für reguläre Sprachen

Mittels des Pumping-Lemma für reguläre Sprachen kann überprüft werden, ob eine Sprache nicht regulär ist.

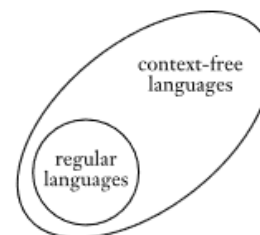
Falls **A** eine reguläre Sprache ist, existiert eine Zahl **p**, wobei jedes Wort **s** aus **A**, das länger als **p** ist, in **s = xyz** aufgeteilt werden kann. Für **x, y, z** gibt es jedoch folgende Bedingungen:

1. Für jedes $i \geq 0$ ist xy^iz auch ein Wort in **A**
2. $|y| > 0$
3. $|xy| \leq p$

Nun überprüft man jede mögliche Aufteilung der Wörter von **A** (die länger als **p** sind) in **xyz** und schaut, ob die Bedingung gehalten werden. Wenn bei jeder möglichen Aufteilung mindestens eine Bedingung gebrochen wird, ist **A** nicht regulär.

[Typ 2] Kontextfreie Sprachen

Die Sprachklasse der kontextfreien Sprachen **CFG** ist mächtiger als **REG**. Jede reguläre Sprache ist eine kontextfreie Sprache.



Kontextfreie Grammatiken

Eine Grammatik ist eine Menge von Produktionsregeln. Eine Regel beschreibt, wie ein **Nonterminal** (linke Seite) mit einem Wort bestehend aus **Nonterminalen** und **Terminalen** (rechte Seite) ersetzt wird. Für ein Nonterminal kann es mehrere Regeln geben.

Bsp: $A \rightarrow a \mid aA \mid B$
 $B \rightarrow b \mid bB \mid \varepsilon$

In diesem Beispiel sind **A** und **B** Nonterminale, und **a** und **b** Terminale.

A kann beispielsweise in a , aa , $aabb$, $aabbbbbbb$ abgeleitet werden, aber nicht in ba .

Alle Wörter die so generiert werden können bilden die **Sprache der Grammatik**. Für eine Grammatik **G** ist die Sprache der Grammatik **L(G)**.

Formell: Eine Kontextfreie Grammatik ist ein 4-Tupel (V, Σ, R, S) :

V	<i>Variablen</i>	Eine endliche Menge Variablen (Nonterminale).
Σ	<i>Terminale</i>	Eine endliche Menge Terminale
R	<i>Regeln</i>	Eine endliche Menge Produktionsregeln.
S	<i>Startvariable</i>	$S \in V$, die Startvariable.

Falls eine Grammatik ein Wort auf mehrere Arten generieren kann, nennen wir dieses Wort *mehrdeutig ableitbar*. Falls eine Grammatik mindestens ein *mehrdeutig ableitbares* Wort hat, ist die Grammatik selbst *mehrdeutig*.

Chomsky Normalform

Eine spezielle Grammatikform. Jede Grammatik kann in eine äquivalente Chomsky Normalform umgewandelt werden.

Dabei darf es nur Regeln dieser Form geben:

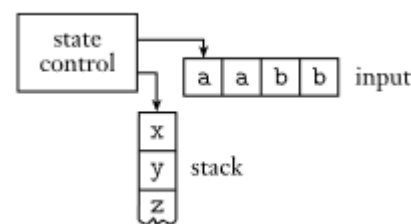
$A \rightarrow BC$ A, B, C sind Variablen, B und C dürfen nicht die Startvariable sein.
 $A \rightarrow a$ a ist ein Terminal.

Außerdem ist folgende Regel erlaubt:

$S \rightarrow \varepsilon$ S ist die Startvariable.

PDA

Ein neues Automatenmodell ähnlich zum NFA. Dieser Automat besitzt hingegen noch einen **Keller** (Stack), in den Symbole geschrieben und abgelesen werden können.



Achtung! Dieser Automat ist nichtdeterministisch.

Formell: Ein PDA ist ein 6-Tupel $(Q, \Sigma, \Gamma, \delta, q_0, F)$:

Q	<i>Zustände</i>	Eine endliche Menge von Zuständen.		
Σ	<i>Inputalphabet</i>	Endliche Zeichenmenge, können in Inputwort vorkommen.		
Γ	<i>Stackalphabet</i>	Endliche Zeichenmenge, können auf Stack geschrieben werden.		
δ	<i>Übergangsfunktion</i>	$\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow P(Q \times \Gamma_\epsilon)$	Gibt neue Zustände wieder.	
q_0	<i>Startzustand</i>	$q_0 \in Q$	Der Startzustand.	
F	<i>Endzustände</i>	$F \subseteq Q$	Mögliche Endzustände.	

Wird ein Zeichen auf den Stack geschrieben, wird es nach ganz oben geschrieben und alle anderen Stackzeichen werden eins nach hinten verschoben (**Push**). Es wird immer das oberste Zeichen gelesen. Wird ein Zeichen gelesen, wird dieses vom Stack entfernt und alle folgenden um eins nach vorne gerückt (**Pop**).

In der Übergangsfunktion wird jeweils der momentane Zustand, das nächste Inputzeichen und das oberste Stackzeichen gelesen, und eine Menge von Zuständen und Stackzeichen zurückgegeben. Wird ein Stackzeichen zurückgegeben, wird es auf den Stack gepusht. Wird ein ϵ gelesen oder geschrieben, passiert an dieser Stelle nichts mit dem Stack.

Wichtig! Hier sind Epsilonanten erlaubt.

PDAs sind gleichmächtig zu Kontextfreien Grammatiken. Eine Sprache ist also kontextfrei falls, und nur falls, ein PDA sie erkennt.

Pumping-Lemma für kontextfreie Sprachen

Das Pumping-Lemma für kontextfreie Sprachen kann genutzt werden, um zu zeigen dass eine Sprache nicht kontextfrei ist.

Falls **A** eine kontextfreie Sprache ist, existiert eine Zahl **p**, wobei jedes Wort **s** aus **A**, das länger als **p** ist, in **s = uvxyz** aufgeteilt werden kann. Für **u, v, x, y, z** gibt es jedoch folgende Bedingungen:

1. Für jedes $i \geq 0$ ist $uv^i xy^i z$ auch ein Wort in **A**
2. $|vy| > 0$
3. $|vxy| \leq p$

Nun überprüft man jede mögliche Aufteilung der Wörter von **A** (die länger als **p** sind) in **uvxyz** und schaut, ob die Bedingung gehalten werden. Wenn bei jeder möglichen Aufteilung mindestens eine Bedingung gebrochen wird, ist **A** nicht kontextfrei.

Abgeschlossenheit Kontextfreier Sprachen

Kontextfreie Sprachen sind gegenüber \cup Vereinigung abgeschlossen, jedoch nicht gegenüber \cap Schnittmengen- und \bar{A} Komplementbildung.

Deterministische Kontextfreie Sprachen

Wir wissen das DFA und NFA gleichmächtig sind.

Jedoch sind PDA und DPDA (deterministische PDA) nicht gleichmächtig!

DPDA erkennen ihre eigene Sprachklasse, die **DCFL** (Deterministische Kontextfreie Sprachen). Diese ist mächtiger als **REG**, aber eine Teilmenge der **CFL**.

DPDA

Die deterministische Variante des PDAs. Der Unterschied ist, dass ein DPDA jeweils nur einen Zustand aus der Übergangsfunktion δ zurückgibt.

Formell: DPDAs sind 6-Tupel $(Q, \Sigma, \Gamma, \delta, q_0, F)$:

Q	<i>Zustände</i>	Eine endliche Menge von Zuständen.	
Σ	<i>Inputalphabet</i>	Endliche Zeichenmenge, können in Inputwort vorkommen.	
Γ	<i>Stackalphabet</i>	Endliche Zeichenmenge, können auf Stack geschrieben werden.	
δ	<i>Übergangsfunktion</i>	$\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow (Q \times \Gamma_\epsilon) \cup \{\emptyset\}$	Gibt neuen Zustand wieder.
q_0	<i>Startzustand</i>	$q_0 \in Q$	Der Startzustand.
F	<i>Endzustände</i>	$F \subseteq Q$	Mögliche Endzustände.

Außerdem muss für die Übergangsfunktion δ folgendes gelten:

Für jedes $q \in Q$, $a \in \Sigma$, und $x \in \Gamma$ muss genau einer der Werte $\delta(q, a, x)$, $\delta(q, a, \epsilon)$, $\delta(q, \epsilon, x)$, oder $\delta(q, \epsilon, \epsilon)$ nicht \emptyset sein.

\Rightarrow Alle anderen sind demnach \emptyset (sonst würde DPDA nichtdeterministisch sein)

Abgeschlossenheit von DCFL

DCFL sind gegenüber \bar{A} Komplementbildung abgeschlossen, jedoch nicht gegenüber \cup Vereinigung, \cap Schnittmengenbildung, $*$ Stern und Umkehrung (Reverse).

Endmarkierte Wörter

Endmarkierte Wörter sind Wörter, an denen das Endmarkierungssymbol $\#$ angehängen ist ($w \#$). Auch Sprachen können endmarkiert werden, dabei wird einfach jedes Wort der Sprache endmarkiert. Diese Sprachen werden **endmarkierte Sprachen** genannt.

Eine Sprache **A** ist nur eine DCFL, falls und nur falls auch **A $\#$** eine DCFL ist.

Logik

Atomare Formeln können entweder **1** oder **0** als Wahrheitswert besitzen.

Formeln werden durch Verknüpfung atomarer Formeln gebaut und können selbst zu einem Wahrheitswert ausgewertet werden.

Logische Verknüpfungen

V	<i>Disjunktion (Oder)</i>	<table> <tr> <th>$\mathcal{A}(F)$</th><th>$\mathcal{A}(G)$</th><th>$\mathcal{A}((F \vee G))$</th></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}((F \vee G))$	0	0	0	0	1	1	1	0	1	1	1	1
$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}((F \vee G))$															
0	0	0															
0	1	1															
1	0	1															
1	1	1															

Λ	<i>Konjunktion (Und)</i>	<table> <tr> <th>$\mathcal{A}(F)$</th><th>$\mathcal{A}(G)$</th><th>$\mathcal{A}((F \wedge G))$</th></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}((F \wedge G))$	0	0	0	0	1	0	1	0	0	1	1	1
$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}((F \wedge G))$															
0	0	0															
0	1	0															
1	0	0															
1	1	1															

\neg	Negation (Nicht)	<table> <tr> <th>$\mathcal{A}(F)$</th> <th>$\mathcal{A}(\neg F)$</th> </tr> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </table>	$\mathcal{A}(F)$	$\mathcal{A}(\neg F)$	0	1	1	0
$\mathcal{A}(F)$	$\mathcal{A}(\neg F)$							
0	1							
1	0							

\rightarrow	Implikation	<table> <tr> <th>$\mathcal{A}(F)$</th> <th>$\mathcal{A}(G)$</th> <th>$\mathcal{A}((F \rightarrow G))$</th> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </table>	$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}((F \rightarrow G))$	0	0	1	0	1	1	1	0	0	1	1	1	$(\neg A \vee B)$
$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}((F \rightarrow G))$																
0	0	1																
0	1	1																
1	0	0																
1	1	1																

\leftrightarrow	<i>Biimplikation</i>	<table> <tr> <th>$\mathcal{A}(F)$</th> <th>$\mathcal{A}(G)$</th> <th>$\mathcal{A}((F \leftrightarrow G))$</th> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </table>	$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}((F \leftrightarrow G))$	0	0	1	0	1	0	1	0	0	1	1	1	$((A \wedge B) \vee (\neg A \wedge \neg B))$
$\mathcal{A}(F)$	$\mathcal{A}(G)$	$\mathcal{A}((F \leftrightarrow G))$																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Belegungen

Früh in der Aussagenlogik wird **A(F)** genutzt, um den Wahrheitswert einer Formel **F** zu bestimmen. Dabei können wir z.B. mit **A(B) = 1** festlegen dass **B wahr** ist, um später zu bestimmen was **A(¬B)** ist. **A(B) = 1** wird eine *Belegung* von **B** genannt.

Sei **F** irgendeine Formel und **A** eine Belegung. Falls **A** für alle in **F** vorkommenden atomaren Formeln definiert ist, so heißt **A** zu **F** *passend*.

Falls **A** zu **F** passend ist und **A(F) = 1** gilt, schreibt man auch **A ⊨ F**. **A** ist ein *Modell* für **F**.

Falls **A** zu **F** passend ist und **A(F) = 0** gilt, schreibt man auch **A ⊭ F**. **A** ist kein *Modell* für **F**.

Gültigkeit, Erfüllbarkeit, Unerfüllbarkeit

Eine Formel **F** heißt *erfüllbar*, wenn **F** mindestens ein Modell besitzt.

Eine Formel **F** heißt *unerfüllbar*, falls **F** kein einziges Modell besitzt.

Eine Formel **F** heißt *gültig*, wenn jede Belegung von **F** ein Modell ist.

F heißt in diesem Fall auch *Tautologie* und man schreibt $\models \mathbf{F}$.

Falls **F** keine Tautologie ist, schreiben wir $\not\models \mathbf{F}$.

alle aussagenlog. Formeln		
gültige Formeln	erfüllbare, aber nicht gültige Formeln	unerfüllbare Formeln
$\neg G$	F $\neg F$	G

Negation bedeutet Spiegelung um die gestrichelte Linie

Äquivalenz von Formeln

Zwei Formeln **F** und **G** heißen *äquivalent*, falls für alle Belegungen **A**, die für **F** und **G** passend sind, $\mathbf{A(F)} = \mathbf{A(G)}$ gilt. Dann schreibt man auch $\mathbf{F \equiv G}$.

Dies kann auch gezeigt werden, in man die zwei Wahrheitstabellen miteinander vergleicht.

Formelsätze

Es gelten die folgenden Äquivalenzen :

$$\begin{aligned} (F \wedge F) &\equiv F \\ (F \vee F) &\equiv F \end{aligned} \quad \text{(Idempotenz)}$$

$$\begin{aligned} (F \wedge G) &\equiv (G \wedge F) \\ (F \vee G) &\equiv (G \vee F) \end{aligned} \quad \text{(Kommutativität)}$$

$$\begin{aligned} ((F \wedge G) \wedge H) &\equiv (F \wedge (G \wedge H)) \\ ((F \vee G) \vee H) &\equiv (F \vee (G \vee H)) \end{aligned} \quad \text{(Assoziativität)}$$

$$\begin{aligned} (F \wedge (F \vee G)) &\equiv F \\ (F \vee (F \wedge G)) &\equiv F \end{aligned} \quad \text{(Absorption)}$$

$$\begin{aligned} (F \wedge (G \vee H)) &\equiv ((F \wedge G) \vee (F \wedge H)) \\ (F \vee (G \wedge H)) &\equiv ((F \vee G) \wedge (F \vee H)) \end{aligned} \quad \text{(Distributivität)}$$

$$\neg \neg F \equiv F \quad \text{(Doppelnegation)}$$

$$\begin{aligned} \neg(F \wedge G) &\equiv (\neg F \vee \neg G) \\ \neg(F \vee G) &\equiv (\neg F \wedge \neg G) \end{aligned} \quad \text{(deMorgansche Regeln)}$$

$$\begin{aligned} (F \vee G) &\equiv F, \text{ falls } F \text{ eine Tautologie} \\ (F \wedge G) &\equiv G, \text{ falls } F \text{ eine Tautologie} \end{aligned} \quad \text{(Tautologieregeln)}$$

$$\begin{aligned} (F \vee G) &\equiv G, \text{ falls } F \text{ unerfüllbar} \\ (F \wedge G) &\equiv F, \text{ falls } F \text{ unerfüllbar} \end{aligned} \quad \text{(Unerfüllbarkeitsregeln)}$$

$$(F \Leftrightarrow G) \equiv (F \Rightarrow G) \wedge (G \Rightarrow F) \quad \text{Elimination von } \Leftrightarrow:$$

$$(F \Leftrightarrow G) \equiv (F \wedge G) \vee (\neg F \wedge \neg G)$$

$$(F \Rightarrow G) \equiv (\neg F \vee G) \quad \text{Elimination von } \Rightarrow:$$

$$(F \wedge \top) \equiv F \quad \text{Tautologieregeln}$$

$$(F \vee \top) \equiv \top$$

$$(F \wedge \perp) \equiv \perp$$

$$(F \vee \perp) \equiv F$$

$$(F \wedge \neg F) \equiv \perp \quad \text{Kontradiktionsregeln:}$$

$$(F \vee \neg F) \equiv \top \quad \text{Komplement:}$$

T steht in diesem Fall für eine Tautologie,
⊥ steht für eine unerfüllbare Regel.

Normalformen

Ein *Literal* ist eine atomare Formel oder die Negation einer atomaren Formel.

Eine Formel ist in *konjunktiver Normalform (KNF)*, falls sie eine \wedge Konjunktion von \vee Disjunktionen von Literalen ist.

Bsp.: $(A \vee B) \wedge (\neg B \vee \neg D) \wedge E$

\Rightarrow Innen \vee oder, außen \wedge und

Eine Formel ist in *disjunktiver Normalform (DNF)*, falls sie eine \vee Disjunktion von \wedge Konjunktionen von Literalen ist.

Bsp.: $(A \wedge B) \vee (\neg B \wedge \neg D) \vee E$

\Rightarrow Innen \wedge und, außen \vee oder

Dabei gibt es für jede Formel eine äquivalente Formel in **KNF** und **DNF**.

Hornformeln

Eine Formel **F** ist eine *Hornformel*, falls **F** in **KNF** ist, und jedes \vee Disjunktionsglied höchstens ein positives Literal besitzt.

$(\neg A \vee \neg B)$	Kein Positives Literal
$(A \vee \neg B)$	Ein Positives Literal
$(A \vee B)$	Zwei Positive Literale

Um es uns leichter zu machen, können wir Hornformeln als \wedge Konjunktionen von \rightarrow Implikation schreiben.

$$F = (A \vee \neg B) \wedge (\neg C \vee \neg A \vee D) \wedge (\neg A \vee \neg B) \wedge D \wedge \neg E$$

$$F \equiv (B \rightarrow A) \wedge (C \wedge A \rightarrow D) \wedge (A \wedge B \rightarrow 0) \wedge (1 \rightarrow D) \wedge (E \rightarrow 0).$$

Dabei werden Teilformeln wie folgt ersetzt: (0 steht für unerfüllbare Formel, 1 für Tautologie)

$\dots \wedge A \wedge \dots$	$(1 \rightarrow A)$
$\dots \wedge \neg B \wedge \dots$	$(B \rightarrow 0)$
$\dots \wedge (C \vee \neg E) \wedge \dots$	$(E \rightarrow C)$
$\dots \wedge (C \vee \neg D \vee \neg E) \wedge \dots$	$(D \wedge E \rightarrow C)$
$\dots \wedge (\neg F \vee \neg G) \wedge \dots$	$(F \wedge G \rightarrow 0)$

Mithilfe von Hornformeln kann leicht auf *Erfüllbarkeit/Unerfüllbarkeit* getestet werden, jedoch nicht auf *Gültigkeit*.

1. Markiere jede atomare Formel **A**, für die es eine Teilformel der Form $(1 \rightarrow A)$ gibt.
2. Solange es Teilformeln der Form $(A_1 \wedge \dots \wedge A_n \rightarrow B)$ oder $(A_1 \wedge \dots \wedge A_n \rightarrow 0)$ gibt und A_1 bis A_n markiert sind, wiederhole diesen Schritt:
 Falls die Teilformel die Form $(A_1 \wedge \dots \wedge A_n \rightarrow B)$ hat, markiere **B** und wiederhole.
 Falls die Teilformel die Form $(A_1 \wedge \dots \wedge A_n \rightarrow 0)$ hat, gib aus dass die Formel unerfüllbar ist und brich ab.
3. Wenn es keine Teilformeln wie in 2. mehr gibt, gib aus dass die Formel erfüllbar ist und brich ab.

Wir können eine Formel **F** auf *Gültigkeit* prüfen, in dem wir **F** zuerst negieren, und dann prüfen ob die Anwendung des Hornformelalgorithmus *unerfüllbar* ausgibt.

Resolution

Mit Hilfe der *Resolution* kann die *Unerfüllbarkeit* einer Formel nachgewiesen werden.

Voraussetzung ist dass die Formel in **KNF** ist.

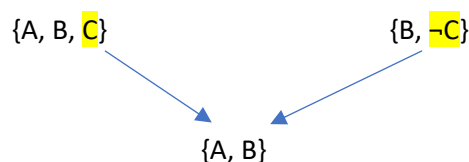
Sei **F** eine Formel in **KNF**:

Zuerst schreiben wir die *Klauseln* von **F** in der Mengenschreibweise auf:

Bsp: $F = (A \vee B \vee C) \wedge (B \vee \neg C) \wedge (\neg A \vee B) \wedge \neg D \wedge (\neg B \vee D)$

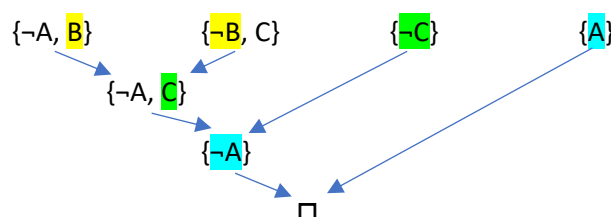
$F = \{\{A, B, C\}, \{B, \neg C\}, \{\neg A, B\}, \{\neg D\}, \{\neg B, D\}\}$

Nun können wir Klauseln, in denen ein **positives Literal** mit Klauseln resolvieren, die ein **negatives Literal** der gleichen Art haben. Dieses Literal wird bei der Resolution entfernt und es bleibt eine Menge der restlichen Literale beider Mengen.



Die entstehende Menge kann wiederum wieder zur Resolution genutzt werden.

Falls während einer Resolution die **leere Klausel** \square entsteht, so ist **F** *unerfüllbar*.



[Typ 0] Turing Maschinen

Turing Maschinen

Formell: Eine Turing Maschine ist ein 7-Tupel $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$:

Q	<i>Zustände</i>	Eine Menge von Zuständen
Σ	<i>Inputalphabet</i>	$\Sigma \subseteq \Gamma$ Inputsymbole <u>ohne</u> blankes Symbol \sqcup
Γ	<i>Tapealphabet</i>	Die Zeichen, die auf das Band geschrieben werden können
δ	<i>Übergangsfunktion</i>	$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ Gibt nächsten Zustand wieder.
q_0	<i>Startzustand</i>	Der Startzustand.
q_{accept}	<i>Akzeptierzustand</i>	Der Akzeptierzustand.
q_{reject}	<i>Ablehnungszustand</i>	Der Ablehnungszustand mit $q_{\text{accept}} \neq q_{\text{reject}}$.

Die Übergangsfunktion $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ funktioniert wie folgt:

Es werden der Zustand **Q** und ein Tapezeichen Γ gelesen und dafür ein neuer Zustand **Q**, ein zu schreibendes Tapezeichen Γ und die Richtung, in die sich der Schreibkopf bewegt (**L** für links, **R** für rechts), zurückgegeben.

Die Turingmaschine befindet sich auf einem unendlich langen Band, auf das sie an einzelne Stellen schreiben und lesen kann. Die Turingmaschine beginnt zunächst auf der linken Speicherzelle, das Eingabewort wird, auch beginnend auf der linken Speicherzelle, Zeichen für Zeichen nach rechts geschrieben. Alle weiteren Zellen sind mit dem *blanken Symbol* \sqcup gefüllt.

Versucht die Turingmaschine sich über das Tape hinaus zu bewegen, bspw. falls sie ganz links ist und versucht sich nach links zu bewegen, bleibt sie diesen Schritt stehen.

Die Turingmaschine berechnet so lange, bis sie entweder im Zustand q_{accept} oder q_{reject} landet, in dem sie stehen bleibt. Es kann jedoch auch passieren, dass die Maschine diese Zustände nie erreicht und dadurch unendlich lang rechnet.

Eine **Konfiguration** einer Turingmaschine ist eine spezielle Schreibweise um den Zustand der Turingmaschine, die Position der Turingmaschine auf dem Tape und den Inhalt des Tapes zu zeigen.

Wir schreiben eine Konfiguration wie folgt: **u q v**

- u** Die Tapesymbole links von der Turingmaschine
- q** Der momentane Zustand der Turingmaschine
- v** Die Tapesymbole rechts von der Turingmaschine. Das erste Symbol von **v** ist die momentane Position der Turingmaschine.

Bspw. würde **123 q₈ 45** bedeuten, dass die Turingmaschine im Zustand **q₈** und an Position **4** ist.

Da (normale) Turingmaschinen deterministisch sind, können wir genau bestimmen welche Konfiguration aus einer anderen folgt. Wir sagen in dem Fall, dass eine Konfiguration eine andere **liefert** (yields).

Erkennbare / Entscheidbare Sprachen

Wir unterscheiden zwischen **erkennbaren** und **entscheidbaren Sprachen**.

Diese Sprachen sind mächtiger als **REG**, **DCFL** und **CFL**.

Die Menge der Wörter die eine Turingmaschine **M** akzeptiert, ist die **Sprache von M**, bzw. die **Sprache die M erkennt**, und wird **L(M)** genannt.

Eine Maschine, für die bei jedem Eingabewort bekannt ist dass sie in endlichen Schritten **q_{accept}** oder **q_{reject}** erreicht, d.h. die nach endlichen Schritten terminiert, wird **Entscheider** genannt. Ein *Entscheider* der eine Sprache erkennt, **entscheidet** diese Sprache.

Sprachen, die von Turingmaschinen *erkannt* werden, werden **Turing-erkennbare Sprachen** oder **rekursiv aufzählbare Sprachen** genannt.

Sprachen, die von Turingmaschinen *entschieden* werden, werden **Turing-entscheidbare Sprachen** oder **rekursive Sprachen** genannt.

Erkenner	Entscheider
Dürfen accepten	Dürfen accepten
Dürfen rejecten	Dürfen rejecten
Dürfen loopen	Dürfen nicht loopen

Wichtig! **Entscheider** sind eine Teilmenge der **Erkenner**. (Jeder Entscheider ist ein Erkenner)

Turingmaschinenvarianten

Es gibt verschiedene Varianten der Turingmaschine, die jedoch alle gleichmächtig zu „normalen“ Turingmaschine sind.

Mehrbandmaschinen

Eine Turingmaschine mit mehreren Bändern. Jedes Band hat seinen eigenen Lese-Schreib-Kopf. Das Inputwort wird nur auf das erste Band geschrieben, alle anderen sind leer.

Die Übergangsfunktion sieht demnach so aus (k steht für die Anzahl Bänder):

$$\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$$

Es wird also von jedem Band separat ein Symbol gelesen, auf jedes Band separat ein Symbol geschrieben und der LSK auf jedem Band separat bewegt.

Mehrbandmaschinen lassen sich mit „normalen“ Turingmaschinen simulieren. Sei **M** also eine Mehrbandmaschine mit **k** Tapes und **S** eine Einzelbandmaschine. Wir schreiben **M**'s Tapes auf **S** mit einem Trennzeichen **#** dazwischen auf, und markieren dort jeweils mit einem Punkt, an welcher Position wir sind. **S** updated die Bänder wie **M**, braucht jedoch deutlich mehr Schritte.

Nichtdeterministische Turingmaschinen

Ähnlich zum NFA können während der Berechnung neue Berechnungsstränge entstehen. Die Übergangsfunktion Nichtdeterministische Turingmaschinen sieht wie folgt aus:

$$\delta: Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R\})$$

Es wird dabei eine Menge von „Updates“ zurückgegeben. Falls einer der Berechnungsstränge zu q_{accept} führt, akzeptiert die gesamte *NTM* das Inputwort.

Auch *NTM* sind gleichmächtig zu normalen *TM*.

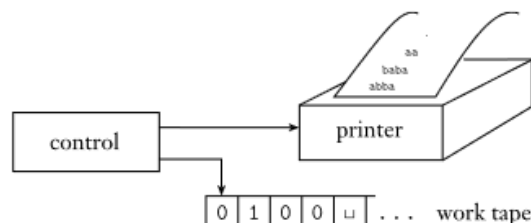
Sei **N** eine *NTM* und **D** eine *TM*. Wir simulieren alle möglichen Berechnungsstränge von **N** auf **D**. Sobald wir einen Berechnungsstrang finden, der zu q_{accept} führt, können wir abbrechen. Wir betrachten dabei **N**'s Berechnung als Baum. Diesen Baum simulieren wir mit Hilfe von **breadth-first-search**, d.h. wir betrachten alle Berechnungsstränge derselben Tiefe bis wir zur nächsten Tiefe fortschreiten.

Enumerator

Ein *Enumerator* ist eine Turingmaschine an die ein Drucker angehängt ist. Jedes Mal wenn die *TM* ein Wort drucken möchte, sendet sie es an den Drucker.

Ein *Enumerator E* fängt mit einem leeren Input auf seinem „Work Tape“ an. Die Sprache von **E**, ist die Menge aller Wörter, die **E** druckt (Reihenfolge/Dopplung egal).

Eine Sprache ist dabei *Turing-erkennbar*, falls ein *Enumerator* existiert, der sie enumeriert (druckt). Dies gilt auch für Sprachen mit unendlich vielen Wörtern.



Abgeschlossenheit von TM

Turing-erkennbare Sprachen sind gegenüber \cup Vereinigung und \cap Schnittmengenbildung abgeschlossen, jedoch nicht gegenüber \bar{A} Komplementbildung.

Turing-entscheidbare Sprachen sind gegenüber \cup Vereinigung und \cap Schnittmengenbildung und \bar{A} Komplementbildung abgeschlossen.

Algorithmen

Ein **Algorithmus** ist eine Reihe von simplen Anweisungen die befolgt werden können, um eine Aufgabe zu lösen.

Algorithmen lassen sich leicht in Sprachen umwandeln. Bspw. ist die Sprache des *Algorithmus*, der prüft ob eine Zahl durch 2 teilbar ist, die Sprache **D** = {**n** | **n** ist durch 2 teilbar}.

Nun können wir zu diesen Sprachen Turingmaschinen zu bauen, die diese Sprachen erkennen oder entscheiden.

Achtung! Da Turingmaschinen nur Wörter als Eingabe akzeptieren, müssen wir unsere Eingabe häufig vorher in ein Wort umwandeln (Wir können beispielsweise nicht direkt Graphen als Eingabe nehmen).

Wir schreiben dabei $\langle P \rangle$ um ein Objekt P in eine Zeichenkette umzuwandeln. Wie genau umgewandelt wird ist egal, solange alle notwendigen Informationen in der Zeichenkette stehen. Falls wir eine Zeichenkette $\langle P \rangle$ in eine Turingmaschine eingeben, müssen wir immer zuerst prüfen, ob diese richtig formatiert/codiert ist.

Entscheidbarkeit

Es gibt verschiedene Algorithmen die für simplere Sprachklassen ($< \text{Typ } 0$) bestimmte Eigenschaften prüfen können. Diese Algorithmen laufen alle über *Turingmaschinen*.

Häufig benutzen wir dabei eine *TM*, um einen simpleren Automaten / Grammatik zu simulieren. Genaue Ausführungen sind im SIPSER auf Seiten 194 – 200.

Es wird jeweils eine Sprache angegeben. Den Algorithmus anzuwenden ist äquivalent zum Prüfen, ob der DFA/CFG/... jeweils in dieser Sprache liegt. Da all diese Sprachen entscheidbar sind, existiert für sie auch ein entscheidender Algorithmus / eine *Entscheider-TM*.

Entscheidbarkeit regulärer Sprachen [Typ 3]

Wir definieren A_{DFA} als die Sprache aller DFAs und der Wörter die sie jeweils akzeptieren. Analog definieren wir auch A_{NFA} .

Wir können prüfen, ob ein DFA B ein Wort w akzeptiert, in dem wir prüfen, ob $\langle B, w \rangle$ in A_{DFA} enthalten ist.

A_{DFA} und A_{NFA} sind entscheidbare Sprachen, was bedeutet, dass auch ein entscheidbarer Algorithmus für sie existiert.

$$A_{\text{DFA}} = \{ \langle B, w \rangle \mid B \text{ ist ein DFA der } w \text{ akzeptiert} \}$$

$$A_{\text{NFA}} = \{ \langle B, w \rangle \mid B \text{ ist ein NFA der } w \text{ akzeptiert} \}$$

A_{REX} ist die Sprache aller regulären Ausdrücke und der Wörter die sie jeweils generieren.

$$A_{\text{REX}} = \{ \langle R, w \rangle \mid R \text{ ist ein regulärer Ausdruck der } w \text{ generiert} \}$$

E_{DFA} ist die Sprache aller DFA, deren Sprache leer ist ($L(A) = \emptyset$).

$$E_{\text{DFA}} = \{ \langle A \rangle \mid A \text{ ist ein DFA und } L(A) = \emptyset \}$$

EQ_{DFA} ist die Sprache aller DFA-Paare, die die gleiche Sprache akzeptieren.

$$EQ_{\text{DFA}} = \{ \langle A, B \rangle \mid A \text{ und } B \text{ sind DFAs und } L(A) = L(B) \}$$

Entscheidbarkeit kontextfreier Sprachen [Typ 3]

A_{CFG} ist die Sprache aller kontextfreie Grammatiken und der Wörter die sie jeweils generieren.

$$A_{CFG} = \{ \langle G, w \rangle \mid G \text{ ist eine kontextfreie Grammatik die } w \text{ generiert} \}$$

E_{CFG} ist die Sprache aller kontextfreien Grammatiken deren Sprache leer ist.

$$E_{CFG} = \{ \langle G \rangle \mid G \text{ ist eine kontextfreie Grammatik und } L(G) = \emptyset \}$$

EQ_{CFG} ist die Sprache aller CFG-Paare, die die gleiche Sprache generieren.

$$EQ_{CFG} = \{ \langle G, H \rangle \mid G \text{ und } H \text{ sind CFGs und } L(A) = L(B) \}$$

Geschrieben von David Rath

david.rath@studium.uni-hamburg.de