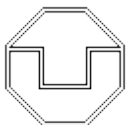


Formale Systeme

Prof. Dr.-Ing. Franz Baader
Lehrstuhl für Automatentheorie
Institut für Theoretische Informatik
TU Dresden

Information zur Vorlesung und zur Organisation der Übungen:

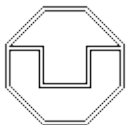
<http://lat.inf.tu-dresden.de/teaching/ws2010-2011/Bachelor/>



Formale Systeme

Literatur:

- Skript zur Vorlesung und Folien (siehe Web-page).
- Uwe Schöning, *Theoretische Informatik – kurzgefasst*, Spektrum Akademischer Verlag, 2001.
- Ingo Wegener, *Theoretische Informatik – eine algorithmenorientierte Einführung*, Teubner, 1999.
- Uwe Schöning, *Logik für Informatiker*, Spektrum Akademischer Verlag, 2000.



Formale Systeme

Ein wichtige Aufgabe in der Informatik ist die **formale Modellierung** der betrachteten Systeme.

Die **Theoretische Informatik** stellt die formalen Grundlagen für eine derartige Modellierung zur Verfügung:

Syntax:

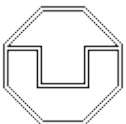
Was kann man sagen?

Semantik:

Was bedeutet das gesagte?

**Automaten und
formale Sprachen**

Logik



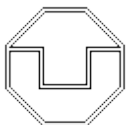
Wir betrachten hier logische Formeln über atomaren Aussagen (**Aussagenlogik**).

- Wie sieht eine logische Formel aus?
Wann ist eine logische Formel wahr bzw. nicht wahr?

Syntax und Semantik der Aussagenlogik

- Wie kann man aus einer Menge von logischen Formeln weitere Aussagen/Formeln schlussfolgern?

Kalküle des logischen Schließens

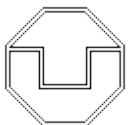


Automaten und Formale Sprachen

Hier betrachtet man **Wörter** (z.B. abb, bbaba, ...) und **Formale Sprachen**, d.h. Mengen von Wörtern (z.B. $\{aa, bb, ab\}$; Wörter über a, b , die eine gerade Anzahl an a s enthalten, ...).

Wichtige Fragestellungen

- (a) **Charakterisierung**: wie beschreibt man die (meist unendlichen) Mengen von Wörtern mit endlichem Aufwand?
- **Automaten** oder **Maschinen**, die genau die Elemente der Menge akzeptieren (endliche Automaten, Kellerautomaten, Turing-Maschinen).
 - **Grammatiken**, die genau die Elemente der Menge generieren (vgl. AuD: kontextfreie Grammatiken, z.B. zur Beschreibung der Syntax von Programmiersprachen).
 - **Ausdrücke**, die beschreiben, wie man die Sprache aus Basissprachen mittels gewisser Operationen (z.B. Vereinigung) erzeugen kann.



Chomsky-Hierarchy

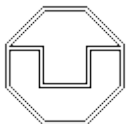
faßt die wichtigsten Klassen
Formaler Sprachen zusammen

Klasse	Automatentyp	Grammatiktyp
Typ 0	Turingmaschine (TM)	allgemeine Chomsky-Grammatik
Typ 1	TM mit linearer Bandbeschränkung	kontextsensitive Grammatik
Typ 2	Kellerautomat	kontextfreie Grammatik
Typ 3	endlicher Automat	einseitig lineare Grammatik

Die aus praktischer Sicht wichtigste Typen sind Typ 2 und Typ 3:

Typ 3: auch Beschreibung durch **reguläre Ausdrücke**,
die z.B. als Suchpattern bei der Textsuche Verwendung finden.

Typ 2: können weitgehend die Syntax von Programmiersprachen
beschreiben.



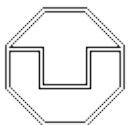
Wichtige Fragestellungen

(b) **Entscheidungsprobleme:** welche Problemstellungen sind für eine Sprachklasse entscheidbar und mit welchem Aufwand?

- **Wortproblem:** geg. eine Beschreibung der Sprache L (durch Automat, Grammatik,..) und ein Wort w . Gilt $w \in L$?

Anwendungsbeispiele:

- * Programmiersprache, deren Syntax durch eine kontextfreie Grammatik beschrieben ist. Entscheide, ob ein geg. Programm P syntaktisch korrekt ist.
- * Suchpattern als regulärer Ausdruck. Suche die Dateien (= Wörter), welche das Suchpattern enthalten (= zu der durch den Ausdruck beschriebenen Sprache gehören).



Wichtige Fragestellungen

(b) **Entscheidungsprobleme:** welche Problemstellungen sind für eine Sprachklasse entscheidbar und mit welchem Aufwand?

- **Leerheitsproblem:** geg. eine Beschreibung der Sprache L .
Ist $L \neq \emptyset$?

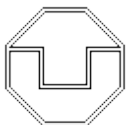
Anwendungsbeispiel:

- * Wenn ein Suchpattern die leere Sprache beschreibt, so muß man die Dateien gar nicht durchsuchen.

- **Äquivalenzproblem:** beschreiben zwei verschiedene Beschreibungen dieselbe Sprache?

Anwendungsbeispiel:

- * Zwei Lehrbücher über eine Programmiersprache beschreiben die Syntax durch verschiedene Grammatiken. Sind diese Beschreibungen wirklich äquivalent?

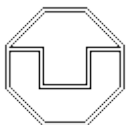


Wichtige Fragestellungen

- (c) **Abschlußeigenschaften:** unter welchen Operationen auf Sprachen (wie Durchschnitt $L_1 \cap L_2$, Vereinigung $L_1 \cup L_2$, Komplement $\bar{L}$) ist die Sprachklasse abgeschlossen?

Anwendungsbeispiel:

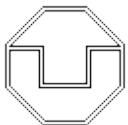
- * Suchpattern: suche nach Dateien, die das Pattern **nicht** enthalten (Komplement) oder die zwei Pattern **gleichzeitig** enthalten (Durchschnitt).
- * Reduziere das Äquivalenzproblem auf das Leerheitsproblem: statt “ $L_1 = L_2$?” entscheide “ $(L_1 \cap \bar{L}_2) \cup (L_2 \cap \bar{L}_1) = \emptyset$?”.



Automaten und Formale Sprachen

Teil I: Endliche Automaten

- 0. Einführung
 - 1. Nichtdeterministische endliche Automaten
 - 2. Deterministische endliche Automaten
 - 3. Nachweis der Nichterkennbarkeit
 - 4. Abschlußeigenschaften und Entscheidungsprobleme
 - 5. Reguläre Ausdrücke und Sprachen



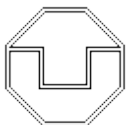
Automaten und Formale Sprachen

Teil II: Grammatiken, kontextfreie Sprachen und Kellerautomaten

6. Die Chomsky Hierarchie
7. Rechtslineare Grammatiken und reguläre Sprachen
8. Normalformen kontextfreier Sprachen
9. Abschlußeigenschaften kontextfreier Sprachen
10. Kellerautomaten

Teil III: Turingmaschinen und Grammatiken

11. Turingmaschinen
12. Zusammenhang zwischen Turingmaschinen und Grammatiken



§ 0. Einführung

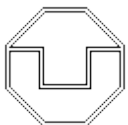
Endliche Automaten sind gekennzeichnet durch

- endliche Menge von (internen) Zuständen
- Übergänge zwischen Zuständen: abhängig von internen Struktur des Automaten und der Eingabe

Im Prinzip sind Rechner ebenfalls endliche Automaten:
endlich viel Speicherplatz und daher nur endliche Menge an Konfigurationen.

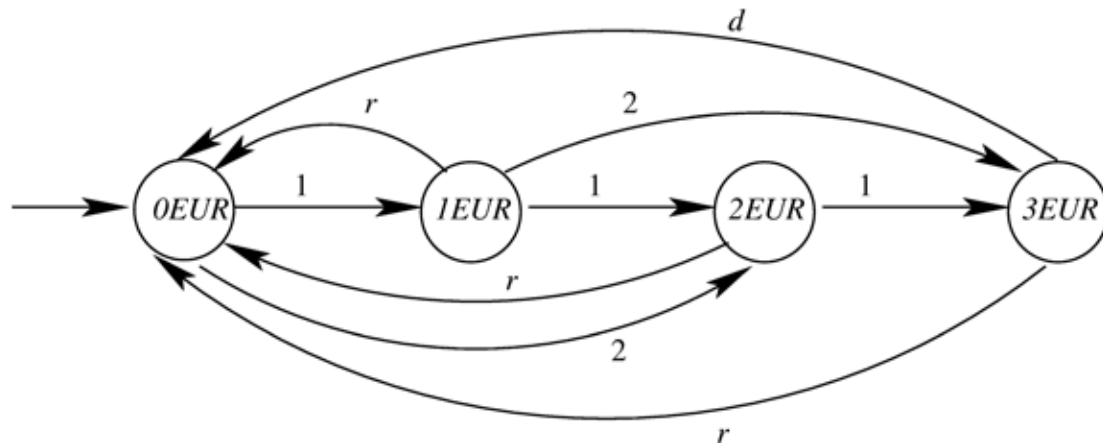
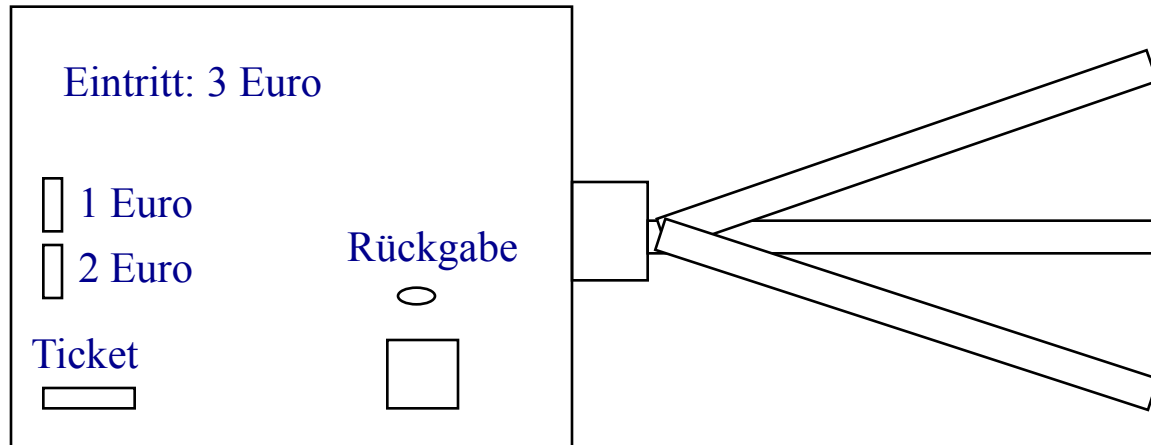
Aber:

Keine geeignete Beschreibungsebene wegen extrem hoher Zustandsanzahl.
Daher abstrahiert man von der Speicherplatzbeschränkung und nimmt potentiell unendlichen Speicher an (in GThI 2: Turing-Maschinen mit unendlichen Bändern).

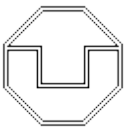


Beispiel

Eintrittskartenautomat



Welche Eingabe führt zum Zustand 3EUR, in dem man durch das Drehgitter darf?



Grundlegende Definitionen

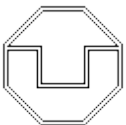
Alphabet: endliche Menge von Buchstaben
 Σ für Alphabete, $a, b, c, \dots$ für Buchstaben.

Wort: endliche Folge von Buchstaben
 $w = a_1 \dots a_n$ mit $a_i \in \Sigma$ heißt Wort über dem Alphabet Σ .

Länge: des Worts $w = a_1 \dots a_n$ ist $|w| = n$;
z.B. $|aba| = 3$.

Anzahl der Vorkommen von Buchstaben:
 $|w|_a$ zählt die Anzahl der Vorkommen von a in w ;
z.B. $|aba|_a = 2$, $|aba|_b = 1$, $|aba|_c = 0$.

Leeres Wort: ε hat Länge 0.



Grundlegende Definitionen

Σ^* : Menge aller Wörter über Σ ;

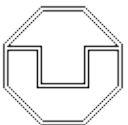
$$\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}.$$

Formale Sprache: Menge von Wörtern;

$L \subseteq \Sigma^*$ ist formale Sprache über Σ .

Beispiele:

- Menge der Wörter über $\{a, \dots, z\}$, die korrekte Wörter der deutschen Sprache sind.
- Menge der Wörter, die das Stichwort “theorie” enthalten.



Operationen auf Sprachen

Boolesche Operationen: Vereinigung, Durchschnitt, Komplement und Differenz;

$$L_1 \cup L_2, L_1 \cap L_2,$$

$$\overline{L} := \{w \in \Sigma^* \mid w \notin L\},$$

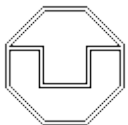
$$L_1 \setminus L_2 := L_1 \cap \overline{L_2}.$$

Konkatenation: Aneinanderhängen von Wörtern;

$$u \cdot v := uv; \text{ z.B. } abb \cdot ab = abbab.$$

$$L_1 \cdot L_2 := \{u \cdot v \mid u \in L_1 \text{ und } v \in L_2\}.$$

Beachte: der Konkatenationspunkt wird auch bei Sprachen meist weggelassen, d.h. $L_1 L_2$ anstelle von $L_1 \cdot L_2$.



Operationen auf Sprachen

Kleene-Stern: Iterierte Konkatination

$$L^0 := \{\varepsilon\},$$

$$L^{n+1} := L^n \cdot L,$$

$$L^* := \bigcup_{n \geq 0} L^n,$$

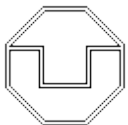
$$L^+ := \bigcup_{n \geq 1} L^n$$

Präfix, Suffix, Infix: Anfangs-, End-, und Mittelstücke von Wörtern;

u ist **Präfix** von v gdw $v = uw$ für ein $w \in \Sigma^*$,

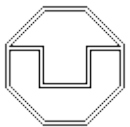
u ist **Suffix** von v gdw $v = wu$ für ein $w \in \Sigma^*$,

u ist **Infix** von v gdw $v = w_1uw_2$ für $w_1, w_2 \in \Sigma^*$.



Teil I: Endliche Automaten

0. Einführung
1. Nichtdeterministische endliche Automaten
2. Deterministische endliche Automaten
3. Nachweis der Nichterkennbarkeit
4. Abschlußeigenschaften und Entscheidungsprobleme
5. Reguläre Ausdrücke und Sprachen

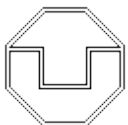


§ 1. Nichtdeterministische endliche Automaten

Definition 1.1 (Transitionssystem)

Ein Transitionssystem ist von der Form $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$, wobei

- Q eine Menge von Zuständen ist (nicht notwendigerweise endlich!),
- Σ ein endliches Alphabet ist,
- $I \subseteq Q$ eine Menge von Anfangszuständen ist,
- $\Delta \subseteq Q \times \Sigma \times Q$ eine Übergangsrelation (Transitionsrelation) ist,
- $F \subseteq Q$ eine Menge von Endzuständen ist.



Definition 1.2 (Nichtdeterministischer endlicher Automat)

Das Transitionssystem $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ heißt nichtdeterministischer endlicher Automat (**NEA**) falls

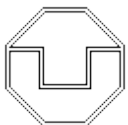
- $|Q| < \infty$ und
- $|I| = 1$.

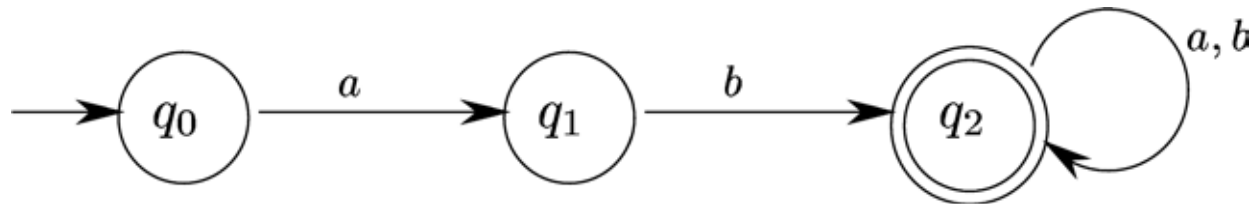
Schreibweise: $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$.

Beispiel 1.3 (Nichtdeterministischer endlicher Automat)

$\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ mit

- $Q = \{q_0, q_1, q_2\}$,
- $\Sigma = \{a, b\}$,
- $\Delta = \{(q_0, a, q_1), (q_1, b, q_2), (q_2, a, q_2), (q_2, b, q_2)\}$,
- $F = \{q_2\}$.

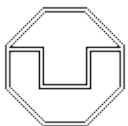




Beispiel 1.3 (Nichtdeterministischer endlicher Automat)

$\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ mit

- $Q = \{q_0, q_1, q_2\}$,
- $\Sigma = \{a, b\}$,
- $\Delta = \{(q_0, a, q_1), (q_1, b, q_2), (q_2, a, q_2), (q_2, b, q_2)\}$,
- $F = \{q_2\}$.



Definition 1.4 (Pfad in Transitionssystem)

Ein **Pfad** in einem Transitionssystem $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ ist eine Folge

$$\pi = (p_0, a_1, p_1)(p_1, a_2, p_2) \cdots (p_{n-1}, a_n, p_n)$$

mit $(p_i, a_{i+1}, p_{i+1}) \in \Delta$ für $i = 0, \dots, n-1$.

Pfad von p nach q : $p_0 = p$ und $p_n = q$.

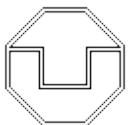
Beschriftung des Pfads π : $a_1 \dots a_n$

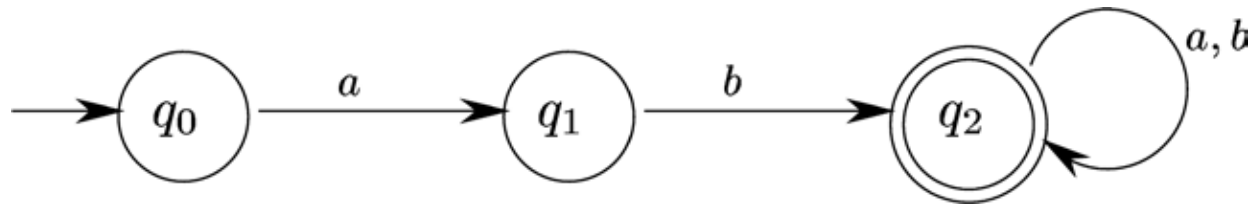
Länge des Pfads π : n

Länge 0: **leerer Pfad** mit Beschriftung ε .

$p \xrightarrow{w}_{\mathcal{A}} q$: es gibt einen Pfad von p nach q mit Beschriftung w .

$Q_1 \xrightarrow{w}_{\mathcal{A}} Q_2$: es gibt $p_1 \in Q_1$ und $p_2 \in Q_2$ mit $p_1 \xrightarrow{w}_{\mathcal{A}} p_2$.





Leerer Pfad von q_0 nach q_0 mit Beschriftung ε .

Pfad von q_0 nach q_1 mit Beschriftung a .

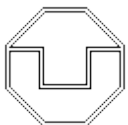
Pfad von q_0 nach q_2 mit Beschriftung ab .

Pfad von q_0 nach q_2 mit Beschriftung aba .

Pfad von q_0 nach q_2 mit Beschriftung abb .

Pfad von q_0 nach q_2 mit Beschriftung $abaa$.

$$q_0 \xrightarrow{abw}_{\mathcal{A}} q_2 \text{ für all } w \in \Sigma^*.$$



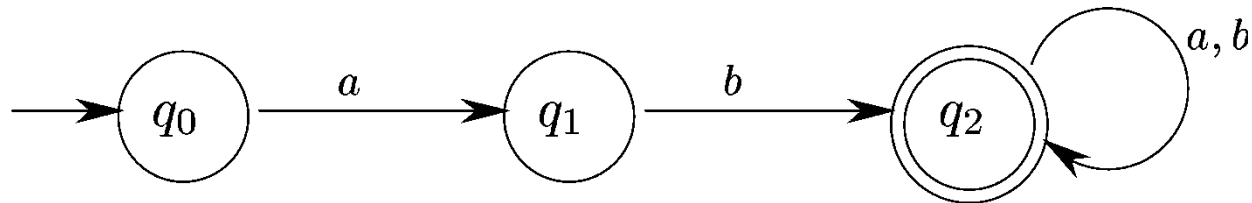
Definition 1.5 (Akzeptierte Sprache)

Das Transitionssystem $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ **akzeptiert (erkennt)** das Wort $w \in \Sigma^*$ gdw

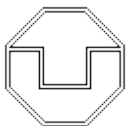
$$I \xrightarrow{w}_{\mathcal{A}} F.$$

Die von $\mathcal{A}$ **akzeptierte (erkannte) Sprache** ist

$$L(\mathcal{A}) := \{w \in \Sigma^* \mid \mathcal{A} \text{ akzeptiert } w\}.$$



$$L(\mathcal{A}) = \{abw \mid w \in \Sigma^*\}$$

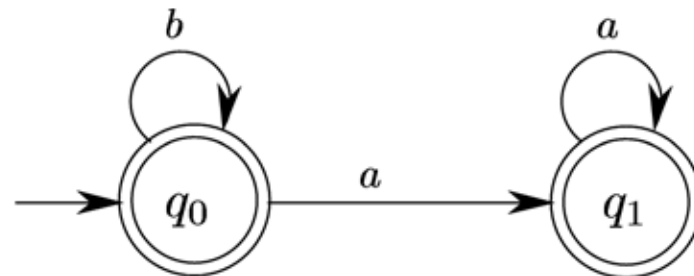


Definition 1.6 (erkennbare Sprachen)

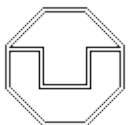
Die Sprache $L \subseteq \Sigma^*$ heißt **erkennbar** gdw es einen **NEA** $\mathcal{A}$ gibt mit $L = L(\mathcal{A})$.

Beispiel 1.7 (eine erkennbare Sprachen)

$L := \{w \in \{a, b\}^* \mid ab \text{ ist nicht Infix von } w\}$.

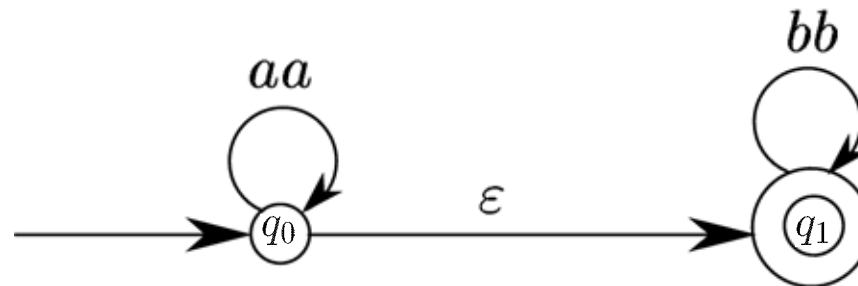


Äquivalente Transitionssysteme: erkennen dieselbe Sprache.



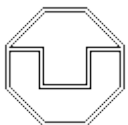
Definition 1.8 (NEA mit Wort- und ε -Übergängen)

1. Ein NEA mit Wortübergängen hat die Form $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$, wobei Q, Σ, q_0, F wie beim NEA definiert sind und $\Delta \subseteq Q \times \Sigma^* \times Q$ eine **endliche** Menge von Wortübergängen ist.
2. Ein ε -NEA ist ein NEA mit Wortübergängen, wobei für alle $(q, w, q') \in \Delta$ gilt: $|w| \leq 1$ (d.h. $w \in \Sigma$ oder $w = \varepsilon$).



Pfad $(q_0, aa, q_0)(q_0, aa, q_0)(q_0, \varepsilon, q_1)(q_1, bb, q_1)$ hat Beschriftung $aa \cdot aa \cdot \varepsilon \cdot bb = aaaabb$.

Akzeptierte Sprache?



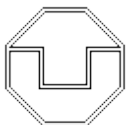
Satz 1.9 (NEA mit Wortübergängen nicht stärker als NEA)

Zu jedem NEA mit Wortübergängen kann man **effektiv** einen äquivalenten NEA (ohne Wortübergänge) konstruieren.

Effektiv: durch einen Algorithmus realisierbar, der als Eingabe den NEA mit Wortübergängen erhält und als Ausgabe den dazu äquivalenten NEA liefert.

Beweis:

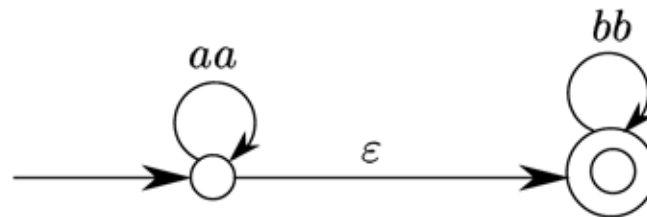
NEA mit Wortübergängen $\xrightarrow{\text{Lemma 1.10}}$ ε -NEA $\xrightarrow{\text{Lemma 1.12}}$ NEA



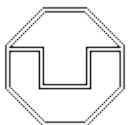
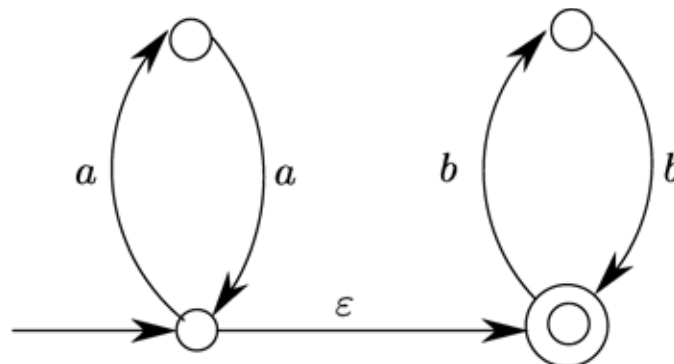
Lemma 1.10 (NEA mit Wortübergängen nicht stärker als ε -NEA)

Zu jedem NEA mit Wortübergängen kann man **effektiv** einen äquivalenten ε -NEA konstruieren.

Beispiel 1.11

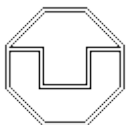
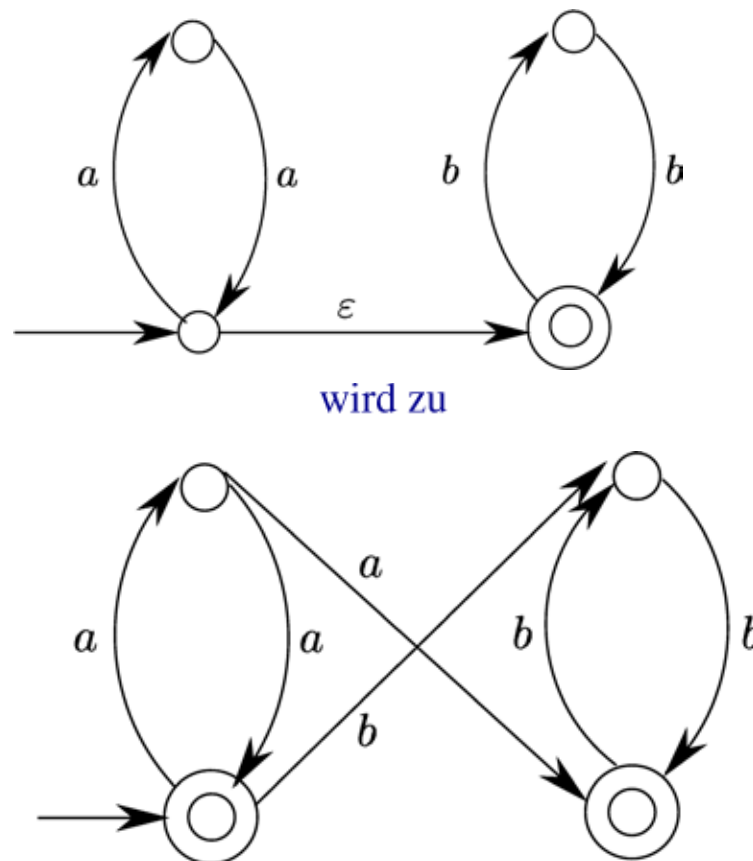


wird zu



Lemma 1.12 (ε -NEA nicht stärker als NEA)

Zu jedem ε -NEA kann man **effektiv** einen äquivalenten NEA konstruieren.



Beweis:

Der ε -NEA

$$\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$$

sei gegeben.

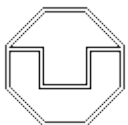
Wir konstruieren daraus einen NEA ohne ε -Übergänge wie folgt:

$$\mathcal{A}' = (Q, \Sigma, q_0, \Delta', F'),$$

wobei

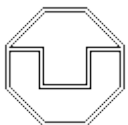
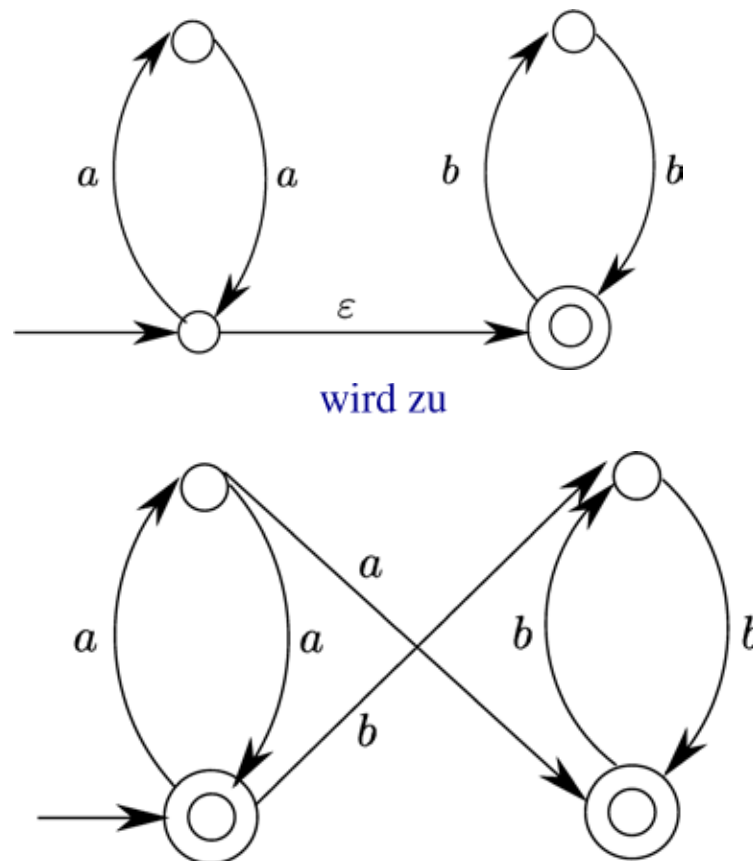
- $\Delta' := \{(p, a, q) \in Q \times \Sigma \times Q \mid p \xrightarrow{a}_{\mathcal{A}} q\}$
- $F' := \begin{cases} F \cup \{q_0\} & \text{falls } q_0 \xrightarrow{\varepsilon}_{\mathcal{A}} F \\ F & \text{sonst} \end{cases}$

Zu zeigen: $\mathcal{A}'$ kann effektiv bestimmt werden und $L(\mathcal{A}) = L(\mathcal{A}')$.



Lemma 1.12 (ε -NEA nicht stärker als NEA)

Zu jedem ε -NEA kann man **effektiv** einen äquivalenten NEA konstruieren.



Lemma 1.12 liefert auch das folgende

Zu jedem **endlichen** Transitionssystem kann man **effektiv** einen äquivalenten NEA konstruieren.

Beweis:

endliches
Transitionssystem $\xrightarrow{\hspace{1cm}}$ ϵ -NEA $\xrightarrow{\text{Lemma 1.12}}$ NEA

