Grundlagen der theoretischen Informatik

Kurt Sieber

Fakultät IV, Department ETI
Universität Siegen

SS 2013

Gesamter Foliensatz zu

Teil I: Automaten und Formale Sprachen

Stand: 13.08.2013

Vorbemerkung

Dieser Gesamtfoliensatz enthält kleine (eher unwesentliche) Abweichungen von den einzelnen Foliensätzen, die während der Vorlesung erschienen sind, z.B.

Seite 85-86: Vereinfachung des Beispiels zu Satz 2.19

Seite 252: Bemerkungen zu den Algorithmen aus Satz 3.29

Außerdem wurden kleine Fehler korrigiert und das Kapitel über kontextfreie Sprachen wurde neu durchnummeriert

Inhalt der Vorlesung

Teil I: Automaten und formale Sprachen (Kurt Sieber)

Zentrale Fragestellungen

- Wie definiert man (formal) die Syntax einer Sprache (z.B. einer Programmiersprache)?
- Wie überprüft man, ob ein Wort (d.h. eine Zeichenreihe) zur Sprache gehört?

Teil II: Berechenbarkeit und Komplexität (Hannes Diener)

Zentrale Fragestellungen

- Welche Probleme (Aufgabenstellungen) sind prinzipiell mit einem Computer lösbar bzw. nicht lösbar?
- Welche Probleme sind mit vernünftigem Aufwand (an Speicherplatz und Laufzeit) lösbar?

Zeichen, Wörter, Sprachen

Grundlagen für beide Teile: Zeichen, Wörter und Sprachen

Definition 1.1 *Ein* Alphabet *oder* Zeichenvorrat *ist eine nichtleere endliche Menge. Die Elemente eines Alphabets nennen wir* Zeichen *oder* Symbole*.*

Konvention: Alphabete werden mit $\Sigma, \Sigma_1, \Sigma', \ldots$ bezeichnet.

Beispiele

- $\Sigma_1 = \{1\}$
- $\Sigma_2 = \{0, 1\}$
- $\Sigma_3 = \{0, \ldots, 9\}$
- $\Sigma_4 = \{a, \ldots, z, A, \ldots, Z\}$
- $\Sigma_5 =$ Menge aller ASCII-Zeichen
- $\Sigma_6 = \{\mathit{begin}, \mathit{end}, \mathit{if}, \mathit{then}, \mathit{else}, ;, :=, +, *, \ldots\}$

Zeichen, Wörter, Sprachen

Definition 1.2 *Ein* Wort *über dem Alphabet* Σ *ist eine endliche Folge* $w = (a_1, \ldots, a_n)$, *wobei* $n \geq 0$ *und* $a_i \in \Sigma$ *für* $i = 1, \ldots, n$. n *heißt* Länge *des Wortes* w, a_i *heißt* i*-tes Zeichen von* w. *Das Wort* () *heißt* leeres Wort.

Kurzschreibweise

- $a_1 \ldots a_n$ statt $(a_1, \ldots, a_n)$
  Vorsicht: Zeichen a und Wort a nicht mehr unterscheidbar!
- ε statt ()

Weitere Schreibweisen

- $|w|$ = Länge von w
- Σ^* = Menge aller Wörter über Σ
- $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ = Menge aller nichtleeren Wörter über Σ

Beispiele

- $\{1\}^* = \{\varepsilon, 1, 11, 111, \ldots\}$
- $\{0,1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\}$
- $\varepsilon, 0, 123, 0815 \in \{0, \ldots, 9\}^*$
- Der Inhalt einer Datei ist *ein* Wort über dem ASCII-Alphabet (denn Leerzeichen, Zeilenwechsel etc. sind Zeichen dieses Alphabets).
- Ein Programm (in irgendeiner Programmiersprache) ist ebenfalls ein Wort über dem ASCII-Alphabet.
- Der Inhalt eines Buches ist ein Wort (über dem Alphabet der Sprache, in der es geschrieben ist).

Alternative Definition für Σ^* und Σ^+

- $\Sigma^n = \{(a_1, \ldots, a_n) \mid a_i \in \Sigma \text{ für } i = 1, \ldots, n\}$
  $= \{a_1 \ldots a_n \mid a_i \in \Sigma \text{ für } i = 1, \ldots, n\}$
  $=$ Menge aller *Wörter der Länge n* über Σ

- Speziell: $\Sigma^0 = \{()\}$
  $= \{\varepsilon\}$

- $\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$
  $=$ Menge *aller Wörter* über Σ

  $\Sigma^+ = \bigcup_{n \geq 1} \Sigma^n$
  $= \Sigma^* \setminus \{\varepsilon\}$

Definition 1.3 *Seien* $v = a_1 \dots a_m$ *und* $w = b_1 \dots b_n$ *Wörter über* Σ. *Dann ist die* Konkatenation $v \circ w$ *von* v *und* w *definiert durch*

$$v \circ w \;=\; a_1 \dots a_m b_1 \dots b_n$$

Kurzschreibweise: vw statt $v \circ w$

Eigenschaften der Konkatenation

- *Assoziativität*: $(uv)w = u(vw)$ für alle $u, v, w \in \Sigma^*$
- *Neutrales Element*: $w\varepsilon = \varepsilon w = w$ für alle $w \in \Sigma^*$
- Also: $(\Sigma^*, \circ)$ ist ein *Monoid* mit neutralem Element ε (vgl. natürliche Zahlen mit Multiplikation)

Definition 1.4 *Sei $w \in \Sigma^*$ und $n \in \mathbb{N}$. Dann ist die n-te Potenz w^n von w definiert durch $w^n = \underbrace{w \ldots w}_{n}$ (speziell: $a^n = \underbrace{a \ldots a}_{n}$ falls $a \in \Sigma$).*

Besser: Definition von w^n durch Induktion über n

- $w^0 = \varepsilon$
- $w^n = ww^{n-1}$ falls $n > 0$

Oder:

- $w^0 = \varepsilon$
- $w^n = w^{n-1}w$ falls $n > 0$

Beispiele

- $(bla)^2 = blabla$
- $(bla\ )^3 = bla\ bla\ bla$

Definition 1.5 *Für* $w = a_1 \dots a_n \in \Sigma^*$ *sei* $w^R \in \Sigma^*$ *das Wort, das durch "Spiegelung" aus* w *entsteht , d.h.*

$$w^R = a_n \dots a_1$$

(R *steht für engl. reverse)*

Beispiel

$(01001)^R = 10010$

Alternative

Definition von w^R durch Induktion über $|w|$
(s. Übung!)

Definition 1.6 *Sei $x \in \Sigma^*$.*

- *u heißt Anfangswort oder Präfix von x, falls ein $v \in \Sigma^*$ existiert mit $x = uv$.*
- *v heißt Endwort oder Suffix von x, falls ein $u \in \Sigma^*$ existiert mit $x = uv$.*
- *v heißt Teilwort von x, falls $u, w \in \Sigma^*$ existieren mit $x = uvw$.*

Beispiel

- Präfixe von 010: $\varepsilon, 0, 01, 010$
- Suffixe von 010: $\varepsilon, 0, 10, 010$
- Teilwörter von 010: $\varepsilon, 0, 1, 01, 10, 010$

Zeichen, Wörter, Sprachen

Die Beziehungen ‘Anfangswort’, ‘Endwort’ und ‘Teilwort’ sind zweistellige Relationen auf Σ^*. Jede dieser Relationen ist

- *reflexiv*: Jedes $w \in \Sigma^*$ steht in Relation zu sich selbst.
- *transitiv*: Wenn u in Relation zu v und v in Relation zu w steht, dann steht auch u in Relation zu w.
- *antisymmetrisch*: Wenn u in Relation zu v steht und v in Relation zu u, dann ist $v = u$.

Also: Jede der Relationen definiert eine *partielle Ordnung* auf Σ^*.

Definition 1.7 *Eine Sprache (oder formale Sprache) über dem Alphabet Σ ist eine (beliebige) Teilmenge von Σ^*.*

Konvention:

Sprachen bezeichnen wir meist mit $L, L_1, L', \ldots$ (engl. *language*).

Alternative Formulierung

Eine Sprache über Σ ist ein Element der *Potenzmenge* $\wp(\Sigma^*)$.

Beispiele

- $L_1 = \emptyset$
- $L_2 = \Sigma^*$
- $L_3 = \{\varepsilon\}$
- $L_4 = \{1^n \mid n \text{ ist eine Primzahl}\}$
- $L_5 =$ Menge aller Binärdarstellungen natürlicher Zahlen
- $L_6 =$ Menge aller Dezimaldarstellungen natürlicher Zahlen
- $L_7 =$ Menge aller Java-Programme
- $L_8 =$ Menge aller Sätze der deutschen Sprache
- $L_9 = \{0^n 1^n \mid n \geq 0\}$
- $L_{10} = \{vw \mid v \in \{0\}^*, w \in \{1\}^*\}$

Definition 1.8 *Seien* $L, L_1, L_2 \subseteq \Sigma^*$. *Dann sei*

- $L_1 \cup L_2$ *die* Vereinigung *von* L_1 *und* L_2.
- $L_1 \cap L_2$ *der* Durchschnitt *von* L_1 *und* L_2.
- $L_1 \setminus L_2 = \{w \in L_1 \mid w \notin L_2\}$ *die* Mengendifferenz *von* L_1 *und* L_2.
- $\bar{L} = \Sigma^* \setminus L$ *das* Komplement *von* L.
- $L_1 \circ L_2 = \{vw \mid v \in L_1, w \in L_2\}$ *die* Konkatenation *von* L_1 *und* L_2.

Kurzschreibweise: $L_1 L_2$ statt $L_1 \circ L_2$

Eigenschaften der Konkatenation

- *Assoziativität*: $(L_1 L_2)L_3 = L_1(L_2 L_3)$ für alle $L_1, L_2, L_3 \subseteq \Sigma^*$
- *Neutrales Element*: $L\{\varepsilon\} = \{\varepsilon\}L = L$ für alle $L \subseteq \Sigma^*$
- Also: $(\wp(\Sigma^*), \circ)$ ist ein *Monoid* mit neutralem Element $\{\varepsilon\}$.

Definition 1.9 *Seien* $L \subseteq \Sigma^*$ *und* $n \in \mathbb{N}$. *Dann ist die* n-te Potenz L^n *von* L *definiert durch*

$$\begin{aligned} L^n &= \underbrace{L \circ \ldots \circ L}_{n} \\ &= \underbrace{L \ldots L}_{n} \\ &= \{w_1 \ldots w_n \mid w_i \in L \text{ für } i = 1, \ldots, n\} \end{aligned}$$

Besser: Definition von L^n durch Induktion über n

- $L^0 = \{\varepsilon\}$
- $L^n = LL^{n-1}$ falls $n > 0$

Oder:

- $L^0 = \{\varepsilon\}$
- $L^n = L^{n-1}L$ falls $n > 0$

Definition 1.10 *Für $L \subseteq \Sigma^*$ sei*

- $L^R = \{w^R \mid w \in L\}$ *die "Spiegelung" von* L
- $L^* = \bigcup_{n \geq 0} L^n$

  $= \{w_1 \dots w_n \mid n \geq 0, w_i \in L \text{ für } i = 1, \dots, n\}$

  der Kleene-Abschluss von L.

  Speziell:

  $\{w\}^* = \{w^n \mid n \geq 0\}$ *falls* $w \in \Sigma^*$

  $\{a\}^* = \{a^n \mid n \geq 0\}$ *falls* $a \in \Sigma$
- $L^+ = \bigcup_{n \geq 1} L^n$

  $= \{w_1 \dots w_n \mid n \geq 1, w_i \in L \text{ für } i = 1, \dots, n\}$

  $= LL^*$

Folgerung: $L^* = L^+ \cup L^0 = L^+ \cup \{\varepsilon\}$

Rechenregeln für Potenzen

. . . von Wörtern

- $w^0 = \varepsilon$ (neutrales Element)
- $w^1 = w$
- $w^m w^n = w^{m+n}$
- $(w^m)^n = w^{mn}$

. . . von Sprachen

- $L^0 = \{\varepsilon\}$ (neutrales Element)
- $L^1 = L$
- $L^m L^n = L^{m+n}$
- $(L^m)^n = L^{mn}$

Diese Regeln gelten in *jedem* Monoid (wenn man das Potenzieren nach dem üblichen Schema definiert). Denn sie folgen allein aus der Assoziativität und der Eigenschaft des neutralen Elements.

Warnung: Es gilt *nicht* $(w_1 w_2)^n = w_1^n w_2^n$ oder $(L_1 L_2)^n = L_1^n L_2^n$. Denn dazu bräuchte man die Kommutativität.

Rechenregeln für * und $^+$

- $L \subseteq L^*$
- $(L^*)^* = L^*$
- $L \subseteq L^+$
- $(L^+)^+ = L^+$

Mengenoperatoren mit diesen beiden Eigenschaften nennt man *Abschluss-* oder *Hüllenoperatoren*.

Beweis für *

1. $L \subseteq L^*$ ist klar wegen $L = L^1$ und $L^* = \bigcup_{n \geq 0} L^n$.
2. $L^* = (L^*)^*$:
   '$\subseteq$' ist klar wegen 1.
   '$\supseteq$': Sei $w \in (L^*)^*$, d.h. $w = w_1 \dots w_n$ mit $n \geq 0$ und $w_i \in L^*$ für $i = 1, \dots, n$. Dann existiert für jedes i ein $m_i \geq 0$ mit $w_i \in L^{m_i}$, also folgt $w = w_1 \dots w_n \in L^{m_1} \dots L^{m_n} = L^{m_1 + \dots + m_n} \subseteq L^*$. $\square$

Zeichen, Wörter, Sprachen

Die Operatoren $\cup, \circ, ^*, \ldots$ lassen sich auch zur *Beschreibung* von Sprachen verwenden. Konvention: Die Postfixoperatoren ^{n}, $^+$ und * binden am stärksten, $\circ$ bindet stärker als $\cup$.

Beispiele

- Die Menge aller Dezimaldarstellungen natürlicher Zahlen:
  $L_{nat} = \{0, \ldots, 9\}^+ = \{0, \ldots, 9\}\{0, \ldots, 9\}^*$
- Die Menge aller Dezimaldarstellungen ganzer Zahlen:
  $L_{int} = \{-, \varepsilon\} L_{nat}$
- Die Menge aller dezimalen Festpunktzahlen:
  $L_{fix} = L_{int}\{.\} \cup \{.\} L_{nat} \cup L_{int}\{.\} L_{nat}$
- Die Menge aller dezimalen Fließpunktzahlen:
  $L_{float} = L_{int}\{e, E\} L_{int} \cup L_{fix}(\{\varepsilon\} \cup \{e, E\} L_{int})$

Zeichen, Wörter, Sprachen

Die Menge aller Wörter über dem Alphabet $\{0,1\}$, ...

- ... die mit 00 beginnen:
  $L_1 = \{00\}\{0,1\}^*$
- ... die 00 als Teilwort enthalten:
  $L_2 = \{0,1\}^*\{00\}\{0,1\}^*$
- ... die mindestens zwei Nullen enthalten:
  $L_3 = \{0,1\}^*\{0\}\{0,1\}^*\{0\}\{0,1\}^* = \{1\}^*\{0\}\{1\}^*\{0\}\{0,1\}^*$
- ... die genau zwei Nullen enthalten:
  $L_4 = \{1\}^*\{0\}\{1\}^*\{0\}\{1\}^*$
- ... die höchstens zwei Nullen enthalten:
  $L_5 = \{1\}^*\{0,\varepsilon\}\{1\}^*\{0,\varepsilon\}\{1\}^*$
- ... deren erstes und letztes Zeichen übereinstimmen:
  $L_6 = \{0,1\} \cup \{0\}\{0,1\}^*\{0\} \cup \{1\}\{0,1\}^*\{1\}$

Reguläre Sprachen

Definition 2.1 *Die Menge aller regulären Sprachen über Σ ist induktiv definiert durch:*

- *Jede endliche Menge $L \subseteq \Sigma^*$ ist regulär.*
- *Wenn $L_1, L_2 \subseteq \Sigma^*$ regulär sind, dann sind auch $L_1 \cup L_2$ und $L_1 \circ L_2$ regulär.*
- *Wenn L regulär ist, dann ist auch L^* regulär.*

Mit anderen Worten:

Eine Sprache ist genau dann regulär, wenn sie sich durch wiederholte Anwendung der Operatoren $\cup, \circ$ und * aus endlichen Sprachen aufbauen lässt.

Damit haben wir eine *endliche Beschreibung* für jede reguläre Sprache.

Beispiele: $L_{nat}, L_{int}, L_{fix}, L_{float}, L_1, \ldots, L_6$ sind reguläre Sprachen.

Reguläre Sprachen

Die 'Mengenausdrücke' aus Definition 2.1 sind gut lesbare Beschreibungen für reguläre Sprachen (zumindest für die Beispiele aus der Praxis wie Zahldarstellungen, Variablennamen, ...)

Aber:

- Nachteil in der Praxis:
  Ein Mengenausdruck für eine Sprache liefert noch keinen Algorithmus, um die Zugehörigkeit zur Sprache zu testen.
- Nachteil in der Theorie:
  Mit Definition 2.1 kann man nur schwer zeigen, dass eine Sprache *nicht* regulär ist.

Deshalb:

- Alternative Charakterisierungen der regulären Sprachen
- Insbesondere *algorithmische* Charakterisierungen

Endliche Automaten

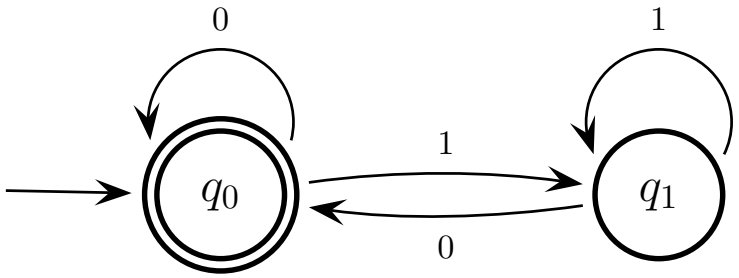
Ein *endlicher Automat* ist eine 'Maschine', die (nur) *endlich viele Zustände* annehmen kann. Einer der Zustände heißt *Startzustand*, einige Zustände heißen *Endzustände* (besser: *akzeptierende Zustände*).

Arbeitsweise

- Der Automat erhält ein Wort w als Eingabe.
- Zu Beginn befindet er sich im Startzustand.
- Er liest w zeichenweise von links nach rechts, wobei jedes Zeichen einen Zustandsübergang bewirkt.
- Ist er nach Abarbeitung von w in einem Endzustand, so wird w akzeptiert, andernfalls wird w abgelehnt.

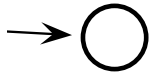
Ein endlicher Automat kann also dazu benutzt werden, die Zugehörigkeit zu einer Sprache zu testen.

Graphische Darstellung eines endlichen Automaten



Konvention:


sind die Zustände


ist der Startzustand

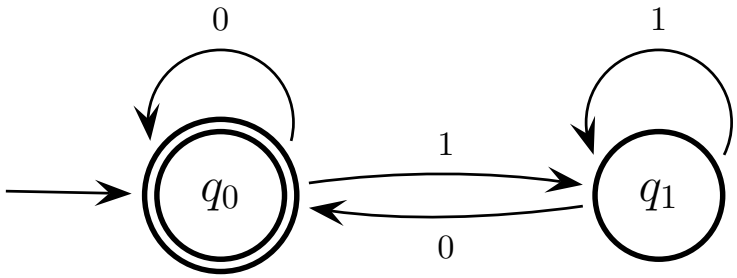

sind die Endzustände

- **Akzeptierte Wörter:** 0, 010, 01100
- **Abgelehnte Wörter:** 1, 101, 11011

Definition 2.2 *Ein deterministischer endlicher Automat (kurz: DEA) ist ein 5-Tupel $A = (\Sigma, Q, s, F, \delta)$ mit:*

- Σ *ist ein Alphabet*
- Q *ist eine endliche Menge, deren Elemente wir Zustände nennen*
- $s \in Q$ *ist der sogenannte Startzustand*
- $F \subseteq Q$ *ist die Menge der sogenannten Endzustände oder akzeptierenden Zustände*
- $\delta : Q \times \Sigma \to Q$ *ist die sogenannte Übergangsfunktion (totale Funktion!)*

Beispiel (wie oben):



$A = (\Sigma, Q, s, F, \delta)$ mit

- $\Sigma = \{0, 1\}$
- $Q = \{q_0, q_1\}$
- $s = q_0$
- $F = \{q_0\}$

- $\delta : Q \times \Sigma \to Q$
  $(q_0, 0) \mapsto q_0$
  $(q_0, 1) \mapsto q_1$
  $(q_1, 0) \mapsto q_0$
  $(q_1, 1) \mapsto q_1$

Oder: δ als Tabelle

	0	1
q_0	q_0	q_1
q_1	q_0	q_1

Die bisherige Definition liefert eine *statische Beschreibung* für DEAs. Es ist nur festgelegt, aus welchen "Teilen" ein DEA besteht (Alphabet, Zustände, Startzustand, Endzustände und Übergangsfunktion).

Jetzt muss noch das *dynamische Verhalten* eines DEA beschrieben werden, d.h. die einzelnen *Rechenschritte*, die er beim Lesen eines Wortes durchführt.

Definition 2.3 *Sei $A = (\Sigma, Q, s, F, \delta)$ ein DEA.*

- *Eine Konfiguration von A ist ein Element $(q, w) \in Q \times \Sigma^*$.*
- *Auf der Menge der Konfigurationen von A definieren wir die Übergangsschrittrelation $\vdash_A$ durch*

$$(q, w) \vdash_A (q', w') \;\Leftrightarrow\; \text{es existiert ein } a \in \Sigma \text{ mit } w = aw' \text{ und } \delta(q, a) = q'$$

Intuition

- Eine Konfiguration (q, w) beschreibt die Situation des Automaten zu einem bestimmten Zeitpunkt: Er befindet sich aktuell im Zustand q und hat noch das (Rest-)Wort w zu lesen.
- $\vdash_A$ beschreibt den *Übergangsschritt* von einem Zeitpunkt zum nächsten: Das erste Zeichen a des Wortes w wird gelesen (falls $w \neq \varepsilon$) und der Automat wechselt vom aktuellen Zustand q in den Folgezustand $q' = \delta(q, a)$. Vom Wort $w = aw'$ bleibt also das Wort w' übrig.

Warnung

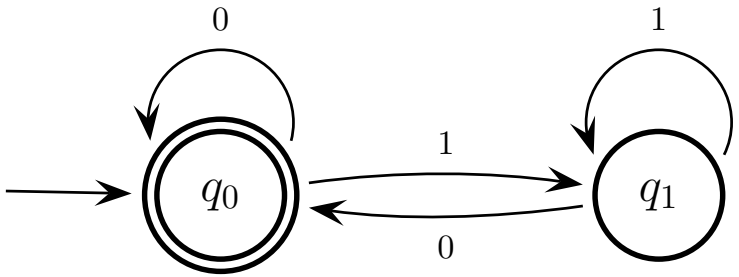
- Die Übergangsfunktion $\delta : Q \times \Sigma \to Q$ gehört zur *statischen* Beschreibung des Automaten A.
- Die Übergangsschrittrelation $\vdash_A \subseteq (Q \times \Sigma^*)^2$ ist zwar mit Hilfe von δ definiert, beschreibt aber das *dynamische* Verhalten des Automaten.

Definition 2.4 *Sei* $A = (\Sigma, Q, s, F, \delta)$ *ein DEA. Aufbauend auf der Übergangsschrittrelation* $\vdash_A$ *definieren wir Relationen* $\vdash_A^n$ $(n \geq 0)$, $\vdash_A^+$ *und* $\vdash_A^*$ *auf der Menge* $Q \times \Sigma^*$ *aller Konfigurationen durch*

- $(q, w) \vdash_A^n (q', w') \Leftrightarrow$ *es existieren* $(q_0, w_0), \ldots, (q_n, w_n) \in Q \times \Sigma^*$ *mit* $(q, w) = (q_0, w_0), (q', w') = (q_n, w_n)$ *und* $(q_0, w_0) \vdash_A \ldots \vdash_A (q_n, w_n)$.
- $(q, w) \vdash_A^+ (q', w') \Leftrightarrow$ *es existiert ein* $n > 0$ *mit* $(q, w) \vdash_A^n (q', w')$
- $(q, w) \vdash_A^* (q', w') \Leftrightarrow$ *es existiert ein* $n \geq 0$ *mit* $(q, w) \vdash_A^n (q', w')$

$\vdash_A^n$ steht also für eine Folge von n Übergangsschritten, $\vdash_A^+$ für eine nichtleere Folge und $\vdash_A^*$ für eine beliebige (möglicherweise leere) Folge von Übergangsschritten.

Beispiel (wie oben):



Eine mögliche Folge von Übergangsschritten:

$$
\begin{aligned}
(q_0, 1001) &\vdash_A (q_1, 001) && \text{wegen } \delta(q_0, 1) = q_1 \\
&\vdash_A (q_0, 01) && \text{wegen } \delta(q_1, 0) = q_0 \\
&\vdash_A (q_0, 1) && \text{wegen } \delta(q_0, 0) = q_0 \\
&\vdash_A (q_1, \varepsilon) && \text{wegen } \delta(q_0, 1) = q_1
\end{aligned}
$$

Also: $(q_0, 1001) \vdash_A^* (q_1, \varepsilon)$

Oder: $(q_0, 1001) \vdash_A^* (q_0, 1)$

Definition 2.5 *Sei $A = (\Sigma, Q, s, F, \delta)$ ein DEA.*

- *Ein Wort $w \in \Sigma^*$ wird von A akzeptiert wenn es einen Zustand $q \in F$ gibt mit $(s, w) \vdash_A^* (q, \varepsilon)$.*
- *Die von A akzeptierte (oder: erkannte) Sprache $L(A)$ ist definiert durch*

$$\begin{aligned} L(A) &= \{w \in \Sigma^* \mid w \text{ wird von } A \text{ akzeptiert}\} \\ &= \{w \in \Sigma^* \mid \text{es existiert ein } q \in F \text{ mit } (s, w) \vdash_A^* (q, \varepsilon)\} \end{aligned}$$

Beispiel

Für den oben definierten Automaten A gilt:

$$\begin{aligned} L(A) &= \{w \in \{0,1\}^* \mid (q_0, w) \vdash^* (q_0, \varepsilon)\} \\ &= \{w \in \{0,1\}^* \mid w \text{ endet auf } 0\} \cup \{\varepsilon\} \end{aligned}$$

Warum schon wieder ^{n},$^+$,*?
Wo ist das Monoid?

Definition 2.6 *Sei M eine Menge (z.B. $M = Q \times \Sigma^*$). Die Komposition $R_1 \circ R_2$ zweier Relationen $R_1, R_2 \subseteq M \times M$ ist definiert durch*

$$R_1 \circ R_2 = \{(x,y) \in M \times M \mid \text{es existiert ein } z \in M \text{ mit } (x,z) \in R_1 \text{ und } (z,y) \in R_2\}$$

- Dann ist $(\wp(M \times M), \circ)$ ein Monoid.
- Neutrales Element ist die *Identität* $id_M = \{(x,x) \mid x \in M\}$.
- Also können wir Potenzen R^n von Relationen $R \subseteq M \times M$ wie üblich definieren (insbesondere $\vdash_A^n$).
- Und weil auch Relationen Mengen sind, definieren wir analog zu Sprachen: $R^+ = \bigcup_{n>0} R^n$ und $R^* = \bigcup_{n\geq 0} R^n$ (insbesondere $\vdash_A^+$ und $\vdash_A^*$)

Lemma 2.7 *Sei M eine Menge und $R \subseteq M \times M$ eine zweistellige Relation. Dann gilt:*

- $R^+ = \bigcup_{n>0} R^n$ *ist die kleinste transitive Relation, die R enthält (d.h. für jede andere transitive Relation R' mit $R \subseteq R'$ gilt auch $R^+ \subseteq R'$.)*
- $R^* = \bigcup_{n\geq 0} R^n$ *die kleinste reflexive transitive Relation, die R enthält (d.h. für jede andere reflexive transitive Relation R' mit $R \subseteq R'$ gilt auch $R^* \subseteq R'$.)*

Definition 2.8

- R^+ *heißt* transitiver Abschluss *der Relation R.*
- R^* *heißt* reflexiver transitiver Abschluss *der Relation R.*

Nächstes Ziel:

"Inhaltliches" Verständnis von endlichen Automaten

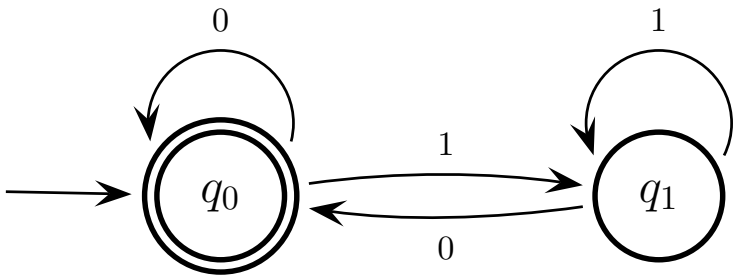
Fragestellungen

- Wie bestimmt man die Sprache $L(A)$ zu einem vorgegebenen Automaten A?
- Wie beweist man, dass ein Automat tatsächlich die gewünschte Sprache akzeptiert?
- Wie findet man einen Automaten, der eine vorgegebene Sprache akzeptiert?
- Wie findet man heraus, ob es überhaupt einen solchen Automaten für eine vorgegebene Sprache gibt?

Dazu: Grundlegende Überlegungen

- Jeder endliche Automat hat nur ein endliches “Gedächtnis”, nämlich seine endlich vielen Zustände.
- Die einzige Information über das bereits gelesene Anfangsstück eines Wortes steckt im aktuellen Zustand.
- Diese begrenzte Information muss ausreichen, um den Rest des Wortes ‘richtig’ abzuarbeiten.

Beispiel (wie oben):



Welche Information über das bereits gelesene Wort w steckt in den Zuständen q_0 bzw. q_1?

- q_0 bedeutet: Entweder $w = \varepsilon$ oder w endet auf 0.
- q_1 bedeutet: w endet auf 1.

Setzt man diese Bedeutung der Zustände voraus, so ist klar, wie der Automat auf das nächste Zeichen reagieren muss:

- Mit 0 muss er stets in q_0 übergehen, da $w0$ auf 0 endet.
- Mit 1 muss er stets in q_1 übergehen, da $w1$ auf 1 endet.

Entwurf eines Automaten für eine vorgegebene Sprache

- Man macht sich klar, welche Information über das bereits gelesene Anfangswort nötig ist, um mit dem Restwort 'richtig' weiterzuarbeiten?
- Reicht eine endliche Anzahl n von unterschiedlichen Informationen aus, so entwirft man einen Automaten mit n Zuständen.
- Die Zustandsübergänge legt man so fest, dass nach dem Lesen eines Zeichens stets wieder die richtige Information vorliegt.
- Reicht eine endliche Anzahl unterschiedlicher Informationen *nicht* aus, so gibt es keinen endlichen Automaten für die Sprache.

Beispiel: Automat A_{int} für die Sprache

$$L_{int} = \{\varepsilon, -\}\{0, \ldots, 9\}^+ = \{\varepsilon, -\}\{0, \ldots, 9\}\{0, \ldots, 9\}^*$$

Sei $\Sigma = \{-, 0, \ldots, 9\}$. Man muss folgende Fälle für das bereits gelesene Wort w unterscheiden:

- $w = \varepsilon$: Noch nichts gelesen, die gesamte Zahl wird noch erwartet.
- $w = -$: Nur das Vorzeichen gelesen, es wird noch eine *nichtleere* Ziffernfolge erwartet.
- $w \in L_{int} = \{\varepsilon, -\}\{0, \ldots, 9\}^+$: Schon mindestens eine Ziffer gelesen, jetzt darf noch eine *beliebige* Ziffernfolge kommen.
- w enthält '$-$' an der falschen Stelle, d.h. nicht an erster Position: Schon alles verdorben!

Also 4 Zustände für diese 4 Situationen: $q_\varepsilon, q_-, q_{int}, q'$

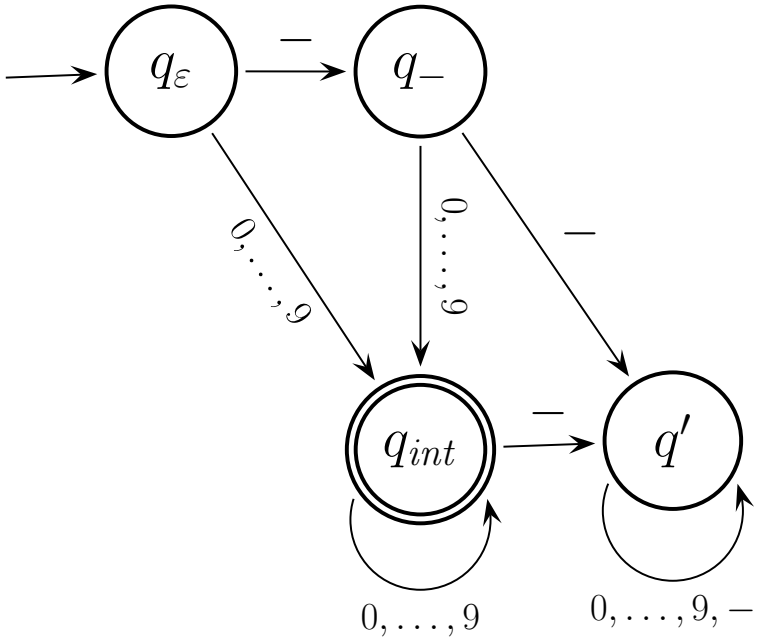
Endliche Automaten

Deshalb definiert man $A_{int} = (\Sigma, Q, s, F, \delta)$, wobei

- $Q = \{q_\varepsilon, q_-, q_{int}, q'\}$
- $s = q_\varepsilon$
- $F = \{q_{int}\}$
- $\delta : Q \times \Sigma \to Q$

$$
\begin{aligned}
(q_\varepsilon, -) &\mapsto q_- \\
(q_\varepsilon, a) &\mapsto q_{int} \quad \text{für alle } a \in \{0, \ldots, 9\} \\
(q_-, -) &\mapsto q' \\
(q_-, a) &\mapsto q_{int} \quad \text{für alle } a \in \{0, \ldots, 9\} \\
(q_{int}, -) &\mapsto q' \\
(q_{int}, a) &\mapsto q_{int} \quad \text{für alle } a \in \{0, \ldots, 9\} \\
(q', a) &\mapsto q' \quad \text{für alle } a \in \Sigma \text{ (Sackgasse!)}
\end{aligned}
$$

Graphische Darstellung



Vereinbarung: Ein Pfeil mit mehreren Zeichen $a_1, \ldots, a_n$ ist Abkürzung für n Pfeile mit den einzelnen Zeichen.

Endliche Automaten

Wie beweist man, dass $L(A_{int}) = L_{int}$? Man zeigt, dass jedes $w \in \Sigma^*$ zum gewünschten Zustand führt, d.h.:

1. Wenn $w = \varepsilon$, dann $(q_\varepsilon, w) \vdash^* (q_\varepsilon, \varepsilon)$.
2. Wenn $w = -$, dann $(q_\varepsilon, w) \vdash^* (q_-, \varepsilon)$.
3. Wenn $w \in L_{int}$, dann $(q_\varepsilon, w) \vdash^* (q_{int}, \varepsilon)$.
4. Wenn $w \notin L_{int} \cup \{\varepsilon, -\}$, dann $(q_\varepsilon, w) \vdash^* (q', \varepsilon)$.

Dies geschieht durch Induktion über $|w|$:
$|w| = 0$, d.h. $w = \varepsilon$: Es gilt $(q_\varepsilon, w) \vdash^0 (q_\varepsilon, \varepsilon)$.
$|w| > 0$, d.h. $w = va$: Fallunterscheidung für v. Wenn $v \in L_{int}$, dann gilt nach Induktionsannahme $(q_\varepsilon, v) \vdash^* (q_{int}, \varepsilon)$. Ist $a \in \{0, \ldots, 9\}$, dann folgt $(q_\varepsilon, w) = (q_\varepsilon, va) \vdash^* (q_{int}, a) \vdash (q_{int}, \varepsilon)$ und das ist der gewünschte Zustand, weil $w = va \in L_{int}$. Ist $a = -$, dann folgt $(q_\varepsilon, w) = (q_\varepsilon, va) \vdash^* (q_{int}, a) \vdash (q', \varepsilon)$ und das ist wieder der richtige Zustand, weil $w \notin L_{int} \cup \{\varepsilon, -\}$. Ähnlich für die übrigen Fälle! □

Beispiel: Ein Automat A für die Sprache $L = \{a^n b^n \mid n \geq 0\}$?

Intuition:

- Um $w \in L$ zu testen, müsste ein endlicher Automat (insbesondere) die Anzahl der a's und b's in w vergleichen.
- Also muss er stets die Anzahl der bisher gelesenen a's kennen.
- Da diese Anzahl beliebig groß werden kann, reichen die endlich vielen Zustände zum Speichern dieser Information nicht aus.
- Also gibt es keinen Automaten A mit $L = L(A)$.

Schlagwort: "Ein endlicher Automat kann nicht zählen."
(Soll heißen: Er kann nicht beliebig hoch zählen).

Exakter Beweis:

Angenommen, es existiert ein DEA $A = (\Sigma, Q, s, F, \delta)$ mit $L(A) = L$.

Wir betrachten alle Wörter der Form a^n mit $n \geq 0$.

Für jedes $n \geq 0$ existiert ein Zustand $q \in Q$ mit $(s, a^n) \vdash_A^* (q, \varepsilon)$ (der Zustand, der durch Lesen von a^n erreicht wird). Da es unendlich viele Wörter a^n gibt, aber nur endlich viele $q \in Q$, muss es (mindestens) zwei Wörter a^i, a^j $(i \neq j)$ geben, mit denen man den *gleichen* Zustand q erreicht, d.h. $(s, a^i) \vdash_A^* (q, \varepsilon)$ und $(s, a^j) \vdash_A^* (q, \varepsilon)$.

Sei $q' \in Q$ mit $(q, b^i) \vdash_A^* (q', \varepsilon)$. Dann gilt $(s, a^i b^i) \vdash_A^* (q, b^i) \vdash_A^* (q', \varepsilon)$ und $(s, a^j b^i) \vdash_A^* (q, b^i) \vdash_A^* (q', \varepsilon)$. Aber $a^i b^i \in L$ und $a^j b^i \notin L$, d.h. sie dürfen nicht beide zum gleichen Zustand q' führen.

Damit ist die Annahme zum Widerspruch geführt, d.h. es existiert kein DEA A mit $L(A) = L$. $\square$

Im Beweis wurde benutzt:

- Das *Schubfachprinzip* (engl. *pigeonhole principle*):
  Wenn man eine Menge von Dingen auf eine gewisse Anzahl von Schubfächern verteilt, und die Anzahl der Dinge ist größer als die Anzahl der Schubfächer, dann landen mindestens zwei Dinge im gleichen Schubfach.
  Hier: Unendlich viele Wörter a^n wurden auf endlich viele Zustände verteilt.
- Folgende Eigenschaft von $\vdash_A^*$:
  Wenn $(q, u) \vdash_A^* (q', \varepsilon)$, dann gilt auch $(q, uv) \vdash_A^* (q', v)$.

Lemma 2.9 *Sei* $A = (\Sigma, Q, s, F, \delta)$ *ein DEA, seien* $u, v \in \Sigma^*$ *und* $q, q' \in Q$. *Dann gilt:*

1. $(q, u) \vdash_A^* (q', \varepsilon) \Leftrightarrow (q, uv) \vdash_A^* (q', v)$
2. $(q, uv) \vdash_A^* (q', \varepsilon) \Leftrightarrow$ *es existiert ein* $q'' \in Q$ *mit* $(q, uv) \vdash_A^* (q'', v) \vdash_A^* (q', \varepsilon)$

Begründung:

1. *Die Übergangsschritte, die* A *bei Abarbeitung eines Wortes* u *macht, werden von einem hinter* u *stehenden Wort* v *nicht beeinflusst. Denn* A *arbeitet ja von links nach rechts, und kann deshalb das Wort* v *während der Abarbeitung von* u *gar nicht sehen.*
2. *Jede Folge von Übergangsschritten für ein Wort* uv *lässt sich aufteilen in die Schritte für* u *und die Schritte für* v. *Das liegt daran, dass der Automat zeichenweise von links nach rechts liest.*

Frage:
Für welche Sprachen L existiert ein DEA A mit $L(A) = L$?

Wir werden beweisen:
Ein DEA A mit $L = L(A)$ existiert genau dann, wenn L eine reguläre Sprache ist.

Die Funktionsschreibweise δ^*

Sei $A = (\Sigma, Q, s, F, \delta)$ ein DEA.

Dann existiert für jede Konfiguration $(q, w) \in Q \times \Sigma^*$ *genau ein* Zustand $q' \in Q$ mit $(q, w) \vdash_A^* (q', \varepsilon)$ (weil für jedes einzelne Zeichen von w stets ein eindeutiger Übergangsschritt existiert).

Diesen eindeutigen Zustand q' bezeichnen wir mit $\delta^*(q, w)$.

Die so definierte Funktion $\delta^* : Q \times \Sigma^* \to Q$ lässt sich auch ohne den Umweg über $\vdash_A^*$ definieren, nämlich durch Induktion über die Länge von w:

- $\delta^*(q, \varepsilon) = q$
- $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$

oder:

- $\delta^*(q, aw) = \delta^*(\delta(q, a), w)$

Vor- und Nachteile beider Schreibweisen:

In der Funktionsschreibweise haben wir eine eigenständige Bezeichnung für den Ergebniszustand q' (in der die Existenz und Eindeutigkeit implizit enthalten ist). In der Relationsschreibweise müssen wir immer erst die Existenz (und evtl. Eindeutigkeit) von q' erwähnen (und damit einen neuen Namen für einen Zustand einführen).

In der Relationsschreibweise kann man Übergangsschritte besser aneinanderhängen und man hat die Schreibweisen $\vdash^n, \vdash^+, \vdash^*$ mit schönen Gesetzmäßigkeiten (Monoid!). Mit der Funktionsschreibweise geht das nicht so einfach (denn $\delta : Q \times \Sigma \to Q$ kann man nicht mit sich selbst verknüpfen).

Konventionen

Als typische Bezeichnungen verwenden wir (bisher):

- a, b, c für Zeichen
- u, v, w, x, y, z für Wörter
- Σ für ein Alphabet
- L für eine Sprache
- A, B für Automaten
- p, q, r, s für Zustände, speziell s für den Startzustand
- P, Q, F für Zustandsmengen, speziell Q für die Menge aller Zustände, F für die Menge der Endzustände

Bisher: Determistische endliche Automaten

- Es gibt *genau einen* Startzustand s.
- Für jeden Zustand $q \in Q$ und jedes Zeichen $a \in \Sigma$ gilt:

  Von q geht *genau ein* mit a markierter Pfeil aus.

  Formal: $\delta : Q \times \Sigma \to Q$ ist eine totale Funktion.

Jetzt: Nichtdeterministische endliche Automaten

- Es darf *beliebig viele* Startzustände geben.
- Für jeden Zustand $q \in Q$ und jedes Zeichen $a \in \Sigma$ gilt:

  Von q dürfen *beliebig viele* mit a markierte Pfeile ausgehen.

  Formal: An die Stelle der Übergangsfunktion $\delta : Q \times \Sigma \to Q$ tritt eine *Übergangsrelation* $\Delta \subseteq Q \times \Sigma \times Q$.

Definition 2.10 *Ein* nichtdeterministischer endlicher Automat *(kurz:* NDEA*) ist ein 5-Tupel* $A = (\Sigma, Q, S, F, \Delta)$ *mit:*

- Σ, Q, F *wie beim DEA*
- $S \subseteq Q$ *ist die Menge der sogenannten* Startzustände
- $\Delta \subseteq Q \times \Sigma \times Q$ *ist die sogenannte* Übergangsrelation

Auch hier gilt wieder:

Definition 2.10 liefert nur die *statische Beschreibung* eines NDEA. Es bleibt noch zu klären, wie sich der NDEA *zur Laufzeit* verhält, d.h.

- wie er ein Eingabewort schrittweise verarbeitet,
- welche Wörter er akzeptiert (d.h. welche Sprache er erkennt).

Definition 2.11

- *Die Übergangsschrittrelation* $\vdash_A$ *auf der Menge* $Q \times \Sigma^*$ *aller Konfigurationen von* A *ist definiert durch*

$$(q, w) \vdash_A (q', w') \;\Leftrightarrow\; \text{es existiert ein } a \in \Sigma \text{ mit } w = aw' \text{ und } (q, a, q') \in \Delta$$

- *Ein Wort* $w \in \Sigma^*$ *wird von* A *akzeptiert wenn Zustände* $s \in S$ *und* $q \in F$ *existieren mit* $(s, w) \vdash_A^* (q, \varepsilon)$.
- *Die von* A *akzeptierte Sprache* $L(A)$ *ist wieder definiert durch*

$$L(A) \;=\; \{w \in \Sigma^* \mid w \text{ wird von } A \text{ akzeptiert}\}$$

Unterschied zum DEA:

- An die Stelle des *eindeutigen* Zustands $q' = \delta(q, a)$ beim DEA tritt ein *beliebiger* Zustand q' mit $(q, a, q') \in \Delta$.
- An die Stelle des *eindeutigen* Startzustands s beim DEA tritt ein *beliebiger* Startzustand $s \in S$.

Alternative Formulierung:

Definition 2.12 *Sei $A = (\Sigma, Q, S, F, \Delta)$ ein NDEA und sei $w \in \Sigma^*$.*

1. *Ein Lauf des Automaten A für das Wort w ist eine Folge von Übergangsschritten*

$$(s, w) \vdash_A \ldots \vdash_A (q, w')$$

   die mit einem Startzustand s beginnt und sich nicht weiter fortsetzen lässt.

2. *Ein Lauf heißt akzeptierend, wenn $q \in F$ und $w' = \varepsilon$.*

Mit diesen neuen Begriffen gilt: Ein Wort $w \in \Sigma^*$ wird genau dann von A akzeptiert, wenn (mindestens) ein akzeptierender Lauf von A für w existiert. Auf die übrigen Läufe von A für w kommt es dann nicht mehr an.

Intuition:

Wenn es einen akzeptierenden Lauf für das Wort w gibt, dann "errät" der NDEA diesen akzeptierenden Lauf.

Wie schafft es der NDEA, immer richtig zu raten?

Diese Frage interessiert uns (zunächst) nicht. Wir verlassen uns (nur) auf die mathematische Definition des Akzeptierens.

Vergleich zur Literatur:

In der Literatur wird oft nur *ein* Startzustand bei einem NDEA zugelassen. Wir lassen hier beliebig viele Startzustände zu, weil es

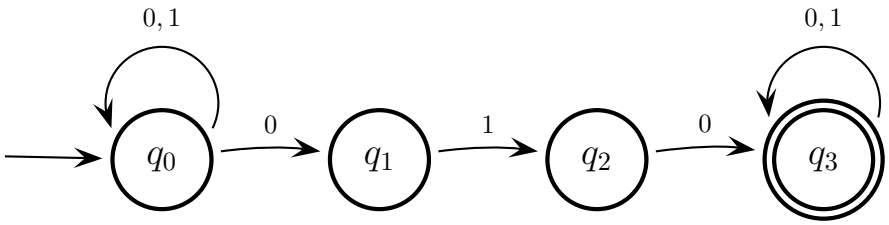
- manchmal nützlich ist
- die Theorie nicht schwieriger macht
- sogar konsequenter ist als die Beschränkung auf einen einzelnen Startzustand.

Beispiel:

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

$L = \{w \in \{0,1\}^* \mid 010 \text{ ist Teilwort von } w\}$.

Ein NDEA A mit $L(A) = L$ ist:



Formal:

$A = (\Sigma, Q, S, F, \Delta)$ mit

$\Sigma = \{0, 1\}$, $Q = \{q_0, q_1, q_2, q_3\}$, $S = \{q_0\}$, $F = \{q_3\}$ und

$\Delta = \{\ (q_0, 0, q_0), (q_0, 1, q_0), (q_0, 0, q_1), (q_1, 1, q_2), (q_2, 0, q_3),$
$(q_3, 0, q_3), (q_3, 1, q_3)\}$

Wieso gilt $L = L(A)$?

Intuition:

Weil der NDEA A die Stelle, an der das Teilwort 010 beginnt, *erraten* und dann mit 010 in den Endzustand q_3 übergehen kann.

Exakter Beweis:

'$\subseteq$': Sei $w \in L$.

Dann existieren $u, v \in \{0,1\}^*$ mit $w = u010v$.

Also gilt $(q_0, w) = (q_0, u010v) \vdash_A^* (q_0, 010v) \vdash_A^* (q_3, v) \vdash_A^* (q_3, \varepsilon)$.

Damit ist $w \in L(A)$ bewiesen, weil $q_3 \in F$.

'$\supseteq$': Sei $w \in L(A)$.

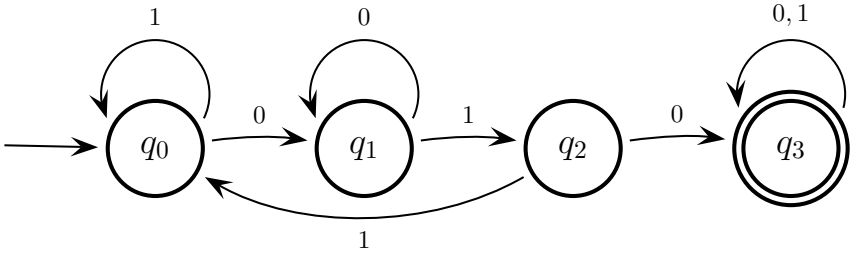
Dann muss $(q_0, w) \vdash_A^* (q_3, \varepsilon)$ gelten, weil q_0 einziger Startzustand und q_3 einziger Endzustand ist.

Da man nur mit 010 von q_0 nach q_3 gelangen kann, muss w das Wort 010 enthalten, d.h. $w \in L$. $\square$

Wieso ist der Beweis so einfach?

Weil dieser NDEA auf einer sehr einfachen Idee beruht.

Zum Vergleich: Ein DEA A' für die gleiche Sprache



Beweisidee:

Sei w das bisher gelesene Wort.

q_0 bedeutet: $w = \varepsilon$, $w = 1$ oder w endet auf 11, aber enthält nicht 010.

q_1 bedeutet: w endet auf 0, aber enthält nicht 010.

q_2 bedeutet: w endet auf 01, aber enthält nicht 010.

q_3 bedeutet: w enthält 010, d.h. $w \in L$.

Fazit:

- Oft findet man leichter einen NDEA als einen DEA für eine vorgegebene Sprache (weil man mehr Freiheiten hat)
- und auch der Korrektheitsbeweis ist oft einfacher (wenn der NDEA auf einer einfacheren Idee beruht).

Wie kann man einen NDEA implementieren?

- Als nichtdeterministisches Programm mit Zufallsgenerator?

  Nein, denn man kann nicht erwarten, dass das Programm den richtigen Weg errät.

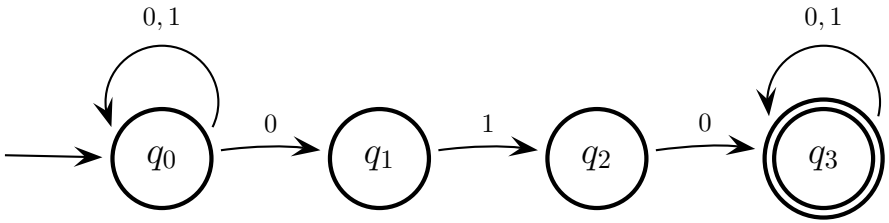
- Durch einen backtracking-Algorithmus, der alle möglichen Wege nacheinander durchspielt?

  Ja, aber man muss sich überlegen, dass es nur endlich viele mögliche Wege gibt, und wie man sie systematisch ausprobiert.

- Durch einen "parallelen" Algorithmus, der alle möglichen Wege gleichzeitig durchspielt?

  Das ist die einfachste Möglichkeit: Man führt einfach Buch über die *Menge* der Zustände, die vom Startzustand aus mit dem bereits gelesenen Wort erreichbar sind. Diese Menge kann auch *leer* sein!

Beispiel: $w = 0011010$ **auf dem NDEA** A



Menge der möglichen Zustände nach Lesen von

- $\varepsilon : \{q_0\}$
- $0 : \{q_0, q_1\}$
- $00 : \{q_0, q_1\}$
- $001 : \{q_0, q_2\}$
- $0011 : \{q_0\}$
- $00110 : \{q_0, q_1\}$
- $001101 : \{q_0, q_2\}$
- $0011010 : \{q_0, q_1, q_3\}$, also $w \in L(A)$, da q_3 mit w erreichbar

Fazit:

- Wir haben einen *determistischen* Algorithmus für einen *N*DEA A,
  - bei dem man das Eingabewort w zeichenweise von links nach rechts liest,
  - bei dem man sich aber Zustands*mengen* anstelle von einzelnen Zuständen merken muss.
- Diesen Algorithmus kann man "auf einem *DEA* simulieren".
- Man muss den DEA nur mit einem hinreichend großen Gedächtnis ausstatten, damit er sich beliebige *Teilmengen* der Zustandsmenge Q des NDEA A merken kann.
- Als Zustände des DEA wählen wir deshalb alle Teilmengen von Q, und legen dann die Übergänge zwischen diesen Zustandsmengen passend fest.
- So erhalten wir einen zu A *äquivalenten* DEA, d.h. einen DEA A' mit $L(A) = L(A')$.

Definition 2.13 *Sei $A = (\Sigma, Q, S, F, \Delta)$ ein NDEA. Der Potenzautomat zu A ist definiert als der DEA $A' = (\Sigma, Q', s', F', \delta)$ mit*

- $Q' = \wp(Q)$, *die Potenzmenge von* Q
- $s' = S \in \wp(Q)$
- $F' = \{P \in \wp(Q) \mid P \cap F \neq \emptyset\}$
  $= \{P \subseteq Q \mid P$ *enthält mindestens einen Zustand aus* $F\}$
- $\delta : \wp(Q) \times \Sigma \to \wp(Q)$

  $\delta(P, a) = \{q \in Q \mid$ *es existiert ein* $p \in P$ *mit* $(p, a, q) \in \Delta\}$

  (d.h. $\delta(P, a)$ *enthält genau die Zustände, die von einem Zustand* $p \in P$ *durch einen mit* a *markierten Pfeil erreichbar sind)*

Satz 2.14 *Sei A ein NDEA und A' der Potenzautomat zu A. Dann gilt $L(A) = L(A')$.*

Beweis

Sei $A = (\Sigma, Q, S, F, \Delta)$ und $A' = (\Sigma, Q', s', F', \delta)$.

Als erstes wird für alle $w \in \Sigma^*$ gezeigt

$$\delta^*(S, w) \;=\; \{q \in Q \mid \exists\, s \in S.\, (s, w) \vdash_A^* (q, \varepsilon)\}$$

d.h.

$$q \in \delta^*(S, w) \;\Leftrightarrow\; \exists\, s \in S.\, (s, w) \vdash_A^* (q, \varepsilon)$$

Intuition: $\delta^*(S, w)$ ist der Zustand, den der Potenzautomat A' von seinem Startzustand S aus mit dem Wort w erreicht. Die Äquivalenz besagt, dass $\delta^*(S, w)$ genau die Zustände enthält, die der NDEA A von *einem* seiner Startzustände aus mit dem Wort w erreichen *kann*, Das bedeutet, dass A'—wie gewünscht—über alle Möglichkeiten des NDEA A Buch führt.

Beweis durch Induktion über $|w|$:

- Für $w = \varepsilon$ ist zu zeigen

$$q \in \delta^*(S, \varepsilon) \;\Leftrightarrow\; \exists\, s \in S.\, (s, \varepsilon) \vdash_A^* (q, \varepsilon)$$

  Das ist klar weil $\delta^*(S, \varepsilon) = S$ per Definition von δ^* und $(s, \varepsilon) \vdash_A^* (q, \varepsilon) \Leftrightarrow s = q$ per Definition von $\vdash_A^*$

- Für $w = va$ ist zu zeigen

$$q \in \delta^*(S, va) \;\Leftrightarrow\; \exists\, s \in S.\, (s, va) \vdash_A^* (q, \varepsilon)$$

  Wegen $\delta^*(S, va) = \delta(\delta^*(S, v), a)$ gilt

$$\begin{aligned} q \in \delta^*(S, va) \;&\Leftrightarrow\; \exists\, p \in \delta^*(S, v).\, (p, a, q) \in \Delta \\ &\quad \text{per Definition von } \delta \text{ im Potenzautomaten } A' \\ &\Leftrightarrow\; \exists\, s \in S, p \in Q.\, (s, v) \vdash_A^* (p, \varepsilon) \wedge (p, a, q) \in \Delta \\ &\quad \text{nach Induktionsannahme für } v \\ &\Leftrightarrow\; (s, va) \vdash_A^* (q, \varepsilon) \\ &\quad \text{nach Lemma 2.9 (gilt auch für NDEAs)} \end{aligned}$$

Damit ist die gewünschte Äquivalenz für alle $w \in \Sigma^*$ bewiesen:

$$q \in \delta^*(S, w) \;\Leftrightarrow\; \exists\, s \in S.\, (s, w) \vdash_A^* (q, \varepsilon)$$

und es folgt:

$$\begin{aligned}
w \in L(A) \;&\Leftrightarrow\; \text{es existieren } s \in S, q \in F \text{ mit } (s, w) \vdash_A^* (q, \varepsilon) \\
&\qquad \text{per Definition der Sprache } L(A) \\
&\Leftrightarrow\; \delta^*(S, w) \text{ enthält einen Zustand } q \in F \\
&\qquad \text{nach der bewiesenen Äquivalenz} \\
&\Leftrightarrow\; \delta^*(S, w) \cap F \neq \emptyset \\
&\Leftrightarrow\; \delta^*(S, w) \in F' \\
&\qquad \text{per Definition von } F' \text{ im Potenzautomaten } A' \\
&\Leftrightarrow\; w \in L(A') \\
&\qquad \text{weil } s' = S \text{ der Startzustand von } A' \text{ ist}
\end{aligned}$$

Also ist $L(A) = L(A')$, d.h. A und A' sind äquivalent. □

Man beachte:

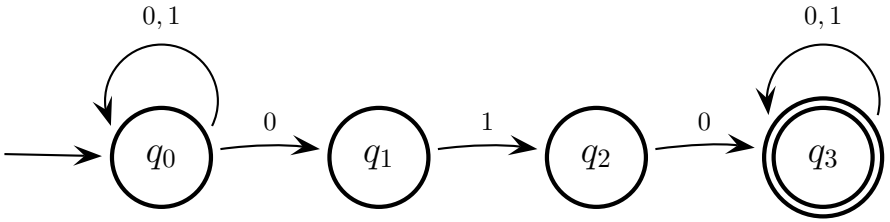
- Im Beweis wurde benutzt, dass Lemma 2.9 auch für NDEAs gilt.
- Die Definition des Potenzautomaten ist *konstruktiv*, d.h. man hat einen *Algorithmus*, um den Potenzautomaten A' aus dem NDEA A zu erhalten.
- Der Potenzautomat ist natürlich sehr groß, denn $|\wp(Q)| = 2^{|Q|}$. Oft sind aber viele Zustände überflüssig, so dass man sie nachträglich entfernen oder sogar von Anfang an weglassen kann.

Definition 2.15 *Sei* $A = (\Sigma, Q, s, F, \delta)$ *ein DEA. Ein Zustand* $q \in Q$ *heißt* erreichbar, *wenn es ein Wort* w *gibt mit* $(s, w) \vdash_A^* (q, \varepsilon)$, *ansonsten heißt er* unerreichbar.

- Unerreichbare Zustände kann man offensichtlich aus einem DEA entfernen, ohne dass sich die akzeptierte Sprache verändert (und ohne dass man die Definition eines DEA verletzt, denn s ist stets erreichbar, und δ lässt sich auf die Menge der erreichbaren Zustände einschränken.)
- Einige "Entwurfsmethoden" für Automaten, wie z.B. die Konstruktion des Potenzautomaten, kann man so abändern, dass der entworfene Automat von vornherein nur erreichbare Zustände enthält.

Beispiel

Sei A wieder der NDEA



Der Potenzautomat von A hat 16 Zustände. Wir bestimmen die erreichbaren unter ihnen.

Der Startzustand $\{q_0\}$ ist erreichbar.

$\delta(\{q_0\}, 0) = \{q_0, q_1\}$, also ist $\{q_0, q_1\}$ erreichbar.

$\delta(\{q_0\}, 1) = \{q_0\}$

$\delta(\{q_0, q_1\}, 0) = \{q_0, q_1\}$

$\delta(\{q_0, q_1\}, 1) = \{q_0, q_2\}$, also ist $\{q_0, q_2\}$ erreichbar.

$\delta(\{q_0, q_2\}, 0) = \{q_0, q_1, q_3\}$, also ist $\{q_0, q_1, q_3\}$ erreichbar.

$\delta(\{q_0, q_2\}, 1) = \{q_0\}$

$\delta(\{q_0, q_1, q_3\}, 0) = \{q_0, q_1, q_3\}$

$\delta(\{q_0, q_1, q_3\}, 1) = \{q_0, q_2, q_3\}$, also ist $\{q_0, q_2, q_3\}$ erreichbar.

$\delta(\{q_0, q_2, q_3\}, 0) = \{q_0, q_1, q_3\}$

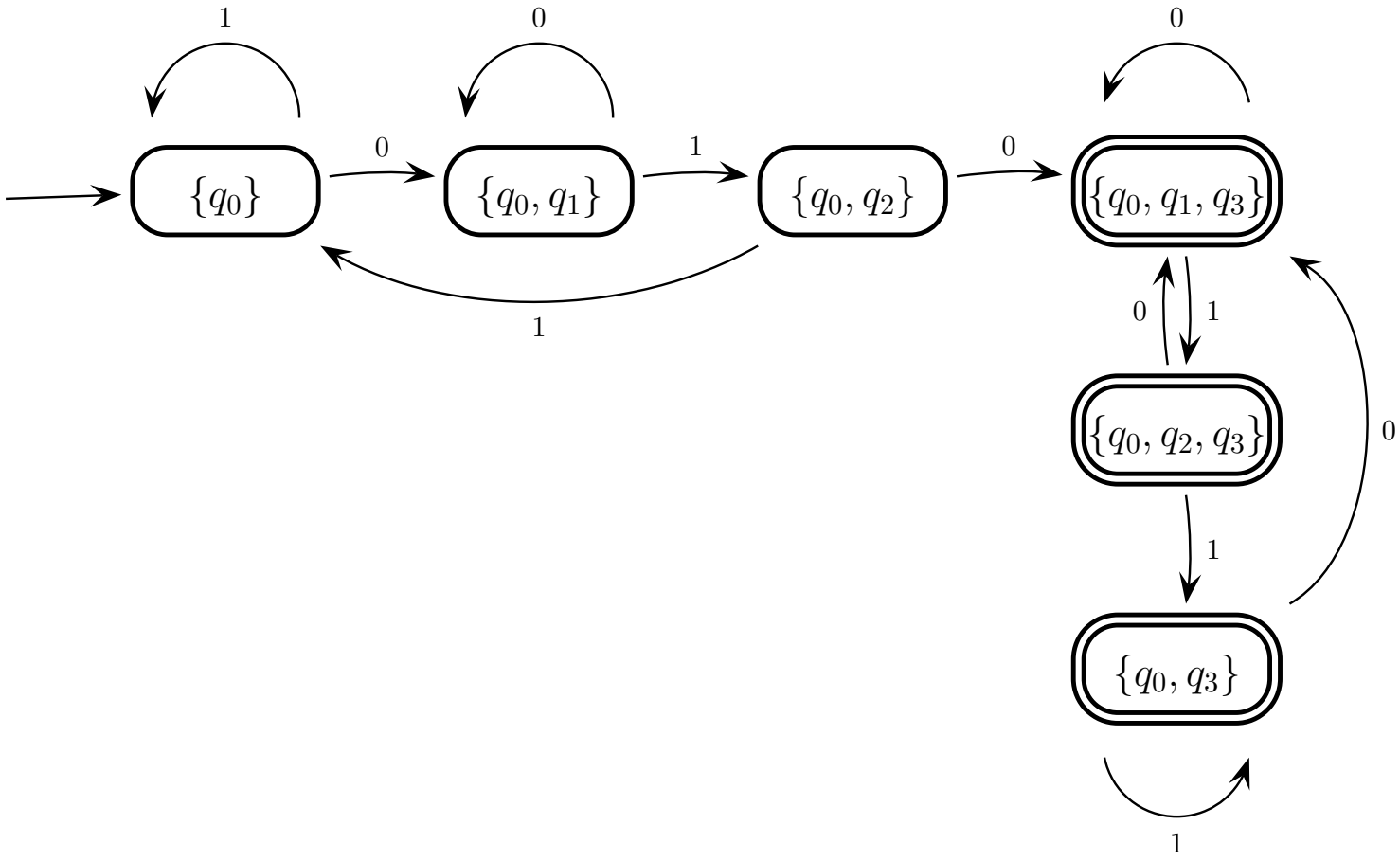
$\delta(\{q_0, q_2, q_3\}, 1) = \{q_0, q_3\}$, also ist $\{q_0, q_3\}$ erreichbar.

$\delta(\{q_0, q_3\}, 0) = \{q_0, q_1, q_3\}$

$\delta(\{q_0, q_3\}, 1) = \{q_0, q_3\}$

Wenn wir uns also auf die erreichbaren Zustände beschränken, so erhalten wir einen DEA, der nur 6 statt 16 Zustände hat. Drei davon sind Endzustände, nämlich diejenigen, die den Endzustand q_3 von A enthalten.

Graphisch: Der erreichbare Teil des Potenzautomaten



Wie bestimmt man die Menge der erreichbaren Zustände?

- Sei $A = (\Sigma, Q, s, F, \delta)$ ein DEA. Für jedes $n \geq 0$ sei

  $R_n = \{q \in Q \mid$ es ex. ein $w \in \Sigma^*$ mit $|w| \leq n$ und $(s, w) \vdash_A^* (q, \varepsilon)\}$

- Die Mengen R_n lassen sich durch Induktion über n definieren:
  - $R_0 = \{s\}$
  - $R_{n+1} = R_n \cup \{\delta(q, a) \mid q \in R_n, a \in \Sigma\}$
- Es gilt $R_0 \subseteq R_1 \subseteq R_2 \subseteq \ldots$ und die Menge R *aller* erreichbaren Zustände erhält man durch

  $R = \bigcup_{n \geq 0} R_n$

- Da alle R_n in der endlichen Menge Q enthalten sind, muss ein n existieren mit $R_n = R_{n+1}$.
- Daraus folgt $R_n = R_m$ für alle $m \geq n$, also $R = R_n$.

Algorithmus

- Berechne die Mengen $R_0, R_1, R_2, \ldots$ mit der induktiven Definition
- Sobald $R_n = R_{n+1}$ gilt, ist R_n die Menge der erreichbaren Zustände.

Man beachte:

- Nach den vorangegangen Überlegungen terminiert der Algorithmus und liefert das korrekte Ergebnis.
- Der gleiche Algorithmus funktioniert auch für NDEAs und auch für einen beliebigen Zustand p anstelle des Startzustands s.
- Er funktioniert sogar, wenn man noch nicht alle Übergänge kennt, denn im Induktionsschritt braucht man nur die Übergänge, die von den bereits gefundenen Zuständen ausgehen.
- Eigentlich handelt es sich um einen Graphalgorithmus (vgl. Vorlesung Algorithmen): Da die Markierung der Pfeile im Algorithmus keine Rolle spielt, kann der Automat als gerichteter Graph gesehen werden.

Gibt es noch weitere überflüssige Zustände?

- Ein Zustand heißt *tot*, wenn man von ihm aus keinen Endzustand mehr erreichen kann.
- Beispiele:

  der Zustand q' im Automaten A_{int}, (der erreicht wird, wenn '$-$' an der falschen Stelle auftritt)

  der Zustand $\emptyset$ in einem Potenzautomaten.
- Auch tote Zustände kann man aus einem Automaten entfernen, ohne dass sich die erkannte Sprache verändert.
- **Aber:** Dabei kann aus der totalen Funktion δ eine partielle Funktion werden, d.h. der entstehende Automat muss kein DEA mehr sein.

Anmerkung zur Literatur:

- Manche Autoren lassen partielle Funktionen in DEAs zu.

  Das entspricht besser dem intuitiven Begriff "deterministisch"

  und man kann tote Zustände weglassen, ohne die Definition von DEAs zu verletzen.

- Aber partielle Funktionen bereiten technische Probleme bei der Schreibweise: Man kann sie nicht auf jedes Element anwenden.

  Deshalb lassen *wir* sie *nicht* in DEAs, sondern nur (als spezielle Relationen) in NDEAs zu.

  Das ist kein großer Nachteil, denn man kommt stets mit *einem* toten Zustand aus.

Sprachklassen

Bei vorgegebenem Alphabet Σ sei

$\mathcal{L}_{reg} = \{L \subseteq \Sigma^* \mid L \text{ regulär.}\}$

$\mathcal{L}_{DEA} = \{L \subseteq \Sigma^* \mid \text{es existiert ein DEA } A \text{ mit } L = L(A)\}$

$\mathcal{L}_{NDEA} = \{L \subseteq \Sigma^* \mid \text{es existiert ein NDEA } A \text{ mit } L = L(A)\}$

Solche Mengen von Sprachen bezeichnet man als *Sprachklassen*,

Konvention: $\mathcal{L}$ für Sprachklassen.

Wir wollen zeigen:

$\mathcal{L}_{DEA} = \mathcal{L}_{NDEA} = \mathcal{L}_{reg}$

Die erste Gleichheit haben wir schon.

Satz 2.16 $\mathcal{L}_{DEA} = \mathcal{L}_{NDEA}$

Beweis:

'$\subseteq$':

Klar, da jeder DEA ein NDEA ist.

Genauer: Jeden DEA $A = (\Sigma, Q, s, F, \delta)$ kann man mit dem NDEA $A' = (\Sigma, Q, S, F, \Delta)$ mit $S = \{s\}$ und $\Delta = \{(p, a, q) \mid \delta(p, a) = q\}$ identifizieren. Dazu macht man sich klar, dass $\vdash_A$ (Definition 2.3) und $\vdash_{A'}$ (Definition 2.11) übereinstimmen, woraus dann insbesondere $L(A) = L(A')$ folgt.

'$\supseteq$':

Sei $L = L(A)$ für einen NDEA A. Dann gilt $L = L(A')$ für den Potenzautomaten A' von A, und A' ist ein DEA. $\square$

Endliche Automaten

Um zu zeigen, dass auch $\mathcal{L}_{reg}$ die gleiche Sprachklasse ist, betrachten wir zunächst eine weitere Verallgemeinerung unseres Automatenbegriffs. Wir lassen Pfeile zu, die mit dem leeren Wort ε markiert sind.

Definition 2.17 *Ein ε-NDEA ist ein 5-Tupel $A = (\Sigma, Q, S, F, \Delta)$ mit:*

- *Σ, Q, S und F sind wie beim NDEA definiert*
- *für die Übergangsrelation Δ gilt $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$*

Intuition:

- Ein ε-NDEA kann *spontane* Zustandsübergänge durchführen, ohne ein Zeichen zu lesen.
- Durch diese weitere Freiheit kann es einfacher sein, einen endlichen Automaten für eine gegebene Sprache zu finden, z.B. indem man dem Automaten ermöglicht, ein 'optionales' Zeichen oder Teilwort mit ε zu überspringen (Sprachen L_{int}, L_{float}).

Arbeitsweise eines ε-NDEA:

- Nur die *Übergangsschrittrelation* $\vdash_A$ muss neu definiert werden durch

$$(q, w) \vdash_A (q', w') \;\Leftrightarrow\; \text{es existiert ein } a \in \Sigma \cup \{\varepsilon\} \text{ mit } w = aw' \text{ und } (q, a, q') \in \Delta$$

- Alles andere wie beim NDEA.
- Die neue Sprachklasse bezeichnen wir mit $\mathcal{L}_{\varepsilon\text{-}NDEA}$.

Übliche Frage:

Kann ein ε-NDEA prinzipiell mehr leisten als ein NDEA?

D.h. gibt es eine Sprache in $\mathcal{L}_{\varepsilon\text{-}NDEA} \setminus \mathcal{L}_{NDEA}$?

Satz 2.18 *Zu jedem ε-NDEA lässt sich ein NDEA konstruieren, der die gleiche Sprache akzeptiert.*

Beweis:

Sei $A = (\Sigma, Q, S, F, \Delta)$ ein ε-NDEA.

Wie kann man die ε-Übergänge aus A entfernen, ohne die erkannte Sprache zu verändern?

Dazu definiert man zunächst den *ε-Abschluss* $E(p)$ eines Zustands $p \in Q$ durch:

$$E(p) = \{q \in Q \mid (p, \varepsilon) \vdash_A^* (q, \varepsilon)\}$$

$E(p)$ enthält also genau die Zustände, die man von p aus mit einer Folge von ε-Schritten erreichen kann.

Insbesondere ist stets $p \in E(p)$, weil auch eine Folge von 0 Schritten zugelassen ist.

Sei nun $A' = (\Sigma, Q, S, F', \Delta')$ mit

- $\Delta' = \{(p, a, q) \in Q \times \Sigma \times Q \mid \text{es ex. ein } p' \in E(p) \text{ mit } (p', a, q) \in \Delta\}$

  d.h. in A' nimmt man genau dann einen a-Übergang von p nach q auf, wenn es in A eine Folge von ε-Übergängen von p zu einem Zustand p', und von p' einen a-Übergang nach q gibt.

- $F' = \{p \in Q \mid E(p) \cap F \neq \emptyset\}$

  d.h. die Endzustände von A' sind genau die Zustände von A, die mit einer Folge von ε-Schritten in einen Endzustand übergehen können.

- A' ist ein NDEA, denn per Definition ist $\Delta' \subseteq Q \times \Sigma \times Q$, d.h. A' enthält *keine* ε-Übergänge mehr.

- Andererseits gilt $\Delta \cap (Q \times \Sigma \times Q) \subseteq \Delta'$, denn wegen $p \in E(p)$ kann man in der Definition von Δ' auch $p' = p$ wählen.

  Das bedeutet, dass alle Pfeile von A, die *nicht* mit ε markiert sind, in A' erhalten bleiben.

Behauptung: $L(A) = L(A')$

Beweis:

'$\subseteq$':

Sei $w = a_1 \dots a_n \in L(A)$, d.h. es existieren Zustände $s \in S$, $f \in F$ mit $(s, w) \vdash_A^* (f, \varepsilon)$. Diese Folge von Übergangsschritten lässt sich so aufteilen

$$\begin{aligned}
(s, a_1 \dots a_n) &\vdash_A^* (q_0, a_1 \dots a_n) \vdash_A (q_1, a_2 \dots a_n) \\
&\vdots \\
&\vdash_A^* (q_{n-1}, a_n) \quad \vdash_A (q_n, \varepsilon) \\
&\vdash_A^* (f, \varepsilon)
\end{aligned}$$

wobei die $\vdash_A^*$ nur aus ε-Schritten bestehen, d.h. $q_0 \in E(s)$, $q_i \in E(q_{i-1})$ für $i = 1, \dots, n-1$ und $f \in E(q_n)$.

Per Definition von Δ' gilt dann

$$(s, a_1 \dots a_n) \vdash_{A'} (q_1, a_2 \dots a_n) \vdash_{A'} \dots \vdash_{A'} (q_n, \varepsilon)$$

und per Definition von F' ist $q_n \in F'$, also ist $w = a_1 \dots a_n \in L(A')$.

'$\supseteq$':

Sei $w \in L(A')$, d.h. $(s, w) \vdash_{A'}^* (f', \varepsilon)$ für ein $f' \in F'$.

Da jeder Übergangsschritt in A' einer Folge von Übergangsschritten in A entspricht, gilt dann auch $(s, w) \vdash_A^* (f', \varepsilon)$ und per Definition von F' gilt $(f', \varepsilon) \vdash_A^* (f, \varepsilon)$ für ein $f \in F$.

Damit ist $(s, w) \vdash_A^* (f, \varepsilon)$ bewiesen, also $w \in L(A)$.

Noch zu zeigen:

Der NDEA A' lässt sich tatsächlich aus A *konstruieren*.

Dazu muss man die Menge $E(p)$ für jeden Zustand $p \in Q$ berechnen. Das geht analog zur Menge R der erreichbaren Zustände.

Für jedes $n \geq 0$ sei

$$E_n(p) = \{q \in Q \mid \exists\, m \leq n.\, (p, \varepsilon) \vdash_A^m (q, \varepsilon)\}$$

die Menge aller Zustände, die von p aus mit höchstens n ε-Schritten erreichbar sind. Auch diese Mengen lassen sich durch Induktion über n berechnen:

$E_0(p) = \{p\}$

$E_{n+1}(p) = E_n(p) \cup \{q \in Q \mid \text{es ex. ein } p' \in E_n(p) \text{ mit } (p', \varepsilon, q) \in \Delta\}$

und bilden eine aufsteigende Folge $E_0(p) \subseteq E_1(p) \subseteq \ldots$

Mit der gleichen Argumentation wie bei den Mengen R_n folgt: Es existiert ein n mit $E_n(p) = E_{n+1}(p)$, und für diese Zahl n ist dann $E(p) = E_n(p)$.

Mit Hilfe der Mengen $E(p)$ lassen sich dann leicht die Übergangsrelation Δ' und die Endzustandsmenge F' berechnen, womit die Konstruktion des NDEA A' abgeschlossen ist. □

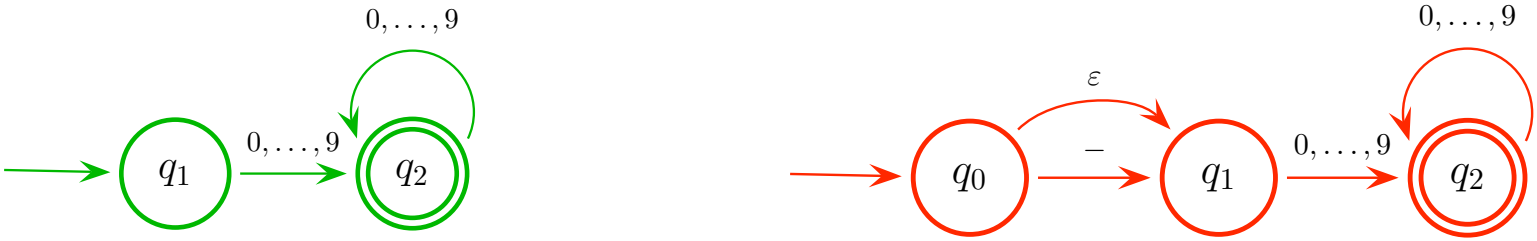
Als unmittelbare Folgerung von Satz 2.18 erhalten wir

Satz 2.19

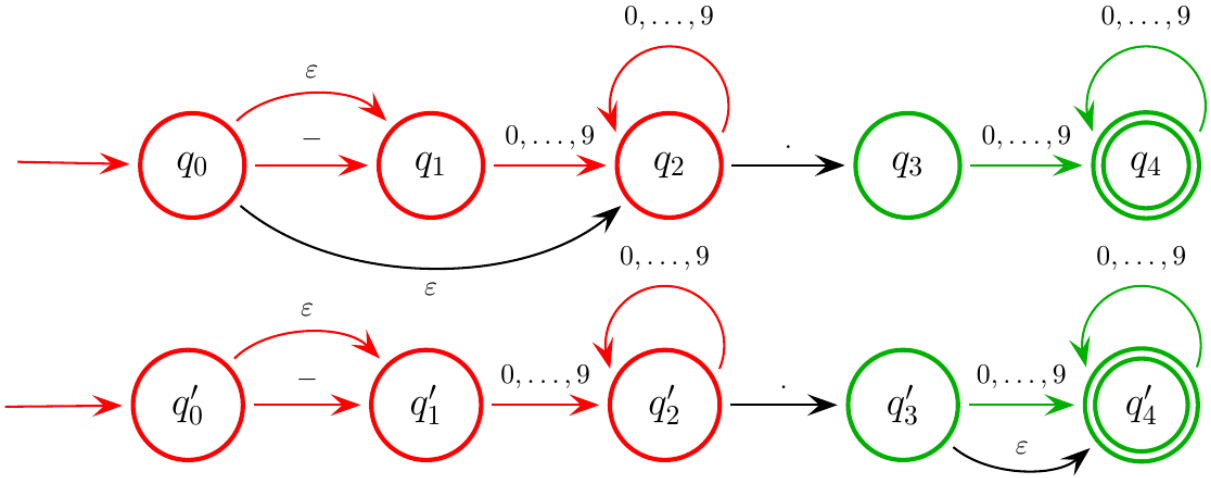
$$\mathcal{L}_{\varepsilon\text{-}NDEA} = \mathcal{L}_{NDEA}$$

Beispiel:

ε-NDEAs für $L_{nat} = \{0,\ldots,9\}^+$ und $L_{int} = \{-,\varepsilon\}L_{nat}$



und für $L_{fix} = (L_{int} \cup \{\varepsilon\})\{.\}L_{nat} \cup L_{int}\{.\}(L_{nat} \cup \{\varepsilon\})$



Umwandlung in einen NDEA:

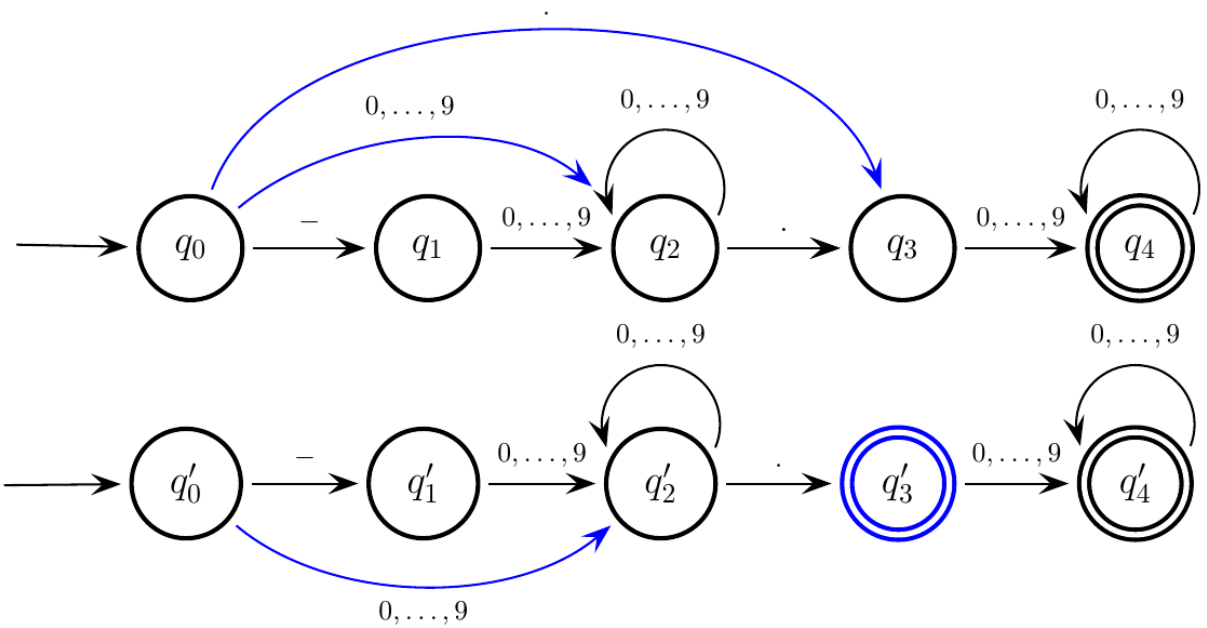
Als *neue Übergänge* erhält man

(q_0, z, q_2) für jede Ziffer z, da $q_1 \in E(q_0)$

$(q_0, .\,, q_3)$ da $q_2 \in E(q_0)$

(q_0', z, q_2') für jede Ziffer z, da $q_1' \in E(q_0')$

und q_3' ist *neuer Endzustand*, weil $q_4' \in E(q_3')$.



Am Beispiel war schon zu sehen, wie man einen ε-NDEA konstruieren kann, der eine gegebene reguläre Sprache erkennt.

Dieses Verfahren wird jetzt präzisiert. So erhalten wir den Beweis, dass $\mathcal{L}_{reg} \subseteq \mathcal{L}_{\varepsilon\text{-}NDEA}$.

Da wir schon wissen, dass

$$\mathcal{L}_{DEA} = \mathcal{L}_{NDEA} = \mathcal{L}_{\varepsilon\text{-}NDEA}$$

gilt, bezeichnen wir diese Sprachklasse jetzt mit

$$\mathcal{L}_{EA}$$

und benutzen den Begriff *endlicher Automat* (kurz: *EA*) als Sammelbegriff (d.h. als Synonym für ε-NDEA).

Satz 2.20 *Für die Sprachklasse $\mathcal{L}_{EA}$ (über dem Alphabet Σ) gilt:*

1. *$\mathcal{L}_{EA}$ enthält die Sprachen $\emptyset$, $\{\varepsilon\}$ und $\{a\}$ für alle $a \in \Sigma$.*
2. *$\mathcal{L}_{EA}$ ist abgeschlossen unter den Operationen $\cup$, $\circ$, $^+$ und *, d.h. wenn die Sprachen L_1, L_2 von endlichen Automaten erkannt werden, dann werden auch $L_1 \cup L_2$, $L_1 \circ L_2$, L_1^+ und L_1^* von endlichen Automaten erkannt.*
3. *Darüber hinaus gibt es für jede der Operationen in 2. einen Algorithmus, mit dem man den neuen Automaten konstruieren kann, also z.B. einen Algorithmus, der zu zwei endlichen Automaten A_1 und A_2 einen endlichen Automaten A mit $L(A) = L(A_1) \cup L(A_2)$ konstruiert.*

Beweis:

1. NDEAs, die $\emptyset$, $\{\varepsilon\}$ und $\{a\}$ erkennen:

2. Abgeschlossenheit unter $\cup$:

   Seien $A_1 = (\Sigma, Q_1, S_1, F_1, \Delta_1)$ und $A_2 = (\Sigma, Q_2, S_2, F_2, \Delta_2)$ endliche Automaten. Wir dürfen annehmen, dass $Q_1 \cap Q_2 = \emptyset$ (denn dies kann man durch Umbenennung der Zustände erreichen).

   Sei $A = (\Sigma, Q, S, F, \Delta)$ mit $Q = Q_1 \cup Q_2$, $S = S_1 \cup S_2$, $F = F_1 \cup F_2$ und $\Delta = \Delta_1 \cup \Delta_2$. Dann ist $L(A) = L(A_1) \cup L(A_2)$, denn offensichtlich gilt für jedes Wort $w \in \Sigma^*$:

   Jeder akzeptierende Lauf von A_1 oder A_2 für w ist auch ein akzeptierender Lauf von A für w.

   Jeder akzeptierende Lauf von A für w ist entweder ein akzeptierender Lauf von A_1 oder von A_2 für w.

Abgeschlossenheit unter $\circ$:

Seien $A_1 = (\Sigma, Q_1, S_1, F_1, \Delta_1)$ und $A_2 = (\Sigma, Q_2, S_2, F_2, \Delta_2)$ mit disjunkten Zustandsmengen Q_1, Q_2.

Einen Automaten A, der die Sprache $L(A_1) \circ L(A_2)$ erkennt, erhält man durch "Hintereinanderschalten" der Automaten A_1 und A_2, d.h.:

Man verbindet jeden Endzustand von A_1 mit jedem Startzustand von A_2 durch einen ε-Übergang. Als Startzustände von A wählt man die Startzustände von A_1 und als Endzustände von A wählt man die Endzustände von A_2.

Formal:

$A = (\Sigma, Q, S, F, \Delta)$ mit $Q = Q_1 \cup Q_2$, $S = S_1$, $F = F_2$ und $\Delta = \Delta_1 \cup \Delta_2 \cup \{(f, \varepsilon, s) \mid f \in F_1, s \in S_2\}$

Zu zeigen: $L(A) = L(A_1) \circ L(A_2)$

$\subseteq$: Sei $w \in L(A)$. Dann existieren Zustände $s \in S = S_1$ und $f \in F = F_2$ mit $(s, w) \vdash_A^* (f, \varepsilon)$. Da man nur durch die neuen ε-Übergänge von A_1 nach A_2 gelangen kann, muss es eine Zerlegung $w = uv$ und Zustände $f_1 \in F_1,\ s_2 \in S_2$ geben mit

$$(s, w) \ = \ (s, uv) \ \vdash_A^* \ (f_1, v) \ \vdash_A \ (s_2, v) \ \vdash_A^* \ (f, \varepsilon)$$

Der erste Teil dieser Folge kann nur aus A_1-Schritten bestehen und der letzte Teil nur aus A_2-Schritten. Also folgt $(s, u) \vdash_{A_1}^* (f_1, \varepsilon)$ mit Lemma 2.9 und $(s_2, v) \vdash_{A_2}^* (f, \varepsilon)$, d.h. $u \in L(A_1)$ und $v \in L(A_2)$. Damit ist $w = uv \in L(A_1) \circ L(A_2)$.

$\supseteq$: Sei $w \in L(A_1) \circ L(A_2)$, d.h. $w = uv$ mit $u \in L(A_1)$ und $v \in L(A_2)$. Dann gibt es Zustände $s_1 \in S_1 = S,\ f_1 \in F_1,\ s_2 \in S_2$ und $f_2 \in F_2 = F$ mit $(s_1, u) \vdash_{A_1}^* (f_1, \varepsilon)$ und $(s_2, v) \vdash_{A_2}^* (f_2, \varepsilon)$. Wegen $\Delta_1, \Delta_2 \subseteq \Delta$ folgt mit Lemma 2.9

$$(s_1, w) \ = \ (s_1, uv) \ \vdash_A^* \ (f_1, v) \ \vdash_A \ (s_2, v) \ \vdash_A^* \ (f_2, \varepsilon)$$

also $w \in L(A)$.

Abgeschlossenheit unter $^+$ und *.

Es genügt, $^+$ zu betrachten, denn $L^* = L^+ \cup \{\varepsilon\}$ und wir wissen schon, dass $\{\varepsilon\} \in \mathcal{L}_{EA}$ und dass $\mathcal{L}_{EA}$ unter $\cup$ abgeschlossen ist.

Sei $A_1 = (\Sigma, Q_1, S_1, F_1, \Delta_1)$.

Einen Automaten A, der $L(A_1)^+$ erkennt, erhält man wie folgt:

Man verbindet jeden Endzustand von A_1 mit jedem Startzustand von A_1. Als Startzustände von A wählt man die Startzustände von A_1 und als Endzustände von A wählt man die Endzustände von A_1.

Formal:

$A = (\Sigma, Q, S, F, \Delta)$ mit $Q = Q_1$, $S = S_1$, $F = F_1$ und
$\Delta = \Delta_1 \cup \{(f, \varepsilon, s) \mid f \in F_1, s \in S_1\}$

Zu zeigen: $L(A) = L(A_1)^+$

$\subseteq$: Sei $w \in L(A)$, d.h. es existieren $s \in S_1, f \in F_1$ mit $(s, w) \vdash_A^* (f, \varepsilon)$ Wenn wir diese Folge von Übergangsschritten bei den neuen ε-Übergängen aufteilen, so erhalten wir eine Zerlegung $w = w_1 \dots w_n$ $(n \geq 1)$ mit:

$$\begin{array}{rcll} (s, w) & = & (s_1, w_1 \dots w_n) & \vdash_A^* \ (f_1, w_2 \dots w_n) \\ & \vdash_A & (s_2, w_2 \dots w_n) & \vdash_A^* \ (f_2, w_3 \dots w_n) \\ & \vdots & & \\ & \vdash_A & (s_n, w_n) & \vdash_A^* \ (f_n, \varepsilon) \end{array}$$

wobei $s_1, \dots, s_n \in S_1$, $f_1, \dots, f_n \in F_1$, $s = s_1$, $f_n = f$ und alle $\vdash_A^*$ nur aus A_1-Schritten bestehen. Mit Lemma 2.9 folgt dann $(s_i, w_i) \vdash_{A_1}^* (f_i, \varepsilon)$, also $w_i \in L(A_1)$ für $i = 1, \dots, n$, und damit $w = w_1 \dots w_n \in L(A_1)^+$.

$\supseteq$: Sei $w \in L(A_1)^+$, d.h. $w = w_1 \dots w_n$ $(n \geq 1)$ mit $w_i \in L(A_1)$ für $i = 1, \dots, n$. Dann existieren Zustände $s_1, \dots, s_n \in S_1$ und $f_1, \dots, f_n \in F_1$ mit $(s_i, w_i) \vdash_{A_1}^* (f_i, \varepsilon)$, also folgt mit Lemma 2.9

$$\begin{aligned}
(s_1, w) &= (s_1, w_1 \dots w_n) \vdash_A^* (f_1, w_2 \dots w_n) \\
&\vdash_A (s_2, w_2 \dots w_n) \vdash_A^* (f_2, w_3 \dots w_n) \\
&\vdots \\
&\vdash_A (s_n, w_n) \vdash_A^* (f_n, \varepsilon)
\end{aligned}$$

und damit $w \in L(A)$. $\square$

Als Folgerung erhalten wir

Satz 2.21 $\mathcal{L}_{reg} \subseteq \mathcal{L}_{EA}$.

Beweis:

Eine Sprache ist regulär, wenn sie sich duch wiederholte Anwendung von $\cup$, $\circ$ und * aus endlichen Mengen aufbauen lässt.

Da $\mathcal{L}_{EA}$ abgeschlossen ist unter $\cup$, $\circ$ und *, bleibt nur noch zu zeigen, dass $\mathcal{L}_{EA}$ alle endlichen Teilmengen von Σ^* enthält.

Zunächst gilt dies für die einelementigen Mengen:

- $\{\varepsilon\} \in \mathcal{L}_{EA}$ nach Satz 2.20.
- Ist $w = a_1 \dots a_n \neq \varepsilon$, so ist $\{w\} = \{a_1\} \circ \dots \circ \{a_n\} \in \mathcal{L}_{EA}$ nach Satz 2.20.

Daraus folgt es für alle endlichen Mengen:

- $\emptyset \in \mathcal{L}_{EA}$ nach Satz 2.20.
- Ist $L = \{w_1, \dots, w_n\} \neq \emptyset$, so ist $L = \{w_1\} \cup \dots \cup \{w_n\} \in \mathcal{L}_{EA}$ nach Satz 2.20.

Die Sätze 2.20 und 2.21 (und ihre Beweise) sind wichtig für die Praxis (Compilerbau).

Denn sie liefern ein Verfahren, um aus der 'Mengenbeschreibung' einer regulären Sprache einen endlichen Automaten zu konstruieren.

Ein solches Verfahren wird in Scanner-Generatoren (z.B. Lex) verwendet, wobei man dort natürlich mehr auf die Effizienz der Algorithmen achten muss (vgl. Vorlesung Compilerbau) als wir es hier tun.

Als Eingabe verlangt ein Scanner-Generator sogenannte *reguläre Ausdrücke*, das sind formale Versionen unserer Mengenbeschreibungen, die wir später noch einführen werden.

Weitere Abschlusseigenschaften von $\mathcal{L}_{EA}$

Satz 2.22 *$\mathcal{L}_{EA}$ ist abgeschlossen unter Komplement, Durchschnitt, Mengendifferenz, Potenzierung und Spiegelung, d.h. wenn die Sprachen L_1 und L_2 von endlichen Automaten erkannt werden, dann werden auch $L_1 \cap L_2$, $\Sigma^* \setminus L_1$, $L_1 \setminus L_2$, $L_1^n (n \geq 0)$ und L_1^R von endlichen Automaten erkannt (und es gibt jeweils einen Algorithmus, mit dem man den neuen Automaten konstruieren kann).*

Beweis:

- Komplement:
  s. Übungsblatt 2, Aufgabe 4

- Durchschnitt:
  Wenn $L_1, L_2 \in \mathcal{L}_{EA}$, dann gilt $L_1 \cap L_2 = \Sigma^* \setminus ((\Sigma^* \setminus L_1) \cup (\Sigma^* \setminus L_2)) \in \mathcal{L}_{EA}$ wegen der Abgeschlossenheit unter Vereinigung und Komplement.

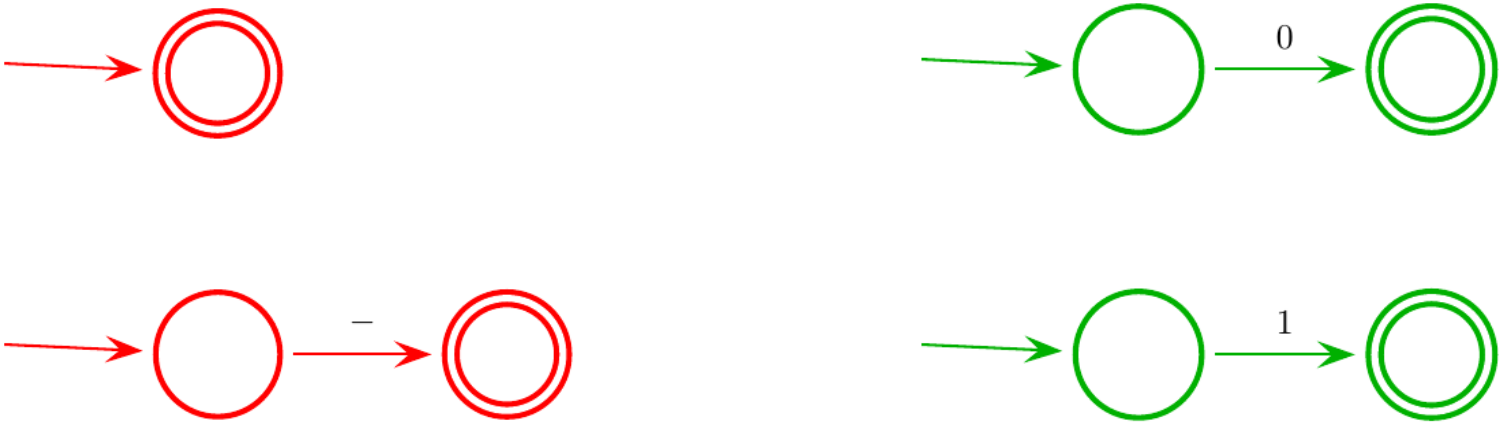
- Mengendifferenz:
  Wenn $L_1, L_2 \in \mathcal{L}_{EA}$, dann gilt $L_1 \setminus L_2 = L_1 \cap (\Sigma^* \setminus L_2) \in \mathcal{L}_{EA}$ wegen der Abgeschlossenheit unter Durchschnitt und Komplement.

- Potenzierung:
  Wenn $L_1 \in \mathcal{L}_{EA}$, dann gilt $L_1^n = L_1 \circ \ldots \circ L_1 \in \mathcal{L}_{EA}$ für alle $n \geq 1$ wegen der Abgeschlossenheit unter $\circ$, und $L_1^0 = \{\varepsilon\} \in \mathcal{L}_{EA}$ gilt sowieso.

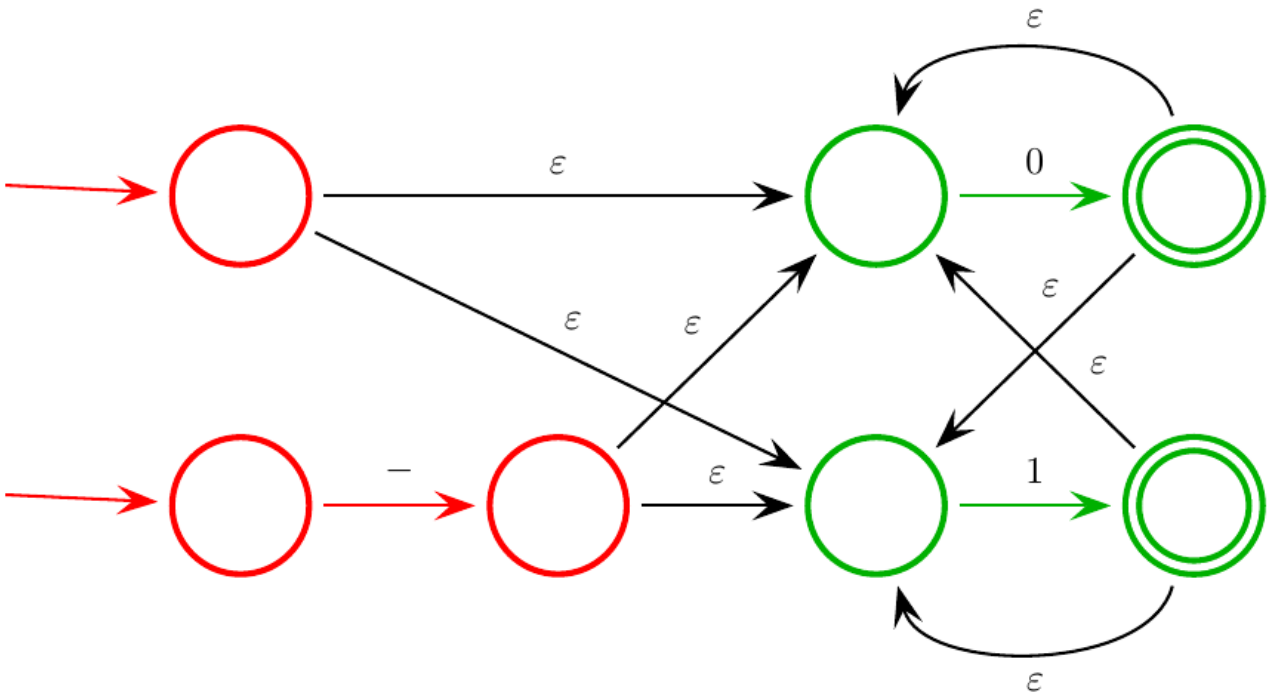
- Spiegelung:
  s. Übungsblatt 2, Aufgabe 4

Die Beweise der Sätze 2.20, 2.21 und 2.22 liefern uns Algorithmen zur Konstruktion von ε-NDEAs aus Mengenausdrücken.

Beispiel:
Mit den Konstruktionen für $\{\varepsilon\}$, $\{a\}$ und $\cup$ erhält man ε-NDEAs für die Sprachen $\{\varepsilon, -\}$ und $\{0, 1\}$



Daraus ergibt sich mit den Konstruktionen für $^+$ und $\circ$ ein ε-NDEA für die Sprache $L_{bin} = \{\varepsilon, -\} \circ \{0,1\}^+$ aller Binärdarstellungen ganzer Zahlen.



Für diesen könnte man jetzt wieder einen äquivalenten NDEA bzw. DEA konstruieren.

Zum Beweis von $\mathcal{L}_{reg} = \mathcal{L}_{EA}$ fehlt noch

Satz 2.23 $\mathcal{L}_{EA} \subseteq \mathcal{L}_{reg}$

Beweis:

Sei $A = (\Sigma, Q, s, F, \delta)$ ein DEA mit $Q = \{q_1, \ldots, q_n\}$ und $s = q_1$.

Es ist zu zeigen, dass $L(A)$ regulär ist.

Dazu betrachten wir—für alle $i, j \in \{1, \ldots, n\}$—die Sprachen

$$L_{ij} = \{w \in \Sigma^* \mid (q_i, w) \vdash_A^* (q_j, \varepsilon)\}$$

$L(A)$ lässt sich als (endliche) Vereinigung einiger dieser Sprachen L_{ij} darstellen, nämlich:

$$\begin{aligned} L(A) &= \{w \in \Sigma^* \mid \text{ es existiert ein } q_j \in F \text{ mit } (q_1, w) \vdash_A^* (q_j, \varepsilon)\} \\ &= \bigcup_{q_j \in F} L_{1j} \end{aligned}$$

Deshalb genügt es zu zeigen, dass die Sprachen L_{ij} regulär sind.

Dazu betrachten wir eine weitere 'Verfeinerung' der Sprachen L_{ij}.

Für jedes $k \in \{1, \ldots, n+1\}$ definieren wir

$$L_{ij}^k = \{w \in \Sigma^* \mid (q_i, w) \vdash_A^* (q_j, \varepsilon), \text{ wobei in } \vdash_A^* \text{ nur Zwischenzustände } q_l \text{ mit } l < k \text{ benutzt werden}\}$$

Dann gilt $L_{ij} = L_{ij}^{n+1}$ (weil $l < n+1$ keine Einschränkung ist),

und die Sprachen L_{ij}^k lassen sich durch Induktion über k definieren:

$$L_{ij}^1 = \{w \in \Sigma^* \mid (q_i, w) \vdash_A^* (q_j, \varepsilon) \quad \textit{ohne} \quad \text{Zwischenzustände}\}$$

$$= \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \text{ falls } i \neq j \\ \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \cup \{\varepsilon\} \text{ falls } i = j \end{cases}$$

$$L_{ij}^{k+1} = L_{ij}^{k} \cup L_{ik}^{k} \circ (L_{kk}^{k})^* \circ L_{kj}^{k}$$

Diese Gleichung erhält man durch folgende Überlegung:

Wenn $w \in L_{ij}^{k+1}$, dann gibt es zwei Möglichkeiten:

Entweder q_k taucht gar nicht als Zwischenzustand in der Folge $(q_i, w) \vdash_A^* (q_j, \varepsilon)$ auf.

Dann gilt schon $w \in L_{ij}^{k}$.

Oder wir können die Folge $(q_i, w) \vdash_A^* (q_j, \varepsilon)$ überall dort unterteilen, wo q_k auftaucht.

Dann erhalten wir eine Zerlegung $w = w_1 \dots w_m$ $(m \geq 2)$ mit

$$(q_i, w_1 \dots w_m) \vdash_A^* (q_k, w_2 \dots w_m) \vdash_A^* \dots \vdash_A^* (q_k, w_m) \vdash_A^* (q_j, \varepsilon)$$

in der alle $\vdash_A^*$ nur noch Zwischenzustände q_l mit $l < k$ enthalten.

Mit Lemma 2.9 folgt dann

$$\begin{array}{rcl} (q_i, w_1) & \vdash_A^* & (q_k, \varepsilon) \\ (q_k, w_l) & \vdash_A^* & (q_k, \varepsilon) \text{ für } l = 2, \ldots, m-1 \\ (q_k, w_m) & \vdash_A^* & (q_j, \varepsilon) \end{array}$$

wobei die $\vdash_A^*$ immer noch die gleichen Zwischenzustände enthalten, also nur Zustände q_l mit $l < k$.

Also gilt $w_1 \in L_{ik}^k$, $w_2, \ldots, w_{m-1} \in L_{kk}^k$ und $w_m \in L_{kj}^k$

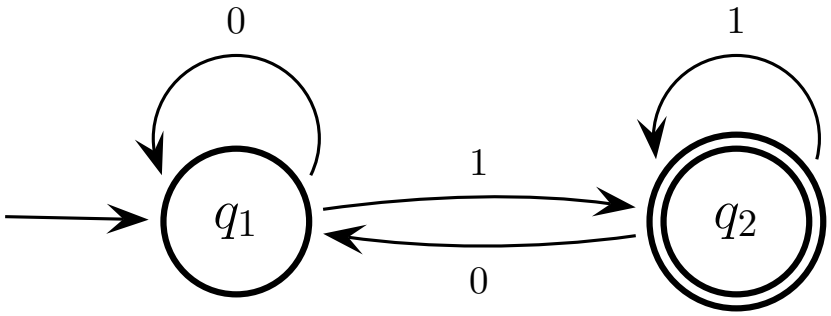
und damit $w = w_1 \ldots w_m \in L_{ik}^k \circ (L_{kk}^k)^* \circ L_{kj}^k$.

Damit ist '$\subseteq$' bewiesen und '$\supseteq$' ergibt sich ähnlich (einfacher!), indem man Folgen von Übergangsschritten *zusammensetzt*.

Da alle L_{ij}^1 endlich sind, und jedes L_{ij}^{k+1} durch Anwendung der Operationen $\cup$, $\circ$ und * aus einigen $L_{i'j'}^k$ entsteht, folgt (durch Induktion über k), dass alle L_{ij}^k regulär sind. $\square$

Auch der Beweis von Satz 2.23 ist konstruktiv. Er zeigt uns, wie wir einen 'Mengenausdruck' für die von einem DEA erkannte Sprache finden können.

Beispiel: Sei A der DEA



Wir bestimmen einen Mengenausdruck für $L(A)$, wobei wir (schon bei den Zwischenrechnungen) Vereinfachungen durchführen, damit der Gesamtausdruck nicht zu groß wird.

$$\begin{aligned} L(A) &= L_{12} \quad \text{da } q_2 \text{ einziger Endzustand ist.} \\ &= L_{12}^3 \quad \text{da } A \text{ zwei Zustände hat.} \end{aligned}$$

$$
\begin{aligned}
L_{12}^3 &= L_{12}^2 \cup L_{12}^2 \circ (L_{22}^2)^* \circ L_{22}^2 && \text{laut Gleichung für } L_{ij}^{k+1} \\
&= L_{12}^2 \cup L_{12}^2 \circ (L_{22}^2)^+ && \text{da } L^* \circ L = L^+ \\
&= L_{12}^2 \circ \{\varepsilon\} \cup L_{12}^2 \circ (L_{22}^2)^+ && \text{da } \{\varepsilon\} \text{ neutrales Element für } \circ \text{ ist} \\
&= L_{12}^2 \circ (\{\varepsilon\} \cup (L_{22}^2)^+) && \text{da } \circ \text{ distributiv über } \cup \text{ ist} \\
&= L_{12}^2 \circ (L_{22}^2)^* && \text{da } \{\varepsilon\} \cup L^+ = L^*
\end{aligned}
$$

$$
\begin{aligned}
L_{12}^2 &= L_{12}^1 \cup L_{11}^1 \circ (L_{11}^1)^* \circ L_{12}^1 && \text{laut Gleichung für } L_{ij}^{k+1} \\
&= \{1\} \cup \{\varepsilon, 0\} \circ \{\varepsilon, 0\}^* \circ \{1\} && \text{laut Gleichung für } L_{ij}^1 \\
&= \{1\} \cup \{\varepsilon, 0\}^+ \circ \{1\} && \text{da } L \circ L^* = L^+ \\
&= \{1\} \cup \{0\}^* \circ \{1\} && \text{da } (L \cup \{\varepsilon\})^+ = L^* \\
&= \{0\}^* \circ \{1\} && \text{da } \{1\} \subseteq \{0\}^* \circ \{1\}
\end{aligned}
$$

$$
\begin{aligned}
L_{22}^2 &= L_{22}^1 \cup L_{21}^1 \circ (L_{11}^1)^* \circ L_{12}^1 && \text{laut Gleichung für } L_{ij}^{k+1} \\
&= \{\varepsilon, 1\} \cup \{0\} \circ \{\varepsilon, 0\}^* \circ \{1\} && \text{laut Gleichung für } L_{ij}^1 \\
&= \{\varepsilon, 1\} \cup \{0\} \circ \{0\}^* \circ \{1\} && \text{da } (L \cup \{\varepsilon\})^* = L^* \\
&= \{\varepsilon, 1\} \cup \{0\}^+ \circ \{1\} && \text{da } L \circ L^* = L^+
\end{aligned}
$$

Also gilt $L(A) = \{0\}^* \circ \{1\} \circ (\{\varepsilon, 1\} \cup \{0\}^+ \circ \{1\})^*$

Natürlich ließe sich dieser Ausdruck noch weiter vereinfachen, letztendlich zu $\{0, 1\}^* \circ \{1\}$, weil das die von A erkannte Sprache ist. In seiner jetzigen Form ist er aber aufschlussreicher, weil er noch die Konstruktionsidee aus dem Beweis von Satz 2.23 erkennen lässt:

Die Menge $\{0\}^* \circ \{1\}$ enthält alle Wörter, die von q_1 nach q_2 führen, wobei nur q_1 als Zwischenzustand benutzt wird. Und die Menge $\{\varepsilon, 1\} \cup \{0\}^+ \circ \{1\}$ enthält alle Wörter, die von q_2 nach q_2 führen, wobei auch wieder nur q_1 als Zwischenzustand benutzt wird.

Reguläre Sprachen

Mit den Sätzen 2.21 und 2.23 ist bewiesen, dass

$$\mathcal{L}_{reg} = \mathcal{L}_{EA}$$

und deshalb sprechen wir ab jetzt nur noch von *regulären Sprachen*.

Mit anderen Worten:

Wir haben bisher nur *eine* Sprachklasse kennengelernt, nämlich die Klasse der regulären Sprachen, aber wir haben unterschiedliche Formalismen zur Darstellung regulärer Sprachen:

- DEAs
- NDEAs
- ε-NDEAs
- und 'Mengenausdrücke'

Problem: Für 'Mengenausdrücke' haben wir keine formale (sondern nur die übliche mathematische) Schreibweise.

Deshalb: Reguläre Ausdrücke

Definition 2.24 *Sei Σ ein Alphabet. Die Menge aller reguläre Ausdrücke über Σ ist induktiv definiert durch:*

1. *$\emptyset$ ist ein regulärer Ausdruck über Σ.*
2. *Jedes $a \in \Sigma$ ist ein regulärer Ausdruck über Σ.*
3. *Wenn α und β reguläre Ausdrücke über Σ sind, dann sind auch $(\alpha\beta)$, $(\alpha \mid \beta)$ und $(\alpha^\star)$ reguläre Ausdrücke über Σ.*

Ein regulärer Ausdruck über Σ ist also ein Wort über dem Alphabet $\Sigma' = \Sigma \cup \{\emptyset, (,), |, \star\}$, das nach den Regeln 1. bis 3. aufgebaut ist.

Damit ist die *Syntax* der regulären Ausdrücke festgelegt.

Es fehlt noch die *Semantik*.

Reguläre Sprachen

Ein regulärer Ausdruck soll natürlich die Sprache beschreiben, die durch den 'entsprechenden Mengenausdruck' definiert ist.

Definition 2.25 *Die von einem regulären Ausdruck α beschriebene Sprache $L(\alpha)$ ist durch Induktion über die Größe von α wie folgt definiert:*

1. $L(\varnothing) = \emptyset$
2. $L(a) = \{a\}$
3. $L((\alpha\beta)) = L(\alpha) \circ L(\beta)$

   $L((\alpha \mid \beta)) = L(\alpha) \cup L(\beta)$

   $L((\alpha^\star)) = (L(\alpha))^*$

Man beachte:
Links stehen *Zeichen* ø, | und *.
Rechts stehen mathematische Schreibweisen: $\emptyset$, $\{\ \}$, $\cup$, $\circ$ und *.

Vereinbarung zur Einsparung von Klammern

- Die Konkatenation bindet stärker als '|'.
- '$\star$' bindet am stärksten.
- Konkatenation und '|' sind linksassoziativ, d.h. $\alpha\beta\gamma$ steht für $(\alpha\beta)\gamma$ und $\alpha \mid \beta \mid \gamma$ für $(\alpha \mid \beta) \mid \gamma$.

  (Ebenso gut könnte man Rechtsassoziativität vereinbaren, da die Mengenoperationen $\cup$ und $\circ$ sowieso assoziativ sind.)

Beispiele

- $L_{nat} = \{0, \ldots, 9\}^+ = L((0 \mid \ldots \mid 9)\,(0 \mid \ldots \mid 9)^\star)$
- $L_{int} = \{\varepsilon, -\} \circ L_{nat} = L((\emptyset^\star \mid -)(0 \mid \ldots \mid 9)\,(0 \mid \ldots \mid 9)^\star)$
- $L(a^\star\, b^\star) = \{a\}^* \circ \{b\}^* = \{a^m\, b^n \mid m, n \geq 0\}$
- $L((ab)^\star) = (\{a\} \circ \{b\})^* = \{ab\}^* = \{(ab)^n \mid n \geq 0\}$

Per Definition der Semantik regulärer Ausdrücke ist es klar, dass die von einem regulären Ausdruck beschriebene Sprache stets regulär ist. Umgekehrt lässt sich jede reguläre Sprache durch einen regulären Ausdruck beschreiben, denn:

- Alle endlichen Sprachen erhält man mit Konkatenation und Vereinigung aus $\emptyset$, $\{\varepsilon\}$ und den Mengen $\{a\}$ mit $a \in \Sigma$ (vgl. Beweis zu Satz 2.23).
- Diese Mengen kann man durch reguläre Ausdrücke beschreiben, nämlich $\emptyset = L(\varnothing)$, $\{\varepsilon\} = \emptyset^* = L(\varnothing^\star)$ und $\{a\} = L(a)$.

Also gilt:

Satz 2.26 *Eine Sprache ist genau dann regulär, wenn sie sich durch einen regulären Ausdruck beschreiben lässt.*

Reguläre Sprachen

Mit den regulären Ausdrücken haben wir eine *formale Syntax* zur Beschreibung regulärer Sprachen (anstelle der "üblichen mathematischen Schreibweise" bei Mengenausdrücken).

Natürlich übertragen sich die bisherigen Beobachtungen über Mengenausdrücke unmittelbar auf reguläre Ausdrücke, insbesondere:

Die Konstruktion von ε-NDEAs aus dem Beweis zu Satz 2.20 (und Satz 2.21) liefert zu jedem regulären Ausdruck einen äquivalenten ε-NDEA.

Die Konstruktion der Sprachen L_{ij} aus dem Beweis von Satz 2.23 lässt sich so abwandeln, dass sie zu jedem DEA einen äquivalenten regulären Ausdruck liefert.

Damit haben wir *Übersetzungsalgorithmen*, um all die unterschiedlichen Darstellungen regulärer Sprachen (DEAs, NDEAs, ε-NDEAs und reguläre Ausdrücke) ineinander überzuführen.

Bezug zur Praxis

Reguläre Ausdrücke werden benutzt

- als Eingabe für Scanner-Generatoren (Compilerbau)
- als Suchmuster in Editoren und Suchmaschinen
- zur Textverarbeitung in Programmiersprachen

Meistens hat man eine erweiterte Syntax für reguläre Ausdrücke:

- zusätzliche Zeichen wie ϵ, $^+$ oder ein Zeichen für das Komplement
- Schreibweisen für endliche Zeichenmengen, z.B. [a-$z$$A$-$Z$]
- Einführung von Namen zur Wiederverwendung eines Ausdrucks, z.B $nat = [0\text{-}9]^+$, $int = (\epsilon \mid -)\, nat$

Intern werden reguläre Ausdrücke in EAs umgewandelt, im Prinzip nach unserer Theorie, aber mit effizienteren Algorithmen (z.B. direkt vom regulären Ausdruck zum DEA).

Reguläre Sprachen

Nachdem wir wissen, dass $\mathcal{L}_{reg} = \mathcal{L}_{EA}$ gilt, können wir jetzt all unsere Formalismen und auch die bewiesenen Abschlusseigenschaften kombinieren, um zu zeigen, dass eine Sprache regulär ist.

Beispiel:
Sei $L = \{w \in \{0,1\}^* \mid w \text{ enthält } 010, \text{ aber nicht } 110\}$.
Dann gilt $L = L((0 \mid 1)^* 010 (0 \mid 1)^*) \setminus L((0 \mid 1)^* 110 (0 \mid 1)^*)$,
also ist L regulär.

Aber auch, um zu zeigen, dass eine Sprache *nicht* regulär ist.

Beispiel:
Sei $L = \{w \in \{a,b\}^* \mid \#_a(w) = \#_b(w)\}$, wobei $\#_a(w)$ für die Anzahl der as im Wort w steht. Dann ist $L \cap L(a^\star b^\star) = \{a^n b^n \mid n \geq 0\}$, d.h. wenn L regulär wäre, dann wäre auch $\{a^n b^n \mid n \geq 0\}$ als Durchschnitt zweier regulärer Sprachen wieder regulär.

Aber von dieser Sprache wissen wir schon, dass sie nicht regulär ist.

Entscheidbarkeitsfragen

Neben den Übersetzungsalgorithmen interessieren uns auch *Entscheidungsalgorithmen*. Das sind Algorithmen, die gewisse Fragestellungen stets korrekt mit 'ja' oder 'nein' beantworten, z.B.

- die Frage, ob ein Automat die leere Sprache erkennt
- die Frage, ob zwei Automaten äquivalent sind.

In jedem Fall muss geklärt werden,

- welche Eingabedaten der Algorithmus erhält,
- welche Frage über die Eingabedaten er beantworten soll

  (also wie seine Ausgabe von der Eingabe abhängen soll)

Satz 2.27 *Für jede der folgenden Fragestellungen gibt es einen Entscheidungsalgorithmus.*

1. *Eingabe: Ein DEA A und ein Wort w. Frage: Ist $w \in L(A)$?*

   (das sogenannte Wortproblem für DEAs)

2. *Eingabe: Ein DEA A. Frage: Ist $L(A) = \emptyset$?.*

   (das Leerheitsproblem für DEAs)

3. *Eingabe: Ein DEA A. Frage: Ist $L(A) = \Sigma^*$?.*

4. *Eingabe: Zwei DEAs A_1 und A_2. Frage: Ist $L(A_1) \subseteq L(A_2)$?*

5. *Eingabe: Zwei DEAs A_1 und A_2. Frage: Ist $L(A_1) = L(A_2)$?*

   (das Äquivalenzproblem für DEAs)

Beweis:

1. Der Algorithmus muss nur den Ablauf des DEA A bei Eingabe w simulieren und dann überprüfen, ob der erreichte Zustand ein Endzustand ist.

2. $L(A) = \emptyset$ gilt genau dann, wenn *kein* Endzustand erreichbar ist. Also berechnet man die Menge der erreichbaren Zustände von A und überprüft, ob darunter ein Endzustand ist.

3. Es gilt $L(A) = \Sigma^* \Leftrightarrow \Sigma^* \setminus L(A) = \emptyset$. Also konstruiert man zu A einen DEA $\bar{A}$ mit $L(\bar{A}) = \Sigma^* \setminus L(A)$ (nach dem Verfahren aus der Übung) und überprüft, ob $L(\bar{A}) = \emptyset$.

4. $L(A_1) \subseteq L(A_2)$ gilt genau dann, wenn $L(A_1) \cap (\Sigma^* \setminus L(A_2)) = \emptyset$. Also konstruiert man zu A_1 und A_2 einen DEA A mit $L(A) = L(A_1) \cap (\Sigma^* \setminus L(A_2))$ und überprüft, ob $L(A) = \emptyset$.

5. Um $L(A_1) = L(A_2)$ zu testen, überprüft man, ob $L(A_1) \subseteq L(A_2)$ und $L(A_2) \subseteq L(A_1)$ gilt. □

Natürlich gilt Satz 2.27 auch für NDEAs, ε-NDEAs oder reguläre Ausdrücke anstelle von DEAs.

Denn wir können jeden dieser Formalismen in einen äquivalenten DEA übersetzen und brauchen dann nur noch die entsprechende Frage für den DEA zu beantworten

(weil es ja Fragen über die erkannte Sprache sind).

Grenzen regulärer Sprachen

Wie beweist man, dass eine Sprache *nicht* regulär ist?

Oder allgemeiner:

Wie findet man heraus, *ob* eine Sprache regulär ist oder nicht?

Wir hatten schon ein Beispiel—nämlich $L = \{a^n b^n \mid n \geq 0\}$—mit dem *Schubfachprinzip* bewiesen.

Jeder DEA A verteilt die Wörter $w \in \Sigma^*$ auf endlich viele 'Schubladen', nämlich auf seine Zustände.

Wenn wir also *unendlich viele* Wörter finden, von denen je zwei *nicht* in der gleichen Schublade stecken dürfen, so kann kein DEA für die Sprache existieren.

Bei der Sprache L waren das die Wörter a^n mit $n \geq 0$.

Die Annahme, dass zwei dieser Wörter, a^i und a^j, in ein- und demselben Zustand landen, führt zum Widerspruch,

denn dann würden auch a^ib^i und a^jb^i in einem gemeinsamen Zustand landen, obwohl $a^ib^i \in L$ und $a^jb^i \notin L$.

Definition 2.28 *Sei $L \subseteq \Sigma^*$. Zwei Wörter $u_1, u_2 \in \Sigma^*$ heißen L-unterscheidbar, wenn ein Wort $v \in \Sigma^*$ existiert mit $u_1v \in L$ und $u_2v \notin L$ oder umgekehrt (d.h. wenn man sie nicht in die gleiche Schublade stecken darf).*

Damit erhalten wir das folgende

Beweisprinzip: Um zu zeigen, dass eine Sprache L nicht regulär ist, genügt es, *unendlich viele* Wörter in Σ^* zu finden, die *paarweise L-unterscheidbar* sind.

Beispiele nicht regulärer Sprachen:

1. "Ein EA kann nicht zählen"

- $L = \{a^n b^n \mid n \geq 0\}$: Die Wörter a^n mit $n \geq 0$ sind paarweise L-unterscheidbar, da $a^i b^i \in L$ und $a^j b^i \notin L$ für alle $i \neq j$.
- $L_1 = \{w \in \{a,b\}^* \mid \#_a(w) = \#_b(w)\}$: Die Wörter a^n mit $n \geq 0$ sind paarweise L_1-unterscheidbar, da $a^i b^i \in L_1$ und $a^j b^i \notin L_1$ für alle $i \neq j$.
- $L_2 = \{a^n b^m \mid 0 \leq n < m\}$: Die Wörter a^n mit $n \geq 0$ sind paarweise L_2-unterscheidbar, denn: Wenn $i \neq j$, dann dürfen wir $i < j$ annehmen (weil L-Unterscheidbarkeit eine symmetrische Relation ist) und erhalten $a^i b^{i+1} \in L_2$, aber $a^j b^{i+1} \notin L_2$ weil $j \geq i+1$.
- $L_3 = \{a^n b^m \mid 0 \leq n < 2^m\}$: Die Wörter a^{2^n-1} mit $n \geq 0$ sind paarweise L_3-unterscheidbar, denn: Für alle $i < j$ gilt $a^{2^i-1} b^i \in L_2$ weil $2^i - 1 < 2^i$ und $a^{2^j-1} b^i \notin L_2$ weil $2^j - 1 \geq 2^i + 2^i - 1 \geq 2^i$.

2. "Ein EA kann sich keine beliebig großen Wörter merken"

- $L_4 = \{ww \mid w \in \{a,b\}^*\}$: Die Wörter $a^n b$ mit $n \geq 0$ sind paarweise L_4-unterscheidbar, denn: Für alle $i \neq j$ ist $a^i b a^i b \in L_4$ und $a^j b a^i b \notin L_4$. (Man kann auch die Wörter a^n betrachten, denn a^i und a^j lassen sich durch ba^ib unterscheiden.)
- $L_5 = \{w \in \{a,b\}^* \mid w = w^R\}$:

  s. Übung

3. "Eine reguläre Sprache kann keine beliebig großen Lücken enthalten"

- $L_5 = \{a^{n^2} \mid n \geq 0\}$:

  s. Übung

Reguläre Sprachen

- $L_6 = \{a^{2^n} \mid n \geq 0\}$: Die Wörter aus L_6 sind paarweise L_6-unterscheidbar, denn für alle $i < j$ gilt $a^{2^i}a^{2^i} = a^{2^i+2^i} = a^{2^{i+1}} \in L_6$ und $a^{2^j}a^{2^i} = a^{2^j+2^i} \notin L_6$, weil $2^j + 2^i < 2^j + 2^j = 2^{j+1}$ zwischen den beiden aufeinanderfolgenden Zweierpotenzen 2^j und 2^{j+1} liegt und deshalb selbst *keine* Zweierpotenz sein kann.

- $L_7 = \{a^p \mid p$ ist Primzahl$\}$:

  Wir wollen beweisen, dass zwei *beliebige* Wörter $a^i, a^j \in \{a\}^*$ L_7-unterscheidbar sind, wobei wir wieder $i < j$ annehmen dürfen.

  Dazu brauchen wir eine Zahl k mit $a^j a^k \in L_7$ und $a^i a^k \notin L_7$, d.h. $j + k$ ist Primzahl und $i + k$ nicht.

  Um ein solches k zu finden, genügt es zu wissen, dass die Menge der Primzahlen beliebig große Lücken enthält, d.h. für jedes $n > 0$ existieren n aufeinanderfolgende Zahlen, die keine Primzahlen sind.

Sei p die kleinste Primzahl, die über einer solchen Lücke der Größe j liegt.

Dann sind (mindestens) die Zahlen $p-j, \ldots, p-1$ *keine* Primzahlen.

Also können wir $k = p - j$ wählen, denn $j + k = p$ ist eine Primzahl und $i + k = i + p - j = p - (j - i)$ fällt in die Lücke und ist deshalb *keine* Primzahl.

Die Existenz beliebig großer Lücken in der Menge der Primzahlen lässt sich leicht einsehen:

Eine Lücke der Größe n bilden z.B. die Zahlen $(n+1)! + 2, \ldots, (n+1)! + n + 1$, denn:

$(n+1)!$ ist durch jede der Zahlen $2, \ldots, n+1$ teilbar, also ist $(n+1)! + m$ durch m teilbar für jedes $m \in \{2, \ldots, n+1\}$, und damit *keine* Primzahl.

Eine alternative Formulierung des Beweisprinzips

Definition 2.29 *Sei $L \subseteq \Sigma^*$. Zwei Wörter $u_1, u_2 \in \Sigma^*$ heißen L-äquivalent ($u_1 \sim_L u_2$), wenn sie nicht L-unterscheidbar sind, d.h. wenn für jedes $v \in \Sigma^*$ entweder $u_1v, u_2v \in L$ oder $u_1v, u_2v \notin L$.*

Wir schreiben $\not\sim_L$ für die Verneinung von $\sim_L$, d.h. $u_1 \not\sim_L u_2$ bedeutet, dass u_1 und u_2 L-unterscheidbar sind.

Für jede Sprache $L \subseteq \Sigma^*$ ist $\sim_L$ eine *Äquivalenzrelation* auf Σ^* (reflexiv, transitiv und symmetrisch). Das sieht man am besten, wenn man die Definition von $\sim_L$ etwas umformuliert: Für jedes $u \in \Sigma^*$ sei

$$Erg_L(u) = \{v \in \Sigma^* \mid uv \in L\}$$

die Menge der *L-Ergänzungen* von u. Dann gilt

$$\begin{aligned} u_1 \sim_L u_2 &\Leftrightarrow Erg_L(u_1) = Erg_L(u_2) \\ &\Leftrightarrow \text{für alle } v \in \Sigma^* gilt : v \in Erg_L(u_1) \Leftrightarrow v \in Erg_L(u_2) \\ &\Leftrightarrow \text{für alle } v \in \Sigma^* gilt : u_1v \in L \Leftrightarrow u_2v \in L \end{aligned}$$

Intuition:
Die L-Ergänzungen von u sind genau die Restwörter, die das Anfangswort u in die Sprache L überführen, also genau die Wörter, die man 'noch erwartet', wenn man bereits u gelesen hat.
Also sind zwei Wörter u_1, u_2 genau dann L-äquivalent, wenn man nach Einlesen von u_1 und u_2 die gleichen Restwörter erwartet, d.h. wenn man u_1 und u_2 in die gleiche 'Schublade' packen darf.

Schreibweise:
Mit $[u]_L$ bezeichnen wir die *L-Äquivalenzklasse* eines Wortes $u \in \Sigma^*$, d.h.

$$[u]_L \;=\; \{u' \in \Sigma^* \mid u' \sim_L u\}$$

Es gilt also

$$u_1 \sim_L u_2 \;\Leftrightarrow\; [u_1]_L = [u_2]_L$$
$$u_1 \not\sim_L u_2 \;\Leftrightarrow\; [u_1]_L \neq [u_2]_L$$

Damit können wir unser **Beweisprinzip** neu formulieren:

Um zu zeigen, dass eine Sprache L *nicht* regulär ist, genügt es, unendlich viele Äquivalenzklassen $[u]_L$ zu finden

(denn unendlich viele paarweise L-unterscheidbare Wörter sind nichts anderes als die Vertreter von unendlich vielen Äquivalenzklassen).

Wir wollen zeigen, dass auch die Umkehrung gilt, d.h. dass eine Sprache L regulär ist, wenn es nur endlich viele L-Äquivalenzklassen gibt.

Beides wird zusammengefasst im **Satz von Myhill und Nerode**, zu dessen Vorbereitung wir zunächst noch einige Eigenschaften der Relation $\sim_L$ beweisen.

Lemma 2.30

1. *Für alle* $u_1, u_2, v \in \Sigma^*$ *gilt: Wenn* $u_1 \sim_L u_2$, *dann* $u_1 v \sim_L u_2 v$.

   (Eine Äquivalenzrelation mit dieser Eigenschaft bezeichnet man als Rechtskongruenzrelation*.)*

2. *Für jedes* $u \in \Sigma^*$ *gilt:* $u \in L \Leftrightarrow [u]_L \subseteq L$. *(Also* $L = \bigcup_{u \in L} [u]_L$*.)*

Beweis:

1. Seien $u_1, u_2, v \in \Sigma^*$ und $u_1 \sim_L u_2$. Dann gilt für alle $v' \in \Sigma^*$:

   $(u_1 v)v' \in L \;\Leftrightarrow\; u_1(vv') \in L \;\Leftrightarrow\; u_2(vv') \in L \;\Leftrightarrow\; (u_2 v)v' \in L$

   Also ist $u_1 v \sim_L u_2 v$.

2. '$\Leftarrow$' ist klar.

   '$\Rightarrow$': Sei $u \in L$ und $u' \sim_L u$.

   Wegen $u\varepsilon \in L$ ist dann auch $u'\varepsilon \in L$, also $u' \in L$. □

Satz 2.31 (Satz von Myhill und Nerode) *Eine Sprache $L \subseteq \Sigma^*$ ist genau dann regulär, wenn die Äquivalenzrelation $\sim_L$ nur endlich viele Äquivalenzklassen besitzt.*

Beweis:

'$\Rightarrow$': Sei $L \subseteq \Sigma^*$ regulär.

Sei $A = (\Sigma, Q, s, F, \delta)$ ein DEA mit $L(A) = L$

und seien $u_1, u_2 \in \Sigma^*$ mit $\delta^*(s, u_1) = \delta^*(s, u_2)$.

Dann gilt für alle $v \in \Sigma^*$:

$\delta^*(s, u_1 v) = \delta^*(\delta^*(s, u_1), v) = \delta^*(\delta^*(s, u_2), v) = \delta^*(s, u_2 v)$,

also $u_1 v \in L \Leftrightarrow \delta^*(s, u_1 v) \in F \Leftrightarrow \delta^*(s, u_2 v) \in F \Leftrightarrow u_2 v \in L$.

Das bedeutet $u_1 \sim_L u_2$.

Damit haben wir bewiesen, dass zwei Wörter, die in A zum gleichen Zustand führen, stets L-äquivalent sind.

Also besitzt $\sim_L$ nur endlich viele Äquivalenzklassen, nämlich höchstens n, wobei n die Anzahl der erreichbaren Zustände von A ist.

'$\Leftarrow$': Sei $L \subseteq \Sigma^*$ eine Sprache, die nur endlich viele L-Äquivalenzklassen besitzt.

Da L-äquivalente Wörter in die gleiche 'Schublade' gesteckt werden können, definieren wir einen DEA, der für jede L-Äquivalenzklasse eine passende Schublade besitzt.

Deshalb nehmen wir die L-Äquivalenzklassen selbst als Schubladen, d.h. als Zustände eines DEA.

Sei $A = (\Sigma, Q, s, F, \delta)$ mit:

$Q = \{[u]_L \mid u \in \Sigma^*\}$ (die endliche Menge der L-Äquivalenzklassen)

$s = [\varepsilon]_L$

$F = \{[u]_L \mid u \in L\}$ $(= \{[u]_L \mid [u]_L \subseteq L\}$ nach Lemma 2.30)

$\delta : Q \times \Sigma \to Q$ $\delta([u]_L, a) = [ua]_L$

Man beachte, dass δ wohldefiniert ist, d.h. dass $[ua]_L$ unabhängig von der Wahl des speziellen Vertreters $u \in [u]_L$ ist.

Wenn nämlich $u' \sim_L u$ ein anderes Element aus $[u]_L$ ist, so folgt $u'a \sim_L ua$ und damit $[u'a]_L = [ua]_L$.

Um zu zeigen, dass A tatsächlich die Sprache L erkennt, beweisen wir, dass jedes Wort $w \in \Sigma^*$ in die passende Schublade gerät, d.h. dass

$$\delta^*(s, w) = [w]_L \quad \text{für alle } w \in \Sigma^* \qquad (*)$$

$w = \varepsilon$:

$$\begin{aligned} \delta^*(s,\varepsilon) &= s && \text{per Definition von } \delta^* \\ &= [\varepsilon]_L && \text{per Definition von } s \end{aligned}$$

$w = va$:

$$\begin{aligned} \delta^*(s,va) &= \delta(\delta^*(s,v),a) && \text{per Definition von } \delta^* \\ &= \delta([v]_L,a) && \text{nach Induktionsannahme für } v \\ &= [va]_L && \text{per Definition von } \delta \end{aligned}$$

Damit erhalten wir:

$$\begin{aligned} w \in L(A) &\Leftrightarrow \delta^*(s,w) \in F && \text{per Definition von } L(A) \\ &\Leftrightarrow [w]_L \in F && \text{wegen (*)} \\ &\Leftrightarrow [w]_L \subseteq L && \text{per Definition von } F \\ &\Leftrightarrow w \in L && \text{nach Lemma 2.30} \end{aligned}$$

□

Im Beweis des Satzes von Myhill und Nerode haben wir gesehen:

1. Wenn $A = (\Sigma, Q, s, F, \delta)$ ein DEA ist, der die Sprache L erkennt,

   und $u_1, u_2 \in \Sigma^*$ mit $\delta^*(s, u_1) = \delta^*(s, u_2)$,

   dann ist $[u_1]_L = [u_2]_L$.

   Also ist die Anzahl der L-Äquivalenzklassen höchstens so groß wie die Anzahl der (erreichbaren) Zustände von A.

2. Wenn $L \subseteq \Sigma^*$ nur endlich viele L-Äquivalenzklassen besitzt,

   dann gibt es einen DEA, der die Sprache L erkennt, und der die L-Äquivalenzklassen als Zustände besitzt.

   Also hat dieser DEA—wegen 1.—die *kleinstmögliche* Anzahl von Zuständen unter allen DEAs, die L erkennen.

Einen solchen DEA nennen wir einen *minimalen DEA* (für L).

Den speziellen minimalen DEA aus dem Beweis von Satz 2.31 nennen wir den *Myhill-Nerode-Automaten* für L.

Wir werden später sehen, dass er—in gewissem Sinne—der einzige minimale DEA für L ist.

Der Beweis von Satz 2.31 zeigt sogar, wie man den Myhill-Nerode-Automaten für L *konstruiert*, unter der Voraussetzung, dass man die L-Äquivalenzklassen 'kennt', d.h. dass man ihre Anzahl und ihre Elemente kennt (oder zumindest einen Teil der Elemente).

Wir wollen das an einem Beispiel illustrieren.

Beispiel: $\Sigma = \{0,1\}$ und $L = \{w \in \Sigma^* \mid 0100 \text{ ist Teilwort von } w\}$.
Bestimmung des Myhill-Nerode-Automaten:

$[\varepsilon]_L$	$Erg_L(\varepsilon) = L$
$[0]_L$	$Erg_L(0) = L \cup \{100\} \circ \Sigma^*$
$[1]_L = [\varepsilon]_L$	denn $Erg_L(1) = L = Erg_L(\varepsilon)$
	also $\delta([\varepsilon]_L, 1) = [1]_L = [\varepsilon]_L$
$[00]_L = [0]_L$	denn $Erg_L(00) = L \cup \{100\} \circ \Sigma^* = Erg_L(0)$
	also $\delta([0]_L, 0) = [00]_L = [0]_L$
$[01]_L$	$Erg_L(01) = L \cup \{00\} \circ \Sigma^*$
$[010]_L$	$Erg_L(010) = L \cup \{0, 100\} \circ \Sigma^*$

Man beachte, dass in $Erg_L(010)$ nicht nur die Wörter enthalten sind, die mit 0 beginnen, sondern auch die, die mit 100 beginnen, weil ja als letztes eine 0 gelesen wurde.

Mit anderen Worten: Man muss immer *alle* Präfixe des Suchmusters 0100 beachten, die man gerade gelesen hat, in diesem Falle nicht nur das Präfix 010, sondern auch das Präfix 0.

Weitere L-Äquivalenzklassen:

$[011]_L = [\varepsilon]_L$	denn $Erg_L(011) = L = Erg_L(\varepsilon)$
	also $\delta([01]_L, 1) = [011]_L = [\varepsilon]_L$
$[0100]_L$	$Erg_L(0100) = \Sigma^*$
$[0101]_L = [01]_L$	denn $Erg_L(0101) = L \cup \{00\} \circ \Sigma^* = Erg_L(01)$
	also $\delta([010]_L, 1) = [0101]_L = [01]_L$
$[0100a]_L = [0100]_L$	denn $Erg_L(0100a) = \Sigma^* = Erg_L(0100)$
	also $\delta([0100]_L, a) = [0100a]_L = [0100]_L$

Damit haben wir alle L-Äquivalenzklassen und die 'nichttrivialen' Übergänge zwischen ihnen bestimmt.

Die 'trivialen' Übergänge sind

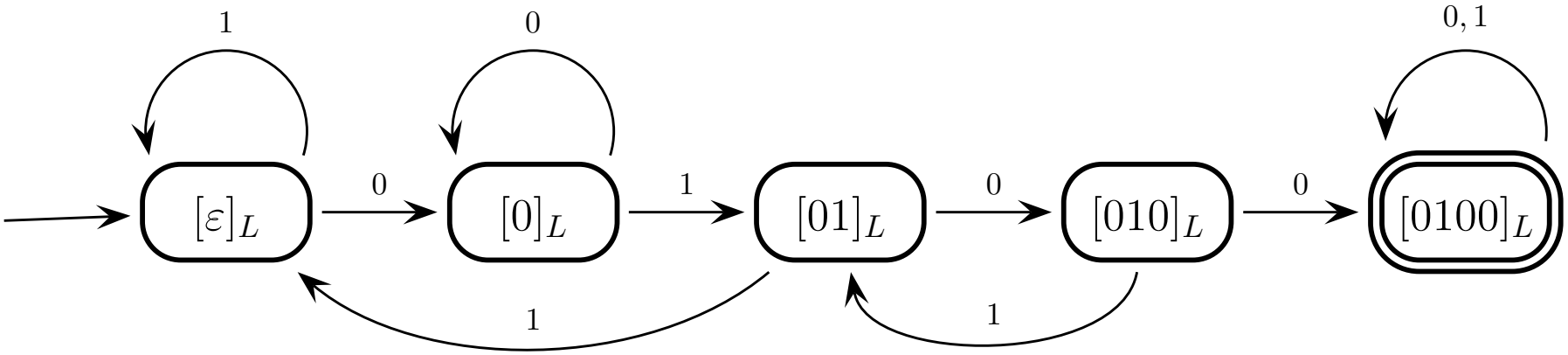
$\delta([\varepsilon]_L, 0) = [0]_L$

$\delta([0]_L, 1) = [01]_L$

$\delta([01]_L, 0) = [010]_L$

$\delta([010]_L, 0) = [0100]_L$

Also sieht der Myhill-Nerode-Automat für L so aus:



Verallgemeinerung des Beipiels:

Sei Σ ein Alphabet und sei $u = a_1 \dots a_n \in \Sigma^*$.
Wie sieht ein (minimaler) DEA A_u aus, der die Sprache

$$L_u = \{w \in \Sigma^* \mid u \text{ ist Teilwort von } w\}$$

erkennt? Offensichtlich genügt es (wie im Beispiel $u = 0100$), dass sich A_u das "bisher erkannte Präfix" des Suchwortes u merkt.

Genauer:

Sei w das bisher gelesene Wort.

Wenn w bereits in L_u liegt, d.h. wenn w schon das ganze Suchwort u enthält, dann sollte sich A_u in einem Endzustand befinden und diesen nicht mehr verlassen.

Wenn w noch nicht in L_u liegt, dann sollte A_u das längste Präfix von u kennen, das zugleich Suffix von w ist.

Damit bieten sich die Präfixe des Suchwortes u als Zustände des DEA A_u an: Sei $u_i = a_1 \dots a_i$ $(i = 0, \dots, n)$ das Präfix der Länge i von u. Dann definiert man einen DEA $A_u = (\Sigma, Q, s, F, \delta)$ mit

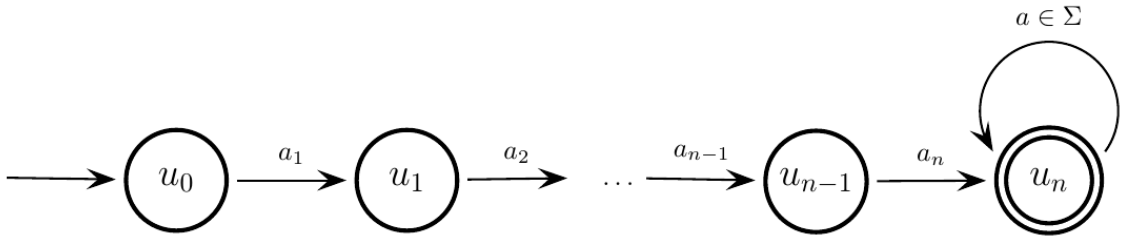
$$Q = \mathit{Pref}(u) = \{u_0, \dots, u_n\}$$

$$s = u_0 = \varepsilon$$

$$F = \{u_n\} = \{u\}$$

$$\delta(u_{i-1}, a_i) = u_i \text{ für } i = 0, \dots, n$$

Damit ist das "Rückgrat" (engl. *spine*) des Automaten A_u festgelegt



und es fehlen nur noch die Übergänge $\delta(u_{i-1}, a)$ mit $a \neq a_i$.

Man erhält sie durch folgende Überlegung: A_u soll sich das längste Präfix von u merken, das Suffix des bisher gelesenen Wortes w ist. Deshalb muss für alle $w \in \Sigma^*$ gelten: Wenn u_{i-1} das längste Wort in $\mathit{Pref}(u) \cap \mathit{Suff}(w)$ ist, dann ist $\delta(u_{i-1}, a) = u_j$ das längste Wort in $\mathit{Pref}(u) \cap \mathit{Suff}(wa)$.

Wählt man insbesondere $w = u_{i-1}$, so ergibt sich $\delta(u_{i-1}, a) = u_j$, wobei u_j das *längste Wort in* $\mathit{Pref}(u) \cap \mathit{Suff}(u_{i-1}a)$ ist. Damit ist die Übergangsfunktion δ des Automaten A_u vollständig festgelegt.

Durch Induktion über die Länge von w lässt sich zeigen, dass jedes Wort $w \in \Sigma^*$ tatsächlich zum gewünschten Zustand führt, d.h.

$$\delta^*(s, w) = \begin{cases} u_n & \text{falls } w \in L_u \\ \text{das längste } u_i \in \mathit{Pref}(u) \cap \mathit{Suff}(w) & \text{sonst} \end{cases}$$

Die Minimalität von A_u ergibt sich daraus, dass $u_i, u_j \in \mathit{Pref}(u)$ mit $i \neq j$ nicht zum gleichen Zustand führen dürfen.

Wir haben also einen *Algorithmus*, der der zu jedem Wort $u \in \Sigma^*$ einen minimalen 'Suchautomaten' für u liefert, d.h. einen minimalen DEA A_u, der genau die Texte akzeptiert, die das Wort u enthalten.

Auf diesem Algorithmus basieren *Suchprogramme*:

Für jedes Suchwort u wird ein DEA konstruiert, der L_u erkennt.
Dieser DEA wird auf den zu durchsuchenden Text angesetzt.

Lohnt sich der Aufwand?

Ja, weil die Länge m des zu durchsuchenden Textes meist viel größer ist als die Länge n des Suchwortes.
Der Zeitaufwand für die Konstruktion des DEA ist linear in n.
Die Laufzeit des DEA beträgt m.
Also ist die Gesamtlaufzeit nur wenig größer als m.

Zum Vergleich:
Beim 'naiven' Suchalgorithmus testet man für $i = 1, \ldots, m-n+1$, ob das Suchwort u an der Stelle i des Textes w beginnt.

Das erfordert im schlimmsten Fall Laufzeit $n\,(m-n+1)$,
z.B. wenn $u = 0^{n-1}1$ und $w = 0^m$:
An jeder Stelle i von w muss man bis zum letzten Zeichen von u laufen, um zu sehen, dass u nicht passt.

In der Praxis ist man natürlich nicht nur daran interessiert, *ob* das Suchwort im Text vorkommt, sondern auch *wo* es vorkommt. Dazu konstruiert man sich einen DEA A'_u, der die Sprache $L'_u = \{w \in \Sigma^* \mid u$ ist *Suffix* von $w\}$ erkennt, und der immer dann eine Markierung im Suchtext setzt, wenn er gerade im Endzustand ist, d.h. wenn er das Suchwort gerade gefunden hat.
A'_u unterscheidet sich von A_u nur darin, dass er nicht unbedingt im Endzustand bleibt, wenn er das Suchwort u schon gefunden hat, sondern von dort mit jedem Zeichen a in den Zustand $[u_j]$ mit $j = \max\{u_i \in \{u_0, \ldots, u_n\} \mid u_i$ ist Suffix von $ua\}$ übergeht.

Minimierung

Bisher:
Konstruktion eines minimalen DEA für L aus den L-Äquivalenzklassen.

Nachteil:
Zur Bestimmung der L-Äquivalenzklassen gibt es keinen Algorithmus, sondern es ist im allgemeinen 'mathematische Argumentation' nötig (die in Spezialfällen wie bei der Sprache L_u einen Algorithmus zur Konstruktion des DEA liefern kann).

Deshalb stellt sich die Frage:
Kann man aus einem beliebigen DEA einen äquivalenten minimalen DEA erhalten?.

Lösungsansatz:
Wenn ein DEA unnötige Unterscheidungen trifft, d.h. wenn zwei L-äquivalente Wörter in unterschiedlichen Zuständen landen, dann kann man diese Zustände miteinander 'verschmelzen'.

'Verschmelzen' bedeutet aus mathematischer Sicht:
Man fasst die Zustände zu *Äquivalenzklassen* zusammen.

Definition 2.32 *Sei $A = (\Sigma, Q, s, F, \delta)$ ein DEA. Zwei Zustände $p, q \in Q$ heißen A-äquivalent (Schreibweise: $p \sim_A q$), wenn für alle $w \in \Sigma^*$ gilt:*

$$\delta^*(p, w) \in F \;\Leftrightarrow\; \delta^*(q, w) \in F$$

Mit anderen Worten:

$$\begin{aligned} p \sim_A q \;\Leftrightarrow\; & \text{für alle } w \in \Sigma^* \text{ gilt:} \\ & \text{entweder } \delta^*(p, w), \delta^*(q, w) \in F \\ & \text{oder } \delta^*(p, w), \delta^*(q, w) \notin F \end{aligned}$$

A-Äquivalenz ist eine Äquivalenzrelation auf der Menge Q. Die A-Äquivalenzklasse eines Zustands q bezeichnen wir mit $[q]_A$, also

$$[q]_A = \{p \in Q \mid p \sim_A q\}$$

Lemma 2.33

1. *Wenn* $p \sim_A q$*, dann gilt auch* $\delta^*(p,v) \sim_A \delta^*(q,v)$ *für alle* $v \in \Sigma^*$.
2. *Für jeden Zustand* $q \in Q$ *gilt:* $q \in F \Leftrightarrow [q]_A \subseteq F$
   (Also $F = \bigcup_{q \in F} [q]_A$*.)*

Beweis:

1. Sei $p \sim_A q$ und $v \in \Sigma^*$.

   Dann gilt für alle $v' \in \Sigma^*$:

   $\delta^*(p, vv') \in F \Leftrightarrow \delta^*(q, vv') \in F$,

   also $\delta^*(\delta^*(p,v), v') \in F \Leftrightarrow \delta^*(\delta^*(q,v), v') \in F$,

   und das bedeutet $\delta^*(p,v) \sim_A \delta^*(q,v)$.

2. '$\Leftarrow$' ist klar.

   '$\Rightarrow$': Sei $q \in F$ und $p \sim_A q$.

   Dann ist wegen $\delta^*(q, \varepsilon) \in F$ auch $p = \delta^*(p, \varepsilon) \in F$. □

Satz 2.34 *Zu jedem DEA A lässt sich ein äquivalenter minimaler DEA $\bar{A}$ konstruieren. $\bar{A}$ ist isomorph zum Myhill-Nerode-Automaten für die Sprache $L(A)$, d.h. er unterscheidet sich von ihm nur durch eine Umbenennung der Zustände.*

Beweis:

Sei $A = (\Sigma, Q, s, F, \delta)$.

Wir dürfen annehmen, dass A nur erreichbare Zustände besitzt.

Wir definieren $\bar{A} = (\Sigma, \bar{Q}, \bar{s}, \bar{F}, \bar{\delta})$ durch

$\bar{Q} = \{[q]_A \mid q \in Q\}$

$\bar{s} = [s]_A$

$\bar{F} = \{[q]_A \mid q \in F\} \quad (= \{[q]_A \mid [q]_A \subseteq F\})$

$\bar{\delta} : \bar{Q} \times \Sigma \to \bar{Q}$

$\bar{\delta}([q]_A, a) = [\delta(q, a)]_A$

$\bar{\delta}$ ist wohldefiniert, denn aus $p \sim_A q$ folgt nach Lemma 2.33 stets $\delta(p,a) \sim_A \delta(q,a)$, und durch Induktion über die Länge von w zeigt man leicht, dass für alle $q \in Q$ und alle $w \in \Sigma^*$ gilt:

$$\bar{\delta}^*([q]_A, w) = [\delta^*(q,w)]_A \qquad (*)$$

Damit folgt:

$$\begin{array}{lcll} w \in L(\bar{A}) & \Leftrightarrow & \bar{\delta}^*([s]_A, w) \in \bar{F} & \text{per Definition von } L(\bar{A}) \\ & \Leftrightarrow & [\delta^*(s,w)]_A \in \bar{F} & \text{wegen } (*) \\ & \Leftrightarrow & [\delta^*(s,w)]_A \subseteq F & \text{per Definition von } \bar{F} \\ & \Leftrightarrow & \delta^*(s,w) \in F & \text{nach Lemma 2.33} \\ & \Leftrightarrow & w \in L(A) & \text{per Definition von } L(A) \end{array}$$

Also ist $L(\bar{A}) = L(A)$, d.h. $\bar{A}$ ist äquivalent zu A.

Außerdem gilt für alle $u_1, u_2 \in \Sigma^*$:

$$
\begin{aligned}
u_1 \sim_{L(A)} u_2 \quad &\Leftrightarrow \quad \text{für alle } w \in \Sigma^* \text{ gilt} \\
&\qquad u_1 w \in L(A) \Leftrightarrow u_2 w \in L(A) \\
&\Leftrightarrow \quad \text{für alle } w \in \Sigma^* \text{ gilt} \\
&\qquad \delta^*(s, u_1 w) \in F \Leftrightarrow \delta^*(s, u_2 w) \in F \\
&\Leftrightarrow \quad \text{für alle } w \in \Sigma^* \text{ gilt} \\
&\qquad \delta^*(\delta^*(s, u_1), w) \in F \Leftrightarrow \delta^*(\delta^*(s, u_2), w) \in F \\
&\Leftrightarrow \quad \delta^*(s, u_1) \sim_A \delta^*(s, u_2) \\
&\Leftrightarrow \quad [\delta^*(s, u_1)]_A = [\delta^*(s, u_2)]_A \\
&\Leftrightarrow \quad \overline{\delta}^*([s]_A, u_1) = \overline{\delta}^*([s]_A, u_2)
\end{aligned}
$$

wobei die letzte Umformung wieder wegen (*) gilt.

Zwei Wörter u_1, u_2 sind also genau dann $L(A)$-äquivalent, wenn sie in $\bar{A}$ zum gleichen Zustand führen.

Das bedeutet, dass die Anzahl der erreichbaren Zustände von $\bar{A}$ nicht größer ist als die Anzahl der $L(A)$-Äquivalenzklassen, d.h. die Anzahl der Zustände des Myhill-Nerode-Automaten.

Und weil $\bar{A}$—wie der ursprüngliche Automat A—nur erreichbare Zustände hat, folgt daraus, dass er die kleinstmögliche Anzahl von Zuständen hat, d.h. $\bar{A}$ ist minimal.

Die folgende genauere Argumentation zeigt, dass er sogar zum Myhill-Nerode-Automaten isomorph ist.

Sei $\tilde{A} = (\Sigma, \tilde{Q}, \tilde{s}, \tilde{F}, \tilde{\delta})$ der Myhill-Nerode-Automat für $L(A)$.

Wir definieren eine Abbildung

$$\Phi : \tilde{Q} \rightarrow \bar{Q}$$
$$\Phi([w]_{L(A)}) = \bar{\delta}^*([s]_A, w)$$

Nach den obigen Betrachtungen über die $L(A)$-Äquivalenzklassen ist Φ wohldefiniert und injektiv, und weil $\bar{A}$ nur erreichbare Zustände hat, ist Φ auch surjektiv.

Darüber hinaus gilt:

- $\Phi(\tilde{s}) = \Phi([\varepsilon]_{L(A)}) = \bar{\delta}^*([s]_A, \varepsilon) = [s]_A$
- $\Phi([w]_{L(A)}) \in \bar{F} \Leftrightarrow \bar{\delta}^*([s]_A, w) \in \bar{F} \Leftrightarrow w \in L(\bar{A})$
  $\Leftrightarrow w \in L(A) \Leftrightarrow [w]_{L(A)} \in \tilde{F}$
- $\Phi(\tilde{\delta}([w]_{L(A)}, a)) = \Phi([wa]_{L(A)}) = \bar{\delta}^*([s]_A, wa)$
  $= \bar{\delta}(\bar{\delta}^*([s]_A, w), a) = \bar{\delta}(\Phi([w]_{L(A)}), a)$

Also ist Φ eine bijektive Abbildung, unter der sich die Start- und Endzustände und die Zustandsübergänge in den beiden Automaten entsprechen.

Das bedeutet, dass Φ ein Isomorphismus ist, d.h. eine reine Umbenennung der Zustände.

Es bleibt zu zeigen, dass sich der DEA $\bar{A}$ aus dem DEA A *konstruieren* lässt, d.h. dass sich die A-Äquivalenzklassen bestimmen lassen (alles andere ist klar).

Dazu definieren wir Relationen $\sim_n$ auf Q durch Induktion über n:

$$\begin{aligned} p \sim_0 q &\Leftrightarrow (p \in F \Leftrightarrow q \in F) \\ &\Leftrightarrow \text{entweder } p, q \in F \text{ oder } p, q \notin F \\ p \sim_{n+1} q &\Leftrightarrow p \sim_n q \text{ und für alle } a \in \Sigma \text{ gilt } \delta(p, a) \sim_n \delta(q, a) \end{aligned}$$

Durch Induktion über n folgt leicht, dass für jedes n gilt:

$$\begin{aligned} p \sim_n q \Leftrightarrow\ & \text{für } \textit{alle } w \in \Sigma^* \text{ mit } |w| \leq n \text{ gilt} \\ & \delta^*(p, w) \in F \Leftrightarrow \delta^*(q, w) \in F \end{aligned}$$

Damit ist klar, dass jedes $\sim_n$ eine Äquivalenzrelation ist, und dass $\sim_A$ der Durchschnitt aller $\sim_n$ ist.

Also bleibt nur noch zu zeigen, dass sich $\bigcap_{n\geq 0} \sim_n$ berechnen lässt.

Zunächst beachte man, dass (per Definition) $\sim_{n+1} \subseteq \sim_n$ für alle $n \geq 0$ gilt, d.h. die $\sim_n$ bilden eine absteigende Folge $\sim_0 \supseteq \sim_1 \supseteq \ldots$ von Relationen auf Q, also eine absteigende Folge von Teilmengen von $Q \times Q$.

Da $Q \times Q$ endlich ist, kann eine solche Folge *nicht echt absteigend* sein, also gibt es eine Zahl m mit $\sim_m = \sim_{m+1}$.

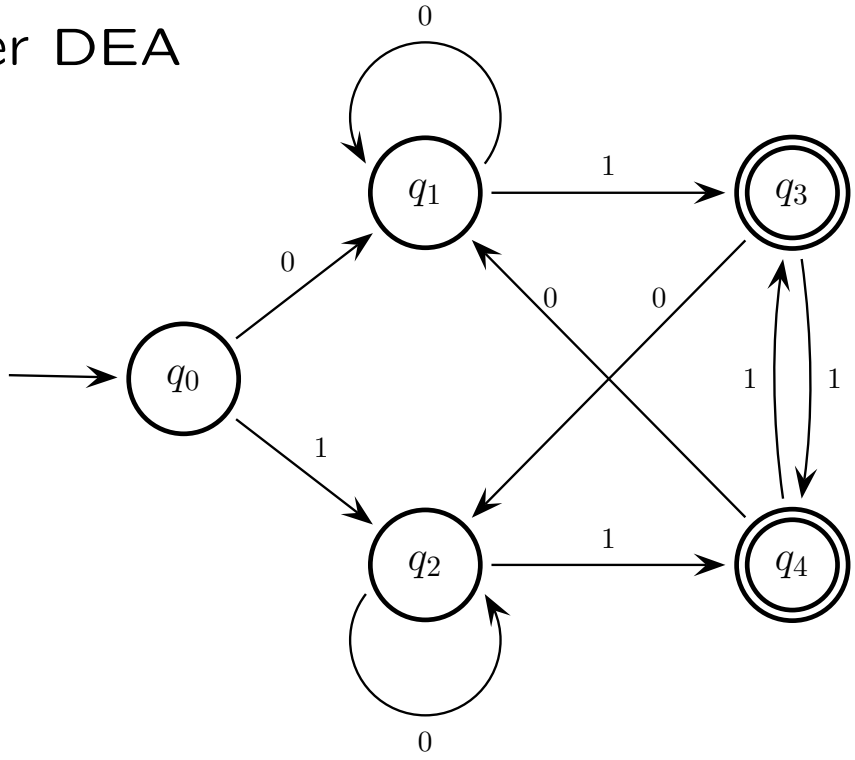
Weil sich $\sim_{m+2}$ wieder auf die gleiche Art aus $\sim_{m+1}$ ergibt wie $\sim_{m+1}$ aus $\sim_m$, folgt dann $\sim_m = \sim_n$ für alle $n \geq m$,

und das bedeutet, dass $\sim_m = \bigcap_{n\geq 0} \sim_n = \sim_A$.

Damit haben wir einen Algorithmus zur Berechnung von $\sim_A$ gefunden: Wir berechnen die Relationen $\sim_0, \sim_1, \ldots$ bis wir das erste m finden mit $\sim_m = \sim_{m+1}$.

Für dieses m gilt dann $\sim_m = \sim_A$. □

Beispiel: Sei A der DEA



$\sim_0$ hat die Äquivalenzklassen $Q \setminus F = \{q_0, q_1, q_2\}$ und $F = \{q_3, q_4\}$.

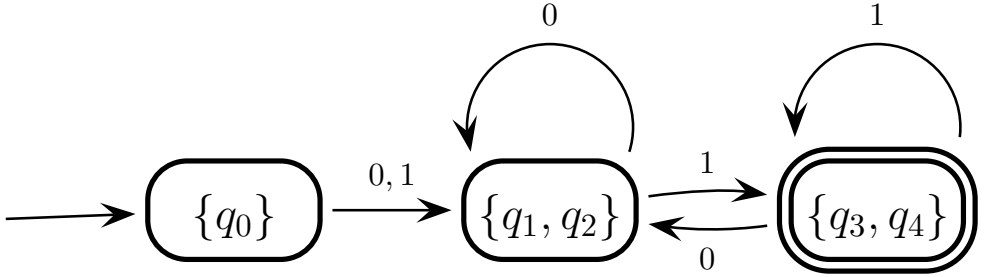
1. Verfeinerungsschritt:

$\delta(q_0, 0) = \delta(q_1, 0) = q_1$ und $\delta(q_2, 0) = q_2$ liegen in $Q \setminus F$,
aber $\delta(q_0, 1) = q_2 \in Q \setminus F$ und $\delta(q_1, 1) = q_3, \delta(q_2, 1) = q_4$ liegen in F.
Also zerfällt $\{q_0, q_1, q_2\}$ in zwei $\sim_1$-Klassen $\{q_0\}$ und $\{q_1, q_2\}$.

$\delta(q_3, 0) = q_2$ und $\delta(q_4, 0) = q_1$ liegen in $Q \setminus F$,
$\delta(q_3, 1) = q_4$ und $\delta(q_4, 1) = q_3$ liegen in F.
Also bleibt die $\sim_0$-Klasse $\{q_3, q_4\}$ als $\sim_1$-Klasse erhalten.
Damit besitzt $\sim_1$ die drei Äquivalenzklassen $\{q_0\}, \{q_1, q_2\}$ und $\{q_3, q_4\}$.

2. Verfeinerungsschritt:

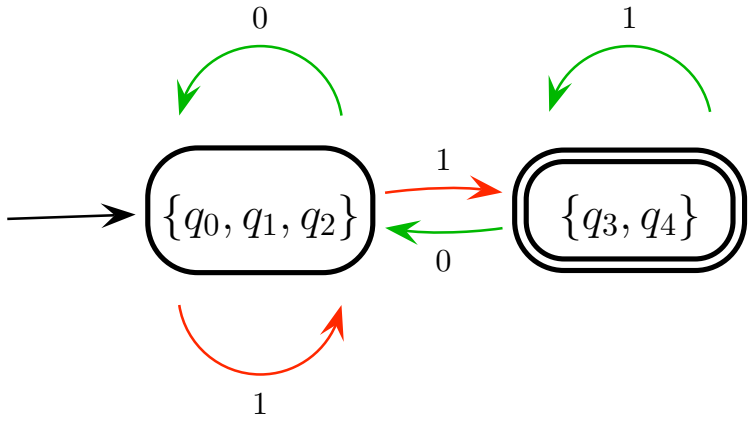
Die $\sim_1$-Klasse $\{q_0\}$ ist nicht mehr weiter aufteilbar.
$\delta(q_1, 0), \delta(q_2, 0) \in \{q_1, q_2\}$ und $\delta(q_1, 1), \delta(q_2, 1) \in \{q_3, q_4\}$.
Also bleibt die $\sim_1$-Klasse $\{q_1, q_2\}$ als $\sim_2$-Klasse erhalten.
$\delta(q_3, 0), \delta(q_4, 0) \in \{q_1, q_2\}$ und $\delta(q_3, 1), \delta(q_4, 1) \in \{q_3, q_4\}$.
Also bleibt auch die $\sim_1$-Klasse $\{q_3, q_4\}$ als $\sim_2$-Klasse erhalten.
Damit ist $\sim_1 = \sim_2$ gezeigt, also ist $\sim_2 = \sim_A$,
und man erhält den folgenden zu A äquivalenten minimalen DEA:



Eine anschauliche Erklärung für die Relationen $\sim_n$:

Man kann jede dieser Äquivalenzrelationen als 'Versuch' auffassen, den minimalen DEA zu konstruieren:
Man nimmt die Äquivalenzklassen von $\sim_n$ als Zustände, und versucht die Zustandsübergänge zwischen ihnen richtig festzulegen.
Scheitert der Versuch, so sieht man daran, wie die Äquivalenzklassen weiter aufgeteilt werden müssen. Gelingt der Versuch, so hat man den minimalen DEA gefunden (weil man mit $\sim_0$, d.h. mit der kleinstmöglichen Anzahl von Zuständen begonnen und immer nur die notwendigen Verfeinerungen vorgenommen hat).

An unserem Beispiel sieht das so aus:
Man versucht zunächst, einen DEA zu konstruieren, der nur die beiden Äquivalenzklassen $\{q_0, q_1, q_2\}$ und $\{q_3, q_4\}$ von $\sim_0$ als Zustände hat:



Das geht gut für die Übergänge

$\delta(q_0, 0), \delta(q_1, 0), \delta(q_2, 0) \in \{q_0, q_1, q_2\}$,

$\delta(q_3, 0), \delta(q_4, 0) \in \{q_0, q_1, q_2\}$,

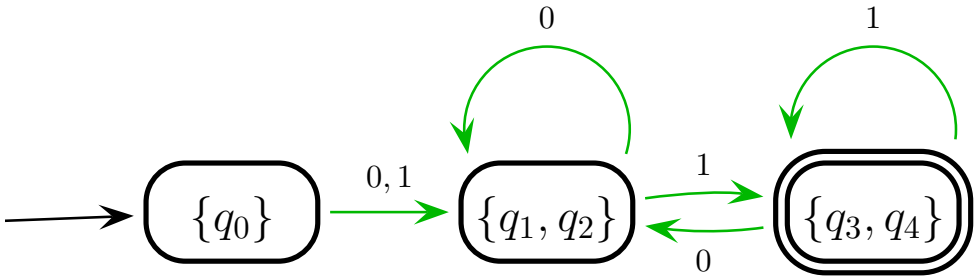
und $\delta(q_3, 1), \delta(q_4, 1) \in \{q_3, q_4\}$,

scheitert aber an den sich widersprechenden Übergängen

$\delta(q_0, 1) \in \{q_0, q_1, q_2\}$ und $\delta(q_1, 1), \delta(q_2, 1) \in \{q_3, q_4\}$.

An den gescheiterten Übergängen sieht man, dass $\{q_0, q_1, q_2\}$ in $\{q_0\}$ und $\{q_1, q_2\}$ aufgeteilt werden muss.

Also versucht man jetzt, einen DEA zu konstruieren, der die drei Äquivalenzklassen $\{q_0\}$, $\{q_1, q_2\}$ und $\{q_3, q_4\}$ von $\sim_1$ als Zustände hat:



Jetzt geht mit den Übergängen alles gut:

$\delta(q_0, 0) \in \{q_1, q_2\}$ $\quad$ $\delta(q_0, 1) \in \{q_1, q_2\}$

$\delta(q_1, 0), \delta(q_2, 0) \in \{q_1, q_2\}$ $\quad$ $\delta(q_1, 1), \delta(q_2, 1) \in \{q_3, q_4\}$

$\delta(q_3, 0), \delta(q_4, 0) \in \{q_1, q_2\}$ $\quad$ $\delta(q_3, 1), \delta(q_4, 1) \in \{q_3, q_4\}$

Also ist der minimale DEA gefunden.

Das Pumping Lemma (dt.: **Schleifensatz**) bietet eine weitere Methode, um zu zeigen, dass eine Sprache *nicht* regulär ist.

Es beschreibt eine *Eigenschaft* regulärer Sprachen. Um zu beweisen, dass eine Sprache *nicht* regulär ist, genügt es zu zeigen, dass sie diese Eigenschaft *nicht* besitzt.

Mit anderen Worten:
Das Pumping Lemma gibt eine *notwendige* Bedingung dafür an, dass eine Sprache regulär ist. Kann man zeigen, dass diese Bedingung nicht erfüllt ist, so weiß man, dass die Sprache nicht regulär ist.

Zum Vergleich:
Der Satz von Myhill und Nerode gibt eine *hinreichende und notwendige* Bedingung dafür an, dass eine Sprache regulär ist. Deshalb kann er für beide Zwecke benutzt werden: Sowohl zum Nachweis, dass eine Sprache regulär ist, als auch dazu, dass sie nicht regulär ist.

Die Idee:

Sei A ein DEA für die reguläre Sprache L, und sei n die Anzahl der Zustände von A.

Sei $x \in L$ mit $|x| \geq n$.

Dann werden $n+1$ Zustände bei der Abarbeitung von x auf A durchlaufen. Da A nur n Zustände hat, wird also mindestens ein Zustand mehrmals durchlaufen, d.h. der Weg, auf dem x abgearbeitet wird, enthält eine *Schleife*.

Durchläuft man diese Schleife öfter oder weniger oft als bei der Abarbeitung des Wortes x, so erhält man andere Wörter, die zum gleichen Endzustand führen wie x, die also ebenfalls in L liegen.

Wie sehen diese Wörter aus?

Sei $x = uvw$, wobei $v \neq \varepsilon$ das Teilwort ist, das in der Schleife abgearbeitet wird. Dann kann man v beliebig oft wiederholen, und erhält damit Wörter der Form $x_i = uv^iw$ $(i \geq 0)$, die alle in L liegen.

Das Wort x_i entsteht sozusagen durch *Aufpumpen* des Wortes x an der Stelle v mit "Pumpfaktor" i. (Man beachte, dass $x_1 = uv^1w = x$, d.h. der Pumpfaktor 1 bewirkt *nichts*.)

Satz 2.35 (Pumping Lemma) *Sei $L \subseteq \Sigma^*$ regulär. Dann existiert eine Zahl $n \geq 1$, so dass für jedes $x \in L$ mit $|x| \geq n$ gilt: Es gibt eine Zerlegung $x = uvw$ mit $v \neq \varepsilon$ und $uv^iw \in L$ für alle $i \geq 0$.*

Intuition:
Wenn L regulär ist, dann kann man jedes hinreichend lange Wort $x \in L$ an mindestens einer Stelle v mit beliebigem Faktor i aufpumpen, ohne dass man die Sprache L verlässt.

Beweis:

Sei $A = (\Sigma, Q, s, F, \delta)$ ein DEA, der L erkennt.

Wir wählen $n = |Q|$.

Wenn $x \in L$ mit $|x| \geq n$, dann wird bei der Abarbeitung von x mindestens ein Zustand mehrmals besucht,

d.h. es existieren $p \in Q, q \in F$ und $u, v, w \in \Sigma^*$ mit $x = uvw$ und

$$(s, x) = (s, uvw) \vdash_A^* (p, vw) \vdash_A^+ (p, w) \vdash_A^* (q, \varepsilon)$$

Dabei ist $v \neq \varepsilon$, weil in $\vdash_A^+$ mindestens ein Zeichen verarbeitet wird.

Indem man die $\vdash_A^+$-Schleife i-mal wiederholt, folgt

$$(s, uv^i w) \vdash_A^* (p, v^i w) \vdash_A^+ \ldots \vdash_A^+ (p, w) \vdash_A^* (q, \varepsilon)$$

also $uv^i w \in L$ für jedes $i \geq 0$. □

Korollar 2.36 *Zum Nachweis, dass L nicht regulär ist, genügt es zu zeigen: Für jedes $n \geq 1$ existiert ein $x \in L$ mit $|x| \geq n$, so dass für alle Zerlegungen $x = uvw$ mit $v \neq \varepsilon$ gilt: Es gibt ein $i \geq 0$ mit $uv^iw \notin L$.*

Intuition:
Es ist zu zeigen, dass *beliebig lange* Wörter $x \in L$ existieren, die sich an *jeder* Stelle v so aufpumpen lassen, dass das Ergebnis nicht mehr in L liegt.

Der Beweis, dass eine Sprache L nicht regulär ist, läuft also so ab: Für *jedes* $n \geq 1$ gibt man ein *passendes* Wort x mit $|x| \geq n$ an. Dann muss man für *jede* Zerlegung $x = uvw$ mit $v \neq \varepsilon$ einen *passenden* Pumpfaktor i wählen mit $uv^iw \notin L$.

Merke:
x und i darf man *passend wählen*, aber man muss *alle* Zerlegungen des Wortes x betrachten. Oft ist dabei eine Fallunterscheidung notwendig, weil man *alle* Möglichkeiten für die Position des Teilwortes v im Wort x durchspielen muss.

Beispiel:

$L_1 = \{a^n b^m \mid 0 \leq n \leq m\}$ ist nicht regulär, denn:

Sei $n \geq 1$.

Man wähle $x = a^n b^n$, also $|x| = 2n \geq n$.

Sei $x = uvw$ eine Zerlegung von x mit $v \neq \varepsilon$.

1. Fall: v liegt in a^n, d.h. $v = a^k$ für ein $k \geq 1$.

Dann gilt $uv^2w = a^{n+k}b^n \notin L_1$, weil $n + k > n$ wegen $k \geq 1$.

2. Fall: v liegt an der Schnittstelle, d.h. $v = a^i b^j$ mit $i, j \geq 1$.

Dann gilt $uv^2w = a^{n-i}a^i b^j a^i b^j b^{n-j} = a^n b^j a^i b^n \notin L_1$, denn wegen $i, j \geq 1$ stehen as hinter bs.

3. Fall: v liegt in b^m, d.h. $v = b^k$ für ein $k \geq 1$.

Dann gilt $uv^0w = a^n b^{n-k} \notin L_1$, weil $n - k < n$ wegen $k \geq 1$.

Am Beispiel wird klar:
Die Schwierigkeit bei der Anwendung des Pumping Lemmas besteht darin, dass man nicht weiß, wo das Teilwort v liegt. Das kann zu mühsamen Fallunterscheidungen führen oder ganz schiefgehen:

Beispiel:

$L_2 = \{w \in \{a,b\}^* \mid \#_a(w) = \#_b(w)\}$.

Dann gibt es für *jedes* $x \in L_2$ mit $|x| \geq 1$ eine Zerlegung $x = uvw$ mit $v \neq \varepsilon$, so dass $uv^iw \in L_2$ für *alle* $i \geq 0$.

Nämlich $u = w = \varepsilon$ und $v = x$.

Dann ist $uv^iw = x^i \in L_2$ für alle $i \geq 0$.

Also kann man gar kein passendes Wort x finden, um mit Satz 2.35 nachzuweisen, dass L_2 nicht regulär ist.

Mit anderen Worten:

L_2 erfüllt die im Pumping Lemma genannte Eigenschaft regulärer Sprachen, ohne selbst regulär zu sein. Also ist diese Eigenschaft eine notwendige, aber keine hinreichende Bedingung dafür, dass die Sprache regulär ist.

Die meisten Beispiele lassen sich bewältigen, wenn man das Pumping Lemma wie folgt verstärkt.

Satz 2.37 (Starkes Pumping Lemma) *Sei $L \subseteq \Sigma^*$ regulär. Dann existiert eine Zahl $n \geq 1$, so dass für jedes $x \in L$ mit $|x| \geq n$ gilt: Es gibt eine Zerlegung $x = uvw$ mit $v \neq \varepsilon$, $|uv| \leq n$ und $uv^iw \in L$ für alle $i \geq 0$.*

Man erhält diese Aussage, indem man den Beweis von Satz 2.35 leicht abändert: Anstelle des Wortes x betrachtet man das Präfix y von x der Länge n. Schon bei der Abarbeitung von y muss mindestens ein Zustand mehrmals auftauchen, also können wir das Wort v so wählen, dass es in y liegt, und das bedeutet $|uv| \leq n$.

Das starke Pumping Lemma eröffnet neue Möglichkeiten, weil wir für das (passend gewählte) Wort x nicht mehr *alle* Zerlegungen $x = uvw$ betrachten müssen, sondern nur noch diejenigen mit $|uv| \leq n$.

Damit vereinfacht sich z.B. der Beweis für $L_1 = \{a^n b^m \mid 0 \leq n \leq m\}$: Man wählt nach wie vor $x = a^n b^n$. Aber jetzt braucht man nur noch Zerlegungen der Form $x = uvw$ mit $|uv| \leq n$ zu untersuchen. $|uv| \leq n$ bedeutet aber, dass v ganz in a^n liegen muss, also bleibt von den drei Fällen im obigen Beweis nur der erste übrig.

Der Beweis für $L_2 = \{w \in \{a,b\}^* \mid \#_a(w) = \#_b(w)\}$ gelingt jetzt auch:

Sei $n \geq 1$.

Man wähle $x = a^n b^n$, also $x \in L_2$ und $|x| = 2n \geq n$.

Sei $x = uvw$ eine Zerlegung mit $v \neq \varepsilon$ und $|uv| \leq n$.

Dann liegt v in a^n, also $v = a^k$ für ein $k \geq 1$.

Also ist $uv^2w = a^{n+k}b^n \notin L_2$, weil $n + k \neq n$ wegen $k \geq 1$.

Weiteres Beispiel:

Sei $L_3 = \{a^p \mid p \text{ ist Primzahl}\}$.
Es wurde schon mit dem Satz von Myhill-Nerode bewiesen, dass L_3 nicht regulär ist. Mit dem Pumping Lemma geht es hier einfacher:

Sei $n \geq 1$.

Man wählt $x = a^p$ für eine Primzahl $p \geq n$, also ist $x \in L_3$ und $|x| \geq n$.

Sei $x = uvw$ eine Zerlegung mit $v \neq \varepsilon$, d.h. $v = a^k$ mit $k \geq 1$.

Dann gilt $uv^{p+1}w = a^{p+kp} = a^{p(1+k)} \notin L_3$, denn wegen $1 + k \geq 2$ ist p ein echter Teiler von $p(1 + k)$, also $p(1 + k)$ keine Primzahl.

Wie kommt man auf diesen Pumpfaktor?

Die Idee besteht darin, das Wort $x = a^p$ um ein Vielfaches von p zu *verlängern*. Mit Pumpfaktor p würde man das Teilwort $v = a^k$ durch $v^p = a^{kp}$ ersetzen, also x nur um $k(p-1)$ verlängern. Deshalb ist nicht p sondern $p + 1$ der passende Pumpfaktor.

Übrigens ist auch die im starken Pumping Lemma genannte Eigenschaft noch keine hinreichende Bedingung für die Regularität einer Sprache, aber in den meisten Fällen reicht es aus, um zu zeigen, dass eine Sprache nicht regulär ist.

Andere Verstärkungen des Pumping Lemmas sind denkbar, z.B. kann man $|uv| \leq n$ durch $|vw| \leq n$ ersetzen.

Offene Frage:

Kann man das Pumping Lemma so verstärken, dass man eine hinreichende und notwendige Bedingung für die Regularität der Sprache erhält?

Kontextfreie Sprachen

- Reguläre Sprachen haben eine sehr einfache Struktur.

  Deshalb kann man mit ihnen nur die kleinsten (d.h. einfachsten) syntaktischen Einheiten einer Programmiersprache erfassen,

  z.B. Schlüsselwörter, Zahldarstellungen, Bezeichner, Kommentare.

- Wie beschreibt man die komplizierteren syntaktischen Strukturen,

  z.B. arithmetische und boolesche Ausdrücke, Anweisungen, Deklarationen?

  Am besten durch eine *Konstruktionsvorschrift*,

  d.h. aus mathematischer Sicht: durch eine *induktive Definition*.

Beispiel: Vollständig geklammerte arithmetische Ausdrücke

Konstruktionsvorschrift

1. Jede Dezimalzahl ist ein arithmetischer Ausdruck.
2. Jeder Bezeichner ist ein arithmetischer Ausdruck.
3. Wenn e ein arithmetischer Ausdruck ist, dann ist auch $(-e)$ ein arithmetischer Ausdruck.
4. Wenn e_1, e_2 arithmetische Ausdrücke sind, dann ist auch $(e_1 + e_2)$ ein arithmetischer Ausdruck.
5. Wenn e_1, e_2 arithmetische Ausdrücke sind, dann ist auch $(e_1 - e_2)$ ein arithmetischer Ausdruck.
6. Wenn e_1, e_2 arithmetische Ausdrücke sind, dann ist auch $(e_1 * e_2)$ ein arithmetischer Ausdruck.

Mathematische Formulierung

Die Sprache L der vollständig geklammerten arithmetischen Ausdrücke ist *induktiv definiert* durch:

1. Jede Dezimalzahl liegt in L.
2. Jeder Bezeichner liegt in L.
3. Wenn $e \in L$, dann ist auch $(-e) \in L$.
4. Wenn $e_1, e_2 \in L$, dann ist auch $(e_1 + e_2) \in L$.
5. Wenn $e_1, e_2 \in L$, dann ist auch $(e_1 - e_2) \in L$.
6. Wenn $e_1, e_2 \in L$, dann ist auch $(e_1 * e_2) \in L$.

Was bedeutet '*induktiv definiert*'? Es bedeutet, dass L die *kleinste* Menge mit den Eigenschaften 1. bis 6. ist (eine andere Menge mit diesen Eigenschaften ist z.B. Σ^*).

Kontextfreie Sprachen

Die mathematische Formulierung genügt uns nicht (so wie uns die Mengenausdrücke bei den regulären Sprachen nicht genügten). Wir brauchen eine *formale Schreibweise* (wie die regulären Ausdrücke).

Definition 3.1 *Eine kontextfreie Grammatik (kurz: KFG) ist ein 4-Tupel $G = (\Sigma, N, S, P)$, wobei gilt:*

- *Σ und N sind Alphabete mit $\Sigma \cap N = \emptyset$. Die Zeichen aus Σ heißen Terminalzeichen, die aus N heißen Nichtterminalzeichen von G (engl.: Nonterminals).*
- *$S \in N$. S heißt Startzeichen von G.*
- *$P \subseteq N \times (\Sigma \cup N)^*$. Die Elemente von P heißen Regeln oder Produktionen von G.*

Eine Produktion ist also ein *Paar* (A, u), wobei A ein Nichtterminalzeichen ist und u ein Wort, das sowohl Terminal- als auch Nichtterminalzeichen enthalten darf. Statt (A, u) schreibt man $A \to u$.

Konvention:
Als Nichtterminalzeichen benutzen wir Großbuchstaben.

Beispiel:

$G = (\Sigma, N, S, P)$ mit

- $\Sigma = \{0, 1, x, y, -, +, *, (,)\}$
- $N = \{E\}$
- $S = E$
- $P = \{E \to 0,\ E \to 1,\ E \to x,\ E \to y,\ E \to (-E),$
  $E \to (E + E),\ E \to (E - E),\ E \to (E * E)\}$

ist eine KFG für vollständig geklammerte arithmetische Ausdrücke (in denen nur die Zahlen $0, 1$ und nur die Bezeichner x, y vorkommen).

Kurzschreibweise:

Man schreibt

$$A \to u_1 \mid \ldots \mid u_n$$

als Abkürzung für eine *Menge* von Produktionen

$$A \to u_1, \ldots, A \to u_n$$

mit dem gleichen Nichtterminalzeichen auf der linken Seite.

Im Beispiel kann man also *alle* Produktionen zusammenfassen zu

$$E \to 0 \mid 1 \mid x \mid y \mid (-E) \mid (E + E) \mid (E - E) \mid (E * E)$$

wobei man "|" als "oder" liest.

Kontextfreie Sprachen

Wie ist eine KFG zu verstehen, d.h. welche Sprache beschreibt sie?

- Entweder als *Konstruktionsvorschrift* für eine Sprache:

  Dann dienen die Produktionen dazu, die Wörter der Sprache *abzuleiten* oder zu *erzeugen*.

- Oder als *induktive Definition* einer Sprache:

  Dann liest man die Produktionen als *Regeln*, die die Sprache erfüllen muss.

Beide Auffassungen sind äquivalent zueinander, üblich ist aber die Formulierung als Konstruktionsvorschrift.

Definition 3.2 *Sei $G = (\Sigma, N, S, P)$ eine KFG.*

Auf der Menge $(\Sigma \cup N)^*$ definieren wir eine Relation $\Rightarrow_G$ (oder kürzer: $\Rightarrow$) durch:

$$u \Rightarrow_G v \quad \Leftrightarrow \quad \text{es gibt eine Produktion } A \to y \text{ in } P$$
$$\text{und Wörter } x, z \in (\Sigma \cup N)^*$$
$$\text{so dass } u = xAz \text{ und } v = xyz$$

Man bezeichnet $u \Rightarrow_G v$ als einen Ableitungsschritt (in G).

In Worten: Ein Ableitungsschritt $u \Rightarrow_G v$ besteht darin, ein Vorkommen eines Nichtterminalzeichens A im Wort u durch die rechte Seite y einer Produktion $A \to y$ zu ersetzen.

Mit $\overset{n}{\Rightarrow}_G$ $(n \geq 0)$, $\overset{+}{\Rightarrow}_G$ und $\overset{*}{\Rightarrow}_G$ bezeichnen wir wieder die n-te Potenz, den transitiven Abschluss und den reflexiven, transitiven Abschluss der Relation $\Rightarrow_G$.

Definition 3.3 *Sei G eine KFG.*

1. *v heißt ableitbar aus u (in G), wenn $u \overset{*}{\Rightarrow}_G v$, d.h. wenn eine Folge $u = w_0 \Rightarrow_G \ldots \Rightarrow_G w_n = v$ $(n \geq 0)$ von Ableitungsschritten in G existiert.*

   Eine solche Folge bezeichnet man als Ableitung von v aus u.

2. *Die von G erzeugte Sprache $L(G)$ ist definiert durch*

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

   $L(G)$ besteht also aus allen Wörtern über dem Terminalalphabet Σ, die aus dem Startzeichen S ableitbar sind.

Definition 3.4 *Eine Sprache $L \subseteq \Sigma^*$ heißt* kontextfrei, *wenn es eine kontextfreie Grammatik G gibt mit $L = L(G)$.*

Beispiel:

Sei G die oben definierte KFG für arithmetische Ausdrücke.

Dann gilt z.B.
$E \Rightarrow (E+E) \Rightarrow ((E*E)+E) \Rightarrow ((x*E)+E) \Rightarrow ((x*y)+E) \Rightarrow ((x*y)+1)$
also $((x * y) + 1) \in L(G)$.

Die Sprache $L(G)$ ist (per Definition) kontextfrei.

Aber ist $L(G)$ die gewünschte Sprache L der vollständig geklammerten arithmetischen Ausdrücke?
Um diese Frage zu beantworten, bräuchte man eine Definition für L, die unabhängig von der Grammatik ist. Die ist schwer zu finden!
Also stellen wir uns auf den Standpunkt, dass L durch die Grammatik G *definiert* ist. Dann ist nichts mehr zu beweisen.

Kontextfreie Sprachen

Sei G wie oben.

Neben der bereits genannten Ableitung
$E \Rightarrow (E{+}E) \Rightarrow ((E{*}E){+}E) \Rightarrow ((x{*}E){+}E) \Rightarrow ((x{*}y){+}E) \Rightarrow ((x{*}y){+}1)$
gibt es andere Ableitungen für das gleiche Wort, z.B.
$E \Rightarrow (E+E) \Rightarrow (E+1) \Rightarrow ((E*E)+1) \Rightarrow ((E*y)+1) \Rightarrow ((x*y)+1)$

In beiden Ableitungen werden die 'gleichen' Ableitungsschritte in unterschiedlicher Reihenfolge benutzt.

Die Reihenfolge der Ableitungsschritte spielt keine Rolle, weil die Ableitungsschritte in einer kontextfreien Grammatik sehr einfach aussehen: Es wird stets ein Nichtterminalzeichen A durch ein Wort y ersetzt. Der *Kontext* des Zeichens A, d.h. der Text vor und hinter A, spielt dabei keine Rolle. Er beeinflusst den Ableitungsschritt nicht, und er wird durch den Ableitungsschritt nicht verändert (daher die Bezeichnung '*kontextfrei*'). Also ist es unerheblich, ob man zuerst A oder zuerst ein Nichtterminalzeichen im Kontext von A ersetzt.

Diese Überlegungen kann man präzisieren,

- entweder indem man eine spezielle Reihenfolge für die Ableitungsschritte *vorschreibt*, und dann zeigt, dass jede Ableitung so umgeformt werden kann, dass sie der Vorschrift genügt,
- oder indem man eine Schreibweise benutzt, die von der Reihenfolge der Ableitungsschritte *abstrahiert*.

Definition 3.5 *Ein Ableitungsschritt* $u \Rightarrow v$ *heißt* Linksableitungsschritt, *wenn es Wörter* $x \in \Sigma^*$, $z \in (N \cup \Sigma)^*$ *und eine Produktion* $A \to y$ *in* G *gibt mit* $u = xAz$ *und* $v = xyz$. *Eine* Linksableitung *ist eine Ableitung, die nur aus Linksableitungsschritten besteht.*

Eine Linksableitung ist also eine Ableitung, bei der stets das erste, d.h. das am weitesten links stehende Nichtterminalzeichen ersetzt wird.

Ein Beispiel für eine Linksableitung haben wir schon gesehen:

$E \Rightarrow (E+E) \Rightarrow ((E*E)+E) \Rightarrow ((x*E)+E) \Rightarrow ((x*y)+E) \Rightarrow ((x*y)+1)$

Das folgende Lemma besagt, dass Linksableitungen ausreichen.

Lemma 3.6 *$u \stackrel{*}{\Rightarrow}_G v$ gilt genau dann, wenn es eine Linksableitung von v aus u in G gibt.*

Beweisidee:

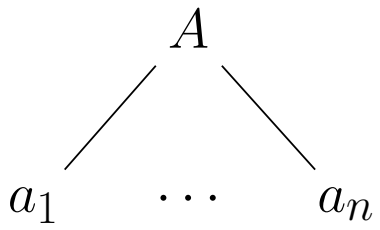
Man kann die Schritte einer beliebigen Ableitung so umordnen, dass eine Linksableitung entsteht. Die Details sind mühsam! □

Man kann Lemma 3.6 so interpretieren:

Wenn wir beliebige Ableitungen betrachten, so gibt es (fast in jeder Grammatik) viele Ableitungen, die sich nur 'unwesentlich' unterscheiden, nämlich nur in der Reihenfolge der Ersetzungen. Betrachten wir nur Linksableitungen, so entfallen diese unwesentlichen Unterschiede, weil die Reihenfolge der Ableitungsschritte fest vorgeschrieben ist.

Ein alternativer Ansatz besteht darin, von der Reihenfolge der Ableitungsschritte zu abstrahieren, indem man *Ableitungsbäume* betrachtet.

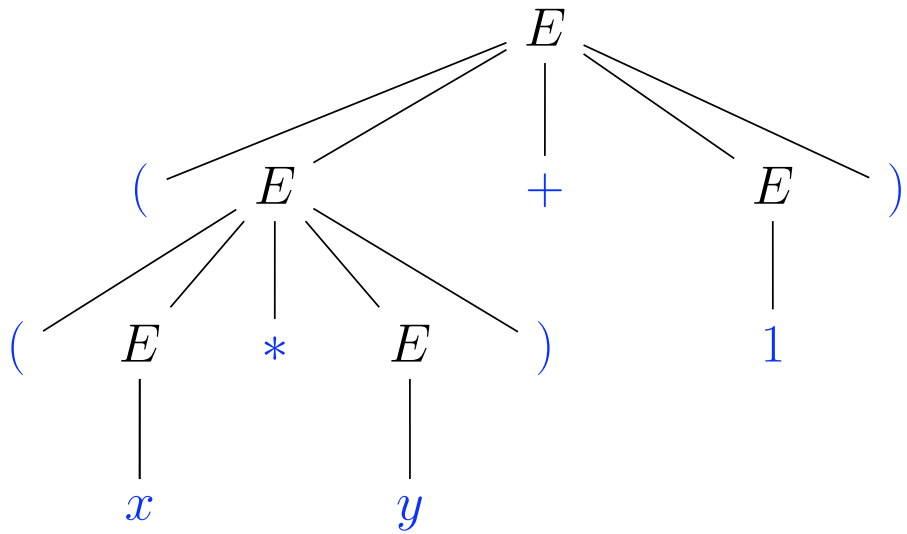
Definition 3.7 *Ein* Ableitungsbaum *(oder* Syntaxbaum*) für die Grammatik $G = (\Sigma, N, S, P)$ ist ein Baum, dessen Knoten mit Zeichen aus $\Sigma \cup N$ markiert sind, und zwar so, dass jeder innere Knoten die Form*



hat, wobei $A \to a_1 \dots a_n$ eine Produktion in P ist.

Beispiel:

Ein Ableitungsbaum für unsere Grammatik G ist



An diesem Baum sieht man, dass $((x * y) + 1)$ aus E ableitbar ist (ohne dass die Reihenfolge der Ableitungsschritte festgelegt wird).

Analog zu Linksableitungen definiert man

Definition 3.8 *Ein Ableitungsschritt $u \Rightarrow v$ heißt* Rechtsableitungsschritt, *wenn es Wörter $x \in (N \cup \Sigma)^*$, $z \in \Sigma^*$ und eine Produktion $A \to y$ in G gibt mit $u = xAz$ und $v = xyz$. Eine* Rechtsableitung *ist eine Ableitung, die nur aus Rechtsableitungsschritten besteht.*

In einem Rechtsableitungsschritt wird also stets das letzte, d.h. das am weitesten rechts stehende Nichtterminalzeichen ersetzt.

Definition 3.9 *Sei $G = (\Sigma, N, S, P)$ eine kontextfreie Grammatik. Ein Wort $u \in (N \cup \Sigma)^*$ heißt* Satzform *(*Linkssatzform*,* Rechtssatzform*) von G, wenn es eine Ableitung (Linksableitung, Rechtsableitung) von u aus dem Startzeichen S gibt. Ein Wort $w \in L(G)$ wird manchmal auch als* Satz *der Grammatik G bezeichnet.*

Lemma 3.10 *Sei $G = (\Sigma, N, S, P)$ eine kontextfreie Grammatik, sei $A \in N$ und $w \in \Sigma^*$. Dann gilt:*

- *Zu jeder Ableitung $A \stackrel{*}{\Rightarrow} w$ existiert ein 'entsprechender' Ableitungsbaum mit Wurzel A und Blattwort w.*
- *Zu jedem Ableitungsbaum mit Wurzel A und Blattwort w existiert genau eine 'entsprechende' Linksableitung $A \stackrel{*}{\Rightarrow} w$.*

Formal:
Für jedes $A \in N$ und jedes $w \in \Sigma^*$ gilt: Es existiert eine bijektive Funktion zwischen der Menge aller Linksableitungen $A \stackrel{*}{\Rightarrow} w$ und der Menge aller Ableitungsbäume mit Wurzel A und Blattwort w. Also ist die Anzahl der unterschiedlichen Linksableitungen $A \stackrel{*}{\Rightarrow} w$ gleich der Anzahl der unterschiedlichen Ableitungsbäume mit Wurzel A und Blattwort w.

All dies gilt auch für Rechtsableitungen anstelle von Linksableitungen.

In der Praxis (Compilerbau) ist es wichtig, dass jedes Wort eine eindeutige Struktur besitzt, denn ein Compiler soll ja nicht nur testen, ob eine eingegebene Zeichenreihe ein arithmetischer Ausdruck ist, sondern er soll diesen Ausdruck auch weiterverarbeiten, d.h. letzten Endes in Maschinencode übersetzen.

Definition 3.11

- *Eine kontextfreie Grammatik heißt* eindeutig, *wenn für jedes Wort* $w \in L(G)$ *genau ein Ableitungsbaum mit Wurzel* S *und Blattwort* w *existiert (oder äquivalent dazu: genau eine Linksableitung* $S \stackrel{*}{\Rightarrow} w$*). Andernfalls heißt sie* mehrdeutig.
- *Eine kontextfreie Sprache* L *heißt* inhärent mehrdeutig, *wenn* jede *kontextfreie Grammatik* G *mit* $L = L(G)$ *mehrdeutig ist.*

Beispiel:
Unsere Grammatik G für vollständig geklammerte arithmetische Ausdrücke ist eindeutig.

Intuitive Begründung:
Durch die vollständige Klammerung ist die Struktur jedes Ausdrucks eindeutig festgelegt.

Beweisskizze:
Man beweist zunächst (durch eine einfache Induktion über die Länge der Ableitung von e), dass

- jedes Wort $e \in L(G)$ genauso viele öffnende wie schließende Klammern hat,
- jedes echte nichtleere Präfix eines Wortes $e \in L(G)$ *mehr* öffnende als schließende Klammern hat.

Es folgt sofort, dass kein Wort $e \in L(G)$ echtes Präfix eines anderen Wortes $e' \in L(G)$ sein kann.

Damit kann man nun zeigen, dass jeder Ausdruck $e \in L(G)$ eine eindeutige Struktur hat: Nehmen wir z.B. an, dass ein Ausdruck $e \in L(G)$ sich sowohl in der Form $(e_1 + e_2)$ als auch in der Form $(e'_1 * e'_2)$ darstellen lässt (mit $e_1, e_2, e'_1, e'_2 \in L(G)$).

Dann ist einer der Ausdrücke e_1, e'_1 kürzer als der andere, und müsste deshalb echtes nichtleeres Präfix des anderen sein (weil die beiden Zeichenreihen (e_1 und (e'_1 Präfixe von e sind). Das ist aber nicht möglich. □

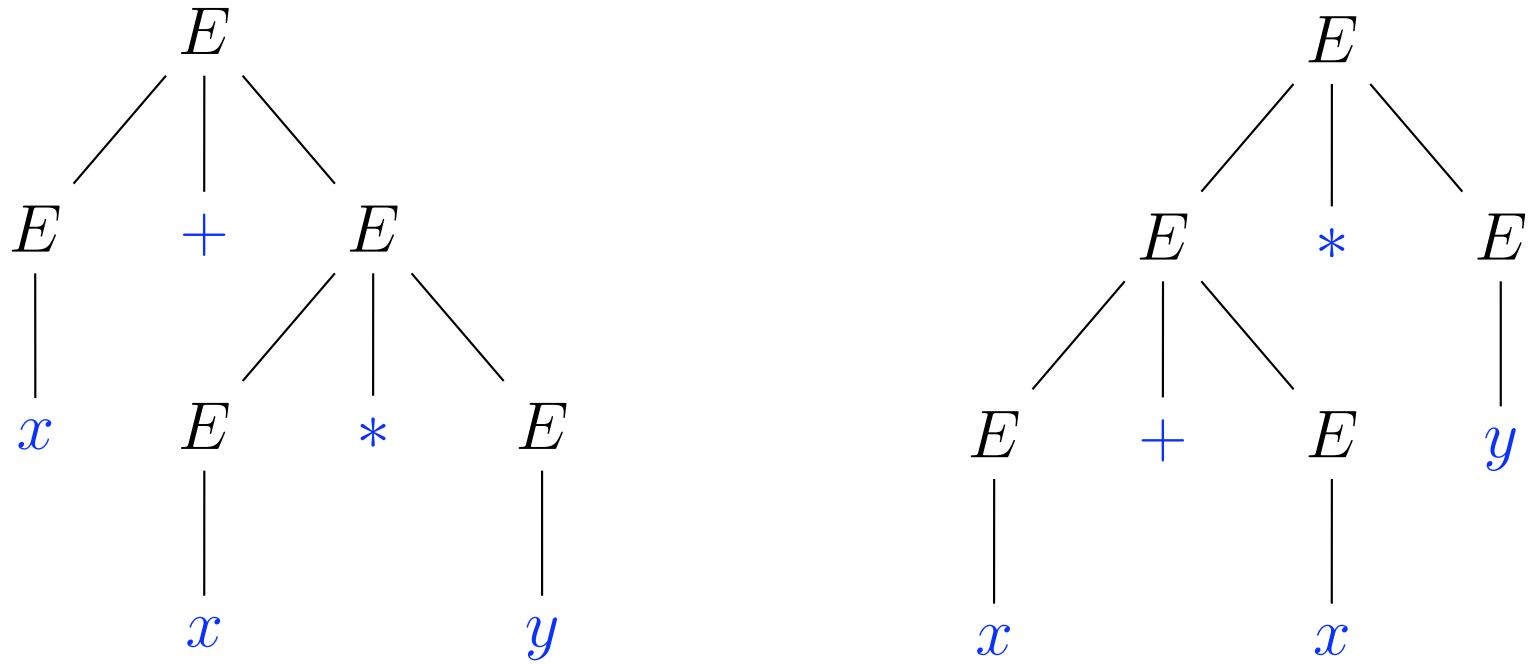
Beispiel für eine mehrdeutige Grammatik:

Sei $G_1 = (\Sigma, N, E, P)$ mit

- $\Sigma = \{0, 1, x, y, -, +, *, (,)\}$
- $N = \{E\}$
- $P = \{E \to 0,\ E \to 1,\ E \to x,\ E \to y,\ E \to -E,$
  $E \to E + E,\ E \to E - E,\ E \to E * E,\ E \to (E)\}$

Kontextfreie Sprachen

G_1 erzeugt die Sprache der unvollständig geklammerten arithmetischen Ausdrücke und ist (in hohem Maße) mehrdeutig,
z.B. hat das Wort $x + x * y$ die beiden folgenden Ableitungsbäume.



Gibt es eine eindeutige KFG für $L(G_1)$?

Sei $G_2 = (\Sigma, N, E, P)$ mit:

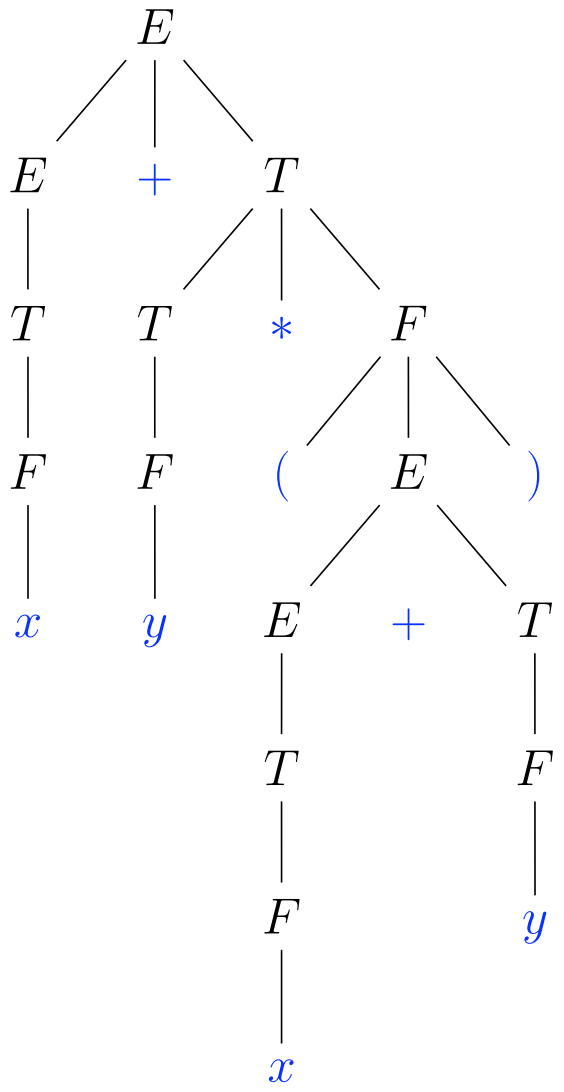
- $\Sigma = \{0, 1, x, y, -, +, *, (,)\}$
- $N = \{E, T, F\}$
- $P = \{$

$$
\begin{array}{lcll}
E & \to & E + T, & (1)\\
E & \to & E - T, & (2)\\
E & \to & T, & (3)\\
T & \to & T * F, & (4)\\
T & \to & F, & (5)\\
F & \to & (E), & (6)\\
F & \to & -F, & (7)\\
F & \to & 0 \mid 1 \mid x \mid y\} & (8)
\end{array}
$$

Die eindeutige Linksableitung für $E \stackrel{*}{\Rightarrow} x + y * (x + y)$:

$$
\begin{array}{lcll}
E & \Rightarrow & E + T & (1)\\
 & \Rightarrow & T + T & (3)\\
 & \Rightarrow & F + T & (5)\\
 & \Rightarrow & x + T & (8)\\
 & \Rightarrow & x + T * F & (4)\\
 & \Rightarrow & x + F * F & (5)\\
 & \Rightarrow & x + y * F & (8)\\
 & \Rightarrow & x + y * (E) & (6)\\
 & \Rightarrow & x + y * (E + T) & (1)\\
 & \vdots & & \\
 & \Rightarrow & x + y * (x + y) &
\end{array}
$$

und der zugehörige Ableitungsbaum



Jetzt wäre noch zu zeigen, dass G_2 eindeutig ist und dass $L(G_2) = L(G_1)$. Das beweisen wir nicht.

Man beachte, dass im Ableitungsbaum die üblichen Prioritäten der Operatoren zum Ausdruck kommen: $*$ bindet stärker als $+$, deshalb ist der Gesamtausdruck von der Form $E + T$. Etwas anderes lässt die Grammatik G_2 nicht zu, weil links von $*$ niemals ein $+$ stehen kann, das nicht durch Klammern 'geschützt' ist.

Wie beweist man, dass eine KFG die gewünschte Sprache erzeugt?

Beispiel:

Sei $L = \{a^n b^n \mid n \geq 0\}$
und $G = (\{a, b\}, \{S\}, S, P)$ mit $P = \{S \to aSb, S \to \varepsilon\}$

Behauptung: $L(G) = L$.

Beweis:

'$\subseteq$': Sei $w \in \Sigma^*$ mit $S \stackrel{*}{\Rightarrow} w$.

Es ist zu zeigen, dass $w \in L$.

Dazu beweist man eine 'passende Behauptung' durch *Induktion über die Länge der Ableitung*.

Für den Induktionsschritt hat man zwei Möglichkeiten: Entweder man spaltet den *ersten* Ableitungsschritt ab oder den *letzten*.

Will man den *letzten* Ableitungsschritt abspalten, so muss man eine Behauptung für alle $u \in (N \cup \Sigma)^*$ mit $S \stackrel{*}{\Rightarrow} u$ (also für alle Satzformen) aufstellen, z.B.:

$$\text{Wenn } u \notin \Sigma^*, \text{ dann ist } u = a^n S b^n \text{ für ein } n \geq 0. \qquad (*)$$

$S \stackrel{0}{\Rightarrow} u$:

Dann ist $u = S = a^0 S b^0$.

$S \stackrel{+}{\Rightarrow} u$, d.h. $S \stackrel{*}{\Rightarrow} v \Rightarrow u$ für ein $v \notin \Sigma^*$:

Dann ist $v = a^n S b^n$ nach Induktionsannahme. Wegen $u \notin \Sigma^*$ kann im Ableitungsschritt $v \Rightarrow u$ nur die Produktion $S \to aSb$ angewandt worden sein, also ist $u = a^{n+1} S b^{n+1}$.

Man kann die Argumentation noch verkürzen: Es genügt offensichtlich zu zeigen, dass (*) für S gilt und bei jedem Ableitungsschritt erhalten bleibt. (*) ist also eine *Invariante* für die Ableitungen aus S.

Aus (*) folgt schließlich $L(G) \subseteq L$:

Wenn $w \in L(G)$, dann gilt $S \stackrel{*}{\Rightarrow} u \Rightarrow w$ für ein $u \notin \Sigma^*$, also $u = a^n S b^n$ für ein $n \geq 0$. Wegen $w \in \Sigma^*$ kann $u \Rightarrow w$ nur mit der Produktion $S \to \varepsilon$ erfolgen, also ist $w = a^n b^n \in L$.

Die Abspaltung des *ersten* Ableitungsschrittes ist in unserem Falle wesentlich einfacher.

Man stellt eine Behauptung für alle $w \in \Sigma^*$ mit $S \stackrel{*}{\Rightarrow} w$ auf, nämlich:

$$w \in L \qquad (**)$$

$S \stackrel{1}{\Rightarrow} w$:

Dann ist $w = \varepsilon \in L$.

$S \Rightarrow aSb \stackrel{*}{\Rightarrow} w$:

Dann ist $w = avb$ mit $S \stackrel{*}{\Rightarrow} v$, also gilt nach Induktionsannahme $v \in L$, d.h. $v = a^n b^n$ für ein $n \geq 0$ und damit $w = a^{n+1} b^{n+1}$.

Geht es immer so einfach?

Im allgemeinen muss man für *jedes* $A \in N$ eine Behauptung über alle $w \in \Sigma^*$ mit $A \stackrel{*}{\Rightarrow} w$ aufstellen, und diese Behauptungen durch *simultane Induktion* beweisen.

Mit anderen Worten: Wenn $|N| = k$, so muss man eine Behauptung über die k Sprachen $L_G(A) = \{w \in \Sigma^* \mid A \stackrel{*}{\Rightarrow} w\}$ aufstellen und durch simultane Induktion beweisen.

'$\supseteq$': Sei $w \in L = \{a^n b^n \mid n \geq 0\}$.

Durch Induktion über $|w|$ beweist man $w \in L(G)$.

$|w| = 0$, d.h. $w = \varepsilon$:

Dann gilt $S \Rightarrow w$ mit Produktion $S \to \varepsilon$.

$|w| > 0$, d.h. $w = a^n b^n$ mit $n > 0$:

Dann ist $w = aa^{n-1}b^{n-1}b$, also $w = avb$ mit $v \in L$. Nach Induktionsannahme gilt $S \stackrel{*}{\Rightarrow} v$, also $S \Rightarrow aSb \stackrel{*}{\Rightarrow} avb = w$.

Auch diese Richtung wird schwieriger, wenn man mehrere Nichtterminalzeichen hat.

Wenn $N = \{A_1, \ldots, A_k\}$, so definiert man k Sprachen $L_1, \ldots, L_k \subseteq \Sigma^*$ und zeigt, dass $L_i \subseteq L_G(A_i)$.

Wir haben im Beispiel die folgenden Eigenschaften von Ableitungen benutzt:

Lemma 3.12

1. *Für alle $A \in N$ und $u, v, w \in (N \cup \Sigma)^*$ gilt:*

   Wenn $A \stackrel{*}{\Rightarrow} v$, dann $uAw \stackrel{*}{\Rightarrow} uvw$.

2. *Für alle $A \in N$ und $u, w, x \in \Sigma^*$ gilt:*

   Wenn $uAw \stackrel{*}{\Rightarrow} x$, dann existiert ein $v \in \Sigma^*$ mit $A \stackrel{*}{\Rightarrow} v$ und $x = uvw$.

Weiteres Beispiel:

Sei $L = \{w \in \{a,b\}^* \mid \#_a(w) = \#_b(w)\}$.

Es gibt eine einfache Grammatik, die L erzeugt, nämlich:

$G = (\{a,b\}, \{S\}, S, P)$ mit $P = \{S \to SS, S \to aSb, S \to bSa, S \to \varepsilon\}$

Behauptung: $L(G) = L$.

Beweis:

'$\subseteq$': Für jedes $u \in (N \cup \Sigma)^*$ mit $S \overset{*}{\Rightarrow} u$ gilt:

$$\#_a(u) = \#_b(u) \qquad (*)$$

denn (*) gilt für S und bleibt bei jedem Ableitungsschritt erhalten (weil in jeder Produktion gleich viele as und bs hinzukommen).

Also gilt (*) insbesondere für jedes $w \in L(G)$, d.h. $L(G) \subseteq L$.

'$\supseteq$': Sei $w \in L$, $w = a_1 \dots a_n$.

Wenn $w = \varepsilon$, so gilt $S \Rightarrow w$ mit Produktion $S \to \varepsilon$.

Es bleibt der Fall $|w| > 0$ zu betrachten.

Sei $f : \{a, b\}^* \to \mathbb{Z}$ mit $f(v) = \#_a(v) - \#_b(v)$.

Dann gilt $L = \{v \in \{a, b\}^* \mid f(v) = 0\}$.

Wir betrachten den 'Verlauf' der Funktion f für die Präfixe von w, d.h. für die Wörter $w_i = a_1 \dots a_i$ mit $0 \leq i \leq n$.

Es gilt $f(w_0) = f(\varepsilon) = 0$ und $f(w_n) = f(w) = 0$, weil $w \in L$.

1. Fall: $f(w_i) \geq 0$ für $i = 0, \dots, n$

Dann ist $f(w_1) = 1$ und $f(w_{n-1}) = 1$, also $w = avb$ für ein $v \in L$.

Nach Induktionsannahme gilt $S \stackrel{*}{\Rightarrow} v$, also $S \Rightarrow aSb \stackrel{*}{\Rightarrow} avb = w$.

2. Fall: $f(w_i) \leq 0$ für $i = 0, \ldots, n$

   Analog zum 1. Fall folgt $w = bva$ für ein $v \in L$,

   also gilt $S \Rightarrow bSa \stackrel{*}{\Rightarrow} bva = w$.

3. Fall: Es existieren $i, j \in \{0, \ldots, n\}$ mit $f(w_i) > 0$ und $f(w_j) < 0$.

   Dann existiert eine Zahl k zwischen i und j mit $f(w_k) = 0$, also $w_k = a_1 \ldots a_k \in L$.

   Dann ist auch $w'_k = a_{k+1} \ldots a_n \in L$ und es gilt $w = w_k w'_k$.

   Nach Induktionsannahme gilt $S \stackrel{*}{\Rightarrow} w_k$ und $S \stackrel{*}{\Rightarrow} w'_k$, also $S \Rightarrow SS \stackrel{*}{\Rightarrow} w_k S \stackrel{*}{\Rightarrow} w_k w'_k = w$. □

Die Beispiele zeigen, dass es kontextfreie Sprachen gibt, die *nicht* regulär sind.

Wir beweisen jetzt: Jede reguläre Sprache ist kontextfrei.

Um reguläre Sprachen zu erzeugen, reichen sehr spezielle Grammatiken aus.

Definition 3.13 *Eine Grammatik heißt* rechtslinear*, wenn jede Produktion von der Form $A \to aB$ oder $A \to \varepsilon$ (mit $A, B \in N$ und $a \in \Sigma$) ist. Sie heißt* linkslinear*, wenn jede Produktion von der Form $A \to Ba$ oder $A \to \varepsilon$ ist.*

In der Literatur sind oft noch Produktionen anderer Art in rechtslinearen Grammatiken zugelassen, etwa $A \to a$, $A \to B$ oder $A \to wB$ mit $w \in \Sigma^*$. Analog für linkslineare Grammatiken. Dadurch werden die Grammatiken nicht mächtiger.

Beispiel:

Sei $L = \{a^{2n} \mid n \geq 0\}$
und $G = (\{a\}, \{S_0, S_1\}, S_0, P)$ mit $P = \{S_0 \to aS_1, S_1 \to aS_0, S_0 \to \varepsilon\}$

Behauptung: $L(G) = L$.

Beweis:

'$\subseteq$': Jedes aus S_0 ableitbare Wort, das nicht in $\{a\}^*$ liegt, hat die Form $a^{2n}S_0$ oder $a^{2n+1}S_1$. (Induktion über die Länge der Ableitung).

Da jedes $w \in L(G)$ nur mit Produktion $S_0 \to \varepsilon$ aus einem solchen Wort entstehen kann, muss $w = a^{2n}$ sein für ein $n \geq 0$.

'$\supseteq$': Für jedes $n \geq 0$ gilt

$S_0 \Rightarrow aS_1 \Rightarrow aaS_0 \Rightarrow \ldots \Rightarrow a^{2n}S_0 \Rightarrow a^{2n}$ $\square$

Am Beispiel sieht man, dass die Nichtterminalzeichen einer rechtslinearen Grammatik eine ähnliche Rolle spielen wie die Zustände eines endlichen Automaten.

In jedem Ableitungsschritt (außer dem letzten) wird ein Terminalzeichen erzeugt und das Nichtterminalzeichen am Ende des Wortes wird eventuell verändert.

So kann man sich im Nichtterminalzeichen eine Information über die bereits erzeugten Terminalzeichen merken.

Im Beispiel: S_0 bedeutet, dass bisher eine gerade Anzahl von as erzeugt wurde, S_1 bedeutet, dass eine ungerade Anzahl von as erzeugt wurde.

Satz 3.14

1. *Zu jedem NDEA A kann man eine rechtslineare Grammatik G mit $L(G) = L(A)$ konstruieren.*
2. *Zu jeder rechtslinearen Grammatik G kann man einen NDEA A mit $L(A) = L(G)$ konstruieren.*

Beweis:

1. Sei $A = (\Sigma, Q, s, F, \Delta)$ ein NDEA.

   Wir dürfen annehmen, dass $\Sigma \cap Q = \emptyset$.

   Sei $G = (\Sigma, Q, s, P)$ mit

   $P = \{p \to aq \mid (p, a, q) \in \Delta\} \cup \{p \to \varepsilon \mid p \in F\}$

   Dann folgt durch eine einfache Induktion über $|w|$:

$$p \stackrel{*}{\Rightarrow}_G wq \;\Leftrightarrow\; (p, w) \vdash (q, \varepsilon) \qquad (*)$$

Also gilt

$$
\begin{aligned}
w \in L(G) \quad &\Leftrightarrow \quad s \stackrel{*}{\Rightarrow}_G w \\
&\Leftrightarrow \quad \text{es existiert ein } p \in Q \text{ mit } s \stackrel{*}{\Rightarrow}_G wp \Rightarrow w \\
&\qquad\quad \text{und } (p \to \varepsilon) \in P \\
&\qquad\quad \text{(weil der letzte Ableitungsschritt nur mit einer} \\
&\qquad\quad \varepsilon\text{-Produktion erfolgen kann)} \\
&\Leftrightarrow \quad \text{es existiert ein } p \in Q \text{ mit } (s, w) \vdash_A^* (p, \varepsilon) \\
&\qquad\quad \text{und } p \in F \\
&\qquad\quad \text{(wegen (*) und der Definition von } P) \\
&\Leftrightarrow \quad w \in L(A)
\end{aligned}
$$

und damit ist $L(G) = L(A)$ bewiesen.

2. Sei $G = (\Sigma, N, S, P)$ eine rechtslineare Grammatik.

   Wir definieren $A = (\Sigma, N, S, F, \Delta)$ mit

   $F = \{B \in N \mid (B \to \varepsilon) \in P\}$ und

   $\Delta = \{(B, a, C) \in N \times \Sigma \times N \mid (B \to aC) \in P\}$

   Dann folgt durch eine einfache Induktion über $|w|$:

$$B \stackrel{*}{\Rightarrow}_G wC \;\; \Leftrightarrow \;\; (B, w) \vdash_A^* (C, \varepsilon)$$

   und ähnlich wie oben ergibt sich $L(A) = L(G)$ □

Damit haben wir eine weitere Charakterisierung regulärer Sprachen.

Korollar 3.15 *Eine Sprache ist genau dann regulär, wenn sie sich von einer rechtslinearen Grammatik erzeugen lässt.*

Außerdem ist damit natürlich beweisen, dass jede reguläre Sprache kontextfrei ist, also gilt

Satz 3.16 $\mathcal{L}_{reg} \subseteq \mathcal{L}_{kf}$

wobei $\mathcal{L}_{kf}$ die Klasse der kontextfreien Sprachen bezeichnet.

Unsere Beispiele zeigen, dass diese Inklusion echt ist, wenn das Alphabet Σ mindestens zwei Elemente enthält. (Bei einelementigem Alphabet stimmen die beiden Sprachklassen überein.)

Definition 3.17 *Ein* Kellerautomat *(engl.* pushdown automaton *oder* PDA*) ist ein 6-Tupel* $M = (\Sigma, \Gamma, Q, s, F, \Delta)$ *mit:*

- Σ *ist ein Alphabet (das* Eingabealphabet*)*
- Γ *ist ein Alphabet (das* Kelleralphabet*)*
- Q *ist eine endliche Menge (von* Zuständen*)*
- $s \in Q$ *(der* Startzustand*)*
- $F \subseteq Q$ *(Menge der* Endzustände *oder* akzeptierenden Zustände*)*
- Δ *ist eine* endliche *Teilmenge von* $(Q \times \Sigma^* \times \Gamma^*) \times (Q \times \Gamma^*)$

Δ *heißt* Übergangsrelation *von* M*, die Elemente von* Δ *heißen* Übergänge *oder* Transitionen *von* M*.*

Man beachte: Ein Kellerautomat kann 'extrem nichtdeterministisch' sein, weil die Übergangsrelation Δ ähnlich wie die eines ε-NDEA 'spontane Zustandsübergänge' erlaubt, bei denen sowohl das Eingabewort als auch der Kellerinhalt ignoriert werden.

Intuition für die Übergänge:

$((p, u, \beta), (q, \gamma)) \in \Delta \subseteq (Q \times \Sigma^* \times \Gamma^*) \times (Q \times \Gamma^*)$ bedeutet:

Wenn der Kellerautomat M

- sich im Zustand p befindet,
- das aktuelle Eingabewort mit u beginnt
- und der aktuelle Kellerinhalt mit β,

dann darf er in einem Übergangsschritt

- u entfernen,
- in den Zustand q wechseln
- und β durch γ ersetzen.

Dabei wird der Kellerinhalt von oben nach unten gelesen, d.h. β besteht aus den ersten Zeichen, die *oben* im Keller stehen.

Spezialfälle:

- $u = \varepsilon$:

  Das aktuelle Eingabewort wird ignoriert und bleibt unverändert.

- $\beta = \varepsilon$:

  Der aktuelle Kellerinhalt wird ignoriert und mit γ aufgefüllt.

- $\gamma = \varepsilon$:

  β wird durch ε ersetzt, d.h. vom Keller entfernt.

- $\beta = \gamma$:

  Der aktuelle Kellerinhalt bleibt unverändert.

- $\beta = \alpha\gamma$:

  $\alpha\gamma$ wird durch γ ersetzt, d.h. α wird vom Keller entfernt.

- $\gamma = \alpha\beta$:

  β wird durch $\alpha\beta$ ersetzt, d.h. der aktuelle Kellerinhalt wird mit α aufgefüllt.

Definition 3.18 *Sei $M = (\Sigma, \Gamma, Q, s, F, \Delta)$ ein Kellerautomat.*

1. *Eine Konfiguration von M ist ein Element $(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$.*

   Intuition: q ist der aktuelle Zustand, w das aktuelle Eingabewort und α der aktuelle Kellerinhalt.

2. *Die Übergangsschrittrelation $\vdash_M$ (oder kurz: $\vdash$) auf der Menge der Konfigurationen ist wie folgt definiert:*

   Wenn $((p, u, \beta), (q, \gamma)) \in \Delta$, dann gilt für alle $v \in \Sigma^*$ und $\alpha \in \Gamma^*$:

   $$(p, uv, \beta\alpha) \vdash_M (q, v, \gamma\alpha)$$

   und das sind die einzigen Konfigurationen, die in Relation $\vdash_M$ stehen.

Ein Paar $(p, uv, \beta\alpha) \vdash_M (q, v, \gamma\alpha)$ heißt Übergangsschritt von M.

$\vdash_M^n$ $(n \geq 0,)$ $\vdash_M^+$ und $\vdash_M^*$ sind wie üblich definiert.

Beispiel:

Sei $M = (\Sigma, \Gamma, Q, s, F, \Delta)$ mit

- $\Sigma = \Gamma = \{a, b\}$
- $Q = \{s, f\}$
- $F = \{f\}$
- $\Delta = \{$ $((s, a, \varepsilon), (s, a))$, (1) a in den Keller verschieben
  $((s, \varepsilon, \varepsilon), (f, \varepsilon))$, (2) spontaner Zustandswechsel
  $((f, b, a), (f, \varepsilon))$ $\}$ (3) b gegen a aufheben

Dann gilt z.B.

$(s, aabb, \varepsilon) \vdash_M (s, abb, a)$ mit (1)
$\vdash_M (s, bb, aa)$ mit (1)
$\vdash_M (f, bb, aa)$ mit (2)
$\vdash_M (f, b, a)$ mit (3)
$\vdash_M (f, \varepsilon, \varepsilon)$ mit (3)

Definition 3.19 *Sei $M = (\Sigma, \Gamma, Q, s, F, \Delta)$ ein Kellerautomat.*

1. *M akzeptiert das Wort $w \in \Sigma^*$, wenn ein $q \in F$ existiert mit $(s, w, \varepsilon) \vdash^* (q, \varepsilon, \varepsilon)$.*
2. *$L(M) = \{w \in \Sigma^* \mid M \text{ akzeptiert } w\}$ heißt die von M akzeptierte oder erkannte Sprache.*

In Worten: M startet im Startzustand s mit dem Wort w auf dem Eingabeband und mit leerem Keller. w wird akzeptiert, wenn M einen Endzustand q erreichen kann, so dass sowohl das Eingabeband als auch der Keller leer sind.

Beispiel:
Der oben definierte Kellerautomat M akzeptiert das Wort $aabb$, weil $(s, aabb, \varepsilon) \vdash_M^* (f, \varepsilon, \varepsilon)$ und $f \in F$. Er 'errät' dabei den Zeitpunkt, in dem er vom Zustand s in den Zustand f wechselt.

Behauptung: $L(M) = \{a^n b^n \mid n \geq 0\}$.

Beweis:

'$\supseteq$': Sei $n \geq 0$. Dann gilt (auch für $n = 0$):

$$\begin{aligned} (s, a^n b^n) &\vdash_M^n (s, b^n, a^n) && \text{mit (1)} \\ &\vdash_M (f, b^n, a^n) && \text{mit (2)} \\ &\vdash_M^n (f, \varepsilon, \varepsilon) && \text{mit (3)} \end{aligned}$$

'$\subseteq$': Sei $w \in L(M)$, d.h. $(s, w, \varepsilon) \vdash_M^* (f, \varepsilon, \varepsilon)$, da f einziger Endzustand ist. In $\vdash_M^*$ muss genau ein Schritt mit Transition (2) vorkommen, davor können nur (1)-Schritte stehen, dahinter nur (3)-Schritte. Also existieren $n \geq 0$, $v \in \Sigma^*$ mit $w = a^n v$ und

$$(s, w, \varepsilon) \vdash_{(1)}^* (s, v, a^n) \vdash_{(2)} (f, v, a^n) \vdash_{(3)}^* (f, \varepsilon, \varepsilon)$$

Das kann aber nur gelten, wenn $v = b^n$, also $w = a^n b^n$. □

Anmerkungen zur Literatur:

Sowohl die Definition des Kellerautomaten als auch die Definition des Akzeptierens variieren stark in der Literatur.

- Unsere Kellerautomaten sind sehr flexibel:

  Sowohl vom Eingabeband als auch vom Keller darf in jedem Schritt ein ganzes Wort (auch ε) gelesen werden. Oft wird stattdessen

  $$\Delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (Q \times \Gamma^*)$$

  gefordert. Das setzt natürlich voraus, dass im Keller stets mindestens ein Zeichen steht, d.h. man benötigt ein spezielles Startzeichen, das zu Beginn schon im Keller steht.

- Unsere Definition des Akzeptierens ist sehr streng:

  Der Kellerautomat muss im Endzustand sein *und* der Keller muss leer sein. Oft wird nur eins von beiden gefordert.

Man kann sich davon überzeugen, dass solche unterschiedlichen Definitionen zur gleichen Sprachklasse führen (s. Übung 7, Aufgabe 2).

Weiteres Beispiel:

Sei $L = \{w \in \{a,b\}^* \mid \#_a(w) = \#_b(w)\}$.

Sei $M = (\Sigma, \Gamma, Q, s, F, \Delta)$ mit

- $\Sigma = \Gamma = \{a, b\}$
- $Q = F = \{s\}$
- $\Delta = \{$ $((s,a,\varepsilon),(s,a))$, (1) a in den Keller verschieben
  $((s,b,\varepsilon),(s,b))$, (2) b in den Keller verschieben
  $((s,a,b),(s,\varepsilon))$, (3) a gegen b aufheben
  $((s,b,a),(s,\varepsilon))$ $\}$ (4) b gegen a aufheben

Behauptung: $L = L(M)$.

Beweis:

'$\subseteq$': Man *kann* die Übergangsschritte stets so wählen, dass im Keller der Überschuss an as bzw. bs des bisher gelesenen Wortes steht, d.h. wenn $w \in \Sigma^*$ mit $\#_a(w) = m$ und $\#_b(w) = n$, dann gilt:

$(s, w, \varepsilon) \vdash_M^* (s, \varepsilon, a^{m-n})$ falls $m \geq n$

$(s, w, \varepsilon) \vdash_M^* (s, \varepsilon, b^{n-m})$ falls $m \leq n$

Das zeigt man durch Induktion über $|w|$:

$w = \varepsilon$:

$(s, w, \varepsilon) \vdash_M^0 (s, \varepsilon, a^{0-0}) = (s, \varepsilon, b^{0-0})$

$w = va$:

Wenn $m > n$, dann ist $m - 1 \geq n$, also nach Induktionsannahme $(s, v, \varepsilon) \vdash_M^* (s, \varepsilon, a^{m-1-n})$ und damit $(s, w, \varepsilon) \vdash_M^* (s, a, a^{m-1-n}) \vdash_M (s, \varepsilon, a^{m-n})$. Wenn $m \leq n$, dann ist $m - 1 < n$, also nach Induktionsannahme $(s, v, \varepsilon) \vdash_M^* (s, \varepsilon, b^{n-(m-1)})$ und damit $(s, w, \varepsilon) \vdash_M^* (s, a, b^{n-(m-1)}) \vdash_M (s, \varepsilon, b^{n-m})$. Damit ist auch im Falle $m = n$ alles bewiesen, weil dann $b^{n-m} = \varepsilon = a^{m-n}$.

$w = vb$:

Analog zu $w = va$ (wegen der Symmetrie).

Damit ist '$\subseteq$' bewiesen, denn für $w \in L$ gilt $m = n$, also $(s, w, \varepsilon) \vdash_M^* (s, \varepsilon, a^0) = (s, \varepsilon, \varepsilon)$ und das bedeutet $w \in L(M)$.

'$\supseteq$': Sei $w \in L(M)$, d.h. $(s, w, \varepsilon) \vdash_M^* (s, \varepsilon, \varepsilon)$.

Für jeden Übergangsschritt von M gilt: Entweder es wird nur ein Zeichen verschoben (vom Eingabeband in den Keller), oder es wird gleichzeitig ein a und ein b entfernt (eines vom Eingabeband, das andere vom Keller).

Da in der letzten Konfiguration $(s, \varepsilon, \varepsilon)$ gleich viele as und bs vorhanden sind (Eingabewort und Kellerwort zusammengerechnet), muss dies auch für jede vorhergehende Konfiguration gelten, also insbesondere für die Anfangskonfiguration (s, w, ε).

Das bedeutet aber $\#_a(w) = \#_b(w)$, d.h. $w \in L$. □

Im Beweis wurde die Tatsache benutzt, dass eine Verlängerung des Eingabewortes vom Kellerautomaten ignoriert werden kann. Gleiches gilt für das Kellerwort.

Lemma 3.20 *Wenn*

$$(p, u, \alpha) \vdash_M^* (q, v, \beta)$$

dann gilt auch

$$(p, uw, \alpha\gamma) \vdash_M^* (q, vw, \beta\gamma)$$

für alle $v \in \Sigma^*$ *und* $\gamma \in \Gamma^*$.

Beweisidee:

Mit $(p, uv, \alpha\gamma)$ kann man die gleiche Folge von Übergangsschritten durchführen wie mit (p, u, α), indem man v und γ ignoriert.

(Die Tatsache, dass man mit $(p, uv, \alpha\gamma)$ vielleicht auch neue Übergangsschritte durchführen kann, indem man γ benutzt, interessiert hier nicht.) □

Satz 3.21 *Zu jeder kontextfreien Grammatik G lässt sich ein Kellerautomat M mit $L(M) = L(G)$ konstruieren.*

Beweisidee:

Man konstruiert den Kellerautomaten so, dass er eine Ableitung für das Eingabewort w in der Grammatik G 'erraten' kann.

Diese Ableitung führt er im Keller durch, d.h. er schreibt zu Beginn das Startzeichen in den Keller und führt dann die Ableitungsschritte der Grammatik im Keller aus.

Problem:

In einem Ableitungsschritt der Grammatik wird ein Nichtterminalzeichen A an einer *beliebigen* Stelle des bisher abgeleiteten Wortes durch die rechte Seite γ einer Produktion $A \to \gamma$ ersetzt.

Das kann der Kellerautomat nicht leisten, da er immer nur auf die Zeichen zugreifen kann, die *oben* im Keller stehen.

Lösung(?):

Wir betrachten nur *Linksableitungen*, bei denen ja immer das linkeste Nichtterminalzeichen ersetzt wird.

Aber auch das muss nicht ganz oben im Keller stehen, es können noch Terminalzeichen im Wege sein.

Die müssen aber, wenn die bisherige Ableitung richtig erraten wurde, mit den ersten Zeichen des Eingabewortes w übereinstimmen.

Also kann man sie in diesem Fall entfernen (und der andere Fall interessiert nicht, weil dann die bisherige Ableitung schon falsch erraten wurde).

Damit ist klar, dass der Kellerautomat drei Arten von Transitionen besitzen sollte, nämlich:

1. eine Transition, um das Startzeichen in den Keller zu legen,
2. für jede Produktion der Grammatik eine Transition, die den entsprechenden Ableitungsschritt durchführt,
3. Transitionen, um Terminalzeichen auf dem Eingabeband und im Keller miteinander zu vergleichen und zu entfernen.

Formal:

Sei $G = (\Sigma, N, S, P)$. Wir definieren $M = (\Sigma, \Gamma, Q, s, F, \Delta)$ mit

- $\Gamma = N \cup \Sigma$
- $Q = \{s, f\}$
- $F = \{f\}$
- $$\begin{aligned} \Delta &= \{((s, \varepsilon, \varepsilon), (f, S))\} && (1) \\ &\cup \{((f, \varepsilon, A), (f, \gamma)) \mid (A \to \gamma) \in P\} && (2) \\ &\cup \{((f, a, a), (f, \varepsilon)) \mid a \in \Sigma\} && (3) \end{aligned}$$

Bevor wir $L(M) = L(G)$ beweisen, wollen wir uns die Arbeitsweise dieses Kellerautomaten M an einem kleinen **Beispiel** verdeutlichen:

Sei $G = (\{a, b\}, \{S\}, S, P\}$ mit $P = \{S \to aSb, S \to \varepsilon\}$.

Dann ist $M = (\{a, b\}, \{a, b, S\}, \{s, f\}, s, \Delta)$

mit

$$\begin{aligned}\Delta \ = \{ \ & ((s, \varepsilon, \varepsilon), (f, S)), \\ & (((f, \varepsilon, S), (f, aSb)), \\ & (((f, \varepsilon, S), (f, \varepsilon), \\ & ((f, a, a), (f, \varepsilon)), \\ & ((f, b, b), (f, \varepsilon))\}\end{aligned}$$

also gilt z.B. bei Eingabe $aabb$:

$$\begin{aligned}(s, aabb, \varepsilon) &\vdash_M (f, aabb, S) \\ &\vdash_M (f, aabb, aSb) \\ &\vdash_M (f, abb, Sb) \\ &\vdash_M (f, abb, aSbb) \\ &\vdash_M (f, bb, Sbb) \\ &\vdash_M (f, bb, bb) \\ &\vdash_M (f, b, b) \\ &\vdash_M (f, \varepsilon, \varepsilon)\end{aligned}$$

Am Beispiel sieht man, dass in einer erfolgreichen Berechnung des Kellerautomaten das aktuelle Eingabewort stets aus dem aktuellen Kellerwort ableitbar ist. Diese Beobachtung bringen wir durch folgende Äquivalenz zum Ausdruck: Für alle $\alpha \in \Gamma^*$ und $w \in \Sigma^*$ gilt

$$(f, w, \alpha) \vdash_M^* (f, \varepsilon, \varepsilon) \;\Leftrightarrow\; \alpha \stackrel{*}{\Rightarrow}_G w \qquad (*)$$

Aus (*) folgt $L(M) = L(G)$, denn:

$$\begin{aligned} w \in L(M) &\Leftrightarrow (s, w, \varepsilon) \vdash_M^* (f, \varepsilon, \varepsilon) \\ &\Leftrightarrow (f, w, S) \vdash_M^* (f, \varepsilon, \varepsilon) \\ &\quad \text{weil als erster Übergangsschritt nur} \\ &\quad (s, w, \varepsilon) \vdash_M (f, w, S) \text{ in Frage kommt} \\ &\Leftrightarrow S \stackrel{*}{\Rightarrow}_G w \\ &\quad \text{wegen (*) mit } \alpha = S \\ &\Leftrightarrow w \in L(G) \end{aligned}$$

Es bleibt also (*) zu beweisen

$$(f, w, \alpha) \vdash_M^* (f, \varepsilon, \varepsilon) \;\Leftrightarrow\; \alpha \stackrel{*}{\Rightarrow}_G w \qquad (*)$$

'$\Rightarrow$': Induktion über die Anzahl n der Übergangsschritte in $\vdash_M^*$

$n = 0$, d.h. $w = \alpha = \varepsilon$:

Dann gilt $\alpha \stackrel{*}{\Rightarrow}_G w$.

$n > 0$, d.h. $(f, w, \alpha) \vdash_M (f, v, \beta) \vdash_M^{n-1} (f, \varepsilon, \varepsilon)$ mit $v \in \Sigma^*$ und $\beta \in \Gamma^*$

Dann gilt nach Induktionsannahme $\beta \stackrel{*}{\Rightarrow}_G v$.

Wenn der erste Übergangsschritt mit (3) erfolgt, so existiert ein $a \in \Sigma$ mit $w = av$ und $\alpha = a\beta$, also gilt $\alpha = a\beta \stackrel{*}{\Rightarrow}_G av = w$.

Wenn der erste Übergangsschritt mit (2) erfolgt, so gilt $w = v$ und β entsteht aus α durch Anwendung einer Produktion $(A \to \gamma) \in P$. Dann gilt $\alpha \Rightarrow_G \beta \stackrel{*}{\Rightarrow}_G v = w$.

$$(f, w, \alpha) \vdash_M^* (f, \varepsilon, \varepsilon) \;\Leftrightarrow\; \alpha \overset{*}{\Rightarrow}_G w \qquad (*)$$

'$\Leftarrow$': Induktion über die Länge n einer *Links*ableitung $\alpha \overset{n}{\Rightarrow}_G w$

$n = 0$, d.h. $\alpha = w \in \Sigma^*$:

Dann gilt $(f, w, \alpha) = (f, w, w) \vdash_M^{|w|} (f, \varepsilon, \varepsilon)$ mit (3).

$n > 0$, d.h. es existiert ein $\alpha_1 \in \Gamma^*$ mit $\alpha \Rightarrow_G \alpha_1 \overset{n-1}{\Rightarrow}_G w$

Sei $A \to \gamma$ die Produktion im Linksableitungsschritt $\alpha \Rightarrow_G \alpha_1$, d.h. es existieren $u \in \Sigma^*, \beta \in \Gamma^*$ mit $\alpha = uA\beta$ und $\alpha_1 = u\gamma\beta$. Wegen $u \in \Sigma^*$ existiert dann ein $v \in \Sigma^*$ mit $w = uv$ und $\gamma\beta \overset{n-1}{\Rightarrow}_G v$, also

$$\begin{aligned} (f, w, \alpha) = (f, uv, uA\beta) &\vdash_M^{|u|} (f, v, A\beta) \text{ mit (3)} \\ &\vdash_M (f, v, \gamma\beta) \text{ mit (2)} \\ &\vdash_M^* (f, \varepsilon, \varepsilon) \text{ nach Induktionsannahme } \square \end{aligned}$$

Die Umkehrung von Satz 3.21 gilt ebenfalls.

Satz 3.22 *Zu jedem Kellerautomaten M lässt sich eine kontextfreie Grammatik G konstruieren mit $L(G) = L(M)$.*

Beweis: s. Literatur □

Also erhalten wir eine zweite Charakterisierung für die Klasse der kontextfreien Sprachen.

Satz 3.23 *Eine Sprache $L \subseteq \Sigma^*$ ist genau dann kontextfrei, wenn es einen Kellerautomaten M gibt mit $L = L(M)$.*

Beweis: Das folgt unmittelbar aus den Sätzen 3.21 und 3.22. □

Abschlusseigenschaften

Satz 3.24 *Die Klasse $\mathcal{L}_{kf}$ der kontextfreien Sprachen ist abgeschlossen unter den Operationen $\cup, \circ, ^*$ und $^+$, d.h. wenn $L_1, L_2 \subseteq \Sigma^*$ kontextfrei sind, dann sind auch die Sprachen*

1. $L_1 \cup L_2$
2. $L_1 \circ L_2$
3. L_1^*
4. L_1^+

kontextfrei. Darüber hinaus gibt es Algorithmen, um Grammatiken für diese Sprachen aus den Grammatiken für L_1 und L_2 zu konstruieren.

Beweis:

Seien $G_i = (\Sigma, N_i, S_i, P_i)$ $(i = 1, 2)$ kontextfreie Grammatiken mit $L(G_i) = L_i$. Wir dürfen annehmen, dass $N_1 \cap N_2 = \emptyset$.

1. siehe Übungsblatt 5, Aufgabe 5
2. Sei $G = (\Sigma, N, S, P)$, wobei

   - S ein neues Zeichen ist, d.h. $S \notin N_1 \cup N_2 \cup \Sigma$,
   - $N = N_1 \cup N_2 \cup \{S\}$,
   - $P = \{S \to S_1 S_2\} \cup P_1 \cup P_2$.

   Dann gilt $L(G) = L_1 \circ L_2$.

   '$\supseteq$': Sei $w \in L_1 \circ L_2$, d.h. $w = w_1 w_2$ mit $w_i \in L_i$ für $i = 1, 2$.

   Dann gilt $S_i \overset{*}{\Rightarrow}_{G_i} w_i$ und damit auch $S_i \overset{*}{\Rightarrow}_{G} w_i$ für $i = 1, 2$, und es folgt $S \Rightarrow_G S_1 S_2 \overset{*}{\Rightarrow}_G w_1 S_2 \overset{*}{\Rightarrow}_G w_1 w_2 = w$, d.h. $w \in L(G)$.

'$\subseteq$': Sei $w \in L(G)$, d.h. es gibt eine Linksableitung $S \overset{*}{\Rightarrow}_G w$.

Da S ein neues Zeichen ist, kann der erste Ableitungsschritt nur mit der neuen Produktion $S \to S_1 S_2$ erfolgt sein, also hat die gesamte Ableitung die Form $S \Rightarrow S_1 S_2 \overset{*}{\Rightarrow}_G w_1 S_2 \overset{*}{\Rightarrow}_G w_1 w_2 = w$, wobei $S_1 \overset{*}{\Rightarrow}_G w_1$ und $S_2 \overset{*}{\Rightarrow}_G w_2$.

Wegen $N_1 \cap N_2 = \emptyset$ können in $S_1 \overset{*}{\Rightarrow}_G w_1$ nur Ableitungsschritte für G_1 vorkommen und in $S_2 \overset{*}{\Rightarrow}_G w_2$ nur solche für G_2.

Also gilt $S_1 \overset{*}{\Rightarrow}_{G_1} w_1$ und $S_2 \overset{*}{\Rightarrow}_{G_2} w_2$, d.h. $w_1 \in L_1, w_2 \in L_2$ und damit $w = w_1 w_2 \in L_1 \circ L_2$.

3. Sei $G = (\Sigma, N, S, P)$, wobei

   $S \notin N_1 \cup \Sigma$,

   $N = N_1 \cup \{S\}$,

   $P = \{S \to S_1 S, S \to \varepsilon\} \cup P_1$.

Dann gilt $L(G) = L_1^*$.

'$\supseteq$': Sei $w \in L_1^*$, d.h. es existieren $n \geq 0$ und $w_1, \ldots, w_n \in L_1$ mit $w = w_1 \ldots w_n$.

Dann gilt $S \Rightarrow S_1 S \Rightarrow \ldots \Rightarrow S_1^n S \Rightarrow S_1^n$, und wegen $S_1 \stackrel{*}{\Rightarrow}_G w_i$ für $i = 1, \ldots n$ folgt $S \stackrel{*}{\Rightarrow}_G w_1 \ldots w_n = w$, also $w \in L(G)$.

'$\subseteq$': Sei $w \in L(G)$. Durch Induktion über die Länge einer Linksableitung $S \stackrel{*}{\Rightarrow}_G w$ beweisen wir $w \in L_1^*$.

Da S ein neues Zeichen ist, kann der erste Ableitungsschritt von $S \stackrel{*}{\Rightarrow}_G w$ nur mit einer der neuen Produktionen $S \to \varepsilon$ oder $S \to S_1 S$ erfolgt sein. Im ersten Fall ist $w = \varepsilon \in L_1^*$. Im zweiten Fall ist die gesamte Ableitung von der Form $S \Rightarrow S_1 S \stackrel{*}{\Rightarrow}_G w_1 S \stackrel{*}{\Rightarrow}_G w_1 w' = w$, wobei $S_1 \stackrel{*}{\Rightarrow}_G w_1$ und $S \stackrel{*}{\Rightarrow}_G w'$.

Da $S_1 \stackrel{*}{\Rightarrow}_G w_1$ nur Ableitungsschritte für G_1 enthalten kann, gilt $w_1 \in L(G_1) = L_1$, und nach Induktionssannahme gilt $w' \in L_1^*$.

Also folgt $w = w_1 w' \in L_1 \circ L_1^* \subseteq L_1^*$.

4. Die Abgeschlossenheit unter $^+$ folgt aus der Abgeschlossenheit unter $\circ$ und *, weil $L_1^+ = L_1 \circ L_1^*$. $\square$

Es ist kein Zufall, dass Durchschnitt und Komplement in Satz 3.24 fehlen. Wir werden später sehen, dass der Durchschnitt zweier kontextfreier Sprachen im allgemeinen *nicht* kontextfrei ist.

Daraus folgt sofort, dass auch das Komplement einer kontextfreien Sprache im allgemeinen *nicht* kontextfrei ist, denn aus der Abgeschlossenheit unter Vereinigung und Komplement würde sich die Abgeschlossenheit unter Durchschnitt ergeben.

Für den Durchschnitt gilt eine *schwächere* Aussage, die sich besser mit Automaten (als mit Grammatiken) beweisen lässt:

Satz 3.25 *Wenn $L_1 \subseteq \Sigma^*$ kontextfrei und $L_2 \subseteq \Sigma^*$ regulär ist, dann ist $L_1 \cap L_2$ kontextfrei.*

Beweis:

Sei $M_1 = (\Sigma, \Gamma, Q_1, s_1, F_1, \Delta_1)$ ein Kellerautomat mit $L(M_1) = L_1$ und sei $A_2 = (\Sigma, Q_2, s_2, F_2, \delta)$ ein DEA mit $L(A_2) = L_2$.

Wir konstruieren einen Kellerautomaten M mit $L(M) = L_1 \cap L_2$.

Idee: M arbeitet wie M_1 und simuliert "parallel dazu" noch die Übergänge des endlichen Automaten A_2.

Diese "Parallelverarbeitung" bringt man dadurch zum Ausdruck, dass man auf der Zustandsmenge $Q_1 \times Q_2$ arbeitet: In der ersten Komponente merkt man sich den aktuellen Zustand von M_1, in der zweiten Komponente den von A_2.

Sei also $M = (\Sigma, \Gamma, Q_1 \times Q_2, (s_1, s_2), F_1 \times F_2, \Delta)$ mit

$$\Delta = \{(((p_1, p_2), u, \beta), ((q_1, q_2), \gamma)) \mid ((p_1, u, \beta), (q_1, \gamma)) \in \Delta_1 \text{ und } (p_2, u) \vdash_{A_2}^* (q_2, \varepsilon)\}$$

Δ ist gerade so konstruiert, dass in jedem Übergangsschritt von M ein Übergangsschritt von M_1 und eine entsprechende Folge von Übergangsschritten von A_2 simuliert werden, d.h. es gilt stets:

$$((p_1, p_2), u, \alpha) \vdash_M ((q_1, q_2), \varepsilon, \alpha') \Leftrightarrow (p_1, u, \alpha) \vdash_{M_1} (q_1, \varepsilon, \alpha') \text{ und } (p_2, u) \vdash_{A_2}^* (q_2, \varepsilon)$$

Das ergibt sich unmittelbar aus der Definition von Δ und der Definition der möglichen Übergangsschritte eines Kellerautomaten.

Der gleiche Zusammenhang gilt dann auch für *Folgen* von Übergangsschritten:

$$((p_1,p_2),u,\alpha) \vdash_M^* ((q_1,q_2),\varepsilon,\alpha') \;\Leftrightarrow\; (p_1,u,\alpha) \vdash_{M_1}^* (q_1,\varepsilon,\alpha') \text{ und } (p_2,u) \vdash_{A_2}^* (q_2,\varepsilon)$$

Das ergibt sich leicht durch Induktion über die Länge der Folge.

Daraus erhält man schließlich das gewünschte Ergebnis:

$w \in L(M)$ $\Leftrightarrow$ es existiert ein $(q_1,q_2) \in F$ mit

$((s_1,s_2),w,\varepsilon) \vdash_M^* ((q_1,q_2),\varepsilon,\varepsilon)$

$\Leftrightarrow$ es existieren $q_1 \in F_1, q_2 \in F_2$ mit

$(s_1,w,\varepsilon) \vdash_{M_1}^* (q_1,\varepsilon,\varepsilon)$

und $(s_2,w) \vdash_{A_2}^* (q_2,\varepsilon)$

$\Leftrightarrow$ $w \in L(M_1) \cap L(A_2) = L_1 \cap L_2$ $\square$

Entscheidbarkeitsfragen

Wir haben gesehen: Viele Fragestellungen über reguläre Sprachen sind entscheidbar, d.h. es gibt (einfache, manchmal auch effiziente) Algorithmen, die stets die richtige Antwort liefern.

Gilt das auch für kontextfreie Sprachen?

Wichtigste Fragestellung ist das *Wortproblem*:

1. Das *spezielle* Wortproblem für *eine* kontextfreie Grammatik G:

   Eingabe: Ein Wort $w \in \Sigma^*$.

   Frage: Ist $w \in L(G)$?

2. Das *allgemeine* Wortproblem für kontextfreie Grammatiken:

   Eingabe: Eine kontextfreie Grammatik G und ein Wort $w \in \Sigma^*$.

   Frage: Ist $w \in L(G)$?

Kontextfreie Sprachen

Für die Praxis ist das spezielle Wortproblem wichtig, denn das ist die Frage, die ein Parser für die Sprache $L(G)$ beantworten muss: Ist die Zeichenreihe, die der Programmierer eingibt, ein syntaktisch korrektes Programm? Diese Frage muss nicht nur entscheidbar sein, sondern sie muss sich *effizient* lösen lassen.

Wir zeigen hier, dass sogar das allgemeine Wortproblem entscheidbar ist (allerdings nicht sehr effizient).

Lösungsansatz:

Um $w \in L(G)$ zu überprüfen, sucht man systematisch nach einer Ableitung für w aus dem Startsymbol S. Man kann z.B. erst alle Wörter $u \in (N \cup \Sigma)^*$ bestimmen, die in einem Schritt aus S ableitbar sind, dann die, die in zwei Schritten ableitbar sind usw.

Wenn w dabei irgendwann auftaucht, ist die Antwort "ja". Aber wann kann man die Antwort "nein" geben, d.h. wann kann man sicher sein, dass w nicht mehr auftaucht?

Wenn wir wüssten, dass bei jedem Ableitungsschritt mindestens ein Zeichen hinzukommt, dann hätten wir ein Abbruchkriterium: Dann bräuchten wir nur Ableitungen zu untersuchen, die höchstens die Länge $|w|$ haben.

Probleme bereiten also Ableitungsschritte, bei denen das Wort *nicht* länger wird. Solche Ableitungsschritte entstehen, wenn man Produktionen $A \to \gamma$ mit $|\gamma| \leq 1$ anwendet.

Definition 3.26

- *Eine Produktion der Form $A \to \varepsilon$ (mit $A \in N$) heißt* ε-Produktion.
- *Eine Produktion der Form $A \to B$ (mit $A, B \in N$) heißt* Einheitsproduktion.

Wir werden zeigen, dass man solche Produktionen (bis auf eine) aus einer kontextfreien Grammatik entfernen kann, ohne dass sich die von der Grammatik erzeugte Sprache verändert.

Satz 3.27 *Zu jeder kontextfreien Grammatik $G = (\Sigma, N, S, P)$ kann man eine äquivalente kontextfreie Grammatik $G' = (\Sigma, N, S, P')$ mit folgenden Eigenschaften konstruieren:*

1. *P' enthält höchstens eine ε-Produktion, nämlich $S \to \varepsilon$,*
2. *diese ε-Produktion wird nur zur Ableitung von ε benötigt.*

Beweis:

Die Konstruktion von G' lässt sich in zwei Phasen unterteilen.

In der *ersten Phase* wird P schrittweise zu einer neuen Produktionenmenge P'' *erweitert*, wobei jeder einzelne Erweiterungsschritt so aussieht:

Wenn bereits zwei Produktionen der Form $B \to \beta A \gamma$ und $A \to \varepsilon$ vorhanden sind (d.h. wenn sie in P liegen oder durch vorhergehende Erweiterungsschritte hinzugekommen sind), dann wird die Produktion $B \to \beta\gamma$ aufgenommen (die sogar eine ε-Produktion sein kann).

Diese Erweiterungsschritte führt man so lange durch, bis keine neuen Produktionen mehr entstehen.

Das Verfahren *terminiert*, weil die rechte Seite einer neuen Produktion stets kürzer ist als die rechte Seite einer der Produktionen in P. Damit kommen von vornherein nur endlich viele unterschiedliche rechte Seiten und damit auch nur endlich viele unterschiedliche neue Produktionen in Frage.

Das Verfahren ist auch *korrekt*, d.h. durch die neu hinzugenommenen Produktionen wird die erzeugte Sprache nicht größer:

Jede Anwendung einer neuen Produktion $B \to \beta\gamma$ entspricht nämlich einer Anwendung von $B \to \beta A\gamma$, gefolgt von einer Anwendung von $A \to \varepsilon$, d.h. man kann jede Anwendung einer Produktion in P'' letztendlich durch mehrere (aufeinanderfolgende) Anwendungen von Produktionen der ursprünglichen Menge P ersetzen.

In der *zweiten Phase* wird P'' zu P' *verkleinert*, indem man einfach alle ε-Produktionen außer $S \to \varepsilon$ aus P'' entfernt.

Es bleibt zu zeigen, dass die erzeugte Sprache durch das Entfernen dieser Produktionen nicht kleiner wird.

Dazu geben wir an, wie man eine Ableitung $S \stackrel{*}{\Rightarrow} w$ eines Wortes $w \in \Sigma^*$ in P'' schrittweise zu einer Ableitung in P' umformen kann.

Jeder einzelne Umformungsschritt sieht so aus:

Wenn im ersten Ableitungsschritt von $S \stackrel{*}{\Rightarrow} w$ eine ε-Produktion angewandt wird, so ist $w = \varepsilon$ und die Ableitung liegt bereits in P'.

Wenn in einem späteren Schritt von $S \stackrel{*}{\Rightarrow} w$ eine ε-Produktion $A \to \varepsilon$ angewandt wird, dann ist das Nichtterminalzeichen A irgendwann vorher durch eine Produktion $B \to \beta A \gamma$ entstanden.

Also kann man sich die Anwendung von $A \to \varepsilon$ ersparen, indem man in diesem früheren Schritt anstelle von $B \to \beta A \gamma$ gleich die Produktion $B \to \beta\gamma$ anwendet (die wir ja in P'' aufgenommen haben).

Den soeben geschilderten Umformungsschritt wiederholt man so oft wie möglich.

Das Verfahren terminiert, weil die Ableitung bei jedem Umformungsschritt um einen Ableitungsschritt kürzer wird (was ja nicht unendlich oft passieren kann).

Also erreicht man irgendwann eine Situation, in der keine Umformung mehr möglich ist.

Das bedeutet aber, dass die Ableitung entweder keine Anwendungen von ε-Produktionen mehr enthält oder nur noch aus dem Ableitungsschritt $S \Rightarrow \varepsilon$ besteht.

In beiden Fällen ist es dann eine Ableitung in P'. □

Satz 3.28 *Zu jeder kontextfreien Grammatik $G = (\Sigma, N, S, P)$ kann man eine äquivalente kontextfreie Grammatik $G' = (\Sigma, N, S, P')$ konstruieren, die keine Einheitsproduktionen enthält (und falls G durch das Verfahren in Satz 3.27 entstanden ist, kann man die Konstruktion so durchführen, dass keine neuen ε-Produktionen entstehen).*

Beweis:

Die Konstruktion von G' verläuft wieder in zwei Phasen.

In der *ersten Phase* wird P schrittweise zu P'' *erweitert*, wobei jeder einzelne Erweiterungsschritt so aussieht:

Wenn bereits zwei Produktionen der Form $A \to B$ und $B \to \gamma$ vorhanden sind, dann wird die Produktion $A \to \gamma$ aufgenommen (wobei man hier *keine* ε-Produktionen aufnimmt, falls G nach dem Verfahren in Satz 3.27 entstanden ist).

Diese Erweiterungsschritte führt man so lange durch, bis keine neuen Produktionen mehr entstehen.

Das Verfahren *terminiert*, weil sich die neuen Produktionen von den alten nur durch das Nichtterminalzeichen auf der linken Seite unterscheiden, also können aus jeder Produktion in P höchstens $|N| - 1$ neue entstehen.

Das Verfahren ist *korrekt* weil auch hier jede neue Produktion nur eine 'Abkürzung' für eine Folge von alten Produktionen ist.

In der *zweiten Phase* wird P'' zu P' *verkleinert*, indem man alle Einheitsproduktionen aus P'' entfernt.

Hier muss man sich wieder davon überzeugen, dass die erzeugte Sprache nicht kleiner wird.

Dazu zeigt man, dass sich die Einheitsproduktionen aus jeder Ableitung $S \stackrel{*}{\Rightarrow} w$ eines Wortes $w \in \Sigma^*$ entfernen lassen.

Sei $A \to B$ die *letzte* Einheitsproduktion, die in der Ableitung $S \stackrel{*}{\Rightarrow} w$ angewandt wird.

Weil das entstehende Nichtterminalzeichen B nicht in w vorkommen kann, muss weiter rechts in der Ableitung eine Produktion $B \to \gamma$ auf *dieses* Zeichen B angewandt werden (dabei ist $B \to \gamma$ *keine* ε-Produktion, falls G durch das Verfahren in Satz 3.27 entstanden ist).

Also kann man diese Anwendung von $B \to \gamma$ einsparen, indem man gleich $A \to \gamma$ anstelle von $A \to B$ anwendet.

Man beachte, dass dabei eine Einheitsproduktion aus der Ableitung verschwindet: Per Definition ist $A \to B$ ja die *letzte* Einheitsproduktion in der Ableitung, also ist $B \to \gamma$ und damit auch das neu entstehende $A \to \gamma$ *keine* Einheitsproduktion.

Durch Wiederholung dieses Umformungsschrittes kann man also nach und nach alle Anwendungen von Einheitsproduktionen aus der Ableitung $S \stackrel{*}{\Rightarrow} w$ entfernen, und erhält so eine Ableitung mit der neuen Produktionenmenge P'. $\square$

Satz 3.29 *Für jede der folgenden Fragestellungen gibt es einen Entscheidungsalgorithmus.*

1. Allgemeines Wortproblem für kontextfreie Grammatiken

   Eingabe: Eine kontextfreie Grammatik G und ein Wort w.

   Frage: Ist $w \in L(G)$?

2. Leerheitsproblem für kontextfreie Grammatiken

   Eingabe: Eine kontextfreie Grammatik G.

   Frage: Ist $L(G) = \emptyset$?.

Beweis:

1. Wegen Satz 3.27 und Satz 3.28 dürfen wir annehmen, dass G keine Einheitsproduktionen enthält und höchstens eine ε-Produktion $S \to \varepsilon$, die nur zur Ableitung von ε benötigt wird.

Im Falle $w = \varepsilon$ brauchen wir nur nachzusehen, ob $S \to \varepsilon$ in der Produktionenmenge liegt.

Im Falle $w \neq \varepsilon$ können wir die Länge einer Ableitung $S \stackrel{*}{\Rightarrow} w$ nach oben abschätzen.

In $S \stackrel{*}{\Rightarrow} w$ werden ja nur Produktionen der Form $A \to \gamma$ mit $|\gamma| > 1$ oder $A \to a$ mit $a \in \Sigma$ angewandt.

Jeder Ableitungsschritt mit einer Produktion der ersten Form macht das bereits abgeleitete Wort länger, also können höchstens $|w| - |S| = |w| - 1$ solche Ableitungsschritte in $S \stackrel{*}{\Rightarrow} w$ vorkommen.

Jeder Ableitungsschritt mit einer Produktion $A \to a$ produziert ein Terminalzeichen, das nicht mehr entfernt werden kann, also können höchstens $|w|$ solche Ableitungsschritte vorkommen.

Damit beträgt die Gesamtlänge der Ableitung $S \stackrel{*}{\Rightarrow} w$ höchstens $2|w| - 1$.

Um zu testen, ob w in $L(G)$ liegt, brauchen wir also nur alle Ableitungen aus S bis zur Länge $2|w| - 1$ zu betrachten und nachzusehen, ob eine von ihnen das Wort w liefert.

2. Um zu testen, ob $L(G) = \emptyset$ ist, müssen wir systematisch nach einer Ableitung eines *beliebigen* Wortes $z \in \Sigma^*$ suchen.

   Auch hier stellt sich die Frage, ob man die Suche irgendwann abbrechen kann, d.h. ob man irgendwann sicher sein kann, dass kein Wort mehr gefunden wird.

   Wir zeigen dazu:

   > Wenn $G = (\Sigma, N, S, P)$ und $L(G) \neq \emptyset$, dann existiert ein Ableitungsbaum für ein Wort $z \in L(G)$ der höchstens die Höhe $|N|$ hat. (*)

   Um zu testen, ob $L(G) = \emptyset$ ist, brauchen wir also nur alle Ableitungsbäume (mit Wurzel S) bis zur Höhe $|N|$ zu erzeugen und nachzusehen, ob einer von ihnen ein Blattwort $z \in \Sigma^*$ hat.

Es bleibt (*) zu zeigen.

> Wenn $G = (\Sigma, N, S, P)$ und $L(G) \neq \emptyset$, dann existiert ein Ableitungsbaum für ein Wort $z \in L(G)$ der höchstens die Höhe $|N|$ hat. (*)

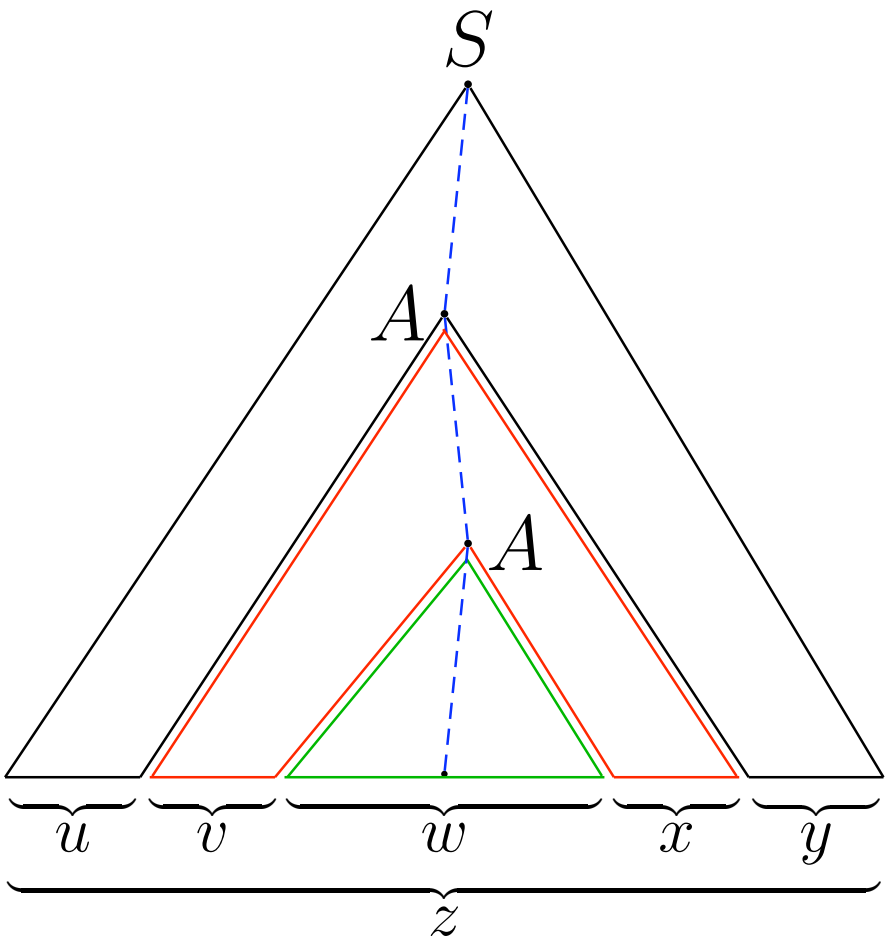
Wenn $L(G) \neq \emptyset$, so existiert zunächst *irgendein* Ableitungsbaum für ein Wort $z \in L(G)$.

Wenn dieser Ableitungsbaum noch nicht die gewünschte Eigenschaft hat, d.h. wenn seine Höhe größer als $|N|$ ist, so existiert ein Pfad von der Wurzel S zu einem Blatt, dessen Länge größer als $|N|$ ist.

Dieser Pfad enthält also mindestens $|N| + 2$ Knoten, von denen nur der letzte (das Blatt) mit einem Terminalzeichen markiert ist.

Also sind mindestens $|N| + 1$ Knoten mit Nichtterminalzeichen markiert, d.h. mindestens ein Nichtterminalzeichen A muss mehrmals auf diesem Pfad vorkommen.

Deshalb sieht der Ableitungsbaum für $S \stackrel{*}{\Rightarrow} z$ so aus:



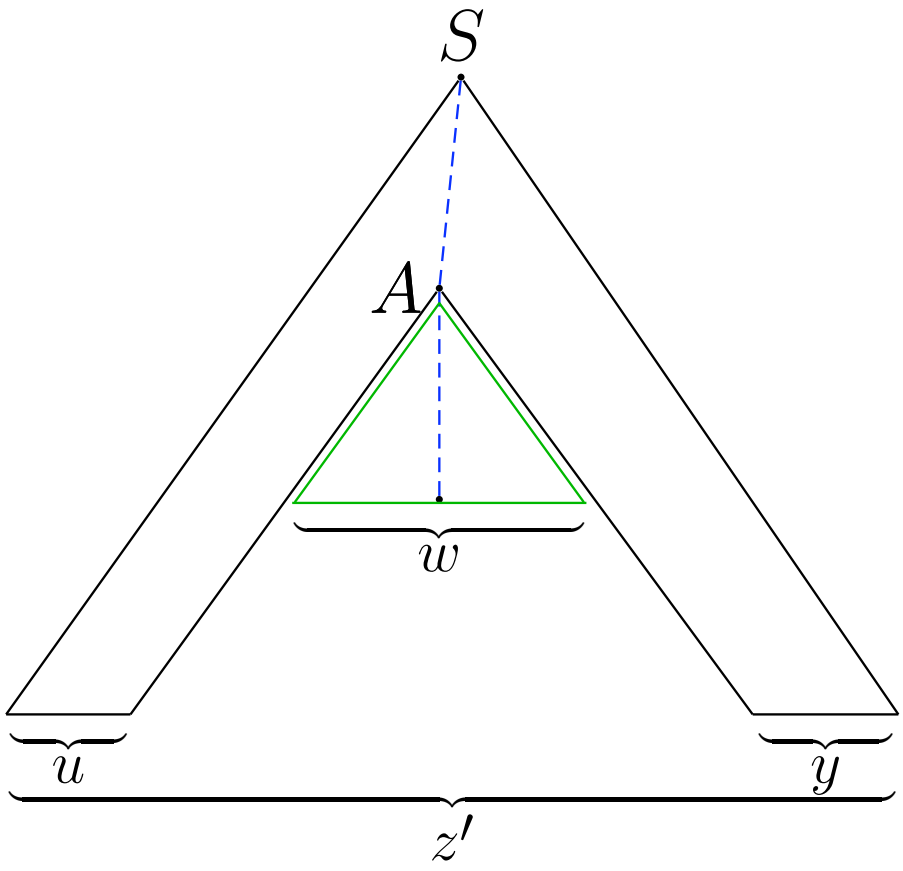
Auf dem *blauen Pfad* kommt A mindestens zweimal vor. u, v, x, y sind die Teilwörter von z, die links bzw. rechts des ersten bzw. zweiten A entstehen. w ist das Wort, das aus dem zweiten A entsteht. Es gilt also

$$S \stackrel{*}{\Rightarrow} uAy$$
$$A \stackrel{+}{\Rightarrow} vAx$$
$$A \stackrel{*}{\Rightarrow} w$$

und damit insgesamt

$$S \stackrel{*}{\Rightarrow} uAy \stackrel{+}{\Rightarrow} uvAxy \stackrel{*}{\Rightarrow} uvwxy = z$$

Jetzt entfernen wir das rote Stück aus dem Ableitungsbaum, und erhalten einen Ableitungsbaum für ein neues Wort z', aus dem mindestens eine Wiederholung eines Nichtterminalzeichens entfernt ist.



Eine zugehörige Ableitung ist $S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uwy = z'$.

Indem man dieses Verfahren wiederholt, erhält man irgendwann einen Ableitungsbaum, dessen Pfade *keine* Wiederholungen von Nichtterminalzeichen mehr enthalten.

Der hat dann höchstens die Höhe $|N|$. □

Bemerkungen:

- Die im Beweis zu Satz 3.29 angegebenen Algorithmen sind sehr ineffizient.

  Ein halbwegs effizienter Algorithmus für das allgemeine Wortproblem ist das *Cocke-Younger-Kasami-* (kurz: *CYK-*) *Verfahren*, das für alle kontextfreien Grammatiken in Chomsky-Normalform (Übung 6, Aufgabe 3) funktioniert, s. Literatur.

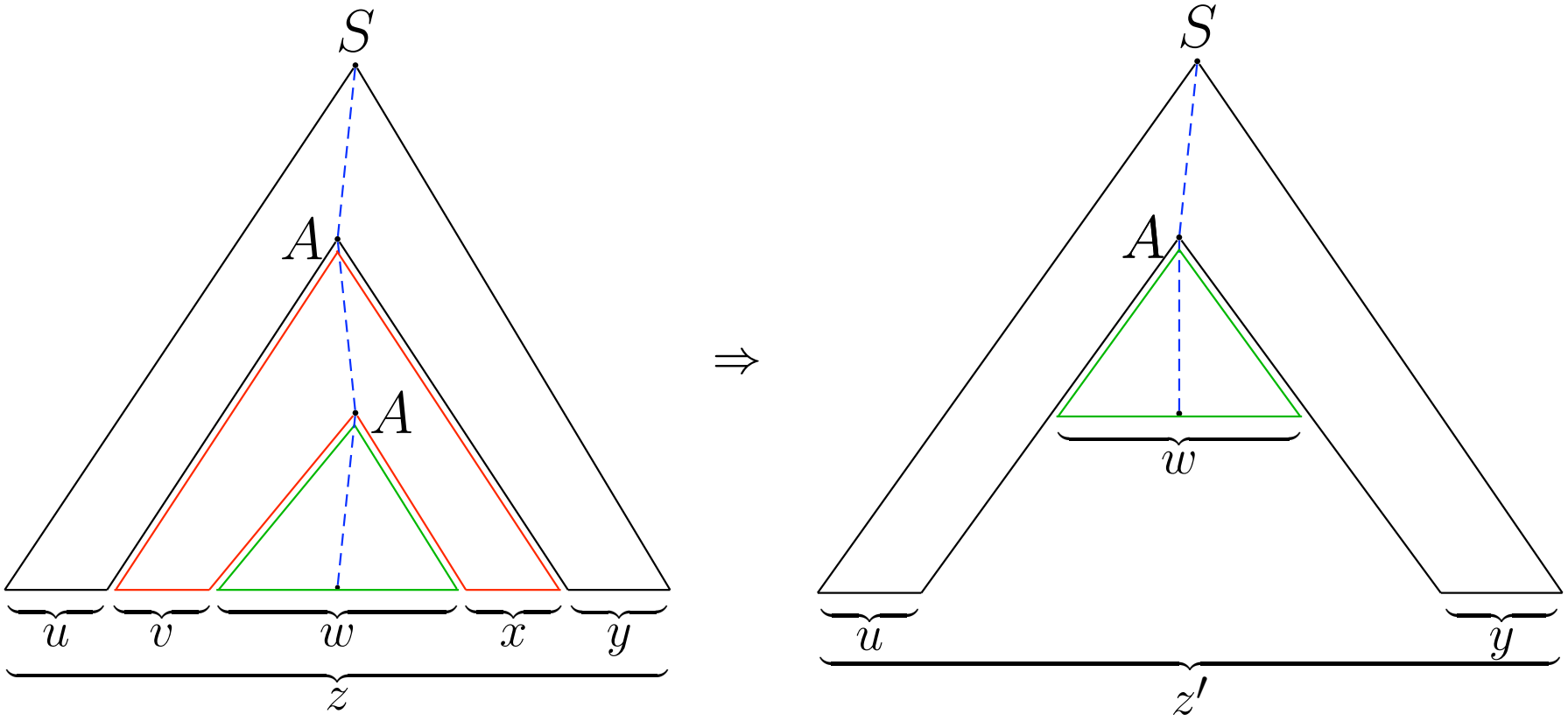
  Ein effizienter Algorithmus für das Leerheitsproblem wird in Übung 7, Aufgabe 1 besprochen.

- Für einige andere Fragestellungen über KFGs gibt es *keine* Entscheidungsalgorithmen, z.B. für das *Äquivalenzproblem*:

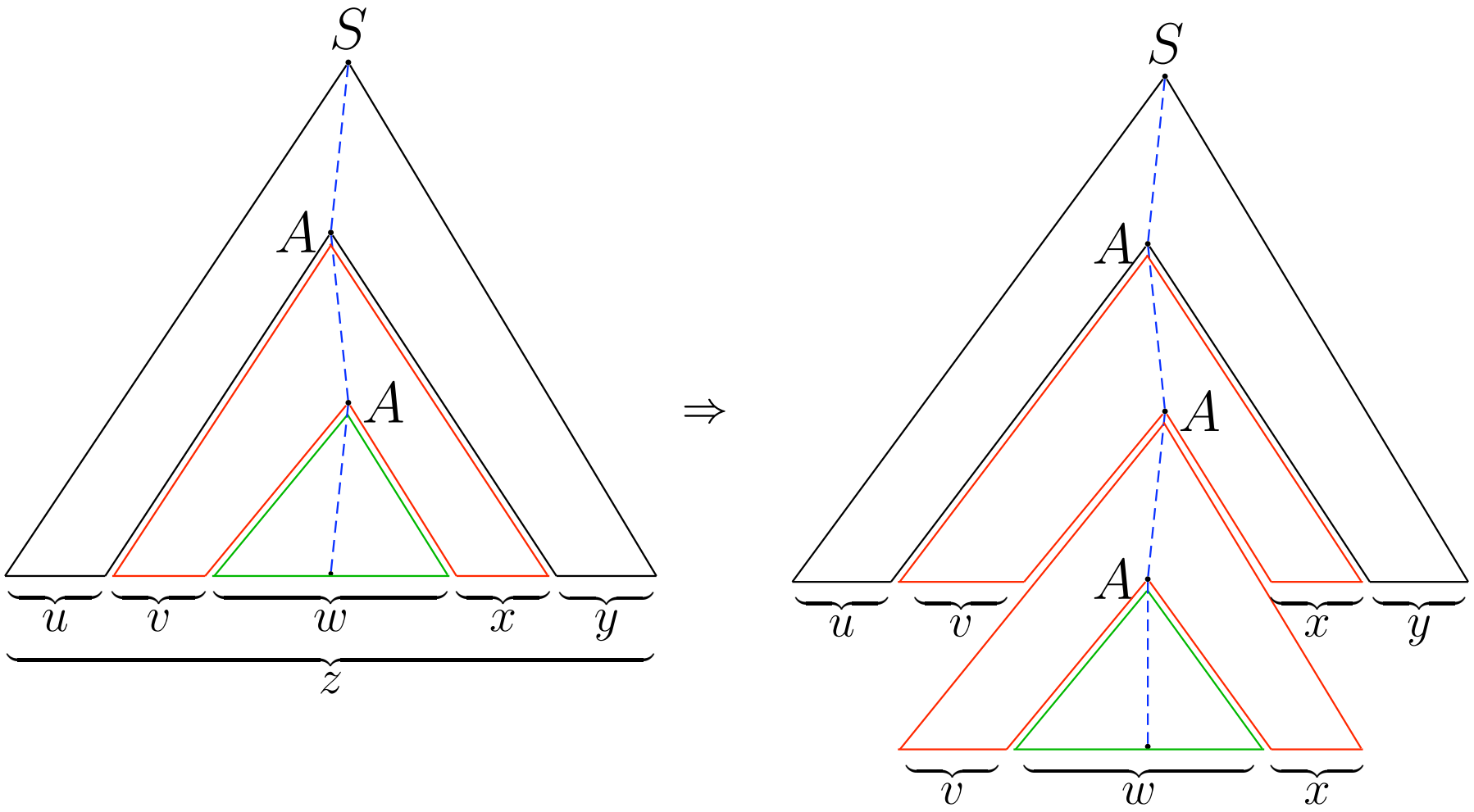
  Es gibt *keinen* Algorithmus, der für zwei beliebige KFGs G_1 und G_2 testet, ob $L(G_1) = L(G_2)$. Solche Ergebnisse über die *Unentscheidbarkeit* gewisser Fragestellungen sind Gegenstand der Berechenbarkeitstheorie (Teil II der Vorlesung).

Grenzen kontextfreier Sprachen

Im Beweis zu Satz 3.29 haben wir aus einem ‘hinreichend großen’ Ableitungsbaum ein Stück herausgeschnitten.



Stattdessen kann man das Stück auch wiederholen:



So erhält man aus der ursprünglichen Ableitung

$$S \overset{*}{\Rightarrow} uAy \overset{+}{\Rightarrow} uvAxy \overset{*}{\Rightarrow} uvwxy$$

neue Ableitungen der Form

$$S \overset{*}{\Rightarrow} uAy \overset{+}{\Rightarrow} \ldots \overset{+}{\Rightarrow} uv^iAx^iy \overset{*}{\Rightarrow} uv^iwx^iy$$

indem man i-mal die Ableitung $A \overset{+}{\Rightarrow} vAx$ wiederholt.

Das ist die Beweisidee für

Satz 3.30 (Pumping Lemma für kontextfreie Sprachen) *Sei $L \subseteq \Sigma^*$ kontextfrei. Dann existiert eine Zahl $n \geq 1$, so dass für jedes $z \in L$ mit $|z| \geq n$ gilt: Es gibt eine Zerlegung $z = uvwxy$ mit $vx \neq \varepsilon$ und $uv^iwx^iy \in L$ für alle $i \geq 0$.*

Beweis:

Sei $L = L(G)$, wobei $G = (\Sigma, N, S, P)$ eine Grammatik ist, die mit den Konstruktionen aus Satz 3.27 und Satz 3.28 entstanden ist, und sei $p = \max\{\,|\gamma| \mid (A \to \gamma) \in P\}$.

Dann hat das Blattwort eines Ableitungsbaums der Höhe m höchstens die Länge p^m (weil der Ableitungsbaum von einer Ebene zur nächsten höchstens um den *Faktor* p breiter werden kann).

Wir wählen $n = p^{|N|} + 1 > p^{|N|}$.

Dann hat jeder Ableitungsbaum für ein Wort z mit $|z| \geq n$ mindestens die Höhe $|N| + 1$, also ist die Ableitung für z von der Form

$$S \overset{*}{\Rightarrow} uAy \overset{+}{\Rightarrow} uvAxy \overset{*}{\Rightarrow} uvwxy$$

und wir erhalten daraus Ableitungen

$$S \overset{*}{\Rightarrow} uAy \overset{+}{\Rightarrow} \ldots \overset{+}{\Rightarrow} uv^iAx^iy \overset{*}{\Rightarrow} uv^iwx^iy$$

für alle Wörter uv^iwx^iy mit $i \geq 0$. Dabei gilt $vx \neq \varepsilon$, weil $A \overset{+}{\Rightarrow} \varepsilon A\varepsilon = A$ ohne Einheits- und ε-Produktionen nicht möglich ist. $\square$

Korollar 3.31 *Zum Nachweis, dass L nicht kontextfrei ist, genügt es zu zeigen: Für jedes $n \geq 1$ existiert ein $z \in L$ mit $|z| \geq n$, so dass für alle Zerlegungen $z = uvwxy$ mit $vx \neq \varepsilon$ gilt: Es gibt ein $i \geq 0$ mit $uv^iwx^iy \notin L$.*

Der Beweis, dass eine Sprache L nicht kontextfrei ist, läuft also ähnlich ab wie beim Pumping Lemma für reguläre Sprachen.

Für *jedes* $n \geq 1$ gibt man ein *passendes* Wort z mit $|z| \geq n$ an. Dann muss man für *jede* Zerlegung $z = uvwxy$ mit $vx \neq \varepsilon$ einen *passenden* Pumpfaktor i wählen mit $uv^iwx^iy \notin L$.

Merke:
z und i darf man *passend wählen*, aber man muss *alle* Zerlegungen des Wortes z betrachten. Dabei ist oft eine aufwändige Fallunterscheidung nötig, weil man *alle* Möglichkeiten für die Positionen der Teilwörter v und x im Wort z durchspielen muss.

Beispiel:

Die Sprache $L = \{a^n b^n c^n \mid n \geq 0\}$ ist nicht kontextfrei, denn:

Sei $n \geq 1$.

Wir wählen $z = a^n b^n c^n$. Dann ist $z \in L$ und $|z| \geq n$.

Sei nun $z = uvwxy$ eine beliebige Zerlegung von z mit $vx \neq \varepsilon$.

1. Fall: Wenn v oder x mehr als eines der Zeichen a, b, c enthält, dann ist $uv^2wx^2y \notin L$, weil in diesem Wort die Zeichen a, b, c nicht mehr in der richtigen Reihenfolge stehen.

2. Fall: Wenn $v = a^k$ und $x = b^l$, dann sind k und l nicht beide 0, also ist $uv^2wx^2y \notin L$, weil dieses Wort mehr as als cs oder mehr bs als cs (oder beides) enthält.

Alle übrigen Fälle (z.B. $v = a^k$ und $x = a^l$) verlaufen analog, weil mindestens eines der Zeichen a, b, c in vx fehlt. Beim Pumpen bleibt die Anzahl dieses Zeichens gleich, während sich die Anzahl eines der anderen beiden Zeichen erhöht.

Korollar 3.32 *Der Durchschnitt zweier kontextfreier Sprachen ist im allgemeinen nicht kontextfrei.*

Beweis:

Sei $L_1 = \{a^n b^n \mid n \geq 0\} \circ \{c\}^* = \{a^n b^n c^m \mid m, n \geq 0\}$

und $L_2 = \{a\}^* \circ \{b^n c^n \mid n \geq 0\} = \{a^m b^n c^n \mid m, n \geq 0\}$.

L_1 und L_2 sind kontextfrei, weil sie beide durch Konkatenation aus kontextfreien Sprachen entstehen,

aber $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ ist *nicht* kontextfrei. □

Auch das Pumping Lemma für kontextfreie Sprachen lässt sich (wie das Pumping Lemma für reguläre Sprachen) so verstärken, dass es besser einsetzbar ist.

Satz 3.33 (Starkes Pumping Lemma für kontextfreie Sprachen) *Sei $L \subseteq \Sigma^*$ kontextfrei. Dann existiert eine Zahl $n \geq 1$, so dass für jedes $z \in L$ mit $|z| \geq n$ gilt: Es gibt eine Zerlegung $z = uvwxy$ mit $vx \neq \varepsilon$, $|vwx| \leq n$ und $uv^iwx^iy \in L$ für alle $i \geq 0$.*

Beweis:

Um die Abschätzung $|vwx| \leq n$ zu erhalten, muss man im Beweis von Satz 3.30

- die Zahl n etwas größer wählen,

- und das Nichtterminalzeichen A sorgfältiger auswählen.

Sei $n = p^{|N|+1}$ und $z \in \Sigma^*$ mit $|z| \geq n$.

Dann wählen wir im Ableitungsbaum für z einen Pfad *maximaler Länge* (von der Wurzel S zu einem Blatt).

Da n sogar größer gewählt ist als im Beweis von Satz 3.30, muss auf diesem Pfad mindestens ein Nichtterminalzeichen mehrmals vorkommen.

Wir wählen das Nichtterminalzeichen A aus, das sich vom Blatt aus gesehen als *erstes* wiederholt.

Diese erste Wiederholung passiert (wenn man wieder vom Blatt ausgeht) nach spätestens $|N|+1$ Schritten, d.h. der Teilpfad vom Blatt zu diesem Zeichen A hat höchstens die Länge $|N|+1$.

Da der Pfad von S zum Blatt maximal gewählt war, kann es von diesem A aus keinen Pfad zu einem anderen Blatt geben, der länger ist als $|N|+1$.

Also hat das Wort vwx, das ja aus diesem Zeichen A entsteht, höchstens die Länge $p^{|N|+1} = n$. □

Beispiel:

Die Sprache $L = \{ww \mid w \in \{a,b\}^*\}$ ist nicht kontextfrei (im Gegensatz zur Sprache $L' = \{ww^R \mid w \in \{a,b\}^*\}$).

Sei $n \geq 1$. Wir wählen $z = a^n b^n a^n b^n$, also $z \in L$ und $|z| = 4n \geq n$.

Mit dem schwachen Pumping Lemma (Satz 3.30) kann man hier nicht argumentieren. Sei nämlich $z = uvwxy$ mit $u = \varepsilon$, $v = x = a^n$ und $w = y = b^n$. Dann gilt $uv^i wx^i y = a^{in} b^n a^{in} b^n \in L$ für *jedes* $i \geq 0$, d.h. für diese Zerlegung von z erhält man *keinen* Widerspruch.

Aber mit dem starken Pumping Lemma kommt man zum Ziel:

Wenn $z = uvwxy$ mit $|vwx| \leq n$ und $vx \neq \varepsilon$, so unterscheiden wir die folgenden drei Fälle:

1. vwx beginnt im linken a^n: Dann liegt vwx in der linken Worthälfte $a^n b^n$.
2. vwx beginnt im linken b^n: Dann liegt vwx im Mittelstück $b^n a^n$.
3. vwx liegt in der rechten Worthälfte $a^n b^n$.

In allen drei Fällen betrachten wir $z_0 = uv^0wx^0y$. Dieses Wort entsteht aus $z = uvwxy$, indem man v und x (also einen Teil von vwx) entfernt. Also hat z_0 in den drei oben genannten Fällen die Form $a^kb^la^nb^n$ oder $a^nb^ka^lb^n$ oder $a^nb^na^kb^l$, wobei $k < n$ oder $l < n$ (oder beides) gilt. All diese Wörter sind offensichtlich *nicht* von der Form ww, also gilt in jedem Fall $z_0 \notin L$, d.h. L ist nicht kontextfrei. □

Das Beispiel erklärt, warum (die meisten) Programmiersprachen nicht kontextfrei sind. Üblicherweise müssen die Programme nämlich sogenannte 'Kontextbedingungen' erfüllen, z.B.: "Jede Funktion muss deklariert sein, bevor sie aufgerufen wird."
Um eine solche Bedingung zu überprüfen, muss man den Funktionsnamen an der Deklarationsstelle mit dem Funktionsnamen an der Aufrufstelle vergleichen (so wie man in L die erste mit der zweiten Worthälfte vergleichen muss). Solche Vergleiche sind—wie wir an der Sprache L gesehen haben—mit einer kontextfreien Grammatik nicht möglich.

Kontextfreie Sprachen

Trotzdem spielen kontextfreie Grammatiken in der Praxis eine wichtige Rolle. Bei der Definition einer Programmiersprache L geht man nämlich so vor:

Zunächst definiert man die sogenannte 'kontextfreie Syntax' von L, d.h. man definiert eine kontextfreie Sprache $L' \supseteq L$. Dann definiert man L als die Menge aller Wörter aus L', die gewisse 'Kontextbedingungen' (wie die oben genannte Bedingung, dass jede Funktion deklariert sein muss, bevor man sie aufruft) erfüllen.

Ganz analog geht man bei der Implementierung der Programmiersprache vor: Durch die *syntaktische Analyse* (die vom *Parser* durchgeführt wird) wird überprüft, ob die vom Programmierer eingegebene Zeichenreihe der kontextfreien Syntax genügt, d.h. ob sie in L' liegt. Wenn ja, so erzeugt der Parser den Syntaxbaum für die Zeichenreihe. Bei der anschließenden *semantischen Analyse* werden dann am Syntaxbaum die *Kontextbedingungen* überprüft.

Weitere Beispiele für die Anwendung des Pumping Lemmas:

Sei $\Sigma = \{a\}$. Dann sind die früher betrachteten Sprachen

- $\{a^{n^2} \mid n \geq 0\}$
- $\{a^{2^n} \mid n \geq 0\}$
- $\{a^p \mid p \text{ Primzahl}\}$

nicht kontextfrei.

Beweis:
Mit Hilfe des Pumping Lemmas für reguläre Sprachen haben wir bewiesen, dass diese Sprachen nicht regulär sind. Da es bei einem einelementigen Alphabet keinen Unterschied macht, ob man an *einer* oder an *zwei* Stellen pumpt, lassen sich diese Beweise leicht so abändern, dass man in ihnen das Pumping Lemma für kontextfreie Sprachen benutzt. Also sind diese Sprachen auch nicht kontextfrei. □

In der Tat gilt über jedem *einelementigen* Alphabet:

$$\mathcal{L}_{reg} \;=\; \mathcal{L}_{kf}$$

Das lässt sich aber nicht mit dem Pumping Lemma beweisen, denn in beiden Fällen (sowohl bei regulären, als auch bei kontextfreien Sprachen) gibt das Pumping Lemma nur eine notwendige und keine hinreichende Bedingung an.

Wir verzichten auf den Beweis dieser Gleichheit (und verwenden sie auch nicht).

Über jedem Alphabet mit mindestens zwei Zeichen gilt natürlich

$$\mathcal{L}_{reg} \;\subsetneq\; \mathcal{L}_{kf}$$

wie wir bereits bewiesen haben.

Deterministisch kontextfreie Sprachen

Die syntaktische Analyse von Programmen wird (im Prinzip) mit Kellerautomaten durchgeführt. Wir haben bereits gesehen, wie man aus einer kontextfreien Grammatik G einen Kellerautomaten M mit $L(M) = L(G)$ erhält, aber dieser Kellerautomat war (in hohem Maße) nichtdeterministisch. Für die Praxis benötigt man natürlich "deterministische" Kellerautomaten. Dieser Begriff muss nun erst einmal definiert werden. Es genügt sicherlich *nicht*, zu fordern, dass die Übergangsrelation Δ des Kellerautomaten M eine (partielle) Funktion ist. Man betrachte z.B.

$$\Delta = \{((p, a, \varepsilon), (p, \varepsilon)), ((p, \varepsilon, b), (p, \varepsilon)\}$$

Diese Relation Δ ist eine partielle Funktion, aber der zugehörige Kellerautomat hat trotzdem die Auswahl zwischen zwei Transitionen, wenn die aktuelle Eingabe mit a beginnt und der aktuelle Kellerinhalt mit b.

Solche "Konflikte" zwischen zwei Transitionen sollten in einem deterministischen Kellerautomaten ausgeschlossen sein. Das erreicht man durch die folgende

Definition 3.34 *Sei $M = (\Sigma, \Gamma, Q, s, F, \Delta)$ ein Kellerautomat.*

1. *Zwei Wörter α_1, α_2 heißen* konsistent*, wenn α_1 Präfix von α_2 oder α_2 Präfix von α_1 ist.*
2. *Zwei Transitionen $((p_1, u_1, \beta_1), (q_1, \gamma_1))$ und $((p_2, u_2, \beta_2), (q_2, \gamma_2))$ heißen* kompatibel*, wenn gilt:*

   - $p_1 = p_2$,
   - u_1, u_2 *sind konsistent und*
   - β_1, β_2 *sind konsistent*

   Ansonsten heißen sie inkompatibel*.*
3. *M heißt* deterministisch*, wenn die Transitionen in Δ paarweise inkompatibel sind.*

Kontextfreie Sprachen

Man beachte, dass nur kompatible Transitionen $((p_1, u_1, \beta_1), (q_1, \gamma_1))$ und $((p_2, u_2, \beta_2), (q_2, \gamma_2))$ auf die gleiche Konfiguration (p, u, β) anwendbar sein können, denn dazu muss gelten:

- $p_1 = p_2 = p$,
- u_1 und u_2 konsistent, weil beides Präfixe von u sind,
- β_1 und β_2 konsistent, weil beides Präfixe von β sind.

Also sind zwei Transitionen eines deterministischen Kellerautomaten niemals auf die gleiche Konfiguration anwendbar, d.h. für einen deterministischen Kellerautomaten M ist die Relation $\vdash_M$ eine partielle Funktion.

Im Prinzip definieren wir jetzt die deterministisch kontextfreien Sprachen als diejenigen, die von deterministischen Kellerautomaten erkannt werden, wobei wir den Automaten aber noch die Möglichkeit geben, das Ende des Eingabeworts zu erkennen.

Definition 3.35 *Eine Sprache $L \subseteq \Sigma^*$ heißt* deterministisch kontextfrei*, wenn es einen deterministischen Kellerautomaten M gibt, der die Sprache*

$$L\$ = \{w\$ \mid w \in L\}$$

erkennt, wobei \$ ein neues Zeichen ist, d.h. $\$ \notin \Sigma$.

(In der Praxis, d.h. bei einem Parser, erkennt man ebenfalls das Ende der Eingabe, z.B. indem man auf 'end of file' testet.)

Beispiel:

Die Sprache $L = \{a^n b^n \mid n \geq 0\}$ ist deterministisch kontextfrei.

Ein deterministischer Kellerautomat, der $L\$$ erkennt, arbeitet wie folgt:

Wenn er im Startzustand schon das Zeichen \$ liest, geht er sofort in den Endzustand.

Wenn er im Startzustand ein a liest, geht er in einen Zustand q_a, in dem weitere as gelesen werden können. Diese as werden in den Keller verschoben.

Sobald er das erste b liest, wechselt er in einen Zustand q_b, in dem nur noch bs (oder \$) gelesen werden dürfen.

Die bs werden gegen die as im Keller aufgehoben.

Sobald \$ erscheint, wechselt der Automat in den Endzustand.

Wenn gleich viele as und bs gelesen wurden, ist der Keller am Ende leer, also wird das Wort akzeptiert.

Andernfalls bleibt der Automat entweder stecken (d.h. das Eingabeband ist nicht leer) oder der Keller ist am Ende nicht leer, also wird das Wort nicht akzeptiert.

Formale Definition dieses Kellerautomaten:

$M = (\{a, b, \$\}, \{a\}, \{s, q_a, q_b, f\}, s, \{f\}, \Delta)$ mit:

$$
\begin{aligned}
\Delta = \{\ & ((s, \$, \varepsilon), (f, \varepsilon)), && (1)\\
& ((s, a, \varepsilon), (q_a, \varepsilon)), && (2)\\
& ((q_a, a, \varepsilon), (q_a, a)), && (3)\\
& ((q_a, b, \varepsilon), (q_b, \varepsilon)), && (4)\\
& ((q_b, b, a), (q_b, \varepsilon)), && (5)\\
& ((q_b, \$, \varepsilon), (f, \varepsilon))\ \} && (6)
\end{aligned}
$$

Wir zeigen, dass tatsächlich $L(M) = L\$ = \{a^n b^n \$ \mid n \geq 0\}$ gilt.

'$\supseteq$': Sei $w = a^n b^n \$$.

Im Falle $n = 0$ gilt $(s, w, \varepsilon) = (s, \$, \varepsilon) \vdash_{(1)} (f, \varepsilon, \varepsilon)$, also $w \in L(M)$.

Im Falle $n > 0$ gilt:

$$\begin{aligned}
(s, w, \varepsilon) \;=\; (s, a^n b^n \$, \varepsilon) \;&\vdash_{(2)} \quad (q_a, a^{n-1} b^n \$, \varepsilon) \\
&\vdash_{(3)}^{n-1} \quad (q_a, b^n \$, a^{n-1}) \\
&\vdash_{(4)} \quad (q_b, b^{n-1} \$, a^{n-1}) \\
&\vdash_{(5)}^{n-1} \quad (q_b, \$, \varepsilon) \\
&\vdash_{(6)} \quad (f, \varepsilon, \varepsilon)
\end{aligned}$$

'$\subseteq$': Sei $w \in L(M)$, d.h. $(s, w, \varepsilon) \vdash_M^* (f, \varepsilon, \varepsilon)$.

Wie kann diese Folge von Übergangsschritten aussehen?

An den Transitionen sieht man zunächst, dass *nur* die folgenden Zustandsübergänge möglich sind:

- von s mit (1) nach f oder mit (2) nach q_a,
- von q_a mit (3) nach q_a oder mit (4) nach q_b,
- von q_b mit (5) nach q_b oder mit (6) nach f.

Insbesondere ist im Zustand f *kein* Übergangsschritt mehr möglich. Also ist die gesamte Schrittfolge von der Form

- $(s, w, \varepsilon) \vdash_{(1)} (f, \varepsilon, \varepsilon)$ oder
- $(s, w, \varepsilon) \vdash_{(2)} (q_a, w_1, \beta_1) \vdash^m_{(3)} (q_a, w_2, \beta_2)$
  $\vdash_{(4)} (q_b, w_3, \beta_3) \vdash^n_{(5)} (q_b, w_4, \beta_4)$
  $\vdash_{(6)} (f, \varepsilon, \varepsilon)$

  mit $m, n \geq 0$

Im ersten Fall ist $w = \$ = a^0 b^0 \$ \in L\$$, und im zweiten Fall erhalten wir wegen der Form der Transitionen:

- $w = aw_1$ und $\beta_1 = \varepsilon$
- $w_1 = a^m w_2$ und $\beta_2 = a^m$
- $w_2 = bw_3$ und $\beta_3 = \beta_2 = a^m$
- $w_3 = b^n w_4$ mit $n \leq m$ und $\beta_4 = a^{m-n}$
- $w_4 = \$$ und $\beta_4 = \varepsilon$, also $m = n$

Insgesamt ergibt sich daraus $w = a^{n+1} b^{n+1} \$ \in L\$$. □

Lemma 3.36 *Jede deterministisch kontextfreie Sprache ist kontextfrei.*

Beweis:

Wenn $L \subseteq \Sigma^*$ deterministisch kontextfrei ist, dann existiert ein Kellerautomat $M = (\Sigma \cup \{\$\}, \Gamma, Q, s, F, \Delta)$, der $L\$$ erkennt.

Das bedeutet aber zunächst nur, dass $L\$$ kontextfrei ist. Also bleibt zu zeigen, dass auch L kontextfrei ist, d.h. dass ein Kellerautomat M' existiert, der L erkennt.

Sei $M' = (\Sigma, \Gamma, Q \times \{0,1\}, (s,0), F \times \{1\}, \Delta')$ mit

$$\begin{aligned}
\Delta' &= \{(((p,0),u,\beta),((q,0),\gamma)) \mid ((p,u,\beta),(q,\gamma)) \in \Delta\} && (1)\\
&\cup \{(((p,0),u,\beta),((q,1),\gamma)) \mid ((p,u\$,\beta),(q,\gamma)) \in \Delta\} && (2)\\
&\cup \{(((p,1),\varepsilon,\beta),((q,1),\gamma)) \mid ((p,\varepsilon,\beta),(q,\gamma)) \in \Delta\} && (3)
\end{aligned}$$

Intuition:

Immer wenn der deterministische Kellerautomat M das Ende der Eingabe mit Hilfe des Endmarkers \$ *erkennt*, dann kann der nichtdeterministische Kellerautomat M' das Ende der Eingabe *erraten*, indem er von 0 zu 1 wechselt. Nach diesem Wechsel von 0 zu 1 kann er kein Eingabezeichen mehr lesen, also kann er nur noch—mit den gleichen Übergangsschritten wie M—den Keller bearbeiten.

Es bleibt zu zeigen, dass tatsächlich $L = L(M')$ gilt.

'$\subseteq$': Sei $w \in L$, also $w\$ \in L\$ = L(M)$.

Dann existiert ein $f \in F$ mit $(s, w\$, \varepsilon) \vdash_M^* (f, \varepsilon, \varepsilon)$

Wenn wir diese Folge von Übergangsschritten dort aufteilen, wo das Ende der Eingabe (einschließlich des Endmarkers) gelesen wird, dann hat sie die Form

$$(s, w\$, \varepsilon) \vdash_M^* (p, u\$, \beta) \vdash_M (q, \varepsilon, \gamma) \vdash_M^* (f, \varepsilon, \varepsilon)$$

In der linken Teilfolge spielt der Endmarker \$ keine Rolle, weil ein Kellerautomat ein Eingabezeichen nicht "sehen" kann ohne es zu entfernen, also gilt auch

$$(s, w, \varepsilon) \vdash_M^* (p, u, \beta)$$

Aus dieser und aus den beiden rechten Teilfolgen ergibt sich dann per Definition von Δ', dass M' die folgenden Übergangsschritte

machen kann:

$$((s,0),w,\varepsilon) \vdash^*_{(1)} ((p,0),u,\beta) \vdash_{(2)} ((q,1),\varepsilon,\gamma) \vdash^*_{(3)} ((f,1),\varepsilon,\varepsilon)$$

Also ist $w \in L(M')$.

'$\supseteq$': Sei $w \in L(M')$, d.h. es existiert ein $f \in F$ mit

$$((s,0),w,\varepsilon) \vdash^*_{M'} ((f,1),\varepsilon,\varepsilon)$$

Da man nur mit (2) von 0 zu 1 wechseln kann, und da man mit (3) keine Eingabe mehr lesen kann, muss diese Folge so aussehen:

$$((s,0),w,\varepsilon) \vdash^*_{(1)} ((p,0),u,\beta) \vdash_{(2)} ((q,1),\varepsilon,\gamma) \vdash^*_{(3)} ((f,1),\varepsilon,\varepsilon)$$

Daraus folgt nun umgekehrt mit der Definition von Δ'

$$(s,w\$,\varepsilon) \vdash^*_M (p,u\$,\beta) \vdash_M (q,\varepsilon,\gamma) \vdash^*_M (f,\varepsilon,\varepsilon)$$

also $w\$ \in L(M) = L\$$ und damit $w \in L$. □

Ist der Endmarker wirklich notwendig in der Definition deterministisch kontextfreier Sprachen?

Im Beweis haben wir zu einem deterministischen Kellerautomaten M, der $L\$$ erkennt, einen (im allgemeinen) nichtdeterministischen Kellerautomaten M' konstruiert, der L erkennt. Das ist ein *Indiz* dafür, dass man den Endmarker benötigt.

Das folgende Beispiel zeigt, dass wir den Endmarker im allgemeinen tatsächlich nicht entbehren können.

Sei $L = \{a\}^* \cup \{a^n b^n \mid n \in \mathbb{N}\}$. Dann gilt:

- L ist deterministisch kontextfrei, d.h. es gibt einen deterministischen Kellerautomaten, der $L\$$ erkennt.
- Aber es gibt *keinen* deterministischen Kellerautomaten, der L erkennt.

Beweis:

Ein deterministischer Kellerautomat, der $L\$$ erkennt, arbeitet wie folgt:

Zunächst verschiebt er as in den Keller, so lange bis entweder \$ oder b auftaucht.

Im Falle von \$ geht er in einen Endzustand über, der ihm erlaubt, noch den Keller zu leeren.

Im Falle von b geht er in einen anderen Endzustand über, in dem er bs gegen as aufhebt und schließlich den Endmarker löscht.

Formale Definition dieses Kellerautomaten:
s. Übung

Es bleibt zu zeigen, dass *kein* deterministischer Kellerautomat existiert, der L erkennt.

Angenommen $L = L(M)$ für einen deterministischen Kellerautomaten $M = (\Sigma, \Gamma, Q, s, F, \Delta)$.

Dann ist $a^n \in L(M)$ für jedes $n \geq 0$, also existiert für jedes $n \geq 0$ ein Zustand $f \in F$ mit $(s, a^n, \varepsilon) \vdash_M^* (f, \varepsilon, \varepsilon)$

Da es nur endlich viele Zustände $f \in F$ gibt, existieren mindestens zwei Wörter a^m und a^n mit $m \neq n$, die zur gleichen Konfiguration $(f, \varepsilon, \varepsilon)$ führen.

Nun betrachte man die Arbeitsweise von M für die beiden Eingaben $a^m b^m$ und $a^n b^m$.

Mit beiden Wörtern gelangt M zunächst in die Konfiguration $(f, \varepsilon, \varepsilon)$ und muss anschließend noch das Wort b^m lesen. Also werden beide Wörter von M gleich behandelt. Das ist ein Widerspruch zur Annahme, weil $a^m b^m \in L$ und $a^m b^n \notin L$. □

Satz 3.37 *Die Klasse der deterministisch kontextfreien Sprachen ist abgeschlossen unter Komplement.*

Beweisidee:

Man vertauscht die Endzustände mit den Nicht-Endzuständen (wie beim DEA).

Das geht aber nicht gut, weil ein Kellerautomat (auch ein deterministischer) viele Möglichkeiten hat, ein Wort 'abzulehnen':

Er kann

- mit nichtleerem Keller halten,

- bei der Berechnung 'steckenbleiben', d.h. in eine Konfiguration geraten, in der das Eingabewort noch nicht abgearbeitet, aber keine Transition mehr anwendbar ist,

- in eine Endlosschleife geraten, ohne dass er das Eingabewort abgearbeitet hat.

Zum Beweis von Satz 3.37 muss man diese Möglichkeiten erst einmal ausschließen,

d.h. man muss zeigen, dass für jede deterministisch kontextfreie Sprache ein "normalisierter" Kellerautomat existiert, der seine Eingabe *immer* vollständig abarbeitet und den Keller *immer* leer macht.

Deshalb verzichten wir hier auf einen exakten Beweis. □

Wir benutzen Satz 3.37, um eine kontextfreie Sprache zu finden, die *nicht* deterministisch kontextfrei ist.

Mit anderen Worten:
Wir zeigen, dass die Klasse $\mathcal{L}_{dkf}$ der deterministisch kontextfreien Sprachen *echt* zwischen den Klassen $\mathcal{L}_{reg}$ und $\mathcal{L}_{kf}$ liegt.
(Jede reguläre Sprache ist deterministisch kontextfrei, weil man einen DEA als deterministischen Kellerautomaten auffassen kann, der den Keller nicht benutzt.)

Korollar 3.38 *Es gibt eine kontextfreie Sprache, die nicht deterministisch kontextfrei ist.*

Beweis:

Sei $L = \{a^l b^m c^n \mid l \neq m \text{ oder } m \neq n\}$.

L ist kontextfrei. Um dies einzusehen, kann man L in der Form

$$L = \{a^l b^m \mid l \neq m\} \circ \{c\}^* \cup \{a\}^* \circ \{b^m c^n \mid m \neq n\}$$

schreiben und für die Sprachen $\{a^l b^m \mid l \neq m\}$ und $\{b^m c^n \mid m \neq n\}$ kontextfreie Grammatiken angeben (s. Übung 6, Aufgabe 2b).

Wäre L deterministisch kontextfrei, dann wäre nach Satz 3.37 auch das Komplement $\bar{L} = \{a, b, c\}^* \setminus L$ (deterministisch) kontextfrei, und damit wäre nach Satz 3.25 auch $\bar{L} \cap \{a\}^*\{b\}^*\{c\}^*$ kontextfrei, weil $\{a\}^*\{b\}^*\{c\}^*$ regulär ist.

Aber es gilt $\bar{L} \cap \{a\}^*\{b\}^*\{c\}^* = \{a^l b^m c^n \mid l = m \text{ und } m = n\} = \{a^n b^n c^n \mid n \geq 0\}$, und diese Sprache ist *nicht* kontextfrei, wie bereits gezeigt wurde. □

Es sei noch erwähnt, dass $\mathcal{L}_{dkf}$ *nicht abgeschlossen* ist unter Vereinigung, Durchschnitt, Konkatenation oder Kleene-Abschluss.

Das Gegenbeispiel für den Durchschnitt ist das gleiche wie für $\mathcal{L}_{kf}$ (Korollar 3.32). Man wähle $L_1 = \{a^n b^n \mid n \geq 0\} \circ \{c\}^*$ und $L_2 = \{a\}^* \circ \{b^n c^n \mid n \geq 0\}$. Beide Sprachen sind deterministisch kontextfrei, denn man kann für sie ähnliche deterministische Kellerautomaten konstruieren wie für die Sprache $\{a^n b^n \mid n \geq 0\}$.

Aber $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ ist—wie wir bereits wissen—noch nicht einmal kontextfrei.

Es folgt sofort, dass $\mathcal{L}_{dkf}$ auch unter Vereinigung nicht abgeschlossen sein kann, weil man ja den Durchschnitt zweier Mengen nach den De Morganschen Regeln durch eine Kombination von Komplement- und Vereinigungsbildung erhalten kann.