Theoretische Informatik – Schreibomat

Florian

January 8, 2013

Tools

- Finite State Machine Designer: `http://madebyevan.com/fsm/`

1 Was ist Informatik?

- Konkrete[1] Konzepte: Algorithmus, Berechnung, Komplexität
- Was will die theoretische Informatik? Formale Bestimmung der Begriffe, unabhängig von Hard- und Software
- Ziel: Grundlegende Eigenschaften von Algorithmen, Berechnungen erkennen. Methoden entwickeln zum Entwurf beweisbar korrekter Hard- und Software (nicht Schwerpunkt dieser Vorlesung)
- Grenzen der automatischen Berechnung aufzeigen
- Typische Fragestellungen: Ist es möglich, ein Programm zu schreiben, das ein anderes Programm als Eingabe erhält und feststellen kann, ob dieses in eine Endlosschleife gerät oder nicht? ⇒ Halteproblem. Nein, es ist natürlich nicht möglich. Eine andere typische Fragestellung ist die Folgende: Wir haben einen Rucksack, und der hat 30 Liter Fassungsvermögen. Und wir haben noch 50 Gegenstände, und jetzt wollen wir wissen: wie lange dauert es, herauszufinden, wieviele Gegenstände in den Rucksack passen? Rucksackproblem, nicht in polynomialer Zeit lösbar ⇒ Es dauert expontiell lange.
- Leider ist es in der Praxis so, dass die Antwort auf ähnliche Fragestellungen nicht immer so offensichtlich ist, und deshalt müssen wir es uns ein bisschen genauer betrachten.

Übung. Gegeben sei das folgende Strassennetz:
(Wildes Gekritzel)

[1] 4. September 2012

(a) Gibt es eine Möglichkeit, alle Strassen genau 1x zu durchlaufen und dann wieder am Ausgangspunkt anzulangen,

(b) Gibt es eine Möglichkeit, alle Kreuzungen genau 1x zu passieren und dann wieder am Ausgangspunkt anzulangen?

Die Antworten sind imfall:

(a) Einfache Aufgabe: Geht immer, wenn die Anzahl der Abzweigungen an jeder Kreuzung gerade ist.

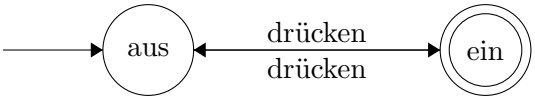
(b) Traveling salesman (TSP). Schwierig! Es ist keine wesentlich bessere Methode bekannt, als einfach alles durchzuprobieren.

Ziel für die Vorlesung: Um solche Fragestellungen systematisch beantworten zu können, brauchen wir ein exaktes mathematisches Modell eines Computers.

2 Automatentheorie

Endliche Automaten, Kontextfreie Grammatiken und Keller-Automaten. Zusätzliche Motivation: Das sind Konzepte, denen man auch häufig in der Praxis begegnet (Compilerbau, Textsuche, Textverarbeitung)

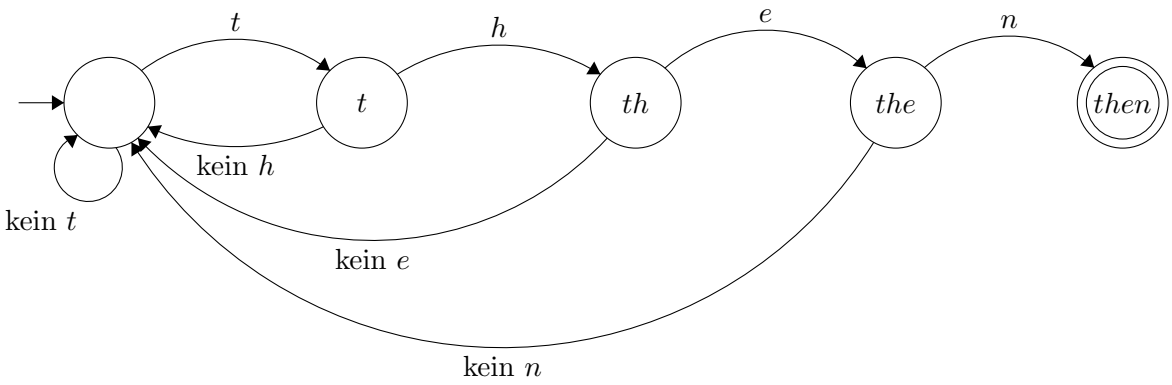
Noch nicht ganz so formal, bisschen intuitiv. Modellierung eines Kippschalters.



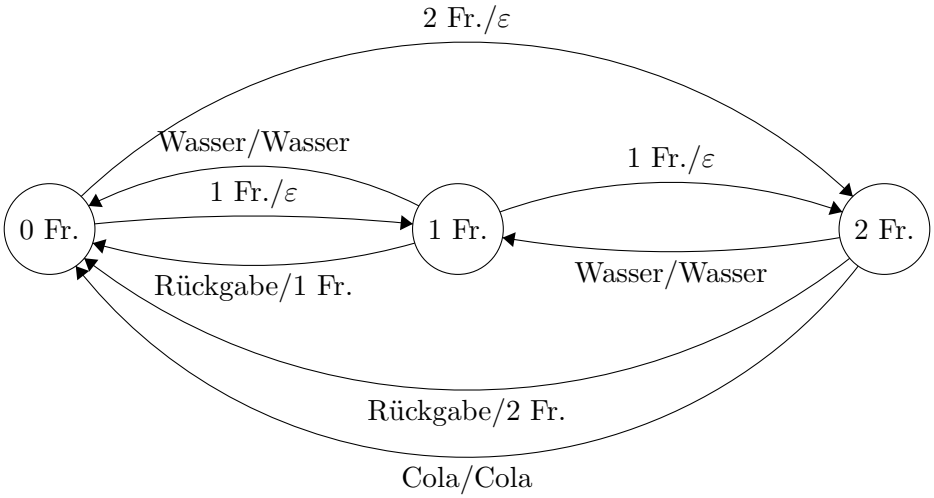
Definition 1 (Endlicher Automat)**.** Wir haben:

- Zwei Zustände, davon ein Startzustand
- und ein akzeptierender Zustand
- Transitionen (Zustandsübergänge): führen den Automaten anhand einer Eingabe von einem Zusand in den nächsten.

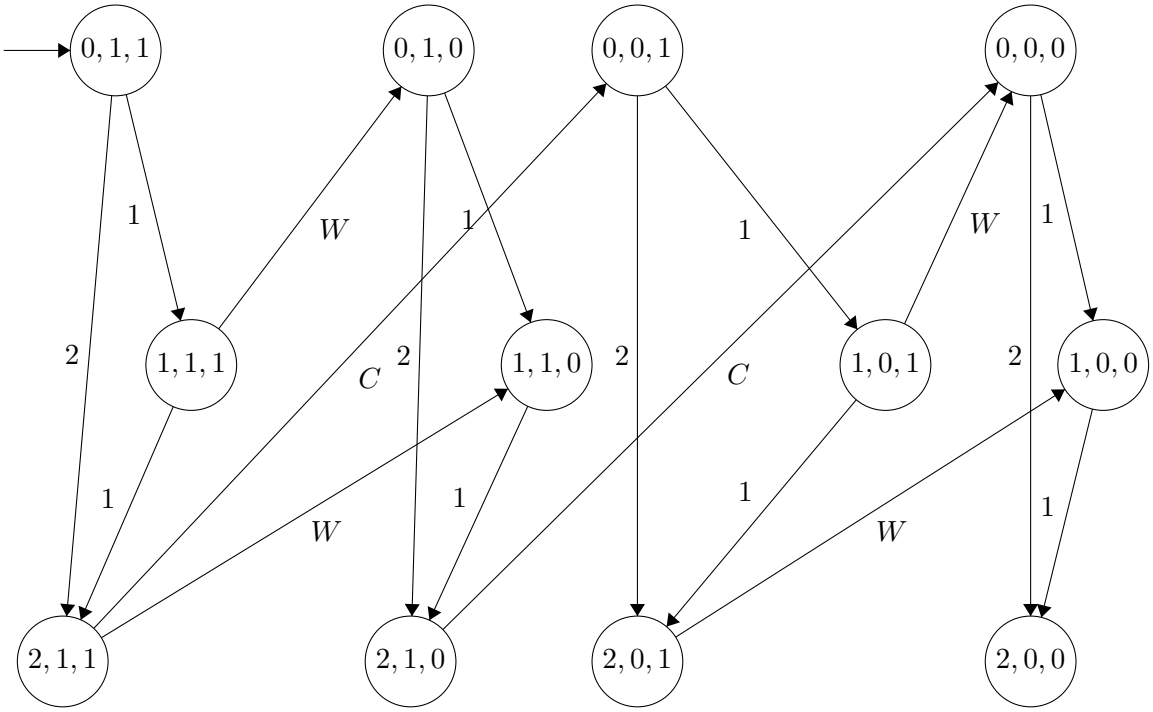
Beispiel: Mustererkennung in Texten: z.B. Suche das Wort "then"



Getränkeautomat (Beispiel für Automat mit Ausgabe): Cola für CHF 2, Wasser für CHF 1. Münzannahme: Münzen zu 1, 2 CHF.



Wenn der Automat nur eine Flasche Cola und eine Flasche Wasser enthält:



3 Formale Sprachen

Ziel: Genaue Beschreibung der Ein- und Ausgaben eines Automaten.

Definition 2 (Alphabet)**.** (endliche, nichtleere Menge von Symbolen). Bsp: Binäres Alphabet $\Sigma_{\text{bin}} = \{0, 1\}$, Tastaturalphabet $\Sigma_{\text{tast}} = \{a, \ldots, z, A, \ldots, Z, 0, \ldots 9, \ldots\}$

Definition 3 (Wort (= Zeichenkette, String))**.** Endliche Folge von Symbolen eines gegebenen Alphabets. Beispiel: 01110 ist ein Wort über Σ_{bin}.

Bemerkung (Eigenschaften von Wörtern)**.** Die da wären:

- Leeres Wort: ε (manchmal λ) = leere Folge von Symbolen (über einem beliebigen Alphabet)
- Länge eines Wortes: $|w|$ bezeichnet die Anzahl der Symbole im Wort w. $|\varepsilon| = 0$.

Bemerkung (Konventionen für Darstellung von Wörtern)**.** Wir sagen:

- $a, b, c, \ldots$ für Buchstaben, Symbole
- $v, w, x, \ldots$ für Wörter

Definition 4 (Potenzen von Wörtern)**.** Es gibt im Angebot:

- Menge aller Wörter einer bestimmten Länge:

Sei Σ Alphabet, so:

$$\begin{aligned} \Sigma^0 &= \{\varepsilon\} \\ \Sigma^1 &= \Sigma \\ \Sigma^2 &= \{ab | a, b \in \Sigma\} \\ \Sigma^i &= \{a_1 \dots a_i | a_1 \dots a_i \in \Sigma\} \end{aligned}$$

- Menge aller Wörter über Σ:

$$\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i, \varepsilon \in \Sigma^*$$

- Menge aller nichtleeren Wörter über Σ:

$$\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i, \varepsilon \notin \Sigma^+$$

Beispiel.

Beispiel:

$$\begin{aligned} \Sigma &= \{0, 1\} \\ \Sigma^2 &= \{00, 01, 10, 11\} \\ \Sigma^* &= \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\} \end{aligned}$$

Definition 5 (Konkatenation (Verkettung) von Wörtern)**.**

$$v = a_1, \dots a_k, w = b_1 \dots, b_k \text{ über } \Sigma \Rightarrow v \cdot w = a_1 \dots a_k b_1 \dots b_k = vw$$

Bemerkung (Rechenregeln für Konkatenation von Wörtern)**.**

$$\begin{aligned} v \cdot (v \cdot w) &= (u \cdot v) \cdot w \\ |x \cdot y| &= |x| + |y| \\ x \cdot \varepsilon &= \varepsilon \cdot x = x \end{aligned}$$

Bemerkung (Infixe, Präfixe, Suffixe)**.** Seien $v, w \in \Sigma^*$ für ein Alphabet Σ, dann ist v ein Teilwort (Infix) von w, falls es $x, y \in \Sigma^*$ gibt, so dass $w = x \cdot v \cdot y$; v ist ein Präfix von w, falls es $y \in \Sigma^*$ gibt, so dass $w = v \cdot y$. Suffix funktionert analog.

Beispiel: abc ist Teilwort von aabcc, aa ist Präfix und Suffix von aabcaa. ε ist Teilwort von jedem Wort.

Definition 6 (Sprache)**.** L über Alphabet Σ ist eine Teilmenge von Σ^*, $L \subseteq \Sigma^*$. Sprache ist Menge von Wörtern, kann unendlich gross sein. Leere Sprache: $\varnothing$ enthält keine Wörter, ist über jedem Alphabet definiert. Spezielle Sprache: L_ε ist die Sprache, die nur das leeere Wort enthält. $L_\varepsilon \neq \varnothing$.

Definition 7 (Konkatenation von Sprachen)**.** $L_1, L_2 \subseteq \Sigma^* \Rightarrow L_1 \cdot L_2 = L_1 L_2 = \{v \cdot w | v \in L_1, w \in L_2\}$

Definition 8 (Potenzen von Sprachen)**.**

$$\begin{aligned} L^0 &= L_\varepsilon = \{\varepsilon\} \\ L^{i+1} &= L^i \cdot L \text{ für } i \in \mathbb{N} \end{aligned}$$

Definition 9 (Kleene-Stern)**.**

$$\begin{aligned} L^* &= \bigcup_{i \in \mathbb{N}} L^i \\ L^+ &= \bigcup_{i \in \mathbb{N}} L^i - L_\varepsilon \end{aligned}$$

Beispiel. Sei $\Sigma = \{a, b\}, L_1 = \{a^i | i \geq 0\} = \{\varepsilon, a, aa, aaa, \dots\}, L_2 = \{b^i | i \geq 0\} = \{\varepsilon, b, bb, bbb, \dots\}$:

$L_1 \cdot L_2 = \{\varepsilon, a, b, ab, aab, abb, aaab, \dots\} = \{a^i b^j | i \geq 0, j \geq 0\}$ ist die Menge aller Wörter über $\{a, b\}$, in dem alle as vor allen bs vorkommen.

Übung. Seien $L_1, L_2, L_3 \subseteq \Sigma^*$.

(a) Gilt $(L_1 \cdot L_2) \cdot L_3 = L_1 \cdot (L_2 \cdot L_3)$? Ja.

(b) Gilt $(L_1 \cup L_2) \cdot L_3 = L_1 \cdot L_3 \cup L_2 \cdot L_3$? Ja.

(c) Gilt $(L_1 \cdot L_2) \cup L_3 = (L_1 \cup L_3) \cdot (L_2 \cup L_3)$? Quatsch.

(d) Gilt $L_1 \cdot L_2 = L_2 \cdot L_1$? Quatsch.

Beweis. (a)

$$\begin{aligned} (L_1 \cdot L_2) \cdot L_3 &= \{vw | v \in L_1 \cdot L_2, w \in L_3\} \\ &= \{xyw | x \in L_1, y \in L_2, w \in L_3\} \\ &= \{xz | x \in L_1, z \in L_2 L_3\} \\ &= L_1 \cdot (L_2 \cdot L_3) \end{aligned}$$

(b)

$$\begin{aligned} (L_1 \cup L_2) \cdot L_3 &= \{vw | v \in L_1 \cup L_2, w \in L_3\} \\ &= \{vw | v \in L_1, w \in L_3\} \cup \{vw | v \in L_2, w \in L_3\} \\ &= (L_1 \cdot L_3) \cup (L_2 \cdot L_3) \end{aligned}$$

(c) Gegenbeispiel: $L_1 = L_2 = \{a\}, L_3 = \{b\}$. Daraus folgt:

$$\begin{aligned} L_1 L_2 = \{aa\} &\Rightarrow (L_1 L_2) \cup L_3 = \{aa, b\} \\ L_1 \cup L_3 = \{a, b\} = L_2 \cup L_3 &= \Rightarrow \dots \end{aligned}$$

□

Bemerkung. (d) gilt aber für $|\Sigma| = 1$.

Definition 10 (Entscheidungsproblem)**.** Die Eingabe ist eine Sprache L über einem Alphabet Σ sowie ein Wort $w \in \Sigma^*$. Die Ausgabe soll lauten: JA falls $w \in L$ oder NEIN falls $w \notin L$.

Die Modellierung von vielen alltäglichen Berechnungsproblemen im Formalismus der formalen Sprachen.

Beispiel (Primzahltest)**.** Das Alphabet $\Sigma = \{0, 1\}$. Die Sprache

$$L = \{w \in \Sigma^* | w \text{ ist Binärdarstellung einer Primzahl}\}$$

Eine Zahl $p \in N$ ist Primzahl genau dann, wenn $Bin(p) \in L$.
Identifizierung von Sprachen als Probleme.

4 Kurze Einführung in formale Beweise

Behauptungen[2] mit Unanfechtbarkeitsanspruch wollen bewiesen werden. Insbesondere negative Aussagen der Form "Dieses Rechenmodell kann diese Aufgabe nicht lösen" brauchen eine präzise Formulierung und Begründung, um glaubwürdig zu sein[3]. Unkontrollierte Hausaufgabe

Als Beweismethoden haben wir im Angebot:

Deduktion (zum Beweis einer Implikation) Zum Beweis von Wenn-Dann-Aussagen, z.B.

$$\text{Wenn } \underbrace{x \geq 4}_{\text{Hypothese}} \text{ , dann } \underbrace{x^2 \geq 16}_{\text{Konklusion}}$$

Vorgehen: Hypothese als wahr annehmen, dann Folgerungen darauf anwenden, bis die Konklusion folgt.

Beispiel. Sei $x \geq 4$. Es gilt $4^2 = 16$ und die Quadratfunktion ist steigend. Daraus folgt: $x^2 \geq 4^2 = 16$

Hierfür ist es oft hilfreich, Definitionen von Begriffen einzusetzen.

Beispiel. Wenn $L = \{w \in \{a, b\}^* | |w| \text{ gerade}\}$, dann gilt für alle $x \in L^2$, dass $|x|$ gerade ist.

Beweis. Sei $x \in L^2$.

Definition von L^2 einsetzen. Es existieren $u, v \in L$, so dass $x = uv$.

Definition von L einsetzen. $|u|$ ist gerade, $|v|$ ist gerade.

$\Rightarrow |x| = |u| + |v| \Rightarrow |x|$ ist gerade. □

[2] 11. September 2012
[3] Hopcraft et al., Abschnitt 1.2–1.4

Doppelte Deduktion (zum Beweis einer Äquivalenz) Zerlegung in zwei Wenn-Dann-Aussagen, wird getrennt bewiesen.

Beispiel (Gleichheit von Mengen)**.**

$$A = B \Rightarrow A \subseteq B \wedge B \subseteq A \Rightarrow A = B$$

Beispiel. Sprachen sind ja auch Mengen. Also: seien $L_1, L_2 \in \Sigma^*$. Dann gilt:

$$L_1 \cdot (L_1 \cup L_2) = L_1^2 \cup L_1 \cdot L_2$$

Beweis. Zweimal:

"$\subseteq$":
Sei $x \in L_1 \cdot (L_1 \cup L_2) \Rightarrow \exists x = uv, u \in L_1, v \in L_1 \cup L_2$.
Fallunterscheidung: (a) $v \in L_1 \Rightarrow x = uv, u \in L_1, v \in L_1 \Rightarrow x \in L_1^2$.
(b) $v \in L_2 \Rightarrow x = uv, u \in L_1, v \in L_2 \Rightarrow x \in L_1 \cdot L_2$.

- "$\supseteq$":
Sei $x \in L_1^2 \cup L_1 \cdot L_2$.
Fallunterscheidung: (a) $x \in L_1^2 \Rightarrow x = uv, u, v \in L_1$
$\Rightarrow v \in L_1 \cup L_2$
$\Rightarrow x \in L_1 \cdot (L_1 \cup L_2)$.
(b) $x \in L_1 \cdot L_2 \Rightarrow x = uv, u \in L_1, v \in L_2$
$\Rightarrow v \in (L_1 \cup L_2)$
$\Rightarrow x \in L_1 \cdot (L_1 \cup L_2)$

□

Widerspruchsbeweis Es wird eine Annahme getroffen und dann solange gefolgert, bis Quatsch herauskommt. Daraus folgt, dass die Annahme auch falsch sein muss.

Beispiel. Seien $a, b \in \mathbb{N}$ und $a, b \geq 2$. Dann gilt: a teilt nicht $ab + 1$.

Beweis. Seien $a, b \in \mathbb{N}$. Angenommen, a teilt $ab + 1$.

Ziel: Annahme zum Widerspruch führen, dann folgt daraus Negation, also die gewünschte Konklusion.

a teilt $ab \Rightarrow a$ teilt ab und $ab + 1$.

$\Rightarrow a$ teilt $(ab + 1) - (ab) = 1$

$\Rightarrow a = 1$. Bonk! Widerspruch zu $a \geq 2$. □

Beispiel. Es gibt unendlich viele Primzahl (äquivalente Formulierung: Es gibt keine grösste Primzahl).

Beweis. Angenommen, es gäbe eine grösste Primzahl. Wir nennen die Anzahl der Primzahlen k.

Seien $p_1 < p_2 < \cdots < p_k$ die Primzahlen. Betrachten wir $x = p_1 \cdot p_2 \cdot \cdots \cdot p_k + 1$. Da p_k die grösste Primzahl ist und $x > p_k$, kann x keine Primzahl sein.

Also gibt es eine Primfaktorzerlegung von x, und p_l sei die kleinste Primzahl in dieser Zerlegung.

Dann gilt: p_l teilt $p_1 \cdot p_2 \cdot dots \cdot p_k$ und p_l teilt darum auch x.

Daraus folgt: p_l teilt $p_1 \cdot p_2 \cdot dots \cdot p_k + 1$. Aus dem vorhergehenden Beweis (a teilt $ab + 1$) folgt, dass das ein Widerspruch ist. □

Induktion Vor allem verwendet für Beweise über natürliche Zahlen. Ziel: Zeige, dass die Aussage $S(n)$ für alle $n \geq n_0$ gilt.

Methode:

1. Induktionsanfang (Anker). Zeige $S(n_0)$.
2. Induktionsschritt. Zeige $S(i) \Rightarrow S(i+1)$.

Idee: Wenn $S(n_0)$ gilt und $S(i) \Rightarrow S(i+1)$ für alle $i \geq n_0$, dann folgt aus $S(n_0)$ und $S(n_0) \Rightarrow S(n_0+1)$, dann gilt $S(n_0+1)$. Undsoweiter, undsofort.

Beispiel (Der kleine Gauss)**.** . Für alle $n \geq 1$ gilt: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Beweis. Anker: $\sum_{i=1}^{1} = \frac{1+(1+1)}{2}$

Schritt: es gelte $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$ (Induktionsvoraussetzung).

Zeige: $\sum_{i=1}^{k+1} i = \frac{(k+1)(k+1+1)}{2}$ (Induktionsbehauptung).

$$\begin{aligned}
\sum_{i=1}^{k+1} i &= \sum_{i=1}^{k} i + (k+1) \\
&= \frac{k(k+1)}{2} + (k+1) \\
&= \frac{k(k+1) + 2(k+1)}{2} \\
&= \frac{(k+2)(k+1)}{2} \\
&= \frac{(k+1+1)(k+1)}{2}
\end{aligned}$$

□

Rekursive Definitionen und strukturelle Induktion

Beispiel (Rekursive Definition von arithmetischen Ausdrücken)**.** Basis: Jede Zahl und jede Variable ist ein arithmetischer Ausdruck: $(a+3)\cdot 5$ ist ein arithmetischer Ausdruck, $(a+3\cdot 5$ [sic] ist keiner (Klammerfehler).

Rekursion: Wenn E und F Ausdrücke sind, dann definieren wir, dass dann auch $E+F$, $E\cdot F$, (E) arithmetische Ausdrücke sind.

Behauptung: Jeder Ausdruck enthält gleichviele linke wie rechte Klammern.

Beweis. Mit struktureller Induktion:

1. I.A.: Basisausdrücke haben keine Klammern. Daraus folgt: gleichviele linke wie rechte Klammern (0).
2. Sei G ein arithmetischer Ausdruck ($G = E+F, E\cdot F, (G)$). Fallunterscheidung:
   (a) $G = E + F$: Induktionsvoraussetzung: E und F haben gleich viele linke wie rechte Klammern. Gilt also auch für G, da keine Klammern hinzukommen.
   (b) $G = E \cdot F$: analog.
   (c) (G): Induktionsvoraussetzung: E hat gleich viele linke wie rechte Klammern. Gilt also auch für G, da 1 linke und 1 rechte Klammer hinzukommt.

□

5 Unendlichkeit

Beweis für 3.7. Behauptung: $|A| < |B| \Leftarrow |A| = |C| \wedge C \subset B$
$A = B = \mathbb{N} = \{0, 1, 2, 3, \dots\}$ $C = \{1, 2, 3, 4, \dots\}$ $(0,1), (1,2), \dots, (k, k+1)$ □

Aufgaben: 3.7, 3.8, 3.11, 3.17, 3.18, 3.19

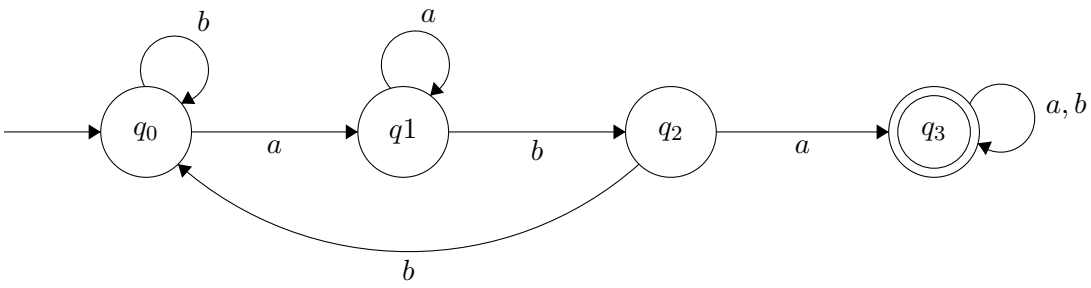
6 Endliche Automaten

Unterscheidung:

Deterministisch Automat kann zu einem Zeitpunkt nur einen Zustand annehmen.

Nichtdeterministisch Automat kann gleichzeitig mehrere Zustände besitzen.

Beispiel. DEA zum Erkennen des Musters aba:



Zustände speichern das bereits gelesene Teilmuster:

- q_0: noch nichts vom Teilwort aba gelesen.
- q_1: zumindest schon das Wort a gelesen.
- q_2: schon ab gelesen.
- q_3: das ganze Muster gelesen, es kann also akzeptiert werden.

Transitionen (Zustandsübergänge£) beschreiben die Änderung beim Lesen eines Zeichens des Wortes.

Definition 11 (Formale Definition (nochmal))**.** Ein DEA A wird beschrieben durch $A = (Q, \Sigma, \delta, q_0, F)$, wobei:

- Q ist eine endliche Menge von Zuständen,
- Σ ist das Alphabet für die Eingabe,
- q_0 ist der Startzustand, $q_0 \in Q$,
- F ist die Menge der akzeptierenden Zustände (Endzustände), $F \subseteq Q$,
- $\delta : Q \times \Sigma \mapsto Q$ ist die Übergangsfunktion (Transitionsfunktion), die für jeden Zustand und jedes Eingabesymbol einen eindeutigen Folgezustand definiert.

Beispiel (von vorher)**.**

$$A = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_3\}),$$

wobei δ gegeben ist durch:

$$\begin{aligned}\delta(q_0, a) &= q_1,\\ \delta(q_0, b) &= q_0,\\ \delta(q_1, a) &= q_1,\\ \delta(q_1, b) &= q_2,\\ \delta(q_2, a) &= q_3,\\ \delta(q_2, b) &= q_0,\\ \delta(q_3, a) &= q_3,\\ \delta(q_3, b) &= q_3\end{aligned}$$

(oder als Transitionstabelle aufgeschrieben):

	a	b
$\mapsto q_0$	q_1	q_0
q_1	q_1	q_2
q_2	q_3	q_0
$*q_3$	q_3	q_3

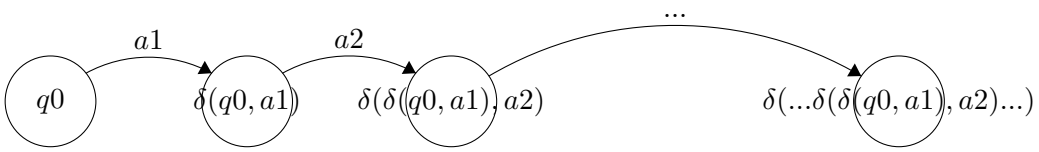
Die Transitionstabelle enthält alle Informationen über einen DEA, meistens ist aber die graphische Darstellung übersichtlicher.

Ziel: Automaten zur Beschreibung von Sprachen verwenden.

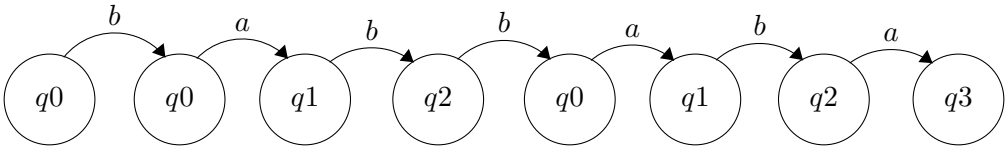
Definition 12 (Berechnung)**.** Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA, $w \in \Sigma^*$ ein Wort.

Die Berechnung von A auf w ist die Folge von Zuständen, die A beim Lesen von w durchläuft, also, gilt für $w = a_1 a_2 \dots a_k$:

$$q_0 \overset{a_1}{\curvearrowright} \delta(q_0, a_1) \overset{a_2}{\curvearrowright} \delta(\delta(q_0, a_1), a_2) \overset{\cdots}{\curvearrowright} \delta(\dots\delta(\delta(q_0, a_1), a_2)\dots)$$



Beispiel. Berechnung von A auf $w = babbaba$:



Sinnvollerweise: Erweiterung von δ af Wörter (erweiterte Übergangsfunktion).

Definition 13 (Erweiterte Übergangsfunktion)**.** $\hat{\delta} : Q \times \Sigma^* \mapsto Q$ ist rekursiv definiert wie folgt:

- $\hat{\delta}(q, a) = \delta(q, a)$ für alle $q \in Q, a \in \Sigma$
- $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$ für alle $q \in Q, x \in \Sigma^*, a \in \Sigma$.

Bemerkung. Anschaulich: $\hat{\delta}(q, w)$ betstimmt, in welchem Zustand A endet, wenn er vom Zustand q aus das Wort w liest.

Beispiel. $\hat{\delta}(q_0, babbaba)$

Definition 14. Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA. Die von A akzeptierte Sprache $L(A) \subseteq \Sigma^*$ ist die Menge aller $w \in \Sigma^*$, so dass $\hat{\delta}(q_0, w) \in F$.

Bemerkung. Anschaulich: Die Menge aller Wörter, die den DEA vom Anfangszustand aus in einen Endzustand führen.

Beispiel. $\hat{\delta}(q_0, babbaba) = q_3 \in F \Rightarrow w \in L(A)$ (w wird von A akzeptiert.)

Definition 15 (Reguläre Sprachen)**.** $\mathcal{L}(\text{DEA}) = \{L(A) | A \text{ ist ein DEA}\}$ ist die Klasse der Sprachen, die von DEAs akzeptiert werden. Man nennt sie die Klasse der regulären Sprachen und $L \in \mathcal{L}(\text{DEA})$ wird regulär genannt.

Beispiel. DEA A entwerfen, der die Sprache

$L = \{w \in \{0, 1\}^* | w \text{ enthält eine gerade Anzahl Nullen und eine gerade Anzahl Einsen}\}$

akzeptiert.

Idee: 4 Zustände, die speichern, wieviele Nullen und Einseln modulo 2 bereits gelesen wurden.

- q_{00}: gerade Anzahl Nullen, gerade Anzahl Einsen
- q_{01}: gerade Anzahl Nullen, ungerade Anzahl Einsen
- q_{10}: ungerade Anzahl Nullen, gerade Anzahl Einsen
- q_{11}: ungerade Anzahl Nullen, ungerade Anzahl Einsen

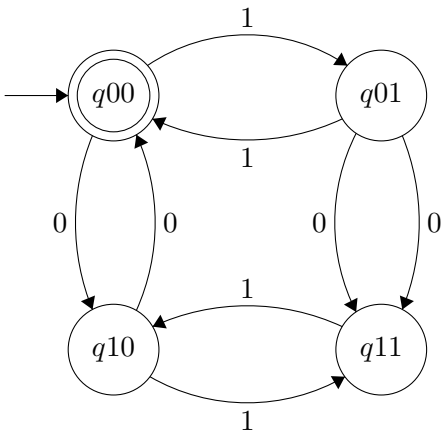
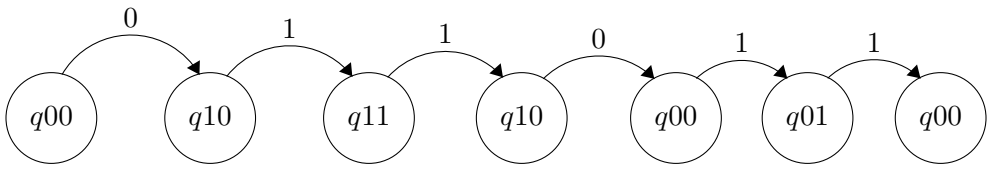


Tabelle:

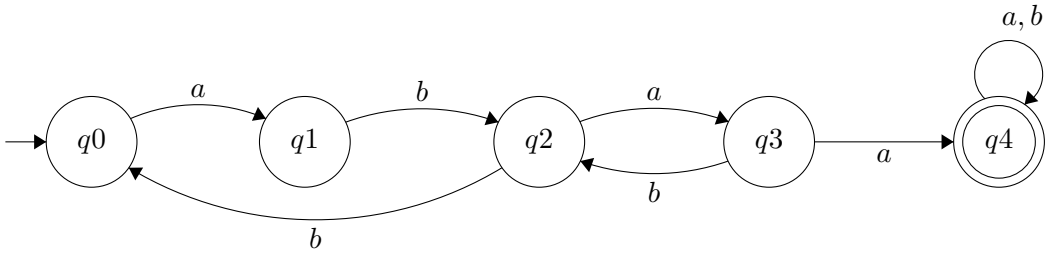
	0	1
$\mapsto *q_{00}$	q_{10}	q_{01}
q_{01}	q_{11}	q_{00}
q_{10}	q_{00}	q_{11}
q_{11}	q_{01}	q_{10}

Berechnung von A auf $w = 011011$:

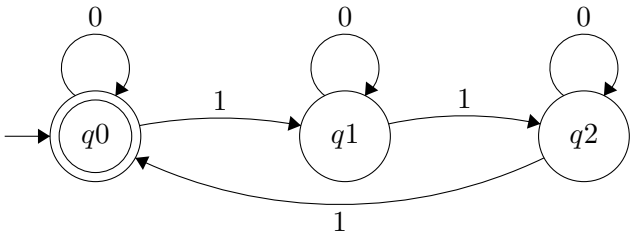


$\Rightarrow w \in L(A), \hat{\delta}(q_{00}, 011011) = q_0$

Lösung der Aufgaben
1a:



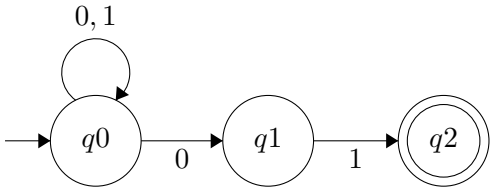
1b:



7 Nichtdeterministische erstaunliche Endliche Automaten (NEA)

Idee: Ein NEA kann sich nach dem Lesen eines Wortes in mehreren Zuständen befinden.
Vorteil: Sie sind einfacher zu entwerfen.

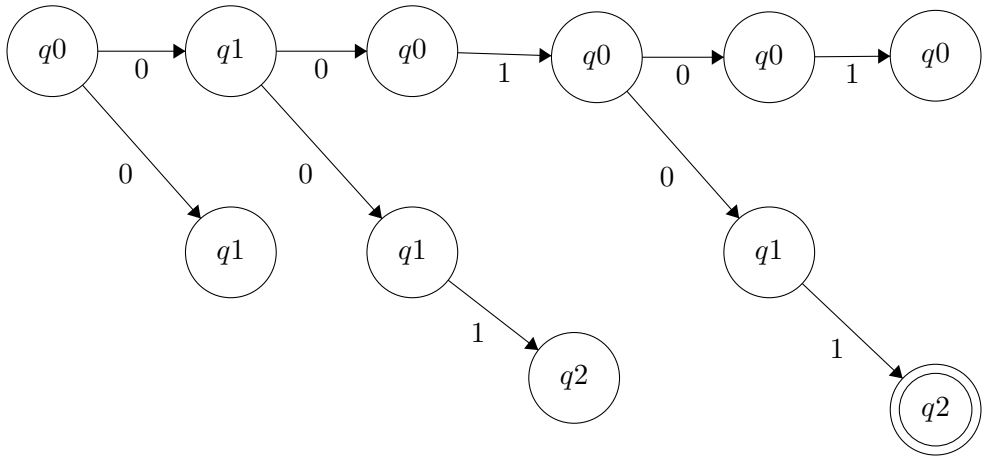
Beispiel. Akzeptiere alle Wörter, die mit 01 enden.



Transitionstabelle:

	0	1
$\mapsto q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\varnothing$	$\{q_2\}$
$*q_2$	$\varnothing$	$\varnothing$

Mögliche Berenchungen auf dem Wort 00101:



Definition 16 (NEA)**.** Ein NEA A wird beschrieben durch $A = (Q, \Sigma, \delta, q_0, F)$, wobei

$$\delta : Q \times \Sigma \mapsto 2^Q = \mathcal{P}(Q)$$

Die Transitionsfunktion definiert für jeden Zustand und jedes Eingabesymbol eine Menge von Folgezuständen.

Definition 17 (Erweiterte NEA-Transitionsfunktion)**.** Sei $w \in \Sigma^*, q \in Q$. Dann definieren wir

$$\hat{\delta} : Q \times \Sigma^* \mapsto 2^Q$$

rekursiv wie folgt:

- $\hat{\delta}(q, \varepsilon) = \{q\}$
- Sei $w = xa$ und $\hat{\delta}(q, x) = \{p_1, \ldots p_k\}$ für $k \in \mathbb{N}$, dann ist

$$\hat{\delta}(q, w) = \bigcup_{i=1}^{k} \delta(p_i, a)$$

Beispiel.

$$\begin{aligned}
\hat{\delta}(q_0, \varepsilon) &= \{q_0\} \text{ (Grundregel)} \\
\hat{\delta}(q_0, 0) = \delta(q_0, 0) &= \{q_0, q_1\} \\
\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \varnothing &= \{q_0, q_1\} \\
\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} &= \{q_0, q_2\} \\
\hat{\delta}(q_0, 0010) = \delta(q_0, 0) \cup \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_1\} \cup \{q_2\} &= \{q_0, q_2\} \text{ falsch} \\
\hat{\delta}(q_0, 00101) = \delta(q_0, 0) \cup \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_1\} \cup \{q_2\} &= \{q_0, q_2\} \text{ falsch}
\end{aligned}$$

Definition 18. Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein NEA. Die von A akzeptierte Sprache $L(A)$ ist $L(A) = \{w \in \Sigma^* | \hat{\delta}(q_0, w) \cap F \neq \varnothing\}$, also die Menge aller Wörter, mit denen vom Startzustand aus ein Endzustand erreichbar ist.

Fragen/Probleme:

1. Wie kann man möglichst effizient entscheiden, ob ein gegebenes Wort in der Sprache eines gegebenen NEA enthalten ist?

2. Sind NEAs mächtiger als DEAs?

Antwort: DEAs und NEAs sind äquivalent.
Ziel: Automatische Umwandlung von NEAs in DEAs.

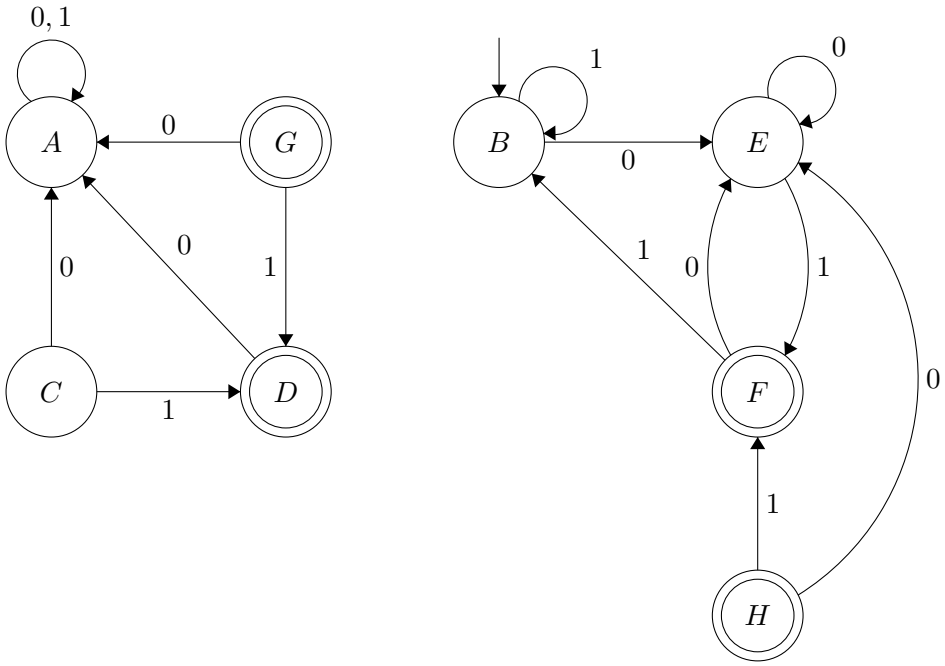
Bemerkung (Teilmengenkonstruktion (Potenzmengenkonstruktion))**.** Idee: Zustände des DEA sind Mengen erreichbarer Zustände des NEA.

Formal: Sei $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ ein NEA. Konstruiere einen äquivalenten DEA $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ mit

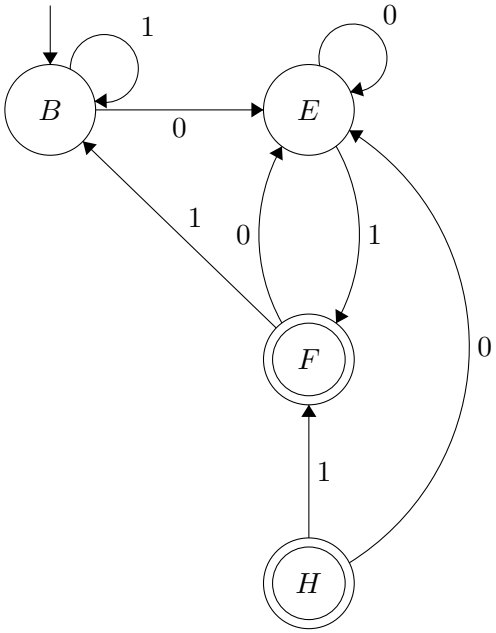
- $Q_D = 2^{Q_N}$
- $F = \{S \subseteq Q_D | S \cap F \neq \varnothing\}$
- $\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$ für $S \in Q_D, a \in \Sigma$

Beispiel (von vorher)**.** Konstruktion des äquivalenten DEAs als Übergangstabelle.

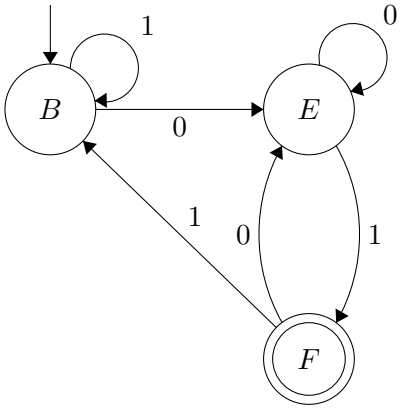
		0	1
A	$\varnothing$	$\varnothing = A$	$\varnothing = A$
B	$\mapsto \{q_0\}$	$\{q_0, q_1\} = E$	$\{q_0\} = B$
C	$\{q_1\}$	$\varnothing = A$	$\{q_2\} = D$
D	$*\{q_2\}$	$\varnothing$	$\varnothing$
E	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
F	$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
G	$*\{q_1, q_2\}$	$\varnothing$	$\{q_2\}$
H	$*\{q_1, q_2, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



Der linke Teil ist nicht erreichbar:



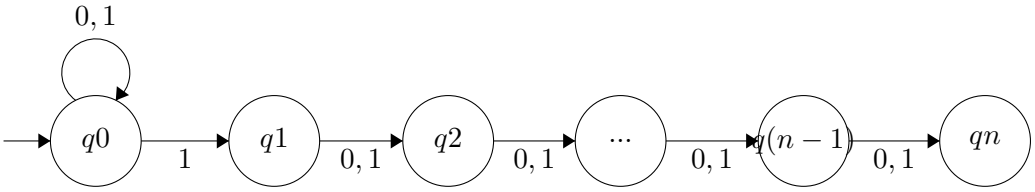
H ist ebenfalls nicht erreichbar:



Beispiel (Ein ungünstiger Fall für die Teilmengenkonstruktion)**.**

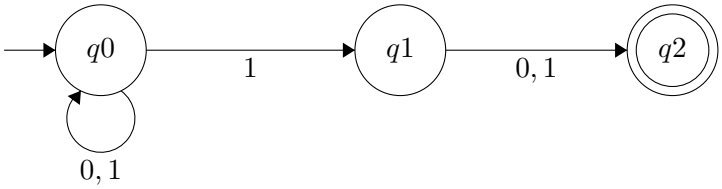
$$\begin{aligned} L_n &= \{w \in \{0,1\}^* | \text{ Das } n\text{-t-letzte Zeichen von } w \text{ ist eine } 1\} \\ &= \{w \in \{0,1\}^* | w = x1y \text{ mit } x \in \{0,1\}^*, y \in \{0,1\}^{n-1}\} \end{aligned}$$

NEA für L_n mit $n+1$ Zuständen:



Aber jeder DEA für L_n braucht mindestens 2^n Zustände, weil der DEA sich an die letzten n gelesenen Symbole erinnern muss, da er nicht weiss, wann das Wortende kommt. D.h. 2^n verschiedene Teilworte mpssen unterscheiden werden. D.h. 2^n Zustände nötig.

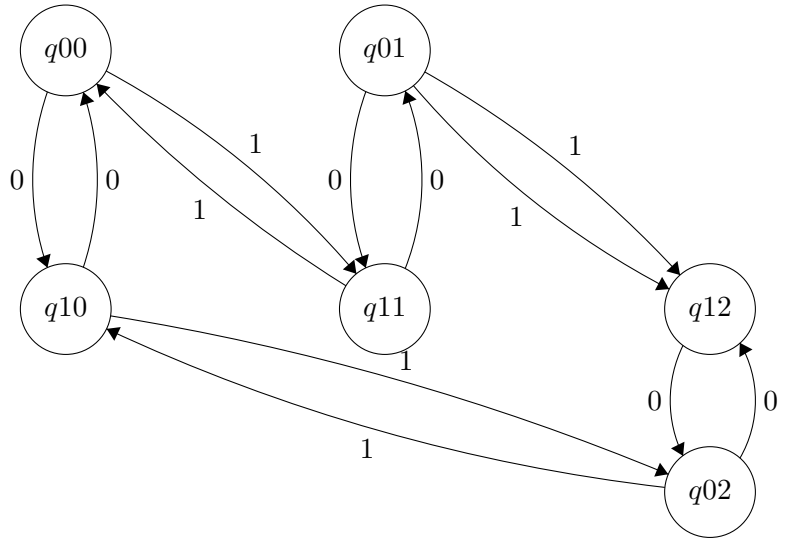
Beispiel mit $n = 2$:



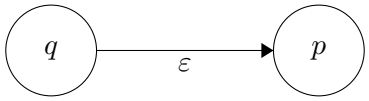
Beispiel (Lösung der Knobelaufgabe)**.** Wir machen uns zunutze:

$$2^k \mod 3 = \begin{cases} 1 \text{ falls } k \text{ gerade} \\ 1 \text{ falls } k \text{ gerade} \end{cases}$$

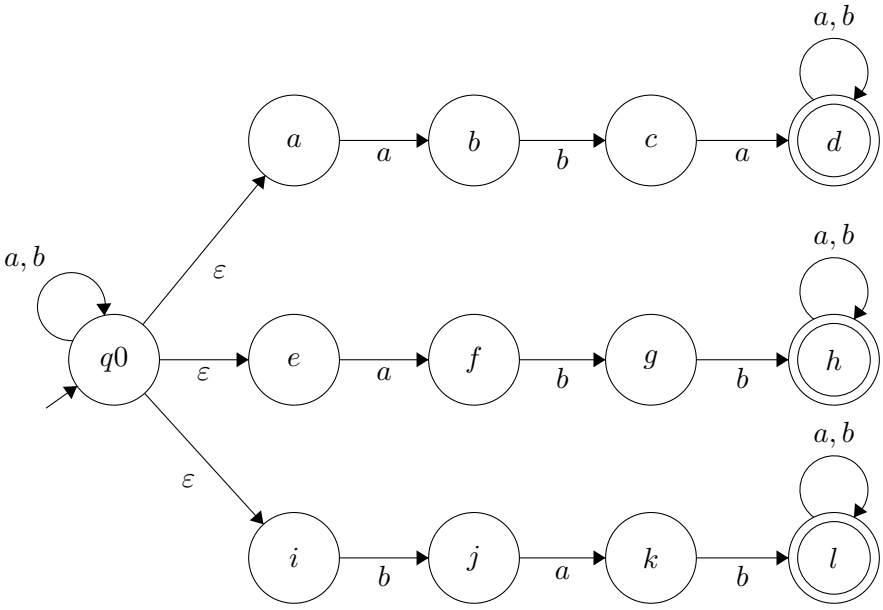
Wir haben Zustände q_{ij}. i ist der Exponent der nächsten 2er-Potenz modulo 2, j ist die Teilsumme.



8 NEAs mit ε-Übergängen



Beispiel. Suche nach mehreren Mustern. $L = \{w \in \{a,b\}^* |$ enthält eines der Teilwörter aba, abb, bab $\}$
Der NEA N mit $L(N) = L$ mit ε-Übergängen sieht aus:

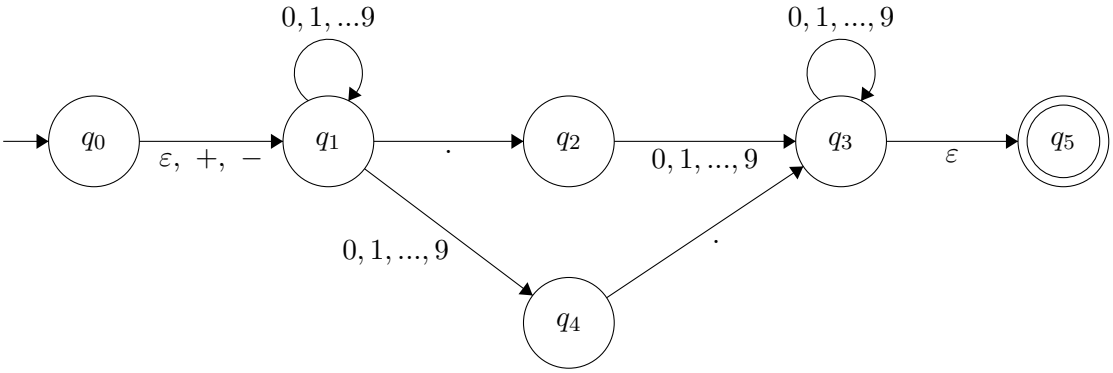


Beispiel. Ein ε-NEA, der Dezimalzahlen akzeptiert, die sich zusammensetzen aus:

- optionales + oder −
- Zeichenreihe von Ziffern
- Dezimalpunkt
- Zeichenreihe von Ziffern

Eine der beiden Zeichenreihen von Ziffern darf leer sein.

Der Automat E sieht dann so aus:



Definition 19 (NEA mit ε-Übergängen)**.** Ein NEA mit ε-Übergängen A wird beschrieben durch $A = (Q, \Sigma, \delta, q_0, F)$, wobei Q, Σ, q_0, F wie beim DEA definiert sind, und

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \mapsto 2^Q$$

die Transitionsfunktion ist.

Bemerkung. Annahme: $\varepsilon \notin \Sigma$

Beispiel (von vorher)**.**

$$E = (\{q_0, q_1, \ldots, q_5\}, \{., +, -, 0, 1, \ldots, 9\}, \delta, q_0, \{q_5\})$$

Übergangstabelle:

	ε	$+, -$	.	$0, 1, \ldots, 9$
q_0	$\{q_1\}$	$\{q_1\}$	$\varnothing$	$\varnothing$
q_1	$\varnothing$	$\varnothing$	$\{q_2\}$	$\{q_1, q_4\}$
q_2	$\varnothing$	$\varnothing$	$\varnothing$	$\{q_3\}$
q_3	$\varnothing$	$\varnothing$	$\{q_3\}$	$\varnothing$
q_4	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
q_5	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$

9 ε-Hüllen

Ziel: Zeigen, dass es zu jedem ε-NEA einen äquivalenten DEA gibt.

Idee: Finde für jeden Zustand die Menge aller Zustände, die von dort aus nur mit ε-Übergängen erreichbar sind.

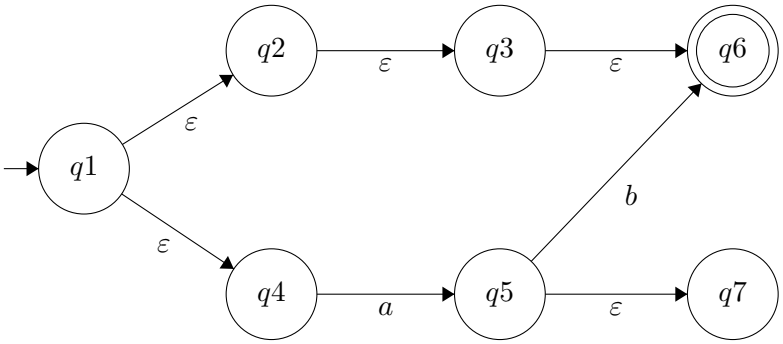
Definition 20 (Rekursive Definition)**.** Die ε-Hülle eines Zustands $q \in Q$ wird bezeichnet mit ε-Hülle(q) und ist definiert durch:

- $q \in \varepsilon$-Hülle(q)
- Falls $p \in \varepsilon$-Hülle(q), dann auch jeder Zustand r, so dass $r \in \delta(p, \varepsilon)$

Beispiel (Von vorher)**.**

$$\begin{aligned}
\varepsilon\text{-Hülle}(q_0) &= \{q_0, q_1\} \\
\varepsilon\text{-Hülle}(q_1) &= \{q_1\} \\
\varepsilon\text{-Hülle}(q_2) &= \{q_2\} \\
\varepsilon\text{-Hülle}(q_3) &= \{q_3, q_5\} \\
\varepsilon\text{-Hülle}(q_4) &= \{q_4\} \\
\varepsilon\text{-Hülle}(q_5) &= \{q_5\}
\end{aligned}$$

Beispiel (Noch eines)**.** . Noch eines:



Bestimme E-Hülle(q1):

1. $q_1 \in \varepsilon - Hulle(q_1)$
2. Finde alle Zustände aus $\delta(q_1, \varepsilon) = \{q_2, q_4\} \Rightarrow \{q_2, q_4\} \subseteq \varepsilon - Hulle$

Definition 21 (ε-Hülle einer Menge S von Zuständen)**.**

$$\varepsilon\text{-Hülle}(S) = \bigcup_{q \in S} \varepsilon\text{-Hülle}(q)$$

10 Erweiterung der Übergangsfunktion auf Wörter

Finde alle Pfade im Graphen, die mit dem Wort w beschriftet sind (mit beliebig vielen ε dazwischen).

Definition 22 (Rekursive Definition)**.** Nun denn:

- $\hat{\delta}(q, \varepsilon) = \varepsilon$-Hülle$(q)$
- $\hat{\delta}(q, xa)$ wird für $x \in \Sigma^*$ und $a \in \Sigma$ definiert durch:
  1. Sei $\hat{\delta}(a, x) = \{p_1, \ldots, p_k\}$
  2. Sei $\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, \ldots, r_m\}$
  3. Dann gilt $\hat{\delta}(q, xa) = \varepsilon$-Hülle$(\{r_1, \ldots, r_m\}) = \bigcup_{i=1}^{m} \varepsilon$-Hülle$(r_i)$

Beispiel ($\hat{\delta}(q_0, 5{,}6)$)**.** hmm...

1. $\hat{\delta}(q_0, \varepsilon) = \varepsilon$-Hülle$(q_0) = \{q_0, q_1\}$
2. $\hat{\delta}(q_0, 5)$: $\delta(q_0, 5) \cup \delta(q_1, 5) = \varnothing \cup \{q_1, q_4\}$
   $\hat{\delta}(q_0, 5) = \varepsilon$-Hülle$(q_1) \cup \varepsilon$-Hülle$(q_4) = \{q_1, q_4\}$
3. $\hat{\delta}(q_0, 5.)$: $\delta(q_1, .) \cup \delta(q_4, .) = \{q_2\} \cup \{q_3\} = \{q_2, q_3\}$
   $\hat{\delta}(q_0, 5.) = \varepsilon$-Hülle$(q_2) \cup \varepsilon$-Hülle$(q_3) = \{q_2, q_3, q_5\}$
4. $\hat{\delta}(q_0, 5.6)$: $\delta(q_2, 6) \cup \delta(q_3, 6) \cup \delta(q_5, 6) = \{q_2\} \cup \{q_3\} = \{q_2, q_3\}$
   $\hat{\delta}(q_0, 5.) = \varepsilon$-Hülle$(q_2) \cup \varepsilon$-Hülle$(q_3) = \{q_2, q_3, q_5\}$ ...

Definition 23. Für einen ε-NEA $E = (Q, \Sigma, \delta, q_0, F)$ ist $L(E) = \{w | \hat{\delta}(q_0, w) \cap F \neq \varnothing\}$.

11 Umwandlung von ε-NEAs in DEAs

Schritt 1: ε-Übergänge eliminieren

Sei $E = (Q_E, \Sigma, \delta_E, q_0, F_E)$ ein ε-NEA. Konstruiere einen äquivalenten DEA $D = (Q_D, \Sigma, \delta_D, q_{0,D}, F_D)$.

- $Q_0 = 2^{Q_E}$. Alle erreichbaren Zustände sind ε-abgeschlossene Teilmengen, das heisst Mengen $S \subseteq Q_E$, so dass $S = \varepsilon$-Hülle(S). D.h. jeder ε-Übergang von einem Zustand $q \in S$ führt zu einem Zustand, der auch in S liegt.
- $q_{0,D} = \varepsilon$-Hülle(q_0)
- $F_D = \{S \in Q_D | S \cap F_E \neq \varnothing\}$
- $\delta_D(S, a)$ wird für alle $S \in Q_D$ und $a \in \Sigma$ wie folgt berechnet:

1. Sei $S = \{p_1, \ldots, p_k\}$
2. Berechne $\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, \ldots, r_m\}$
3. Dann ist $\delta_D(S, a) = \bigcup_{j=1}^{n} \varepsilon\text{-Hüllen}(r_j)$

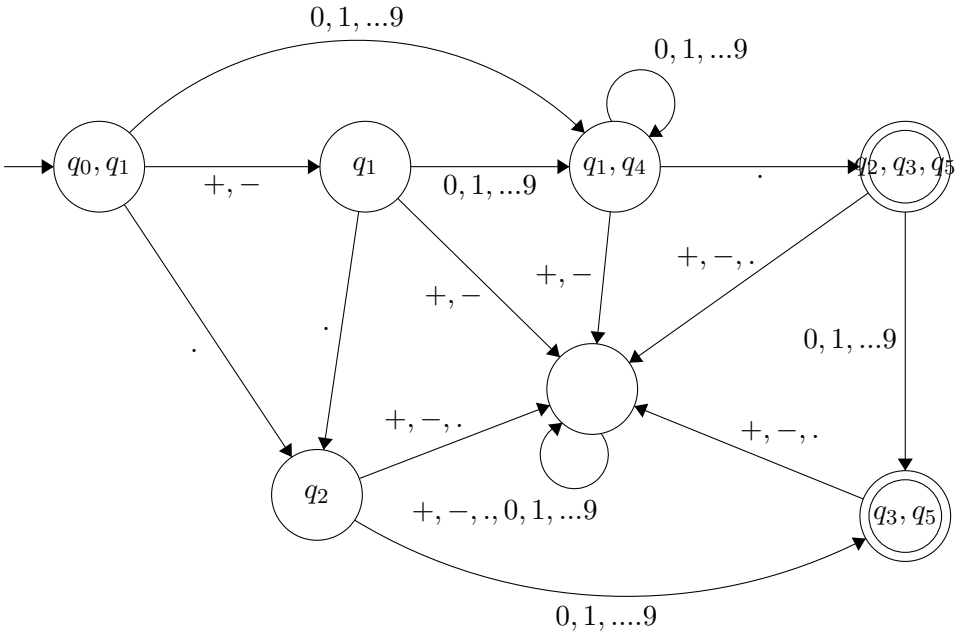
Beispiel (Von vorher)**.** Äquivalenter DEA D: Startzustand ist ε-Hülle$(q_0) = \{q_0, q_1\}$
Finde die Folgezustände von $\{q_0, q_1\}$ für alle Symbole des Alphabets:

$+, -$ Kein Übergang von q_1 aus, aber von q_0 nach q_1. Daraus folgt ε-Hülle$(q_1) = \{q_1\}$.

. Kein Übergang von q_0, aber von q_0 nach q_2. Daraus folgt ε-Hülle$(q_2) = \{q_2\}$.

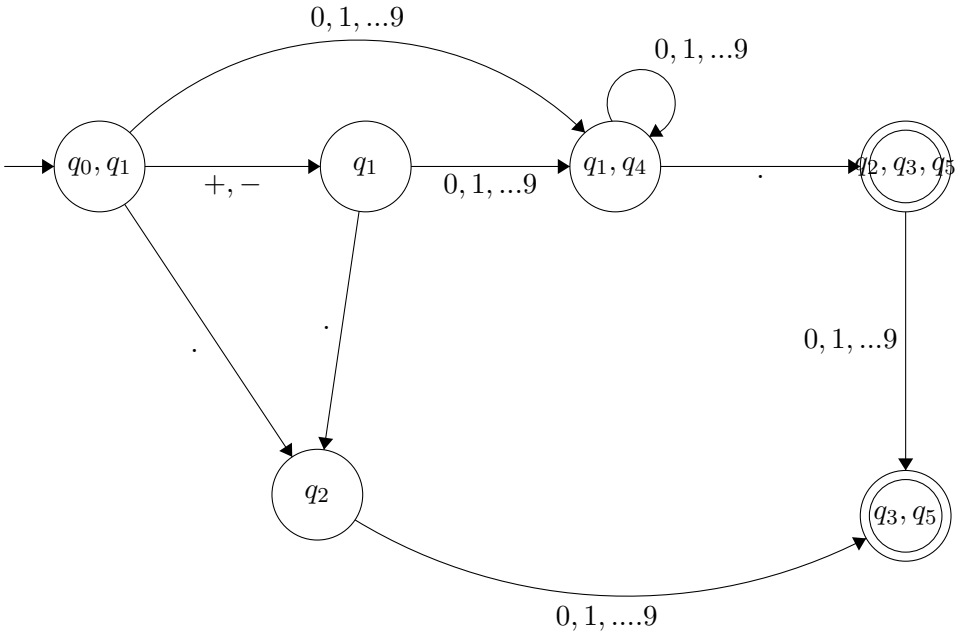
$0, 1, \ldots, 9$ Kein Übergang von q_0, aber von q_1 nach q_1 oder q_4. Daraus folgt ε-Hülle$(q_1, q_4) = \{q_1, q_4\}$

Transitionstabelle (siehe Excel =¿ transferieren)!

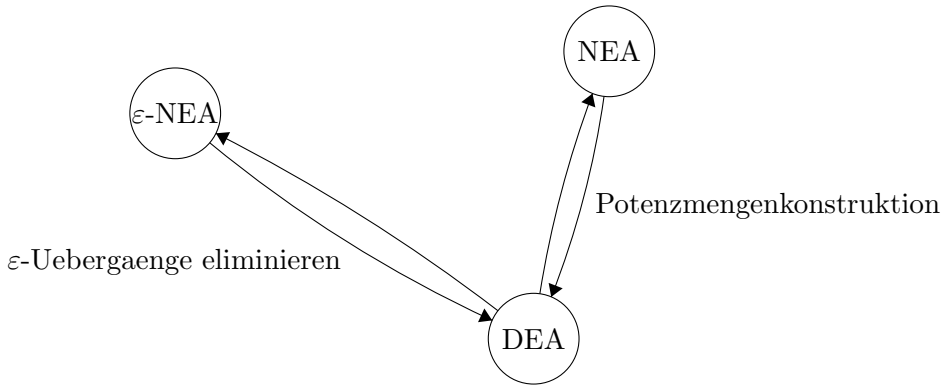


Bemerkung (Konvention)**.** Die da wären:

- Der Zustand $\varnothing$ wird Senkezustand (Abfallzustand) genannt und darf weggelassen werden.
- Darstellung eines DEA mit fehlenden Transitionen sind immer so zu verstehen, dass die fehlenden Transitionen in einen solchen Senkezustand führen:



Zusammenfassung: DEAs und ε-NEAs sind äquivalent.



12 Reguläre Ausdrücke

Besonders geeignet für die Mustersuche, anderer Formalismus zur Beschreibung von regulären Sprachen.

13 Operatoren für reguläre Ausdrücken

- Die Vereinigung von Sprachen L und M, $L \cup M$, ist die Menge aller Zeichenreihen, die entweder in L oder in M oder in beiden enthalten sind.

- Die Verkettung (Konkatenation) von L und M, $L \cdot M$ oder LM, ist die Menge aller Zeichenreihe, gebildet aus einer Verkettung einer Zeichenreihe aus L umit einer beliebigen aus M.

  Formal:
  $$LM = \{w \in \Sigma^* | w = uv \text{ mit } u \in L \text{ und } v \in M\}$$

- Die Kleensche Hülle (Stern) von L, L^*, ist die Menge aller Zeichenreihen, die durch die Verkettung einer beliebigen Anzahl von Zeichenreihen aus L gebildet wird.

  Formal:
  $$\begin{aligned} L^0 &= \{\varepsilon\} \\ L^1 &= L \\ L^n &= L \cdot L^{n-1} \\ L^* &= \bigcup_{i=0}^{\infty} L^i \end{aligned}$$

  Anschaulich: $w \in L^*$, wenn sich w zerlegen lässt in endlich viele Wörter $w = w_1 w_2 \dots w_k$ für ein $k \in \mathbb{N}$, so dass $w_1 \in L_1, w_2 \in L_2 \dots w_i L_i$ gilt.

  Spezialfälle:

  - $\varnothing^* = \{\varepsilon\}$, da $\varnothing^0 = \{\varepsilon\}$
  - $\{\varepsilon\}^* = \{\varepsilon\}$

  Aber: $\varnothing^i = \varnothing$ für $i > 1$.

Definition 24 (Reguläre Ausdrücke)**.** Reguläre Ausdrücke lassen sich wie folgt definieren:

1. ε und $\varnothing$ sind reguläre Ausdrücke.
2. Jedes Symbol $a \in \Sigma$ ist ein regulärer Ausdruck.
3. Wenn α und β reguläre Ausdrücke sind, dann sind:
   - $(\alpha \cdot \beta)$ (Konkatenation)
   - $(\alpha + \beta)$ (Vereinigung)

   reguläre Ausdrücke.
4. Wenn α ein regulärer Ausdruck ist, dan ist auch $(\alpha)^*$ ein regulärer Ausdruck. (Kleenscher Stern)

Bemerkung (Konvention)**.** Man tut:

- Überflüssige Klammern weglassen
- Stern bindet stärker als Verkettung
- Verkettung bindet stärker als Vereinigung

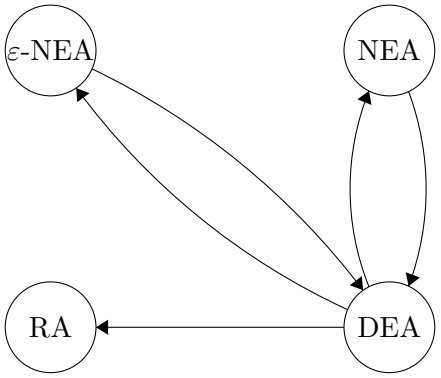
14 Bedeutung regulärer Ausdrücke

$L(\alpha)$ bezeichnet die durch den regulären Ausdruck beschriebene Sprache. Dabei gilt:

- $L(\varepsilon) = \{\varepsilon\}$
- $L(\varnothing) = \varnothing$
- $L(a) = \{a\}$
- $L(\alpha \cdot \beta) = L(\alpha) \cdot L(\beta)$
- $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$
- $L(\alpha^*) = (L(\alpha))^*$

Beispiel (Beschreibung von Sprachen durch reguläre Ausdrücke)**.** Nun:

1. $L_1 = \{w \in \{0,1\}^* | w$ beginnt mit 011 und enthält das Teilwort 00 $\}$. RA für L_1: $011 \cdot (0+1)^* \cdot 00 \cdot (0+1)^*$
2. $L_2 = \{w \in \{a,b,c\}^* || w|_a$ ist gerade $\}$.
   RA für L_2: $((b+c)^* a (b+c)^* a (b+c)^*)^*$
3. $L_3 = \{w \in \{0,1\}^* |$ In w kommen die Nullen und Einsen immer abwechselnd vor$\}$
   RA für L_3: $(01)^* + (10)^* + 1(01)^* + 0(10)^* = (\varepsilon + 1)(01)^*(\varepsilon + 0)$



Lösung der Aufgabe 6 (Aufgabenblatt 2):

a) $(a+b)^* : \{a,b\}^*$.
   $(a^*b^*) : \varepsilon \in \mathcal{L}(a^*)$ und $\varepsilon \in \mathcal{L}(b^*)$. $\Rightarrow a \in \mathcal{L}(a^*b^*)$ und $b \in \mathcal{L}(a^*b^*)$

15 Anwendungsbeispiel für RA

Spezifikation einer Eingabemaske für Kontostände. Bedingungen:

- Währungseingabe: CHF, EUR, USD
- optionales Vorzeichen vor dem Betrag: $\oplus$, $\ominus$
- Betrag ganzzahlig oder mit Dezimalpunkt und zwei Nachkommastellen
- keine führende Nullen

$\Sigma = \{C, D, E, F, H, R, S, U, \oplus, \ominus, ., 0, 1, \ldots, 9\}$
Idee: Setze den RA aus Teilen zusammen:
zahlung = waehrung vorzeichen betrag
waehrung = $(CHF + EUR + USD)$
vorzeichen = $(\varepsilon + \oplus + \ominus)$
betrag = ganzzahl $(\varepsilon$ + .nachkomma)
ganzzahl = $(0 + (1 + 2 + \cdots + 9)(0 + 1 + \cdots + 9)^*$
nachkomma = $(0, 1, \ldots, 9) + (0, 1, \ldots, 9)$
$\Rightarrow$ zahlung = $(CHF + EUR + USD)$ $(\varepsilon + \oplus + \ominus)$ $(0 + (1 + 2 + \cdots + 9)(0 + 1 + \cdots + 9)^*$ $(\varepsilon$ + .(0,1,. . . ,9)+(0,1,. . . ,9))

16 Eine ganz neue Idee

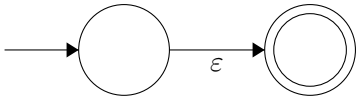
Theorem 1. Reguläre Ausdrücke und endliche Automaten sind äquivalent:

1. für jeden RA α existiert ein DEA A_α mit $\mathcal{L}(\alpha) = \mathcal{L}(A_\alpha)$
2. für jeden DEA A existiert ein RA α_A mit $\mathcal{L}(A) = \mathcal{L}(\alpha_A)$

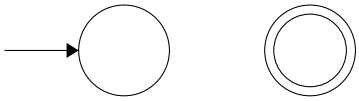
Beweis für 1. Nun. Umwandlung von RA in DEA. Vorgehensweise: RA $\mapsto$ ε-NEA$\mapsto$ DEA. Für einen gegebenen RA konstruiere einen ε-NEA induktiv entsprechend des regulären Ausdrucks des RA mit folgenden zusätzlichen Eigenschaften:

1. genau ein akzeptierender Zustand
2. keine Transitionen zum Startzustand
3. keine Transitionen vom akzeptierenden Zustand wegführende Transitionen

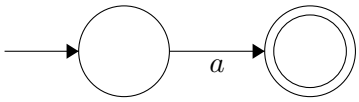
Induktionsanfang: ε-NEAfür ε:



ε-NEAfür $\varnothing$:



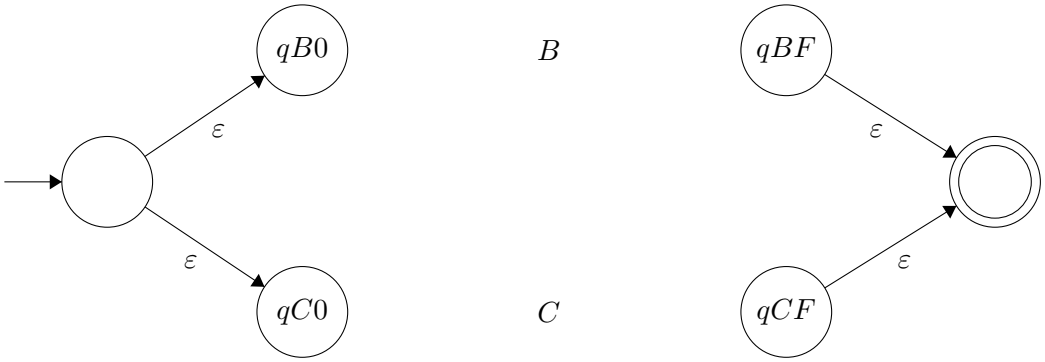
ε-NEAfür $a \in \Sigma$:



Induktionsschritt:

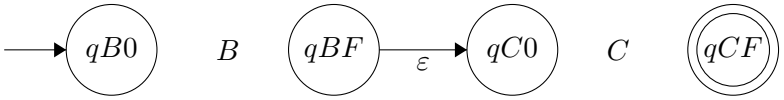
- Sei $\alpha = \beta + \gamma$ ein RA, seien B und C ε-NEAs für β und γ wie oben und mit Startzuständen q_{B0} und q_{C0} und akzeptierenden Zuständen q_{BF} und q_{CF}.

  Konstruiere hieraus einen ε-NEAfür α wie folgt:



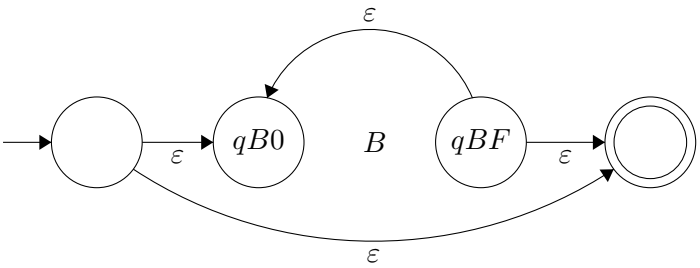
  erfüllt alle Bedingungen.

- Seien $\alpha = \beta \cdot \gamma$ und B, C ε-NEAs für β, γ. ε-NEAfür α:



  erfüllt alle Bedingungen.

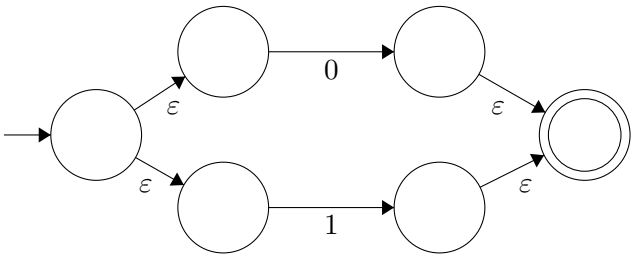
- Sei $\alpha = \beta^*$, B ein ε-NEAfür β. ε-NEAfür α:



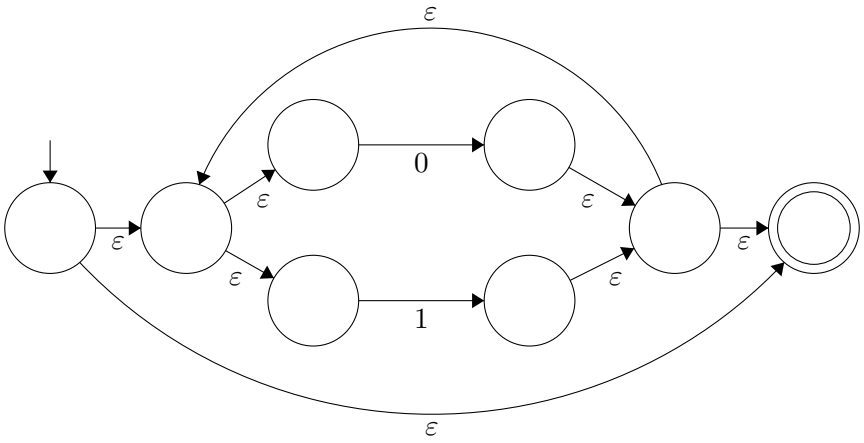
erfüllt alle Bedingungen.

□

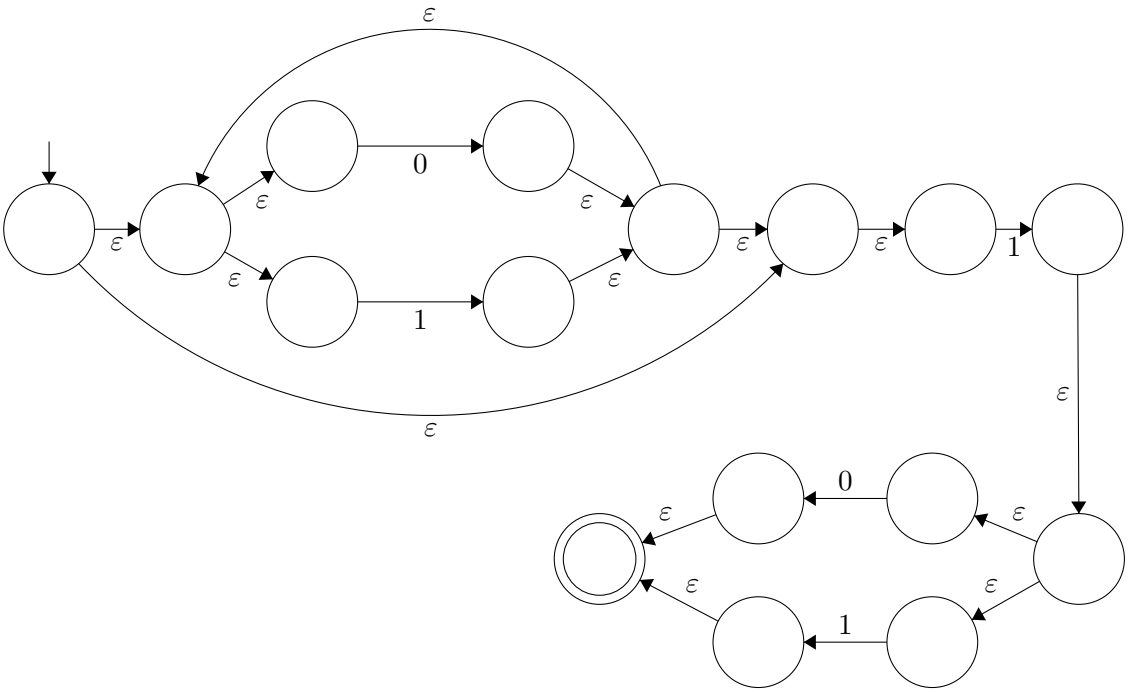
Beispiel. Beispiel: ε-NEAA für $\alpha = (0+1)^*1(0+1)$:
ε-NEAfür $(0+1)$:



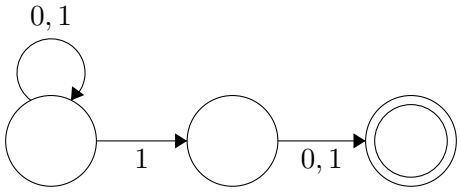
ε-NEAfür $(0+1)^*$:



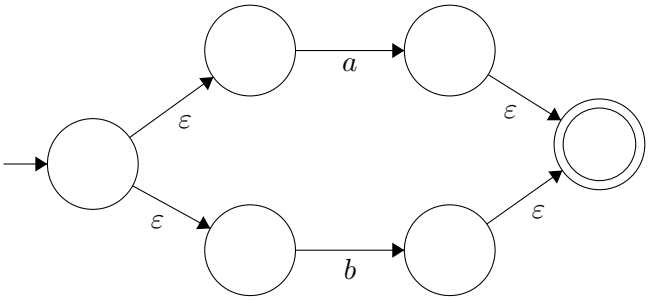
ε-NEAfür $(0+1)^*1(0+1)$:



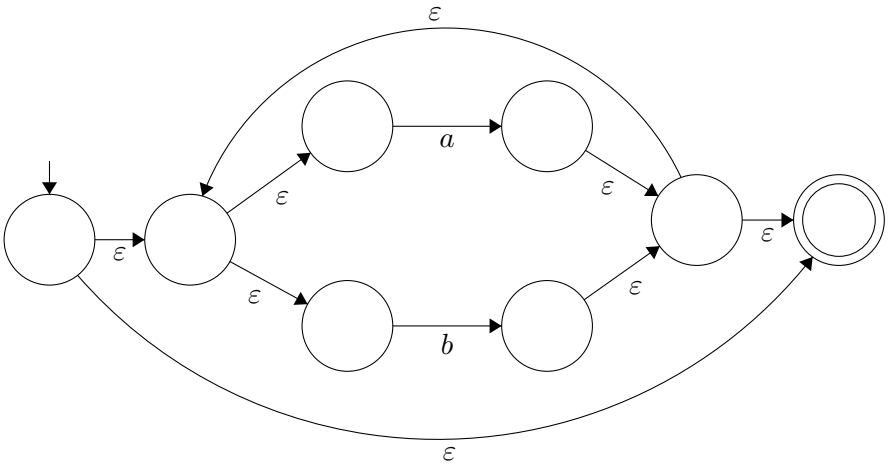
Vereinfachung:



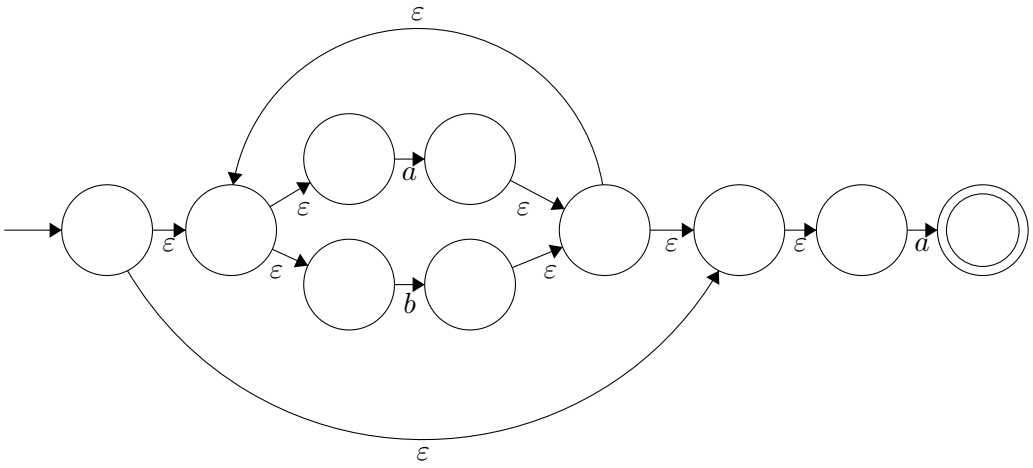
Übung (Aufgabe 1)**.** a) $a + b$:



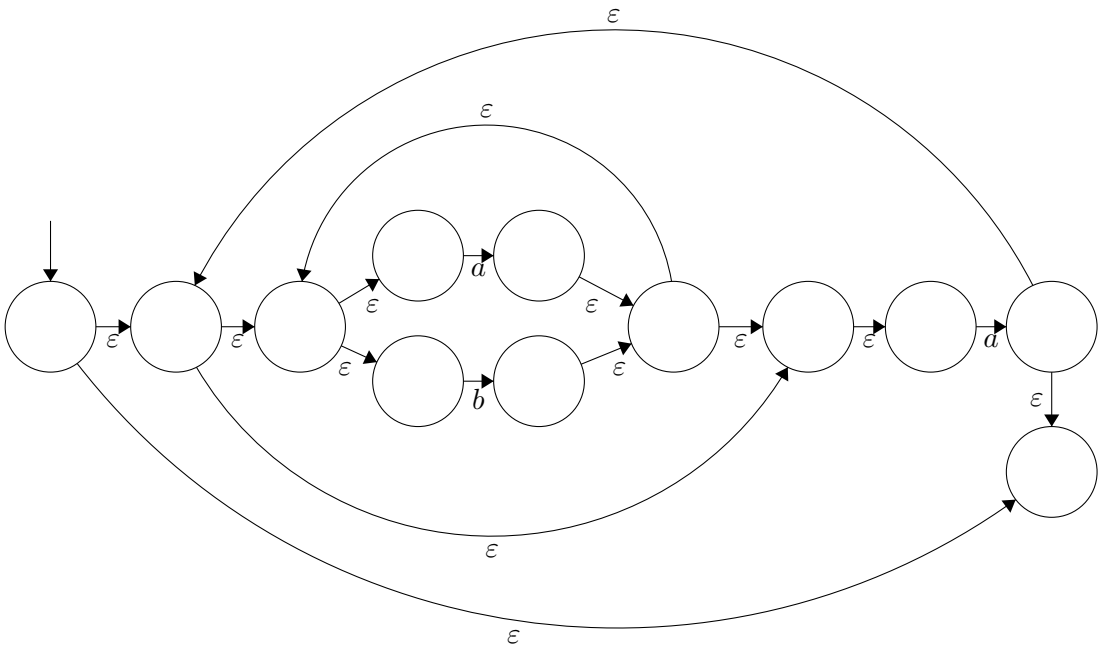
$(a + b)^*$:



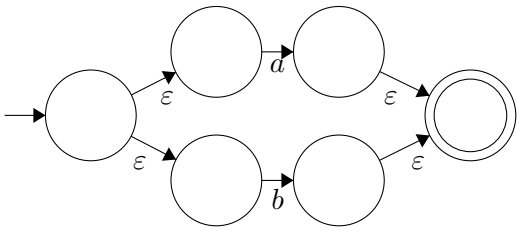
$(a + b)^* a$



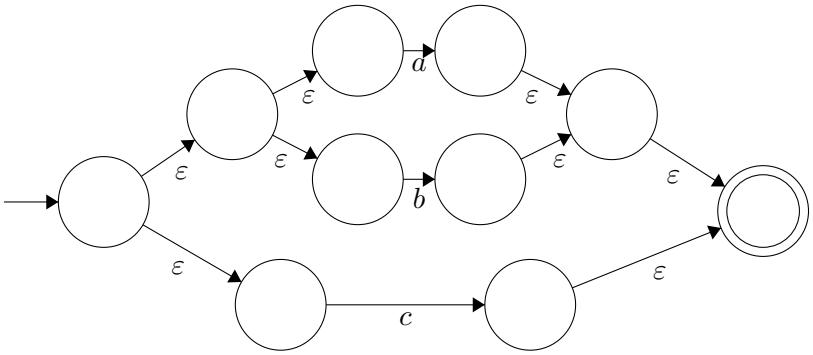
$((a + b)^* a)^*$



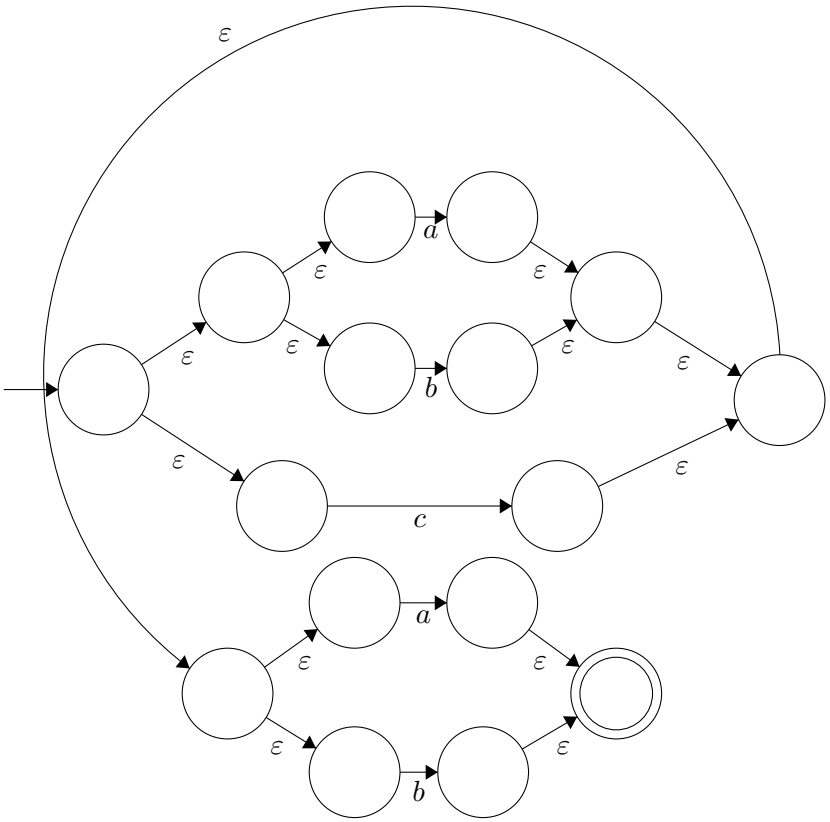
b) $a + b$



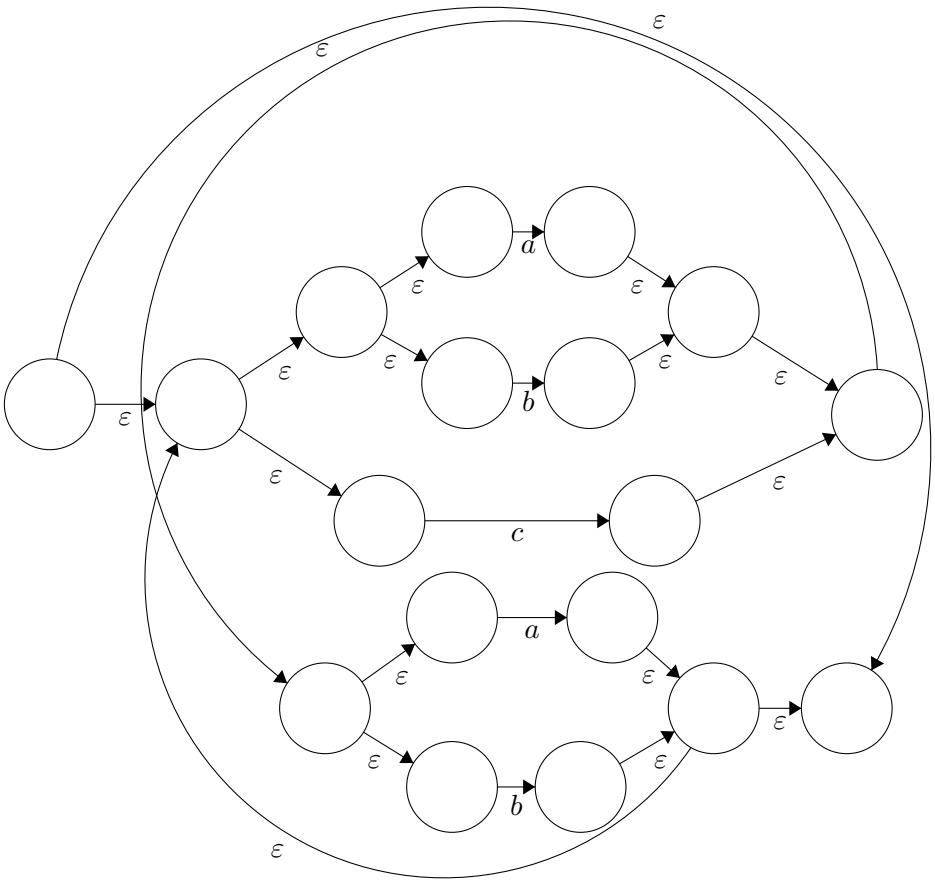
$a + b + c$



$(a + b + c) \cdot (a + b)$



Letzter Schritt:

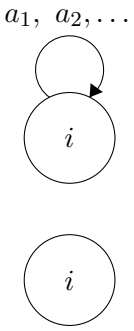


Beweis für 2. Idee: Dynamische Programmierung. Die Teilprobleme hier sind: Für jedes Paar (p, q) von Zuständen finde einen regulären Ausdruck, der alle Wörter beschreibt, die von p nach q führen.

Genauer: Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA, seien die Zustände durchnumeriert, d.h. $Q = \{1, 2, \ldots, n\}$, $q_0 = 1$.

1. Berechne für alle $i, j \in \{1, 2, \ldots n\}$ reguläre Ausdrücke $\alpha_{i,j}^{(0)}$, die die direkten Verbindungen von Zustand i zu Zustand j beschreiben:

   a) $i = j$:



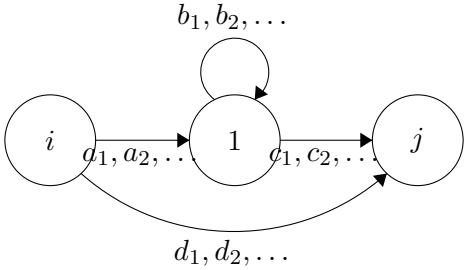
$\alpha_{i,i}^{(0)} = \varepsilon + a_1 + a_2 + \ldots$ oder $\alpha_{i,i}^{(0)} = \varepsilon$

b) $i \neq j$

$\alpha_{i,i}^{(0)} = a_1 + a_2 + \ldots$ oder $\alpha_{i,j}^{(0)} = \varnothing$

2. Nun werden nacheinander alle anderen Zustände als mögliche Zwischenstation auf dem Weg von i nach j hinzugenommen – zunächst den Zustand 1:

   $\alpha_{i,j}^{(1)}$ beschreibt die Wörter, mit denen man von i nach j kommt und zwischendurch nur den Zustand 1 besucht.



$\alpha_{i,j}^{(1)} = \alpha_{i,j}^{(0)} + \alpha_{i,1}^{(0)} (\alpha_{1,1}^{(0)})^* \alpha_{1,j}^{(0)}$

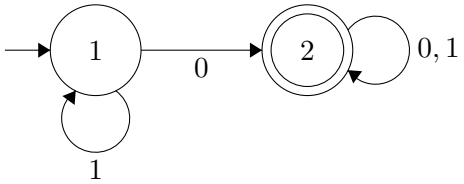
Induktion: Wir nehmen an, dass wir $\alpha_{i,j}^{(k-1)}$ für alle i, j schon berechnet haben. Dann gilt: $\alpha_{i,1}^{(k)} = \alpha_{i,j}^{(k-1)} + \alpha_{i,k}^{(k-1)} (\alpha_{k,k}^{(k-1)})^* \alpha_{k,j}^{(k-1)}$

Daraus folgt der Reguläre Ausdruck für den DEA A mit $q_0 = 1$ und $F = \{f_1, f_2, \ldots f_m\}$:

$$\alpha_{1,f_1}^{(n)} + \alpha_{1,f_2}^{(n)} + \cdots + \alpha_{1,f_m}^{(n)}$$

□

Beispiel. DEA A:



1. $\alpha_{1,1}^{(0)} = 1 + \varepsilon$

   $\alpha_{1,2}^{(0)} = 1$

   $\alpha_{2,1}^{(0)} = \varnothing$

   $\alpha_{2,2}^{(0)} = \varepsilon + 0 + 1$

2. $\alpha_{1,1}^{(1)} = \alpha_{1,1}^{(0)} + \alpha_{1,1}^{(0)}(\alpha_{1,1}^{(0)})^*\alpha_{1,1}^{(0)} = 1 + \varepsilon + (1+\varepsilon)(\alpha_{1,1}^{(0)})^*\alpha_{1,1}^{(0)} = (1+\varepsilon)^* = 1^*$

   $\alpha_{1,2}^{(1)} = \alpha_{1,2}^{(0)} + \alpha_{1,1}^{(0)}(\alpha_{1,1}^{(0)})^*\alpha_{1,2}^{(0)} = 0 + (1+\varepsilon)(1+\varepsilon)^*0 = 1^*0$

   $\alpha_{2,1}^{(1)} = \alpha_{2,1}^{(0)} + \alpha_{2,1}^{(1)}(\alpha_{1,1}^{(1)})^*\alpha_{1,1}^{(1)} = \varnothing + \varnothing(1+\varepsilon)^*(1+\varepsilon) = \varnothing$

   $\alpha_{2,2}^{(1)} = \alpha_{2,2}^{(0)} + \alpha_{2,1}^{(0)}(\alpha_{1,1}^{(0)})^*\alpha_{1,2}^{(0)} = (\varepsilon + 0 + 1) + \varnothing\cdots = \varepsilon + 0 + 1$

3. $\alpha_{1,1}^{(1)} + \alpha_{1,2}^{(1)}(\alpha_{2,2}^{(1)})^*\alpha_{2,1}^{(1)} = 1^* + 1^*0(\varepsilon + 0 + 1)\varnothing = 1^*$

   $\alpha_{1,2}^{(2)} = \alpha_{1,2}^{(1)} + \alpha_{1,2}^{(1)}(\alpha_{2,2}^{(1)})^*\alpha_{2,2}^{(1)} = 1^*0 + 1^*0(\varepsilon+0+1)^*(\varepsilon+0+1) = 1^*0(0+1)^* = \mathcal{L}(A)$

17 Reguläre Sprachen und ihre Eigenschaften

Definition 25 (Reguläre Sprache)**.** Eine Sprache heisst regulär, wenn sie von einem DEA akzeptiert wird.

$$\mathcal{L}_{\text{reg}} = \mathcal{L}(\text{DEA}) = \{L | L\text{ist regulär}\}$$

Ziel: Charakterisierung von regulären Sprachen.

17.1 Abschlusseigenschaften regulärer Sprachen

Vereinigung Wenn L und M reguläre Sprachen sind, dann ist auch L vereinigt mit M regulär.

Beweis. Wenn L und M regulär sind, dann existieren reguläre Ausdrücke α und β für L und M, d.h. $L(\alpha) = L$ und $L(\beta) = M$.

Daraus folgt, dass $\alpha + \beta$ ebenfalls ein RA ist, daraus folgt $L(\alpha + \beta) = L \cup M$ (gemäss Definition der Vereinigung von RA). □

Verkettung Wenn L und M reguläre Sprachen sind, dann ist auch L verkettet mit M regulär.

Beweis. Analog der Vereinigung. □

Kleene'scher Stern Wenn L eine reguläre Sprache ist, dann ist auch L^* eine reguläre Sprache.

Beweis. Analog der Vereinigung. □

Komplement Wenn L eine reguläre Sprache ist, dann ist auch $\bar{L} = \Sigma^* - L$ regulär.

Beweis. Sei A ein DEA für L. Dann gibt es für jedes Wort w aus Σ^* einen eindeutigen Zustand, in dem der DEA endet, wenn er w liest. Alle Wörter aus $L(A) = L$ führen in Endzustände, alle Wörter aus $\Sigma^* - L = \bar{L}$ in nichtakzeptierende Zustände. Vertauschen von End- und Nicht-Endzuständen ergibt einen DEA für $\overline{L}$. □

Durchschnitt Wenn L und M reguläre Sprachen sind, dann ist auch $L \cap M$ regulär.

Beweis 1 mit De Morgan'schen Gesetzen. Es gelten ja:

$$\overline{A \cap B} = \overline{A} \cup \overline{B} \quad (1)$$
$$\overline{A \cup B} = \overline{A} \cap \overline{B} \quad (2)$$

Da L, M regulär sind, sind auch $\overline{L}, \overline{M}$ regulär. $\overline{L} \cup \overline{M}$ ist ebenfalls regulär. $\overline{\overline{L} \cup \overline{M}}$ ist ebenfalls regulär. Durch De Morgan No. 1 ist $\overline{\overline{L} \cap \overline{M}} = L \cap M$ schnurstracks auch regulär.

□

Beweis 2 durch Konstruktion des Produktautomaten.

Definition 26 (Produktautomat)**.** Sei $A = (Q_A, \Sigma, \delta_A, q_{0A}, F_A)$ ein DEA für L, $B = (Q_B, \Sigma, \delta_B, q_{0B}, F_B)$. Dann ist der Produktautomat für A und B der DEA $C = (Q_C, \Sigma, \delta_C, q_{0C}, F_C)$ mit

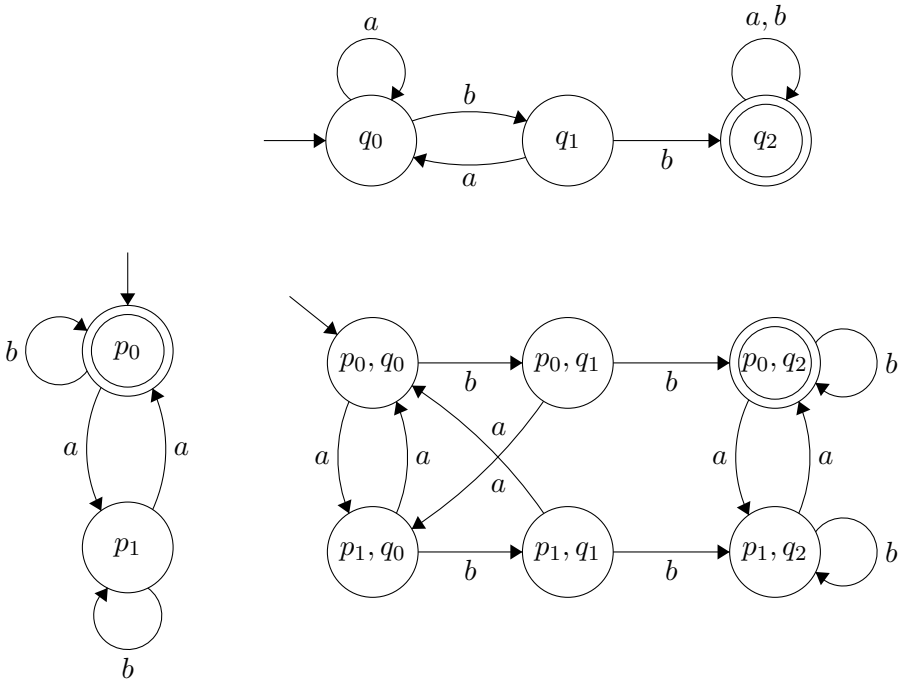
- $Q_C = Q_A \times Q_B$
- $\delta_C(p, q) = (\delta_A(p), \delta_B(q))$ für alle $(p, q) \in Q_C$
- $q_0 C = (q_{0A}, q_{0B})$
- $F_C = \{(p, q) \in Q_C | p \in F_A \text{ und } q \in F_B\}$

Anschaulich kann man sagen, dass C einfach die Arbeit von A und B simuliert und genau dann akzeptiert, wenn auch A und B akzeptieren.

Es gilt darum: $L(C) = L(A) \cap L(B) = L \cap M$

□

Beispiel. Sei $L = \{w \in \{a, b\}^* | |w|_a \text{ ist gerade }\}$. Sei $M = \{w \in \{a, b\}^* | w$ enthält das Teilwort $ab.\}$.



Bemerkung. Mit $F_C = \{(p, q) \in Q_C | p \in F_A \text{ oder } q \in F_B\}$ funktionert auch für die Vereinigung.

Mengendifferenz Wenn L und M regulär sind, ist auch $L - M$ regulär.

Beweis. Es gilt $L - M = L \cap \overline{M}$. Das Komplement ist regulär, der Schnitt ist regulär, ergo ist $L \cap \overline{M}$ regulär. □

Spiegelung

Definition 27. Sei $w = a_1 \dots a_k$ ein Wort. Dann ist $w^{\mathrm{R}} = a_k \dots a_1$ die Spiegelung von w.

Sei $L \subseteq \Sigma^*$ eine Sprache. Dann ist $L^{\mathrm{R}} = \{w \in \Sigma^* | w^{\mathrm{R}} \in L\}$ die Spiegelung von L.

Bemerkung. Einige coole Sachen:

- $\varepsilon^{\mathrm{R}} = \varepsilon$
- $(w^{\mathrm{R}})^{\mathrm{R}} = w$
- Wenn L regulär ist, dann ist auch L^{R} regulär.

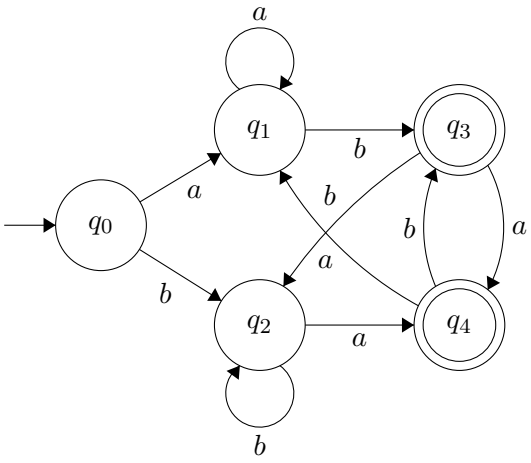
Beweis. Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA für L. Wir konstruieren einen ε-NEAB für L^{R} wie folgt:

$B = (Q \cup \{q_S\}, \Sigma, \delta_B, q_S, F_B)$ mit

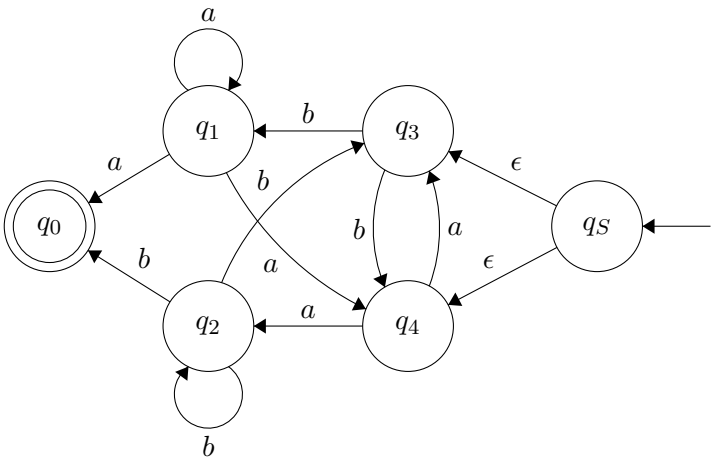
1. $F_B = \{q_0\}$
2. $\delta_B(q, a) = \{p\}$ für $\delta(p, a) = q$
3. $\delta_B(q_S, \varepsilon) = \{p\}$ für alle $p \in F$

B simuliert A in umgekehrter Richtung $\Rightarrow$ B akzeptiert L^{R}. □

Beispiel. $L = \{w \in \{a, b\}^* | w \text{ endet auf } ba \text{ oder } ab\}$.



$L^{\mathrm{R}} = \{w \in \{a, b\}^* | w \text{ beginnt mit } ba \text{ oder } ab\}$.



Beweis über Reguläre Ausdrücke. Sei α ein regulärer Ausdruck mit $L(\alpha) = L$.
Zu zeigen: Es gibt einen RA β mit $L(\beta) = (L(\alpha))^{\mathrm{R}} = L^{\mathrm{R}}$.
Strukturelle Induktion über den Aufbau von α. Induktionsanfang:

- $\varepsilon^{\mathrm{R}} = \varepsilon$

- $\varnothing^{\mathrm{R}} = \varnothing$
- $a^{\mathrm{R}} = a$ für ein $a \in \Sigma$

Induktionsschritt:

- Vereinigung $\alpha = \alpha_1 + \alpha_2$:

$$L(\alpha_1 + \alpha_2)^{\mathrm{R}} = (L(\alpha_1) \cup L(\alpha_2))^{\mathrm{R}} = L(\alpha_1)^{\mathrm{R}} \cup L(\alpha_2)^{\mathrm{R}} \Rightarrow \alpha^{\mathrm{R}} = \alpha_1^{\mathrm{R}} + \alpha_2^{\mathrm{R}}$$

- Konkatenation $\alpha = \alpha_1\alpha_2$:

$$L(\alpha_1\alpha_2)^{\mathrm{R}} = (L(\alpha_1)L(\alpha_2))^{\mathrm{R}} = L(\alpha_1)^{\mathrm{R}}L(\alpha_2)^{\mathrm{R}} \Rightarrow \alpha^{\mathrm{R}} \overset{(w_1w_2)^{\mathrm{R}}=w_2^{\mathrm{R}}w_1^{\mathrm{R}}}{=} \alpha_1^{\mathrm{R}}\alpha_2^{\mathrm{R}}$$

- Kleene'scher Stern $\alpha = \alpha_1^*$:

$$L(\alpha_1^*)^{\mathrm{R}} = (L(\alpha_1)^{\mathrm{R}})^* \to \alpha^{\mathrm{R}} = (\alpha_1^{\mathrm{R}})^*$$

□

Homomorphismus

Definition 28. Ein (Wort-)Homomorphismus ist eine Abbildung

$$h : \Sigma_1 \mapsto \Sigma_2^*$$

die die Symbole eines Alphabets auf Wörter abbildet.

Beispiel. $\Sigma_1 = \{0, 1\}, \Sigma_2 = \{a, b\}, h(0) = ab, h(1) = \varepsilon$

Erweiterung auf Wörter: Sei $w = a_1 \dots a_k$. Dann ist $h(w) = h(a_1) \cdot h(a_2) \cdot \dots \cdot h(a_k)$.

Beispiel. $h(0101) = ab \cdot \varepsilon \cdot ab \cdot \varepsilon = abab$

Definition 29. Sei $h : \Sigma_1 \mapsto \Sigma_2^*$ ein Homomorphismus, sei $L \subseteq \Sigma_1^*$ eine Sprache. Dann ist

$$h(L) = \{w \in \Sigma^* | w = h(x) \text{ für ein } x \in L\}$$

Satz: Sei $h : \Sigma_1 \mapsto \Sigma_2^*$ ein Homomorphismus. Wenn $L \subseteq L_1^*$ regulär ist, dann ist auch $h(L)$ regulär.

Beweis über rock-the-bottom strukturelle Induktion. Für einen regulären Ausdruck α über Σ_1 ist $h(\alpha)$ der reguläre Ausdruck über Σ_2, der entsteht, wenn jedes Vorkommen von $a \in \Sigma_1$ durch $h(a)$ ersetzt wird. □

Beispiel. $\Sigma_1 = \{0, 1\}, \Sigma_2 = \{a, b\}, h(0) = ab, h(1) = \varepsilon.$

$$\alpha = ((0 + 1)0)^*1 \Rightarrow h(\alpha)((ab + \varepsilon) \cdot ab)^*\varepsilon = (ab)^*$$

Und latürnich gilt: $L(h(\alpha)) = h(L)$

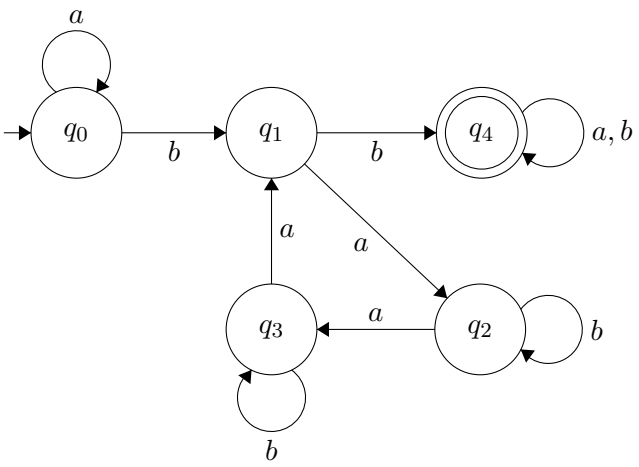
Beispiel (Übungsaufgaben 9. Oktober – Aufgabe 3)**.** Deifniere die Menge der a-Vorgänger eines Zustands q als $V_a(q) = \{p \in Q | \delta(p, a) = q\}$. Dann ist $B = (Q, \Sigma, \delta, q_0, V_a(F))$ ein DEA für L/a.

17.2 Nichtregularität

Ziel: Zeige, dass es Sprachen gibt, die nicht regulär sind.

Idee: DEAs analysieren: DEAs können nicht beliebig weit zählen.

Beispiel. Betrachte den folgenden DEA:



Arbeit von A auf dem Wort $w = abaaab$:

$$q_0 \overset{a}{\mapsto} q_0 \overset{b}{\mapsto} q_1 \overset{a}{\mapsto} q_2 \overset{a}{\mapsto} q_3 \overset{a}{\mapsto} q_1 \overset{b}{\mapsto} q_4$$

Enthält einen Zykel $q_1 \overset{a}{\mapsto} q_2 \overset{a}{\mapsto} q_3 \overset{a}{\mapsto} q_1$.

Wiederholung des Zykels: $w' = abaaaaaab$

$$q_0 \overset{a}{\mapsto} q_0 \overset{b}{\mapsto} q_1 \overset{a}{\mapsto} q_2 \overset{a}{\mapsto} q_3 \overset{a}{\mapsto} q_1 \overset{a}{\mapsto} q_2 \overset{a}{\mapsto} q_3 \overset{a}{\mapsto} q_1 \overset{b}{\mapsto} q_4$$

Da $q_4 \in F$, gilt $w' \in L(A)$

Allgemein: Sei A ein DEA. Sei $w = u \cdot x \cdot v$ ein Wort in $L(A)$, $u, v, x \in \Sigma^*, x \neq \varepsilon$. Falls der Automat auf dem Teilwort x in einer Schleife läuft, dann ist auch das Wort $w' = uxxv \in L(A)$.

Formal: Falls $\hat{\delta}(q_0, u) = q$ und $\hat{\delta}(q, x) = q$ und $\hat{\delta}(q, v) \in F$, dann ist $w' = uxxv \in L(A)$.

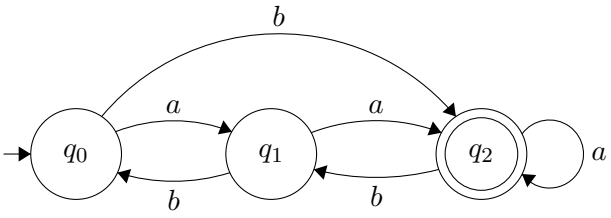
Im Beispiel oben:

$$w = \underbrace{u}_{ab}\underbrace{x}_{aaa}\underbrace{v}_{b} \in L(A)$$

, $w' = \underbrace{u}_{ab}\underbrace{x}_{aaa}\underbrace{x}_{aaa}\underbrace{v}_{b} \in L(A)$

Bemerkung. Ein DEA ist vollständig, d.h. für jedes $q \in Q$ und jedes $a \in \Sigma$ existiert genau ein Folgezustand $\delta(q, a)$. Daraus folgt direkt, dass man mit jedem Wort aus Σ^* kann man durch den Automaten laufen, ohne stecken zu bleiben. Daraus wiederum folgt, dass bei der Arbeit des DEA A mit n Zuständen auf einem Wort der Länge $\geq n$ wird mindestens ein Zustand doppelt besucht.

Beispiel. Sei



$|Q| = 3$.
z.B. $w_1 = abb$: $q_0 \overset{a}{\mapsto} q_1 \overset{b}{\mapsto} q_0 \overset{b}{\mapsto} q_2$
$\Rightarrow q_0$ doppelt besucht.

Beobachtung: Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA mit $|Q| = n$. Sei $w \in L(A)$ mit $|w| \geq n$. Dann wiederholt sich in der Berechnung von A auf w mindestens einmal ein Zustand. Wir nennen einen dieser wiederholt besuchten Zustände q_x. Dann lässt sich w zerlegen in drei Teile

$$w = \underbrace{u}_{\text{bis zum ersten } q_x} \cdot \underbrace{x}_{\text{von } q_x \text{ nach } q_x} \cdot \underbrace{v}_{\text{nach } q_x}$$

Berechnug von A auf w hat die Form

$$q_0 \overset{u}{\rightsquigarrow} q_x \overset{x}{\rightsquigarrow} q_x \overset{v}{\rightsquigarrow} q_F \in F$$

Damit gibt es auch eine Berechnung von A auf $w' = uxxv$:

$$q_0 \overset{u}{\rightsquigarrow} q_x \overset{x}{\rightsquigarrow} q_x q_x \overset{x}{\rightsquigarrow} q_x \overset{v}{\rightsquigarrow} q_F \in F$$

Beispiel. Diese Beobachtung können wir nutzen, um zu zeigen, dass $L = \{a^k b^k | k \geq 0\}$ nicht regulär ist.

Beweis durch Widerspruch. Für L kann es keinen DEA mit 7 Zuständen geben.

Annahme: Es gibt einen DEA $A = (Q, \Sigma, \delta, q_0, F)$ mit $|Q| = 7$ und $\Sigma = \{a, b\}$, so dass $L(A) = L$.

Es gibt zwei Möglichkeiten zur Widerlegung:

1. alle möglichen DEA mit 7 Zuständen ausprobieren. Klappt nicht, weil zu viele.

2. Beobachtung über Zykel ausnutzen.

   Betrachte das Wort $w = aaaabbbb \in L$. Es gilt $|w| = 8 > 7 = |Q|$. Also muss sich in der Berechnung von A auf w mindestens ein Zustand wiederholen.

   Zerteilen wir w in $w = u \cdot x \cdot v$ mit der Berechnung

   $$q_0 \overset{u}{\rightsquigarrow} q_x \overset{x}{\rightsquigarrow} q_x \overset{v}{\rightsquigarrow} q_F \in F$$

   Daraus folgt $w' = uxxv \in L(A)$. Wie kann das Wort x aussehen? Fallunterscheidung:

(a) x besteht nur aus as. z.B. $w = \underbrace{aa}_{u}\underbrace{aa}_{x}\underbrace{bbbb}_{v}$. Daraus folgt $w' = \underbrace{aa}_{u}\underbrace{aa}_{x}\underbrace{aa}_{x}\underbrace{bbbb}_{v} = a^6b^4 \notin L$. Widerspruch!

(b) x besteht nur aus bs. z.B. $w = \underbrace{aaaab}_{u}\underbrace{b}_{x}\underbrace{bb}_{v}$. Daraus folgt $w' = \underbrace{aaaab}_{u}\underbrace{b}_{x}\underbrace{b}_{x}\underbrace{bb}_{v} = a^4b^5 \notin L$. Widerspruch!

(c) x besteht aus as und bs. z.B. $w = \underbrace{aa}_{u}\underbrace{aab}_{x}\underbrace{bbb}_{v}$. Daraus folgt $w' = \underbrace{aa}_{u}\underbrace{aab}_{x}\underbrace{aab}_{x}\underbrace{bbb}_{v} \notin L$. Widerspruch!

Daraus folgt, dass es keinen DEA mit 7 Zuständen für L gibt.

Denselben Beweis hätten wir auch für jede andere Anzahl von Zuständen führen können. Daraus kann gefolgert werden, dass es überhaupt keinen DEA für L gibt. Daraus folgt, dass L nicht regulär ist.

□

Beispiel. Argumentieren wir, warum ein DEA mit 5 Zuständen die Sprache

$$L = \{a^k b^{2k} | k \geq 0\}$$

nicht akzeptieren kann.

Beweis. Annahme: Es gibt einen DEA mit $|Q| = 5$. Betrachten wir uns das Wort $w = aabbbb \in L$. Es gilt $|w| = 6$, das heisst es wird mindestens 1 Zustand mehrfach durchlaufen. w lässt sich also zerlegen in $w = uxv$:

$$q_0 \overset{u}{\rightsquigarrow} q_x \overset{x}{\rightsquigarrow} q_F \in F.$$

Dann gilt $w' = uxxv \in L(A)$.

Fallunterscheidung über die Form von x:

1. x besteht nur aus as. D.h. w' enthält mehr as als w, aber nicht mehr bs $\Rightarrow w' \notin L$. Widerspruch!
2. x besteht nur aus bs. Analoger Widerspruch.
3. x besteht aus as und bs. $\Rightarrow$ in w' kommt ein b vor einem $a \Rightarrow w' \notin L$. Darum gibt es keinen DEA mit 5 Zuständen, der L akzeptieren könnte.

□

17.3 Verallgemeinerung der Methode

Ziel: Zeige, dass es gar keinen DEA für L gibt.

Definition 30 (Pumping-Lemma für reguläre Sprachen)**.** Sei L eine reguläre Sprache. Dann gibt es eine Konstante n_0 (die von L abhängt), so dass für jedes Wort $w \in L$ mit $|w| \geq n_0$ gilt:

w kann zerlegt werden in $w = uxv$ mit den folgenden Eigenschaften:

(a) $|w| \geq 1$, d.h. $x \neq \varepsilon$

(b) $|ux| \leq n_0$

(c) Für alle $k \geq 0$ ist auch $ux^k v \in L$.

Begründung der Korrektheit. Falls L regulär ist, existiert ein DEA A für L. Setzen wir $n_0 = |Q|$. x ist dann die Beschriftung eines Zykels in A, der in den ersten $|Q|$ Berechnungsschritten auftritt ($|ux| \leq n_0$). Eine Wiederholung des Zykels ändert nichts an der Akzeptanz des Wortes. □

Beispiel. Zeigen wir, dass $L = \{a^k b^k | k \in \mathbb{N}\}$.

Beweis. Annahme: L ist regulär. Das Pumping-Lemma gibt uns die Existenz einer Konstanten n_0, sodass für alle Wörter $w \in L$ mit $|w| \geq n_0$ alle Bedingungen gelten. Um einen Widerspruch herzuleiten, können wir uns ein Wort $w \in L$ auswählen, sei also $w = a^{n_0} b^{n_0}$. Nach dem Pumping-Lemma gibt es eine Zerlegung $w = uxv$ mit $|ux| \leq n_0$ und $|x| \geq 1$.

Daraus folgt, dass:

$$u = a^j \text{ für ein } 0 \leq j \leq n_0 \leq n_0$$

und

$$x = a^i \text{ für ein } 1 \leq i \leq n_0.$$

Weiter gilt nach dem Pumping-Lemma: $ux^k v \in L$ für alle k. Um den Widerspruch herzuleiten, wählen wir $k = 2$. Nach dem Pumping-Lemma muss gelten, dass

$$ux^2 v = a^{n_0+i} b^{n_0} \in L.$$

Das ist ein Widerspruch weil $n_0 + i \neq n_0$. Also kann L nicht regulär sein. □

Bemerkung. So nützlich es auch ist, aber:

- Das Pumping-Lemma gibt nur notwendige Eigenschaften für reguläre Sprachen an, aber keine hinreichenden. Die Regularität einer Sprache kann also damit nicht gezeigt werden.
- Es gibt Sprachen, die nicht regulär sind und trotzdem das Pumping-Lemma erfüllen. Dumm gelaufen.

17.4 Anwendung des Pumping-Lemmas gegen einen Gegner

Wir spielen gegen Darth Vader, die Lemmings und die Achse der bösen Allquantoren.

1. Wir wählen die Sprache, deren Nichtregularität wir nachweisen wollen.
2. Darth Vader wählt n_0.
3. Wir wählen das Wort w mit $|w| \geq n_0$.

4. Darth Vader lässt sich davon nicht beeindrucken und zerlegt $w = uxv$ gemäss dem Pumping-Lemma.

5. Wir wählen k und gewinnen, falls wir einen Widerspruch herleiten können.

Beispiel. Viele viele Aufgaben:

(a) Nicht regulär: $L = \{0^m 10^m\}, m \geq 1$

Annahme: L ist regulär. Darum gibt es n_0, sodass alle Pumping-Lemma-Bedingungen gelten. $w = 0^{n_0}10^{n_0}$.

Zerlegung: $w = uxv$. $u = 0^j$, $x = 1$. $ux^kv \in L$ für alle k. $k = 2$. Daraus folgt: $ux^2v = 0^{n_0}1^20^{n_0}$. Foul.

(b) Nicht regulär: $L = \{yy^{\mathrm{R}} | y \in \{a, b^*\}\}$

Annahme: L ist regulär. Darum gibt es n_0, sodass alle Pumping-Lemma-Bedingungen gelten. $w = a^{n_0}bba^{n_0}$. Zerlegung. $\Rightarrow |ux| \leq n_0, |x| \geq 1$, also

$$u = a^j, x = a^i$$

Gem. Pumping-Lemma ist $ux^0v = uv - a^{n_0-i}bba^{n_0}$. Widerspruch.

Beispiel. 6a. Differenz: Offenbar 0^*1^* regulär. Annahme: L_1 ist regulär.

$$\{0^m1^m\} = 0^*1^* - \{0^i1^j | i \neq j\}.$$

6b. Homomorphismus: $h : \{a, b, c\} \mapsto \{0, 1\}$. $h(a) = 0, h(b) = 0, h(c) = 1$.
$h(L_2) = \{0^m0^l1^{m+l} | l, m \geq 0\}$

18 Reguläre Grammatiken

Idee: Mechanismus zur Erzeugung aller Wörter aus einer Sprache.

Beispiel. Reguläre Sprache:

$$L = \{1w | w \in 0^*\} \subseteq \{0, 1\}^* = \{1, 10, 100, 1000, \dots\}$$

Vorgehen bei der Konstruktion solcher Wörter:

- 1 an den Anfang setzen
- beliebig viele 0 auffüllen.

Regeln:

(1) start $\rightarrow 1N$

(2) $N \rightarrow 0N$

(3) $N \to \varepsilon$

Konstruktion von 1000:
start $\overset{(1)}{\Rightarrow} 1N \overset{(2)}{\Rightarrow} 10N \overset{(2)}{\Rightarrow} 100N \overset{(2)}{\Rightarrow} 1000N \overset{(3)}{\Rightarrow} 1000$

Definition 31. Eine Rechtslineare Grammatik G erzeugt alle Wörter einer Sprache buchstabenweise von links nach rechts. Sie wird beschrieben durch

$$G = (N, T, P, S),$$

wobei

- T das Terminalalphabet (Alphabet der Sprache)
- N das Nichtterminalalphabet (Menge von Hilfssymbolen)
- $P \subseteq N \times (TN \cup T \cup \{\varepsilon\})$ die Menge von Produktionen (Regeln). Für $p = (l, r)$ heisst l die linke Seite von P und r die rechte Seite von P. Wir schreiben statt $p = (l, r)$ auch $l \to r$
- $S \in N$ ist das Startsymbol

Anschauung: Produktion $p = l \to r$ ist eine Ersetzungsregel. In einem Wort über $T \cup N$ wird durch Anwendung von p ein Vorkommen von l durch r ersetzt.

Definition 32 (Ableitungsschritt)**.** Sei $G = (N, T, P, S)$ eine rechtslineare Grammatik. Ein Ableitungsschritt in G ist eine Relation

$$\Rightarrow \subseteq T^*N \times T^*(N \cup \varepsilon)$$

definiert durch $uA \Rightarrow uw$ genau dann, wenn $A \to w \in P$, wobei $u \in T^*, A \in N, w \in TN \cup T \cup \{\varepsilon\}$.

Definition 33 (Ableitung)**.** Eine Ableitung ist eine endliche Folge von Ableitungsschritten. Formal ist $\overset{*}{\Rightarrow}$ die reflexive, transitive Hülle von $\Rightarrow$.

Die von G erzeugte Sprache ist definiert als

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

also die Menge aller Wörter, die aus S in endlich vielen Ableitungsschritten erzeugt werden können.

Definition 34 (Äquivalenz von Grammatiken)**.** Zwei Grammatiken sind G_1, G_2 heissen äquivalent, falls

$$L(G_1) = L(G_2).$$

Beispiel. Ein Beispiel:

$$G = (\{S, A, B\}, \{0, 1\}, P, S),$$

wobei

$$P = \{S \overset{(1)}{\to} 0S, S \overset{(2)}{\to} 1S, S \overset{(3)}{\to} 0A, A \overset{(4)}{\to} 0B, A \overset{(5)}{\to} 1B, B \overset{(6)}{\to} 0, B \overset{(7)}{\to} 1\}$$

Ableitung von 011010:

$$S \overset{(1)}{\Rightarrow} 0S \overset{(2)}{\Rightarrow} 01S \overset{(2)}{\Rightarrow} 011S \overset{(3)}{\Rightarrow} 0110A \overset{(5)}{\Rightarrow} 01101B \overset{(6)}{\Rightarrow} 011010$$

Ableitung von 0111:
Zwei Regeln für S:

$$S \overset{(1)}{\Rightarrow} 0S \overset{(2)}{\Rightarrow} 01S \overset{(2)}{\Rightarrow} 011S \overset{(2)}{\Rightarrow} 0111S \ldots$$

Klappt nicht.

$$S \overset{(3)}{\Rightarrow} 0A \overset{(5)}{\Rightarrow} 01B \overset{(7)}{\Rightarrow} 011$$

Klappt ebenfalls nicht.

Definition 35.

$$\mathcal{L}_{\text{rlin}} = \{L | \text{ es existiert eine rechtslineare Grammatik } G \text{ mit } L = L(G)\}$$

Vereinfachung rechtslinearer Grammatiken. 3 Typen von Regeln:

(1) $A \to aB$

(2) $A \to a$

(3) $A \to \varepsilon$

Zeige: Die Regel (2) kann vermieden werden: Ersetze $A \to a$ durch $A \to aC$ und $C \to \varepsilon$.

Weitere Vereinfachung: Es reicht, eine Regel vom Typ (3), zusätzlich wird gegebenenfalls $S \to \varepsilon$, falls $\varepsilon \in L(G)$. Wie funktioniert das? Ersetze die Regeln $A_1 \to a_1B_1, \ldots, A_k \to a_kB_k$ und $B_1 \to \varepsilon, \ldots, B_k \to \varepsilon$ durch $A_1 \to a_1T, \ldots, A_k \to a_kT$ und $T \to \varepsilon$.

Theorem 2 (Äquivalenz von rechtslinearen Grammatiken und Endlichen Automaten)**.** Zu jeder rechtslinearen Grammatik über $T = \Sigma$ existiert ein NEA A über Σ mit $L(A) = L(G)$.

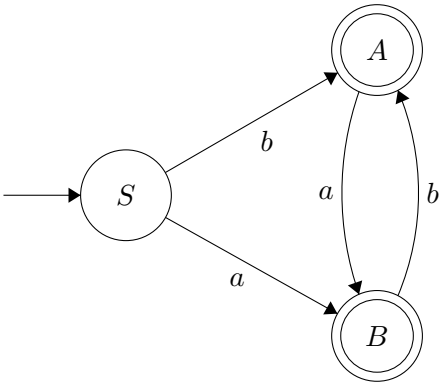
Beweis. $G = (N, \Sigma, P, S)$ ohne Typ (2).
Daraus folgt: NEA $A = (Q, \Sigma, \delta, q_0, F)$ mit

- $Q = N$

- $q_0 = S$
- $F = \{X \in N | X \to \varepsilon \in P\}$
- δ definiert durch: $X \in \delta(Y, a) \Leftrightarrow Y \to aY \in P$

□

Beispiel. $G = (\{S, A, B\}, \{a, b\}, P, S)$ mit
$P = \{S \to aB, S \to bA, A \to aB, B \to bA, A \to \varepsilon, B \to \varepsilon\}$.
NEA für $L(G)$:



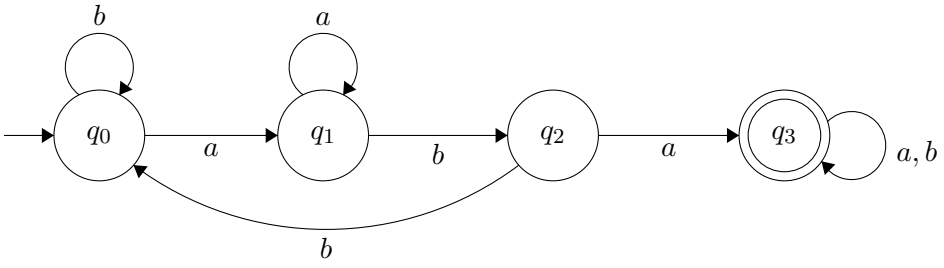
Theorem 3. Zu jedem DEA A über Σ existiert eine rechtslineare Grammatik G über $\Sigma = T$ mit $L(G) = L(A)$.

Proof. $A = (Q, \Sigma, \delta, q_0, F)$
$\Rightarrow G = (N, \Sigma, P, S)$ mit

- $N = G$
- $S = q_0$
- P definiert durch $p \to aq \in P \Leftrightarrow \delta(p, a) = q$ und $q \to \varepsilon \in P \Leftrightarrow q \in F$.

□

Beispiel.



19 Kontextfreie Grammatiken

Definition 36 (Kontextfreie Grammatik)**.** Eine kontextfreie Grammatik G wird beschrieben durch

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

wobei:

- Σ das Terminalalphabet;
- N das Nichtterminalalphabet;
- $P \subseteq N \times (T \cup N)^*$ die Menge der Produktionen (Regeln). (Statt (X, abY) nutzen wir $X \to abY$;
- S das Startsymbol

ist.

Beispiel. Kontextfreie Grammatik für die folgende Sprache: $L = \{a^n b^n | n \geq 0\}$:

$$G = (\{a, b\}, \{S\}, P, S),$$

wobei

$$P = \{S \to aSb, S \to \varepsilon\}$$

.

Beispielableitung für $aabb \in L$:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aa\varepsilon bb = aabb$$

Definition 37 (Kontextfreie Sprache)**.** Eine Sprache L ist kontextfrei, wenn es eine Kontextfreie Grammatik G gibt mit

$$L = L(G).$$

Die Familie

$$\mathcal{L}_{\text{CF}}$$

ist die Menger aller kontextfreien Sprachen. Es gilt:

$$\mathcal{L}_{\text{REG}} \subseteq \mathcal{L}_{\text{CF}}$$

Beweis. Jede reguläre Grammatik ist auch eine kontextfreie Grammatik. □

Beispiel. Kontextfreie Grammatik zur Erzeugung aller arithmetischer Ausdrücke.
Induktion:

- a ist ein arithmetischer Ausdruck.
- Für arithmetische Ausdrücke α, β sind

- $(\alpha + \beta)$,
- $(\alpha - \beta)$,
- $\alpha \cdot \beta$,
- α / β

ebenso arithmetische Ausdrücke.

Zum Beispiel $(a + a), (a \cdot (a + a \cdot a) + a)$.

Beschreibung durch eine kontextfreie Grammatik $G = (\Sigma, N, P, S)$ mit

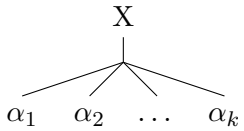
- $\Sigma = \{(,), a, +, -, \cdot, /\}$
- $N = \{S, A, B, C\}$ (A für einen ganzen Ausdruck, B für Addition/Division, C für Multiplikation/Division),
- $P = \{S \to A, A \to a, A = (ABA), A \to ACA, B \to +, B \to -, C \to \cdot, C \to /\}$

Beispielableitung von $(a \cdot (a - a) + a)$:

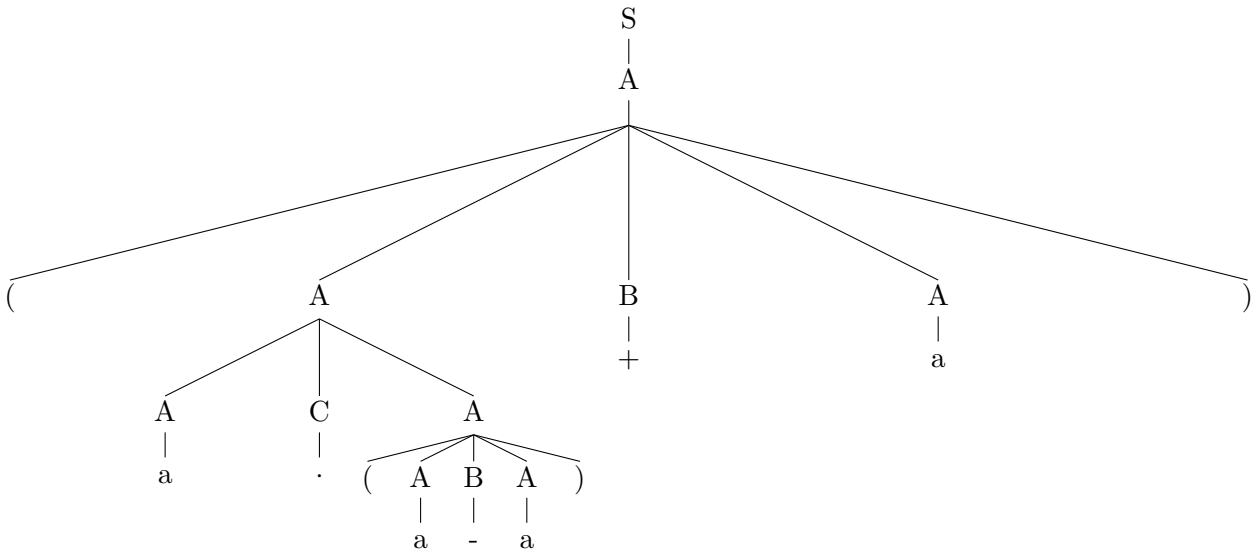
$$\begin{aligned} S &\Rightarrow A \Rightarrow (ABA) \Rightarrow (A + A) \Rightarrow (ACA + A) \Rightarrow (A \cdot A + A) \\ &\Rightarrow (A \cdot (ABA) + A) \Rightarrow (A \cdot (A - A) + A) \Rightarrow (a \cdot (A - A) + A) \\ &\Rightarrow (a \cdot (a - A) + A) \Rightarrow (a \cdot (a - a) + A) \Rightarrow (a \cdot (a - a) + a) \end{aligned}$$

Bessere Notation mit einem Parsebaum:

- Startsymbol als Wurzel
- Terminale als Blätter
- Nichtterminale als innere Knoten
- Regelanwendung: $X \to \alpha_1 \alpha_2 \ldots \alpha_k \mapsto$:



Ableitungsbaum für $(a \cdot (a - a) + a)$ in G:



20 Vereinfachung von kontextfreien Grammatiken

(1) Elimination der ε-Regeln (Regeln der Form $X \to \varepsilon$.

Definition 38. Eine kontextfreie Grammatik G heisst ε-frei, wenn sie keine Regeln der Form $X \to \varepsilon$ hat.

Theorem 4. Sei G eine kontextfreie Grammatik. Dann gibt es eine ε-freie kontextfreie Grammatik G' mit

$$L(G') = L(G) - \{\varepsilon\}.$$

Beweis. Transformation von G in G':

(a) Sei N_1 die Menge aller linken Seiten von ε-Regeln:

$$N_1 = \{A \in N | A \to \varepsilon\}$$

(b) Bestimme alle in ε ableitbaren Nichtterminale. Dann gilt

$$N_1 = \{A \in N | A \stackrel{*}{\Rightarrow} \varepsilon\}$$

Berechnung: $N_1 \leftarrow N_1 \cup \{A \in N | A \to \alpha, \alpha \in N_1^*\}$

(c) Für jede Regel $A \to \alpha_1 B \alpha_2$ mit $B \in N_1$, $\alpha_1, \alpha_2 \in (\Sigma \cup N)^*$ füge die Regel $A \to \alpha_1\alpha_2$ zu P hinzu und erhalte P_1. Iteriere so lange bis sich P_1 nicht mehr ändert.

(d) Eliminiere alle ε-Regeln aus P_1:

$$P_2 \leftarrow P_1 - \{r \in P_1 | r = A \to \varepsilon\}.$$

Daraus ergibt sich die Sprache $G' = (\Sigma, N, P_2, S)$.

□

Beispiel. Betrachte $G = (\{a, b, c\}, \{S, A, B, C\}, P, S)$ mit $P = \{S \to AB, S \to C, A \to aB, a \to \varepsilon, B \to bA, B \to \varepsilon, C = c\}$.

(a) $N_1 = \{A, B\}$

(b) Wegen $S \to AB$ und $A, B \in N_1$ folgt $N_1 \leftarrow N_1 \cup \{S\}$. $\Rightarrow N_1 = \{A, B, S\}$.

(c) $P_1 \leftarrow P \cup \{S \to B, S \to A, A \to a, B \to b\}$.

(d) $P_2 \leftarrow P_1 - \{A \to \varepsilon, B \to \varepsilon\}$

(2) Elimination nutzloser Symbole

(a) Ein Symbol x heisst erreichbar, falls es eine Ableitung $S \overset{*}{\Rightarrow} \alpha$ gibt, sodass X in α vorkommt.

Bestimmung der Menge erreichbarer Symbole: Sei $G = (\Sigma, N, P, S)$ eine kontextfreie Grammatik.

i. $E_N = \{S\}$ (erreichbare Nichtterminale), $E_\Sigma = \varnothing$ (erreichbare Terminale)

ii. Für alle $A \to \alpha$ in P mit $A = E_N$: Füge die Symbole von α zu E_N bzw. E_Σ hinzu. Wiederhole solange, bis sich E_N und E_Σ nicht mehr ändern.

iii. Reduziere die Grammatik zu $G' = (E_\Sigma, E_N, P', S)$ mit

$$P' = \{A \to \alpha \in P | A \in E_N\}$$

(b) Ein Nichtterminal x heisst produktiv (oder erzeugend), falls es eine Ableitung $x \overset{*}{\Rightarrow} w$ gibt für ein $w \in \Sigma^*$.

Bestimmung der Menge der produktiven Nichtterminale: Sei $G = (\Sigma, N, P, S)$ eine kontextfreie Grammatik.

i. $\text{Prod} := \{A \in N | A \to w \in P, w \in \Sigma^*\}$.

ii. Für alle $A \in N$: Falls $A \to \alpha \in P$ mit $\alpha \in (\Sigma \cup \text{Prod})^*$, füge A zu Prod hinzu. Wiederhole solange, bis sich Prod nicht mehr ändert.

iii. Reduziere die Grammatik zu $G' = (\Sigma, \text{Prod}, P', S)$ mit

$$P' = \{A \to \alpha \in P | A \in \text{Prod und } \alpha \in (\text{Prod} \cup \Sigma)^*\}$$

Beispiel. Betrachte die kontextfreie Grammatik

$$G = (\{a, b, c\}, \{S, A, B, C\}, P, S)$$

mit

$$P = \{S \to A, S \to AB, A \to Aa, B \to bB, B \to Bb, C \to c\}$$

Erreichbare Symbole: $\{S, A, B, a, b\}$

Also erhalten wir die Grammatik $G' = (\{a, b\}, \{S, A, B\}, P', S)$ mit

$$P' = \{S \to A, S \to AB, A \to Aa, A \to a, B \to bB, B \to Bb\}.$$

Produktive Nichtterminale in G': $\{S, A\}$.
Reduzierte Grammatik $G'' = (\{a, b\}, \{S, A\}, P'', S)$ mit

$$P'' = \{S \to A, A \to Aa, A \to a\}$$

Definition 39. Eine kontextfreie Grammatik G heisst reduziert, falls sie keine unproduktiven und keine nichterreichbaren Symbole enthält.

(3) Elimination von Kettenregeln (Regeln der Form $X \to Y$ mit $X, Y \in N$)

(a) Bestimme für jedes Nichtterminal A die Menge:

$$K(A) = \{D \in N | A \stackrel{*}{\Rightarrow} D\}$$

aller Nichtterminale, die in G aus A (nur mit Kettenregeln) abgeleitet werden können. Iterativ berechnen wir:

$$\begin{aligned} K(A) &:= \{A\} \\ K(A) &:= K(A) \cup \{X \in N | \text{ es existiert } C \in K(A) \text{ mit } C \to X \in P\} \end{aligned}$$

(b) Für jedes $X \in K(A)$ und jede Regel $X \to \alpha$, die keine Kettenregel ist, füge die Regel $A \to \alpha$ zu P' hinzu.

Beispiel. $G = (\{a, b, c, d\}, \{F, S\}, P, S)$ wobei

$$P = \{S \to F, F \to a, F \to bF, F \to d, F \to c\}$$

(a) $K(S) = \{S, F\}, K(F) = \{F\}$
(b) $P' = \{S \to a, S \to bF, S \to d, S \to c, F \to a, F \to bF, F \to d, F \to c\}$

21 Chomsky-Normalform

Definition 40. Zu jeder kontextfreien Grammatik G mit $\varepsilon \notin L(G)$ existiert eine äquivalente Grammatik G' in Chomsky-Normalform.

Ziel: Umwandlung einer beliebigen kontextfreien Grammatik in Chomsky-Normalform.

Theorem 5. Sei $G = (\Sigma, N, P, S)$ eine ε-freie, reduzierte Grammatik ohne Kettenregeln mit $\varepsilon \notin L(G)$.

Beweis. Sei G gemäss der Annahme. Forme alle Regeln, die nicht die gewünschte Form haben, um: Regeln der Form $A \to X_1 X_2 \ldots X_m, m \geq 2, X_i \in \Sigma \cup N$.

1. Falls $X_i \in \Sigma$, dann ersetze X_i durch neues Nichtterminal C_{X_i} und füge die Regel $C_{X_i} \to X_i$ hinzu.

2. Nach Schritt 1 haben alle nicht passenden Regeln die Form

$$A \to B_1 B_2 \dots B_m$$

für ein $m \geq 3$ und $B_i \in N$.

Zur Umformung dieser Regeln definiere neue Nichtterminale $D_1, \dots D_{m-2}$ und ersetze die Regel durch folgende Regeln:

$$\begin{aligned} A &\to B_1 D_1 \\ D_1 &\to B_2 D_2 \\ &\vdots \\ D_{m-3} &\to B_{m-2} D_{m-2} \\ D_{m-2} &\to B_{m-1} B_m \end{aligned}$$

□

Beispiel. $G = (\{a, b\}, \{S, A, B\}, P, S)$ mit $P = \{S \to bA, S \to aB, A \to bAA, A \to aS, A \to a, B \to aBB, B \to bS, B \to b\}$.

Transformation in Chomsky-Normalform:

1. Ersetzen:

Alte Regel	Neue Regel	
$S \to bA$	$S \to C_b A$	$C_b \to b$
$S \to aB$	$S \to C_a B$	$C_a \to a$
$A \to bAA$	$A \to C_b AA$	
$A \to aS$	$A \to C_a S$	
$B \to aBB$	$A \to C_a BB$	
$B \to bS$	$B \to C_b S$	

2. Noch nicht in Chomsky-Normalform: $A \to C_b AA$ und $B \to C_a BB$. Also:

Alte Regel	Neue Regel	
$A \to C_b AA$	$A \to C_b D_1$	$D_1 \to AA$
$B \to C_a BB$	$B \to C_1 E_1$	$E_1 \to BB$

Daraus folgt eine äquivalente kontextfreie Grammatik G' in Chomsky-Normalform:

$$G' = (\{a, b\}, \{S, A, B, C_a, C_b, D_1, E_1\}, P', S)$$

mit

$$P' = \dots$$

22 Anwendung der Chomsky-Normalform

Überprüfung, ob ein gegebenes Wort in der Sprache der Grammatik enthalten ist (sogenanntes Wortproblem). Motivation: Ist gegebenes Programm syntaktisch korrekt?

23 CYK (Cocke, Younger, Kasami)-Algorithmus

Algorithmus basiert auf dynamischer Programmierung, d.h. Zusammensetzen der Lösung aus Teillösungen.

Sei G eine kontextfreie Grammatik in Chomsky-Normalform. Sei ferner $w = a_1 a_2 \dots a_n \in \Sigma^n$. Idee: Bestimme für alle Teilwörter $a_i \dots a_j$ von w die Menge $V(i,j)$ aller Nichtterminale $X \in N$, sodass

$$X \stackrel{*}{\Rightarrow} a_i \dots a_j.$$

Dann ist $w \in L(G)$ genau dann, wenn $S \in V(1,n)$.

Beispiel. $G = (\{a,b\}, \{S,A,B,C\}, P, S)$ mit $P = \{S \to AB, S \to BC, A \to BA, A \to a, B \to CC, B \to b, C \to AB, C \to a\}$ und $w = bbab$.

1. Initialisierung der V-Mengen für Teilwörter der Länge 1.

   Für alle $i \in \{1, \dots, n\}$ gilt: $V(i,j) = \{X \in N | X \to a_i \in P\}$. Also:

   - $V(1,1) = \{B\}$, da $B \to b \in P$
   - $V(2,2) = \{B\}$
   - $V(3,3) = \{A, C\}$
   - $V(4,4) = \{B\}$

   Darstellung als Tabelle:

4				
3				
2				
1	B	B	A, C	B
Länge	b	b	a	a

2. Bestimmung der V-Mengen für Teilwörter der Länge 2: $a_i a_{i+1}$.

   -

$$\begin{aligned} V(1,2) &= \{X \in N | X \to YZ \in P \text{ mit } Y \in V(1,1) \wedge Z \in V(2,2)\} \\ &= \{X \in N | X \to YZ \in P \text{ mit } Y \in \{B\} \wedge Z \in \{B\}\} \\ &= \varnothing \end{aligned}$$

- $$\begin{aligned} V(2,3) &= \{X \in N | X \to YZ \in P \text{ mit } Y \in V(2,2) \wedge Z \in V(3,3)\} \\ &= \{X \in N | X \to YZ \in P \text{ mit } Y \in \{B\} \wedge Z \in \{A,C\}\} \\ &= \{S, A\}, \end{aligned}$$

  da $S \to BC$ und $A \to BA$.

4				
3				
2	$\varnothing$	S, A	C, S	
1	B	B	A, C	B
Länge	b	b	a	a

Bemerkung (Allgemeine Regel für längere Teilwörter)**.** Für alle $i, j \in \{1, \ldots, n\}$ mit $i < j$ ist

$$\begin{aligned} V(i,j) &= \{X \in N | \text{ es existiert ein } k \text{ mit } i \leq k < j, \\ &\quad \text{so dass } X \to YZ \in P \text{ mit } Y \in V(i,j) \text{ und } Z \in V(k,j)\} \end{aligned}$$

Beispielsweise lässt sich bba aufteilen als $b|ba$ ($k = 1$) oder $bb|a$ ($k = 2$).

3. Wörter der Länge 3:

4				
3	A	C, S		
2	$\varnothing$	S, A	C, S	
1	B	B	A, C	B
Länge	b	b	a	a

4. Wörter der Länge 4:

4	C, S			
3	A	C, S		
2	$\varnothing$	S, A	C, S	
1	B	B	A, C	B
Länge	b	b	a	a

24 Kellerautomaten (Pushdown Automata, PDA)

Ziel: Automatenmodell für kontextfreie Sprachen.

Idee: Nichtdeterministischer EA mit Zusatzspeicher in Form eines Stacks.

Arbeitsschritte (Übergänge):

Abhängig von Zustand, gelesenem Eingabesymbol und oberstem Symbol auf dem Stack:

- Bestimme neuen Zustand
- Entferne oberstes Symbol des Stacks und werfe beliebig viele Symbole auf den Stack.

Definition 41 (Kellerautomat)**.** Ein Kellerautomat wird beschrieben durch

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

wobei:

- Q: Menge der Zustände,
- Σ: Eingabealphabet (wer hätte das gedacht),
- Γ: Kelleralphabet,
- δ: Übergangsfunktion,
- $q_0 \in Q$: Startsymbol,
- $\bot$: Kellerbodensymbol ($\bot \in \Gamma, \bot \notin \Sigma$),
- F: Menge der akzeptierenden Zustände.

Die Übergangsfunktion ist definiert als

$$\delta : Q \times \Sigma \cup \{\varepsilon\} \times \Gamma \mapsto \mathcal{P}(Q \times \Gamma^*)$$

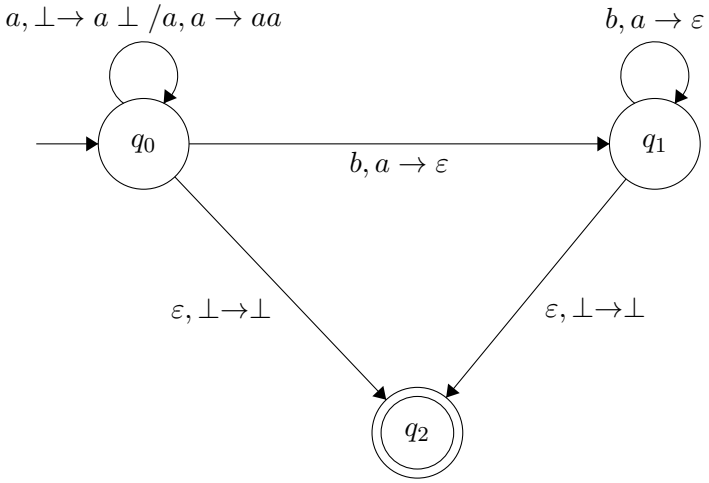
Ein Kellerautomat akzeptiert ein Wort w, falls es eine Berechnung auf w gibt, die die gesamte Eingabe liest und in einem akzeptierenden Zustand endet.

Beispiel. Kellerautomat für die Sprache $L = \{a^k b^k | k \geq 0\}$. Idee der Arbeitsweise: Der PDA liest alle as und speichert sie auf dem Keller, dann entfernt er für jedes gelesene b ein a aus dem Keller. Bei gleicher Anzahl as und bs akzeptiert er, sonst hat er keine Möglichkeit zum Weiterarbeiten, also verwirft er.

$$A = (\{q_0, q_1, q_2\}, \{a, b\}, \{A, \bot\}, \delta, q_0, \bot, \{q_2\}$$

mit

$$\begin{aligned}
\delta(q_0, \varepsilon, \bot) &= \{(q_2, \bot)\} \\
\delta(q_0, a, \bot) &= \{(q_0, a)\} \\
\delta(q_0, a, a) &= \{(q_0, aa)\} \\
\delta(q_0, b, a) &= \{(q_1, \varepsilon)\} \\
\delta(q_1, b, a) &= \{(q_1, \varepsilon)\} \\
\delta(q_1, \varepsilon, \bot) &= \{(q_2, \bot)\}
\end{aligned}$$



Berechnung von A auf dem Wort $aaabbb$:

$$\begin{aligned}
&(q_0, aaabbb, \bot) \to (q_0, aabbb, a \bot) \to (q_0, abbb, aa \bot) \to \\
&(q_0, bbb, aaa \bot) \to (q_1, bb, aa \bot) \to (q_1, b, a \bot) \to \\
&(q_1, \varepsilon, \bot) \to (q_2, \varepsilon, \bot)
\end{aligned}$$

25 Äquivalenz von kontextfreier Grammatik und PDA

Theorem 6. Sei G eine kontextfreie Grammatik. Dann gibt es einen PDA A, so dass $L(A) = L(G)$.

Idee des Beweises: Simuliere die Schritte einer Ableitung von G auf dem Keller A. Beobachtung: Sei G eine kontextfreie Grammatik, sei $w \in L(G)$. Dann gibt es eine Ableitung von w in G, in der in jedem Ableitungsschritt das jeweils am weitesten links stehende Nichtterminal abgeleitet wird (sogenannte Linksableitung).

Beweis. Der zu G äquivalente PDA A arbeitet wie folgt:

- Falls das oberste Symbol auf dem Stack gleich dem nächsten Eingabesymbol ist, werden beide Symbole weggelesen.

- Falls das oberste Symbol auf dem Stack ein Nichtterminal ist, wird es durch die rechte Seite einer passenden Regel ersetzt. Dabei wird nichts von der Eingabe gelesen (ε-Übergang).

□

26 Turingmaschine

Motivation:

Berechenbarkeit Zeigen, dass es Probleme gibt, die nicht lösbar (entscheidbar) sind.

Komplexität Zeigen, dass es Probleme gibt, die nicht handhabbar sind.

Ziel: Berechnungsmodell, das so allgemein wie möglich, aber trotzdem einfach aufgebaut ist.

Definition 42 (Turingmaschine)**.** Eine Turingmaschine (TM) ist ein 6-Tupel

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F),$$

wobei

- Q: Zustandsmenge
- Σ: Eingabealphabet, $\textvisiblespace \notin \Sigma$
- Γ: Arbeitsalphabet, $\textvisiblespace \in \Gamma$, $\Sigma \subset \Gamma$
- q_0: Anfangszustand
- F: Akzeptierende Zustände
- δ: Transitionsfunktion mit

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R, N\}$$

27 Berechnung eine TM auf der Eingabe x

Folge von Konfigurationen $K_1, \ldots, K_n$, so dass

- K_1 ist Anfangskonfiguration (q_0, x) mit Kopf auf dem ersten Symbol der Eingabe
- K_{i+1} entsteht aus K_i durch die Anwendung einer Transition von M

Die TM M akzeptiert das Wort x, falls die Berechnung von M auf x in einem akzeptierenden Zustand hält.

$$L(M) = \{w \in \Sigma^* | M \text{ akzeptiert } w\}$$

Definition 43. Eine Sprache L heisst rekursiv aufzählbar (semi-entscheidbar), falls es eine TM M gibt, so dass

$$L = L(M).$$

$\mathcal{L}_{\mathrm{RE}} = \{L(M) | M \text{ ist eine TM } \}$ ist die Menge aller rekursiv aufzählbaren Sprachen.

Bemerkung. Beobachtung: Eine Berechnung einer TM kann unendlich sein. Wenn das Wort nicht in der Sprache ist, kann M entweder verwerfen oder unendlich lange rechnen.

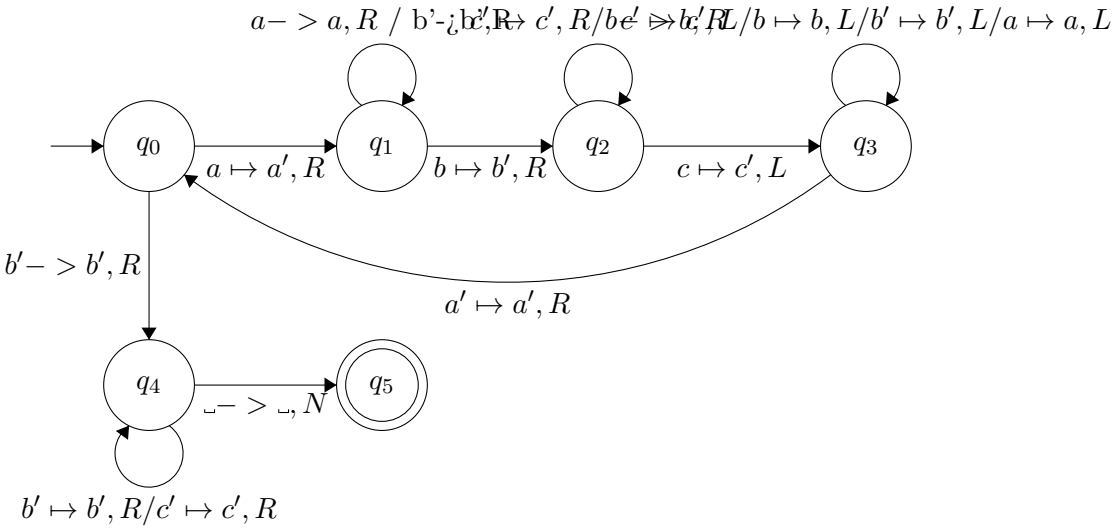
Definition 44. Eine Sprache heisst rekursiv (entscheidbar), wenn es eine TM M gibt mit $L = L(M)$, die auf jeder Eingabe w hält, und zwar in einem akzeptierenden Zustand für alle $w \in L$ und in einem nichtakzeptierenden Zustand für alle $w \notin L$.

$$\mathcal{L}_{\mathrm{R}} = \text{Menge aller rekursiven Sprachen. } \mathcal{L}_{\mathrm{R}} \subseteq \mathcal{L}_{\mathrm{RE}}$$

Beispiel. TM für $L = \{a^k b^k c^k | k \geq 0\}$. Idee: Wiederhole solange es unmarkierte as gibt: markiere das a, gehe nach rechts bis zum ersten nichtmarkierten b und markiere es, gehe nach rechts bis zum ersten c und markiere es, gehe links bis zum linksten unmarkierten a.

Überprüfe von kinks nach rechts ob alle Eingaben markiert sind.

Graphische Darstellung:



Theorem 7. Eine TM mit mehreren Bändern kann simuliert werden durch eine TM mit einem Band.

Beweis. Idee: Grösseres Alphabet, das ale Bänder und Kopfpositionen beschreibt.

28 Universelle Turingmaschine

Ziel: Konstruktion einer TM, die alle anderen Turingmaschinen simulieren kann.

Sei M eine beliebige TM mit einem Band. M kann vollständig beschrieben werden durch die Transitionstabelle:

Jede Transition ist darstellbar als ein Wort

$$\underbrace{p}_{\text{aktueller Zustand}}\ \underbrace{A}_{\text{Arbeitssymbol}}\ \underbrace{q}_{\text{neuer Zustand}}\ \underbrace{B}_{\text{neues Bandsymbol}}\ \underbrace{S}_{\text{Kopfbewegung}}$$

Die Turingmaschine M ist darstellbar als Wort:

$$\text{Kod}(M) = q_0\#q_{\text{accept}}\#q_{\text{reject}}\#t_1\#t_2\#\ldots\#t_n$$

über dem Alphabet $Q \cup \Gamma \cup \{L, R, N\} \cup \{\#\}$, wobei t_i die wie oben codierte Transitionen von M sind.

Bemerkung. Im Prinzip ist auch eine Binärcodierung möglich.

Konstruktion einer universellen Turingmaschine U:
Eingabe: $\text{Kod}(M)\#\#w$ für eine TM M, die auf dem Wort w simuliert werden soll.
Arbeitsweise von U:

1. U überprüft, ob $\text{Kod}(M)$ eine gültige Codierung einer TM ist. Falls nicht, verwirft sie die Eingabe.
2. U schreibt die Anfangskonfiguration von M auf w auf das Band.
3. M simuliert die Berechnung von M auf w wie folgt: Solange der Zustand in der aktuellen Konfiguration von M nicht q_{accept} oder q_{reject} ist, generiere die Nachfolgekonfiguration von M auf w auf dem Band (suche passende Transition in der Eingabe und führe diese Änderung durch).
4. Wenn der Zustand von M q_{accept} ist, dann akzeptiere die Eingabe, sonst verwerfe.

Bemerkung. Falls M auf w unendlich lange läuft, dann auch U.

Behauptung: Turingmaschinen sind ein geeignetes formales Modell zur Beschreibung der Rechenstärke realer Computer.

Hierfür: Zeige, wie man ein Programm einer beliebigen Programmiersprache durch eine äquivalente TM darstellt.

Beobachtung: Es reicht, eine solche Transformation für Assembler-Programme anzugeben.

Idee der Transformation:

- Eingabeband mit Programmzeile 1#Programmzeile 2#...#letzte Programmzeile
- Arbeitsband mit Register 1#Register 2#...
- Hauptspeicherband

- Arbeitsband zum Rechnen (Adressberechnungen etc)

Beobachtung: Alle bisher entwickelten Modelle zur Beschreibung der algorithmischen Lösbarkeit sind äquivalent zum Modell der Turingmaschine.

Definition 45 (Church'sche These)**.** Das Modell der TM, die immer hält, ist eine geeignete Formalisierung des intuitiven Begriffs "Algorithmus", d.h. die Klasse der rekursiven (entscheidbaren) Sprachen stimmt mit der Klasse der algorithmisch erkennbaren Sprachen überein.

29 Aufzählung von Wörtern

Definition 46 (Kanonische Reihenfolge)**.** Sei $\Sigma = \{a_1, a_2, \ldots, a_n\}$ ein Alphabet. Es sei $a_1 < a_2 < \cdots < a_n$. Dann ist die kanonische Reihenfolge auf Σ^* definiert durch

$$w <_{\text{kan}} u$$

genau dann, wenn

- $|w| < |u|$
- $|w| = |u| = k, w = a_1 \ldots a_k, u = b_1 \ldots b_k$: Es gibt ein i mit $1 \leq i \leq k$, so dass $a_j = b_j$ für $j < i$ und $a_i < b_i$.

Beispiel. $ba <_{\text{kan}} aba$ und $abb <_{\text{kan}} aca$

30 Aufzählung von Turingmaschinen

Sei $\mathcal{M}$ die Menge aller TM. Dann ist die kanonische Reihenfolge auf $\mathcal{M}$ definiert durch $M_1 <_{\text{kan}} M_2$ genau dann, wenn $\text{Kod}(M_1) <_{\text{kan}} \text{Kod}(M_2)$.

31 Berechenbarkeit

Ziel: Zeigen, dass es keine TM U gibt, die hält, die für eine gegebene TM M und ein gegebenes Wort W entscheidet, ob $w \in L(M)$ gilt.

Motivation: Gemäss der Church'schen These bedeutet dies, dass es nicht möglich ist, ein Programm zu schreiben, das beliebige andere Programme verifiziert.

32 Die Diagonalisierungssprache

Idee: Wir zeigen, dass es mehr Sprachen als TMs gibt. Das heisst, es gibt (unendlich viele) Sprachen, für die es keine TM gibt.

32.1 Methode der Diagonalisierung

Wir haben gesehen, dass wir alle Wörter und TMs aufzählen können.

Sei w_i das i-te Wort in kanonischer Reihenfolge, sei M_i die i-te Turingmaschine in kanonischer Reihenfolge. Wir konstruieren eine (unendlich grosse) Tabelle D mit Einträgen $\{0, 1\}$, so dass

$$D(i, j) = 1$$

genau dann, wenn M_i das Wort w_j akzeptiert. Beispielsweise:

D	w_1	w_2	w_3	$\dots$	w_i	$\dots$
M_1	0	1	0	$\cdots$	0	$\vdots$
M_2	1	1	0	$\cdots$	1	$\vdots$
M_3	0	0	1	$\cdots$	1	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
M_i	0	0	0	$\cdots$	0	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\dots$	$\vdots$	$\vdots$	$\ddots$

Wir betrachten die Diagonaleinträge $D(i, i)$:

$D(i, i)$ beschreibt, ob die TM M_i das Wort w_i mit gleichem Index der Aufzählung akzeptiert.

Wir definieren nun die Sprache:

$$\begin{aligned} L_{\mathrm{d}} &= \{w \in \{0,1\}^* | w = w_i \text{ und } M_i \text{ akzeptiert } w_i \text{ nicht}\} \\ &= \{w \in \{0,1\}^* | w = w_{\text{ und } D(i,i=0}\} \end{aligned}$$

In der Tabelle: Alle Spalten mit einer 0 in der Diagonalen.

Theorem 8. Es gibt keine TM, die L_{d} akzeptiert.

Beweis. Angenommen, es gibt eine TM M, die L_{d} akzeptiert. Da wir in der Tabelle D alle TMs aufgezählt haben, gibt es einen Index j, so dass $M = M_j$, also gilt, dass

$$L_{\mathrm{d}} = L(M_j)$$

Für den gewünschten Widerspruch betrachten wir das entsprechende Wort w_j. Es gilt:

$$w_j \in L(M_j) \Leftrightarrow D(j, j) = 1$$

nach Definition von D. Aber ebenfalls gilt:

$$w_j \in L_{\mathrm{d}} \Leftrightarrow D(j, j) = 0$$

nach Definition von L_{d}.

Es kann also nicht $L_{\mathrm{d}} = L(M_j)$ gelten, also war die Annahme falsch, somit gibt es keine TM, die L_{d} akzeptiert.

□

Wiederholung:

- Eine Sprache L heisst rekursiv aufzählbar, wenn es eine TM M gibt, die L akzeptiert. (Wenn $w \notin L$, darf M unendlich lange laufen.)
- Eine Sprache L heisst rekursiv (entscheidbar), wenn es eine TM M gibt, die L entscheidet:
  - für alle $w \in L$: akzeptiert
  - für alle $w \notin L$: verworfen.

  M darf nicht unendlich lange laufen.

33 Reduktion

Wir wollen die Unlösbarkeit von Sprachen auf andere Sprachen ausweiten. Die Idee ist: eine Reduktion "leichter oder gleich schwer" bezüglich der algorithmischen Lösbarkeit einführen.

Definition 47. Seien L_1 und L_2 zwei Sprachen. L_1 ist auf L_2 rekursiv reduzierbar, geschrieben:

$$L_1 \leq_{\mathrm{R}} L_2,$$

falls es eine immer-haltende Turingmaschine A gibt, die L_2 entscheidet und man daraus eine TM B bauen kann, die L_1 entscheidet.

Existiert eine Reduktion $L_1 \leq \mathrm{R}L_2$, dann gilt

- Ist L_1 unentscheidbar, dann ist L_2 unentscheidbar.
- Ist L_1 nicht rekursiv aufzählbar, dann ist L_2 ebenfalls nicht rekursiv aufzählbar.

34 Komplemente rekursiver Sprachen

Sei $\bar{L} = \Sigma^* - L$ das Komplement von L.

Theorem 9. L ist rekursiv. Daraus folgt, dass $\bar{L}$ rekursiv ist.

Beweis. Wir zeigen, dass $\bar{L} \leq_{\mathrm{R}} L$.

Sei A eine TM, die L entscheidet. Eine TM B, die $\bar{L}$ entscheidet, lässt sich nach folgendem Schema bauen:

(Biltli) □

Theorem 10. $\bar{L}\mathrm{d}$ ist rekursiv aufzählbar.

$$\bar{L}\mathrm{d} = \{w \in \{0,1\}^* | w = w_i \text{ und } M_i \text{ akzeptiert } w_i \}$$

Eine TM D, die $\bar{L}_{\mathrm{d}}$ akzeptiert, kann wie folgt arbeiten:

1. Berechne i, so dass w das i-te Wort w_i in der kanonischen Reihenfolge ist
2. Generiere die $\text{Kod}(M_i)$ der i-ten TM M_i
3. Simuliere die Berechnung von M_i auf dem Wort w_i:
   - Falls M_i das Wort w_i akzeptiert, dann akzeptiert auch D.
   - Falls M_i das Wort w_i verwirft, dann verwirft auch D.
   - Falls M_i auf w_i unendlich lange arbeitet, dann auch D.

35 Die universelle Sprache

Zeigen, dass eine generelle Programmverifikation nicht möglich ist.

Sei $L_\text{u} = \{\text{Kod}(M)\#|M \text{ ist eine TM, die } w \text{ akzeptiert }\}$ die universelle Sprache.

Wir haben bereits gesehen, dass es eine universelle TM gibt, also ist L_u rekursiv aufzählbar: $L_\text{u} \in \mathcal{L}_\text{RE}$.

Theorem 11. $L_\text{u} \notin \mathcal{L}_\text{R}$.

Beweis. Idee: wir zeigen, dass wir eine TM, die L_u entscheidet, verwenden können, um $\bar{L_\text{d}}$ zu entscheiden. Da wir bereits wissen, dass $\bar{L_\text{d}} \notin \mathcal{L}_\text{R}$, kann L_u nicht rekursiv sein.

Anders formuliert: Wir zeigen $\bar{L_\text{d}} \leq_\text{R} L_\text{u}$, d.h. wir zeigen, dass L_u mindestens so schwer ist wie $\bar{L_\text{d}}$.

Sei A eine TM, die L_u entscheidet. Wir bauen daraus eine TM B, die $\bar{L_\text{d}}$ akzeptiert.

Die TM B hält immer, weil wir angenommen haben, dass A immer hält. Daraus folgt B entscheidet $\bar{L_\text{d}}$.

Dies ist ein Widerspruch, da wir schon gezeigt haben, dass $\bar{L_\text{d}} \notin \mathcal{L}_\text{R}$. Da die TM C offenbar existiert, muss die Annahme falsch gewesen sein, dass es eine TM A gibt, die L_u entscheidet.

Daraus folgt $\bar{L_\text{u}} \notin \mathcal{L}_\text{R}$.

□

36 Asymptotische Komplexitätsmessung

Definition 48. Seien $f, g : \mathbb{N} \to \mathbb{R}^+$ zwei Funktionen. Dann gilt

$$f(n) \in \mathcal{O}(g(n)),$$

falls ein $n \in \mathbb{N}$ und ein $c \in \mathbb{N}$ existiert, so dass für $n \geq n_0$ gilt:

$$f(n) \leq c \cdot g(n).$$

Definition 49. Seien $f, g : \mathbb{N} \to \mathbb{R}^+$ zwei Funktionen. Dann gilt

$$f(n) \in \Omega(g(n)),$$

falls ein $n \in \mathbb{N}$ und ein $d \in \mathbb{N}$ existiert, so dass für $n \geq n_0$ gilt:

$$f(n) \geq \frac{1}{d} \cdot g(n).$$

Definition 50. Seien $f, g : \mathbb{N} \to \mathbb{R}^+$ zwei Funktionen. Dann gilt

$$f(n) \in \Theta(g(n)),$$

falls $f(n) \in \mathcal{O}(g(n))$ und $f(n) \in \Omega(g(n))$ gilt.

Bemerkung (Allgemeine Beobachtung)**.**

- Kostante Vorfaktoren kann man weglassen.
- Bei Polynomen ist nur die höchste Potenz entscheidend.
- $\mathcal{O}$-Notation ist transitiv: Aus

  $$f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(h)$$

  folgt

  $$f \in \mathcal{O}(h).$$

37 Komplexitätstheorie

Ziel: Klassifizierung von (entscheidbaren) Problemen nach ihrer Schwierigkeit (Komplexität). Die Komplexitätstheorie ist die Theorie der quantitativen Gesetze und Grenzen der algorithmischen Informationsverarbeitung.

38 Verschiedene Komplexitätsmasse

Zeitkomplexität Laufzeit des besten Programms, das das Problem löst

Platzkomplexität Speicherplatzbedarf des besten Programms

Beschreibungskomplexität Länge des kürzesten Programms

Definition 51. Im Formalismus der Turingmaschine: Sei M eine TM mit Eingabealphabet Σ, die immer hält und sei $w \in \Sigma^*$.

Dann ist der Zeitbedarf von M auf der Eingabe w definiert als:

$$\text{Time}_M(w) = \text{ Anzahl der Konfigurationsübergänge in der Berechnung von } M \text{ auf } w.$$

Verallgemeinerung: Zeitkomplexität in Abhängigkeit der Eingabelänge: Für $n \in \mathbb{N}$ ist:

$$\text{Time}_M(n) = \max\{\text{Time}_M(w) || w| = n\}$$

der Zeitbedarf von M auf Eingaben der Länge genau n im schlechtesten Fall.

39 Exkurs: Alternatives Modell einer Turingmaschine

¢, \$: Endmarkierungen. Eingabeband: Nur Lesekopf, Arbeitsband: Lese-/Schreibkopf.

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

mit

- Q: Zustandsmenge,
- Σ: Eingabealphabet, $\sqcup, ¢, \$ \notin \Sigma$
- Γ: Arbeitsalphabet, $\sqcup, ¢, \$ \notin \Sigma$
- q_0: Anfangszustand
- q_{accept}: akzeptierender Zustand
- q_{reject}: verwerfender Zustand
- δ: Transitionsfunktion, definiert als:

$$\delta : Q \times (\Sigma \cup \{¢, \$\}) \times (\Lambda \cup \{¢\}) \to Q \times \{L, R, N\} \times \Lambda \times \{L, R, N\}$$

Beispiel (Zeitkomplexität einer TM zum Vergleich von zwei Wörtern x und y der Länge jeweils n)**.** Nun:

- x auf Arbeitsband kopieren: $\mathcal{O}(n)$
- Lesekopf auf dem Arbeitsband an den Anfang zurückverschieben: $\mathcal{O}(n)$
- Vergleiche Zeichen für Zeichen: $\mathcal{O}(n)$

Also Total: $\mathcal{O}(n)$.

40 Zeitkomplexität auf Problemen

im Allgemeinen nicht exakt bestimmbar.

Definition 52. Seien $f, g : \mathbb{N} \to \mathbb{R}^+$. Sei U ein (entscheidbares) Problem.

- $\mathcal{O}(f(n))$ ist eine obere Schranke für die Zeitkomplexität von U, falls eine TM M existiert, die U löst und eine Zeitkomplexität $\in \mathcal{O}(f(n))$ hat.
- $\Omega(g(n))$ ist eine untere Schranke für die Zeitkomplexität von U, falls für alle TM M, die U lösen, gilt, dass $\text{Time}_M(n) \in \Omega(f(n))$.

Bemerkung. Eine obere Schranke zu finden, ist einigermassen einfach, wogegen das Finden einer unteren Schranke sehr kompliziert ist.

Idee: Zur Klassifizierung von Problemen nach Schwierigkeit -¿ verwende relative Schwierigkeit. "Wenn Problem A schnell lösbar wäre, dann auch tausend andere Probleme."

Was heisst "schnell lösbar"? Üblicher Ansatz: Problem U heisst in Polynomzeit lösbar, wenn es eine obere Schranke $\mathcal{O}(n^c)$ gibt für eine Konstante $c \geq 1$.

Definition 53. Die Klasse der in Polynomzeit entscheidbaren Sprachen nennen wir P.

41 Nichtdeterministische Turingmaschine

Prinzipieller Aufbau:

Eingabeband, Arbeitsband gleich wie vorher.

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

wobei

$$\delta : Q \times (\Sigma \cup \{¢, \$\}) \times (\Gamma \cup \{¢\}) \to 2^{Q \times \{L,R,N\} \times \Gamma \times \{L,R,N\}}$$

Die Menge aller Berechnungen einer NTM kann dargestellt werden als Berechnungsbaum.

Wurzel: Anfangskonfiguration, Blätter: akzeptierende oder verwerfende Berechnungen.

Eine NTM akzeptiert, wenn es einen Pfad im Baum gibt von der Wurzel zu einem akzeptierenden Blatt.

Bemerkung. Wichtig! Eine NTM ist nicht ausdrucksstärker als eine deterministische TM.

Theorem 12. Sei M eine NTM. Dann gibt es eine äquivalente deterministische Turingmaschine A, also eine DTM A mit $L(A) = L(M)$.

Beweis. Idee: Breitensuche auf dem Berechnungsbam. A arbeitet wie folgt:

1. A schreibt Anfangskonfiguration auf das erste Arbeitsband.
2. prüft, ob eine akzeptierende Konfiguration auf dem ersten Band ist. Falls ja: halten, akzeptieren.
3. auf das zweite Band: alle Nachfolgekonfigurationen. Falls es keine Nachfolgekonfigurationen gibt, hält A und verwirft
4. A löscht erstes Band, kopiert Inhalt des zweiten Bands auf das erste, löscht dann das zweite Band und geht zu Schritt 2.

Falls M eine akzeptierende Konfiguration erreicht, akzeptiert natürlich auch A. Falls jede Berechnung von M endlich ist, hält auch A. □

Bemerkung (Zeitkomplexität einer NTM)**.** Länge der kürzesten akzeptierenden Berechnung.

Definition 54. Die Klasse aller von einer NTM in Polynomzeit entscheidbaren Sprachen nennen wir NP.

Theorem 13. NP = Menge aller Sprachen, für die ein Polynomzeit-Verifizierer existiert.

Bemerkung. P entspricht: Lösung finden in Polynomzeit
NP entspricht: Lösung verifizieren in Polynomzeit

Bemerkung. Offene Frage: Gilt P = NP? Vermutung: P ≠ NP

42 NP-Vollständigkeit

Konzept zum Beweis relativer Schwierigkeit von Problemen.

42.1 Polynomielle Reduktion

Seien $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ zwei Sprachen. L_1 ist polynomiell auf L_2 reduzierbar:

$$L_1 \leq_p L_2,$$

falls eine TM M existiert, mit $\text{Time}_M(m) \in \mathcal{O}(n^k)$ für ein $k \in \mathbb{N}$, die für eine Eingabe $x \in \Sigma_1^*$ ein Wort $M(x) \in \Sigma_2^*$ berechnet mit

$$x \in L_1 \Leftrightarrow M(x) \in L_2.$$

$L_1 \leq_p L_2$ bedeutet, dass L_2 mindestens so schwer ist (bezüglich der Lösbarkeit in polynomieller Zeit) wie L_1.

Wenn $L_1 \leq_p L_2$ gilt, dann könnte man eine polynomielle TM B für L_2 dafür verwenden, eine polynomielle TM A für L_1 zu bauen.

Definition 55. Eine Sprache L heisst NP-schwer, falls für alle Sprachen $L' \in$ NP gilt, dass $L' \leq_p L$.

Eine Sprache L heisst NP-vollständig, falls

1. $L \in$ NP und
2. L ist NP-schwer.

Die Menge der NP-vollständigen Sprachen betrachten wir als die gesuchte Teilklasse von schwersten Entscheidungsproblemen in NP.

Bemerkung (Beobachtung)**.** Wenn es gelingt, für ein NP-schweres L nachzuweisen, dass $L \in$ P, dann gilt P = NP.

42.2 Ein erstes NP-vollständiges Problem

Definition 56 (Boole'sche Formel)**.** Boole'sche Formeln sind induktiv definiert durch:

1. Die Konstanten 0 und 1 und die Variablen $x_1, x_2, \ldots x_n$ sind Boole'sche Formeln.
2. Negation: Wenn φ eine Boole'sche Formel ist, dann auch $\neg\varphi$.
3. Konjunktion: Wenn φ und ψ Boole'sche Formeln sind, dann auch $(\varphi \wedge \psi)$.
4. Disjunktion: Wenn φ und ψ Boole'sche Formeln sind, dann auch $(\varphi \vee \psi)$.

Bemerkung (Konvention zur Klammerersparnis)**.** :

- $\neg$ bindet stärker als $\wedge$ und $\vee$
- $\wedge$ bindet stärker als $\vee$
- statt $\neg x_i$ schreiben wir $\bar{x_i}$.

Definition 57. Eine Boole'sche Formel ist in konjunktiver Normalform (KNF), wenn sie eine Konjunktion von Disjunktionen von einfachen und negierten Variablen ist.

Beispiel. KNF:

$$(x_1 \vee x_2 \vee \bar{x_3}) \wedge (x_5 \vee x_6)$$

Definition 58 (Belegung)**.** Eine Belegung ist eine Funktion, die jeder Variablen einen Wert aus $\{0, 1\}$ zuordnet. Eine Belegung erfüllt eine Formel φ, wenn die Auswertung der Formel nach den üblichen Regeln der Aussagelogik den Wert 1 ergibt.

Definition 59 (Erfüllbarkeit)**.** Eine Formel ist erfüllbar, wenn es eine Belegung dafür gibt.

Definition 60 (SAT)**.** SAT ist das Problem, zu entscheiden, ob eine gegebene Formel in KNF erfüllbar ist.

Theorem 14 (Satz von Cook)**.** SAT ist NP-vollständig.

Beweisidee.
- SAT $\in$ NP: P-Verifizierer für SAT. Zeige $l_1 l_2 \ldots l_n$ mit $l_i = 1$ falls $x_i = 1$ für eine erfüllbare Belegung und $l_i = 0$ analog. Die Prüfung ist dann polynomial.
- SAT ist NP-schwer: $L \leq_p$ SAT für alle $L \in$ NP. Kodiere die Berechnung einer polynomialzeitbeschränkten NTM durch eine KNF-Formel.

□

42.3 Nachweis der NP-Vollständigkeit weiterer Probleme durch Reduktion auf SAT

(oder andere Probleme, die NP-vollständig sind)

Satz: Wenn P_1 NP-vollständig ist und P_2 in NP enthalten ist und eine polynomielle Reduktion von $P_1 \leq_p P_2$ existiert, dann ist P_2 NP-vollständig.

Theorem 15. CLIQUE ist NP-vollständig.

Beweis. 1. CLIQUE $\in$ NP. Siehe vorher.

2. SAT $\leq_p$ CLIQUE. Sei $\varphi = F_1 \wedge F_2 \wedge \cdots \wedge F_m$ mit $F_i = L_{i1} \vee L_{i2} \ldots L_{ik}$, $k \in \mathbb{N}$ für $1 \leq i < m$ eine Formel in KNF .

   Konstruiere eine Eingabe (G, k) für CLIQUE, so dass:

   $$\varphi \in \mathsf{SAT} \Leftrightarrow (G, k) \in \text{CLIQUE}.$$

   Setze $k = m$. $G = (V, E)$ mit $V = \{[i, j] | 1 \leq i \leq m, 1 \leq j \leq k\}$. (ein Knoten für jedes Auftreten eines Literals in φ.)

   $$E = \{\{[i, j], [r, s]\} | [i, j], [r, s] \in V, i \neq r, l_{ij} \neq \bar{l_{rs}}\}$$

   (Kanten zwischen Knoten aus unterschiedlichen Klauseln, alls Literale nicht Negationen voneinander sind)

   Bleibt zu zeigen: φ erfüllbar $\Leftrightarrow$ G enthält CLIQUE der Grösse k.

   Idee: Keine Kante zwischen x und $\bar{x}$. Daraus folgt: Jede Clique in G entspricht einer Menge von gleichzeitig auf 1 setzbaren Literalen. Daraus folgt: eine Clique der Grösse k enthält einen Knoten pro Klausel, also ist jede Klausel erfüllt.

   Umgekehrt: erfüllende Belegung definiert eine Clique der Grösse k.

□

Definition 61 (Vertex Cover-Problem (VC))**.** Gegeben ist ein ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl $k \leq |V|$.

Frage ist: Gibt es eine Menge $S \subseteq V$ mit $|S| = k$, sodass jede Kante mindestens einen Endpunkt in S hat?

Theorem 16. CLIQUE $\leq_p VC$.

Beweis. Sei $G = (V, E)$ und $k \in \mathbb{N}$ eine Eingabe des Clique-Problems. Wir konstruieren hieraus einen Eingabe für das VC wie folgt:

(G', m), wobei $G' = (V, E')$ (E': Komplementärgraph) und $m = |V| - k$ □

Bemerkung (Beobachtung)**.** Falls S eine Clique in G ist, dann ist $V - S$ ein VC in G' und umgekehrt.