Grundlagen der Theoretischen Informatik

Vorlesungsskript

von

V. Claus und E.-R. Olderog

Ausgabe Wintersemester 2019/20

Inhaltsverzeichnis

Kapitel I

Grundbegriffe

§1 Modellbildungen in der Theoretischen Informatik

Kennzeichen der Theorie:

- vom Speziellen zum Allgemeinen
- Modellbildung zur Beantwortung allgemeiner Fragen
- Analyse und Synthese von Modellen

Frage: Was ist zur Syntaxbeschreibung von Programmiersprachen notwendig?
Modellbildung Sprachen:
Wir betrachten CHOMSKY-Sprachen und als Spezialfälle

- reguläre Sprachen
- kontextfreie Sprachen

Wir werden zeigen: Reguläre Sprachen werden von endlichen Automaten erkannt.
Kontextfreie Sprachen werden von Kellerautomaten erkannt.

Frage: Wie lassen sich parallele und kommunizierende Systeme beschreiben?
Modellbildung Prozesse und Prozesskalküle:

- Operatoren für Parallelität und Kommunikation

Frage: Wie lassen sich zeitkritische Systeme beschreiben?
Modellbildung Realzeitautomaten:

- Erweiterung endlicher Automaten um Uhren

Frage: Was ist mit dem Computer berechenbar?
Modellbildung Computer:

- Turingmaschinen (1936)

- Grammatiken (1959)

Wir werden zeigen: Diese Ansätze sind äquivalent. Es gibt nicht berechenbare Funktionen. Beispiele nicht berechenbarer Funktionen werden explizit angegeben. Endliche Automaten und Kellerautomaten können als Spezialfälle von Turingmaschinen aufgefasst werde.

Frage: Wieviel Zeit und Speicherplatz braucht eine Rechnung?
Modellbildung Komplexität:

- „schnelle“, d.h. polynomielle Algorithmen
- die Problemklassen P und NP

Offenes Problem: Ist $P = NP$?

Themen der Vorlesung:

- Automatenmodelle
- Formale Sprachen und Grammatiken
- Berechenbarkeit
- Komplexität

§2 Logik, Mengen, Relationen und Funktionen

Im Folgenden sind die Notationen zusammengestellt, die wir hierzu in diesem Skript benutzen.

- In logischen Formeln benutzen wir die Junktoren $\neg$ (*Negation*), $\wedge$ (*Konjunktion*), $\vee$ (*Disjunktion*), $\Rightarrow$ (*Implikation*) und $\Leftrightarrow$ (*Äquivalenz*) sowie die Quantoren $\forall$ (*Allquantor*) und $\exists$ (*Existenzquantor*).

- Endliche Mengen können durch Auflistung ihrer Elemente beschrieben werden, beispielsweise $\{a, b, c, d\}$ oder $\{Kaffee, Tee, Zucker\}$. Als Spezialfall ergibt sich die *leere Menge* $\{\}$, die auch mit $\emptyset$ bezeichnet wird. Ein wichtiges Beispiel einer unendlichen Menge ist die Menge $\mathbb{N}$ der *natürlichen Zahlen*: $\mathbb{N} = \{0, 1, 2, 3, ...\}$.
  $x \in X$ besagt, dass x *Element* der Menge X ist, und $x \notin X$ besagt, dass x *nicht* Element der Menge X ist.
  Es gilt das *Extensionalitätsprinzip*, d.h. zwei Mengen sind genau dann gleich, wenn sie dieselben Elemente haben. Zum Beispiel gilt

  $$\{a, b, c, d\} = \{a, a, b, c, d, d\}.$$

  $X \subseteq Y$ besagt, dass X *Teilmenge* von Y ist, d.h. es gilt $\forall\, x \in X : x \in Y$. Aus gegebenen Mengen können Teilmengen mit Hilfe von logischen Formeln *ausgesondert* werden. Zum Beispiel beschreibt

  $$M = \{n \in \mathbb{N} \mid n \; mod \; 2 = 0\}$$

die Menge aller geraden natürlichen Zahlen.

Für Mengen $X, Y \subseteq Z$ gibt es die üblichen mengentheoretischen Verknüpfungen

$$\begin{array}{lrcl} \textit{Vereinigung}: & X \cup Y & = & \{z \in Z \mid z \in X \vee z \in Y\} \\ \textit{Durchschnitt}: & X \cap Y & = & \{z \in Z \mid z \in X \wedge z \in Y\} \\ \textit{Differenz}: & X - Y & = & X \backslash Y = \{z \in Z \mid z \in X \wedge z \notin Y\} \\ \textit{Komplement}: & \overline{X} & = & Z - X \end{array}$$

Wir schreiben ferner $X \dot{\cup} Y$ für die *disjunkte Vereinigung* von X und Y, d.h. um auszudrücken, dass zusätzlich $X \cap Y = \emptyset$ gilt.

$|X|$ und $card(X)$ bezeichnen die *Kardinalität* oder *Mächtigkeit* der Menge X. Für endliche Mengen X ist $|X|$ die Anzahl der Elemente von X. So ist $|\{\textit{Kaffee}, \textit{Tee}, \textit{Zucker}\}| = 3$.

$\mathcal{P}(Z)$ und 2^Z bezeichnen die *Potenzmenge* einer Menge Z, d.h. die Menge aller Teilmengen von Z: $\mathcal{P}(Z) = \{X \mid X \subseteq Z\}$. Es gilt insbesondere $\emptyset \in \mathcal{P}(Z)$ und $Z \in \mathcal{P}(Z)$.

$X \times Y$ bezeichnet das (*kartesische*) *Produkt* zweier Mengen X und Y, bestehend aus allen Paaren, deren erste Komponente aus X und deren zweite Komponente aus Y stammt: $X \times Y = \{(x, y) \mid x \in X \wedge y \in Y\}$. Allgemeiner bezeichnet $X_1 \times ... \times X_n$ die Menge aller n-Tupel, deren i-te Komponente aus X_i für $i \in \{1, ..., n\}$ stammt: $X_1 \times \cdots \times X_n = \{(x_1, ..., x_n) \mid x_1 \in X_1 \wedge ... \wedge x_n \in X_n\}$.

- Relationen sind spezielle Mengen. Eine (*2-stellige* oder *binäre*) *Relation* R zwischen zwei Mengen X und Y ist eine Teilmenge des Produkts $X \times Y$, also $R \subseteq X \times Y$. Für Elementbeziehung $(x, y) \in R$ wird auch gerne die Infix-Notation xRy benutzt. Der *Definitionsbereich* (engl. *domain*) von R wird durch

$$dom(R) = \{x \in X \mid \exists\, y \in Y : (x, y) \in R\}$$

und der *Bildbereich* (engl. *range*) von R durch

$$ran(R) = \{y \in Y \mid \exists\, x \in X : (x, y) \in R\}$$

festgelegt. Eine Relation $R \subseteq X \times Y$ heißt

$$\begin{array}{rll} \textit{linksseindeutig}, & \text{falls} & \forall\, x_1, x_2 \in X, y \in Y : (x_1, y) \in R \wedge (x_2, y) \in R \Rightarrow x_1 = x_2, \\ \textit{rechtseindeutig}, & \text{falls} & \forall\, x \in X, y_1, y_2 \in Y : (x, y_1) \in R \wedge (x, y_2) \in R \Rightarrow y_1 = y_2, \\ \textit{linkstotal}, & \text{falls} & X = dom(R), \\ \textit{rechtstotal}, & \text{falls} & ran(R) = Y. \end{array}$$

Mit id_X wird die *Identitätsrelation* auf X bezeichnet: $id_X = \{(x, x) \mid x \in X\}$. Die *inverse Relation* oder *Umkehrrelation* von $R \subseteq X \times Y$ ist $R^{-1} \subseteq Y \times X$, die wie folgt definiert ist:

$$\forall x \in X, y \in Y : (x, y) \in R \Leftrightarrow (y, x) \in R^{-1}$$

Die *Komposition* oder *Hintereinanderausführung* $\circ$ zweier Relationen $R \subseteq X \times Y$ und $S \subseteq Y \times Z$ sei wie folgt definiert: Für alle $x \in X$ und $z \in Z$ gilt

$$(x, z) \in R \circ S \quad \Leftrightarrow \quad \exists\, y \in Y : (x, y) \in R \text{ und } (y, z) \in S.$$

Unter einer 2-stelligen Relation *auf einer Menge* X verstehen wir eine Relation $R \subseteq X \times X$. Eine solche Relation heißt

reflexiv,	falls	$\forall\, x \in X : (x,x) \in R$,
		oder in Relationenschreibweise: $id_X \subseteq R$,
irreflexiv,	falls	$\neg \exists\, x \in X : (x,x) \in R$,
		oder in Relationenschreibweise: $R \cap id_X = \emptyset$,
symmetrisch,	falls	$\forall\, x,y \in X : (x,y) \in R \Rightarrow (y,x) \in R$,
		oder in Relationenschreibweise: $R = R^{-1}$,
antisymmetrisch,	falls	$\forall\, x,y \in X : (x,y) \in R \wedge (y,x) \in R \Rightarrow x = y$,
		oder in Relationenschreibweise: $R \cap R^{-1} \subseteq id_X$,
transitiv,	falls	$\forall\, x,y,z \in X : (x,y) \in R \wedge (y,z) \in R \Rightarrow (x,z) \in R$,
		oder in Relationenschreibweise: $R \circ R \subseteq R$.

R heißt *Äquivalenzrelation*, falls R reflexiv, symmetrisch und transitiv ist. (Merke die Anfangsbuchstaben *r-s-t* dieser Eigenschaften.) Eine Äquivalenzrelation R auf X teilt die Menge X in disjunkte Teilmengen auf, so dass in jeder dieser Teilmengen sämtliche jeweils zueinander äquivalente Elemente aus X liegen. Für ein Element $x \in X$ bezeichne $[x]_R$ die *Äquivalenzklasse* von x, d.h. die Menge

$$[x]_R = \{y \in X \mid (x,y) \in R\}.$$

Ein Element einer Äquivalenzklasse heißt *Repräsentant* dieser Klasse, weil aus ihm und R die gesamte Klasse bestimmt werden kann. Dabei kann ein beliebiges Element der Klasse zum Repräsentanten gewählt werden, denn es gilt

$$\forall x,y \in X : (x,y) \in R \Leftrightarrow [x]_R = [y]_R.$$

Unter dem *Index von* R wird die Kardinalität der Menge aller Äquivalenzklassen von R auf X verstanden. Notation:

$$Index(R) = |\{\ [x]_R \mid x \in X\}|$$

Ist R aus dem Zusammenhang klar, so wird einfach $[x]$ statt $[x]_R$ geschrieben.

Eine Äquivalenzrelation R auf X heißt *Verfeinerung* einer Äquivalenzrelation S auf X, falls $R \subseteq S$ gilt. Dann ist nämlich jede Äquivalenzklasse von R Teilmenge einer Äquivalenzklasse von S.

Die *n-te Potenz* von $R \subseteq X \times X$ ist induktiv definiert:

$$R^0 = id_X \qquad \text{und} \qquad R^{n+1} = R \circ R^n$$

Die *transitive Hülle* R^+ und die *reflexive, transitive Hülle* R^* von R sind wie folgt definiert:

$$R^+ = \bigcup_{n \in \mathbb{N}\setminus\{0\}} R^n \qquad \text{und} \qquad R^* = \bigcup_{n \in \mathbb{N}} R^n$$

- Funktionen sind spezielle Relationen. Eine Relation $f \subseteq X \times Y$ heißt *partielle Funktion* (oder *partielle Abbildung*) von X nach Y, falls f rechtseindeutig ist. Dieses wird durch die Schreibweise $f : X \stackrel{part}{\longrightarrow} Y$ gekennzeichnet. Statt $(x, y) \in f$ wird dann $f(x) = y$ geschrieben. f heißt (*totale*) *Funktion* von X nach Y, falls f zusätzlich linkstotal ist. Dieses wird durch die Schreibweise $f : X \longrightarrow Y$ gekennzeichnet.

  Eine Funktion $f : X \longrightarrow Y$ heißt

injektiv,	falls	f linkseindeutig ist,
surjektiv,	falls	f rechtstotal ist,
bijektiv,	falls	f injektiv und surjektiv ist.

  Eine bijektive Funktion wird auch *Bijektion* genannt.

§3 Alphabete, Wörter und Sprachen

In dieser Vorlesung werden insbesondere formale Sprachen untersucht. Dazu verwenden wir folgende Notationen.

- *Alphabet* = endl. Menge von Zeichen (Symbolen) = Zeichenvorrat
  Wir benutzen A, B, Σ, Γ als typische Namen für Alphabete und a, b als typische Namen für Zeichen, also Elemente von Alphabeten.

- *Wort* über einem Alphabet Σ = endliche Kette von Zeichen aus Σ. Insbesondere gibt es das *leere Wort* ε. Wir benutzen u, v, w als typische Namen für Wörter.

  Beispiel : Sei $\Sigma = \{1, 2, +\}$. Dann sind $1 + 2$ und $2 + 1$ Wörter über Σ.

Σ^*	=	Menge aller Wörter über Σ. Es gilt $\Sigma \subseteq \Sigma^*$.
Σ^+	=	Menge aller nicht leeren Wörter über Σ, d.h. $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$.
$\lvert u \rvert$	=	Länge des Wortes u = Anzahl der in u vorkommenden Zeichen. Insbesondere: $\lvert\varepsilon\rvert = 0$.

  Die Zeichen eines Wortes u der Länge n bezeichnen wir auch mit $u_1, \ldots, u_n$.

  Die *Verkettung* (*Konkatenation*) zweier Wörter u und v geschieht durch Hintereinanderschreiben und wird mit $u \cdot v$ oder einfach uv bezeichnet.

  Beispiel : Die Verkettung von $1+$ und $2 + 0$ ergibt $1 + 2 + 0$.

  Ein Wort v heißt *Teilwort* eines Wortes w, falls $\exists u_1, u_2 : w = u_1 v u_2$ gilt.
  Ein Wort v heißt *Anfangsteilwort* (*Präfix*) eines Wortes w, falls $\exists u : w = vu$ gilt.
  Ein Wort v heißt *Endteilwort* (*Suffix*) eines Wortes w, falls $\exists u : w = uv$ gilt.
  Es kann mehrere *Vorkommen* desselben Teilwortes in einem Wort w geben.

  Beispiel : Im Wort $w = 1 + 1 + 1$ gibt es zwei Vorkommen des Teilwortes $1+$ und drei Vorkommen des Teilwortes 1.

- Eine (*formale*) *Sprache* über einem Alphabet Σ ist eine Teilmenge von Σ^*. Wir benutzen L als typischen Namen für Sprachen.

Für Sprachen gibt es die üblichen mengentheoretischen Verknüpfungen :

$L_1 \cup L_2$	(Vereinigung)
$L_1 \cap L_2$	(Durchschnitt)
$L_1 - L_2$ oder $L_1 \backslash L_2$	(Differenz)
$\overline{L} =_{df} \Sigma^* - L$	(Komplement)

Ferner gibt es spezielle Operatoren auf Sprachen.
Die *Verkettung* (*Konkatenation*) von Wörtern wird auf *Sprachen* L_1 und L_2 ausgedehnt:

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

Die *n-te Potenz* einer Sprache L ist induktiv definiert:

$$L^0 = \{\varepsilon\} \text{ und } L^{n+1} = L \cdot L^n$$

Der (KLEENE*sche*) *Sternoperator* (auch KLEENE-*Abschluss* oder *Iteration*) einer Sprache L ist

$$L^* = \bigcup_{n \in \mathbb{N}} L^n = \{w_1 \dots w_n \mid n \in \mathbb{N} \text{ und } w_1, \dots, w_n \in L\}.$$

Es gilt stets $\varepsilon \in L^*$. Neben L^* ist auch L^+ definiert:

$$L^+ = \bigcup_{n \in \mathbb{N} \setminus \{0\}} L^n = \{w_1 \dots w_n \mid n \in \mathbb{N} \text{ und } w_1, \dots, w_n \in L\}.$$

Es ist $\varepsilon \in L^+$ genau dann, wenn $\varepsilon \in L$ gilt. Ferner gilt $L^* = L^+ \cup \{\varepsilon\}$.

Beispiel : Für $L_1 = \{1+, 2+\}$ und $L_2 = \{1+0, 1+1\}$ ist

$$\begin{aligned} L_1 \cdot L_2 &= \{1+1+0, 1+1+1, 2+1+0, 2+1+1\}, \\ L_1^2 &= \{1+1+, 1+2+, 2+1+, 2+2+\}, \\ L_1^* &= \{\varepsilon, 1+, 2+, 1+1+, 1+2+, 2+1+, 2+2+, 1+1+1+, 1+1+2+, ...\} \\ L_1^+ &= \{1+, 2+, 1+1+, 1+2+, 2+1+, 2+2+, 1+1+1+, 1+1+2+, ...\}. \end{aligned}$$

§4 Literaturverzeichnis

Folgende Bücher sind als Ergänzung zu dieser Vorlesung empfehlenswert:

- U. Schöning: Theoretische Informatik – kurzgefasst. Spektrum Akademischer Verlag, 5. Auflage, 2008.

- J.E. Hopcroft, R. Motwani & J.D. Ullmann: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 2nd Edition, 2001.

  deutsche Übersetzung:
  J.E. Hopcroft, R. Motwani & J.D. Ullmann: Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie. Pearson Studium, 2. Auflage, 2001.

- D. Harel: Algorithmics – The Spirit of Computing. Addison-Wesley, 1987.

  deutsche Übersetzung:
  D. Harel und Y.A. Feldman: Algorithmik: die Kunst des Rechnens. Springer, 2006

Zur Ausarbeitung der Vorlesung haben wir ferner folgende Quellen benutzt:

J. Albert & T. Ottmann: Automaten, Sprachen und Maschinen für Anwender. BI 1983 (nur einführendes Buch).

E. Börger: Berechenbarkeit, Komplexität, Logik. Vieweg, Braunschweig 1986 (2. Auflage).

W. Brauer: Automatentheorie. Teubner Verlag, Stuttgart 1984.

E. Engeler & P. Läuchli: Berechnungstheorie für Informatiker. Teubner, Stuttgart 1988.

H. Hermes: Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit. 2. Auflage, Springer-Verlag, Berlin 1971.

J.E. Hopcroft & J.D. Ullmann: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979.

A.R. Lewis & C.H. Papadimitriou: Elements of the Theory of Computation. Prentice Hall, Englewood Cliffs 1981.

K.R. Reischuk: Einführung in die Komplexitätstheorie. Teubner Verlag, Stuttgart 1990.

A. Salomaa: Computation and Automata. Cambridge University Press, Cambridge 1985.

A. Salomaa: Formal Languages. Academic Press, New York 1973.

F. Setter: Grundbegriffe der Theoretischen Informatik. Springer-Verlag, Heidelberg 1988 (einführendes Buch).

W. Thomas: Grundzüge der Theoretischen Infomatik. Vorlesung im Wintersemester 1989/90 an der RWTH Aachen.

Kapitel II

Endliche Automaten und reguläre Sprachen

In diesem Kapitel untersuchen wir ein sehr einfaches Modell von Maschinen: den endlichen Automaten. Wir werden sehen, dass

- endliche Automaten in der Informatik vielseitig eingesetzt werden können,
- die von endlichen Automaten akzeptierten Sprachen viele Struktureigenschaften besitzen, z.B. Darstellbarkeit durch reguläre Ausdrücke,
- Fragen über die von endlichen Automaten durchgeführten Berechnungen entscheidbar sind.

Ferner können endliche Automaten leicht realisiert werden als Tabellen in Programmen und als Schaltungen (vgl. die Vorlesung „Technische Informatik I“).

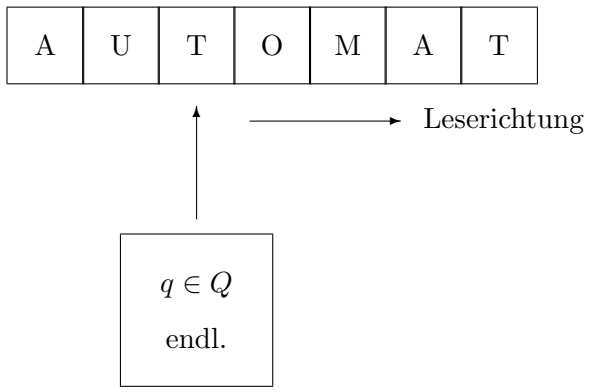
§1 Endliche Automaten

Wir wollen endliche Automaten zum Akzeptieren von Sprachen einsetzen. Wir können uns einen endlichen Automaten vorstellen als eine Maschine mit Endzuständen, die Zeichen auf einem Band liest and dabei den Lesekopf nur nach rechts bewegen darf und keine neuen Symbole auf das Band drucken kann.

Deshalb wird bei endlichen Automaten die Überführungsfunktion meistens als Abbildung

$\delta : Q \times \Sigma \to Q$ definiert, wobei Q die Zustandsmenge und Σ das Eingabealphabet des Automaten sind. Ein solcher Automat wird angesetzt auf ein Wort $w \in \Sigma^*$, dessen Akzeptanz überprüft werden soll.

Skizze:



Für die graphische Darstellung und die Definition des Akzeptanzverhaltens von Automaten ist aber die Darstellung von Automaten als sogenannte *Transitionssysteme* günstiger.

1.1 **Definition (deterministischer endl. Automat)**: Ein *deterministischer endlicher Automat (Akzeptor)*, kurz DEA, ist eine Struktur

$$\mathcal{A} = (\Sigma, Q, \delta, q_0, F) \quad \text{bzw.} \quad \mathcal{A} = (\Sigma, Q, \rightarrow, q_0, F)$$

mit folgenden Eigenschaften:

1. Σ ist eine endliche Menge, das *Eingabealphabet*,
2. Q ist eine endliche Menge von *Zuständen*,
3. $\delta : Q \times \Sigma \rightarrow Q$ ist die *Überführungsfunktion*

   bzw. $\rightarrow \subseteq Q \times \Sigma \times Q$ ist eine *deterministische Transitionsrelation*, d.h. es gilt

   $$\forall\, q \in Q \quad \forall\, a \in \Sigma \quad \exists \text{ genau ein } q' \in Q : (q, a, q') \in \rightarrow,$$

4. $q_0 \in Q$ ist der *Anfangszustand*,
5. $F \subseteq Q$ ist die Menge der *Endzustände*.

Die beiden Darstellungen von Automaten sind verknüpft durch :

$$\delta(q, a) = q' \Leftrightarrow (q, a, q') \in \rightarrow$$

Die Elemente $(q, a, q') \in \rightarrow$ heißen *Transitionen* . Wir schreiben meist $q \xrightarrow{a} q'$ statt $(q, a, q') \in \rightarrow$. Für gegebenes $a \in \Sigma$ fassen wir $\xrightarrow{a}$ auch als 2-stellige Relation $\xrightarrow{a} \subseteq Q \times Q$ auf:

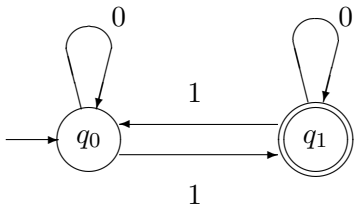
$$\forall\, q, q' \in Q : q \xrightarrow{a} q' \Leftrightarrow (q, a, q') \in \rightarrow$$

Ein DEA kann graphisch durch ein *Zustandsdiagramm* dargestellt werden. Das ist ein gerichteter Graph, der für jeden Zustand q des Automaten einen mit q beschrifteten Knoten enthält und für jede Transition $q \xrightarrow{a} q'$ eine gerichtete, beschriftete Kante. Der Anfangszustand q_0 ist durch einen hineingehenden Pfeil markiert. Endzustände q sind durch einen zusätzlichen Kreis gekennzeichnet.

Beispiel : Betrachte $\mathcal{A}_1 = (\{0,1\}, \{q_0, q_1\}, \to, q_0, \{q_1\})$ mit

$$\to = \{(q_0, 0, q_0), (q_0, 1, q_1), (q_1, 0, q_1), (q_1, 1, q_0)\}.$$

Dann wird $\mathcal{A}_1$ durch folgendes Zustandsdiagramm dargestellt:



1.2 **Definition (Akzeptanz und Erreichbarkeit)**:
Sei $\mathcal{A} = (\Sigma, Q, \to, q_0, F)$ bzw. $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ ein DEA.

1. Wir erweitern die Transitionsrelationen $\xrightarrow{a}$ von einzelnen Eingabesymbolen $a \in \Sigma$ auf Wörter $w \in \Sigma^*$ solcher Symbole und definieren die zugehörige Relationen $\xrightarrow{w}$ induktiv :

   - $q \xrightarrow{\varepsilon} q'$ gdw. $q = q'$
     oder in Relationenschreibweise: $\xrightarrow{\varepsilon} = id_Q$
   - $q \xrightarrow{aw} q'$ gdw. $\exists\, q'' \in Q : q \xrightarrow{a} q''$ und $q'' \xrightarrow{w} q'$
     oder in Relationenschreibweise: $\xrightarrow{aw} = \xrightarrow{a} \circ \xrightarrow{w}$

   Analog hierzu definieren wir die erweiterte Überführungsfunktion δ^*

   $$\delta^* : Q \times \Sigma^* \to Q \text{ mit } \delta^*(q, \varepsilon) = q \text{ und } \delta^*(q, aw) = \delta^*(\delta(q, a), w)$$

   für alle $q, q' \in Q$, $a \in \Sigma$ und $w \in \Sigma^*$. Es gilt:

   $$\delta^*(q, w) = q' \quad \Leftrightarrow \quad q \xrightarrow{w} q'$$

2. Die von $\mathcal{A}$ *akzeptierte (oder erkannte) Sprache* ist
   $L(\mathcal{A}) = \{w \in \Sigma^* \mid \exists\, q \in F : q_0 \xrightarrow{w} q\}$ bzw. $L(\mathcal{A}) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$
   Eine Sprache L heißt *endlich akzeptierbar*, falls es einen DEA $\mathcal{A}$ mit $L = L(\mathcal{A})$ gibt.

3. Ein Zustand $q \in Q$ heißt in $\mathcal{A}$ *erreichbar*, falls $\exists w \in \Sigma^* : q_0 \xrightarrow{w} q$.

Beispiel : Für den Automaten des letzten Beispiels gilt:

$$L(\mathcal{A}_1) = \{w \in \{0,1\}^* \mid w \text{ enthält } \textit{ungerade} \text{ viele Symbole } 1\}.$$

Bemerkung : Für alle $a_1, \ldots, a_n \in \Sigma$ und $q, q' \in Q$ gilt :

$$q \xrightarrow{a_1 \cdots a_n} q' \quad \Leftrightarrow \quad \exists\, q_1, \ldots, q_n \in Q : q \xrightarrow{a_1} q_1 \quad \ldots \quad q_{n-1} \xrightarrow{a_n} q_n = q'.$$

oder in Relationenschreibweise:

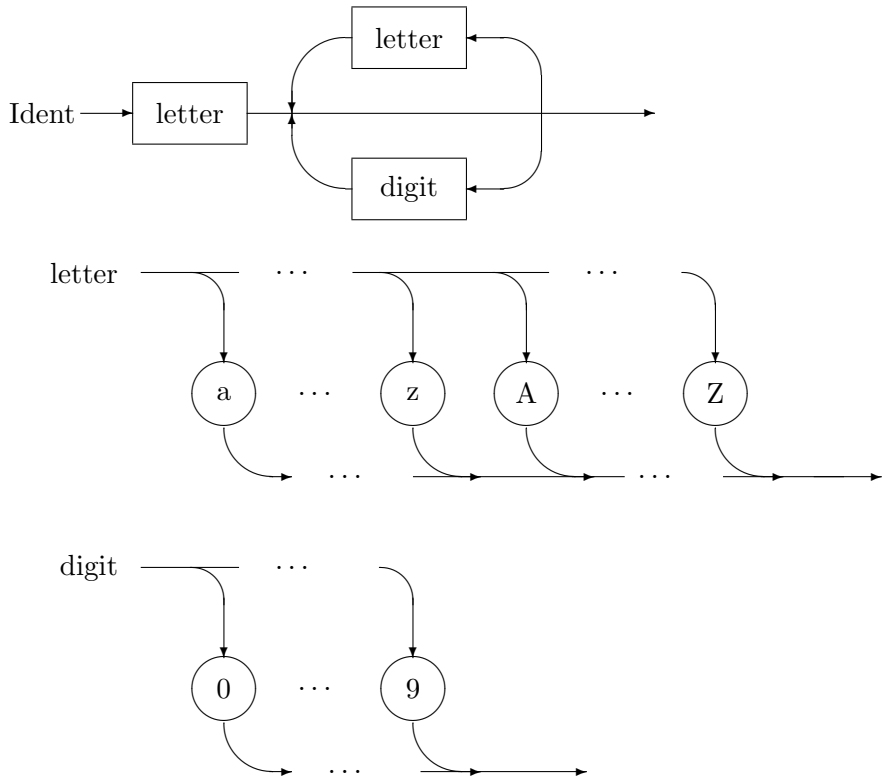
$$\overset{a_1 \cdot \ldots \cdot a_n}{\rightarrow} = \overset{a_1}{\rightarrow} \circ \cdots \circ \overset{a_n}{\rightarrow}$$

Die Kette $q \overset{a_1}{\rightarrow} q_1 \quad \ldots \quad q_{n-1} \overset{a_n}{\rightarrow} q_n$ heißt auch *Transitionenfolge* .

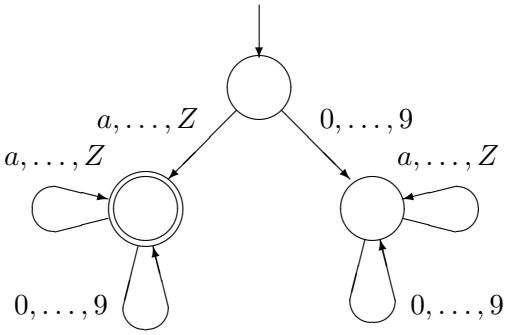
Beispiel (aus dem Bereich des Übersetzerbaus): Ein Übersetzer für eine Programmiersprache arbeitet in mehreren, zeitlich möglicherweise überlappenden Phasen:

- Lexikalische Analyse : In dieser Phase fasst der sogenannte *Scanner* den Eingabetext zu einer Folge von *Token* zusammen, das sind Bezeichner, Schlüsselworte und Begrenzer.
- Syntaxanalyse : Die erzeugte Tokenfolge ist Eingabe des sogenannten *Parser*, der entscheidet, ob diese Folge ein syntaktisch korrektes Programm darstellt.
- Codeerzeugung : Die vom Parser erkannte Syntaxstruktur wird zur Erzeugung des Maschinencodes benutzt.
- Optimierung : Durch meist lokales Verändern des entstandenen Maschinencodes soll die Laufzeit des übersetzten Programms verbessert werden.

Die lexikalische Analyse ist eine einfache Aufgabe, die von endlichen Automaten bewältigt werden kann. Als Beispiel betrachten wir den üblichen Aufbau von Bezeichnern (Identifikatoren) in Programmiersprachen. Als Syntaxdiagramm finden sich etwa für MODULA



Die so aufgebauten Identifikatoren können durch folgenden endl. Automaten erkannt werden :



Der Übersichtlichkeit halber haben wir die Kanten im Zustandsdiagramm mit mehreren Symbolen beschriftet.

Beispiel (aus dem Bereich der Betriebssysteme): Wird im Multiprogramming-Betrieb von mehreren Programmen auf eine gemeinsame Ressource zugegriffen, so müssen diese Zugriffe synchronisiert werden, damit keine fehlerhaften Resultate entstehen. Als Beispiel betrachten wir zwei Programme P_1 und P_2, die auf einen gemeinsamen Drucker zugreifen. P_1 und P_2 sind so zu synchronisieren, dass sie nicht gleichzeitig Daten zum Drucker schicken. Wir konstruieren einen endlichen Automaten, der die Druckerbenutzung von P_1 und P_2 überwacht.

- P_1 meldet dem Automaten Beginn und Ende einer Druckerbenutzung durch die Symbole b_1 und e_1

- P_2 verhält sich entsprechend durch b_2 und e_2.

Zu jedem Zeitpunkt ist das Verhalten von P_1 und P_2 bezüglich des Druckers durch ein endliches Wort

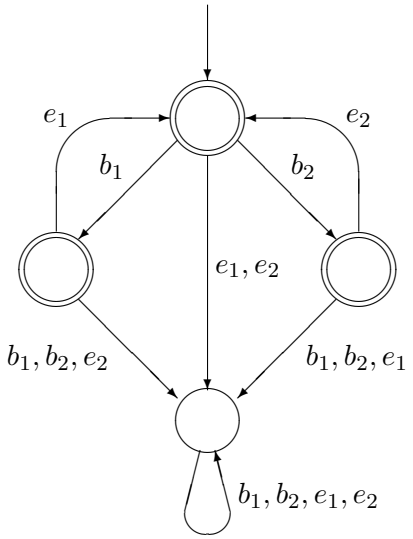
$$w \in \{b_1, e_1, b_2, e_2\}^*$$

gegeben. Der Automat soll ein solches Wort w akzeptieren, falls

- jedes P_i einzeln betrachtet den Drucker korrekt benutzt, d.h. die Symbole b_i und e_i, alternierend in w vorkommen, beginnend mit b_i, $i = 1, 2$.

- P_1 und P_2 den Drucker nicht gleichzeitig benutzen, d.h. es gibt in w weder ein Teilwort b_1b_2 noch b_2b_1.

Z.B. soll $w_1 = b_1e_1b_2e_2$ akzeptiert werden, nicht aber $w_2 = b_1b_2e_1e_2$. Folgender endlicher Automat leistet das Gewünschte :

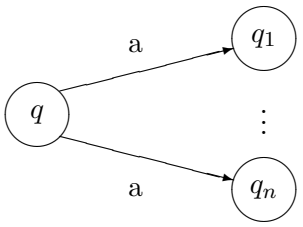


Nichtdeterminismus

In vielen Anwendungen können endliche Automaten vereinfacht werden, wenn *Nichtdeterminismus* zugelassen wird, d.h. wir betrachten dann eine *beliebige* Relation

$$\rightarrow \subseteq Q \times \Sigma \times Q$$

als Transitionsrelation. Es darf dann vorkommen, dass es für gewisse $q \in Q, a \in \Sigma$ *mehrere* Nachfolgezustände $q_1, \ldots, q_n$ gibt, so dass (in graphischer Darstellung)



gilt. Der Automat geht dann beim Akzeptieren von a nichtdeterministisch in einen der Folgezustände $q_1, \ldots, q_n$ von q über. Ein Spezialfall ist $n = 0$; dann gibt es zu $q \in Q$ und $a \in \Sigma$ keinen Nachfolgezustand q' mit $q \xrightarrow{a} q'$. Anschaulich *stoppt* der Automat dann und weigert sich, das Symbol a zu akzeptieren. Diese Bemerkungen führen zu folgender Definition.

1.3 **Definition** : Ein *nichtdeterministischer endl. Automat* (oder *Akzeptor*), kurz NEA, ist eine Struktur

$$\mathcal{B} = (\Sigma, Q, \rightarrow, q_0, F),$$

wobei Σ, Q, q_0 und F wie bei den DEAs definiert sind und für $\rightarrow$ gilt:

$$\rightarrow \subseteq Q \times \Sigma \times Q.$$

Transitionen werden wie bei DEAs geschrieben ($q \xrightarrow{a} q'$) und auf Wörter erweitert ($q \xrightarrow{w} q'$).

Die graphische Darstellung durch Zustandsdiagramme bleibt unverändert.

1.4 **Definition (Akzeptanz und Äquivalenz)**:

(i) Die von einem NEA $\mathcal{B} = (\Sigma, Q, \to, q_0, F)$ *akzeptierte* (oder *erkannte*) *Sprache* ist

$$L(\mathcal{B}) = \{w \in \Sigma^* \mid \exists\, q \in F : q_0 \xrightarrow{w} q\}.$$

Mit *NEA* bezeichnen wir die Klasse der von NEAs akzeptierten Sprachen.

(ii) Zwei NEAs $\mathcal{B}_1$ und $\mathcal{B}_2$ heißen *äquivalent*, falls $L(\mathcal{B}_1) = L(\mathcal{B}_2)$ gilt.

Anschaulich akzeptiert ein NEA ein Wort w, falls er beim Lesen von w eine Transitionsfolge durchlaufen *kann*, die in einem Endzustend endet. Andere Transitionsfolgen dürfen durchaus in Nicht-Endzuständen enden; es genügt die Existenz *einer* akzeptierten Transitionsfolge.

DEAs sind offensichtlich ein Spezialfall von NEAs. Somit ist auch die Äquivalenz zwischen beliebigen endlichen Automaten definiert.

Beispiel (Suffix-Erkennung): Gegeben sei ein Alphabet Σ und ein Wort $v = a_1 \dots a_n \in \Sigma^*$ mit $a_i \in \Sigma$ für $i = 1, \dots, n$.

Wir wollen die Sprache

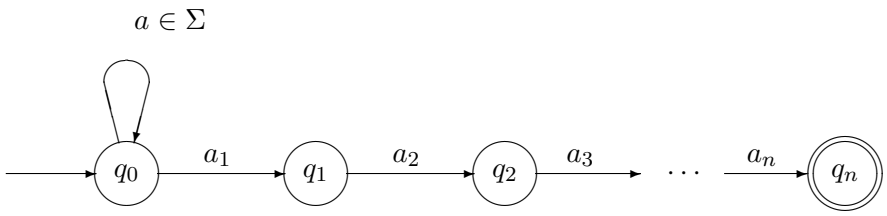
$$L_v = \{wv \mid w \in \Sigma^*\}$$

aller Wörter über Σ mit dem Suffix v erkennen.

Dazu betrachten wir den NEA

$$\mathcal{B}_v = (\Sigma, \{q_0, \dots, q_n\}, \to, q_0, \{q_n\}),$$

wobei $\to$ durch folgendes Zustandsdiagramm $\mathcal{B}_v$ erklärt ist:



$\mathcal{B}_v$ ist im Anfangszustand q_0 nichtdeterministisch: Beim Lesen eines Wortes kann sich $\mathcal{B}_v$ bei jedem Vorkommen von a_1 entscheiden, ab jetzt das Endwort v zu akzeptieren zu versuchen. Dazu geht $\mathcal{B}_v$ nach q_1 über und erwartet jetzt $a_2 \dots a_n$ als Restwort. Sollte dieses nicht der Fall sein, so stoppt $\mathcal{B}_v$ für irgendein $i \in \{1, \dots, n\}$ und $a \neq a_i$.

Es stellt sich die Frage: Akzeptieren NEAs mehr Sprachen als DEAs ?

Die Antwort lautet „nein “.

1.5 **Satz (**Rabin und Scott**, 1959)**: Zu jedem NEA gibt es einen äquivalenten DEA.

Beweis : Gegeben sei ein NEA $\mathcal{B} = (\Sigma, Q, \to, q_0, F)$. Wir führen einen DEA $\mathcal{A}$ mit $L(\mathcal{A}) = L(\mathcal{B})$ durch folgende *Potenzmengen-Konstruktion* ein:

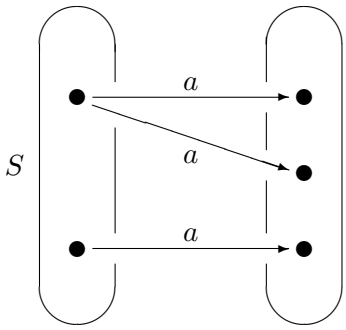
$$\mathcal{A} \;=\; (\Sigma, \mathcal{P}(Q), \to_{\mathcal{A}}, \{q_0\}, F_{\mathcal{A}}),$$

wobei für $S, S' \subseteq Q$ und $a \in \Sigma$ gilt:

$$S \to^a_{\mathcal{A}} S' \text{ gdw } S' \;=\; \{q' \in Q \mid \exists\, q \in S : q \stackrel{a}{\to} q'\},$$

Die Endzustandsmenge sei $F_{\mathcal{A}} \;=\; \{S \subseteq Q \mid S \cap F \neq \emptyset\}$.

$\mathcal{A}$ hat also für jede Teilmenge S der Zustandsmenge Q von $\mathcal{B}$ einen eigenen Zustand, den wir mit S selbst bezeichnen. Es gilt $S \to^a_{\mathcal{A}} S'$ gdw S' die Menge aller Folgezustände ist, die durch a-Transitionen des nichtdet. Automaten $\mathcal{B}$ von Zuständen aus S erreicht werden können. Veranschaulichung:



liefert $S \stackrel{a}{\to}_{\mathcal{A}} S'$

Man sieht, dass $\to_{\mathcal{A}}$ deterministisch ist, also

$$\forall\, S \subseteq Q \quad \forall\, a \in \Sigma \quad \exists \text{ genau ein } S' \subseteq Q : S \stackrel{a}{\to}_{\mathcal{A}} S'.$$

Ferner gilt für alle $q, q' \in Q$, $S, S' \subseteq Q, a \in \Sigma$:

(i) Wenn $q \stackrel{a}{\to} q'$ und $q \in S$, dann $\exists\, S' \subseteq Q :\quad S \to^a_{\mathcal{A}} S'$ und $q' \in S'$.

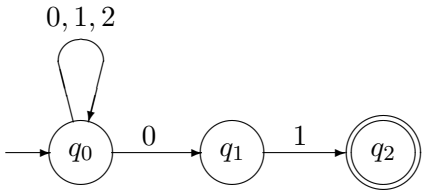
(ii) Wenn $S \to^a_{\mathcal{A}} S'$ und $q' \in S'$, dann $\exists\, q \in Q :\quad q \stackrel{a}{\to} q'$ und $q \in S$.

Damit können wir leicht zeigen, dass $L(\mathcal{A}) \;=\; L(\mathcal{B})$ gilt. Sei $w \;=\; a_1 \ldots a_n \in \Sigma^*$ mit $a_i \in \Sigma$ für $i \;=\; 1, \ldots, n$. Dann gilt:

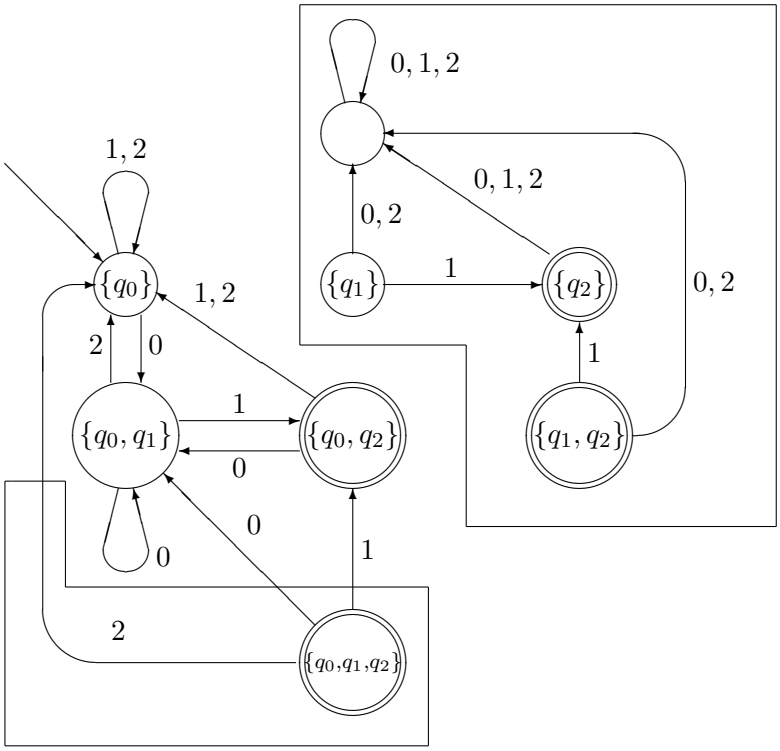
$$\begin{aligned}
w \in L(\mathcal{B}) \Leftrightarrow\;& \exists\, q \in F :\quad q_0 \stackrel{w}{\to} q \\
\Leftrightarrow\;& \exists\, q_1, \ldots, q_n \in Q : \\
& q_0 \stackrel{a_1}{\to} q_1 \ldots q_{n-1} \stackrel{a_n}{\to} q_n \text{ mit } q_n \in F \\
\Leftrightarrow\;& \{\text{„}\Rightarrow\text{“ wegen (i) und „}\Leftarrow\text{“ wegen (ii)}\} \\
& \exists\, S_1, \ldots, S_n \subseteq Q : \\
& \{q_0\} \to^{a_1}_{\mathcal{A}} S_1 \quad \ldots \quad S_{n-1} \to^{a_n}_{\mathcal{A}} S_n \text{ mit } S_n \cap F \neq \emptyset \\
\Leftrightarrow\;& \exists\, S \in F_A : \{q_0\} \to^w_{\mathcal{A}} S \\
\Leftrightarrow\;& w \in L(\mathcal{A}).
\end{aligned}$$

□

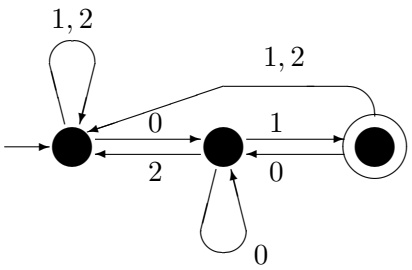
Beispiel : Wir betrachten die Suffix-Erkennung von 01 in Wörtern über dem Alphabet $\Sigma = \{0, 1, 2\}$. Nach dem vorangegangenen Beispiel wird die Sprache $L_{01} = \{w01 \mid w \in \Sigma^*\}$ von dem NEA $\mathcal{B}_{01}$ mit



erkannt. Wir wenden jetzt auf $\mathcal{B}_{01}$ die Potenzmengen-Konstruktion aus dem Satz von Rabin und Scott an. Das Ergebnis ist der folgende DEA $\mathcal{A}_{01}$:



Die umkasteten Teile □ sind vom Anfangszustand $\{q_0\}$ von $\mathcal{A}_{01}$ unerreichbar. $\mathcal{A}_{01}$ kann deshalb zu folgendem äquivalenten DEA vereinfacht werden:



Dieser DEA lässt sich in seiner Zustandszahl nicht weiter minimieren. Es gibt Beispiele, wo der

im Beweis des Satzes von Rabin und Scott angegebene DEA nicht weiter vereinfacht werden kann, d.h. wenn der gegebene NEA n Zustände besitzt, so benötigt der DEA im schlechtesten Fall tatsächlich 2^n Zustände.

Spontane Übergänge

Noch bequemer lassen sich Automaten angeben, wenn neben Nichtdeterminismus auch ε-Übergänge zugelassen werden, das sind Übergänge, die ein Automat selbständig ("spontan") macht, ohne dass ein Buchstabe des Eingabewortes gelesen wird.

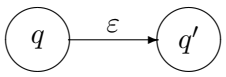
1.6 **Definition** : Ein *nichtdet. endl. Automat (Akzeptor) mit ε-Übergängen*, kurz ε-NEA, ist eine Struktur

$$\mathcal{B} \;=\; (\Sigma, Q, \to, q_0, F),$$

wobei Σ, Q, q_0 und F wie bei NEAs bzw. DEAs definiert sind und wobei für die Transitionsrelation $\to$ gilt:

$$\to \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q.$$

Transitionen $q \xrightarrow{\varepsilon} q'$ heißen *ε-Übergänge* oder *ε-Transitionen* und werden in den Zustandsdiagrammen entsprechend dargestellt:



Um das Akzeptanz-Verhalten von ε-NEAs zu definieren, benötigen wir eine erweiterte Transitionsrelation $q \overset{w}{\Rightarrow} q'$. Dazu machen wir zwei terminologische Vorbemerkungen.

- Für jedes $\alpha \in \Sigma \cup \{\varepsilon\}$ fassen wir $\xrightarrow{\alpha}$ als 2-stellige Relation über Q auf, also $\xrightarrow{\alpha} \subseteq Q \times Q$,

  $$\forall\; q, q' \in Q \text{ gilt: } q \xrightarrow{\alpha} q' \Leftrightarrow (q, \alpha, q') \in \to \;.$$

  Wir benutzen hier die übliche Infix-Notation für 2-stellige Relationen, also $q \xrightarrow{\alpha} q'$ statt $(q, q') \in \xrightarrow{\alpha}$. Wir nennen $\xrightarrow{\alpha}$ die *α-Transitionsrelation.*

- Damit können wir die Komposition $\circ$ für solche Transitionsrelationen benutzen. Für alle $\alpha, \beta \in \Sigma \cup \{\varepsilon\}$ ist $\xrightarrow{\alpha} \circ \xrightarrow{\beta}$ folgendermaßen definiert : $q, q' \in Q$ gilt

  $$q \xrightarrow{\alpha} \circ \xrightarrow{\beta} q' \Leftrightarrow \exists\; q'' \in Q : q \xrightarrow{\alpha} q'' \text{ und } q'' \xrightarrow{\beta} q'.$$

1.7 **Definition** : Sei $\mathcal{B} = (\Sigma, Q, \to, q_0, F)$ ein ε-NEA. Dann wird induktiv für jedes Wort $w \in \Sigma^*$ eine 2-stellige Relation $\overset{w}{\Rightarrow} \subseteq Q \times Q$ definiert:

- $q \overset{\varepsilon}{\Rightarrow} q'$, falls $\exists\; n \geq 0 :\quad q \underbrace{\xrightarrow{\varepsilon} \circ \cdots \circ \xrightarrow{\varepsilon}}_{n\text{-mal}} q'$

- $q \overset{aw}{\Rightarrow} q'$, falls $q \overset{\varepsilon}{\Rightarrow} \circ \overset{a}{\to} \circ \overset{w}{\Rightarrow} q'$

Dabei gelte $q, q' \in Q$, $a \in \Sigma$ und $w \in \Sigma^*$.

Bemerkung : Für alle $q, q' \in Q$ und $a_1, \ldots, a_n \in \Sigma$ gilt:

(i) $q \overset{\varepsilon}{\Rightarrow} q$, aber aus $q \overset{\varepsilon}{\Rightarrow} q'$ folgt nicht $q = q'$.

(ii)
$$\begin{aligned} q \overset{a_1 \ldots a_n}{\Longrightarrow} q' \quad &\Leftrightarrow \quad q \overset{\varepsilon}{\Rightarrow} \circ \overset{a_1}{\to} \circ \overset{\varepsilon}{\Rightarrow} \cdots \circ \overset{\varepsilon}{\Rightarrow} \circ \overset{a_n}{\to} \circ \overset{\varepsilon}{\Rightarrow} q' \\ &\Leftrightarrow \quad q \overset{a_1}{\Rightarrow} \circ \cdots \circ \overset{a_n}{\Rightarrow} q' \end{aligned}$$

(iii) Es ist entscheidbar, ob für gegebene Zustände $q, q' \in Q$ die Relation $q \overset{\varepsilon}{\Rightarrow} q'$ gilt.

Beweis : (i) und (ii) folgen unmittelbar aus der Definition.

„(iii)“: Sei k die Anzahl der Zustände in Q. Dann gilt:

$$q \overset{\varepsilon}{\Rightarrow} q' \Leftrightarrow \exists\, n \leq k-1 : \quad q \underbrace{\overset{\varepsilon}{\to} \circ \cdots \circ \overset{\varepsilon}{\to}}_{n\text{-mal}} q'.$$

Wenn nämlich mindestens k Transitionen $\overset{\varepsilon}{\to}$ hintereinander ausgeführt werden, dann wiederholen sich Zustände im Pfad von q nach q'. Um also alle von q durch ε-Transitionen erreichbaren Zustände zu bestimmen, genügt es, die endlich vielen Folgen mit maximal $k-1$ Transitionen $\overset{\varepsilon}{\to}$ zu überprüfen. Daraus ergibt sich die Entscheidbarkeit von $q \overset{\varepsilon}{\Rightarrow} q'$. □

1.8 **Definition (Akzeptanz und Äquivalenz)**: Sei $\mathcal{B} = (\Sigma, Q, \to, q_0, F)$ ein ε-NEA.

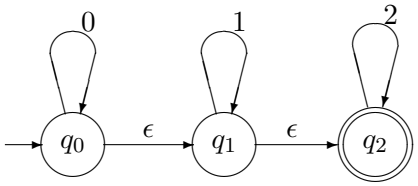
(i) Die von $\mathcal{B}$ *akzeptierte* (oder *erkannte*) *Sprache* ist

$$L(\mathcal{B}) = \{w \in \Sigma^* \mid \exists\, q \in F : q_0 \overset{w}{\Rightarrow} q\}.$$

(ii) Zwei ε-NEAs $\mathcal{B}_1$ und $\mathcal{B}_2$ heißen *äquivalent*, falls $L(\mathcal{B}_1) = L(\mathcal{B}_2)$ gilt.

Offensichtlich sind NEAs ein Spezialfall von ε-NEAs. Daher ist Äquivalenz auch zwischen NEAs und ε-NEAs definiert.

Beispiel : Zum Eingabealphabet $\Sigma = \{0, 1, 2\}$ betrachten wir den durch folgendes Zustandsdiagramm gegebenen ε-NEA $\mathcal{B}$:



Dann gilt:

$$L(\mathcal{B}) \;=\; \{w \in \{0,1,2\}^* \mid \exists\, k,l,m \geq 0 : w \;=\; \underbrace{0\ldots0}_{k\text{-mal}}\underbrace{1\ldots1}_{l\text{-mal}}\underbrace{2\ldots2}_{m\text{-mal}}\}$$

1.9 **Satz** : Zu jedem ε-NEA gibt es einen äquivalenten NEA.

Beweis : Gegeben sei ein ε-NEA $\mathcal{B} \;=\; (\Sigma, Q, \to, q_0, F)$.

Wir konstruieren folgenden NEA $\mathcal{A} \;=\; (\Sigma, Q, \to_{\mathcal{A}}, q_0, F_{\mathcal{A}})$, wobei für $q, q' \in Q$ und $a \in \Sigma$ gilt:

- $q \overset{a}{\to}_{\mathcal{A}} q'$ gdw $q \overset{a}{\Rightarrow} q'$ in $\mathcal{B}$, also $q \overset{\varepsilon}{\Rightarrow} \circ \overset{a}{\to} \circ \overset{\varepsilon}{\Rightarrow} q'$ in $\mathcal{B}$
- $q \in F_{\mathcal{A}}$ gdw $\exists\, q' \in F : \quad q \overset{\varepsilon}{\Rightarrow} q'$

Nach Definition ist $\mathcal{A}$ ein NEA ohne ε-Übergänge mit $F \subseteq F_{\mathcal{A}}$. Es bleibt zu zeigen: $L(\mathcal{A}) = L(\mathcal{B})$. Sei $w \;=\; a_1 \ldots a_n \in \Sigma^*$ mit $n \geq 0$ und $a_i \in \Sigma$ für $i \;=\; 1, \ldots, n$. Dann gilt:

$$\begin{aligned}
w \in L(\mathcal{A}) \Leftrightarrow\;& \exists\, q \in F_{\mathcal{A}} : \quad q_0 \overset{w}{\to}_{\mathcal{A}} q \\
\Leftrightarrow\;& \exists\, q \in F_{\mathcal{A}} : \quad q_0 \overset{a_1}{\to}_{\mathcal{A}} \circ \cdots \circ \overset{a_n}{\to}_{\mathcal{A}} q \\
\Leftrightarrow\;& \exists\, q \in F_{\mathcal{A}} : \quad q_0 \overset{a_1}{\Rightarrow} \circ \cdots \circ \overset{a_n}{\Rightarrow} q \\
\Leftrightarrow\;& \{\text{„}\Rightarrow\text{“: Wähle } q' \text{ mit } q \overset{\varepsilon}{\Rightarrow} q'. \\
& \quad \text{„}\Leftarrow\text{“: Wähle } q \;=\; q'.\} \\
& \quad \exists\, q' \in F : \quad q_0 \overset{a_1}{\Rightarrow} \circ \cdots \circ \overset{a_n}{\Rightarrow} q' \\
\Leftrightarrow\;& \exists\, q' \in F : \quad q_0 \overset{w}{\Rightarrow} q' \\
\Leftrightarrow\;& w \in L(\mathcal{B})
\end{aligned}$$

Im Spezialfall $w \;=\; \varepsilon$ verkürzt sich das obige Argument wie folgt:

$$\begin{aligned}
\varepsilon \in L(\mathcal{A}) \Leftrightarrow\;& \exists\, q_0 \in F_{\mathcal{A}} \\
\Leftrightarrow\;& \exists\, q \in F : \quad q_0 \overset{\varepsilon}{\Rightarrow} q \\
\Leftrightarrow\;& \varepsilon \in L(\mathcal{B})
\end{aligned}$$

□

Bemerkungen :

(i) Im Beweis hätte man $F_{\mathcal{A}}$ auch so definieren können:

$$F_{\mathcal{A}} \;=\; \begin{cases} F \cup \{q_0\} & falls \quad \exists\, q \in F : \quad q_0 \overset{\varepsilon}{\Rightarrow} q \\ F & sonst \end{cases}$$

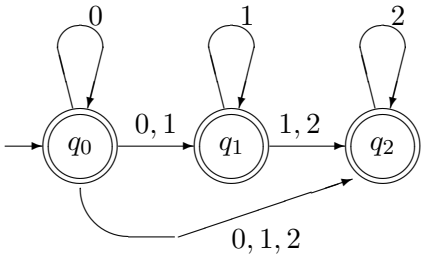
Vgl. dazu den Beweis in Hopcroft & Ullman.

(ii) Die obige Konstruktion des NEAs $\mathcal{A}$ aus dem gegebenen ε-NEA $\mathcal{B}$ ist algorithmisch durchführbar, weil die Relation $q \overset{\varepsilon}{\Rightarrow} q'$ entscheidbar ist:

Vgl. dazu die vorangegangene Bemerkung.

(iii) Falls es ε-Zyklen im ε-NEA $\mathcal{B}$ gibt, kann die Zustandsmenge des NEAs $\mathcal{A}$ durch eine vorgeschaltete Kontraktion der ε-Zyklen verkleinert werden; jeder ε-Zyklus kann auf einen Zustand zusammengezogen werden.

Beispiel : Wir wenden die im Beweis vorgestellte Konstruktion auf den ε-NEA $\mathcal{B}$ des vorangegangenen Beispiels an. Das Ergebnis ist der folgende NEA $\mathcal{A}$:



Es ist leicht nachzuprüfen, dass tatsächlich $L(\mathcal{A}) = L(\mathcal{B})$ gilt.

Wir haben also folgendes Endergebnis:

$$\text{DEA} = \text{NEA} = \varepsilon\text{-NEA},$$

d.h. die Klassen der von DEAs, NEAs und ε-NEAs akzeptierbaren Sprachen stimmen überein, es handelt sich um die *Klasse der endlich akzeptierten Sprachen*. Wenn wir Eigenschaften dieser Klasse zeigen wollen, können wir den Automatentyp wählen, der für den jeweiligen Beweis am bequemsten ist.

§2 Abschlusseigenschaften

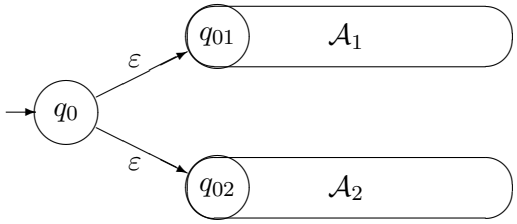
Wir untersuchen jetzt, unter welchen Operationen die Klasse der endlich akzeptierbaren Sprachen abgeschlossen ist. Dazu betrachten wir die Mengenoperationen Vereinigung, Durchschnitt, Komplement, Differenz sowie die in Kapitel I eingeführten Sprachoperationen Konkatenation und Iteration (KLEENEscher Sternoperator).

2.1 **Satz** : Die Klasse der endl. akzeptierbaren Sprachen ist abgeschlossen unter den Operationen

1. Vereinigung,
2. Komplement,
3. Durchschnitt,
4. Differenz,
5. Konkatenation,
6. Iteration.

Beweis : Seien $L_1, L_2 \subseteq \Sigma^*$ endlich akzeptierbar. Dann gibt es DEAs $A_i = (\Sigma, Q_i, \to_i, q_{0i}, F_i)$ mit $L_i = L(\mathcal{A}_i),\ i = 1, 2$, und $Q_1 \cap Q_2 = \emptyset$. Wir zeigen, dass $L_1 \cup L_2, \overline{L_1}, L_1 \cap L_2, L_1 \backslash L_2, L_1 \cdot L_2$ und L_1^* endl. akzeptierbar sind. Für $L_1 \cup L_2, L_1 \cdot L_2$ und L_1^* werden wir akzeptierende ε-NEAs angeben. Dieses erleichtert die Aufgabe sehr.

1. $L_1 \cup L_2$: Konstruiere den ε-NEA $\mathcal{B} = (\Sigma, \{q_0\} \cup Q_1 \cup Q_2, \to, q_0, F_1 \cup F_2)$ wobei $q_0 \notin Q_1 \cup Q_2$ und
$$\to = \{(q_0, \varepsilon, q_{01}), (q_0, \varepsilon, q_{02})\} \cup \to_1 \cup \to_2$$
gilt. Anschaulich sieht $\mathcal{B}$ so aus:



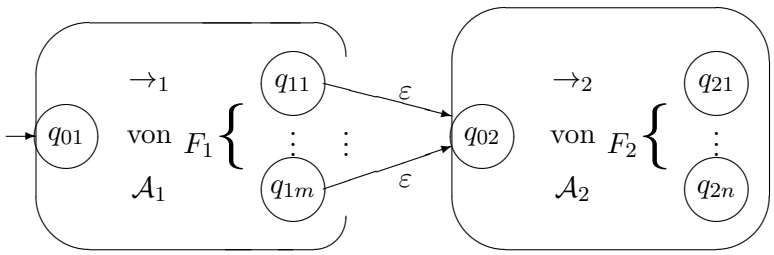
   Es ist klar, dass $L(\mathcal{B}) = L_1 \cup L_2$ gilt.

2. $\overline{L_1}$: Betrachte den DEA $\mathcal{A} = (\Sigma, Q_1, \to_1, q_{01}, Q_1 \backslash F_1)$. Dann gilt für alle $w \in \Sigma^*$:
$$\begin{aligned} w \in L(\mathcal{A}) \quad &\Leftrightarrow \quad \exists\, q \in Q_1 \backslash F_1 :\ q_{01} \overset{w}{\to}_1 q \\ \{\to_1 \text{ determ.}\} \quad & \\ &\Leftrightarrow \quad \neg\exists\, q \in F_1 :\ q_{01} \overset{w}{\to}_1 q \\ &\Leftrightarrow \quad w \notin L(\mathcal{A}_1) \\ &\Leftrightarrow \quad w \notin L_1 \end{aligned}$$
   Also gilt $L(\mathcal{A}) = \overline{L_1}$.

3. $L_1 \cap L_2$: Klar, da $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ gilt.

4. $L_1 \backslash L_2$: Klar, da $L_1 \backslash L_2 = L_1 \cap \overline{L_2}$ gilt.

5. $L_1 \cdot L_2$: Konstruiere den ε-NEA $\mathcal{B} = (\Sigma, Q_1 \cup Q_2, \to, q_{01}, F_2)$ mit
$$\to = \to_1 \cup \{(q, \varepsilon, q_{02}) \mid q \in F_1)\} \cup \to_2 .$$
   Anschaulich sieht $\mathcal{B}$ so aus:

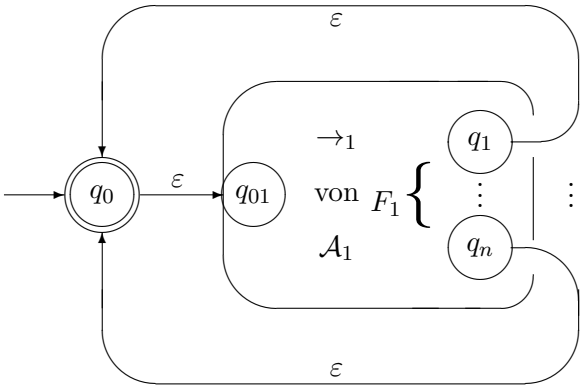


   Es ist leicht zu zeigen, dass $L(\mathcal{B}) = L_1 \cdot L_2$ gilt.

6. L_1^*: Für die Iteration konstruieren wir den ε-NEA
   $\mathcal{B} = (\Sigma, \{q_0\} \cup Q_1, \to, q_0, \{q_0\})$, wobei $q_0 \notin Q_1$ und

$$\to = \{(q_0, \varepsilon, q_{01})\} \cup \to_1 \cup \{(q, \varepsilon, q_0) \mid q \in F_1\}$$

   gilt. Anschaulich sieht $\mathcal{B}$ so aus:



   Wieder ist leicht zu zeigen, dass $L(\mathcal{B}) = L_1^*$ gilt.

Damit ist alles gezeigt. □

Bemerkung : Für die Vereinigung und den Durchschnitt gibt es auch eine interessante direkte Konstruktion von akzeptierenden DEAs. Seien $\mathcal{A}_1$ und $\mathcal{A}_2$ wie im obigen Beweis. Dann betrachten wir auf dem kartesischen Produkt $Q = Q_1 \times Q_2$ der Zustandsmengen folgende Transitionsrelation $\to \subseteq Q \times \Sigma \times Q$:

für alle $q_1, q_1' \in Q_1$ und $q_2, q_2' \in Q_2$ und $a \in \Sigma$ gilt

$$(q_1, q_2) \xrightarrow{a} (q_1', q_2') \text{ gdw } q_1 \xrightarrow{a}_1 q_1' \text{ und } q_2 \xrightarrow{a}_2 q_2'.$$

Die Relation $\xrightarrow{a}$ modelliert das *synchrone parallele Fortschreiten* der Einzelautomaten $\mathcal{A}_1$ und $\mathcal{A}_2$ beim Einlesen des Symbols a.

Betrachte man die DEAs

$$A_\cup = (\Sigma, Q, \to, (q_{01}, q_{02}), F_\cup),$$
$$A_\cap = (\Sigma, Q, \to, (q_{01}, q_{02}), F_\cap)$$

mit

$$F_\cup = \{(q_1, q_2) \in Q \mid q_1 \in F_1 \text{ oder } q_2 \in F_2\},$$
$$F_\cap = \{(q_1, q_2) \in Q \mid q_1 \in F_1 \text{ und } q_2 \in F_2\}.$$

Dann gilt $L(\mathcal{A}_\cup) = L_1 \cup L_2$ und $L(\mathcal{A}_\cap) = L_1 \cap L_2$.

Beweis : Wir zeigen die Aussage für $\mathcal{A}_\cap$. Für beliebiges $w \in \Sigma^*$ gilt:

$$\begin{aligned} w \in L(\mathcal{A}_\cap) &\Leftrightarrow \exists\ (q_1, q_2) \in F_\cap :\ (q_{01}, q_{02}) \xrightarrow{w} (q_1, q_2) \\ &\Leftrightarrow \exists\ q_1 \in F_1, q_2 \in F_2 :\ q_{01} \xrightarrow{w}_1 q_1 \text{ und } q_{02} \xrightarrow{w}_2 q_2 \\ &\Leftrightarrow w \in L(\mathcal{A}_1) \text{ und } w \in L(\mathcal{A}_2) \\ &\Leftrightarrow w \in L_1 \cap L_2 \end{aligned}$$

□

§3 Reguläre Ausdrücke

Mit Hilfe regulärer Ausdrücke können wir induktiv genau die endlich akzeptierbaren Sprachen beschreiben. Dazu betrachten wir ein vorgegebenes Alphabet Σ.

3.1 **Definition (reguläre Ausdrücke und Sprachen)**:

1. Die *Syntax der regulären Ausdrücke über* Σ ist wie folgt gegeben:
   - $\emptyset$ und ε sind reguläre Ausdrücke über Σ.
   - Für jedes $a \in \Sigma$ ist a ein regulärer Ausdruck über Σ.
   - Wenn re, re_1, re_2 reguläre Ausdrücke über Σ sind, so auch $(re_1 + re_2), (re_1 \cdot re_2), re^*$.
2. Die *Semantik eines regulären Ausdrucks* re *über* Σ ist die Sprache $L(re) \subseteq \Sigma^*$, die induktiv wie folgt definiert ist:
   - $L(\emptyset) = \emptyset$
   - $L(\varepsilon) = \{\varepsilon\}$
   - $L(a) = \{a\}$ für $a \in \Sigma$
   - $L((re_1 + re_2)) = L(re_1) \cup L(re_2)$
   - $L((re_1 \cdot re_2)) = L(re_1) \cdot L(re_2)$
   - $L(re^*) = L(re)^*$
3. Eine Sprache $L \subseteq \Sigma^*$ heißt *regulär*, falls es einen regulären Ausdruck re über Σ gibt mit $L = L(re)$.

Zur Klammereinsparung bei regulären Ausdrücken vereinbaren wir:

$$* \text{ bindet stärker als } \cdot \text{ und } \cdot \text{ bindet stärker als } +.$$

Außerdem lassen wir außenstehende Klammern weg und nutzen die Assoziativität von $\cdot$ und $+$ aus. Der Konkatenationspunkt $\cdot$ wird oft weggelassen.

Beispiel : Wir geben reguläre Ausdrücke zur Beschreibung einiger zuvor betrachteter Sprachen an.

1. Die für die lexikalische Analyse betrachtete Sprache der Identifikatoren wird durch den regulären Ausdruck

   $$re_1 = (a + \ldots + Z)(a + \ldots + Z + 0 + \ldots + 9)^*$$

   beschrieben.

2. Die zur Synchronisation zweier Programme bei der Benutzung eines gemeinsamen Druckers benutzte Sprache über $\{b_1, e_1, b_2, e_2\}$ wird durch den regulären Ausdruck

$$re_2 = (b_1e_1 + b_2e_2)^*(\varepsilon + b_1 + b_2)$$

beschrieben.

3. Sei $\Sigma = \{a_1, \dots a_n, b_1, \dots, b_m\}$ und $v = a_1 \dots a_n$. Dann wird die Sprache $L_v = \{wv \mid w \in \Sigma^*\}$ der Wörter mit Suffix v durch den regulären Ausdruck

$$re_3 = (a_1 + \dots a_n + b_1 + \dots + b_m)^* a_1 \dots a_n$$

beschrieben: $L(re_3) = L_v$

Wir zeigen allgemeiner:

3.2 **Satz (**Kleene**)**: Eine Sprache ist genau dann regulär, wenn sie endlich akzeptierbar ist.

Beweis : Wir betrachten eine Sprache $L \subseteq \Sigma^*$.

„$\Rightarrow$“: Es gelte $L = L(re)$ für einen regulären Ausdruck re über Σ. Wir zeigen mit Induktion über die Struktur von re: $L(re)$ ist endlich akzeptierbar.

Induktionsanfang: Natürlich sind $L(\emptyset),\ L(\varepsilon)$ und $L(a)$ für $a \in \Sigma$ endlich akzeptierbar.

Induktionsschritt: Seien $L(re)$, $L(re_1)$ und $L(re_2)$ bereits endlich akzeptierbar. Dann sind auch $L(re_1 + re_2), L(re_1 \cdot re_2)$ und $L(re^*)$ endlich akzeptierbar, weil die Klasse der endlich akzeptierbaren Sprachen gegenüber Vereinigung, Konkatenation und Iteration abgeschlossen ist.

„$\Leftarrow$“: Es gelte $L = L(\mathcal{A})$ für einen DEA $\mathcal{A}$ mit n Zuständen. O.B.d.A. gelte $\mathcal{A} = (\Sigma, Q, \to, 1, F)$ mit $Q = \{1, \dots, n\}$. Für $i, j \in \{1, \dots, n\}$ und $k \in \{0, 1, \dots, n\}$ definieren wir

$$L^k_{i,j} \ = \ \{w \in \Sigma^* \mid \ i \xrightarrow{w} j \text{ und } \forall\ u \in \Sigma^*, \forall\ l \in Q : \\ \text{aus } (\exists\ v\ :\ v \neq \varepsilon,\ v \neq w \text{ und } uv = w) \text{ und } i \xrightarrow{u} l \text{ folgt } l \leq k\}.$$

$L^k_{i,j}$ besteht also aus allen Wörtern w, die beim Einlesen den Automaten $\mathcal{A}$ vom Zustand i in den Zustand j überführen, so dass zwischendurch nur Zustände auftreten, deren Nummer höchstens k ist.

Wir zeigen jetzt mit Induktion nach k, dass die Sprachen $L^k_{i,j}$ alle regulär sind.

$k = 0$: Für Wörter aus $L^0_{i,j}$ darf überhaupt kein Zwischenzustand benutzt werden. Also gilt:

$$L^0_{i,j} \ = \ \begin{cases} \{a \in \Sigma \mid i \xrightarrow{a} j\} & \text{falls } i \neq j \\ \{\varepsilon\} \cup \{a \in \Sigma \mid i \xrightarrow{a} j\} & \text{falls } i = j \end{cases}$$

Damit ist $L^0_{i,j}$ als endliche Teilmenge von $\Sigma \cup \{\varepsilon\}$ regulär.

$k \to k + 1$: Seien für alle $i, j \in \{1, \dots, n\}$ die Sprachen $L^k_{i,j}$ bereits regulär. Dann gilt für alle $i, j \in \{1, \dots, n\}$:

$$L^{k+1}_{i,j} = L^k_{i,j} \cup L^k_{i,k+1} \cdot (L^k_{k+1,k+1})^* \cdot L^k_{k+1,j},$$

Denn um vom Zustand i aus den Zustand j zu erreichen, wird entweder der Zwischenzustand $k+1$ nicht benötigt, dann reicht $L_{i,j}^k$ zur Beschreibung aus; oder der Zustand $k+1$ wird ein- oder mehrfach als Zwischenzustand durchlaufen, dann wird $L_{i,k+1}^k \cdot (L_{k+1,k+1}^k)^* \cdot L_{k+1,j}^k$ zur Beschreibung verwendet. Mit Induktion nach k ergibt sich daher:

$$L_{i,j}^{k+1} \text{ ist regulär.}$$

Aus der Regularität der Sprachen $L_{i,j}^k$ folgern wir, dass L selbst regulär ist, denn es gilt

$$L = L(\mathcal{A}) = \bigcup_{j \in F} L_{1,j}^n.$$

Damit ist also der Satz von KLEENE bewiesen. □

Bemerkung : Alternativ zur oben vorgestellten Konstruktion eines regulären Ausdrucks aus einem gegebenen endlichen Automaten kann das Lösen bewachter regulärer Gleichungssysteme genommen werden: siehe Übungen.

§4 Struktureigenschaften regulärer Sprachen

Da reguläre Sprachen genau die endl. akzeptierbaren Sprachen sind, können wir aus der Endlichkeit der Zustandsmengen der akzeptierenden Automaten wichtige Eigenschaften über reguläre Sprachen ableiten. Wir betrachten zunächst das sogenannte Pumping Lemma, das eine notwendige Bedingung für reguläre Sprachen nennt. Sei dazu Σ ein beliebiges Alphabet.

4.1 **Satz (Pumping Lemma für reguläre Sprachen)**: Zu jeder regulären Sprache $L \subseteq \Sigma^*$ existiert eine Zahl $n \in \mathbb{N}$, so dass es für alle Wörter $z \in L$ mit $|z| \geq n$ eine Zerlegung $z = uvw$ mit $v \neq \varepsilon$ und $|uv| \leq n$ und folgender Eigenschaft gibt: für alle $i \in \mathbb{N}$ gilt $uv^iw \in L$, d.h. das Teilwort v kann beliebig oft „aufgepumpt“ werden, ohne dass man aus der regulären Sprache L herausfällt.

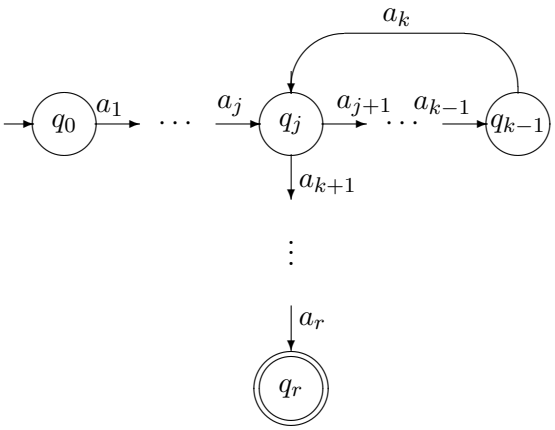
In Quantorenschreibweise:

$\forall$ regulären $L \subseteq \Sigma^* \; \exists \; n \in \mathbb{N} \, \forall \; z \in L$ mit $|z| \geq n$
$\exists \; u, v, w \in \Sigma^* : \; z = uvw$ und $v \neq \varepsilon$ und $|uv| \leq n$ und $\forall \; i \in \mathbb{N} : \; uv^iw \in L$

Beweis : Sei $L \subseteq \Sigma^*$ regulär.

Nach den Satz von KLEENE gibt es einen DEA $\mathcal{A} = (\Sigma, Q, \to, q_0, F)$ mit $L = L(\mathcal{A})$. Wir wählen $n = |Q|$ und betrachten ein Wort $z \in L$ mit $|z| \geq n$. Dann muss $\mathcal{A}$ beim Einlesen von Z mindestens einen Zustand zweimal durchlaufen. Genauer gilt folgendes:

Sei $z = a_1 \ldots a_r$ mit $r = |z| \geq n$ und $a_i \in \Sigma$ für $i = 1, \ldots, r$ und seien $q_1, \ldots, q_r \in Q$ so definiert: $q_{i-1} \xrightarrow{a_i} q_i$ für $i = 1, \ldots, r$. Dann gibt es $j, k \in \{1, \ldots, n\}$ mit $0 \leq j < k \leq n \leq r$, so dass $q_j = q_k$ gilt. Anschaulich:



Wir setzen: $u = a_1 \dots a_j$, $v = a_{j+1} \dots a_k$, $w = a_{k+1} \dots a_r$. Dann gilt $v \neq \varepsilon$ und $|uv| \leq n$ nach den Eigenschaften von j und k. Außerdem ist klar, dass der Automat $\mathcal{A}$ die q_j-Schleife beim Akzeptieren beliebig oft durchlaufen kann, d.h. es gilt für alle $i \in \mathbb{N}$: $uv^i w \in L$. □

Wir bringen eine typische Anwendung des Pumping-Lemmas, bei dem es um den Nachweis geht, dass eine bestimmte Sprache *nicht* regulär ist.

Beispiel : Die Sprache $L = \{a^n b^n \mid n \in \mathbb{N}\}$ ist nicht regulär. Wir geben einen Widerspruchsbeweis.

Annahme: L ist regulär. Dann gibt es nach dem Pumping-Lemma ein $n \in \mathbb{N}$ mit den dortgenannten Eigenschaften. Betrachte jetzt $z = a^n b^n$. Da $|z| \geq n$ gilt, lässt sich z zerlegen in $z = uvw$ mit $v \neq \varepsilon$ und $|uv| \leq n$, so dass für alle $i \in \mathbb{N}$ gilt: $uv^i w \in L$. Aber v besteht nur aus Symbolen a, so dass speziell $uw = a^{n-|v|} b^n \in L$ gelten müsste. *Widerspruch*

Das obige Beispiel zeigt, dass reguläre Sprachen nicht unbeschränkt zählen können. Weitere Anwendungen des Pumping-Lemmas werden wir in Abschnitt über Entscheidbarkeitsfragen kennenlernen.

NERODE-Rechtskongruenz

Eine charakteristische, also notwendige und hinreichende Bedingung für die Regularität von Sprachen $L \subseteq \Sigma^*$ erhalten wir durch Betrachtung der sogenannten NERODE-Rechtskongruenz.

4.2 **Definition** :

Sei $L \subseteq \Sigma^*$ eine beliebige Sprache. Die NERODE-*Rechtskongruenz* von L ist eine zweistellige Relation $\equiv_L$ auf Σ^*, also $\equiv_L \subseteq \Sigma^* \times \Sigma^*$, die für $u, v \in \Sigma^*$ wie folgt definiert ist:

$u \equiv_L v$ genau dann, wenn für alle $w \in \Sigma^*$ gilt: $uw \in L \Leftrightarrow vw \in L$.

Es gilt also $u \equiv_L v$, wenn sich u und v in gleicher Weise zu Wörtern aus L verlängern lassen. Insbesondere gilt (mit $w = \varepsilon$) $u \in L \Leftrightarrow v \in L$.

Bemerkung : Der Name „Rechtskongruenz“ ist durch folgende Eigenschaften gerechtfertigt:

1. $\equiv_L$ ist eine Äquivalenzrelation auf Σ^*, also reflexiv, symmetrisch und transitiv.
2. $\equiv_L$ ist verträglich mit der Konkatenation von rechts, d.h. aus $u \equiv_L v$ folgt $uw \equiv_L vw$ für alle $w \in \Sigma^*$

Da $\equiv_L$ eine Äquivalenzrelation ist, können wir den *Index von* $\equiv_L$, d.h. die Anzahl der Äquivalenzklassen von $\equiv_L$ untersuchen.

4.3 **Satz** (Myhill und Nerode):

Eine Sprache $L \subseteq \Sigma^*$ ist genau dann regulär, wenn $\equiv_L \subseteq \Sigma^* \times \Sigma^*$ einen endlichen Index hat.

Beweis : „⇒“: Sei L regulär, also $L = L(\mathcal{A})$ für einen DEA $\mathcal{A} = (\Sigma, Q, \to, q_0, F)$. Wir führen folgende zweistellige Relation $\equiv_A$ auf Σ^* ein. Für $u, v \in \Sigma^*$ gilt:

$u \equiv_A v$ genau dann, wenn es ein $q \in Q$ gibt, mit $q_0 \xrightarrow{u} q$ und $q_0 \xrightarrow{v} q$,

d.h. falls die Eingaben u und v den Automaten $\mathcal{A}$ von q_0 aus in den selben Zustand q überführen. Man beachte, dass q eindeutig bestimmt ist, weil $\mathcal{A}$ deterministisch ist. Daher ist $\equiv_A$ eine Äquivalenzrelation auf Σ^*.

Wir zeigen: $\equiv_A$ ist eine *Verfeinerung* von $\equiv_L$, d.h. für alle $u, v \in \Sigma^*$ gilt:

$$\text{Aus } u \equiv_A v \text{ folgt } u \equiv_L v.$$

Sei nämlich $u \equiv_A v$ und $w \in \Sigma^*$. Dann gilt:

$$\begin{aligned} uw \in L &\Leftrightarrow \exists\, q \in Q\, \exists\, q' \in F : q_0 \xrightarrow{u} q \xrightarrow{w} q' \\ &\Leftrightarrow \{u \equiv_A v\} \\ &\quad \exists\, q \in Q, q' \in F : q_0 \xrightarrow{v} q \xrightarrow{w} q' \\ &\Leftrightarrow vw \in L. \end{aligned}$$

Es gibt also mindestens so viele Äquivalenzklassen von $\equiv_A$ wie von $\equiv_L$. Daher gilt

$$\begin{aligned} & Index(\equiv_L) \\ \leq\ & Index(\equiv_A) \\ =\ & \text{Anzahl der von } q_0 \text{ aus erreichbaren Zustände} \\ \leq\ & |Q|, \end{aligned}$$

so dass $\equiv_L$ einen endlichen Index hat.

„⇐“: Sei $L \subseteq \Sigma^*$ eine Sprache und $k \in \mathbb{N}$ der endliche Index von $\equiv_L$. Wir wählen k Wörter $u_1, \ldots, u_k \in \Sigma^*$ mit $u_1 = \varepsilon$ als Repräsentanten der Äquivalenzklassen von $\equiv_L$. Dann ist Σ^* als disjunkte Vereinigung dieser Äquivalenzklassen darstellbar:

$$\Sigma^* = [u_1] \,\dot\cup \ldots \dot\cup\, [u_k].$$

Insbesondere gibt es also für jedes Wort $u \in \Sigma^*$ ein $i \in \{1, \ldots, k\}$ mit $[u] = [u_i]$.

Wir konstruieren jetzt folgenden *Äquivalenzklassen-Automaten*

$\mathcal{A}_L = (\Sigma, Q_L, \to_L, q_L, F_L)$:

$$\begin{aligned} Q_L &= \{[u_1], \ldots, [u_k]\}, \\ q_L &= [u_1] = [\varepsilon] \\ F_L &= \{[u_j] \mid u_j \in L\}, \end{aligned}$$

und für $i, j \in \{1, \ldots, k\}$ und $a \in \Sigma$ setzen wir

$$[u_i] \xrightarrow{a}_L [u_j] \text{ gdw } [u_j] = [u_i a].$$

Dann ist $\mathcal{A}_L$ ein DEA und für alle Wörter $w \in \Sigma^*$ gilt:

$$[\varepsilon] \xrightarrow{w}_L [u_j] \text{ gdw } [u_j] = [w],$$

genauer ist für $w = a_1 \ldots a_n$

$$[\varepsilon] \xrightarrow{w}_L [u_j] \text{ gdw } [\varepsilon] \xrightarrow{a_1}_L [a_1] \ldots \xrightarrow{a_n}_L [a_1 \ldots a_n] = [u_j],$$

und damit

$$\begin{aligned} & w \in L(\mathcal{A}_L) = \\ \Leftrightarrow \quad & \exists\, [u_j] \in F_L : [\varepsilon] \xrightarrow{w}_L [u_j] \\ \Leftrightarrow \quad & \exists\, u_j \in L : [u_j] = [w] \\ \Leftrightarrow \quad & w \in L \end{aligned}$$

Also akzeptiert $\mathcal{A}_L$ die Sprache L. Daher ist L regulär. □

Wir wenden die Beweistechnik des Satzes von Myhill und Nerode an, um die Zustandszahl von Automaten zu minimieren. Dabei beziehen wir uns insbesondere auf den deterministischen Äquivalenzklassen-Automaten $\mathcal{A}_L$ aus dem Beweis.

4.4 **Korollar** : Sei $L \subseteq \Sigma^*$ regulär und $k = Index(\equiv_L)$. Dann besitzt jeder DEA, der L akzeptiert, wenigstens k Zustände. Die Minimalzahl k wird von dem DEA $\mathcal{A}_L$ erreicht. Es kann aber NEA 's mit weniger als k Zuständen geben, die L akzeptieren.

Beweis : Im Beweis des Satzes von Myhill und Nerode haben wir unter „⇒“ gezeigt, dass jeder DEA $\mathcal{A} = (\Sigma, Q, \to, q_0, F)$, der L akzeptiert, wenigstens k Zustände besitzt. Dabei ist $k = Index(\equiv_L) \leq |Q|$. Unter „⇐“ haben wir den DEA $\mathcal{A}_L$ mit k Zuständen konstruiert, der L akzeptiert. □

Der Äquivalenzklassen-Automat $\mathcal{A}_L$ ist der Prototyp aller L akzeptierenden DEA's mit minimaler Zustandszahl k. Wir können nämlich zeigen, dass jeder andere DEA, der L akzeptiert und k Zustände hat, zu $\mathcal{A}_L$ isomorph ist, d.h. aus $\mathcal{A}_L$ durch eine bijektive Umbenennung der Zustände entsteht.

4.5 **Definition** : Zwei DEA's oder NEA's $\mathcal{A}_i = (\Sigma, Q_i, \to_i, q_{0i}, F_i), i = 1, 2$, heißen *isomorph* , falls es eine Bijektion $\beta : Q_1 \to Q_2$ mit folgenden Eigenschaften gibt:

- $\beta(q_{01}) = q_{02}$,
- $\beta(F_1) = \{\beta(q) \mid q \in F_1\} = F_2$,
- $\forall\, q, q' \in Q_1\ \forall\, a \in \Sigma : q \xrightarrow{a}_1 q' \Leftrightarrow \beta(q) \xrightarrow{a}_2 \beta(q')$.

Die Bijektion β heißt *Isomorphismus* von $\mathcal{A}_1$ auf $\mathcal{A}_2$.

Man beachte, dass Isomorphie eine Äquivalenzrelation auf endlichen Automaten ist. Wir zeigen jetzt die angekündigte Aussage.

4.6 **Satz** : Sei $L \subseteq \Sigma^*$ regulär und $k = Index(\equiv_L)$. Dann ist jeder DEA $\mathcal{A}$, der L akzeptiert und k Zustände besitzt, zu $\mathcal{A}_L$ isomorph.

Beweis : Gegeben sei $\mathcal{A} = (\Sigma, Q, \to, q_1, F)$ mit $L(\mathcal{A}) = L$ und $|Q| = k$ und der Äquivalenzklassen-Automat des Satzes von Myhill und Nerode mit $Q_L = \{u_1, \ldots, [u_k]\}$ und $u_1 = \varepsilon$. Wir definieren für jedes $i \in \{1, \ldots, k\}$ den Zustand $q_i \in Q$ durch die Transition $q_1 \xrightarrow{u_i} q_i$.

Man beachte, dass q_i eindeutig bestimmt ist, da die Transitionsrelation $\to$ deterministisch ist.

Jetzt definieren wir die Abbildung $\beta : Q_L \to Q$ durch

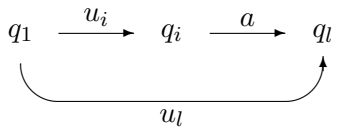
$$\beta([u_i]) = q_i$$

für $i \in 1, \ldots, k$ und zeigen, dass β ein Isomorphismus von $\mathcal{A}_L$ auf $\mathcal{A}$ ist.

1. β ist injektiv: Sei $q_i = q_j$. Dann gilt $q_1 \xrightarrow{u_i} q_i$ und $q_1 \xrightarrow{u_j} q_i$. Also gilt für alle $w \in \Sigma^* : u_i w \in L \Leftrightarrow u_j w \in L$. Also ist $u_i \equiv_L u_j$ und damit $[u_i] = [u_j]$.
2. β ist surjektiv: Diese Eigenschaft folgt aus (1) und der Tatsache, dass $k = |Q|$. Damit gilt insbesondere $Q = \{q_1, \ldots, q_k\}$.
3. $\beta([q_L]) = \beta([u_1]) = \beta([\varepsilon]) = q_1$
4. $\beta(F_L) = F : [u_j] \in F_L \Leftrightarrow u_j \in L \Leftrightarrow q_j \in F$
5. Für alle $i, j \in \{1, \ldots, k\}$ und $a \in \Sigma$ gilt:

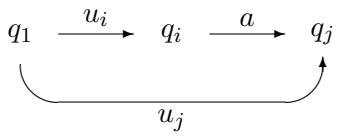
$$[u_i] \xrightarrow{a}_L [u_j] \Leftrightarrow q_i \xrightarrow{a} q_j.$$

- Beweis von „⇒“: Sei $[u_i] \xrightarrow{a}_L [u_j]$. Dann gilt nach Definition von $\to_L$: $[u_i a] = [u_j]$. Es gibt ein $l \in \{1, \ldots, k\}$ mit $q_i \xrightarrow{a} q_l$. Nach Definition von q_i und q_l haben wir folgendes Bild:



  Da die Wörter $u_i a$ und u_l zum selben Zustand q_l führen, folgt $u_i a \equiv_L u_l$. Also gilt $[u_j] = [u_i a] = [u_l]$. Nach Wahl der $u_1, \ldots, u_k$ in A_L folgt $u_j = u_l$, sogar $j = l$. Damit folgt $q_i \xrightarrow{a} q_j$ wie gewünscht.

- Beweis von „⇐“: Sei $q_i \xrightarrow{a} q_j$. Damit haben wir analog wie oben folgendes Bild:



  Daraus schließen wir wie oben: $u_i a \equiv_L u_j$. Also gilt $[u_i a] = [u_j]$, woraus nach Definition von $\rightarrow_L$ die Transition $[u_i] \xrightarrow{a}_L [u_j]$ folgt.

Also sind $\mathcal{A}_L$ und $\mathcal{A}$ isomorph. □

Zu jeder regulären Sprache $L \subseteq \Sigma^*$ mit k als Index von $\equiv_L$ gibt es also einen bis auf Isomorphie eindeutigen DEA, mit der Minimalzahl von k Zuständen, der L akzeptiert.

4.7 **Definition** : Der *Minimalautomat* für eine reguläre Sprache $L \subseteq \Sigma^*$ ist der bis auf Isomorphie eindeutige DEA, der L akzeptiert und dessen Zustandszahl gleich dem Index der NERODE-Rechtskongruenz $\equiv_L$ ist.

Wir können den Minimalautomaten für eine reguläre Sprache $L \subseteq \Sigma^*$ aus jedem DEA $\mathcal{A} = (\Sigma, Q, \rightarrow, q_0, F)$, der L akzeptiert, durch *Reduktion* algorithmisch konstruieren. Die Reduktion umfasst die folgenden beiden Schritte:

1. Eliminieren nicht erreichbarer Zustände.

   Ein Zustand $q \in Q$ heißt erreichbar, falls es ein Wort $w \in \Sigma^*$ mit $q_0 \xrightarrow{w} q$ gibt. Die Teilmenge der erreichbaren Zustände von Q ist berechenbar, weil man sich bei den Wörtern w mit $q_0 \xrightarrow{w} q$ auf solche der Länge $\leq |Q|$ einschränken kann(vgl. den Beweis des Pumping Lemmas).

2. Zusammenfassen äquivalenter Zustände.

   Für $q \in Q, w \in \Sigma^*$ und $S \subseteq Q$ schreiben wir $q \xrightarrow{w} S$, falls es ein $q' \in S$ mit $q \xrightarrow{w} q'$ gibt. Zwei Zustände $q_1, q_2 \in Q$ heißen *äquivalent*, abgekürzt $q_1 \sim q_2$, falls für alle $w \in \Sigma^*$ gilt:

   $$q_1 \xrightarrow{w} F \Leftrightarrow q_2 \xrightarrow{w} F,$$

   d.h. von q_1 und q_2 führen die selben Wörter zu akzeptierenden Endzuständen.

   Für erreichbare Zustände besteht folgender enger Zusammenhang zwischen Äquivalenz und der NERODE-Rechtskongruenz $\equiv_L$. Sei $q_0 \xrightarrow{u} q_1$ und $q_0 \xrightarrow{v} q_2$. Dann gilt :

   $$q_1 \sim q_2 \Leftrightarrow u \equiv_L v \Leftrightarrow [u] = [v].$$

   Man sieht durch Vergleich mit dem Äquivalenzklassen-Automaten $\mathcal{A}_L$, dass im Minimalautomaten äquivalente Zustände zusammenfallen müssen. In $\mathcal{A}_L$ werden q_0, q_1 und q_2 nämlich durch die Äquivalenzklassen $[\varepsilon]$, $[u]$ und $[v]$ dargestellt, so dass aus $[\varepsilon] \xrightarrow{u} [u]$ und $[\varepsilon] \xrightarrow{v} [v]$ folgt:

   $$[u] \sim [v] \Leftrightarrow u \equiv_L v \Leftrightarrow [u] = [v].$$

   Algorithmen zur Berechnung der Zustandsäquivalenz werden in den Übungen behandelt.

§5 Entscheidbarkeitsfragen

Wir stellen zunächst fest, dass die folgenden Konstruktionen algorithmisch durchführbar sind:

- ε-NEA $\rightarrow$ NEA $\rightarrow$ DEA
- DEA $\rightarrow$ Minimalautomat
- ε-NEAs für folgende Operationen auf endlich akzeptierbaren Sprachen:
  Vereinigung, Komplement, Durchschnitt, Differenz, Konkatenation und Iteration
- regulärer Ausdruck $\rightarrow$ NEA $\rightarrow$ DEA $\rightarrow$ regulärer Ausdruck

Damit können wir uns Entscheidbarkeitsfragen für die durch endliche Automaten oder reguläre Ausdrücke dargestellten Sprachen zuwenden. Wegen der genannten Konstruktionen beschränken wir uns auf Sprachen, die durch DEA's dargestellt sind. Wir betrachten folgende Probleme für reguläre Sprachen.

Wortproblem	Gegeben:	DEA $\mathcal{A}$ und ein Wort w
	Frage:	Gilt $w \in L(\mathcal{A})$?
Leerheitsproblem	Gegeben:	DEA $\mathcal{A}$
	Frage:	Gilt $L(\mathcal{A}) = \emptyset$?
Endlichkeitsproblem	Gegeben:	DEA $\mathcal{A}$
	Frage:	Ist $L(\mathcal{A})$ endlich ?
Äquivalenzproblem	Gegeben:	DEA 's $\mathcal{A}_1$ und $\mathcal{A}_2$
	Frage:	Gilt $L(\mathcal{A}_1) = L(\mathcal{A}_2)$?
Inklusionsproblem	Gegeben:	DEA 's $\mathcal{A}_1$ und $\mathcal{A}_2$
	Frage:	Gilt $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$?
Schnittproblem	Gegeben:	DEA 's $\mathcal{A}_1$ und $\mathcal{A}_2$
	Frage:	Gilt $L(\mathcal{A}_1) \cap L(\mathcal{A}_2) = \emptyset$?

5.1 **Satz (Entscheidbarkeit)**: Für reguläre Sprachen sind

- das Wortproblem,
- das Leerheitsproblem,
- das Endlichkeitsproblem,
- das Äquivalenzproblem,
- das Inklusionsproblem,
- das Schnittproblem

alle algorithmisch entscheidbar.

Beweis : *Wortproblem*: Man setzt $\mathcal{A}$ auf das vorgelegte Wort w an und stellt fest, ob ein Endzustand von $\mathcal{A}$ erreicht wird. $\mathcal{A}$ ist also selbst das Entscheidungsverfahren.

Leerheitsproblem: Sei n die nach dem Pumping Lemma zur regulären Sprache $L(\mathcal{A})$ gehörige Zahl. Dann gilt:

$$L(\mathcal{A}) = \emptyset \Leftrightarrow \neg\exists\ w \in L(\mathcal{A}) : |w| < n \qquad (*)$$

Beweis von $(*)$: „$\Rightarrow$“ ist klar. „$\Leftarrow$“ durch Kontraposition: Sei $L(\mathcal{A}) \neq \emptyset$. Falls es ein Wort $w \in L(\mathcal{A})$ mit $|w| < n$ gibt, ist nichts zu zeigen. Sonst gibt es ein Wort $w \in L(\mathcal{A})$ mit $|w| \geq n$. Durch sukzessives Anwenden des Pumping Lemmas, jeweils mit $i = 0$, erhalten wir ein Wort $w_0 \in L(\mathcal{A})$ mit $|w_0| < n$. Damit sind wir fertig.

Das Entscheidungsverfahren für $L(\mathcal{A}) = \emptyset$ löst nun für jedes Wort über dem Eingabealphabet von $\mathcal{A}$ mit mit $|w| < n$ das Wortproblem „$w \in L(\mathcal{A})$?“. Falls die Antwort stets „nein“ lautet, wird „$L(\mathcal{A}) = \emptyset$“ ausgegeben. Sonst wird „$L(\mathcal{A}) \neq \emptyset$“ ausgegeben.

Endlichkeitsproblem: Sei n wie eben. Dann gilt:

$$L(\mathcal{A}) \text{ ist endlich } \Leftrightarrow \neg\exists\ w \in L(\mathcal{A}) : n \leq |w| < 2n \qquad (**)$$

Beweis von $(**)$: „$\Rightarrow$“: Gäbe es ein Wort $w \in L(\mathcal{A})$ mit $|w| \geq n$, so wäre $L(\mathcal{A})$ nach dem Pumping Lemma unendlich. „$\Leftarrow$“ durch Kontraposition: Sei $L(\mathcal{A})$ unendlich. Dann gibt es Wörter beliebig großer Länge in $L(\mathcal{A})$, insbesondere ein Wort w mit $w \geq 2n$. Sukzessives Anwenden des Pumping Lemmas, jeweils mit $i = 0$, liefert ein Wort w_0 mit $n \leq |w_0| < 2n$, weil sich mit $i = 0$ das vorgelegte Wort um maximal n Buchstaben verkürzt.

Durch $(**)$ kann das Endlichkeitsproblem durch endlichmaliges Lösen des Wortproblems entschieden werden.

Äquivalenzproblem: Man konstruiere zunächst einen DEA $\mathcal{A}$ mit folgender Eigenschaft:

$$L(\mathcal{A}) = (L(\mathcal{A}_1) \cap \overline{L(\mathcal{A}_2)}) \cup (L(\mathcal{A}_2) \cap \overline{L(\mathcal{A}_1)}).$$

Offenbar gilt:

$$L(\mathcal{A}_1) = L(\mathcal{A}_2) \Leftrightarrow L(\mathcal{A}) = \emptyset \qquad (***)$$

Damit wird das Äquivalenzproblem für Automaten auf das Leerheitsproblem für $\mathcal{A}$ zurückgeführt. Natürlich ist die obige Konstruktion von $\mathcal{A}$ aufwendig durchzuführen. Alternativ kann man folgende *Produktkonstruktion* analog zur letzten Bemerkung in Abschnitt 2 dieses Kapitels benutzen. Seien $\mathcal{A}_i = (\Sigma, Q_i, \rightarrow_i, q_{0i}, F_i), i = 1, 2$ gegeben. Betrachte dann $Q = Q_1 \times Q_2$ mit folgender Transitionsrelation $\rightarrow \subseteq Q \times \Sigma \times Q$

für alle $q_1, q_1' \in Q_1$ und $q_2, q_2' \in Q_2$ und $a \in \Sigma$ gilt

$(q_1.q_2) \xrightarrow{a} (q_1', q_2')$ gdw $q_1 \xrightarrow{a} q_1'$ und $q_2 \xrightarrow{a} q_2'$. Definiere dann $\mathcal{A} = (\Sigma, Q, \rightarrow, (q_{01}, q_{02}), F)$ mit $F = \{(q_1, q_2) \in Q \mid q_1 \in F_1 \Leftrightarrow q_2 \notin F_2\}$.

Dann gilt (***) für diesen DEA $\mathcal{A}$:

$$\begin{aligned} L(\mathcal{A}_1) = L(\mathcal{A}_2) \Leftrightarrow \quad & \forall\, w \in \Sigma^* :\ (w \in L(\mathcal{A}_1) \Leftrightarrow w \in L(\mathcal{A}_2)) \\ \Leftrightarrow \quad & \forall\, w \in \Sigma^* :\ ((\exists\, q_1 \in F_1 :\ q_0 \xrightarrow{w} q_1) \Leftrightarrow (\exists\, q_2 \in F_2 :\ q_0 \xrightarrow{w} q_2)) \\ \Leftrightarrow \quad & \forall\, w \in \Sigma^*\ \forall\, (q_1, q_2) \in Q :\ (q_{01}, q_{02}) \xrightarrow{w} (q_1, q_2) \Rightarrow (q_1, q_2) \notin F \\ \Leftrightarrow \quad & L(\mathcal{A}) = \emptyset \end{aligned}$$

Inklusionsproblem: Man konstruiere einen DEA $\mathcal{A}$, so dass

$$L(\mathcal{A}) = L(\mathcal{A}_1) \cap \overline{L(\mathcal{A}_2)}$$

gilt. Wegen

$$L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2) \Leftrightarrow L(\mathcal{A}) = \emptyset$$

kann das Inklusionsproblem für $\mathcal{A}_1$ und $\mathcal{A}_2$ auf das Leerheitsproblem von $\mathcal{A}$ zurückgeführt werden.

Schnittproblem: Man konstruiere einen DEA $\mathcal{A}$ mit

$$L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2),$$

so dass das Schnittproblem $\mathcal{A}_1$ und $\mathcal{A}_2$ auf das Leerheitsproblem von $\mathcal{A}$ zurückgeführt werden kann. Für $\mathcal{A}$ kann die Produktkonstruktion $\mathcal{A}_\cap$ aus der letzten Bemerkung des Abschnitts 2 genommen werden. □

§6 Automatische Verifikation

Für Programme, die durch endliche Automaten dargestellt werden können, ist eine automatische Verifikation durchführbar. Allgemein lautet das *Verifikationsproblem* wie folgt:

Gegeben	:	Ein Programm P und eine Spezifikation S.
Frage	:	Erfüllt P die Spezifikation S ? Oder: Ist P bezüglich S korrekt ?

Im Englischen wird dieses Problem auch als *Model Checking* bezeichnet, weil im Sinne der Logik gefragt wird, ob das Programm P ein Modell der Spezifikation S ist. Wir betrachten hier folgenden Spezialfall dieses Problems:

- Programm P $\stackrel{\wedge}{=}$ endlicher Automat (DEA, NEA oder ε-NEA) $\mathcal{A}$
- Spezifikation S $\stackrel{\wedge}{=}$ ein *erweiterter* regulärer Ausdruck re, und zwar erweitert um die Operatoren $re_1 \cap re_2$ (Durchschnitt), $\overline{re}$ (Komplement) bzw. $\neg re$ (Negation) sowie Σ als Abkürzung für $a_1 + \ldots + a_n$, falls $\Sigma = \{a_1, \ldots, a_n\}$ gilt. Die Klasse der regulären Sprachen wird damit natürlich nicht verlassen, wie wir aus den Abschlusseigenschaften von regulären Sprachen folgern. Die Erweiterung dient nur dazu, möglichst übersichtliche reguläre Ausdrücke zu erhalten.
- P erfüllt S $\stackrel{\wedge}{=}$ $L(\mathcal{A}) \subseteq L(re)$. Dieser Test ist automatisch durchführbar, weil das Inklusionsproblem für reguläre Sprachen entscheidbar ist.

Wir erläutern jetzt die Methode der *Spezifikation durch Gegenbeispiele*. Die Idee ist, dass ein regulärer Ausdruck re die Menge aller Wörter beschreibt, die der Automat $\mathcal{A}$ *nicht* akzeptieren soll, die also Gegenbeispiele für das korrekte Verhalten von $\mathcal{A}$ darstellen. Daher spezifiziert $\neg re$ die Menge der erlaubten Wörter. Wir formen die Korrektheitsaussage, dass $\mathcal{A}$ die Spezifikation $\neg re$ erfüllt, wie folgt äquivalent um:

$$\begin{aligned} & \mathcal{A} \text{ erfüllt } \neg re \\ \Leftrightarrow \quad & \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\neg re) \\ \Leftrightarrow \quad & \mathcal{L}(\mathcal{A}) \subseteq \overline{\mathcal{L}(re)} \\ \Leftrightarrow \quad & \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(re) = \emptyset \\ \Leftrightarrow \quad & \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}(re)) = \emptyset \\ \Leftrightarrow \quad & \mathcal{L}(\mathcal{A} \mid\mid \mathcal{A}(re)) = \emptyset. \end{aligned}$$

Dabei ist $\mathcal{A}(re)$ der Automat, der die durch re beschriebene reguläre Sprache akzeptiert. Der Operator $||$ ist die synchrone parallele Komposition, die unten definiert wird.

Die Gleichung $\mathcal{L}(\mathcal{A} \mid\mid \mathcal{A}(re)) = \emptyset$ gilt genau dann, wenn in der synchronen parallelen Komposition keiner der Endezustände von $\mathcal{A}(re)$ erreicht wird. Diese Endzustände werden deshalb auch *schlechte Zustände* genannt. Insgesamt gilt also

$$\begin{aligned} & \mathcal{A} \text{ erfüllt } \neg re \\ \Leftrightarrow \quad & \text{In } \mathcal{A} \mid\mid \mathcal{A}(re) \text{ wird kein schlechter Zustand von } \mathcal{A}(re) \text{ erreicht.} \end{aligned}$$

Da für endliche Automaten (DEA, NEA oder ε-NEA) das Erreichbarkeitsproblem von Zuständen entscheidbar ist, ist das Verifikationsproblem für Spezifikationen durch Gegenbeispiele auf diese Weise entscheidbar.

Gegeben seien zwei ε-NEAs $\mathcal{A}_i = (\Sigma, Q_i, \rightarrow_i, q_{0i}, F_i)$, $i = 1, 2$. Dann ist die *synchrone parallele Komposition* durch folgenden ε-NEA $\mathcal{A}_1 \mid\mid \mathcal{A}_2$ definiert:

$$\mathcal{A}_1 \mid\mid \mathcal{A}_2 = (\Sigma, Q, \rightarrow, q_0, F)$$

mit

- $Q = Q_1 \times Q_2$ (Produkt der einzelnen Zustandsmengen),
- $q_0 = (q_{01}, q_{02})$ (Paar der einzelnen Anfangszustände),
- $F = F_1 \times F_2$ (Produkt der einzelnen Endzustandsmengen)

und der folgenden Transitionsrelation $\rightarrow$, die wir induktiv durch folgende Regeln definieren:

- spontane Übergänge:

$$\frac{q_1 \xrightarrow{\varepsilon}_1 q_1'}{(q_1, q_2) \xrightarrow{\varepsilon} (q_1', q_2)} \quad \text{und} \quad \frac{q_2 \xrightarrow{\varepsilon}_2 q_2'}{(q_1, q_2) \xrightarrow{\varepsilon} (q_1, q_2')}$$

- synchrones paralleles Fortschreiten:

$$\frac{q_1 \xrightarrow{a}_1 q_1' \ , \ q_2 \xrightarrow{a}_2 q_2'}{(q_1, q_2) \xrightarrow{a} (q_1', q_2')} \quad \text{für } a \in \Sigma.$$

Bemerkung : Es gilt $\mathcal{L}(\mathcal{A}_1 \mid\mid \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

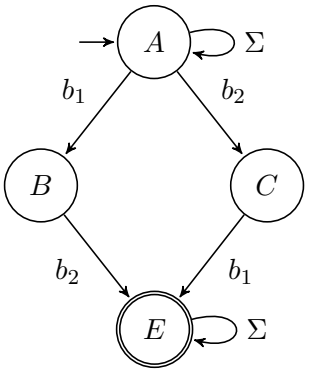
Für DEA $\mathcal{A}_1, \mathcal{A}_2$ entspricht $\mathcal{A}_1 \mid\mid \mathcal{A}_2$ der Definition von $\mathcal{A}_\cap$ auf Seite 23.

Beispiel : Betrachten wir noch einmal das Beispiel aus dem Bereich der Betriebssysteme, wo zwei Programme durch Operationen b_1 und b_2 auf einen gemeinsamen Drucker zugreifen und mit e_1 und e_2 die Beendigung der Benutzung anzeigen (siehe Seite 13). Wir setzen $\Sigma = \{b_1, b_2, e_1, e_2\}$ und beschreiben die Menge der Gegenbeispiele durch folgenden erweiterten regulären Ausdruck:

$$re = \Sigma^* b_1 b_2 \Sigma^* + \Sigma^* b_2 b_1 \Sigma^*.$$

Der reguläre Ausdruck re wird durch die Verwendung von Σ übersichtlicher. Der folgende Automat akzeptiert die spezifizierte reguläre Sprache.

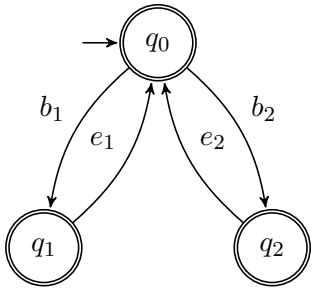
$\mathcal{A}(re)$:



Dabei ist E der “schlechte” Zustand.

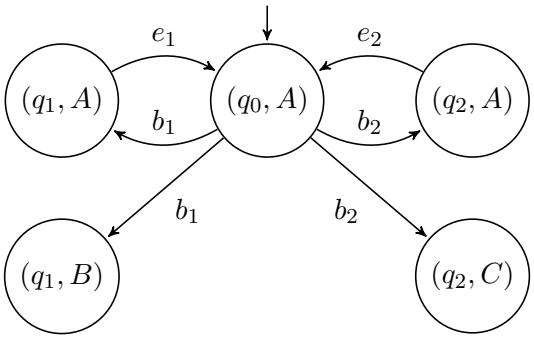
Zunächst betrachten wir eine Druckersteuerung, die durch folgenden Automaten gegeben ist.

$\mathcal{A}_1$:



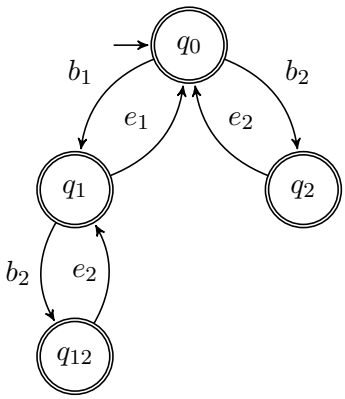
Um zu prüfen, ob $\mathcal{A}_1$ die Spezifikation $\neg\, re$ erfüllt, betrachten wir folgende synchrone parallele Komposition.

$\mathcal{A}_1 \mid\mid \mathcal{A}(re)$:



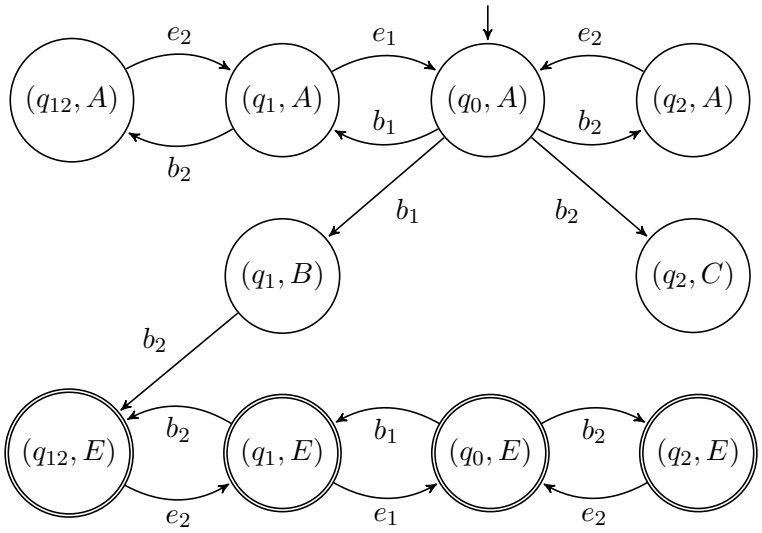
Da der schlechte Zustand E in $\mathcal{A}_1 \mid\mid \mathcal{A}(re)$ nicht erreicht wird, gilt: $\mathcal{A}_1$ erfüllt $\neg\, re$. Als nächstes betrachten wir die durch den folgenden Automaten gegebene Druckersteuerung.

$\mathcal{A}_2$:



Um zu prüfen, ob $\mathcal{A}_2$ die Spezifikation $\neg\, re$ erfüllt, betrachten wir folgende synchrone parallele Komposition.

$\mathcal{A}_2 \mid\mid \mathcal{A}(re)$:



$\mathcal{A}_2$ erfüllt $\neg re$ *nicht*, weil in $\mathcal{A}_2 \mid\mid \mathcal{A}(re)$ der schlechte Zustand E von $\mathcal{A}(re)$ erreichbar ist.

Kapitel III

Kontextfreie Sprachen und Kellerautomaten

Im letzten Kapitel haben wir gesehen, dass reguläre Sprachen in der Informatik vielfache Anwendungen finden (z.B. lexikalische Analyse und Teilworterkennung) und besonders leicht zu handhaben sind (Darstellbarkeit durch endliche Automaten und reguläre Ausdrücke, angenehme Abschluss- und Entscheidbarkeitseigenschaften). Dennoch reichen sie für eine wichtige Aufgabe der Informatik nicht aus: die *Syntaxbeschreibung von Programmiersprachen.*

Der Grund ist, dass Programmiersprachen Klammerstrukturen beliebiger Schachtelungstiefe zulassen, zum Beispiel als

- arithmetische Ausdrücke der Form $3 * (4 - (x + 1))$,
- Listen der Form ($\texttt{CAR}(\texttt{CONS}\ x\ y\ )$) oder
- Anweisungen der Form

$$
\begin{array}{l}
\texttt{while}(b1)\ \{ \\
\quad x = e1; \\
\quad \texttt{while}(b1)\ \{ \\
\quad\quad y = e2; \\
\quad\quad z = e3; \\
\quad \} \\
\}
\end{array}
$$

Im Abschnitt 4 hatten wir mit Hilfe des Pumping Lemmas gezeigt, dass bereits das einfachste Muster einer solchen Klammerstruktur, nämlich die Sprache

$$L = \{a^n b^n \mid n \in \mathbb{N}\}$$

nicht mehr regulär ist. Für die Syntaxbeschreibung von Programmiersprachen haben sich kontextfreien Sprachen durchgesetzt.

§1 Kontextfreie Grammatiken

Wir untersuchen zunächst die zugehörigen Grammatiken.

1.1 **Definition** : Eine *kontextfreie Grammatik* ist eine Struktur $G = (N, T, P, S)$, wobei folgendes gilt:

(i) N ist ein Alphabet von *Nichtterminalsymbolen*,

(ii) T ist ein Alphabet von *Terminalsymbolen* mit $N \cap T = \emptyset$,

(iii) $S \in N$ ist das *Startsymbol*,

(iv) $P \subseteq N \times (N \cup T)^*$ ist eine endliche Menge von *Produktionen* oder *Regeln*.

Dabei benutzen wir folgende Konventionen:

- $A, B, C, \ldots$ stehen für Nichtterminalsymbole,
- $a, b, c, \ldots$ stehen für Terminalsymbole,
- $u, v, w, \ldots$ stehen für Wörter aus Terminal- und Nichtterminalsymbolen.

Eine Produktion $(A, u) \in P$ wird meist in Pfeilnotation $A \to u$ geschrieben. Falls mehrere Produktionen dieselbe linke Seite besitzen, etwa

$$A \to u_1, A \to u_2, \ldots, A \to u_k,$$

so schreiben wir dieses kürzer als eine einzige „Metaregel“

$$A \to u_1 \,|\, u_2 \,|\, \ldots \,|\, u_k$$

oder auch

$$A ::= u_1 \,|\, u_2 \,|\, \ldots \,|\, u_k. \qquad (*)$$

Dabei ist $|$ ein „Metasymbol“ zur Trennung der Alternativen $u_1, \ldots, u_k$, das nicht in $N \cup T$ vorkommen möge.

Werden die Produktionen einer kontextfreien Grammatik in der Form $(*)$ dargestellt, spricht man von *Backus-Naur-Form* oder kurz von BNF-Notation. Diese Notation wurde 1960 von John Backus und Peter Naur für die Definition der Programmiersprache ALGOL 60 eingeführt. Weitere Abkürzungsmöglichkeiten bietet die Erweiterte BNF-Notation, auch EBNF genannt. Die EBNF-Notation lässt sich 1–1 in die 1970 von Niklaus Wirth für die Definition der Programmiersprache PASCAL eingeführten Syntaxdiagramme übersetzen.

Zu jeder kontextfreien Grammatik G gehört die zweistellige *Ableitungsrelation* $\vdash_G$ auf $(N \cup T)^*$:

$$x \vdash_G y \quad \text{gdw} \quad \exists A \to u \in P \; \exists w_1, w_2 \in (N \cup T)^* : \\ x = w_1 \boxed{A} w_2 \text{ und } y = w_1 \boxed{u} w_2.$$

Mit $\vdash_G^*$ wird wieder die reflexive und transitive Hülle von $\vdash_G$ bezeichnet. Wir lesen $x \vdash^* y$ als „y ist aus x ableitbar“. Die von G *erzeugte Sprache* ist

$$L(G) = \{w \in T^* \mid S \vdash_G^* w\}.$$

Zwei kontextfreie Grammatiken G_1 und G_2 heißen *äquivalent* , falls $L(G_1) = L(G_2)$ ist.

1.2 **Definition** : Eine *Sprache* $L \subseteq T^*$ heißt *kontextfrei*, falls es eine kontextfreie Grammatik G mit $L = L(G)$ gibt.

Beispiel :

(1) Die Sprache $L = \{a^n b^n \mid n \in \mathbb{N}\}$ wird durch die Grammatik $G_1 = (\{S\}, \{a, b\}, P_1, S)$ erzeugt, wobei P_1 wie folgt gegeben ist:

$$S \to \varepsilon \mid aSb.$$

Zum Beispiel gilt $a^2b^2 \in L(G_1)$, weil

$$S \vdash_{G_1} aSb \vdash_{G_1} aaSbb \vdash_{G_1} aabb.$$

(2) Die arithmetischen Ausdrücke mit Variablen a, b, c und Operatoren $+$ und $*$ werden durch die Grammatik $G_2 = (\{S\}, \{a, b, c, +, *, (,)\}, P_2, S)$ mit folgendem P_2 erzeugt:

$$S \to a \mid b \mid c \mid S + S \mid S * S \mid (S).$$

Zum Beispiel ist $(a + b) * c \in L(G_2)$, weil

$$S \vdash_{G_2} S * S \vdash_{G_2} (S) * S \vdash_{G_2} (S + S) * S \\ \vdash_{G_2} (a + S) * S \vdash_{G_2} (a + b) * S \vdash_{G_2} (a + b) * c.$$

Wir untersuchen jetzt die Ableitung von Wörtern in einer kontextfreien Grammatik genauer.

1.3 **Definition** : Eine *Ableitung* von A nach w in G der Länge $n \geq 0$ ist eine Folge von Ableitungsschritten

$$A = z_0 \vdash_G z_1 \vdash_G \cdots \vdash_G z_n = w. \qquad (**)$$

Diese Ableitung heißt *Linksableitung*, falls in jedem Ableitungsschritt $z_i \vdash_G z_{i+1}$ das am weitesten links stehende Nichtterminalsymbol ersetzt wird, falls also z_i und z_{i+1} stets von der Form

$$z_i = w_1 A w_2 \text{ und } z_{i+1} = w_1 u w_2 \text{ mit } w_1 \in T^*$$

sind. Entsprechend sind *Rechtsableitungen* definiert (dann gilt: $w_2 \in T^*$).

Jede Ableitung lässt sich graphisch als Baum darstellen.

1.4 **Definition** : Ein *Ableitungsbaum* von A nach w in G ist ein Baum mit folgenden Eigenschaften:

(i) Jeder Knoten ist mit einem Symbol aus $N \cup T \cup \{\varepsilon\}$ beschriftet. Die Wurzel ist mit A beschriftet und jeder innere Knoten ist mit einem Symbol aus N beschriftet.

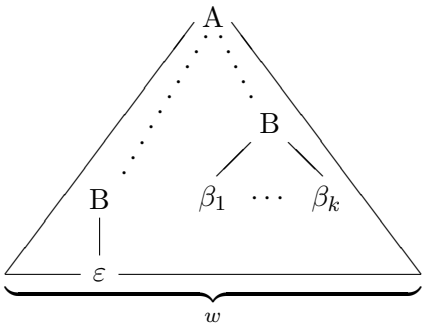
(ii) Wenn ein mit B beschrifteter innerer Knoten k Nachfolgeknoten besitzt, die in der Reihenfolge von links nach rechts mit den Symbolen $\beta_1, \ldots, \beta_k$ beschriftet sind, dann gilt

a) $k = 1$ und $\beta_1 = \varepsilon$ und $B \to \varepsilon \in P$
oder

b) $k \geq 1$ und $\beta_1, \ldots, \beta_k \in N \cup T$ und $B \to \beta_1 \ldots \beta_k \in P$.

(iii) Das Wort w entsteht, indem man die Symbole an den Blättern von links nach rechts konkateniert.

Veranschaulichung



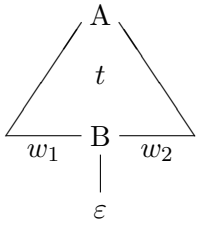
Den zu einer Ableitung von A nach w der Form $(**)$ gehörigen Ableitungsbaum von A nach w konstruieren wir mit Induktion nach der Länge n der Ableitung.

$n = 0$: Zur trivialen Ableitung A gehört der triviale Ableitungsbaum A.

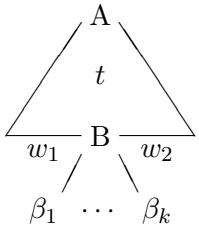
$n \to n + 1$: Betrachte eine Ableitung

$$A = z_0 \vdash_G \ldots \vdash_G z_n = w_1 B w_2 \vdash_G w_1 u w_2 = z_{n+1}.$$

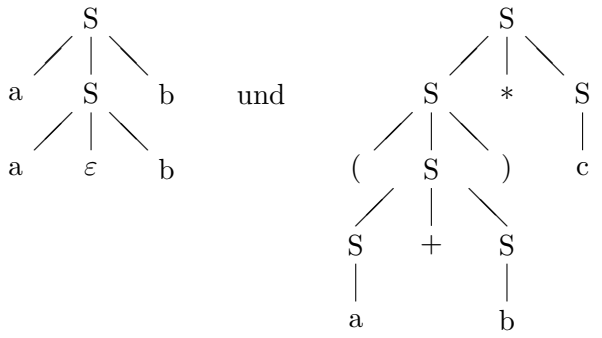
Sei t der zur Ableitung $A = z_0 \vdash_G \ldots \vdash_G z_n$ gehörige Ableitungsbaum.
Falls $u = \varepsilon$, so ist der Gesamtableitungsbaum wie folgt:



Falls $u = \beta_1 \ldots \beta_k$ mit $\beta_1, \ldots, \beta_k \in N \cup T$, so ist der Gesamtableitungsbaum wie folgt:



Beispiel : Ableitungsbäume zu den im vorangegangenen Beispiel genannten Ableitungen sind



Bemerkung : Es gelten folgende Beziehungen zwischen Ableitungen und Ableitungsbäumen:

(i) $A \vdash_G^* w \quad \Leftrightarrow \quad$ Es gibt einen Ableitungsbaum von A nach w in G.

(ii) Einem gegebenen Ableitungsbaum von A nach w entsprechen im allgemeinen mehrere Ableitungen von A nach w, jedoch nur genau eine Linksableitung und genau eine Rechtsableitung.

Beweis :

(i) „$\Rightarrow$“ ist nach obiger Konstruktion klar.
„$\Leftarrow$“ zeigt man mit Induktion über die Tiefe des Ableitungsbaumes.

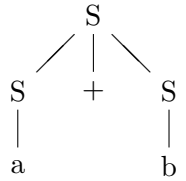
(ii) Ableitungsbäume abstrahieren von unwesentlichen Reihenfolgen bei der Regelanwendung, die möglich werden, wenn mehrere Nichtterminalsymbole gleichzeitig auftreten. Zum Beispiel ergeben die beiden Ableitungen

$$S \vdash_{G_2} S + S \vdash_{G_2} a + S \vdash_{G_2} a + b$$

und

$$S \vdash_{G_2} S + S \vdash_{G_2} S + b \vdash_{G_2} a + b$$

denselben Ableitungsbaum:



Bei Festlegung auf Linksableitungen bzw. Rechtsableitungen sind solche Wahlmöglichkeiten ausgeschlossen.

□

Sei jetzt die Syntax einer Programmiersprache PL durch eine kontextfreie Grammatik G gegeben. Ein Übersetzer für PL erstellt in der Phase der Syntaxanalyse für jedes gegebene PL-Programm einen Ableitungsbaum in G. Die Bedeutung oder Semantik des PL-Programms hängt von der Struktur des erstellten Ableitungsbaumes ab. Der Übersetzer erzeugt nämlich den Maschinencode des PL-Programms anhand des Ableitungsbaumes.

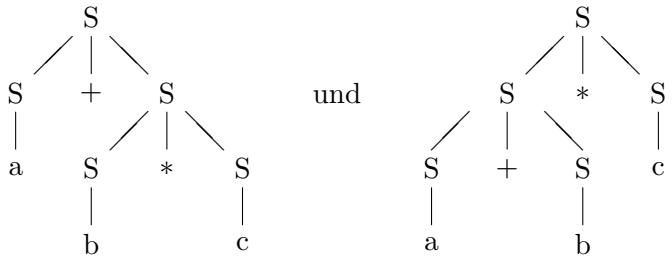
Für die Benutzung der Programmiersprache PL ist es wichtig, dass jedes PL-Programm eine eindeutige Semantik hat. Daher sollte es zu jedem PL-Programm genau einen Ableitungsbaum geben.

1.5 **Definition** :

(i) Eine kontextfreie *Grammatik* $G = (N, T, P, S)$ heißt *eindeutig*, wenn es zu jedem Wort $w \in T^*$ höchstens einen Ableitungsbaum bzw. höchstens eine Linksableitung von S nach w in G gibt. Andernfalls heißt G *mehrdeutig*.

(ii) Eine kontextfreie *Sprache* $L \subseteq T^*$ heißt *eindeutig*, wenn es eine eindeutige kontextfreie Grammatik G mit $L = L(G)$ gibt. Andernfalls heißt L *(inhärent) mehrdeutig*.

Beispiel : Die oben angegebene Grammatik G_2 für arithmetische Ausdrücke ist mehrdeutig. Für das Wort $a + b * c \in L(G_2)$ gibt es nämlich die folgenden beiden Ableitungsbäume:



Diese entsprechen den beiden semantisch verschiedenen Klammerungen $a+(b*c)$ und $(a+b)*c$.

In Programmiersprachen wählt man daher eine Grammatik G_3 für arithmetische Ausdrücke, in der die folgende Auswertungsstrategie festgelegt ist:

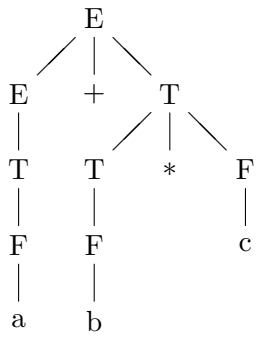
- Die Auswertung erfolgt von links nach rechts. Damit wird z.B. $a + b + c$ wie $(a + b) + c$ ausgewertet.
- Multiplikation $*$ erhält eine höhere Priorität als $+$. Damit wird z.B. $a + b * c$ wie $a + (b * c)$ ausgewertet.

Möchte man andere Auswertungsreihenfolgen haben, müssen explizit Klammern (und) gesetzt werden.

Betrachte $G_3 = (\{E, T, F\}, \{a, b, c, +, *, (,)\}, P_3, E)$ mit folgendem P_3:

$$\begin{aligned} E &\to T \mid E + T \\ T &\to F \mid T * F \\ F &\to (E) \mid a \mid b \mid c \end{aligned}$$

G_3 ist eindeutig, und es gilt $L(G_3) = L(G_2)$. Zum Beispiel hat $a+b*c$ in G_3 den Ableitungsbaum



Beispiel : (Chomsky, 1964) Eine inhärent mehrdeutige kontextfreie Sprache ist

$$L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ und } (i = j \text{ oder } j = k)\}$$

Zum Beweis siehe Literatur oder Vorlesung „Formale Sprachen“.

§2 Pumping Lemma

Auch für kontextfreie Sprachen gibt es ein Pumping Lemma. Es beschreibt eine notwendige Bedingung dafür, dass eine gegebene Sprache kontextfrei ist.

2.1 **Satz (Pumping Lemma für kontextfreie Sprachen oder $uvwxy$-Lemma)**: Zu jeder kontextfreien Sprache $L \subseteq T^*$ existiert eine Zahl $n \in \mathbb{N}$, so dass es für alle Wörter $z \in L$ mit $|z| \geq n$ eine Zerlegung $z = uvwxy$ mit folgenden Eigenschaften gibt:

(i) $vx \neq \varepsilon$,

(ii) $|vwx| \leq n$,

(iii) für alle $i \in \mathbb{N}$ gilt: $uv^iwx^iy \in L$.

Die Teilwörter v und x können also beliebig oft „aufgepumpt“ werden, ohne dass man aus der kontextfreien Sprache L herausfällt.

Zum Beweis des Pumping Lemmas benötigen wir folgendes allgemeine Lemma über Bäume, das wir dann speziell für Ableitungsbäume anwenden werden. Wir definieren für endliche Bäume t:

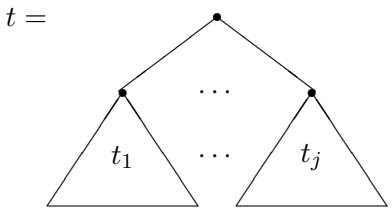
- Verzweigungsgrad von t = maximale Anzahl von Nachfolgeknoten, die ein Knoten in t besitzt.
- Ein Pfad der Länge m in t ist eine Kantenfolge von der Wurzel bis zu einem Blatt von t mit m Kanten. Der Trivialfall $m = 0$ ist zugelassen.

2.2 **Lemma** : Sei t ein endlicher Baum mit dem Verzweigungsgrad $\leq k$, in dem jeder Pfad die Länge $\leq m$ hat. Dann ist in t die Anzahl der Blätter $\leq k^m$.

Beweis : Induktion über $m \in \mathbb{N}$:

$m = 0$**:** t besteht nur aus $k^0 = 1$ Knoten.

$m \to m + 1$**:** t besitzt j Unterbäume $t_1, \ldots, t_j$ mit $j \leq k$, in denen die Pfade die Länge $\leq m$ haben:



Nach Induktionsvoraussetzung ist für jeden der Unterbäume $t_1, \ldots, t_j$ die Anzahl der Blätter $\leq k^m$. Damit gilt in t:

$$\text{Anzahl der Blätter } \leq j \cdot k^m \leq k \cdot k^m = k^{m+1}.$$

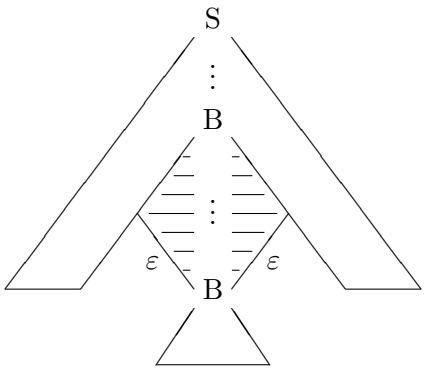
□

Wir sind jetzt vorbereitet für den

Beweis des Pumping Lemmas: Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik mit $L(G) = L$. Wir setzen:

- $k =$ Wortlänge der längsten rechten Seite einer Produktion aus P, wenigstens jedoch 2
- $m = |N|$,
- $n = k^{m+1}$.

Sei jetzt $z \in L$ mit $|z| \geq n$. Dann gibt es einen Ableitungsbaum t von S nach z in G, in dem es kein Teilstück gibt, das einer Ableitung der Form $B \vdash_G^* B$ entspricht:

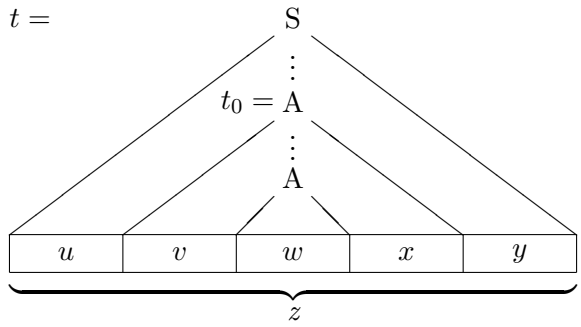


Jedes solche Teilstück könnte nämlich aus t entfernt werden, ohne dass das abgeleitete Wort z verändert würde.

Nach der Wahl von k und $|z|$ hat t einen Verzweigungsgrad $\leq k$ und $\geq k^{m+1}$ Blätter. Also gibt es nach dem vorangegangenen Lemma in t einen Pfad der Länge $\geq m + 1$. Auf diesem Pfad liegen $\geq m + 1$ innere Knoten, so dass es eine Wiederholung eines Nichtterminalsymbols bei der Beschriftung dieser Knoten gibt. Wir benötigen diese Wiederholung in einer speziellen Lage.

Unter einem *Wiederholungsbaum* in t verstehen wir einen Unterbaum von t, in dem sich die Beschriftung der Wurzel bei einem weiteren Knoten wiederholt. Wir wählen jetzt in t einen minimalen Wiederholungsbaum t_0, d.h. einen solchen, der keinen weiteren Wiederholungsbaum als echten Unterbaum enthält. In t_0 hat jeder Pfad eine Länge $\leq m + 1$.

Sei A die Wurzelbeschriftung von t_0. Dann hat t folgende Struktur:



Aus dieser Struktur erhalten wir eine Zerlegung $z = uvwxy$ mit

$$S \vdash_G^* uAy \vdash_G^* uvAxy \vdash_G^* uvwxy. \qquad (*)$$

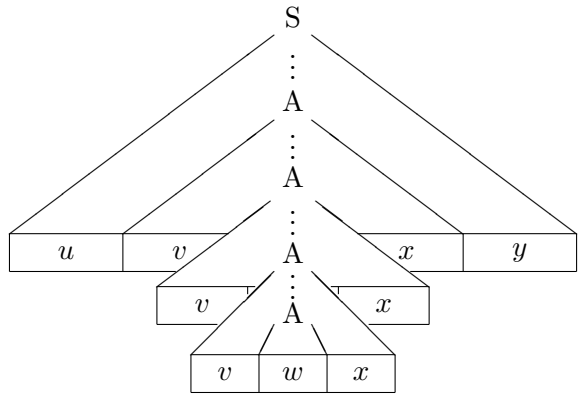
Wir zeigen, dass diese Zerlegung von z den Bedingungen des Pumping Lemmas genügt:

(i) Nach der Wahl von t gilt $vx \neq \varepsilon$.

(ii) Nach der Wahl von t_0 und dem vorangegangenen Lemma gilt $|vwx| \leq k^{m+1} = n$.

(iii) Aus $(*)$ folgt sofort, dass für alle $i \in \mathbb{N}$ gilt: $uv^iwx^iy \in L(G)$.

Für $i = 3$ sieht der Ableitungsbaum von S nach uv^iwx^iy in G so aus:



□

Wie bei regulären Sprachen kann das obige Pumping Lemma zum Nachweis benutzt werden, dass eine bestimmte Sprache *nicht* kontextfrei ist.

Beispiel : Die Sprache $L = \{a^nb^nc^n \mid n \in \mathbb{N}\}$ ist nicht kontextfrei. Wir geben einen Widerspruchsbeweis.

Annahme: L ist kontextfrei. Dann gibt es nach dem Pumping Lemma ein $n \in \mathbb{N}$ mit den dort genannten Eigenschaften. Betrachte jetzt $z = a^nb^nc^n$. Da $|z| \geq n$ gilt, lässt sich z zerlegen in $z = uvwxy$ mit $vx \neq \varepsilon$ und $|vwx| \leq n$, so dass für alle $i \in \mathbb{N}$ gilt: $uv^iwx^iy \in L$. Wegen $|vwx| \leq n$ kommen im Teilwort vwx keine a's oder keine c's vor. Deshalb werden beim Aufpumpen zu

uv^iwx^iy höchstens *zwei* der Buchstaben a, b, c berücksichtigt. Solche Wörter liegen aber nicht in L. *Widerspruch.* □

Mit dem Pumping Lemma lässt sich auch zeigen, dass kontextfreie Grammatiken nicht ausreichen, um die Syntax von höheren Programmiersprachen wie Java *vollständig* zu beschreiben. Obwohl die Grundstruktur der syntaktisch korrekten Programme mit kontextfreien Grammatiken (in BNF- oder EBNF-Notation) beschrieben wird, gibt es Nebenbedingungen, die kontextsensitiv sind.

Beispiel : Die Programmiersprache Java, bestehend aus allen syntaktisch korrekten Java-Programmen, ist nicht kontextfrei. Wir führen einen Widerspruchsbeweis.

Annahme: Java sei kontextfrei. Dann gibt es ein $n \in \mathbb{N}$ mit den im Pumping Lemma genannten Eigenschaften. Betrachte nun folgende syntaktisch korrekte Java-Klasse:

```
class C {
  int X 1...1;        (n)
  void m() {
    X 1...1 = X 1...1   (n)  (n)
  }
}
```

Bei jeder $uvwxy$-Zerlegung dieses Programms berührt das $uvwx$-Mittelstück wegen $|vwx| \leq n$ maximal zwei der drei Vorkommen der Variablen $X\underbrace{1\ldots1}_{n}$. Deshalb entstehen beim Aufpumpen zu uv^iwx^iy entweder Zeichenreihen, die nicht einmal den Anforderungen der Java-Syntaxdiagramme genügen, oder aber Zeichenreihen der Form

```
class C {
  int X 1...1;        (k)
  void m() {
    X 1...1 = X 1...1   (l)  (m)
  }
}
```

wobei die k, l, m nicht alle gleich sind. Diese Zeichenreihen verletzen die folgende Bedingung für syntaktisch korrekte Java-Programme:

„Jede Variable muss vor ihrem Gebrauch deklariert sein." (∗∗)

Hier ist entweder $X\underbrace{1\ldots1}_{l}$ oder $X\underbrace{1\ldots1}_{m}$ nicht deklariert.

Widerspruch □

Bedingungen wie (∗∗) sind kontextsensitiv und werden daher bei der Definition von Programmiersprachen *zusätzlich* zur kontextfreien Grundstruktur der Programme genannt. Ein Übersetzer überprüft diese kontextsensitiven Bedingungen durch Anlegen geeigneter Tabellen zum Abspeichern der deklarierten Variablen oder allgemeiner Identifikatoren.

§3 Kellerautomaten

Bisher haben wir kontextfreie Sprachen über die Erzeugbarkeit von Grammatiken definiert. Wir wollen jetzt die *Erkennbarkeit* durch Automaten untersuchen. Unser Ziel ist eine Erweiterung des Modells des endlichen Automaten, das genau die kontextfreien Sprachen erkennen kann.

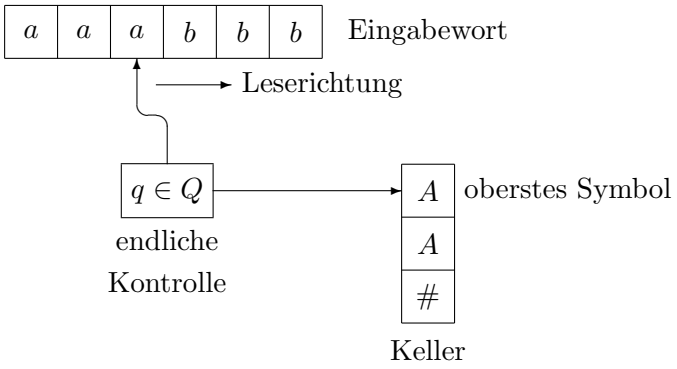
Die Schwäche der endlichen Automaten ist das Fehlen eines Speichers für unbeschränkt große Informationen. Intuitiv kann ein endlicher Automat eine Sprache wie

$$L = \{a^n b^n \mid n \in \mathbb{N}\}$$

deshalb nicht erkennen, weil er zum Zeitpunkt, wenn das erste b gelesen wird, nicht mehr wissen kann, wieviele a's eingelesen wurden. Die einzige gespeicherte Information ist der jeweils aktuelle Zustand, der Element einer endlichen Zustandsmenge ist.

Wir betrachten jetzt sogenannte *Kellerautomaten* oder *Pushdown-Automaten*. Das sind nichtdeterministische endliche Automaten mit ε- Übergängen (ε-NEA's), erweitert um einen Speicher, der unbeschränkt große Wörter aufnehmen kann, aber auf den nur sehr eingeschränkt zugegriffen werden kann. Der Speicher ist als *Keller* oder *Stack* oder *Pushdown-Liste* organisiert, bei dem nur auf das jeweils oberste Symbol zugegriffen werden kann. Die Transitionen eines Kellerautomaten hängen vom aktuellen Zustand, dem gelesenen Symbol des Eingabewortes und dem obersten Symbol des Kellers ab; sie verändern den Zustand und den Inhalt des Kellers.

Skizze



3.1 **Definition (Kellerautomat)**: Ein (*nichtdeterministischer*) *Kellerautomat* (oder *Pushdown-Automat*), kurz KA (oder auch PDA), ist eine Struktur

$$\mathcal{K} = (\Sigma, Q, \Gamma, \to, q_0, Z_0, F)$$

mit folgenden Eigenschaften:

(i) Σ ist das *Eingabealphabet*,

(ii) Q ist eine endliche Menge von *Zuständen*,

(iii) Γ ist das *Kelleralphabet*,

(iv) $\rightarrow \subseteq\ Q \times \Gamma \times (\Sigma \cup \{\varepsilon\}) \times Q \times \Gamma^*$ ist die *Transitionsrelation*,

(v) $q_0 \in Q$ ist der *Anfangszustand*,

(vi) $Z_0 \in \Gamma$ ist das *Startsymbol des Kellers*,

(vii) $F \subseteq Q$ ist die Menge der *Endzustände*.

Im Zusammenhang mit KA's benutzen wir folgende typische Buchstaben: $a, b, c \in \Sigma$, $u, v, w \in \Sigma^*$, $\alpha \in \Sigma \cup \{\varepsilon\}$, $q \in Q$, $Z \in \Gamma$, $\gamma \in \Gamma^*$. Diese Buchstaben können auch mit Indizes und Strichen „verziert" sein. Die Elemente $(q, Z, \alpha, q', \gamma') \in \rightarrow$ heißen *Transitionen*. Statt $(q, Z, \alpha, q', \gamma') \in \rightarrow$ schreiben wir meistens $(q, Z) \xrightarrow{\alpha} (q', \gamma')$.

Wir haben dabei folgende anschauliche Vorstellung. Eine Transition $(q, Z) \xrightarrow{a} (q', \gamma')$ besagt: Ist q der aktuelle Zustand und Z das oberste Kellersymbol, so kann der KA das Eingabesymbol a lesen, in den Zustand q' übergehen und an der Kellerspitze das Symbol Z durch das Wort γ' ersetzen. Analog besagt eine Transition $(q, Z) \xrightarrow{\varepsilon} (q', \gamma')$: Im Zustand q und Z als oberstem Kellersymbol kann der KA selbständig in den Zustand q' übergehen und an der Kellerspitze Z durch γ' ersetzen. Es wird dabei kein Eingabesymbol gelesen.

Ist $\gamma' = \varepsilon$, so spricht man bei einer Transition $(q, Z) \xrightarrow{\alpha} (q', \varepsilon)$ von einem *pop - Schritt*, da das oberste Kellersymbol entfernt wird. Ist $\gamma' = \gamma Z$, so spricht man bei einer Transition $(q, Z) \xrightarrow{\alpha} (q', \gamma Z)$ von einem *push - Schritt*, da das Wort γ an der Kellerspitze hinzugefügt wird.

Um das Akzeptanz - Verhalten von KA's zu definieren, müssen wir – ähnlich wie bei ε-NEA's – zuerst die Transitionsrelation erweitern. Dazu benötigen wir den Begriff der Konfiguration eines KA's.

3.2 **Definition** : Sei $\mathcal{K} = (\Sigma, Q, \Gamma, \rightarrow, q_0, Z_0, F)$ ein KA.

(i) Unter einer *Konfiguration* von $\mathcal{K}$ verstehen wir ein Paar $(q, \gamma) \in Q \times \Gamma^*$, das den momentanen Zustand q und den momentanen Kellerinhalt γ von $\mathcal{K}$ bescheibt.

(ii) Für jedes $\alpha \in \Sigma \cup \{\varepsilon\}$ ist $\xrightarrow{\alpha}$ eine 2-stellige Relation auf den Konfigurationen von $\mathcal{K}$, die wie folgt definiert ist:

$$(q, \gamma) \xrightarrow{\alpha} (q', \gamma'), \text{ falls } \exists Z \in \Gamma, \quad \exists \gamma_0, \gamma_1 \in \Gamma^* :$$

$$\gamma = Z\gamma_0 \text{ und } (q, Z, \alpha, q', \gamma_1) \in \rightarrow \text{ und } \gamma' = \gamma_1\gamma_0$$

Wir nennen $\xrightarrow{\alpha}$ die α-*Transitionsrelation* auf den Konfigurationen.

(iii) Für jedes Wort $w \in \Sigma^*$ ist $\overset{w}{\Rightarrow}$ eine 2-stellige Relation auf den Konfigurationen von $\mathcal{K}$, die induktiv definiert ist:

- $(q, \gamma) \overset{\varepsilon}{\Rightarrow} (q', \gamma')$, falls $\exists n \geq 0 : (q, \gamma) \underbrace{\xrightarrow{\varepsilon} \circ \ldots \circ \xrightarrow{\varepsilon}}_{n-\text{mal}} (q', \gamma')$

- $(q,\gamma) \overset{aw}{\Rightarrow} (q',\gamma')$, falls $(q,\gamma) \overset{\varepsilon}{\Rightarrow} \circ \overset{a}{\to} \circ \overset{w}{\Rightarrow} (q',\gamma')$, für alle $a \in \Sigma$.

Wir nennen $\overset{w}{\Rightarrow}$ die *erweiterte w - Transitionsrelation* auf den Konfigurationen.

Bemerkung : Für alle $\alpha \in \Sigma \cup \{\varepsilon\}, a_1, \ldots, a_n \in \Sigma$ und $w_1, w_2 \in \Sigma^*$ gilt:

(i) $(q,\gamma) \overset{\alpha}{\to} (q',\gamma')$ impliziert $(q,\gamma) \overset{\alpha}{\Rightarrow} (q',\gamma')$.

(ii) $(q,\gamma) \overset{a_1 \ldots a_n}{\Longrightarrow} (q',\gamma')$ gdw.$(q,\gamma) \overset{\varepsilon}{\Rightarrow} \circ \overset{a_1}{\to} \circ \overset{\varepsilon}{\Rightarrow} \ldots \overset{\varepsilon}{\Rightarrow} \circ \overset{a_n}{\to} \circ \overset{\varepsilon}{\Rightarrow} (q',\gamma')$
gdw. $(q,\gamma) \overset{a_1}{\Rightarrow} \circ \ldots \circ \overset{a_n}{\Rightarrow} (q',\gamma')$

(iii) $(q,\gamma) \overset{w_1 w_2}{\Longrightarrow} (q',\gamma')$ gdw. $(q,\gamma) \overset{w_1}{\Rightarrow} \circ \overset{w_2}{\Rightarrow} (q',\gamma')$

Für KA's gibt es zwei Varianten von Sprach-Akzeptanz, die wir jetzt definieren können.

3.3 **Definition (Akzeptanz)**:
Sei $\mathcal{K} = (\Sigma, Q, \Gamma, \to, q_0, Z_0, F)$ ein KA und $w \in \Sigma^*$.

(i) $\mathcal{K}$ *akzeptiert* w, falls $\exists q \in F \quad \exists \gamma \in \Gamma^* : \quad (q_0, Z_0) \overset{w}{\Longrightarrow} (q,\gamma)$.
Die von $\mathcal{K}$ *akzeptierte* (oder *erkannte*) *Sprache* ist

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

(ii) $\mathcal{K}$ *akzeptiert* w *mit dem leeren Keller*,

falls

$$\exists q \in Q : \quad (q_0, Z_0) \overset{w}{\Longrightarrow} (q,\varepsilon).$$

Die von $\mathcal{K}$ mit leerem Keller akzeptierte (oder erkannte) Sprache ist

$$L_\varepsilon(\mathcal{K}) = \{w \in \Sigma^* \mid \mathcal{K} \text{ akzeptiert } w \text{ mit leerem Keller }\}.$$

Beispiel : Wir konstruieren einen Kellerautomaten, der die bekannte Sprache $L = \{a^n b^n \mid n \in \mathbb{N}\}$ akzeptiert. Setze

$$\mathcal{K} = (\{a,b\}, \{q_0,q_1,q_2\}, \{a,Z\}, \to, q_0, Z, \{q_0\}),$$

wobei die Transitionsrelation $\to$ aus den folgenden Transitionen besteht:

(1) $(q_0, Z) \overset{a}{\to} (q_1, aZ)$
(2) $(q_1, a) \overset{a}{\to} (q_1, aa)$
(3) $(q_1, a) \overset{b}{\to} (q_2, \varepsilon)$
(4) $(q_2, a) \overset{b}{\to} (q_2, \varepsilon)$
(5) $(q_2, Z) \overset{\varepsilon}{\to} (q_0, \varepsilon)$

$\mathcal{K}$ akzeptiert $a^n b^n$ für $n \geq 1$ durch folgende $2n+1$ Transitionen, die wir der Deutlichkeit halber mit ihren Nummern 1 bis 5 indiziert haben:

$$\begin{aligned}(q_0, Z) &\overset{a}{\to}_1 (q_1, aZ) \overset{a}{\to}_2 (q_1, aaZ) \ldots \overset{a}{\to}_2 (q_1, a^n Z) \\ &\overset{b}{\to}_3 (q_2, a^{n-1} Z) \overset{b}{\to}_4 (q_2, a^{n-2} Z) \ldots \overset{b}{\to}_4 (q_2, Z) \\ &\overset{\varepsilon}{\to}_5 (q_0, \varepsilon)\end{aligned}$$

Damit gilt für $n \geq 1$ die Beziehung $(q_0, Z) \stackrel{a^n b^n}{\Longrightarrow} (q_0, \varepsilon)$. Für $n = 0$ gilt trivialerweise $(q_0, Z) \stackrel{\varepsilon}{\Rightarrow} (q_0, Z)$. Da q_0 Endzustand von $\mathcal{K}$ ist, folgt $L \subseteq L(\mathcal{K})$.

Für die Inklusion $L(\mathcal{K}) \subseteq L$ müssen wir das Transitions - Verhalten von $\mathcal{K}$ analysieren. Dazu indizieren wir die Transitionen wie eben. Wir stellen fest, dass $\mathcal{K}$ *deterministisch* ist, d.h. beim buchstabenweisen Lesen eines gegebenen Eingabewortes ist jeweils genau eine Transition anwendbar. Und zwar sind genau die folgenden Transitionssequenzen möglich, wobei $n \geq m \geq 0$ gilt:

- $(q_0, Z) \stackrel{a}{\rightarrow}_1 \circ \left(\stackrel{a}{\rightarrow}_2\right)^n (q_1, a^{n+1}Z)$
- $(q_0, Z) \stackrel{a}{\rightarrow}_1 \circ \left(\stackrel{a}{\rightarrow}_2\right)^n \circ \stackrel{b}{\rightarrow}_3 \circ \left(\stackrel{b}{\rightarrow}_4\right)^m (q_2, a^{n-m}Z)$
- $(q_0, Z) \stackrel{a}{\rightarrow}_1 \circ \left(\stackrel{a}{\rightarrow}_2\right)^n \circ \stackrel{b}{\rightarrow}_3 \circ \left(\stackrel{b}{\rightarrow}_4\right)^n \circ \stackrel{\varepsilon}{\rightarrow}_5 (q_0, \varepsilon)$

Daher akzeptiert $\mathcal{K}$ nur Wörter der Form $a^n b^n$. Insgesamt gilt also $L(\mathcal{K}) = L$.

Wir bemerken ferner, dass bis auf den Fall $n = 0$ der Automat $\mathcal{K}$ alle Wörter $a^n b^n$ mit dem leeren Keller akzeptiert: $L_\varepsilon(\mathcal{K}) = \{a^n b^n \mid n \geq 1\}$.

Wir wollen jetzt allgemeine Eigenschaften von Kellerautomaten herleiten. Dazu benötigen wir häufig das folgende Top – Lemma. Es besagt anschaulich, dass Veränderungen an der Spitze (dem Top) des Kellers unabhängig vom tieferen Inhalt des Kellers sind.

3.4 **Lemma (Top des Kellers)**:Sei $\mathcal{K} = (\Sigma, Q, \Gamma, \rightarrow, q_0, Z_0, F)$ ein Kellerautomat. Dann gilt für alle $w \in \Sigma^*$, $q, q' \in Q$, $Z \in \Gamma$ und $\gamma \in \Gamma^*$: wenn

$$(q, Z) \stackrel{w}{\Longrightarrow} (q', \varepsilon),$$

so auch

$$(q, Z\gamma) \stackrel{w}{\Longrightarrow} (q', \gamma).$$

Beweis : Übungsaufgabe □

Wir zeigen jetzt, dass die beiden Varianten von Sprach-Akzeptanz äquivalent sind.

3.5 **Satz (Akzeptanz)**:

(1) Zu jedem KA $\mathcal{A}$ kann ein KA B mit $L(\mathcal{A}) = L_\varepsilon(B)$ konstruiert werden.

(2) Zu jedem KA $\mathcal{A}$ kann ein KA B mit $L_\varepsilon(\mathcal{A}) = L(B)$ konstruiert werden.

Beweis : Sei $\mathcal{A} = (\Sigma, Q, \Gamma, \rightarrow_{\mathcal{A}}, q_0, Z_0, F)$.
Zu (1): Die Beweisidee ist einfach: B arbeitet wie $\mathcal{A}$ und leert von Endzuständen aus den

Keller. Es muss jedoch darauf geachtet werden, dass B keinen leeren Keller erhält durch Eingabewörter, die $\mathcal{A}$ nicht akzeptiert. Deshalb benutzt B ein zusätzliches Symbol # zur Markierung des Kellerbodens. Genauer konstruieren wir:

$$B = (\Sigma, Q \cup \{q_B, q_\varepsilon\}, \Gamma \cup \{\#\}, \to_B, q_B, \#, \emptyset)$$

mit $q_B, q_\varepsilon \notin Q$ und $\# \notin \Gamma$ und folgender Transitionsrelation:

$$\begin{array}{lll} \to_B & = \{(q_B, \#, \varepsilon, q_0, Z_0\#)\} & \text{„Starten von } \mathcal{A}\text{“} \\ & \cup \to_{\mathcal{A}} & \text{„Arbeiten wie } \mathcal{A}\text{“} \\ & \cup \{(q, Z, \varepsilon, q_\varepsilon, \varepsilon) \mid q \in F, Z \in \Gamma \cup \{\#\}\} \\ & \cup \{(q_\varepsilon, Z, \varepsilon, q_\varepsilon, \varepsilon) \mid Z \in \Gamma \cup \{\#\}\} \end{array} \Big\} \quad \text{„Leeren des Kellers“}$$

Dann gilt für alle $w \in \Sigma^*, q \in F$ und $\gamma \in \Gamma^*$:

$$(q_0, Z_0) \overset{w}{\Longrightarrow}_{\mathcal{A}} (q, \gamma)$$

gdw.

$$(q_B, \#) \overset{\varepsilon}{\to}_B (q_0, Z_0\#) \overset{w}{\Longrightarrow}_{\mathcal{A}} (q, \gamma\#) \overset{\varepsilon}{\to}_B (q_\varepsilon, \varepsilon).$$

(Für die „wenn-dann“ - Richtung wird das Top - Lemma angewandt.) Mit einer Analyse der Anwendbarkeit der neuen Transitionen in B erhält man daraus $L(\mathcal{A}) = L_\varepsilon(B)$.

Zu (2): Beweisidee: B arbeitet wie $\mathcal{A}$, benutzt aber ein zusätzliches Symbol # zur Markierung des Kellerbodens. Sobald $\mathcal{A}$ seinen Keller geleert hat, liest B das Symbol # und geht in einen Endzustand über. Die genaue Konstruktion von B ist eine Übungsaufgabe. □

Wir wollen jetzt zeigen, dass die nichtdeterministischen Kellerautomaten genau die kontextfreien Sprachen (mit leerem Keller) akzeptieren. Zunächst konstruieren wir zu einer gegebenen kontextfreien Grammatik G einen Kellerautomaten, der einen nichtdeterministischen „top-down“ Parser der Sprache $L(G)$ darstellt.

3.6 **Satz** : Zu jeder kontextfreien Grammatik G kann man einen nichtdeterministischen Kellerautomaten $\mathcal{K}$ mit $L_\varepsilon(\mathcal{K}) = L(G)$ konstruieren.

Beweis : Sei $G = (N, T, P, S)$. Wir konstruieren $\mathcal{K}$ so, dass er die *Linksableitungen* in G simuliert:

$$\mathcal{K} = (T, \{q\}, N \cup T, \to, q, S, \emptyset),$$

wobei die Transitionsrelation $\to$ aus den folgenden Transitionstypen besteht:

(1) $(q, A) \overset{\varepsilon}{\to} (q, u)$, falls $A \to u \in P$,
(2) $(q, a) \overset{a}{\to} (q, \varepsilon)$, falls $a \in T$.

Die Arbeitsweise von $\mathcal{K}$ ist so: Zunächst steht S im Keller. Eine Regelanwendung $A \to u$ der Grammatik wird im Keller nachvollzogen, indem das oberste Kellersymbol A durch u ersetzt wird. Stehen dann Terminalsymbole an der Kellerspitze, so werden sie mit den Symbolen des Eingabewortes verglichen und bei Übereinstimmung aus dem Keller entfernt. Auf diese Weise wird

schrittweise eine Linksableitung des Eingabewortes durch $\mathcal{K}$ hergestellt. Gelingt diese Ableitung, so akzeptiert $\mathcal{K}$ das Eingabewort mit dem leeren Keller. Die Anwendung der Transitionen vom Typ (1) ist *nichtdeterministisch*, wenn es mehrere Regeln mit demselben Nichtterminalsymbol A in P gibt. Der einzige Zustand q spielt beim Transitionsverhalten von $\mathcal{K}$ keine Rolle, muss aber der Vollständigkeit halber bei der Definition von $\mathcal{K}$ angegeben werden.

Um zu zeigen, dass $L(G) = L_\varepsilon(\mathcal{K})$ gilt, untersuchen wir den Zusammenhang zwischen Linksableitungen in G und Transitionsfolgen in $\mathcal{K}$ genauer. Dabei benutzen wir folgende Abkürzungen für Wörter $w \in (N \cup T)^*$:

- w_T = längstes Präfix von w mit $w_T \in T^*$,
- w_R ist der Rest von w, definiert durch $w = w_T w_R$.

□

Behauptung 1 Für alle $A \in N, w \in (N \cup T)^*, n \geq 0$ und Linksableitungen

$$A \underbrace{\vdash_G \ldots \vdash_G}_{n-\text{mal}} w$$

der Länge n gilt

$$(q, A) \overset{w_T}{\Longrightarrow} (q, w_R).$$

Beweis mit Induktion nach n:

$\underline{n = 0:}$ Dann ist $w = A$, also $w_T = \varepsilon$ und $w_R = A$. Trivialerweise gilt $(q, A) \overset{\varepsilon}{\Rightarrow} (q, A)$.

$\underline{n \to n+1:}$ Wir analysieren den letzten Schritt einer Linksableitung der Länge $n+1$:

$$A \underbrace{\vdash_G \ldots \vdash_G}_{n-\text{mal}} \tilde{w} = \tilde{w}_T B v \vdash_G \tilde{w}_T u v = w$$

für $B \in N$ und $u, v \in (N \cup T)^*$ mit $B \to u \in P$. Nach Induktions-Voraussetzung gilt

$$(q, A) \overset{\tilde{w}_T}{\Longrightarrow} (q, Bv).$$

Mit dem Transitionstyp (1) folgt

$$(q, Bv) \overset{\varepsilon}{\Rightarrow} (q, uv).$$

Mit dem Transitionstyp (2) gilt außerdem

$$(q, uv) \overset{(uv)_T}{\Longrightarrow} (q, (uv)_R).$$

Da $w_T = (\tilde{w}_T u v)_T = \tilde{w}_T (uv)_T$ und $w_R = (\tilde{w}_T u v)_R = (uv)_R$ gilt, erhalten wir insgesamt

$$(q, A) \overset{w_T}{\Longrightarrow} (q, w_R).$$

Damit ist Behauptung 1 bewiesen.

Behauptung 2 Für alle $A \in N, m \geq 0, \alpha_1, \ldots, \alpha_m \in T \cup \{\varepsilon\}, \gamma_0, \ldots, \gamma_m \in (N \cup T)^*$ und alle Transitionsfolgen

$$(q, A) = (q, \gamma_0) \xrightarrow{\alpha_1} (q, \gamma_1) \cdots \xrightarrow{\alpha_m} (q, \gamma_m)$$

der Länge m gilt

$$A \vdash_G^* \alpha_1 \ldots \alpha_m \gamma_m.$$

Beweis mit Induktion nach m:

$\underline{m = 0:}$ Dann ist $\gamma_0 = A$. Trivialerweise gilt $A \vdash_G^* A$.

$\underline{m \to m+1:}$ Wir analysieren die letzte Transition

$$(q, \gamma_m) \xrightarrow{\alpha_{m+1}} (q, \gamma_{m+1}).$$

Nach Induktions-Voraussetzung gilt $A \vdash_G^* \alpha_1 \ldots \alpha_m \gamma_m$.

Fall $\alpha_{m+1} = \varepsilon$
Dann wurde Transitionstyp (1) angewandt und die Transition ist von der Form

$$(q, \gamma_m) = (q, Bv) \xrightarrow{\varepsilon} (q, uv) = (q, \gamma_{m+1})$$

für gewisse $B \in N$ und $u, v \in (N \cup T)^*$ mit $B \to u \in P$. Damit gilt

$$A \vdash_G^* \alpha_1 \ldots \alpha_m Bv \vdash_G \alpha_1 \ldots \alpha_m uv = \alpha_1 \ldots \alpha_m \alpha_{m+1} \gamma_{m+1}.$$

Fall $\alpha_{m+1} = a \in T$
Dann wurde Transitionstyp (2) angewandt und die Transition ist von der Form

$$(q, \gamma_m) = (q, av) \xrightarrow{a} (q, v) = (q, \gamma_{m+1})$$

für ein gewisses $v \in (N \cup T)^*$. Dann gilt

$$A \vdash_G^* \alpha_1 \ldots \alpha_m av = \alpha_1 \ldots \alpha_m \alpha_{m+1} \gamma_{m+1}.$$

Damit ist auch Behauptung 2 bewiesen.

Aus den Behauptungen 1 und 2 folgt insbesondere, dass für alle Wörter $w \in T^*$ gilt:

$$\begin{aligned} S \vdash_G^* w \quad & \text{gdw. } (q, S) \overset{w}{\Rightarrow} (q, \varepsilon) \\ & \text{gdw. } \mathcal{K} \text{ akzeptiert } w \text{ mit leerem Keller.} \end{aligned}$$

Also gilt $L(G) = L_\varepsilon(\mathcal{K})$ wie gewünscht. □

Beispiel : Wir betrachten noch einmal die Sprache $L = \{a^n b^n \mid n \in \mathbb{N}\}$. Wir hatten bereits gesehen, dass die Sprache durch die kontextfreie Grammatik $G_1 = (\{S\}, \{a, b\}, P_1, S)$, wobei P_1 aus den Produktionen

$$S \to \varepsilon \mid aSb$$

besteht, erzeugt wird: $L(G_1) = L$. Die im obigen Beweis benutzte Konstruktion liefert den Kellerautomaten

$$\mathcal{K}_1 = (\{a,b\}, \{q\}, \{S,a,b\}, \to, q, S, \emptyset),$$

wobei die Transitionsrelation $\to$ aus folgenden Transitionen besteht:

$$\begin{array}{lcl} (q,S) & \stackrel{\varepsilon}{\to} & (q,\varepsilon) \\ (q,S) & \stackrel{\varepsilon}{\to} & (q,aSb) \\ (q,a) & \stackrel{a}{\to} & (q,\varepsilon) \\ (q,b) & \stackrel{b}{\to} & (q,\varepsilon) \end{array}$$

Aus dem Beweis folgt: $L_\varepsilon(\mathcal{K}_1) = L(G_1)$. Zur Veranschaulichung sei die Transitionsfolge von $\mathcal{K}_1$ beim Akzeptieren von a^2b^2 betrachtet:

$$\begin{array}{lllll} (q,S) & \stackrel{\varepsilon}{\to} & (q,aSb) & \stackrel{a}{\to} & (q,Sb) \\ & \stackrel{\varepsilon}{\to} & (q,aSbb) & \stackrel{a}{\to} & (q,Sbb) \\ & \stackrel{\varepsilon}{\to} & (q,bb) & \stackrel{b}{\to} & (q,b) \stackrel{b}{\to} (q,\varepsilon). \end{array}$$

Jetzt konstruieren wir umgekehrt zu jedem gegebenen Kellerautomat eine passende kontextfreie Grammatik.

3.7 **Satz** : Zu jedem Kellerautomaten $\mathcal{K}$ kann man eine kontextfreie Grammatik G mit $L(G) = L_\varepsilon(\mathcal{K})$ konstruieren.

Beweis : Sei $\mathcal{K} = (\Sigma, Q, \Gamma, \to, q_0, Z_0, F)$. Wir konstruieren $G = (N, T, P, S)$ mit

$$N = \{S\} \cup \{[q, Z, q'] \,|\, q, q' \in Q \text{ und } Z \in \Gamma\}.$$

Die Idee der Nichtterminalsymbole $[q, Z, q']$ ist wie folgt:

(1) Von $[q, Z, q']$ aus sollen in G alle Wörter $w \in \Sigma^*$ erzeugt werden, die $\mathcal{K}$ von der Konfiguration (q, Z) aus mit leerem Keller und dem Zustand q' akzeptieren kann: $(q, Z) \stackrel{w}{\Rightarrow} (q', \varepsilon)$.

(2) Eine Transition $(q, Z) \stackrel{\alpha}{\to} (r_0, Z_1 \dots Z_k)$ von $\mathcal{K}$ wird deshalb in G durch folgende Produktionen nachgebildet:

$$[q, Z, r_k] \to \alpha[r_0, Z_1, r_1][r_1, Z_2, r_2] \dots [r_{k-1}, Z_k, r_k],$$

wobei die $r_1, \dots, r_k$ über ganz Q laufen. Von $[r_0, Z_1, r_1]$ aus werden die Wörter erzeugt, die von $\mathcal{K}$ bis zum Abbau des Symbols Z_1 akzeptiert werden, von $[r_1, Z_2, r_2]$ die Wörter, die von $\mathcal{K}$ bis zum Abbau des Symbols Z_2 akzeptiert werden, usw. Die Zwischenzustände $r_1, \dots, r_{k-1}$ sind diejenigen, die $\mathcal{K}$ unmittelbar nach dem Abbau der Symbole $Z_1, \dots, Z_{k-1}$ erreicht.

Genauer besteht P aus den folgenden Transitionen:

- **Typ (1):** $S \to [q_0, Z_0, r] \in P$ für alle $r \in Q$,

- **Typ (2):** Für jede Transition $(q, Z) \xrightarrow{\alpha} (r_0, Z_1 \dots Z_k)$ mit $\alpha \in \Sigma \cup \{\varepsilon\}$ und $k \geq 1$ in $\mathcal{K}$: $[q, Z, r_k] \to \alpha [r_0, Z_1, r_1] \dots [r_{k-1}, Z_k, r_k] \in P$ für alle $r_1, ..., r_k \in Q$.
- **Typ (3):** (Spezialfall von (2) für $k = 0$.) Für jede Transition $(q, Z) \xrightarrow{\alpha} (r_0, \varepsilon)$ in $\mathcal{K}$: $[q, Z, r_0] \to \alpha \in P$.

Um zu zeigen, dass $L(G) = L_\varepsilon(\mathcal{K})$ gilt, untersuchen wir den Zusammenhang zwischen Ableitungen in G und Transitionfolgen in $\mathcal{K}$.

Behauptung 1 Für alle $q, q' \in Q,\ Z \in \Gamma,\ w \in \Sigma^*,\ n \geq 1$ und Ableitungen in G

$$[q, Z, q'] \underbrace{\vdash_G \dots \vdash_G}_{\leq n-\text{mal}} w$$

der Länge $\leq n$ gilt für $\mathcal{K}$

$$(q, Z) \overset{w}{\Rightarrow} (q', \varepsilon).$$

Beweis mit Induktion nach n:

<u>$n = 1$:</u> Aus $[q, Z, q'] \vdash_G w$ folgt wegen $w \in \Sigma^*$, dass es sich um den Produktionstyp (3) in G handelt. Daher gilt $w = \alpha \in \Sigma \cup \{\varepsilon\}$ und $(q, Z) \xrightarrow{\alpha} (q', \varepsilon)$. Also $(q, Z) \overset{\alpha}{\Rightarrow} (q', \varepsilon)$.

<u>$n \to n + 1$:</u> Wir analysieren den ersten Schritt einer Ableitung der Länge $n + 1$, der mit dem Produktionstyp (2) erfolgen muss:

$$[q, Z, r_k] \vdash_G \alpha\, [r_0, Z_1, r_1] \dots [r_{k-1}, Z_k, r_k] \underbrace{\vdash_G \dots \vdash_G}_{n-\text{mal}} \alpha\, w_1 \dots w_k = w,$$

wobei $(q, Z) \xrightarrow{\alpha} (r_0, Z_1 \dots Z_k)$ in $\mathcal{K}$, $r_k = q',\ \alpha \in \Sigma \cup \{\varepsilon\},\ w_1, \dots, w_k \in \Sigma^*$ und

$$[r_{i-1}, Z_i, r_i] \underbrace{\vdash_G \dots \vdash_G}_{\leq n-\text{mal}} w_i$$

für $i = 1, \dots, k$ gilt. Nach Induktions-Voraussetzung gilt in $\mathcal{K}$

$$(r_{i-1}, Z_i) \overset{w_i}{\Rightarrow} (r_i, \varepsilon)$$

für $i = 1, \dots, k$ und daher mit dem Top–Lemma

$$\begin{array}{rcl} (q, Z) \xrightarrow{\alpha} \ (r_0, Z_1 \dots Z_k) & \overset{w_1}{\Rightarrow} & (r_1, Z_2 \dots Z_k) \\ & \vdots & \\ (r_{k-1}, Z_k) & \overset{w_k}{\Rightarrow} & (r_k, \varepsilon). \end{array}$$

Also insgesamt $(q, Z) \overset{w}{\Rightarrow} (q', \varepsilon)$ für $\mathcal{K}$ wie gewünscht.

Behauptung 2 Für alle $q, q' \in Q,\ Z \in \Gamma,\ n \geq 1,\ \alpha_1, \dots, \alpha_n \in \Sigma \cup \{\varepsilon\}$ und alle Transitionsfolgen

$$(q, Z) \xrightarrow{\alpha_1} \circ \dots \circ \xrightarrow{\alpha_n} (q', \varepsilon)$$

in $\mathcal{K}$ der Länge n gilt in G

$$[q, Z, q'] \vdash_G^* \alpha_1 \dots \alpha_n.$$

Beweis mit Induktion nach n:

$\underline{n = 1:}$ Dann gilt $(q, Z) \overset{\alpha_1}{\to} (q', \varepsilon)$. Nach Definition von P in G — siehe Produktionstyp (3) — folgt $[q, Z, q'] \vdash_G \alpha_1$.

$\underline{n \to n+1:}$ Wir analysieren den ersten Schritt einer Transitionsfolge in $\mathcal{K}$ der Länge $n+1$:

$$(q, Z) \overset{\alpha_1}{\to} (r_0, Z_1 \dots Z_k) \overset{\alpha_2}{\to} \circ \cdots \circ \overset{\alpha_{n+1}}{\longrightarrow} (q', \varepsilon),$$

wobei $k \geq 1$ gilt. Nach Definition von P gibt es in G eine Produktion vom Typ (2)

$$[q, Z, q'] \to \alpha_1 [r_0, Z_1, r_1] \cdots [r_{k-1}, Z_k, r_k]$$

mit $r_k = q'$. Wir betrachten den sukzessiven Abbau des Kellerinhalts $Z_1 \dots Z_k$ von $\mathcal{K}$ genauer. Dazu gibt es in $\mathcal{K}$ Transitionsfolgen

$$\begin{aligned}
&(r_0, Z_1) \overset{\alpha_{11}}{\longrightarrow} \circ \cdots \circ \overset{\alpha_{1m_1}}{\longrightarrow} (r_1, \varepsilon) \\
&\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\
&(r_{k-1}, Z_k) \overset{\alpha_{k1}}{\longrightarrow} \circ \cdots \circ \overset{\alpha_{km_k}}{\longrightarrow} (r_k, \varepsilon) = (q', \varepsilon)
\end{aligned}$$

mit $\alpha_2 \dots \alpha_{n+1} = \alpha_{11} \dots \alpha_{1m_1} \dots \alpha_{k1} \dots \alpha_{km_k}$ und $m_1, \dots, m_k \leq n$. Nach Induktions-Voraussetzung gilt in G

$$[r_{i-1}, Z_i, r_i] \vdash_G^* \alpha_{i1} \dots \alpha_{im_i}$$

für $i = 1, \dots, k$. Insgesamt ergibt sich in G

$$[q, Z, q'] \vdash_G^* \alpha_1 \alpha_2 \dots \alpha_{n+1}$$

wie gewünscht.

Aus den Behauptungen 1 und 2 folgt: für alle $q \in Q$ und $w \in \Sigma^*$ gilt in G

$$S \vdash_G [q_0, Z_0, q] \vdash_G^* w$$

gdw. in $\mathcal{K}$

$$(q_0, Z_0) \overset{w}{\Rightarrow} (q, \varepsilon)$$

gilt. Damit haben wir insgesamt $L(G) = L_{\mathcal{E}}(\mathcal{K})$ gezeigt. □

§4 Abschlusseigenschaften

Wir untersuchen jetzt, unter welchen Operationen die Klasse der kontextfreien Sprachen abgeschlossen ist. Im Gegensatz zu den regulären (also endlich akzeptierbaren) Sprachen haben wir folgendes Resultat.

4.1 **Satz** : Die Klasse der kontextfreien Sprachen ist abgeschlossen unter den Operationen

(i) Vereinigung,

(ii) Konkatenation,

(iii) Iteration,

(iv) Durchschnitt mit regulären Sprachen.

Dagegen ist die Klasse der kontextfreien Sprachen *nicht* abgeschlossen unter den Operationen

(v) Durchschnitt,

(vi) Komplement.

Beweis : Seien $L_1, L_2 \subseteq T^*$ kontextfrei. Dann gibt es kontextfreie Grammatiken $G_i = (N_i, T, P_i, S_i)$ mit $L(G_i) = L_i$, wobei $i = 1, 2$ und $N_1 \cap N_2 = \emptyset$. Wir zeigen zunächst, dass $L_1 \cup L_2$, $L_1 \cdot L_2$ und L_1^* kontextfrei sind. Anschließend betrachten wir die Operationen Durchschnitt und Komplement. Sei $S \notin N_1 \cup N_2$ ein neues Startsymbol.

(i) $L_1 \cup L_2$: Betrachte die kontextfreie Grammatik $G = (\{S\} \cup N_1 \cup N_2, T, P, S)$ mit $P = \{S \to S_1, S \to S_2\} \cup P_1 \cup P_2$. Offenbar gilt $L(G) = L_1 \cup L_2$.

(ii) $L_1 \cdot L_2$: Betrachte die kontextfreie Grammatik $G = (\{S\} \cup N_1 \cup N_2, T, P, S)$ mit $P = \{S \to S_1 S_2\} \cup P_1 \cup P_2$. Offenbar gilt $L(G) = L_1 \cdot L_2$.

(iii) L_1^*: Betrachte die kontextfreie Grammatik $G = (\{S\} \cup N_1, T, P, S)$ mit $P = \{S \to \varepsilon, S \to S_1 S\} \cup P_1$. Dann gilt $S \vdash_G^* S_1^n$ für alle $n \geq 0$ und damit $L(G) = L_1^*$.

(iv) $L_1 \cap \mathit{regSpr}$: Für den Durchschnitt mit *regulären* Sprachen nutzen wir die Darstellung von kontextfreien und regulären Sprachen durch Kellerautomaten bzw. endliche Automaten aus. Sei $L_1 = L(\mathcal{K}_1)$ für den (nichtdeterministischen) KA $\mathcal{K}_1 = (T, Q_1, \Gamma, \to_1, q_{01}, Z_0, F_1)$ und $L_2 = L(\mathcal{A}_2)$ für den DEA $\mathcal{A}_2 = (T, Q_2, \to_2, q_{02}, F_2)$. Wir konstruieren aus $\mathcal{K}_1$ und $\mathcal{A}_2$ den (nichtdeterministischen) KA

$$\mathcal{K} = (T, Q_1 \times Q_2, \Gamma, \to, (q_{01}, q_{02}), F_1 \times F_2),$$

wobei die Transitionsrelation $\to$ für $q_1, q_1' \in Q_1, q_2, q_2' \in Q_2, Z \in \Gamma, \alpha \in T \cup \{\varepsilon\}$ und $\gamma' \in \Gamma^*$ so definiert ist:

$$((q_1, q_2), Z) \xrightarrow{\alpha} ((q_1', q_2'), \gamma') \text{ in } \mathcal{K}$$

gdw.

$$(q_1, Z) \xrightarrow{\alpha}_1 (q_1', \gamma') \text{ in } \mathcal{K}_1 \text{ und } q_2 \xrightarrow{\alpha}_2 q_2' \text{ in } \mathcal{A}_2.$$

Man beachte, dass im Spezialfall $\alpha = \varepsilon$ die Notation $q_2 \xrightarrow{\varepsilon}_2 q_2'$ für den DEA $\mathcal{A}_2$ einfach $q_2 = q_2'$ bedeutet. (Vergleiche dazu die Definition der erweiterten Transitionsrelation $q \xrightarrow{w}_2 q'$ für DEA's im Abschnitt1) Die Relation $\xrightarrow{\alpha}$ von $\mathcal{K}$ modelliert also das *synchrone parallele Fortschreiten* der Einzelautomaten $\mathcal{K}_1$ und $\mathcal{A}_2$. Im Spezialfall $\alpha = \varepsilon$ schreitet jedoch nur $\mathcal{K}_1$ durch einen spontanen ε-Übergang voran, während der DEA $\mathcal{A}_2$ im aktuellen Zustand stehen bleibt.

Wir zeigen, dass für das Akzeptieren mit Endzuständen gilt: $L(\mathcal{K}) = L(\mathcal{K}_1) \cap L(\mathcal{A}_2) = L_1 \cap L_2$. Sei dazu $w = a_1 \dots a_n \in T^*$ mit $n \geq 0$ und $a_i \in T$ für $i = 1, \dots, n$. Dann gilt:

$$\begin{array}{lll} w \in L(\mathcal{K}) & \Leftrightarrow \exists (q_1, q_2) \in F, \gamma \in \Gamma^* : & ((q_{01}, q_{02}), Z_0) \overset{a_1}{\Rightarrow} \circ \dots \circ \overset{a_n}{\Rightarrow} ((q_1, q_2), \gamma) \\ & \Leftrightarrow \exists q_1 \in F_1, q_2 \in F_2, \gamma \in \Gamma^* : & (q_{01}, Z_0) \overset{a_1}{\Rightarrow}_1 \circ \dots \circ \overset{a_n}{\Rightarrow}_1 (q_1, \gamma) \\ & & \text{und } q_{02} \xrightarrow{a_1}_2 \circ \dots \circ \xrightarrow{a_n}_2 q_2 \\ & \Leftrightarrow w \in L(\mathcal{K}_1) \cap L(\mathcal{A}_2). & \end{array}$$

(v) *nicht* $L_1 \cap L_2$: Dagegen sind die kontextfreien Sprachen *nicht* gegenüber dem Durchschnitt mit anderen kontextfreien Sprachen abgeschlossen. Betrachte dazu

$$L_1 = \{a^m b^n c^n \mid m, n \geq 0\}$$

und

$$L_2 = \{a^m b^m c^n \mid m, n \geq 0\}.$$

Es ist leicht einzusehen, dass L_1 und L_2 kontextfrei sind. Zum Beispiel kann L_1 duch die kontextfreie Grammatik $G = (\{S, A, B\}, \{a, b, c\}, P, S\}$ mit folgendem P erzeugt werden:

$$\begin{aligned} S &\to AB, \\ A &\to \varepsilon \mid aA, \\ B &\to \varepsilon \mid bBc. \end{aligned}$$

Der Durchschnitt von L_1 und L_2 liefert jedoch die Sprache

$$L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\},$$

von der wir mit Hilfe des Pumping Lemmas gezeigt haben, dass sie nicht kontextfrei ist.

(vi) *nicht* $\overline{L_i}$: Die kontextfreien Sprachen sind auch *nicht* gegenüber dem Komplement abgeschlossen. Diese Aussage folgt sofort aus (i) und (v) mit den De Morganschen Rechenregeln. Wären die kontextfreien Sprachen unter dem Komplement abgeschlossen, so wären sie auch gegenüber dem Durchschnitt abgeschlossen, weil $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ gilt.

□

§5 Transformation in Normalformen

Für Untersuchungen über kontextfreie Sprachen ist es günstig, wenn die Regeln der zugrundegelegten kontextfreien Grammatiken von möglichst einfacher Bauart sind. Wir stellen deshalb in diesem Abschnitt Transformationen vor, die gegebene kontextfreie Grammatiken in äquivalente Grammatiken überführen, deren Regeln zusätzlichen Bedingungen genügen.

5.1 **Definition** : Eine *ε-Produktion* ist eine Regel der Form $A \to \varepsilon$. Eine kontextfreie Grammatik $G = (N, T, P, S)$ heißt *ε-frei*, falls es in G

(i) entweder überhaupt keine ε-Produktion gibt

(ii) oder nur die ε-Produktion $S \to \varepsilon$ und S dann nicht auf der rechten Seite irgendeiner Produktion in G auftritt.

Im Fall (i) gilt $\varepsilon \notin L(G)$ und im Fall (ii) gilt $\varepsilon \in L(G)$.

5.2 **Satz** : Jede kontextfreie Grammatik lässt sich in eine äquivalente ε-freie Grammatik transformieren.

Beweis : Sei $G = (N, T, P, S)$ kontextfrei. G möge ε-Produktionen enthalten; anderenfalls ist nichts zu tun. Ein Symbol $A \in N$ heiße *löschbar*, falls $A \vdash_G^* \varepsilon$ gilt.

Schritt 1 Wir bestimmen zunächst alle löschbaren $A \in N$. Dazu bestimmen wir induktiv Mengen $N_1, N_2, N_3, \ldots$ von löschbaren Nichtterminalsymbolen:

$$\begin{aligned} N_1 &= \{A \in N \mid A \to \varepsilon \in P\} \\ N_{k+1} &= N_k \cup \{A \in N \mid A \to B_1 \ldots B_n \in P \text{ mit } B_1, \ldots, B_n \in N_k\} \end{aligned}$$

Diese Mengen bilden eine nach oben beschränkte, aufsteigende Folge:

$$N_1 \subseteq N_2 \subseteq N_3 \subseteq \ldots \subseteq N.$$

Da N endlich ist, gibt es ein kleinstes k_0 mit $N_{k_0} = N_{k_0+1}$.

Behauptung: $A \in N_{k_0} \iff A$ ist löschbar.
„$\Rightarrow$“ ist klar nach Definition von N_{k_0}.
„$\Leftarrow$“ zeigt man mit Induktion über die Tiefe des Ableitungsbaumes von A nach ε.

Schritt 2 Wir konstruieren eine zu G äquivalente Grammatik $G' = (N', T, P', S')$. Dabei sei S' ein neues Startsymbol und $N' = \{S'\} \cup N$. Die Produktionsmenge P' wird in zwei Schritten definiert. Zunächst führen wir die Menge P_0 ein, die aus P entsteht, indem jede Produktion

$$A \to \beta_1 \ldots \beta_n \in P$$

mit $\beta_1, \ldots, \beta_n \in N \cup T$ und $n \geq 1$ durch sämtliche Produktionen der Form

$$A \to \alpha_1 \ldots \alpha_n$$

ersetzt wird, wobei folgendes gilt:

- Falls $\beta_i \in N$ löschbar ist, dann ist $\alpha_i = \varepsilon$ oder $\alpha_i = \beta_i$.
- Falls $\beta_i \in T$ oder $\beta_i \in N$ nicht löschbar ist, dann ist $\alpha_i = \beta_i$.
- Nicht alle α_i's sind ε.

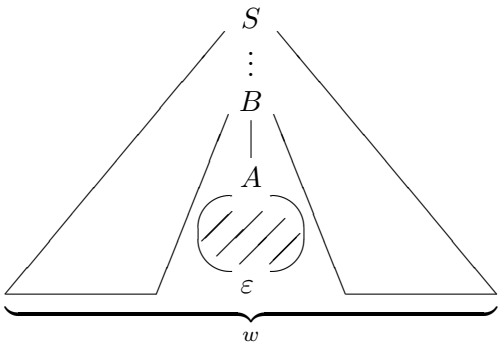
Dann gilt: P_0 enthält keine ε-Produktionen und $P - \{A \to \varepsilon \mid A \in N\} \subseteq P_0$. P' entsteht dann aus P_0 wie folgt:

$$P' = \{S' \to \varepsilon \mid S \text{ ist löschbar } \} \cup \{S' \to u \mid S \to u \in P_0\} \cup P_0$$

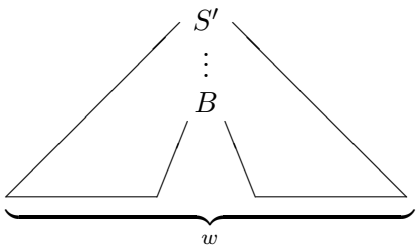
Damit ist G' vollständig definiert. Es bleibt zu zeigen: $L(G') = L(G)$.

„$\subseteq$“: Diese Inklusion ist klar, weil aus $S' \to u \in P'$ folgt, dass $S \vdash_G^* u$ gilt, und aus $A \to u \in P'$ mit $A \in N$ folgt, dass $A \vdash_G^* u$ gilt.

„$\supseteq$“: Falls $\varepsilon \in L(G)$ gilt, ist S löschbar und daher $S' \to \varepsilon \in P'$. Also gilt dann auch $\varepsilon \in L(G')$. Sei jetzt $w \in L(G) - \{\varepsilon\}$. Betrachte einen Ableitungsbaum von S nach w in G:



Wir erhalten daraus einen Ableitungsbaum von S' nach w in G', indem wir S durch S' ersetzen und alle maximalen Unterbäume, die Ableitungen der Form $A \vdash_G^* \varepsilon$ darstellen, entfernen:



Daher gilt $w \in L(G')$. □

5.3 **Definition** : Eine kontextfreie Grammatik $G = (N, T, P, S)$ ist in *Chomsky-Normalform* , wenn folgendes gilt:

- G ist ε-frei (so dass höchstens $S \to \varepsilon \in P$ erlaubt ist),
- jede Produktion in P anders als $S \to \varepsilon$ ist von der Form

$$A \to a \quad \text{oder} \quad A \to BC,$$

  wobei $A, B, C \in N$ und $a \in T$ sind.

5.4 **Satz** : Jede kontextfreie Grammatik lässt sich in eine äquivalente Grammatik in Chomsky-Normalform transformieren.

Beweis : siehe Literatur. □

5.5 **Definition** : Eine kontextfreie Grammatik $G = (N, T, P, S)$ ist in *Greibach-Normalform*, wenn folgendes gilt:

- G ist ε-frei (so dass höchstens $S \to \varepsilon \in P$ erlaubt ist),
- jede Produktion in P anders als $S \to \varepsilon$ ist von der Form

$$A \to aB_1 \dots B_k,$$

  wobei $k \geq 0, A, B_1, \dots, B_k \in N$ und $a \in T$ sind.

5.6 **Satz** : Jede kontextfreie Grammatik lässt sich in eine äquivalente Grammatik in Greibach-Normalform transformieren.

Beweis : siehe Literatur. □

§6 Deterministische kontextfreie Sprachen

Im Abschnitt 3 haben wir gezeigt, dass jede kontextfreie Sprache von einem im allgemeinen nichtdeterministischen Kellerautomaten akzeptiert werden kann. Es stellt sich die Frage, ob wir wie bei endlichen Automaten den Nichtdeterminismus beseitigen können und stets äquivalente deterministische Kellerautomaten konstruieren können. Diese Frage ist auch von praktischer Bedeutung für die Konstruktion von Parsern für gegebene kontextfreie Sprachen. Wir präzisieren zunächst den Begriff Determinismus für Kellerautomaten und kontextfreie Sprachen.

6.1 **Definition** :

(i) Ein *Kellerautomat* $\mathcal{K} = (\Sigma, Q, \Gamma, \to, q_0, Z_0, F)$ heißt *deterministisch*, falls die Transitionsrelation $\to$ folgende Bedingungen erfüllt:

$\forall q \in Q,\ Z \in \Gamma,\ a \in \Sigma$:
(Anzahl der Transitionen der Form $(q, Z) \xrightarrow{a} \cdots$
+ Anzahl der Transitionen der Form $(q, Z) \xrightarrow{\varepsilon} \cdots$) ≤ 1

(ii) Eine kontextfreie *Sprache* L heißt *deterministisch*, falls es einen deterministischen Kellerautomaten $\mathcal{K}$ mit $L = L(\mathcal{K})$ (Akzeptanz mit Endzuständen) gibt.

Beispiel : Die Sprache $L = \{a^n b^n \mid n \in \mathbb{N}\}$ ist deterministisch, denn im Abschnitt 3 haben wir einen deterministischen Kellerautomaten $\mathcal{K}$ mit $L(\mathcal{K}) = L$ angegeben.

Beispiel : Auch die Sprache $PAL_c = \{wcw^R \mid w \in \{a, b\}^*\}$ von Palindromen mit c als Mittelsymbol ist deterministisch kontextfrei. Die Notation w^R bedeute das Wort w rückwärts gelesen.

Für deterministische Kellerautomaten gilt der Satz 3.5 nicht, so dass wir $L(\mathcal{K})$ (Akzeptanz mit Endzuständen) nicht stets durch $L_\varepsilon(\mathcal{K})$ (Akzeptanz mit leerem Keller) ersetzen können.

Wir zeigen jetzt, dass nicht alle kontextfreien Sprachen deterministisch sind. Dazu benutzen wir den folgenden Satz.

6.2 **Satz** : Deterministische kontextfreie Sprachen sind unter Komplementbildung abgeschlossen.

Zum Beweis: Man könnte wie bei endlichen Automaten vorgehen und zu einem deterministischen KA $\mathcal{K} = (\Sigma, Q, \Gamma, \to, q_0, Z_0, F)$ den deterministischen KA $\mathcal{K}' = (\Sigma, Q, \Gamma, \to, q_0, Z_0, Q - F)$ betrachten. Leider gilt im allgemeinen $L(\mathcal{K}') \subsetneq \Sigma^* - L(\mathcal{K})$. Den Grund dafür, dass nicht alle Wörter aus dem Komplement von $L(\mathcal{K})$ akzeptiert werden, bilden die nichtterminierenden Berechnungen. Wenn z.B. eine Transition

$$(q, A) \xrightarrow{\varepsilon} (q, AA)$$

einmal verwendet wird, dann muss sie im deterministischen Fall immer wieder verwendet werden; das Kellerband wird dann unendlich lange beschrieben, und $\mathcal{K}$ hält nicht an. Falls diese Situation bei einem Eingabewort $w \in \Sigma^*$ eintritt, dann gilt $w \notin L(\mathcal{K})$. Die gleiche Situation tritt aber dann bei $\mathcal{K}$' auf, d.h. es gilt ebenfalls $w \notin L(\mathcal{K}')$.

Man muss also zunächst $\mathcal{K}$ in einen äquivalenten deterministischen KA umwandeln, der für *jedes* Eingabewort nach endlich vielen Schritten anhält. Eine solche Konstruktion ist bei Kellerautomaten tatsächlich möglich, da man die Menge

$$\{(q, A) \mid \exists \gamma \in \Gamma^* \text{ mit } (q, A) \stackrel{\varepsilon}{\Rightarrow} (q, A\gamma)\}$$

effektiv zu $\mathcal{K}$ konstruieren und die entsprechenden Transitionen $(q, A) \stackrel{\varepsilon}{\to} (q', \gamma')$ streichen, bzw. geeignet ersetzen kann.

Details: selbst überlegen oder in der Literatur nachsehen. □

6.3 **Korollar** : Es gibt kontextfreie Sprachen, die nicht deterministisch sind.

Beweis : Wenn alle kontextfreien Sprachen deterministisch wären, so wären die kontextfreien Sprachen unter Komplementbildung abgeschlossen. Widerspruch zum Satz aus Abschnitt 4.1 □

6.4 **Lemma** : Deterministische kontextfreie Sprachen sind

(i) abgeschlossen gegen Durchschnitt mit regulären Sprachen,

(ii) nicht abgeschlossen gegen Vereinigung, Durchschnitt, Konkatenation und Iteration.

Beweis : (i) beweist man mit der selben Konstruktion des nichtdeterministischen Falls (siehe Abschnitt 4.1); der dort aus $\mathcal{K}_1$ und $\mathcal{A}_2$ gebildete KA $\mathcal{K}$ ist deterministisch, falls $\mathcal{K}_1$ deterministisch ist.

(ii) Die Sprachen $L_1 = \{a^m b^n c^n \mid m, n \geq 0\}$ und $L_2 = \{a^m b^m c^n \mid m, n \geq 0\}$ sind beide deterministisch kontextfrei, ihr Durchschnitt ist jedoch nicht einmal kontextfrei. Wegen $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ sind die deterministisch kontextfreien Sprachen auch nicht gegen Vereinigung abgeschlossen. Zu Konkatenation und Iteration: siehe Literatur oder in den Übungen. □

Als Beispiel für eine kontextfreie Sprache, die nicht deterministisch ist, betrachten wir

$$PAL = \{ww^R \mid w \in \{a, b\}^* \wedge w \neq \varepsilon\},$$

die Sprache aller nicht leeren *Palindrome gerader Länge*. Die Notation w^R bedeute das Wort w rückwärts gelesen. Natürlich ist PAL kontextfrei: zur Erzeugung genügen die Produktionen

$$S \to aa \mid bb \mid aSa \mid bSb.$$

Um zu zeigen, dass PAL nicht deterministisch ist, benutzen wir einen Hilfsoperator *Min*.

6.5 **Definition** : Zu einer Sprache $L \subseteq \Sigma^*$ sei

$$Min(L) = \{w \in L \mid \text{ es gibt kein echtes Präfix } v \text{ von } w \text{ mit } v \in L\},$$

wobei v *echtes Präfix* von w heißt, falls $v \neq w$ und $\exists u \in \Sigma^* : w = v \cdot u$.

6.6 **Lemma** : Wenn L mit $\varepsilon \notin L$ deterministisch kontextfrei ist, so auch $Min(L)$.

Beweis : Betrachte einen deterministischen KA $\mathcal{K} = (\Sigma, Q, \Gamma, \to, q_0, Z_0, F)$ mit $L(\mathcal{K}) = L$. Wegen $\varepsilon \notin L$ gilt $q_0 \notin F$. Wir ändern $\mathcal{K}$ zu einem KA $\mathcal{K}_1$ ab, der wie $\mathcal{K}$ arbeitet, aber in jeder Transitionsfolge *höchstens einmal* in einen Endzustand aus F gerät und dann sofort anhält. Betrachte dazu einen neuen Zustand $q_1 \notin Q$ und definiere $\mathcal{K}_1 = (\Sigma, Q \cup \{q_1\}, \Gamma, \to_1, q_0, Z_0, \{q_1\})$ mit

$$\begin{aligned} \to_1 &= \{(q, Z, \alpha, q', \gamma') \mid q \in Q - F \text{ und } (q, Z, \alpha, q', \gamma') \in \to\} \\ &\cup \{(q, Z, \varepsilon, q_1, Z) \mid q \in F \text{ und } Z \in \Gamma\} \end{aligned}$$

$\mathcal{K}_1$ ist deterministisch, und es gilt $L(\mathcal{K}_1) = Min(L(\mathcal{K}))$, weil $\mathcal{K}$ deterministisch ist. □

Wir zeigen jetzt:

6.7 **Satz** : Die kontextfreie Sprache PAL ist nicht deterministisch.

Beweis : *Annahme: PAL* ist deterministisch. Nach den vorangegangenen beiden Lemmata ist dann auch die Sprache

$$L_0 = Min(PAL) \cap L((ab)^+(ba)^+(ab)^+(ba)^+)$$

deterministisch kontextfrei. Dabei steht $(ab)^+$ für den regulären Ausdruck $ab(ab)^*$ und analog $(ba)^+$ für den regulären Ausdruck $ba(ba)^*$. Da alle Wörter in L_0 Palindrome gerader Länge ohne echtes Präfix sind, gilt

$$L_0 = \{(ab)^i(ba)^j(ab)^j(ba)^i \mid i > j > 0\}.$$

Nach dem Pumping Lemma gibt es für L_0 ein $n \in \mathbb{N}$ mit den dort genannten Eigenschaften. Insbesondere lässt sich dann das Wort

$$z = (ab)^{n+1}(ba)^n(ab)^n(ba)^{n+1} \in L_0$$

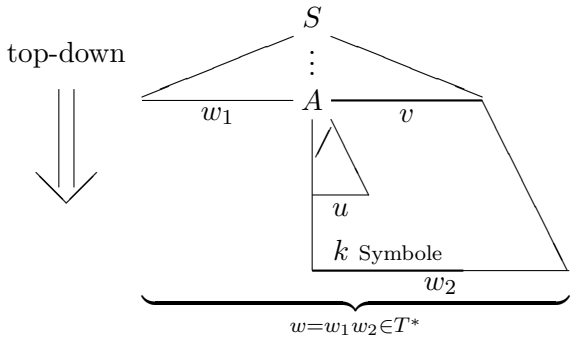
zerlegen in $z = uvwxy$. Da der Mittelteil vwx die Bedingungen $|vwx| \leq n$ und $vx \neq \varepsilon$ erfüllt, lässt sich zeigen, dass nicht alle Wörter der Form uv^iwx^iy mit $i \in \mathbb{N}$ in L_0 liegen können. Daher ist L_0 nicht einmal kontextfrei, geschweige denn deterministisch kontextfrei. *Widerspruch* □

Anmerkung: Da L_0 nicht kontextfrei ist, folgt aus den Abschlusseigenschaften, dass die kontextfreien Sprachen nicht gegen den Operator *Min* abgeschlossen sind.

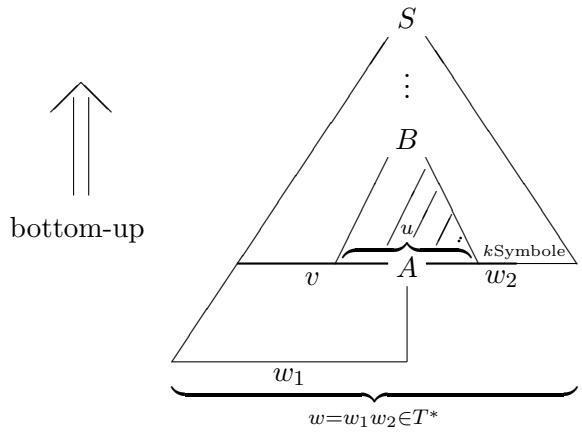
Für die praktische Syntaxanalyse von Programmiersprachen werden genau die deterministischen kontextfreien Sprachen eingesetzt. Es gibt zwei verschiedene Methoden zur Syntaxanalyse: In der top-down Methode werden die Ableitungsbäume vom Startsymbol der Grammatik aus konstruiert und in der bottom-up Methode vom vorgegebenen Wort aus rekonstruiert. Es wird verlangt,

dass der jeweils nächste Ableitungsschritt eindeutig aus einer Vorausschau („look ahead“) von k Symbolen für ein geeignetes $k \geq 0$ ermittelt werden kann.

Für die top-down Methode sind die sogenannten $LL(k)$-*Grammatiken* gebräuchlich. Das sind kontextfreie Grammatiken $G = (N, T, P, S)$, wo für jedes Zwischenwort $w_1 Av$ in einer Linksableitung $S \vdash_G^* w_1 Av \vdash_G^* w_1 w_2 = w$ von S zu einem Wort $w \in T^*$ das Stück Av und die ersten k Symbole des Restes w_2 von w *eindeutig* den nächsten Linksableitungsschritt $w_1 Av \vdash_G w_1 uv$ nach w bestimmen. Graphisch kann diese $LL(k)$-Bedingung wie folgt dargestellt werden:



Für die bottom-up Methode sind sogenannte $LR(k)$-*Grammatiken* gebräuchlich. Das sind kontextfreie Grammatiken $G = (N, T, P, S)$, wo für jedes Zwischenwort vAw_2 in der bottom-up Rekonstruktion einer Rechtsableitung $S \vdash_G^* vAw_2 \vdash_G^* w_1 w_2 = w$ von S zu einem Wort $w \in T^*$ das Stück vA und die ersten k Symbole vom Rest w_2 von w eindeutig den *vorher* nötigen Rechtsableitungsschritt $\tilde{v}B\tilde{w}_2 \vdash_G \tilde{v}u\tilde{w}_2 = vAw_2$ bestimmen. Graphisch kann diese $LR(k)$-Bedingung wie folgt dargestellt werden:



$LL(k)$-Grammatiken wurden 1968 von P.M. Stearns und R.E. Stearns und $LR(k)$-Grammatiken 1965 von D.E. Knuth eingeführt. Für die von diesen Grammatiken erzeugten $LL(k)$- und $LR(k)$-Sprachen gelten folgende Beziehungen:

- $\left(\bigcup_{k\geq 0} LL(k)\text{-Sprachen}\right) \subsetneq \left(\bigcup_{k\geq 0} LR(k)\text{-Sprachen}\right) =$ determ. kontextfreie Sprachen

- Sogar: $LR(1)$-Sprachen = determ. kontextfreie Sprachen

Genaue Aussagen über diese Grammatiken und Sprachen werden in den Vorlesungen „Formale Sprachen“ und „Compilerbau“ gemacht.

§7 Entscheidbarkeitsfragen

Die folgenden Konstruktionen sind algorithmisch berechenbar:

- KA akzeptierend mit Endzuständen $\leftrightarrow$ KA akzeptierend mit leerem Keller
- KA $\leftrightarrow$ kontextfreie Grammatik
- kontextfreie Grammatik $\mapsto$ ε-freie Grammatik
- kontextfreie Grammatik $\mapsto$ Chomsky- oder Greibach-Normalform

Entscheidbarkeitsfragen über kontextfreie Sprachen können daher nach Belieben über die Darstellung durch kontextfreie Grammatiken (in Normalformen) oder durch Kellerautomaten beantwortet werden. Wir untersuchen dieselben Probleme wie für reguläre Sprachen (vgl. Kapitel II, Abschnitt 5).

7.1 **Satz (Entscheidbarkeit)**: Für kontextfreie Sprachen sind

- das Wortproblem,
- das Leerheitsproblem,
- das Endlichkeitsproblem

entscheidbar.

Beweis :
Wortproblem: Gegeben sei eine kontextfreie Grammatik $G = (N, T, P, S)$ in Greibach-Normalform und ein Wort $w \in T^*$. Die Frage lautet: Gilt $w \in L(G)$? Der Fall $w = \varepsilon$ ist sofort zu entscheiden, da G ε-frei ist: $\varepsilon \in L(G) \iff S \to \varepsilon \in P$. Sei jetzt $w \neq \varepsilon$. Dann gilt Folgendes:

$$
\begin{aligned}
w \in L(G) \quad &\Leftrightarrow \quad \exists n \geq 1 : S \underbrace{\vdash_G \ldots \vdash_G}_{n-\text{mal}} w \\
&\Leftrightarrow \quad \{ \text{ In Greibach-Normalform produziert jeder Ableitungsschritt genau einen Buchstaben von } w. \} \\
&\qquad\quad S \underbrace{\vdash_G \ldots \vdash_G}_{|w|-\text{mal}} w
\end{aligned}
$$

Um $w \in L(G)$ festzustellen, genügt es also, alle Ableitungsfolgen der Länge $|w|$ in G zu überprüfen. Daraus folgt die Entscheidbarkeit des Wortproblems. Effizientere Verfahren zur Lösung des Wortproblems für kontextfreie Sprachen werden in den Vorlesungen „Formale Sprachen“ und „Compilerbau“ vorgestellt.

Leerheitsproblem: Gegeben sei eine kontextfreie Grammatik $G = (N, T, P, S)$. Die Frage lautet: Gilt $L(G) = \emptyset$? Sei n die nach dem Pumping Lemma zur kontextfreien Sprache $L(G)$ gehörige Zahl. Wie für reguläre Sprachen zeigt man:

$$L(G) = \emptyset \iff \neg\exists w \in L(G) : |w| < n.$$

Damit lässt sich das Leerheitsproblem entscheiden, indem für alle Wörter $w \in T^*$ mit $|w| < n$ das Wortproblem entschieden wird.

Endlichkeitsproblem: Gegeben sei eine kontextfreie Grammatik $G = (N, T, P, S)$. Die Frage lautet: Ist $L(G)$ endlich? Sei n wie eben. Dann zeigt man wie für reguläre Sprachen:

$$L(G) \text{ ist endlich} \iff \neg\exists w \in L(G) : n \leq |w| < 2 \cdot n$$

Damit kann das Endlichkeitsproblem durch endlichmaliges Lösen des Wortproblems entschieden werden. □

Im Gegensatz zu regulären Sprachen gilt jedoch folgendes Resultat.

7.2 **Satz (Unentscheidbarkeit)**: Für kontextfreie Sprachen sind

- das Schnittproblem,
- das Äquivalenzproblem,
- das Inklusionsproblem

unentscheidbar.

Dieser Satz können wir erst beweisen, wenn wir den Begriff des Algorithmus formalisiert haben.

Ein weiteres Unentscheidbarkeitsresultat betrifft die Mehrdeutigkeit von kontextfreien Grammatiken. Für die praktische Anwendung von kontextfreien Grammatiken zur Syntaxbeschreibung von Programmiersprachen wäre es angenehm, einen algorithmischen Test auf Mehrdeutigkeit zu haben. Wir zeigen jedoch später, dass es einen solchen Test nicht gibt.

7.3 **Satz** : Es ist unentscheidbar, ob eine gegebene kontextfreie Grammatik mehrdeutig ist.

In der Praxis lässt sich das Problem, die Mehrdeutigkeit testen zu müssen, recht einfach durch die Beschränkung auf $LR(1)$-Grammatiken bzw. Teilklassen hiervon umgehen. Die $LR(1)$-Eigenschaft kann algorithmisch entschieden werden, und da LR-Grammatiken stets eindeutig sind (weil der letzte Rechtsableitungsschritt immer eindeutig bestimmt sein muss und daher jedes Wort nur *eine* Rechtsableitung haben kann), entfällt das Problem der Mehrdeutigkeit.

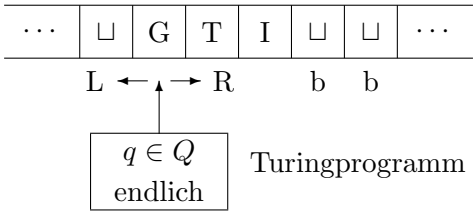
Kapitel IV

Zum Begriff des Algorithmus: Was ist maschinell berechenbar?

§1 Turingmaschinen

1936 von A.M. Turing (1912–1954) eingeführt. Es soll ein möglichst elementares Modell für das Rechnen mit Bleistift und Papier angegeben werden. Hierzu benötigt man ein Arbeitsband, auf dem man Zeichen eintragen und verändern kann und eine endliche Berechnungsvorschrift (Programm).

Skizze



L: bedeutet: gehe ein Feld auf dem Band nach links.
R: analog nach rechts.
S: keine Bewegung.

1.1 **Definition** : Eine *Turingmaschine*, kurz TM, ist eine Struktur $\tau = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup)$ mit folgenden Eigenschaften:

(i) Q ist endliche, nichtleere Menge von *Zuständen*,
(ii) $q_0 \in Q$ ist der Anfangszustand,
(iii) Γ ist endliche nichtleere Menge, das *Bandalphabet*, mit $Q \cap \Gamma = \emptyset$,
(iv) $\Sigma \subseteq \Gamma$ ist das Eingabealphabet,
(v) $\sqcup \in \Gamma - \Sigma$ ist das *Leerzeichen* oder *Blank*

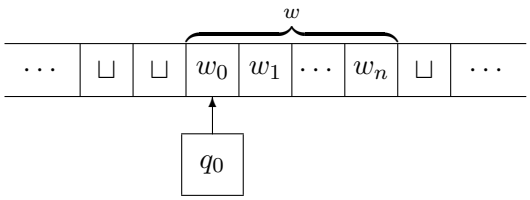
(vi) $\delta : Q \times \Gamma \xrightarrow{part} Q \times \Gamma \times \{R, L, S\}$ ist die *Überführungsfunktion.*
Darstellung von δ als *Turingtafel* oder *Turingprogramm*:

$$\begin{array}{lc|l} \delta: & q_1 a_1 & q_1' a_1' P_1 \\ & \dots & \dots \\ & \dots & \dots \\ & \dots & \dots \\ & \dots & \dots \\ & q_n a_n & q_n' a_n' P_n \end{array} \qquad \begin{array}{ll} \text{mit} & q_i, q_i' \in Q, \\ & a_i, a_i' \in \Gamma, \\ & P_i \in \{R, L, S\} \end{array}$$

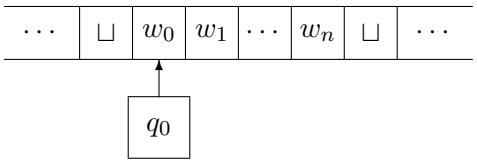
Arbeitsweise einer TM: informell

Konfigurationen der TM beschreiben den momentanen Zustand, den Bandinhalt und das betrachtete Feld.

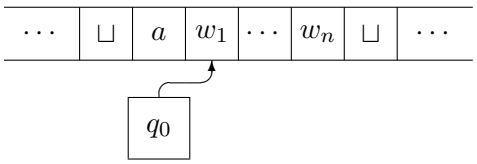
- Anfangskonfiguration:



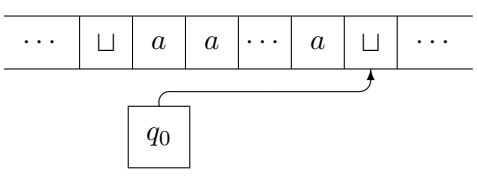
- Ausführen eines Schrittes:



z.B. $\delta(q_0, w_0) = (q_0, a, R)$ führt zu



- Wiederholung dieses Schrittes liefert das Ergebnis:



wobei $\delta(q_0, \sqcup)$ undefiniert sei.
Das Ergebnis der Berechnung ist dann das Wort $aa \ldots a$, d.h. der Bandinhalt ohne die Blanks.

Beispiel : Berechne die Funktion
gerade: $\{\,|\,\}^* \longrightarrow \{0, 1\}$
mit

$$gerade(|^n) \quad = \quad \begin{cases} 1 & \text{falls } n \text{ durch 2 teilbar ist} \\ 0 & \text{sonst} \end{cases}$$

für $(n \geq 0)$. Wähle

$$\tau_3 = (\{q_0, q_1, q_e\}, \{\,|\,\}, \{\,|, 0, 1, \sqcup\}, \delta, q_0, \sqcup)$$

mit folgender Turingtafel:

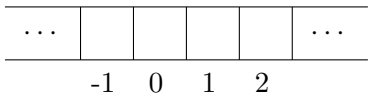
$$\delta: \quad \begin{array}{cc|ccc} q_0 & | & q_1 & \sqcup & R \\ q_0 & \sqcup & q_e & 1 & S \\ q_1 & | & q_0 & \sqcup & R \\ q_1 & \sqcup & q_e & 0 & S \end{array}$$

Man rechnet leicht nach, dass τ_3 die Funktion *gerade* berechnet.

Begriff der Konfiguration

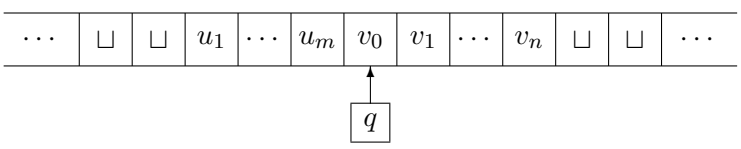
Es ist der momentane Zustand q, die momentane Bandinschrift und die momentane Position des Schreib/Lesekopfes zu beschreiben. Zwei Modellierungen sind üblich:

(1) Felder des unendlichen Bandes durchnumerieren:



Bandinschrift: $f : \mathbb{Z} \longrightarrow \Gamma$ ($\mathbb{Z}$ = Menge der ganzen Zahlen)
Konfiguration: Tripel (q, f, i) mit $i \in \mathbb{Z}$
Nachteil: umständliche Handhabung

(2) Nur ein endlicher Ausschnitt des Bandes ist verschieden von $\sqcup$. Abstraktion von Blanks und Feldnummern. Die Situation



lässt sich eindeutig darstellen als Wort $\overbrace{u_1 \ldots u_m}^{u} q \overbrace{v_0 v_1 \ldots v_n}^{v}$ mit $u_i \in \Gamma, v_j \in \Gamma, m, n \geq 0$.
Beachte: $Q \cap \Gamma = \emptyset$

1.2 **Definition** : Die Menge $\mathcal{K}_\tau$ der *Konfigurationen* einer TM $\tau = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup)$ ist durch

$$\mathcal{K}_\tau = \Gamma^* \cdot Q \cdot \Gamma^+$$

gegeben. Eine gegebene Konfiguration uqv bedeutet, dass sich die TM im Zustand q befindet, der Bandinhalt $\sqcup^\infty uv\sqcup^\infty$ ist und das erste (linkeste) Symbol des Wortes v gelesen wird.

1.3 **Definition (Arbeitsweise einer TM $\tau = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup)$)**:

(1) Die *Anfangskonfiguration* $\alpha(v)$ zu einem Wort $v \in \Sigma^*$ lautet

$$\alpha(v) = \begin{cases} q_0 v & \text{falls } v \neq \varepsilon \\ q_0\sqcup & \text{sonst} \end{cases}$$

(2) Die *Transitionsrelation* $\vdash_\tau \subseteq \mathcal{K}_\tau \times \mathcal{K}_\tau$ ist wie folgt definiert:

$$K \vdash_\tau K' \qquad (K' \text{ heißt } \textit{Folgekonfiguration} \text{ von } K)$$

falls $\exists u, v \in \Gamma^*\ \exists a, b \in \Gamma\ \exists q, q' \in Q$:

$$\begin{aligned}
&(K = uqav \ \wedge \delta(q,a) = (q', a', S) \wedge K' = uq'a'v\ ) \\
\vee\ &(K = uqabv \wedge \delta(q,a) = (q', a', R) \wedge K' = ua'q'bv) \\
\vee\ &(K = uqa \ \ \wedge \delta(q,a) = (q', a', R) \wedge K' = ua'q'\sqcup\ ) \\
\vee\ &(K = ubqav \wedge \delta(q,a) = (q', a', L) \wedge K' = uq'ba'v) \\
\vee\ &(K = qav \ \ \wedge \delta(q,a) = (q', a', L) \wedge K' = q'\sqcup a'v\ )
\end{aligned}$$

Mit $\vdash_\tau{}^*$ werde die *reflexive transitive Hülle* von $\vdash_\tau$ bezeichnet, d.h. es gilt

$$\begin{aligned}
K \vdash_\tau{}^* K' \text{ falls } \ & \exists K_0, \ldots, K_n, n \geq 0 : \\
& K = K_0 \vdash_\tau \ldots \vdash_\tau K_n = K'
\end{aligned}$$

Weiterhin werde mit $\vdash_\tau{}^+$ die *transitive Hülle* von $\vdash_\tau$ bezeichnet, d.h. es gilt

$$\begin{aligned}
K \vdash_\tau{}^+ K' \text{ falls } \ & \exists K_0, \ldots, K_n, n \geq 1 : \\
& K = K_0 \vdash_\tau \ldots \vdash_\tau K_n = K'
\end{aligned}$$

(3) Eine *Endkonfiguration* ist eine Konfiguration $K \in \mathcal{K}_\tau$, die keine Folgekonfiguration besitzt.

(4) Das *Ergebnis* (oder die sichtbare Ausgabe) einer Konfiguration uqv ist

$$\omega(uqv) = \overline{u\overline{v}},$$

wobei $\overline{u}$ das kürzeste Wort mit $u = \sqcup \ldots \sqcup \overline{u}$ und $\overline{v}$ das kürzeste Wort mit $v = \overline{v}\sqcup \ldots \sqcup$ ist. Man streicht also q, sowie führende und endende Blanks; Blanks, die sich zwischen Zeichen befinden, bleiben dagegen stehen.

Bemerkung : Da δ eine (partielle) Funktion ist, ist $\vdash_\tau$ rechtseindeutig, d.h. aus $K \vdash_\tau K_1 \wedge K \vdash_\tau K_2$ folgt $K_1 = K_2$.

1.4 **Definition** : Die von der TM $\tau = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup)$ *berechnete Funktion* ist

$$h_\tau : \quad \Sigma^* \xrightarrow{part} \Gamma^*$$

mit

$$h_\tau(v) = \begin{cases} w & \text{falls} \quad \exists \text{ Endkonfiguration } K \in \mathcal{K}_\tau : \\ & \qquad \alpha(v) \vdash_\tau{}^* K \wedge w = \omega(K) \\ \text{undef.} & \text{sonst.} \end{cases}$$

Es wird auch Res_τ für h_τ geschrieben („Resultatsfunktion“von τ).

Bemerkung : Da $\vdash_\tau$ rechtseindeutig ist, ist h_τ eine partielle Funktion.

Veranschaulichung der Resultatsfunktion:

$$\begin{array}{ccccccc} \Sigma^* & \ni & v & \xrightarrow{\qquad h_\tau \qquad} & w & \in & \Gamma^* \\ & & \alpha \downarrow & & \uparrow \omega & & \\ \mathcal{K}_\tau & \ni & \alpha(v) \vdash_\tau & \cdots \quad \vdash_\tau & K & \in & \mathcal{K}_\tau \end{array}$$

1.5 **Definition** : Eine Menge $M \in \Sigma^*$ heißt *Haltebereich* oder *Definitionsbereich* von τ, falls gilt:

$$M = \{v \in \Sigma^* \mid h_\tau(v) \text{ ist definiert } \}$$

Eine Menge $N \in \Gamma^*$ heißt *Ergebnisbereich* oder *Wertebereich* von τ, falls

$$N = \{w \in \Gamma^* \mid \exists v \in \Sigma^* : h_\tau(v) = w\}.$$

1.6 **Definition (Turing-Berechenbarkeit)**:
Es seien A, B Alphabete.

(i) Eine partiell definierte *Funktion* $h : A^* \xrightarrow{part} B^*$ heißt *Turing-berechenbar*, falls es eine TM $\tau = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup)$ gibt mit $A = \Sigma$, $B \subseteq \Gamma$ und $h = h_\tau$, d.h. $h(v) = h_\tau(v)$ für alle $v \in A^*$.

(ii) $\mathcal{T}_{A,B} =_{def} \{h : A^* \xrightarrow{part} B^* \mid \ h \text{ ist Turing-berechenbar } \}$

(iii) $\mathcal{T}$ sei die Klasse aller Turing-berechenbaren Funktionen (für beliebige Alphabete A, B).

1.7 **Definition (Turing-Entscheidbarkeit)**:
Es sei A ein Alphabet.

(i) Eine *Menge* $L \subseteq A^*$ heißt *Turing-entscheidbar*, falls die *charakteristische Funktion von* L

$$\chi_L : \quad A^* \longrightarrow \{0,1\}$$

Turing-berechenbar ist. Hierbei ist χ_L folgende totale Funktion:

$$\chi_L(v) = \begin{cases} 1 & \text{falls } v \in L \\ 0 & \text{sonst} \end{cases}$$

(ii) Eine *Eigenschaft* $E : A^* \longrightarrow \{$ *wahr, falsch* $\}$ heißt Turing-entscheidbar, falls die Menge $\{v \in A^* \mid E(v) =$ *wahr* $\}$ der Wörter mit der Eigenschaft E Turing-entscheidbar ist.

Anmerkungen zur Berechenbarkeit

(1) *Berechenbarkeit mehrstelliger Funktionen:*
k-Tupel als Eingabe lassen sich durch eine Funktion $h : (A^*)^k \xrightarrow{part} B^*$ beschreiben. Für die Berechenbarkeit solcher Funktionen modifiziert man einfach die Definition der Anfangskonfiguration durch Benutzung von *Trennzeichen* # :

$$\alpha(v_1, \ldots, v_k) = q_0 v_1 \# v_2 \# \ldots \# v_k$$

Falls $v_1 = \varepsilon$ ist, beobachtet q_0 das erste Trennsymbol. Abgesehen von dieser Änderung benutzen wir die bisherige Definition von Berechenbarkeit.

(2) *Berechenbarkeit zahlentheoretischer Funktionen:*
$f : \mathbb{N} \xrightarrow{part} \mathbb{N}$
Wir benutzen die *Strichdarstellung* natürlicher Zahlen:

$$n \mathrel{\widehat{=}} |^n = \underbrace{|\ldots|}_{n-\text{mal}}$$

Dann heißt f Turing-berechenbar, falls die Funktion

$$h_f : \{\,|\,\}^* \xrightarrow{part} \{\,|\,\}^*$$

mit

$$h_f : (\,|^n\,) = |^{f(n)}$$

Turing-berechenbar ist.

Konstruieren von Turingmaschinen: die Flussdiagrammschreibweise

Man definiert zunächst nützliche elementare TM. Sei $\Gamma = \{a_0, \ldots, a_n\}$ das Bandalphabet mit $a_0 = \sqcup$.

- Kleine Rechtsmaschine r:
  geht einen Schritt nach rechts und hält dann. Turingtafel:

r	q_0	a_0	q_e	a_0	R
	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
	q_0	a_n	q_e	a_n	R

- Kleine Linksmaschine l:
  geht einen Schritt nach links und hält dann. Turingtafel:

l	q_0	a_0	q_e	a_0	L
	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
	q_0	a_n	q_e	a_n	L

- Druckmaschine a für $a \in \Gamma$:
  druckt das Symbol a und hält dann. Turingtafel:

a	q_0	a_0	q_e	a	S
	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
	q_0	a_n	q_e	a	S

Wir nehmen im folgenden an, dass alle konstruierten TM genau einen *Endzustand* besitzen, d.h. es gibt einen Zustand q_e, so dass für alle Endkonfigurationen uqv gilt:

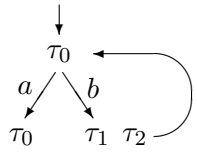
$$q = q_e.$$

Offenbar erfüllen die TM r, l, a diese Bedingung. Solche TM können wir wie folgt zusammensetzen:

$\tau_1 \xrightarrow{a} \tau_2$	bedeutet anschaulich, dass zuerst τ_1 arbeitet. Hält τ_1 auf einem Feld mit dem Symbol a an, wird τ_2 gestartet.
$\tau_1 \longrightarrow \tau_2$	bedeutet, dass zuerst τ_1 arbeitet. Sobald τ_1 anhält, wird τ_2 gestartet.
$\tau_1\tau_2$	ist eine Abkürzung für $\tau_1 \longrightarrow \tau_2$.
$\tau_1 \xrightarrow{\neq a} \tau_2$	bedeutet, dass zuerst τ_1 arbeitet. Hält τ_1 auf einem Feld mit dem Symbol $\neq a$ an, wird τ_2 gestartet.

Aus gegebenen TM können Flussdiagramme aufgebaut werden. Die Knoten dieser Flussdiagramme sind mit den Namen der TM bezeichnet. Die Kanten werden durch Pfeile der Form $\xrightarrow{a}$, $\longrightarrow$ oder $\xrightarrow{\neq a}$ bezeichnet. Schleifen sind erlaubt. Eine TM τ im Flussdiagramm ist durch einen Pfeil $\longrightarrow \tau$ als Start-TM gekennzeichnet.

Veranschaulichung:



Ein Flussdiagramm beschreibt eine „große“ TM, deren Turingtafel man wie folgt erhält:

Schritt 1: Für jedes Vorkommen einer TM τ_i im Flussdiagramm die zugehörige Turingtafel aufstellen.

Schritt 2: Zustände in verschiedenen Tafeln disjunkt machen.

Schritt 3: Gesamttafel erzeugen, indem alle Einzeltafeln (in irgendeiner Reihenfolge) untereinander geschrieben werden.

Schritt 4: *Kopplung*: für jeden Pfeil $\tau_1 \xrightarrow{a} \tau_2$ im Flussdiagramm füge der Gesamttafel die Zeile

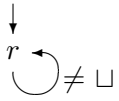
$$q_{e\tau_1} a q_{0\tau_2} a S$$

hinzu. Dabei sei $q_{e\tau_1}$ der (gemäß Schritt 2 umbenannte) Endzustand von τ_1 und $q_{0\tau_2}$ der (gemäß Schritt 2 umbenannte) Anfangszustand von τ_2. Analog für $\tau_1 \to \tau_2$ und $\tau_1 \xrightarrow{\neq a} \tau_2$.

Beispiel :

- Große Rechtsmaschine $\mathcal{R}$:
  geht zuerst einen Schritt nach rechts. Anschließend geht $\mathcal{R}$ so lange nach rechts, bis ein Blank $a_0 = \sqcup$ beobachtet wird.

  $\mathcal{R}$

  Konstruktion der Turingtafel von $\mathcal{R}$:

  $\mathcal{R}$

q_0	a_0	q_e	a_0	R
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
q_0	a_n	q_e	a_n	R

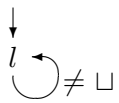
  r

q_e	a_1	q_0	a_1	S
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
q_e	a_n	q_0	a_n	S

  $\neq a_0$

- Große Linksmaschine $\mathcal{L}$: entsprechend.

  $\mathcal{L}$

Anwendung: Man gebe eine TM zur Berechnung der „minus“-Funktion an:

$$f : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$$

mit

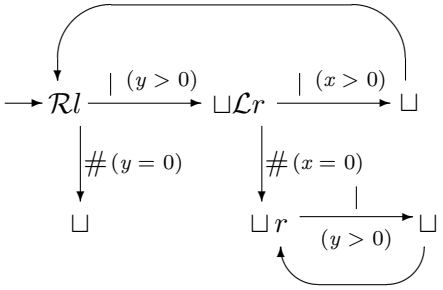
$$f(x,y) = x \dot{-} y = \begin{cases} x - y & \text{für } x \geq y \\ 0 & \text{sonst} \end{cases}$$

Die Anfangskonfiguration der TM ist:

$$\sqcup q_0 \underbrace{\| \ldots \|}_{x-\text{mal}} \# \underbrace{\| \ldots \|}_{y-\text{mal}} \sqcup$$

Idee: x-Striche so oft löschen, wie y-Striche da sind. Dann restliche y-Striche und # löschen.

Konstruktion der TM mit Hilfe eines Flussdiagramms:



Das Aufstellen der Gesamt-Turingtafel sei als Übung empfohlen.

Varianten von Turingmaschinen

Es gibt viele Varianten der Definition von TM. Oft sind diese Varianten bequemer, wenn es darum geht, eine bestimmte Funktion als berechenbar nachzuweisen. Man kann zeigen, dass diese Varianten die Leistungsfähigkeit der in Definition 1.1 definierten TM nicht vergrößert. Wir betrachten zunächst TM mit *mehreren Bändern.*

1.8 **Definition (k-Band Turingmaschine)**: Eine *k-Band TM* $\tau = (Q, \Sigma, \Gamma, \delta, k, q_0, \sqcup)$ ist eine Struktur mit folgenden Eigenschaften:

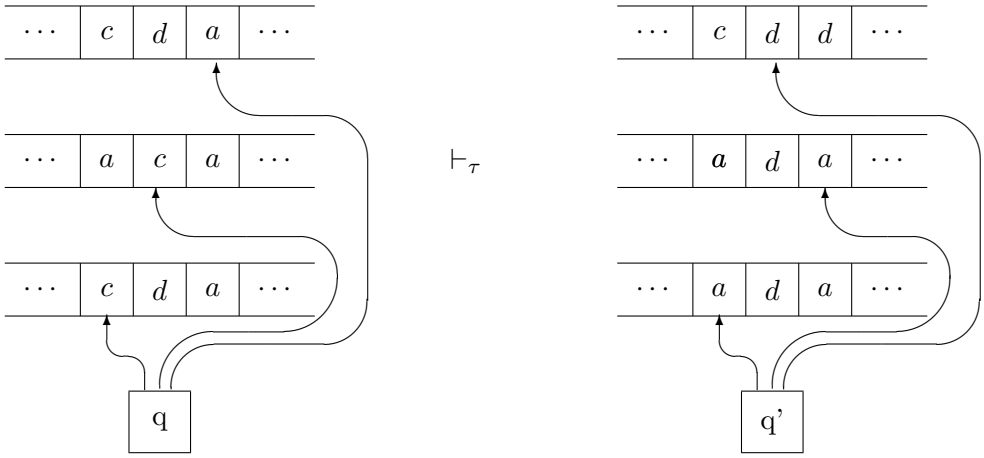
(i)-(v) $Q, \Sigma, \Gamma, q_0, \sqcup$ wie in Definition 1.1.

(vi) $k \in \mathbb{N}$ ist die Anzahl der Bänder

(vii) Die Übergangsfunktion δ ist jetzt wie folgt gegeben:

$$\delta : Q \times \Gamma^k \stackrel{part}{\longrightarrow} Q \times \Gamma^k \times \{R, L, S\}^k$$

Veranschaulichung von δ für $k = 3$:



Dieser Konfigurationsübergang erfolgt für

$$\delta(q, (a, c, c)) = (q', (d, d, a), (L, R, S))$$

Konfiguration einer k-Band TM τ:

$$K = (u_1 q v_1, \ldots, u_k q v_k) \in \mathcal{K}_\tau$$

mit $u_1, \ldots, u_k \in \Gamma^*$, $q \in Q$, $v_1, \ldots, v_k \in \Gamma^+$.

Arbeitsweise einer k-Band TM τ:

(1) Anfangskonfiguration zu einem Wort $v \in \Sigma^*$ ist

$$\alpha_k(v) = (\alpha(v), \underbrace{q_0\sqcup, \ldots, q_0\sqcup}_{(k-1)-\text{mal}}),$$

d.h. v wird auf das erste Band geschrieben.

(2) Transitionsrelation $\vdash_\tau \subseteq \mathcal{K}_\tau \times \mathcal{K}_\tau$: analog

(3) Endkonfiguration: keine Folgekonfiguration (wie bisher)

(4) Das Ergebnis ω_k einer k-Band Konfiguration ist

$$\omega_k(u_1 q v_1, \ldots, u_k q v_k) = \omega(u_1 q v_1),$$

d.h. wir nehmen das Ergebnis des ersten Bandes; die übrigen Bänder werden nur als Hilfsbänder zum Rechnen betrachtet.

Die *berechnete Funktion* einer k-Band TM ist $h_\tau : \Sigma^* \overset{part}{\longrightarrow} \Gamma^*$ mit

$$h_\tau(v) = \begin{cases} w & \text{falls } \exists\text{Endkonfig. } K \in \mathcal{K}_\tau : \\ & \alpha_k(v) \vdash_\tau{}^* K \wedge w = \omega_k(K) \\ r\text{undef.} & \end{cases}$$

Anmerkung: Mehrere Bänder eignen sich gut zur Berechnung mehrstelliger Funktionen; in diesem Fall verteilt man die Eingabewörter geeignet auf die Bänder. Details selbst überlegen.

Ziel: Wir wollen beweisen, dass jede von einer k-Band TM berechnete Funktion bereits von einer normalen 1-Band TM berechnet werden kann.

Beweisidee: Jede k-Band TM kann durch eine geeignet konstruierte 1-Band TM „simuliert“ werden. Hierzu präzisieren wir zunächst den Begriff der Simulation.

Begriff der Simulation

Der Begriff der Simulation ist zentral für den Nachweis, dass verschiedenartige Machinen dasselbe leisten.

1.9 **Definition** : Wir betrachten zwei (1- oder k-Band) TM τ und τ' über demselben Eingabealphabet Σ, mit den Konfigurationsmengen $\mathcal{K}_\tau$ und $\mathcal{K}_{\tau'}$, mit Transitionsrelationen $\vdash_\tau, \vdash_{\tau'}$, Anfangskonfigurationen α, α' und Ergebnis-Funktionen ω, ω'. Anschaulich wird τ die „höhere“ TM und τ' die „niedrigere“ TM.

Eine *Simulation von τ durch τ'* ist eine total definierte Funktion

$$sim : \mathcal{K}_\tau \longrightarrow \mathcal{K}_{\tau'}$$

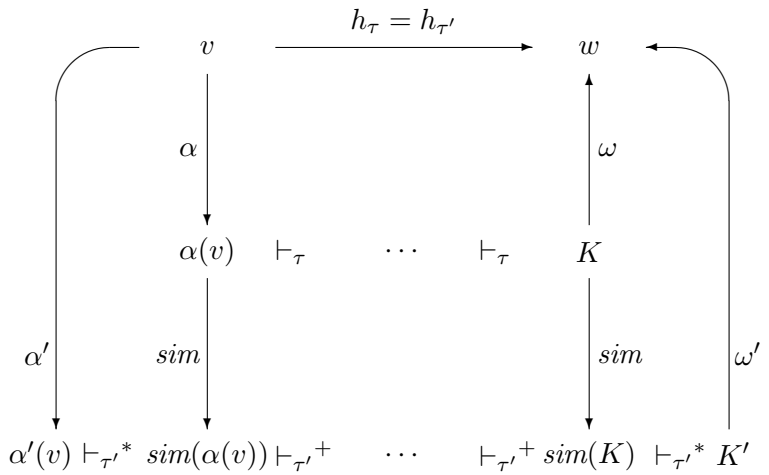
mit folgenden Eigenschaften:

(1) $\forall v \in \Sigma^* : \quad \alpha'(v) \vdash_{\tau'}{}^* sim(\alpha(v))$

(2) $\forall K_1, K_2 \in \mathcal{K}_\tau$: Aus $K_1 \vdash_\tau K_2$ folgt $sim(K_1) \vdash_{\tau'}{}^+ sim(K_2)$,
d.h. jeder Schritt von τ kann durch möglicherweise mehrere Schritte von τ' „simuliert“ werden.

(3) $\forall$Endkonfig. $K \in \mathcal{K}_\tau \quad \exists$Endkonfig. $K' \in \mathcal{K}_{\tau'}$:
$sim(K) \vdash_{\tau'}{}^* K'$ und $\omega(K) = \omega'(K')$.

Veranschaulichung:



1.10 **Satz (Simulations-Satz)**: Seien τ und τ' (1- oder k-Band) TM über demselben Eingabealphabet Σ. Es gebe eine Simulation *sim* von τ durch τ'. Dann stimmen die von τ und τ' berechneten Funktionen h_τ und $h_{\tau'}$ überein, d.h.

$$\forall v \in \Sigma^* : \quad h_\tau(v) = h_{\tau'}(v)$$

(Die Gleichheit ist hier zu verstehen in dem Sinne, dass entweder beide Seiten undefiniert sind oder denselben Funktionswert haben).

Beweis : Sei $v \in \Sigma^*$.
Fall 1: $h_\tau(v)$ ist undefiniert.
Dann gibt es eine unendliche Konfigurationsfolge

$$\alpha(v) = K_1 \vdash_\tau K_2 \vdash_\tau K_3 \vdash_\tau \dots$$

von τ. Mit Eigenschaften (1) und (2) von *sim* gibt es dann auch eine unendliche Konfigurationsfolge

$$\alpha'(v) \vdash_{\tau'}{}^* sim(K_1) \vdash_{\tau'}{}^+ sim(K_2) \vdash_{\tau'}{}^+ sim(K_3) \vdash_{\tau'}{}^+ \dots$$

von τ'. Also ist $h_{\tau'}(v)$ auch undefiniert.
Fall 2: $h_\tau(v) = w$ ist definiert.
Dann gibt es eine endliche Konfigurationsfolge

$$\alpha(v) = K_1 \vdash_\tau \dots \vdash_\tau K_n, \quad n \geq 1,$$

von τ, wobei K_n Endkonfiguration ist und $w = \omega(K_n)$ gilt. Mit den Eigenschaften (1) - (3) von *sim* gibt es eine endliche Konfigurationsfolge

$$\alpha'(v) \vdash_{\tau'}{}^* sim(K_1) \vdash_{\tau'}{}^+ \dots \vdash_{\tau'}{}^+ sim(K_n) \vdash_{\tau'}{}^* K'_n$$

von τ', wobei K'_n Endkonfiguration mit $\omega(K_n) = \omega'(K'_n)$. Also gilt

$$h_\tau(v) = w = \omega(K_n) = \omega'(K'_n) = h_{\tau'}(v).$$

□

Wir können nun zeigen:
1.11 **Satz** : Für jede k-Band TM τ gibt es eine 1-Band TM τ' mit $h_\tau = h_{\tau'}$.

Beweis : Sei $\tau = (Q, \Sigma, \Gamma, \delta, k, q_0, \sqcup)$. Wir konstruieren $\tau' = (Q', \Sigma', \Gamma', \delta', q'_0, \sqcup)$ so, dass es eine Simulation *sim* von τ durch τ' gibt. Definition von *sim*:

K	$= ($	$u_1 q a_1 v_1$,	wird	$sim(K) =$
		$\dots$,	simuliert	$u_1 q \widetilde{a_1} v_1 \# \dots \# u_k \widetilde{a_k} v_k$
		$u_k q a_k v_k$ $)$	durch	
	$\in$	$\mathcal{K}_\tau$		$\in \mathcal{K}_{\tau'}$

D.h. wir markieren die von den k Köpfen gelesenen Symbole $a_1, \dots, a_k$ durch Benutzung neuer Symbole $\widetilde{a_1}, \dots, \widetilde{a_k}$, schreiben die k Bandinhalte hintereinander auf das eine Band von τ', jeweils durch # getrennt, und setzen den Zustand q so, dass $\widetilde{a_1}$ beobachtet wird.

Wir wählen also: $\Gamma' = \Gamma \cup \{\tilde{a} \mid a \in \Gamma\} \cup \{\#\}$. Die Wahl von Q' und δ' skizzieren wir nur anhand der folgenden Simulationsidee. Ein Schritt von τ der Form

$$\begin{array}{rcl} K_1 = (u_1 q_1 a_1 v_1, & & K_2 = (u_1 q_2 b_1 v_1, \\ \quad \cdots \quad , & \vdash_\tau & \quad \cdots \quad , \\ u_k q_1 a_k v_k) & & u_k q_2 b_k v_k) \end{array}$$

erzeugt durch

$$\delta(q_1, (a_1, \ldots, a_k)) = (q_2, (b_1, \ldots, b_k), (S, \ldots, S))$$

wird von τ' durch 2 Phasen von Schritten simuliert.

- *Lese-Phase:*

$$sim(K_1) = u_1 q_1 \widetilde{a_1} v_1 \# \ldots \# u_k \widetilde{a_k} v_k$$
$$\vdash_{\tau'}$$
$$u_1 [lesen, q_1] \widetilde{a_1} v_1 \# \ldots \# u_k \widetilde{a_k} v_k$$
$$\vdash^*_{\tau'}$$
$$u_1 \widetilde{a_1} v_1 \# \ldots \# u_k [lesen, q_1, a_1, \ldots, a_k] \widetilde{a_k} v_k$$

- *Änderungs-Phase:*
  Jetzt wird gemäß $\delta(q_1, (a_1, \ldots, a_k))$ die Konfiguration abgeändert.

$$u_1 \widetilde{a_1} v_1 \# \ldots \# u_k [lesen, q_1, a_1, \ldots, a_k] \widetilde{a_k} v_k$$
$$\vdash_{\tau'}$$
$$u_1 \widetilde{a_1} v_1 \# \ldots \# u_k [ändern, q_2, b_1, \ldots, b_k, S, \ldots, S] \widetilde{a_k} v_k$$
$$\vdash^*_{\tau'}$$
$$u_1 [ändern, q_2] \widetilde{b_1} v_1 \# \ldots \# u_k \widetilde{b_k} v_k$$
$$\vdash_{\tau'}$$
$$sim(K_2) = u_1 q_2 \widetilde{b_1} v_1 \# \ldots \# u_k \widetilde{b_k} v_k$$

Ein Schritt von τ wird also durch maximal so viele Schritte von τ' simuliert, wie der doppelten Länge aller auf den k Bändern stehenden Wörter entspricht.

Ähnlich werden andere δ-Übergänge behandelt. Falls jedoch bei R- oder L-Schritten auf einem der k Bänder ein leeres Feld $\sqcup$ „angeklebt“ werden muss, müssen auf dem einen Band von τ' die Inhalte der Nachbar #-Abschnitte *verschoben* werden.

Wir betrachten jetzt noch die Anfangs- und Endkonfigurationen.

- *Anfangskonfiguration:* Sei $v \in \Sigma^*$ und $v \neq \varepsilon$.
  Für τ ist dann
  $\alpha(v) = (q_0 v\,,$ und $sim(\alpha(v)) = q_0 v \# \sqcup \# \ldots \# \sqcup$
  $q_0 \sqcup,$
  $\ldots$
  $q_0 \sqcup)$
  Für τ' ist $\alpha'(v) = q_0' v$. Natürlich kann man τ' so programmieren, dass

  $$\alpha'(v) \vdash_{\tau'}{}^* sim(\alpha(v))$$

  gilt.

- *Endkonfiguration:* Ist $K \in \mathcal{K}_\tau$ Endkonfiguration der Form

  $$K = (u_1 q_e a_1 v_1\,,\ \ldots\ ,\ u_k q_e a_k v_k),$$

  so muss diese Tatsache für

  $$sim(K) = u_1 q_e \widetilde{a_1} v_1 \# \ldots \# u_k \widetilde{a_k} v_k$$

  erst durch eine Lese-Phase festgestellt werden. Anschließend muss $sim(K)$ durch Löschen aller Symbole ab dem ersten # in die Ergebnisform von 1-Band TM gebracht werden. Deshalb können wir τ' so programmieren, dass

  $$sim(K) \vdash_{\tau'}{}^* u_1 q_e' a_1 v_1$$

  gilt und $K' = u_1 q_e' a_1 v_1$ Endkonfiguration von τ' mit $\omega_k(K) = \omega'(K')$ ist.

Wir haben also:

$$\begin{aligned} Q' = Q \cup \{&q_0', q_e' && \} \\ \cup \{&[lesen, q, a_1, \ldots, a_j] \mid q \in Q, a_1, \ldots, a_j \in \Gamma, 0 \leq j \leq k && \} \\ \cup \{&[ändern, q, b_1, \ldots, b_j, P_1, \ldots, P_j] \mid q \in Q, 0 \leq j \leq k, b_1, \ldots, b_j \in \Gamma, \\ &P_1, \ldots, P_j \in \{R, L, S\} && \} \\ \cup \{&\ldots \text{ weitere Zustände } \ldots && \} \end{aligned}$$

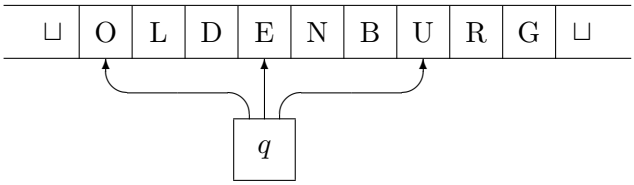
Insgesamt haben wir jetzt gezeigt, dass *sim* eine Simulation von τ durch τ' ist. Nach dem Simulations-Satz folgt damit $h_\tau = h_{\tau'}$.

□

Weitere Varianten von Turingmaschinen

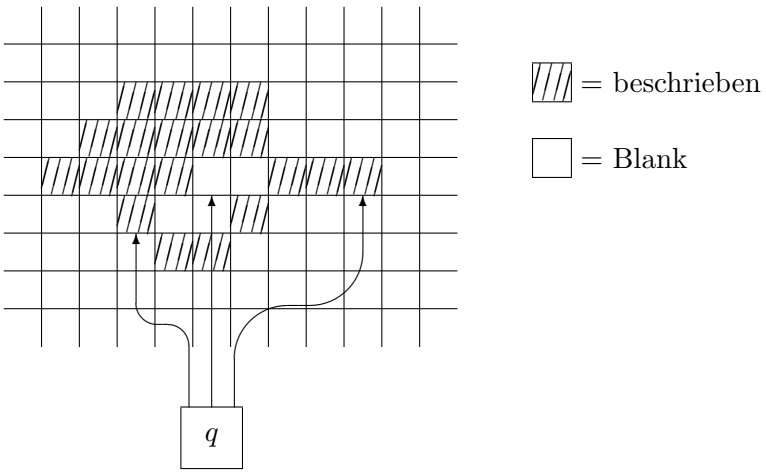
Im folgenden seien weitere Varianten der Turingmaschine aufgezeigt:

- Turingmaschinen mit k *Köpfen* auf einem Band:

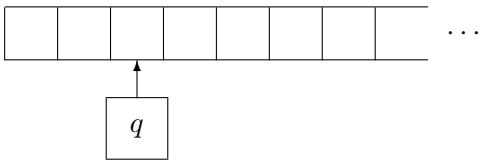


  Definition der Übergangsfunktion δ: wie in Definition 1.8 von k-Band TM, aber die Definition von Konfiguration ist anders, da k Stellen im Bandinhalt zu markieren sind.

- Turingmaschinen mit einem *zweidimensionalen* in Felder aufgeteilten Rechenraum („Rechnen auf kariertem Papier“), eventuell mit mehreren Köpfen:



- Turingmaschinen mit einem *einseitig* unendlichen Band:



  Normale TM (mit zweiseitig unendlichem Band) lassen sich wie folgt durch TM mit einseitig unendlichem Band simulieren:

  2-seitige Konfiguration $K = a_m \dots a_1 q b_1 \dots b_n$ mit $m \geq 0, n \geq 1$
  wird (für $m \leq n$) simuliert durch die

  1-seitige Konfiguration $sim(K) =$

q	b_1	$\cdots$	b_m	b_{m+1}	$\cdots$	b_n
	a_1		a_m	$\sqcup$		$\sqcup$

  d.h. als Symbole des 1-seitigen Bandes treten Paare von Symbolen des 2-seitigen Bandes auf.

- Turingmaschinen mit *Endzuständen*:

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

wobei $F \subseteq Q$ die Menge der Endzustände ist und alle übrigen Komponenten wie bisher definiert sind.

TM mit Endzuständen werden benutzt, um Sprachen $L \subseteq \Sigma^*$ zu „akzeptieren“.

1.12 **Definition** :

(1) Eine Konfiguration $K = uqv$ von τ heißt *akzeptierend*, falls $q \in F$ ist.

(2) Sei $v \in \Sigma^*$. Die TM τ *akzeptiert* v, falls $\exists$ akzeptierende Endkonfiguration $K : \alpha(v) \vdash_\tau{}^* K$.

(3) Die von τ *akzeptierte Sprache* ist

$$L(\tau) = \{v \in \Sigma^* \mid \tau \text{ akzeptiert } v\}.$$

Eine Sprache $L \subseteq \Sigma^*$ heißt *Turing-akzeptierbar*, falls es eine TM τ mit Endzuständen gibt, für die $L = L(\tau)$ gilt.

(4) Mit $\mathcal{T}$ bezeichnen wir die Menge aller Turing-akzeptierbaren Sprachen.

1.13 **Satz** : Sei $L \subseteq \Sigma^*$ und $\overline{L} = \Sigma^* - L$.
L und $\overline{L}$ sind Turing-akzeptierbar $\Leftrightarrow$ L ist Turing-entscheidbar.

Beweis : „$\Leftarrow$“: Sei L durch τ entscheidbar. Durch Hinzufügen von Endzustandsübergängen erhält man akzeptierende TM für L und $\overline{L}$.
„$\Rightarrow$“: L werde durch τ_1 und $\overline{L}$ durch τ_2 akzeptiert. Konstruiere nun τ mit 2 Bändern so, dass *ein Schritt von τ_1 auf Band 1 und gleichzeitig ein Schritt von τ_2 auf Band 2* ausgeführt wird. Akzeptiert τ_1 das vorgegebene Wort, so gibt τ den Wert 1 aus. Sonst akzeptiert τ_2 das vorgegebene Wort, und τ gibt den Wert 0 aus. Somit entscheidet τ die Sprache L. □

Bemerkung : Wenn L nur Turing-akzeptierbar ist, so folgt noch nicht, dass L auch Turing-entscheidbar ist. Die akzeptierende TM könnte nämlich für Wörter aus $\overline{L}$ *nicht anhalten*.

- *Nichtdeterministische Turingmaschinen:*
  Wir betrachten 1-Band TM mit Endzuständen. Die Übergangsfunktion δ wird wie folgt erweitert:

$$\delta : \quad Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{R, L, S\}).$$

Ein Tupel

$$(q', b, R) \in \delta(q, a)$$

bedeutet, dass die TM, falls im Zustand q das Symbol a gelesen wird, Folgendes tun kann:

Zustand q' annehmen, b drucken, nach rechts gehen.

Falls es ein weiteres Tupel

$$(q'', c, S) \in \delta(q, a)$$

gibt, so kann die TM stattdessen auch folgendes tun:

Zustand q'' annehmen, c drucken, stoppen.

Die Auswahl zwischen diesen beiden möglichen Schritten der TM ist *nicht deterministisch*, d.h. ins Belieben der TM gestellt. Die Transitionsrelation $\vdash_\tau$ einer nichtdeterministischen TM τ ist nicht mehr rechtseindeutig. Davon abgesehen ist die von τ akzeptierte Sprache $L(\tau)$ wie für deterministische TM definiert.

1.14 **Satz** : Jede von einer nichtdeterministischen TM akzeptierte Sprache ist auch durch eine deterministische TM akzeptierbar.

Beweis : Gegeben sei eine nichtdeterministische 1-Band TM τ. Dann gibt es eine maximale Anzahl r von nichtdeterministischen Auswahlen, die τ gemäß δ in einem Zustand q und Symbol a hat, d.h.

$$\begin{aligned} &\forall q \in Q \quad \forall a \in \Gamma : \quad |\delta(q,a)| \leq r \\ \text{und} \quad &\exists q \in Q \quad \exists a \in \Gamma : \quad |\delta(q,a)| = r. \end{aligned}$$

Wir nennen r den *Grad des Nichtdeterminismus* von τ. Für jedes Paar (q, a) numerieren wir die gemäß $\delta(q, a)$ möglichen Auswahlen von 1 bis (maximal) r durch. Dann lässt sich jede endliche Folge von nichtdeterministischen Auswahlen als Wort über dem Alphabet $\{1, \ldots, r\}$ darstellen.

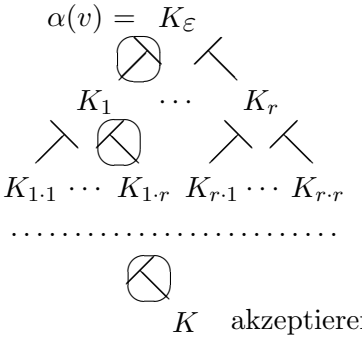
Beispiel: Für $r = 5$ bedeutet z.B. $1 \cdot 3 \cdot 5$:

- im 1. Schritt die 1. Auswahl nehmen,
- im 2. Schritt die 3. Auswahl nehmen,
- im 3. Schritt die 5. Auswahl nehmen.

Sei jetzt $v \in \Sigma^*$. Dann akzeptiert τ das Wort v, falls es eine akzeptierende Berechnung von τ gibt:

$$\alpha(v) \vdash_\tau{}^* K \quad \text{akzeptierend.}$$

Die Menge aller in $\alpha(v)$ startenden Berechnungen von τ können als Baum dargestellt werden, deren Knoten mit Konfigurationen benannt sind:



⊖ markieren die Auswahlen, die zur akzeptierenden Konfiguration K geführt haben.

Wir „konstruieren“ jetzt eine deterministische TM τ' so, dass τ' jeden Berechnungsbaum von τ in *Breitensuche* generiert und durchläuft:

0. $\rightarrow$
1. $\longrightarrow$
2. $\longrightarrow$
3. $\longrightarrow$

$\ldots$

Dazu benötigt τ' 3 Bänder.

- Band 1 speichert das Eingabewort v.
- Band 2 wird zur systematischen Erzeugung aller Wörter über $\{1, \ldots, r\}$ benutzt. Um die Breitensuche zu modellieren, werden die Wörter wie folgt erzeugt:
  (i) der Länge nach, kürzere zuerst,
  (ii) bei gleicher Länge in lexikographischer Ordnung. Also in folgender Reihenfolge: $\varepsilon, 1, 2, \ldots, r, 1 \cdot 1, 1 \cdot 2, \ldots, 1 \cdot r, \ldots, r \cdot 1, \ldots, r \cdot r, \ldots$
- Band 3 dient zur Simulation von τ durch τ'. Zuerst wird das Eingabewort v auf Band 3 kopiert, dann eine Transitionsfolge von τ gemäß dem auf Band 2 stehenden Wort von Auswahlen aus $\{1, \ldots, r\}$ simuliert.

Wenn τ eine v akzeptierende Berechnung besitzt, so wird deren Auswahlwort von τ' auf Band 2 generiert werden, und τ' wird dann v akzeptieren. Wenn es keine v akzeptierende Berechnung von τ gibt, so wird τ' unendlich lange laufen und nach und nach alle Wörter über $\{1, \ldots, r\}$ auf Band 2 generieren.
Damit gilt $L(\tau) = L(\tau')$. □

Bemerkung : Die obige Simulation von τ durch τ' ist von *exponentiellem Schrittaufwand*: um eine akzeptierende Berechnung von τ mit n Schritten zu finden, muss τ' einen Berechnungsbaum der Breite r und Tiefe n, also mit r^n Knoten, erzeugen und durchsuchen.

Hier betrachtete Varianten von Turingmaschinen

- k-Band TM
- mehrere Köpfe
- 2-dimensionale TM
- TM mit einseitig unendlichem Band
- Endzustände: zum Akzeptieren
- nichtdeterministisch

§2 Grammatiken

Wir haben im letzten Kapitel kontextfreie Grammatiken kennengelernt. Diese sind ein Spezialfall der 1959 vom amerikanischen Linguisten NOAM CHOMSKY eingeführten CHOMSKY-Grammatiken. Es werden mehrere Typen solcher Grammatiken unterschieden (Typ 0—3). Wir betrachten hier den allgemeinsten Typ 0.

2.1 **Definition (Grammatiken)**:
Eine (CHOMSKY-0- oder kurz *CH0-*)*Grammatik* ist eine Struktur $G = (N, T, P, S)$ mit

(i) N ist ein Alphabet von *Nichtterminalsymbolen*,

(ii) T ist ein Alphabet von *Terminalsymbolen* mit $N \cap T = \emptyset$,

(iii) $S \in N$ ist das *Startsymbol*,

(iv) $P \subseteq (N \cup T)^* \times (N \cup T)^*$ ist eine endliche Menge von *Produktionen* oder *Regeln*, wobei für $(u, v) \in P$ verlangt wird, dass u mindestens *ein Nichtterminalsymbol* enthält.

Wie bei kontextfreien Grammtiken werden Produktionen $(u, v) \in P$ meist in Pfeilnotation $u \to v$ geschrieben. Zu jeder Grammatik G gehört die zweistellige *Ableitungsrelation* $\vdash_G$ auf $(N \cup T)^*$:

$$x \vdash_G y \quad \text{gdw} \quad \exists u \to v \in P \; \exists w_1, w_2 \in (N \cup T)^* : \\ x = w_1 \boxed{u} w_2 \text{ und } y = w_1 \boxed{v} w_2.$$

Mit $\vdash_G^*$ wird wieder die reflexive und transitive Hülle von $\vdash_G$ bezeichnet. Wir lesen $x \vdash^* y$ als „y ist aus x ableitbar“. Es gilt

$$x \vdash^* y, \text{ falls } \exists z_0, \ldots, z_n \in \Sigma^*, n \geq 0 : \; x = z_0 \vdash_G \ldots \vdash_G z_n = y$$

Die Folge $x = z_0 \vdash_G \ldots \vdash_G z_n = y$ heißt auch eine *Ableitung* von y aus x (oder von x nach y) der *Länge* n. Insbesondere gilt: $x \vdash_G^* x$ mit einer Ableitung der Länge 0.

2.2 **Definition** :

(i) Die von G *erzeugte Sprache* ist

$$L(G) = \{w \in T^* \mid S \vdash_G^* w\},$$

d.h. wir sind nur an Wörtern über den Terminalsymbolen interessiert; die Nichtterminalsymbole werden nur als Hilfssymbole innerhalb von Ableitungen benutzt.

(ii) $L \subseteq T^*$ heißt CHOMSKY-0- (oder kurz *CH0-*) *Sprache*, falls es eine CHOMSKY-0-Grammatik G mit $L = L(G)$ gibt. Die Klasse aller CH0-Sprachen wird auch kurz mit *CH0* bezeichnet.

2.3 **Satz** ($\mathcal{T} \subseteq CH0$):
Jede Turing-akzeptierbare Sprache $L \subseteq T^*$ ist eine CHOMSKY-0-Sprache.

Beweis : L werde akzeptiert von einer TM $\tau = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$ mit $F = \{q_e\}$, so dass für alle Konfigurationen $uqv \in K_\tau$ gilt:

$$uqv \text{ ist Endkonfiguration} \iff u = \varepsilon \text{ und } q = q_e.$$

Man sieht leicht ein, dass man zu jeder TM eine TM mit dieser Eigenschaft angeben kann, die die gleiche Sprache akzeptiert. Wir konstruieren jetzt eine CHOMSKY-0-Grammatik $G = (N, T, P, S)$ mit $L(G) = L$ in 4 Schritten.

Schritt 1: (Doppel–Anfangskonfigurationen erzeugen)
Definiere eine Teilgrammatik $G_1 = (N_1, T, P_1, S)$ mit $S, ¢, \$, q_0, \sqcup \in N_1$, so dass für alle $w \in T^*$

$$S \vdash_{G_1}^* w¢\alpha(w)\$$$

gilt und sogar für alle $v \in (N \cup T)^*$:

$$\begin{array}{rl} & S \vdash_{G_1}^* v \text{ und } v \text{ enthält } q_0 \\ \Rightarrow & \exists w \in T^* : v = w¢\alpha(w)\$. \end{array}$$

Dabei ist die Anfangskonfiguration $\alpha(w)$ wie üblich definiert:

$$\alpha(w) = \begin{cases} q_0 w & \text{falls } w \neq \varepsilon \\ q_0 \sqcup & \text{sonst.} \end{cases}$$

Wähle $N_1 = \{S, ¢, \$, q_0, \sqcup, A, B\} \cup \{C_a \mid a \in T\}$ und P_1 wie folgt:

$$\begin{array}{lll} P_1 = \{ \quad S \to ¢q_0\sqcup\$, & & \\ \quad S \to aA¢C_a\$ & \text{für alle } a \in T, & \\ \quad C_a b \to bC_a & \text{für alle } a, b \in T, & \\ \quad C_a\$ \to Ba\$ & \text{für alle } a \in T, & \\ \quad bB \to Bb & \text{für alle } b \in T, & \\ \quad A¢B \to ¢q_0, & & \\ \quad A¢B \to aA¢C_a & \text{für alle } a \in T & \} \end{array}$$

Wirkungsweise von P_1:

$$S \vdash_{G_1} ¢q_0\sqcup\$ = \varepsilon¢\alpha(\varepsilon)\$$$

oder

$$\begin{array}{ll} S \vdash_{G_1}^* & aA¢C_a\$ \\ S \vdash_{G_1}^* & w\ \boxed{A¢B}\ w\$ \\ S \vdash_{G_1}^* & w\ aA¢C_a w\$ \\ S \vdash_{G_1}^* & w\ aA¢wC_a\$ \\ S \vdash_{G_1}^* & w\ aA¢wBa\$ \\ S \vdash_{G_1}^* & w\ a\ \boxed{A¢B}\ wa\$ \\ S \vdash_{G_1}^* & w\ a¢q_0 wa\$ \end{array}$$

Schritt 2: (Transitionsrelation $\vdash_\tau$ simulieren)
Definiere $G_2 = (N_2, T, P_2, S)$ mit $N_2 = \{S, ¢, \$\} \cup Q \cup \Gamma$ so, dass für alle $w \in T^*, v \in \Gamma^*$ gilt:

$$\alpha(w) \vdash_\tau{}^* q_e v \quad \Longleftrightarrow \quad w¢\alpha(w)\$ \vdash_{G_2}^* w¢q_e v\$.$$

Wir können nicht einfach $P_2 = \vdash_\tau$ wählen, weil $\vdash_\tau$ unendlich ist. Z.B. folgt aus $\delta(q,a) = (q', a', S)$ für alle $u, v \in \Gamma^*$: $uqav \vdash_\tau uq'a'v$. Wir können uns aber am endlichen δ orientieren und eine Regel der Form

$$qa \to q'a'$$

wählen. Genauer definieren wir:

$$\begin{aligned}
P_2 = \quad & \{qa \to q'a' && \mid q, q' \in Q \text{ und } a, a' \in \Gamma \text{ und } \delta(q,a) = (q', a', S) && \} \\
\cup\ & \{qab \to a'q'b && \mid q, q' \in Q \text{ und } a, a', b \in \Gamma \text{ und } \delta(q,a) = (q', a', R) && \} \\
\cup\ & \{ \underbrace{qa\$}_{\text{TM steht vor rechtem Rand}} \to a'q'⊔\$ && \mid q, q' \in Q \text{ und } a, a' \in \Gamma \text{ und } \delta(q,a) = (q', a', R) && \} \\
\cup\ & \{bqa \to q'ba' && \mid q, q' \in Q \text{ und } a, a', b \in \Gamma \text{ und } \delta(q,a) = (q', a', L) && \} \\
\cup\ & \{ \underbrace{¢qa}_{\text{TM steht vor linkem Rand}} \to ¢q'⊔a' && \mid q, q' \in Q \text{ und } a, a' \in \Gamma \text{ und } \delta(q,a) = (q', a', L) && \}
\end{aligned}$$

Schritt 3: (Endkonfiguration löschen)
Definiere $G_3 = (N_3, T, P_3, S)$ mit $N_3 = \{S, ¢, \$, q_e, ⊔, D\}$ so, dass für alle $w \in T^*, v \in \Gamma^*$ gilt:

$$w¢q_e v\$ \vdash_{G_3}^* w.$$

Wähle P_3 wie folgt:

$$\begin{aligned}
P_3 = \{ \quad & ¢q_e \to D, \\
& Da \to D \quad \text{für alle } a \in \Gamma, \\
& D\$ \to \varepsilon \quad \}
\end{aligned}$$

Wirkungsweise:

$$w¢q_e v\$ \vdash_{G_3} wDv\$ \vdash_{G_3}^* wD\$ \vdash_{G_3} w.$$

Schritt 4: (G aus G_1, G_2, G_3 zusammenstellen)
Definiere jetzt $G = (N, T, P, S)$ wie folgt:

$$\begin{aligned}
& N = N_1 \cup N_2 \cup N_3, \\
& P = P_1 \dot\cup P_2 \dot\cup P_3 \qquad \text{(disjunkte Vereinigung).}
\end{aligned}$$

Dann gilt für alle $w \in T^*, v \in \Gamma^*$:

$$\begin{aligned}
\alpha(w) \vdash_\tau{}^* q_e v \quad \Longleftrightarrow \quad & S \vdash_G^* w¢\alpha(w)\$ && \text{(Regeln } P_1\text{)} \\
& \vdash_G^* w¢q_e v\$ && \text{(Regeln } P_2\text{)} \\
& \vdash_G^* w && \text{(Regeln } P_3\text{)}
\end{aligned}$$

Für „$\Leftarrow$“ beachte man, dass auf ein vorgegebenes Wort über $N \cup T$ höchstens aus einer der drei Regelmengen P_1, P_2 oder P_3 eine Regel angewandt werden kann.

Insgesamt erhalten wir $L(G) = L$. □

2.4 **Korollar** : Die Funktion $h : T^* \xrightarrow{part} T^*$ werde von einer Turingmaschine berechnet. Dann ist der *Graph von h*, d.h. die Menge

$$L = \{w\#h(w) \mid w \in T^* \text{ und } h(w) \text{ ist definiert } \},$$

eine CHOMSKY-0-Sprache.

Beweis : Wenn h von einer TM τ berechnet wird, so gibt es eine 2-Band TM τ', die L akzeptiert. Die TM τ' arbeitet wie folgt:

(1) τ' belässt ein vorgegebenes Eingabewort der Form $w\#v$ unverändert auf dem 1. Band.

(2) τ' kopiert den Anteil w auf das anfangs leere 2. Band und simuliert dann τ auf diesem Band.

(3) Falls τ hält, wird das Ergebnis $h(w)$ der Endkonfiguration mit dem Anteil v des 1. Bandes verglichen. Falls $h(w) = v$ gilt, akzeptiert τ' die Eingabe $w\#v$ auf dem 1. Band. Anderenfalls akzeptiert τ' die Eingabe $w\#v$ auf dem 1. Band nicht.

Nach dem obigen Satz ist L damit eine CHOMSKY-0-Sprache. □

2.5 **Satz** (*CH0* $\subseteq \mathcal{T}$):
Jede CHOMSKY-0-Sprache $L \subseteq T^*$ ist Turing-akzeptierbar.

Beweis : L werde von einer CHOMSKY-0-Grammatik $G = (N, T, P, S)$ erzeugt: $L(G) = L$.

Wir konstruieren eine *nichtdeterministische 2-Band TM* τ, die L akzeptiert. Die TM τ arbeitet wie folgt:

(1) τ belässt ein vorgegebenes Eingabewort $w \in T^*$ unverändert auf dem 1. Band.

(2) Auf dem anfangs leeren 2. Band erzeugt τ schrittweise Wörter über $N \cup T$ gemäß den Regeln aus P, beginnend mit dem Startwort S. In jedem Schritt wählt τ nichtdeterministisch ein Teilwort u aus dem zuletzt erzeugten Wort und eine Regel $u \to v$ aus P und ersetzt dann u durch v. Entsteht in irgendeinem Schritt das Eingabewort w, so wird w akzeptiert.

Damit gilt: τ akzeptiert L. □

Neben den allgemeinen CHOMSKY-0-Grammatiken $G = (N, T, P, S)$, wo für Regeln $p \to q \in P$ lediglich verlangt wird, dass $p, q \in (N \cup T)^*$ gilt und p mindestens ein Nichtterminalsymbol enthält, gibt es weitere Klassen von Grammatiken.

2.6 **Definition** : Eine CHOMSKY-0-Grammatik (oder kurz *CH0-Grammatik*) $G = (N, T, P, S)$ heißt (für ε siehe unten)

(i) *kontextsensitiv* (CHOMSKY-1- oder kurz *CH1-Grammatik*) gdw.
$\forall\, p \to q \in P\ \exists\, \alpha \in N, u, v, w \in (N \cup T)^*, w \neq \varepsilon :\ p = u\alpha v \wedge q = uwv$
(d.h. ein Nichtterminalsymbol $\alpha \in N$ wird „im Kontext“ $u \ldots v$ ersetzt durch das nichtleere Wort w).

(ii) *kontextfrei* (CHOMSKY-2- oder kurz *CH2-Grammatik*) gdw.
$\forall\, p \to q \in P :\ p \in N \quad (q = \varepsilon$ ist zugelassen).

(iii) *rechtslinear* (CHOMSKY-3- oder kurz *CH3-Grammatik*) gdw.
$\forall\, p \to q \in P :\ p \in N \wedge q \in T^* \cdot N \cup T^*$

Für kontextsensitive Grammatiken $G = (N, T, P, S)$ gibt es folgende Sonderregelung, damit $\varepsilon \in L(G)$ möglich wird: $S \to \varepsilon$ darf als *einzige* ε-Produktion auftreten. Alle weiteren Ableitungen starten dann mit einem neuen Nichtterminalsymbol S':
$S \to S'$
$S' \to \ldots$

2.7 **Definition** : Eine Sprache L heißt *kontextsensitiv, kontextfrei, rechtslinear*, wenn es eine Grammatik vom entsprechenden Typ gibt, die L erzeugt.

Für irgendein Alphabet T seien die Sprachklassen definiert:

CH0	=	Klasse	der	Sprachen,	die	von	Chomsky-0-	Grammatiken	erzeugt	werden,		
CS	=	“	“	“ ,	“	“	kontextsensitiven	“	“	“ ,		
CF	=	“	“	“ ,	“	“	kontextfreien	“	“	“ ,		
RLIN	=	“	“	“ ,	“	“	rechtslinearen	“	“	“ ,		

Auf Grund der Definition gilt bereits (die Inklusion $CF \subseteq CS$ ergibt sich allerdings erst aus späteren Sätzen):

2.8 **Korollar** : $RLIN \subseteq CF \subseteq CS \subseteq CH0$

Wir wollen jetzt zeigen, dass die CHOMSKY-3- (also rechtslinearen) Sprachen genau die endlich akzeptierbaren (also regulären) Sprachen aus Kapitel II sind.

2.9 **Lemma** : Jede endlich akzeptierbare Sprache ist CHOMSKY-3

Beweis : Gegeben sei ein DEA $\mathcal{A} = (\Sigma, Q, \to, q_0, F)$. Konstruiere folgende CHOMSKY-3-Grammatik G:

$$G = (Q, \Sigma, P, q_0),$$

wobei für $q, q' \in Q$ und $a \in \Sigma$ gilt

$$\begin{aligned} P &= \{q \to aq' \mid q \xrightarrow{a} q'\} \\ &\cup \{q \to \varepsilon \mid q \in F\}. \end{aligned}$$

Es ist leicht zu zeigen, dass $L(\mathcal{A}) = L(G)$ gilt. □

Für die Umkehrung benutzen wir folgendes Lemma.

Bemerkung : Jede CHOMSKY-3-Sprache kann von einer Grammatik $G = (N, T, P, S)$ mit $P \subseteq N \times (T \cdot N \cup \{\varepsilon\})$ erzeugt werden.

Beweis : Übung □

2.10 **Lemma** : Jede CHOMSKY-3-Sprache ist endlich akzeptierbar.

Beweis : Gegeben sei eine CHOMSKY-3-Sprache $L \subseteq T^*$. Nach der Bemerkung kann L von einer Grammatik $G = (N, T, P, S)$ mit $P \subseteq N \times (T \cdot N \cup \{\varepsilon\})$ erzeugt werden: $L = L(G)$. Wir konstruieren jetzt folgenden ε-NEA $\mathcal{B}$:

$$\mathcal{B} = (T, N \cup \{q_e\}, \to, S, \{q_e\}),$$

wobei $q_e \notin N$ gelte und die Transitionsrelation $\to$ für $\beta, \gamma \in N$ und $a \in T$ wie folgt definiert ist:

- $\beta \xrightarrow{a} \gamma$ gdw $\beta \to \alpha\gamma \in P$
- $\beta \xrightarrow{\varepsilon} q_e$ gdw $\beta \to \varepsilon \in P$

Es ist wieder leicht zu zeigen, dass $L(G) = L(\mathcal{B})$ gilt. □

Aus diesen beiden Lemmata erhalten wir:

2.11 **Satz (CHOMSKY-3 = DEA)**: Eine Sprache ist genau dann CHOMSKY-3 (rechtslinear), wenn Sie endlich akzeptierbar (regulär) ist.

Zusammenfassung des Kapitels

Wir haben folgendes Ergebnis bewiesen:

2.12 **Hauptsatz** : Für Funktionen $f : A^* \xrightarrow{part} B^*$ sind äquivalent

- f ist Turing-berechenbar.
- Der Graph von f ist eine CHOMSKY-0-Sprache.

In der Literatur wird zusammenfassend von „Rekursivität“ gesprochen.

2.13 **Definition** : Eine *Funktion* $f : A^* \xrightarrow{part} B^*$ heißt *rekursiv*, falls f Turing-berechenbar bzw. der Graph von f eine CHOMSKY-0-Sprache ist.

Eine Menge $L \subseteq A^*$ heißt *rekursiv*, falls $\chi_L : A^* \to \{0, 1\}$ rekursiv ist, d.h. falls L Turing-entscheidbar ist.

Bemerkung : Manchmal wird bei partiell definierten, rekursiven Funktionen explizit von „partiell rekursiven“ Funktionen gesprochen; dann versteht man unter „rekursiv“ nur die total rekursiven Funktionen. Man überzeuge sich daher stets, was genau mit „rekursiv“ gemeint ist.

Die **Churchsche These (1936)** besagt: Die mit Hilfe von Algorithmen intuitiv berechenbaren Funktionen sind genau die rekursiven Funktionen, d.h. „Rekursivität“ ist die mathematische Präzisierung des Begriffs „Algorithmus“.

Die Churchsche These lässt sich formal nicht beweisen, da „intuitiv berechenbar“ kein mathematisch präziser Begriff ist, sondern nur duch Indizien belegen.

Diese Indizien sind allerdings so gewichtig, dass die Churchsche These allgemein akzeptiert wird. Insbesondere gilt:

- In mehr als 50 Jahren Erfahrung mit berechenbaren Funktionen wurden keine Gegenbeispiele gefunden.
- Jede weitere Präzisierung des Begriffs Algorithmus hat sich als äquivalent zur Rekursivität erwiesen: weitere Beispiele für solche Präzisierung siehe unten.
- Rekursive Funktionen haben Eigenschaften, z.B. Abgeschlossenheit gegenüber dem μ-Operator (while-Schleifen), die man auch Algorithmen zubilligen muss.

Deshalb auch folgender lockerer Sprachgebrauch für Funktionen f und Mengen L:

- f ist „berechenbar“ bedeutet f ist partiell rekursiv.
- L ist „entscheidbar“ bedeutet L ist rekursiv.

Neben den bereits besprochenen Präzisierungen des Begriffs „berechenbar“ gibt es weitere, ebenfalls äquivalente Präzisierungen:

- **μ-rekursive Funktionen:**
  Nach Arbeiten von Gödel, Herbrand, Kleene und Ackermann wird die Menge aller berechenbaren Funktionen $f : \mathbb{N}^n \xrightarrow{part} \mathbb{N}$ induktiv definiert. Hierzu werden zunächst sehr einfache Basisfunktionen (wie Nachfolgerfunktion, Projektion, Kontsantenfunktion) eingeführt. Dann wird definiert, wie aus bereits bekannten Funktionen neue Funktionen (mit den Prinzipien der Komposition, primitiven Rekursion und μ-Rekursion) gewonnen werden können.

- **Registermaschinen mit $k \geq 2$ Registern:**
  Jedes Register enthält eine natürliche Zahl. Die Maschine kann jedes dieser Register testen, ob sein Inhalt gleich 0 oder größer ist. In Abhängigkeit dieses Tests und des momentanen Zustandes kann die Maschine den Wert der Register

  – unverändert lassen
  – um 1 erhöhen
  – um 1 vermindern, sofern keine negative Zahl entsteht.

  Bemerkenswert ist, dass bereits 2 Registern zur Berechnung aller berechenbaren Funktionen $f : \mathbb{N} \xrightarrow{part} \mathbb{N}$ (bis auf Codierung) ausreichen. 2-Registermaschinen werden auch 2-*Zählermaschinen* oder Minsky-*Maschinen* genannt.

- **λ-Kalkül:**
  Dieser von Church eingeführte Kalkül beschreibt das Rechnen mit höheren Funktionen, alsoFunktionen, die selbst Funktionen als Parameter benutzen. Der λ-Kalkül ist heute die Grundlage für die Semantik funktionaler Sprachen.

- **Prädikatenkalkül 1.Stufe:**
  Der Erfüllbarkeitsbegriff von Formeln erweist sich als gleichwertig zur Berechenbarkeit.

Kapitel V

Nicht berechenbare Funktionen — Unentscheidbare Probleme

§1 Existenz nicht berechenbarer Funktionen

In diesem Abschnitt betrachten wir total definierte Funktionen $f : \mathbb{N} \to \mathbb{N}$.

1.1 **Frage von** Gödel, Turing, Church **1936** :

Ist jede Funktion $f : \mathbb{N} \to \mathbb{N}$ algorithmisch berechenbar?
Präziser gefragt: Gibt es eine Turingmaschine, die f berechnet, bzw. ist der Graph von f eine Chomsky-0-Sprache? Anders formuliert: Ist f rekursiv?

Die Antwort lautet „nein“. Wir werden im Folgenden zeigen, dass es nicht berechenbare Funktionen $f : \mathbb{N} \to \mathbb{N}$ gibt. Genauer:

1.2 **Satz (Existenz nicht-berechenbarer Funktionen)**:
Es gibt Funktionen $f : \mathbb{N} \to \mathbb{N}$, die nicht Turing-berechenbar sind.

Beweisidee : Die Menge *aller* Funktionen ist *mächtiger* (nämlich überabzählbar) als die Menge der Turing-*berechenbaren* Funktionen (die abzählbar ist).

Wir benötigen einige Vorbereitungen, um diesen Satz zu beweisen. Deshalb stellen wir Begriffe und Ergebnisse aus der Theorie der Mengen zusammen.

1.3 **Definition** : M und N seien Mengen. Dann definieren wir

1. $M \sim N$ (M ist *gleichmächtig* zu N), falls $\exists$ Bijektion $\beta : M \to N$
2. $M \preceq N$(M ist *höchstens so mächtig wie* N oder N ist *mindestens so mächtig wie* M), falls $\exists N' \subseteq N : M \sim N'$, das heißt M ist gleichmächtig zu einer Teilmenge von N.
3. $M \prec N$ (N ist *mächtiger* als M), falls $M \preceq N$ und $M \not\sim N$.
4. M ist (höchstens) *abzählbar* , falls $M \preceq \mathbb{N}$.
5. M ist *überabzählbar* oder *nicht abzählbar*, falls $\mathbb{N} \prec M$.
6. M ist *endlich* , falls $M \prec \mathbb{N}$.
7. M ist *unendlich*, falls $\mathbb{N} \preceq M$

Bemerkung :

- $M \preceq N \Leftrightarrow \exists$ Injektion $\beta : M \to N$.
- Jede endliche Menge ist abzählbar.
- Jede überabzählbare Menge ist unendlich.

1.4 **Satz (**Schröder-Bernstein**)**: Aus $M \preceq N$ und $N \preceq M$ folgt $M \sim N$.

Beweis : Siehe z.B. P.R. Halmos, Naive Mengenlehre. □

1.5 **Korollar** : Eine unendliche Menge M ist abzählbar gdw. $\mathbb{N} \sim M$, d.h. es gilt $M = \{\beta(0), \beta(1), \beta(2), \ldots\}$ für eine Bijektion $\beta : \mathbb{N} \to M$.

Beispiele : Die folgenden Mengen sind abzählbar:

- $\mathbb{N}$,
- $\{n \in \mathbb{N} \mid n \text{ gerade}\}$ und
- $\mathbb{N} \times \mathbb{N}$ (vgl. Beweis von Lemma 1.6).

Dagegen sind folgende Mengen überabzählbar:

- $\mathcal{P}(\mathbb{N})$,
- $\mathbb{N} \to \mathbb{N}$ (Menge der Funktionen von $\mathbb{N}$ nach $\mathbb{N}$: s. Lemma 1.7).

Wir zeigen zunächst:

1.6 **Lemma** :

Die Menge der TURING-berechenbaren Funktionen $f : \mathbb{N} \to \mathbb{N}$ ist abzählbar.

Beweis : Wir gehen aus von einer abzählbaren Menge $\{q_0, q_1, q_2, \ldots\}$ von Zuständen und einer hierzu disjunkten abzählbaren Menge $\{\gamma_0, \gamma_1, \gamma_2, \ldots\}$ von Bandsymbolen mit $\gamma_0 = \sqcup$ und $\gamma_1 = |$.

Jede TURING-berechenbare Funktion $f : \mathbb{N} \to \mathbb{N}$ kann von einer Turingmaschine der Form

$$\begin{aligned} \tau &= (Q, \{|\}, \Gamma, \delta, q_0, \sqcup) \text{ mit} \\ Q &= \{q_0, \ldots, q_k\} \text{ für ein } k \geq 0, \\ \Gamma &= \{\gamma_0, \ldots, \gamma_l\} \text{ für ein } l \geq 1 \end{aligned}$$

berechnet werden. Sei $\mathcal{T}$ die Menge aller dieser Turingmaschinen .

Es genügt zu zeigen: $\mathcal{T}$ ist abzählbar, da die Menge der TURING-berechenbaren Funktionen nicht größer als die Menge zugehöriger Turingmaschinen sein kann.

Definiere für $k \geq 0, l \geq 1$:

$\mathcal{T}_{k,l} = \{\tau \in \mathcal{T} \mid \tau \text{ hat } k+1 \text{ Zustände und } l+1 \text{ Bandsymbole}\}$.

(Bemerkung: "+1"bei k und l, da die Zustände und Bandsymbole von 0 an gezählt werden.)

Für jedes $\tau \in \mathcal{T}_{k,l}$ gilt also

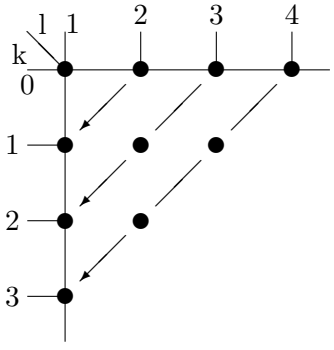
$$\tau = (\underbrace{\{q_0, \ldots, q_k\}}_{Q}, \{|\}, \underbrace{\{\gamma_0, \ldots, \gamma_l\}}_{\Gamma}, \delta, q_0, \sqcup),$$

wobei δ eine der endlich vielen Funktionen

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{R, L, S\} \text{ ist.}$$

Daher gilt: $\mathcal{T}_{k,l}$ ist endlich.

Da $\mathcal{T} = \bigcup_{\substack{k \geq 0 \\ l \geq 1}} \mathcal{T}_{k,l}$ gilt, ist $\mathcal{T}$ abzählbar durch eine Bijektion $\beta : \mathbb{N} \to \mathcal{T}$, die nach folgendem *Diagonalschema* zur Anordnung der Mengen $\mathcal{T}_{k,l}$ definiert ist:



β zählt die Turingmaschinen in $\mathcal{T}$ in folgender Reihenfolge ab:

1. Diagonale: alle Turingmaschinen in $\mathcal{T}_{0,1}$;
2. Diagonale: alle Turingmaschinen in $\mathcal{T}_{0,2}$, dann in $\mathcal{T}_{1,1}$;
3. Diagonale: alle Turingmaschinen in $\mathcal{T}_{0,3}$, dann in $\mathcal{T}_{1,2}$,
dann in $\mathcal{T}_{2,1}$;
usw.

Damit ist Lemma 1.6 bewiesen. □

Bemerkung : Die Grundidee zur Abzählbarkeit von $\mathcal{T}$ ist also die Abzählbarkeit der Menge aller Paare $\{(k,l) \mid k \geq 0, l \geq 1\}$ nach dem Diagonalschema. Dieses Schema geht auf G. Cantors Beweis zur Abzählbarkeit von $\mathbb{N} \times \mathbb{N}$ zurück.

Dagegen gilt:

1.7 **Lemma** :

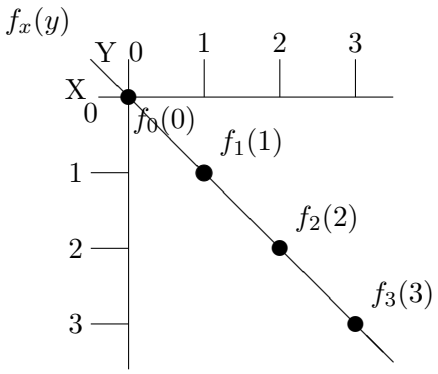
Die Menge *aller* Funktionen $f : \mathbb{N} \to \mathbb{N}$ ist überabzählbar.

Beweis :

Sei $\mathcal{F} = \{f \mid f : \mathbb{N} \to \mathbb{N}\}$.
Annahme : $\mathcal{F}$ ist abzählbar, d.h. es gibt eine Bijektion $\beta : \mathbb{N} \to \mathcal{F}$, so dass $\mathcal{F} = \{f_0, f_1, f_2, \ldots\}$ gilt mit $f_n = \beta(n)$ für $n \in \mathbb{N}$. Definiere jetzt $g \in \mathcal{F}$ durch $g(n) = f_n(n) + 1$ für alle $n \in \mathbb{N}$. Nach der Annahme gibt es einen Index $m \in \mathbb{N}$ mit $g = f_m$. Dann gilt aber $f_m(m) = g(m) = f_m(m)+1$. Widerspruch! □

Bemerkung :

Wir benutzen hier das *Cantors Diagonalverfahren*, d.h. es werden die Werte von $f_x(y)$ auf der „Diagonalen“ $x = y$ betrachtet; der Widerspruch entsteht dann an einer Stelle m :



Aus den Lemmata 1.6 und 1.7 folgt der Satz über die Existenz nicht berechenbarer Funktionen.

Diese Existenzaussage wirft folgende Frage auf:

Gibt es für die Informatik wichtige Funktionen, die nicht berechenbar sind?

§2 Konkrete unentscheidbare Probleme: Halten von Turingmaschinen

Im Folgenden betrachten wir stets das „binäre“ Alphabet $B = \{0, 1\}$. Wir geben jetzt Mengen (Sprachen) $L \subseteq B^*$ an, die unentscheidbar sind, für die also das Problem:

Gegeben : $w \in B^*$
Frage : Gilt $w \in L$?

nicht algorithmisch lösbar ist.

Als erstes wollen wir das *Halteproblem für Turingmaschinen* mit dem Eingabealphabet B betrachten:

Gegeben : Turingmaschine τ und Wort $w \in B^*$
Frage : Hält τ angesetzt auf w, d.h. ist $h_\tau(w)$ definiert?

Informatik-Relevanz dieses Problems: Ist es entscheidbar, ob ein gegebenes Programm für gegebene Eingabewerte terminiert?

Zunächst übersetzen wir das Halteproblem in eine passende Sprache $L \subseteq B^*$. Dazu benutzen wir eine Binärkodierung für Turingmaschinen. Wir gehen aus von abzählbaren Mengen

- $\{q_0, q_1, q_2, \ldots\}$ von Zuständen und
- $\{\gamma_0, \gamma_1, \gamma_2, \ldots\}$ von Bandsymbolen, jetzt mit $\gamma_0 = 0, \gamma_1 = 1, \gamma_2 = \sqcup$.

Wir betrachten nur Turingmaschinen $\tau = (Q, B, \Gamma, \delta, q_0, \sqcup)$ mit $Q = \{q_0, \ldots, q_k\}$ für ein $k \geq 0$, $\Gamma = \{\gamma_0, \ldots, \gamma_l\}$ für ein $l \geq 2$.

2.1 **Definition** : Die *Standardkodierung* einer solchen Turingmaschine τ ist folgendes Wort w_τ über $\mathbb{N} \cup \{\sharp\}$:

$w_\tau \;=\; k \,\sharp\, l$
.
.
.
$\boxed{\sharp\ i\ \sharp\ j\ \sharp\ s\ \sharp\ t\ \sharp\ nr(P)}$ für $\delta(q_i, \gamma_j) = (q_s, \gamma_t, P)$
.
.
.

wobei $P \in \{R, L, S\}$ und $nr(R) = 0, nr(L) = 1, nr(S) = 2$ gilt und die Teilworte der Form $\boxed{\ldots}$ von w_τ in der Reihenfolge der lexikographischen Ordnung der Paare (i, j) aufgeschrieben sind.

Die *Binärkodierung* von τ ist dann das Wort $bw_\tau \in B^*$, das aus w_τ entsteht, indem folgende Ersetzung vorgenommen wird:

$\sharp \to \varepsilon$
$n \to 01^{n+1}$ für $n \in \mathbb{N}$

Beispiel :

Gegeben sei die kleine Rechtsmaschine $r = (\{q_0, q_1\}, B, B \cup \{\sqcup\}, \delta, q_0, \sqcup)$ mit der TURING-Tafel:

δ:				
q_0	0	q_1	0	R
q_0	1	q_1	1	R
q_0	$\sqcup$	q_1	$\sqcup$	R

Dann ist

$w_\tau = 1\ \sharp\ 2$
$\sharp\ 0\ \sharp\ 0\ \sharp\ 1\ \sharp\ 0\ \sharp\ 0$
$\sharp\ 0\ \sharp\ 1\ \sharp\ 1\ \sharp\ 1\ \sharp\ 0$
$\sharp\ 0\ \sharp\ 2\ \sharp\ 1\ \sharp\ 2\ \sharp\ 0$

und

$bw_\tau = 011\ 0111$
$01\ 01\ 011\ 01\ 01$
$01\ 011\ 011\ 011\ 01$
$01\ 0111\ 011\ 0111\ 01$

Bemerkung :

1. Nicht jedes Wort $w \in B^*$ ist Binärkodierung einer Turingmaschine , z.B. $w = \varepsilon$ nicht.
2. Es ist entscheidbar, ob ein gegebenes Wort $w \in B^*$ Binärkodierung einer Turingmaschine ist, d.h. die Sprache
   $BW = \{w \in B^* \mid \exists\, Turingmaschine\ \tau : w = bw_\tau\}$ ist entscheidbar.
3. Die Binärkodierung ist injektiv, d.h. aus $bw_{\tau_1} = bw_{\tau_2}$ folgt $\tau_1 = \tau_2$.

2.2 **Definition (spezielles Halteproblem oder Selbstanwendungsproblem für Turingmaschinen)**: Das *spezielle Halteproblem* oder *Selbstanwendungsproblem für Turingmaschinen* ist die Sprache

$$K = \{bw_\tau \in B^* \mid \tau \text{ angesetzt auf } bw_\tau \text{ hält}\}.$$

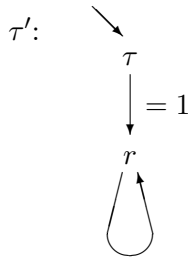
2.3 **Satz** : $K \subseteq B^*$ ist unentscheidbar.

Beweis : Wir wenden wieder das Cantorsche Diagonalverfahren in einem Widerspruchsbeweis an.

Annahme: $K \subseteq B^*$ ist entscheidbar. Dann existiert eine Turingmaschine τ, die $\chi_K : B^* \to \{0,1\}$ berechnet. Dabei ist bekanntlich

$$\chi_K(w) = \begin{cases} 1 & \text{falls } w \in K \\ 0 & \text{sonst} \end{cases}$$

Wir ändern τ wie folgt zu einer Turingmaschine τ' ab. In Flussdiagrammschreibweise ist



Es gilt also: τ' gerät in eine Endlosschleife, falls τ mit 1 hält, und τ' hält (mit 0), falls τ mit 0 hält.

Dann gilt:

$$\begin{aligned} \tau' \text{ angesetzt auf } bw_{\tau'} \text{ hält} &\Leftrightarrow \tau \text{ angesetzt auf } bw_{\tau'} \text{hält mit 0} \\ &\Leftrightarrow \chi_K(bw_{\tau'}) = 0 \\ &\Leftrightarrow bw_{\tau'} \notin K \\ &\Leftrightarrow \tau' \text{ angesetzt auf } bw_{\tau'} \text{ hält nicht.} \\ &\quad \text{Widerspruch!} \end{aligned}$$

□

Wir geben jetzt eine allgemeine Technik an, mit der wir aus bekanntermaßen unentscheidbaren Problemen (wie z.B. K) weitere Probleme als unentscheidbar nachweisen können, die sogenannte *Reduktion*.

2.4 **Definition (Reduktion)**: Seien $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ Sprachen. Dann heißt L_1 *auf* L_2 *reduzierbar*, abgekürzt $L_1 \leq L_2$, falls es eine totale, berechenbare Funktion $f : \Sigma_1^* \to \Sigma_2^*$ gibt, so dass für alle $w \in \Sigma_1^*$ gilt : $w \in L_1 \Leftrightarrow f(w) \in L_2$. Wir schreiben auch: $L_1 \leq L_2$ *mittels* f.

Anschauung: L_1 ist ein *Spezialfall* von L_2.

2.5 **Lemma (Reduktion)**: Sei $L_1 \leq L_2$. Dann gilt:

(i) Falls L_2 entscheidbar ist, so auch L_1.

(ii) Falls L_1 unentscheidbar ist, so auch L_2

Beweis : Da (ii) die Kontraposition von (i) ist, genügt es, (i) zu zeigen. Sei also $L_1 \leq L_2$ mittels der totalen, berechenbaren Funktion f und sei L_2 entscheidbar, also χ_{L_2} berechenbar. Dann ist die *Komposition* $\chi_{L_2}(f) = f \circ \chi_{L_2}$ (erst f anwenden, dann χ_{L_2}) berechenbar. Es gilt

$$\chi_{L_1} = f \circ \chi_{L_2},$$

denn für alle $w \in \Sigma_1$ ist

$$\chi_{L_1}(w) = \left\{ \begin{array}{lll} 1 & \text{falls} & w \in L_1 \\ 0 & \text{sonst} & \end{array} \right\} = \left\{ \begin{array}{lll} 1 & \text{falls} & f(w) \in L_2 \\ 0 & \text{sonst} & \end{array} \right\} = \chi_{L_2}(f(w))$$

Also ist χ_{L_1} berechenbar, d.h. L_1 ist entscheidbar. □

Wir wenden die Reduktions-Technik auf das allgemeine Halteproblem für Turingmaschine an.

2.6 **Definition (allgemeines Halteproblem für TM)**: Das *(allgemeine) Halteproblem* für Turingmaschinen ist die Sprache

$$H = \{bw_\tau 00u \in B^* \mid \tau \text{ angesetzt auf } u \text{ hält } \}.$$

2.7 **Satz** : $H \subseteq B^*$ ist unentscheidbar.

Beweis : Nach dem Reduktions-Lemma genügt es, $K \leq H$ zu zeigen, d.h. K ist ein Spezialfall von H. Das ist hier trivial: Wähle $f : B^* \to B^*$ mit $f(w) = w00w$. Dann ist f berechenbar und es gilt:

$$bw_\tau \in K \Leftrightarrow f(bw_\tau) \in H.$$

□

2.8 **Definition** : Das *Halteproblem für Turingmaschinen auf dem leeren Band* ist die Sprache

$$H_0 = \{bw_\tau \in B^* \mid \tau \text{ angesetzt auf das leere Band hält}\}.$$

2.9 **Satz** : H_0 ist unentscheidbar.

Beweis : Wir zeigen $H \leq H_0$. Wir beschreiben zunächst zu vorgegebener Turingmaschine τ und vorgegebenem Wort $u \in B^*$ eine Turingmaschine τ_u, die folgendermaßen arbeitet:

- Angesetzt auf das leere Band schreibt τ_u zunächst u auf das Band. Dann arbeitet τ_u wie τ (angesetzt auf u).
- Es ist unerheblich, wie τ_u arbeitet, wenn das Band anfangs nicht leer ist.

τ_u kann man aus τ und u konstruieren. Also gibt es eine berechenbare Funktion $f : B^* \to B^*$ mit

$$f(bw_\tau 00u) = bw_{(\tau_u)}.$$

Dann gilt

$$\begin{aligned} & bw_\tau 00u \in H \\ \Leftrightarrow \quad & \tau \text{ angesetzt auf } u \text{ hält.} \\ \Leftrightarrow \quad & f(bw_\tau 00u) \in H_0. \end{aligned}$$

Also gilt $H \leq H_0$ mittels f. □

Wir haben drei Varianten des Halteproblems für Turnigmaschinen kennengelernt, nämlich K, H und H_0. Alle drei Varianten sind unentscheidbar.

Für K haben wir dieses direkt durch ein *Diagonalverfahren* gezeigt, und für H und H_0 haben wir dieses durch *Reduktion* gezeigt: $K \leq H \leq H_0$.

In allen drei Varianten ging es um die Frage, ob es ein Entscheidungsverfahren gibt, das für *jede* vorgelegte Turingmaschine das Halten entscheidet.

In allen Fällen war die Antwort „nein“.

Vielleicht gibt es wenigstens für jede einzelne Turingmaschine τ ein Entscheidungsverfahren, welches das Halten von τ entscheidet.

2.10 **Definition** : Gegeben sei eine Turingmaschine τ. Das *Halteproblem für* τ ist die Sprache

$$H_\tau = \{w \in B^* \mid \tau \text{ angesetzt auf } w \text{ hält}\}.$$

Für gewisse Turingmaschinen ist H_τ entscheidbar, z.B. gilt für die kleine Rechtsmaschine r

$$H_r = B^*$$

und für die Turingmaschine τ mit :

τ

$$H_\tau = \emptyset.$$

Es gibt aber auch Turingmachinen τ, für die H_τ unentscheidbar ist. Dies sind die „programmierbaren“ oder *universellen Turingmaschinen*.

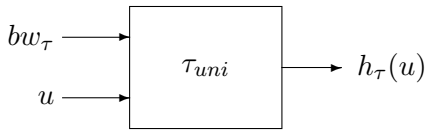
2.11 **Definition** : Eine Turingmaschine τ_{uni} mit dem Eingabealphabet B heißt *universell*, wenn für die von τ_{uni} berechnete Funktion $h_{\tau_{uni}}$ folgendes gilt:

$$h_{\tau_{uni}}(bw_\tau 00u) = h_\tau(u),$$

d.h. τ_{uni} kann die Arbeitsweise jeder anderen Turingmaschine τ bezüglich jedes Eingabewortes $u \in B^*$ simulieren.

Relevanz zur Informatik :

Die Turingmaschine τ_{uni} ist ein *Interpreter* für Turingmaschinen, der selbst als Turingmaschine geschrieben ist:



Die Konstruktion von τ_{uni} entspricht daher dem Schreiben eines Interpreters für eine einfache Programmiersprache in dieser Sprache selbst.

2.12 **Lemma** : Universelle Turingmaschinen τ_{uni} kann man effektiv konstruieren.

Beweis : Anschaulich arbeitet τ_{uni} angesetzt auf ein Wort $w \in B^*$ wie folgt :

- τ_{uni} stellt fest, ob w von der Form $w = bw_\tau 00u$ für eine Turingmaschine τ (mit Eingabealphabet B) und ein Wort $u \in B^*$ ist.
- Wenn nein, so geht τ_{uni} in eine Endlosschleife.
- Wenn ja, so simuliert τ_{uni} die Turingmaschine τ angesetzt auf u.

Letzteres kann genauer so geschehen:

1. τ_{uni} ändert die Bandinschrift ab zu
$$bw_\tau \not\$ q_0 u\$,$$
wobei q_0 der Anfangszustand von τ ist.

2. Im allgemeinen liegt τ_{uni} eine Bandinschrift
$$bw_\tau \not\$ v_l q a v_r \$$$
vor, wobei $v_l q a v_r$ die momentane Konfiguration der Turingmaschine τ darstellt. τ_{uni} merkt sich das Paar (q, a) und stellt in der in bw_τ binärkodierten Übergangsfunktion δ von τ den Wert $\delta(q, a)$ fest.

3. Falls $\delta(q, a)$ definiert ist, läuft τ_{uni} zur Konfiguration $v_l q a v_r$ von τ zurück und führt dann den gewünschten Übergang aus. In der Regel muss τ_{uni} mehrmals hin- und herlaufen, da sie in ihrem Gedächtnis Q nur endlich viel Information speichern kann. Schließlich ergibt sich eine neue Bandinschrift der Form
$$bw_\tau \not\$ v_l' q' a' v_r' \$,$$
und τ_{uni} arbeitet weiter wie unter (2) beschrieben.

4. Falls $\delta(q,a)$ undefiniert ist, also $v_l q a v_r$ eine Endkonfiguration von τ ist, löscht τ_{uni} die Teilworte $bw_\tau\not\in$ und \$ und hält selbst in einer Endkonfiguration der Form

$$v_l q_e a v_r.$$

Offensichtlich sind die Ergebnisse von τ_{uni} und τ gleich:

$$\omega(v_l q a v_r) = \omega(v_l q_e a v_r).$$

Insgesammt gilt:

$$h_{\tau_{uni}}(bw_\tau 00u) = h_\tau(u)$$

wie gewünscht. □

2.13 **Satz** : $H_{\tau_{uni}} \subseteq B^*$ ist unentscheidbar.

Beweis : Nach der Definition von τ_{uni} gilt $H \leq H_{\tau_{uni}}$ mittels $f = id_{B^*}$. Für die im Beweis konstruierte Turingmaschine τ_{uni} gilt sogar $H = H_{\tau_{uni}}$, da τ_{uni} in eine Endlosschleife gerät, falls ein vorgegebenes Wort $w \in B^*$ nicht aus H, also nicht von der Form $w = bw_\tau 00u$ ist. □

Insagesamt haben wir gezeigt: $K \leq H = H_{\tau_{uni}} \leq H_0$

§3 Rekursive Aufzählbarkeit

In diesem Abschnitt geht es um eine Abschwächung des Begriffs Entscheidbarkeit (Rekursivität) für Sprachen L über einem Alphabet A.

3.1 **Definition** : Eine Sprache $L \subseteq A^*$ heißt *(rekursiv) aufzählbar* , abgekürzt *r.a.*, falls $L = \emptyset$ ist oder es eine totale, berechenbare Funktion $\beta : \mathbb{N} \to A^*$ gibt mit

$$L = \beta(\mathbb{N}) = \{\beta(0), \beta(1), \beta(2), \ldots\},$$

d.h. L ist der Wertebereich von β.

Zum Vergleich sei der Begriff "abzählbar" wiederholt. $L \subseteq A^*$ ist *abzählbar*, falls $L \preceq \mathbb{N}$ ist oder äquivalent: falls es eine totale Funktion $\beta : \mathbb{N} \to A^*$ gibt mit

$$L = \emptyset \text{ oder } L = \beta(\mathbb{N}) = \{\beta(0), \beta(1), \beta(2), \ldots\}.$$

Der Unterschied ist also, dass bei "abzählbar" die Funktion β nicht berechenbar zu sein braucht.

Wir wollen den neuen Begriff der rekursiven Aufzählbarkeit näher untersuchen.

3.2 **Definition (Semi-Entscheidbarkeit)**:

Eine Sprache $L \subseteq A^*$ heißt *semi-entscheidbar*, falls die *partielle charakteristische Funktion von* L

$$\psi_L : A^* \xrightarrow{part} \{1\}$$

berechenbar ist. Die partielle Funktion ψ_L ist wie folgt definiert:

$$\psi_L(v) = \begin{cases} 1 & \text{falls} \quad v \in L \\ \text{undef.} & \text{sonst} \end{cases}$$

Ein Semi-Entscheidungsverfahren für eine Menge $L \subseteq A^*$ ist also ein "Ja-Verfahren", während ein Entscheidungsverfahren ein "Ja-Nein-Verfahren" ist. Wir halten fest:

Bemerkung : Für alle Sprachen $L \subseteq A^*$ gilt:

1. L ist semi-entscheidbar $\Leftrightarrow$ L ist TURING-akzeptierbar.
2. L ist entscheidbar $\Leftrightarrow$ L und $\overline{L} = A^* - L$ sind TURING-akzeptierbar oder semi-entscheidbar.

Beweis : (1) folgt aus der Def. von "semi-entscheidbar". (2) folgt aus einem entsprechenden Satz über TURING-Akzeptierbarkeit. □

3.3 **Lemma** : Für alle Sprachen $L \subseteq A^*$ gilt: L ist rekursiv aufzählbar $\Leftrightarrow$ L ist semi-entscheidbar.

Beweis : "⇒": Sei L rekursiv aufzählbar mittels der Funktion $\beta : \mathbb{N} \to A^*$. Sei $f : A^* \stackrel{part}{\longrightarrow} \{1\}$ durch folgenden Algorithmus berechnet:

- Eingabe: $w \in A^*$
- Wende β sukzessive auf $n = 0, 1, 2, \ldots$ an.
- Falls irgendwann $\beta(n) = w$ gilt, halte mit Ausgabe 1. (Sonst hält der Algorithmus nicht an.)

Es gilt $f = \psi_L$. Also ist L semi-entscheidbar.

"⇐" : (dove tailing)

Sei L semi-entscheidbar mittels der Turingmaschine τ. Falls $L \neq \emptyset$, haben wir eine totale, berechenbare Funktion $\beta : \mathbb{N} \to A^*$ mit $\beta(\mathbb{N}) = L$ anzugeben.

Zur Vorbereitung benötigen wir eine Primzahlkodierung von Wörtern und Tupeln.
Um *Wörter* über A als natürliche Zahlen darzustellen, benutzen wir folgende *Primzahlkodierung.* Sei $A = \{a_1, \ldots, a_M\}$ und $nr : A \longrightarrow \mathbb{N}$ eine injektive Abbildung mit $nr(a_i) = i$ für $i = 1, \ldots, M$, die die Nummer des Symbols a_i liefert. Es sei p_j die j-te Primzahl, also

$$p_1 = 2, p_2 = 3, p_3 = 5, p_4 = 7, p_5 = 11, \ldots$$

Dann ist die Primzahlkodierung von A^* die Funktion $\pi : A^* \longrightarrow \mathbb{N}$ mit

$$\pi(\varepsilon) = 1,$$
$$\pi(x_1 \ldots x_r) = p_1^{nr(x_1)} \cdot \ldots \cdot p_r^{nr(x_r)} = \prod_{j=1}^{r} p_j^{nr(x_j)}.$$

Bemerkung: π ist injektiv, da die Primzahlzerlegung natürlicher Zahlen eindeutig ist.

Beispiel: Die Zahl 720 ist die Primzahlkodierung des Wortes $a_4a_2a_1$, weil

$$\pi(a_4a_2a_1) = 2^4 \cdot 3^2 \cdot 5^1 = 720$$

gilt. Wir führen auch eine Primzahlkodierung für *n-Tupel* natürlicher Zahlen ein. Diese ist definiert durch die Funktion

$$\pi^n : \mathbb{N}^n \longrightarrow \mathbb{N}$$

mit

$$\pi^n(x_1, \ldots, x_n) = p_1^{x_1} \cdots p_n^{x_n}.$$

Jetzt setzen wir den Beweis fort. Sei w_0 irgendein festgewähltes Wort aus L, werde β durch folgenden Algorithmus berechnet:

- Eingabe : $n \in \mathbb{N}$
- Stelle fest, ob n Primzahlkodierung eines Paares $(w, k) \in A^* \times \mathbb{N}$ ist, d.h. ob $n = \pi^2(\pi(w), k)$ gilt.
- Wenn nein, dann Ausgabe: w_0.
- Andernfalls stelle fest, ob τ angesetzt auf w in maximal k Schritten hält (also den Wert 1, d.h. "$w \in L$" ausgibt).
  Wenn ja, dann Ausgabe: w
  Sonst Ausgabe: w_0

Wir zeigen: $\beta(\mathbb{N}) = L$.
"$\subseteq$" : Obiger Algorithmus gibt nur Wörter aus L aus.
"$\supseteq$" : Sei $w \in L$. Dann gibt es eine Schrittzahl $k \in \mathbb{N}$, so dass τ angesetzt auf w in k Schritten hält und deshalb 1 ("$w \in L$") ausgibt. Setze $n = \pi^2(\pi(w), k)$. Dann gilt $w = \beta(n) \in \beta(\mathbb{N})$. □

3.4 **Satz (Charakterisierung rekursiver Aufzählbarkeit)**: Für alle Sprachen $L \subseteq A^*$ sind folgende Aussagen äquivalent:

1. L ist rekursiv aufzählbar.
2. L ist Ergebnisbereich einer Turingmaschine τ, d.h. $L = \{v \in A^* \mid \exists w \in \Sigma^* \text{ mit } h_\tau(w) = v\}$.
3. L ist semi-entscheidbar.
4. L ist Haltebereich einer Turingmaschine τ, d.h. $L = \{v \in A^* \mid h_\tau(v) \text{ existiert}\}$.
5. L ist TURING-akzeptierbar.
6. L ist Chomsky-0.

3.5 **Korollar** : Für alle Sprachen $L \subseteq A^*$ gilt:

L ist entscheidbar (rekursiv) $\Leftrightarrow$ L und $\overline{L} = A^* - L$ sind rekursiv aufzählbar.

Für Reduktionen gilt folgende Erweiterung des Reduktions-Lemmas:

3.6 **Lemma** : Sei $L_1 \leq L_2$. Dann gilt : Falls L_2 rekursiv aufzählbar ist, so auch L_1.

Beweis : Sei $L_1 \leq L_2$ mittels f. Dann gilt $\psi_{L_1} = f \circ \psi_{L_2}$ (erst f anwenden, dann ψ_{L_2}). □

Wir zeigen jetzt, dass die Halteprobleme für Turingmaschinen rekursiv aufzählbar sind.

3.7 **Satz** : $H_0 \subseteq B^*$ ist rekursiv aufzählbar.

Beweis : H_0 ist semi-entscheidbar durch die Turingmaschine τ_0, die angesetzt auf $w \in B^*$ folgendermaßen arbeitet:

- τ_0 stellt fest, ob w von der Form $w = bw_\tau$ für eine Turingmaschine τ ist.
- Wenn nein, so geht τ_0 in eine Endlosschleife.
- Wenn ja, so lässt τ_0 die Turingmaschine τ angesetzt auf das leere Band laufen. Falls τ hält, so gibt τ_0 den Wert 1 aus. Sonst läuft τ_0 unendlich weiter.

□

Wir erhalten damit folgendes Hauptergebnis über das Halten von Turingmaschinen.

3.8 **Hauptsatz (Halten von Turingmaschinen)**:

1. Die Halteprobleme K, H, H_0 und $H_{\tau_{uni}}$ sind rekursiv aufzählbar,
   aber nicht entscheidbar.
2. Die komplementären Probleme $\overline{K}, \overline{H}, \overline{H_0}$ und $\overline{H_{\tau_{uni}}}$ sind abzählbar aber nicht rekursiv aufzählbar

Beweis :

Zu 1: Es gilt $K \leq H = H_{\tau_{uni}} \leq H_0$.
Zu 2: Wären die komplementären Probleme rekursiv aufzählbar, so wären wegen "Teil 1: r.a." und des obigen Korollars die Halteprobleme entscheidbar. Widerspruch! □

§4 Automatische Programmverifikation

Kann der Computer für uns Programme verifizieren, d.h. festellen, ob ein Program $\mathcal{P}$ eine vorgegebene Spezifikation $\mathcal{S}$ erfüllt ? Genauer betrachten wir folgendes Verifikationsproblem:

Gegeben : Programm $\mathcal{P}$ und Spezifikation $\mathcal{S}$
Frage : Erfüllt $\mathcal{P}$ die Spezifikation $\mathcal{S}$?

Wir formalisieren dieses Problem wie folgt:

- Programm $\mathcal{P}$ $\stackrel{\wedge}{=}$ Turingmaschine τ mit Eingabealphabet $B = \{0, 1\}$
- Spezifikation $\mathcal{S}$ $\stackrel{\wedge}{=}$ Teilmenge $\mathcal{S}$ von $\mathcal{T}_{B,B}$, der Menge aller TURING-berechenbaren Funktionen $h : B^* \stackrel{part}{\longrightarrow} B^*$
- $\mathcal{P}$ erfüllt $\mathcal{S}$ $\stackrel{\wedge}{=}$ $h_\tau \in \mathcal{S}$

Die Antwort gibt der Satz von RICE (1953, 1956):

Das Verifikationsproblem ist unentscheidbar bis auf zwei triviale Ausnahmen:

- $\mathcal{S} = \emptyset$: Antwort stets "nein".
- $\mathcal{S} = \mathcal{T}_{B,B}$: Antwort stets "ja"

4.1 **Satz (Satz von RICE)**: Sei $\mathcal{S}$ eine beliebige, nicht-triviale Teilmenge von $\mathcal{T}_{B,B}$, d.h. es gelte $\emptyset \subset \mathcal{S} \subset \mathcal{T}_{B,B}$. Dann ist die Sprache

$$BW(\mathcal{S}) = \{bw_\tau \mid h_\tau \in \mathcal{S}\} \subseteq B^*$$

der Binärkodierung aller Turingmaschinen τ, deren berechnete Funktion h_τ in der Funktionenmenge $\mathcal{S}$ liegt, unentscheidbar.

Beweis : Um die Unentscheidbarkeit von $BW(\mathcal{S})$ zu zeigen, reduzieren wir das unentscheidbare Problem H_0 bzw. dessen Komplement $\overline{H_0} = B^* - H_0$ auf $BW(\mathcal{S})$.

Dazu betrachten wir zunächst eine beliebige Funktion $g \in \mathcal{T}_{B,B}$ und eine beliebige Turingmaschine τ. Sei τ_g eine TM, die g berechnet. Ferner sei τ' folgende TM, die von τ und g abhängt und die wir deshalb ausführlicher als $\tau' = \tau'(\tau, g)$ schreiben. Angesetzt auf ein Wort $v \in B^*$ arbeitet $\tau'(\tau, g)$ so:

1. Zunächst wird v ignoriert und gearbeitet wie τ angesetzt auf das leere Band.
2. Falls τ hält, wird anschließend gearbeitet wie τ_g angesetzt auf v.

Sei $\Omega \in \mathcal{T}_{B,B}$ die überall undefinierte Funktion. Dann ist die von $\tau'(\tau, g)$ berechnete Funktion $h_{\tau'(\tau,g)}$ durch folgende Fallunterscheidung beschreibbar:

$$h_{\tau'(\tau,g)} = \begin{cases} g & \text{falls } \tau \text{ auf dem leeren Band hält} \\ \Omega & \text{sonst.} \end{cases}$$

Zu vorgegebenem g gibt es eine totale, berechenbare Funktion $f_g : B^* \to B^*$ mit

$$f_g(bw_\tau) = bw_{\tau'(\tau,g)},$$

d.h. f_g berechnet aus einer gegebenen Binärkodierung einer Turingmaschine τ die Binärkodierung der Turingmaschine $\tau'(\tau, g)$.

Wir benutzen diese Funktion f_g zur Reduktion. Dabei unterscheiden wir zwei Fälle.

Fall 1 : $\Omega \notin \mathcal{S}$
Wähle irgendeine Funktion $g \in \mathcal{S}$. Dies ist möglich, da $\mathcal{S} \neq \emptyset$.
Wir zeigen: $H_0 \leq BW(\mathcal{S})$ mittels f_g.

$$\begin{aligned}
bw_\tau \in H_0 &\Leftrightarrow \tau \text{ angesetzt auf das leere Band hält} \\
&\Leftrightarrow h_{\tau'(\tau,g)} = g \\
&\Leftrightarrow \{\text{Es gilt } h_{\tau'(\tau,g)} \in \{g, \Omega\},\ \Omega \notin \mathcal{S} \text{ und } g \in \mathcal{S}.\} \\
&\quad\ h_{\tau'(\tau,g)} \in \mathcal{S} \\
&\Leftrightarrow bw_{\tau'(\tau,g)} \in BW(\mathcal{S}) \\
&\Leftrightarrow f_g(bw_\tau) \in BW(\mathcal{S})
\end{aligned}$$

Fall 2 : $\Omega \in \mathcal{S}$
Wähle irgendeine Funktion $g \notin \mathcal{S}$. Dies ist möglich, da $\mathcal{S} \neq \mathcal{T}_{B,B}$.
Wir zeigen: $\overline{H}_0 \leq BW(\mathcal{S})$ mittels f_g.

$$\begin{aligned}
bw_\tau \notin H_0 &\Leftrightarrow \tau \text{ angesetzt auf das leere Band hält nicht} \\
&\Leftrightarrow h_{\tau'(\tau,g)} = \Omega \\
&\Leftrightarrow \{\text{Es gilt } h_{\tau'(\tau,g)} \in \{g, \Omega\},\ \Omega \in \mathcal{S} \text{ und } g \notin \mathcal{S}.\} \\
&\quad\ h_{\tau'(\tau,g)} \in \mathcal{S} \\
&\Leftrightarrow bw_{\tau'(\tau,g)} \in BW(\mathcal{S}) \\
&\Leftrightarrow f_g(bw_\tau) \in BW(\mathcal{S})
\end{aligned}$$

Damit ist der Satz von Rice bewiesen. □

Anschaulich besagt der Satz von Rice, dass es hoffnungslos ist, aus der syntaktischen Form von Turingmaschinen oder Programmen in algorithmischer Weise etwas über deren Semantik, d.h. deren Ein-Ausgabe-Verhalten, verifizieren zu wollen.

Trotzdem ist automatische Programmverifikation (auch :„*model checking*“ genannt, wobei „model“ hier „Programm“ entspricht) heute ein sehr aktives Forschungsgebiet. Es ist zum Beispiel möglich für Programme, deren Verhalten durch *endliche Automaten* beschrieben werden kann.

§5 Grammatikprobleme und Postsches Korrespondenzproblem

Wir betrachten zunächst zwei Probleme über CHOMSKY-0-Grammatiken.

5.1 **Definition (Wortproblem für CHOMSKY-0)**:
Das *Wortproblem für CHOMSKY-0-Grammatiken* lautet :

Gegeben : CHOMSKY-0-Grammatik $G = (N, T, P, S)$ und ein Wort $w \in T^*$
Frage : Gilt $w \in L(G)$?

5.2 **Satz** : Das Wortproblem für CHOMSKY-0-Grammatiken ist unentscheidbar.

Beweisidee : Mit geeigneter Binärcodierung von Grammatiken G und Worten w lässt sich das Halteproblem für Turingmaschinen auf das Wortproblem reduzieren :
$H \leq Wortproblem.$

Vgl. für direkten Beweis auch die Beweise von Satz 1.2 und Lemma 1.6.

5.3 **Definition** : Das *Ableitungsproblem für CHOMSKY-0-Grammatiken* ist Folgendes:

Gegeben : CHOMSKY-0-Grammatik $G = (N, T, P, S)$ und
zwei Wörter $u, v \in (N \cup T)^*$
Frage : Gilt $u \vdash_G^* v$?

5.4 **Satz** : Das Ableitungsproblem für CHOMSKY-0-Grammatiken ist unentscheidbar.

Beweis : Offensichtlich gilt $Wortproblem \leq Ableitungsproblem$, weil für alle Grammatiken $G = (N, T, P, S)$ und Wörter $w \in T^*$ gilt:

$$w \in L(G) \Leftrightarrow S \vdash_G^* w.$$

□

Als nächstes betrachten wir eine Art Puzzle-Spiel, das POSTsche Korrespondenzproblem. Es wurde von E. POST (1946) angegeben und wird mit *PCP* (vom englischen "Post's Correspondence Problem") abgekürzt. Es geht darum, ein Wort auf zwei verschiedene Weisen zu konstruieren.

Für spätere Kodierungszwecke gehen wir von einer abzählbaren, unendlichen Menge

$$SYM = \{a_0, a_1, a_2, \ldots\}$$

von Symbolen aus. Wir betrachten stets Wörter über SYM. Sei ferner $X \subseteq SYM$ ein Alphabet.

5.5 **Definition** : Eine *Eingabe/Instanz des PCP* ist eine endliche Folge Y von Wortpaaren, also

$$Y = ((u_1, v_1), \ldots, (u_n, v_n))$$

mit $n \geq 1$ und $u_i, v_i \in SYM^*$ für $i = 1, \ldots, n$. Falls $u_i, v_i \in X^*$ gilt, heißt Y auch eine *Eingabe des PCP über* X .

Eine *Korrespondenz* oder *Lösung* von Y ist eine endliche Folge von Indizes

$$(i_1, \ldots, i_m)$$

mit $m \geq 1$ und $i_1, \ldots, i_m \in \{1, \ldots, n\}$, so dass

$$u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$$

gilt. Y heißt *lösbar*, falls es eine Lösung von Y gibt.

5.6 **Definition** :

1. Das Post*sche Korrespondenzproblem (PCP)* ist folgendes Problem:

   Gegeben: Eingabe Y des PCP
   Frage: Besitzt Y eine Korrespondenz ?

2. Das *PCP über* X ist folgendes Problem:

   Gegeben: Eingabe Y des PCP über X
   Frage: Besitzt Y eine Korrespondenz ?

3. Das *modifizierte PCP (kurz MPCP)* ist folgendes Problem:

   Gegeben: Eingabe $Y = ((u_1, v_1), \ldots, (u_n, v_n))$ des PCP,
   wobei $u_i \neq \varepsilon$ für $i = 1, \ldots, n$
   Frage: Besitzt Y eine Korrespondenz $(i_1, \ldots, i_m)$ mit $i_1 = 1$?
   Die Korrespondenz soll also mit dem ersten Wortpaar beginnen, so dass $u_{i_1} u_{i_2} \ldots u_{i_m} = v_{i_1} v_{i_2} \ldots v_{i_m}$ gilt.

Beispiel : Wir betrachten zunächst

$$Y_1 = ((10, 00), (1, 101), (011, 11))$$

also

$$\begin{array}{lllllll} u_1 & = & 10 & , v_1 & = & 00 \\ u_2 & = & 1 & , v_2 & = & 101 \\ u_3 & = & 011 & , v_3 & = & 11 \end{array}$$

Dann ist $(2, 3, 1, 3)$ eine Korrespondenz von Y_1, denn es gilt:

$$u_2 u_3 u_1 u_3 = 101110011 = v_2 v_3 v_1 v_3$$

(2) Als nächstes betrachten wir

$$Y_2 = ((00, 10), (1, 0), (101, 0)).$$

Y_2 besitzt keine Korrespondenz, da für jedes Wortpaar $(u, v) \in Y_2$ gilt: weder u ist Anfangswort von v, noch ist v Anfangswort von u.

Selbst einfache Eingaben des PCP können einen hohen Schwierigkeitsgrad besitzen. Zum Beispiel besitzt für

$$Y = ((001, 0), (01, 011), (01, 101), (10, 001))$$

die kürzeste Lösung bereits **66** Indizes (vgl. Schöning, 4. Auflage, S. 132). Wir untersuchen deshalb die Frage, ob das PCP entscheidbar ist.

Bemerkung : Für einelementige Alphabete X ist das PCP über X entscheidbar.

Beweis : Sei etwa $X = \{I\}$. Jedes Wort I^n über X kann mit der natürlichen Zahl n identifiziert werden. Jede Eingabe Y des PCP über X kann daher als Folge von Paaren natürlicher Zahlen aufgefasst werden:

$$Y = ((u_1, v_1), \ldots, (u_n, v_n))$$

mit $n \geq 1$ und $u_i, v_i \in \mathbb{N}$ für $i = 1, \ldots, n$. Y ist lösbar, genau dann, wenn es eine Folge von Indizes $(i_1, \ldots, i_m)$ mit $m \geq 1$ und $i_j \in \{1, \ldots, n\}$ für $j = 1, \ldots, m$ gibt, so dass

$$\sum_{j=1}^{m} u_{i_j} = \sum_{j=1}^{m} v_{i_j}$$

gilt. Wir haben es also mit der Lösbarkeit eines Problems für natürliche Zahlen zu tun.

Wir zeigen:

Y ist lösbar $\Leftrightarrow$ (a) $\exists j \in \{1, \ldots, n\} : u_j = v_j$
oder
(b) $\exists k, j \in \{1, \ldots, n\}$:
$u_k < v_k$ und $u_j > v_j$

"$\Rightarrow$": Kontrapositionsbeweis: Falls weder (a) noch (b) gilt, haben alle Paare $(u, v) \in Y$ die Eigenschaft $u < v$ oder alle Paare $(u, v) \in Y$ die Eigenschaft $u > v$. Die durch u-Teile zusammengesetzten Wörter wären also stets kürzer oder stets länger als durch v-Teile zusammengesetzte Wörter. Damit ist Y nicht lösbar.

"$\Leftarrow$": Falls $u_j = v_j$ gilt, ist die triviale Indexfolge (j) Lösung von Y. Falls $u_k < v_k$ und $u_l > v_l$ gilt, ist die Folge

$$(\underbrace{k, \ldots, k}_{(u_l - v_l)mal}, \underbrace{l, \ldots, l}_{(v_k - u_k)mal})$$

Lösung von Y, denn es gilt

$$u_k(u_l - v_l) + u_l(v_k - u_k) = v_k(u_l - v_l) + v_l(v_k - u_k).$$

Offenbar sind die Eigenschaften (a) und (b) entscheidbar. Damit ist das PCP über einem einelementigen Alphabet entscheidbar. □

Wir untersuchen jetzt den allgemeinen Fall des PCP. Unser Ziel ist folgender Satz:

5.7 **Satz** : Für jedes Alphabet X mit $|X| \geq 2$ ist das PCP über X unentscheidbar.

Wir zeigen den Satz durch Reduktion des Ableitungsproblems für Chomsky-0-Grammatiken auf das PCP über $\{0,1\}$. Genauer beweisen wir drei interessante Reduktionen:

Ableitungsproblem $\leq$ MPCP $\leq$ PCP $\leq$ PCP über $\{0,1\}$.

Wir beginnen mit der ersten Reduktion.

5.8 **Lemma** : *Ableitungsproblem* $\leq MPCP$

Beweis : Wir geben einen Algorithmus an, der zu gegebener CHOMSKY-0-Grammatik $G = (N, T, P, S)$ mit $(N \cup T \subseteq SYM)$ und gegebenen Wörtern $u, v \in (N \cup T)^*$ eine Eingabe $Y_{G,u,v}$ für das MPCP konstruiert, so dass folgendes gilt:

(1) $u \vdash_G^* v \Leftrightarrow Y_{G,u,v}$ besitzt eine Korrespondenz, die mit dem ersten Wortpaar beginnt.

Wir benutzen ein Symbol $\sharp$, das nicht in $N \cup T$ vorkommen möge. Dann besteht $Y_{G,u,v}$ aus folgenden Wortpaaren:

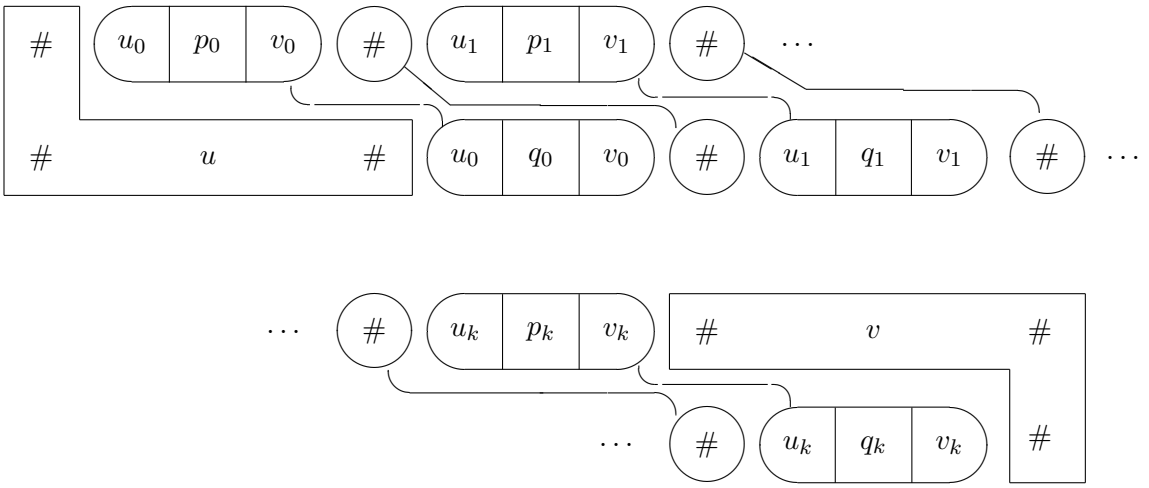
- Erstes Wortpaar : $(\sharp, \sharp u \sharp)$
- Produktionen : (p, q) für alle $p \to q \in P$
- Kopieren : (a, a) für alle $a \in N \cup T \cup \{\sharp\}$
- Abschluss : $(\sharp v \sharp, \sharp)$

Bis auf erste Wortpaare ist die genaue Reihenfolge der Wortpaare in $Y_{G,u,v}$ unerheblich. Wir zeigen jetzt, dass (1) gilt.

"$\Rightarrow$" : Es gelte $u \vdash_G^* v$. Dann gibt es eine Ableitung von u nach v der Form

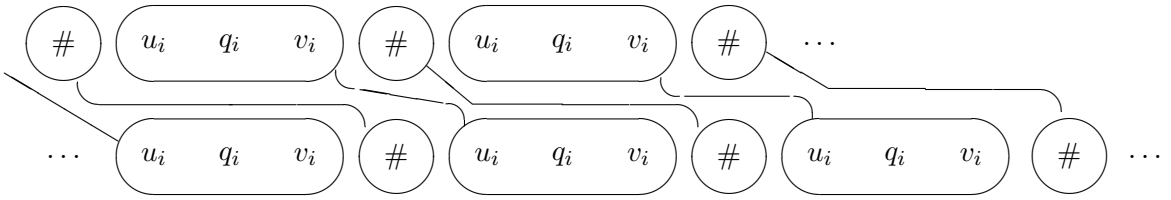
$$\begin{aligned}(2)\ u = u_0 p_0 v_0 &\vdash_G u_0 q_0 v_0 = u_1 p_1 v_1 \\ &\vdash_G u_1 q_1 v_1 = u_2 p_2 v_2 \\ &\dots \\ &\vdash_G u_k q_k v_k = v\end{aligned}$$

wobei $p_0 \to q_0, \dots, p_k \to q_k \in P$ gilt. Zu dieser Ableitung gibt es folgende Korrespondenz von $Y_{G,u,v}$, wobei wir die beiden Komponenten der Wortpaare in zwei Zeilen aneinanderreihen :



Die "Winkelstücke" mit u und v stellen das erste Wortpaar bzw. das Abschlusspaar dar. Die rund umrandeten Teilwörter $u_i, v_i, \sharp$ werden symbolweise durch die Kopier-Paare (a, a) für $a \in N \cup T \cup \{\sharp\}$ gebildet. Die eckig umrandeten Teilwörter p_i, q_i werden durch das entsprechende Produktionenpaar (p_i, q_i) gelegt. Durch verbindende Linien ist die Zugehörigkeit zum selben Wortpaar angedeutet.

"$\Leftarrow$" : Gegeben sei jetzt eine Korrespondenz von $Y_{G,u,v}$, die mit dem ersten Wortpaar beginnt. Aus der Gestalt der Wortpaare in $Y_{G,u,v}$ folgt, dass die Korrespondenz so aufgebaut sein muss wie unter "$\Rightarrow$" gezeigt mit der Ausnahme, dass die Teilwörter u bzw. $u_i q_i v_i$ durch reines Anwenden der Kopier-Paare (a, a) sich endlich oft wiederholen können :



Um die Korrespondenz zu vervollständigen, muss das Abschlusspaar gelegt werden können. Dazu müssen Produktionspaare (p_i, q_i) eingefügt werden, so dass die Korrespondenz eine Ableitung (2) von u nach v beschreibt.

Man beachte, dass "$\Leftarrow$" nur gilt, weil wir im Sinne der $MPCP$ mit dem *ersten* Wortpaar beginnen. Ohne diese Einschränkung lieferte z.B. das Kopierpaar $(\sharp, \sharp)$ eine triviale Korespondenz, zu der nicht $u \vdash_G^* v$ folgt. □

5.9 **Lemma** : $MPCP \leq PCP$

Beweis : Wir geben einen Algorithmus an, der zu gegebener Eingabe $Y = ((u_1, v_1), \ldots, (u_n, v_n))$ für das $MPCP$, also mit $u_i \neq \varepsilon$ für $i = 1, \ldots, n$, eine Eingabe Y_{PCP} für das PCP konstruiert, so dass folgendes gilt :

(3) Y besitzt eine Korrespondenz, die mit dem ersten Wortpaar beginnt.

$\Leftrightarrow Y_{PCP}$ besitzt eine Korrespondenz.

Idee : Konstruiere Y_{PCP} so, dass jede Korrespondenz notwendigerweise mit dem ersten Wortpaar beginnt.

Dazu benutzen wir neue Symbole β und γ , die in keinem der Wörter aus Y vorkommen mögen. Wir definieren zwei Abbildungen

$\rho : SYM^* \to SYM^*$,
$\lambda : SYM^* \to SYM^*$,

wobei für $w \in SYM^*$

- $\rho(w)$ aus w entsteht, indem *rechts* von jedem Buchstaben von w das Symbol β eingefügt wird,
- $\lambda(w)$ aus w entsteht, indem *links* von jedem Buchstaben von w das Symbol β eigefügt wird.

Zum Beispiel gilt

$$\rho(GTI) = G\beta T\beta I\beta \quad \text{und} \quad \lambda(GTI) = \beta G\beta T\beta I.$$

Für die Abbildungen ρ und λ gelten folgende Gesetze: für alle $u, v \in SYM^a st$

1. $\rho(\varepsilon) = \varepsilon$ und $\lambda(\varepsilon) = \varepsilon$
2. $\rho(uv) = \rho(u)\rho(v)$ und $\lambda(uv) = \lambda(u)\lambda(v)$
3. $u = v \Leftrightarrow \beta\rho(u) = \lambda(v)\beta$

Die Gesetze 1 und 2 besagen, dass ρ und λ *Worthomomorphismen* sind, d.h. ε und die Konkatenation bleiben erhalten.

Aus Y konstruieren wir folgendes Y_{PCP} mit Wortpaaren, die von 0 bis $n+1$ durchnummeriert sind:

$$\begin{aligned} Y_{PCP} = (\ & (\beta\rho(u_1), \lambda(v_1)), && \text{0.Wortpaar} \\ & (\rho(u_1), \lambda(v_1)) , && \text{1.Wortpaar} \\ & \cdots \qquad , && \cdots \\ & (\rho(u_n), \lambda(v_n)) , && n\text{-tes.Wortpaar} \\ & (\gamma, \beta\gamma)) \qquad , && n+1\text{-tes Wortpaar} \end{aligned}$$

Wir zeigen jetzt. dass (3) gilt:

"⇒": Sei $(1, i_1, \ldots, i_m)$ eine Korrespondenz von Y, d.h. $u_1 u_{i_2} \ldots u_{i_m} = v_1 v_{i_2} \ldots v_{i_m}$. Mit Gesetz 3. gilt :

$$\beta\rho(u_1 u_{i_2} \ldots u_{i_m})\gamma = \lambda(v_1 v_{i_2} \ldots v_{i_m})\beta\gamma$$

Durch mehrfaches Anwenden von Gesetz 2. erhalten wir :

$=$

$\beta\rho(u_1)$	$\rho(u_{i_2})$	$\dots$	$\rho(u_{i_m})$	γ
$\lambda(v_1)$	$\lambda(v_{i_2})$	$\dots$	$\lambda(v_{i_m})$	$\beta\gamma$

Durch die Einrahmungen haben wir Wörter zusammengefasst, die jeweils in einem Wortpaar von Y_{PCP} vorkommen. Wir sehen:

$$(1, i_2, \dots, i_m, n+1)$$

ist eine Korrespondenz von Y_{PCP}.

"$\Leftarrow$": Wir zeigen diese Richtung nur für den folgenden *Fall*: $v_i \neq \varepsilon$ für $i = 1, \dots, n$
Sei $(i_1, \dots, i_m)$ irgendeine Korrespondenz von Y_{PCP}. Dann gilt $i_1 = 0$ und $i_m = n+1$, da nur die Wortpaare im 1.Wortpaar $(\beta\rho(u_1), \lambda(v_1))$ mit dem selben Symbol beginnen und nur die Wörter im $(n+1)$-ten Wortpaar $(\gamma, \beta\gamma)$ mit demselben Symbol enden. Sei $k \in \{2, \dots, m\}$ der kleinste Index mit $i_k = n+1$. Dann ist auch $(i_1, \dots, i_k)$ eine Korrespondenz von Y_{PCP}, da γ nur als letztes Symbol im zugehörigen Korrespondenzwort vorkommt. Aus der Form der Wortpaare folgt weiterhin:

$$i_j \neq 1 \text{ für } j = 2, \dots, k-1.$$

Andernfalls gäbe es im Teilwort $\rho(u_{i_{j-1}})\beta\rho(u_1)$ zwei aufeinanderfolgende β's, die wegen $v_i \neq \varepsilon$ durch kein Legen von $\lambda(v_i)$'s nachgebildet werden könnten.

Damit hat das Korrespondenzwort folgende Struktur

$=$

$\beta\rho(u_1)$	$\rho(u_{i_2})$	$\dots$	$\rho(u_{i_{k-1}})$	γ
$\lambda(v_1)$	$\lambda(v_{i_2})$	$\dots$	$\lambda(u_{i_{k-1}})$	$\beta\gamma$

Durch Anwenden der Gesetze (ii) und (iii) schließen wir

$$u_1 u_{i_2} \dots u_{i_{k-1}} = v_1 v_{i_2} \dots v_{i_{k-1}}.$$

Damit ist $(1, i_2, \dots, i_{k-1})$ eine Korrespondenz von Y. Im Falle, dass es ein $v_i = \varepsilon$ gibt, ist die Argumentation schwieriger.

□

5.10 **Lemma** : $PCP \leq PCP$ über $\{0, 1\}$

Beweis : Zur Reduktion benutzen wir eine Binärkodierung der Wörter über SYM, d.h. eine injektive, berechenbare Funktion

$$bw : SYM^* \to \{0,1\}^*,$$

wie wir sie bei der Binärkodierung von Turingmaschinen kennengelernt haben.

Sei jetzt eine Eingabe $Y = ((u_1, v_1), \ldots, (u_n, v_n))$ des PCP gegeben. Dann definieren wir

$$bw(Y) = ((bw(u_1), bw(v_1)), \ldots, (bw(u_n), bw(v_n)))$$

als Eingabe des PCP über $\{0,1\}$. Offenbar gilt:

$$Y \text{ist lösbar} \Leftrightarrow bw(Y) \text{ ist lösbar.}$$

□

Aus den obigen drei Lemmata gewinnen wir die Unentscheidbarkeit der folgenden Probleme:

- $MPCP$,
- PCP,
- PCP über $\{0,1\}$ und
- PCP über X mit $|X| \geq 2$.

Insbesondere haben wir obigen Satz bewiesen.

§6 Unentscheidbarkeitsresultate für kontextfreie Sprachen

Wir können jetzt die in Kapitel III angkündigten Unentscheidbarkeitsresultate für kontextfreie Sprachen beweisen. Im Gegensatz zu regulären Sprachen gilt Folgendes:

6.1 **Satz** : Für kontextfreie (also CHOMSKY-2-) Sprachen sind

- das Schnittproblem,
- das Äquivalenzproblem,
- das Inklusionsproblem

unentscheidbar.

Beweis :
Schnittproblem: Gegeben seien zwei kontextfreie Grammatiken G_1 und G_2. Die Frage lautet: Gilt $L(G_1) \cap L(G_2) = \emptyset$? Wir zeigen, dass das Postsche Korrespondenzproblem auf das Schnittproblem reduzierbar ist

$$PCP \leq \text{ Schnittproblem}$$

Daraus folgt die Unentscheidbarkeit des Schnittproblems.

Gegeben sei jetzt eine beliebige Eingabe $Y = ((u_1, v_1), \ldots, (u_n, v_n))$ des PCP mit $u_i, v_i \in X^*$ für ein Alphabet X. Wir geben einen Algorithmus an, der für jede solche Eingabe Y zwei kontextfreie Grammatiken G_1 und G_2 konstruiert, so dass folgendes gilt

$$Y \text{ besitzt eine Korrespondenz.} \iff L(G_1) \cap L(G_2) \neq \emptyset. \qquad (*)$$

Die Idee ist, dass G_1 alle Wörter erzeugt, die durch Aneinanderlegen der u_i's entstehen können, und G_2 alle Wörter, die durch Aneinanderlegen der v_i's entstehen können. Damit die gewünschte Beziehung $(*)$ gilt, müssen G_1 und G_2 auch die Indizes i der gelegten u_i und v_i festhalten. Dazu benutzen wir n neue Symbole $a_1, \ldots, a_n \notin X$ und wählen für G_1 und G_2 als Menge der Terminalsymbole

$$T = \{a_1, \ldots, a_n\} \cup X$$

Setze dann $G_i = (\{S\}, T, P_i, S), i = 1, 2$. Dabei ist P_1 durch folgende Produktionen gegeben:

$$S \to a_1 u_1 \,|\, a_1 S u_1 \,|\, \ldots \,|\, a_n u_n \,|\, a_n S u_n$$

P_2 ist durch folgende Produktionen gegeben:

$$S \to a_1 v_1 \,|\, a_1 S v_1 \,|\, \ldots \,|\, a_n v_n \,|\, a_n S v_n$$

Offenbar gilt

$$L(G_1) = \{a_{i_m} \ldots a_{i_1} u_{i_1} \ldots u_{i_m} \,|\, m \geq 1 \text{ und } i_1, \ldots, i_m \in \{1, \ldots, n\}\}$$

und

$$L(G_2) = \{a_{i_m} \dots a_{i_1} v_{i_1} \dots v_{i_m} \mid m \geq 1 \text{ und } i_1, \dots, i_m \in \{1, \dots, n\}\}.$$

Daraus folgt:

$$\begin{aligned} & Y \text{ besitzt die Korrespondenz } (i_1, \dots, i_m). \\ \Leftrightarrow \quad & a_{i_m} \dots a_{i_1} u_{i_1} \dots u_{i_m} = a_{i_m} \dots a_{i_1} v_{i_1} \dots v_{i_m} \in L(G_1) \cap L(G_2). \end{aligned}$$

Also gilt $(*)$ und damit die Unentscheidbarkeit des Schnittproblems.

Wir haben sogar ein stärkeres Resultat bewiesen: die Unentscheidbarkeit des Schnittproblems für *deterministische* kontextfreie Sprachen. Wie man leicht nachprüft, sind nämlich die Sprachen $L(G_1)$ und $L(G_2)$ deterministisch. Wir halten daher fest: Das Schnittproblem bleibt unentscheidbar, wenn statt der kontextfreien Grammatiken G_1 und G_2 zwei (deterministische) Kellerautomaten $\mathcal{K}_1$ und $\mathcal{K}_2$ gegeben sind und die Frage lautet, ob $L(\mathcal{K}_1) \cap L(\mathcal{K}_2) = \emptyset$ gilt.

Äquivalenzproblem: Gegeben seien zwei kontextfreie Grammatiken G_1 und G_2. Die Frage lautet: Gilt $L(G_1) = L(G_2)$? Wir zeigen die Unentscheidbarkeit dieses Problems durch folgende Reduktion:

$$\begin{aligned} & \text{Schnittproblem für deterministische kontextfreie Sprachen} \\ & \leq \text{ Äquivalenzproblem.} \end{aligned}$$

Seien zwei deterministische Kellerautomaten $\mathcal{K}_1$ und $\mathcal{K}_2$ gegeben. Wir zeigen, dass daraus algorithmisch zwei (nicht notwendig deterministische) kontextfreie Grammatiken G_1 und G_2 konstruiert werden können, so dass Folgendes gilt:

$$L(\mathcal{K}_1) \cap L(\mathcal{K}_2) = \emptyset \iff L(G_1) = L(G_2).$$

Wir nutzen dabei aus, dass zu $\mathcal{K}_2$ der Komplement-Kellerautomat $\overline{\mathcal{K}_2}$ mit $L(\overline{\mathcal{K}_2}) = \overline{L(\mathcal{K}_2)}$ konstruiert werden kann (vgl. Kapitel III, Abschnitt 6). Aus $\mathcal{K}_1$ und $\overline{\mathcal{K}_2}$ können dann algorithmisch kontextfreie Grammatiken G_1 und G_2 mit

$$L(G_1) = L(\mathcal{K}_1) \cup L(\overline{\mathcal{K}_2}) \text{ und } L(G_2) = L(\overline{\mathcal{K}_2})$$

konstruiert werden. Damit gilt

$$\begin{aligned} L(\mathcal{K}_1) \cap L(\mathcal{K}_2) = \emptyset \quad & \Leftrightarrow \quad L(\mathcal{K}_1) \subseteq \overline{L(\mathcal{K}_2)} \\ & \Leftrightarrow \quad L(\mathcal{K}_1) \subseteq L(\overline{\mathcal{K}_2}) \\ & \Leftrightarrow \quad L(\mathcal{K}_1) \cup L(\overline{\mathcal{K}_2}) = L(\overline{\mathcal{K}_2}) \\ & \Leftrightarrow \quad L(G_1) = L(G_2) \end{aligned}$$

wie gewünscht.

Inklusionsproblem: Gegeben seien zwei kontextfreie Grammatiken G_1 und G_2. Die Frage lautet: Gilt $L(G_1) \subseteq L(G_2)$? Offensichtlich gilt die Reduktion

$$\text{Äquivalenzproblem} \leq \text{Inklusionsproblem}$$

Daher ist auch das Inklusionsproblem unentscheidbar. □

Ein weiteres Unentscheidbarkeitsresultat betrifft die Mehrdeutigkeit von kontextfreien Grammatiken. Für die praktische Anwendung von kontextfreien Grammatiken zur Syntaxbeschreibung von Programmiersprachen wäre es angenehm, einen algorithmischen Test auf Mehrdeutigkeit zu haben. Wir zeigen jedoch später, dass es einen solchen Test nicht gibt.

6.2 **Satz** : Es ist unentscheidbar, ob eine gegebene kontextfreie Grammatik mehrdeutig ist.

Beweis : Wir zeigen folgende Reduktion

$$PCP \leq \text{ Mehrdeutigkeitsproblem.} \qquad (*)$$

Gegeben sei eine Eingabe $Y = ((u_1, v_1), \ldots, (u_n, v_n))$ des PCP. Wir konstruieren zunächst die kontextfreien Grammatiken $G_i = (\{S_i\}, T, P_i, S_i), i = 1, 2$, wie für die Reduktion des PCP auf das Schnittproblem. Anschließend konstruieren wir daraus die kontextfreie Grammatik $G = (\{S, S_1, S_2\}, T, P, S)$ mit

$$P = \{S \to S_1, S \to S_2\} \cup P_1 \cup P_2.$$

Da G_1 und G_2 eindeutig sind, liegt die einzig mögliche Mehrdeutigkeit von G bei der Konstruktion eines Ableitungsbaumes von G zu einem Wort $w \in T^*$ in der Benutzung der Produktionen $S \to S_1$ bzw. $S \to S_2$. Daher ergibt sich folgendes:

$$\begin{aligned} G \text{ ist mehrdeutig} &\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \\ &\Leftrightarrow Y \text{ besitzt eine Korrespondenz.} \end{aligned}$$

Damit ist die Reduktion $(*)$ gezeigt, und es folgt die Unentscheidbarkeit des Mehrdeutigkeitsproblems. □

Kapitel VI

Komplexität

§1 Berechnungskomplexität

Bisher haben wir uns mit der *Effektivität* oder *Berechenbarkeit* von Problemen beschäftigt, d.h. mit der Frage, ob vorgegebene Probleme überhaupt algorithmisch lösbar sind, und wir haben uns mit einer Art *struktureller Komplexität* beschäftigt, d.h. mit der Unterscheidung, welcher Typ von Maschine zur algorithmischen Lösung von Problemen benötigt wird. Dabei haben wir folgende Hierarchie kennengelernt:

(1) reguläre Sprachen $\leftrightarrow$ endliche Automaten

(2) kontextfreie Sprachen $\leftrightarrow$ Kellerautomaten

(3) CHOMSKY-0-Sprachen $\leftrightarrow$ Turingmaschinen

Jetzt wollen wir uns mit der *Effizienz* oder *Berechnungskomplexität* befassen, d.h. mit der Frage, wieviel *Rechenzeit* und wieviel *Speicherplatz* man benötigt, um ein Problem algorithmisch zu lösen. Genauer werden Zeit und Platz in Abhängigkeit von der *Größe der Eingabe* studiert. Man unterscheidet zwei Arbeitsrichtungen:

a) Für konkrete Probleme *möglichst effiziente Algorithmen* angeben:

   - wichtig für praktische Problemlösungen
   - theoretisch interessant als Nachweis von *oberen Schranken* für das Problem: z.B. besagt ein existierender n^3-Algorithmus, dass das Problem höchstens n^3 „schwer“ ist.

b) Nachweis von *unteren Schranken* für ein Problem, so dass *jeder* Algorithmus mindestens die Komplexität der unteren Schranke annimmt. Triviale untere Schranke für die Zeitkomplexität ist n, die Größe der Eingabe.

Aussagen über die Komplexität hängen vom benutzten *Maschinenmodell* ab. In der Theorie betrachtet man meistens deterministische oder auch nichtdeterministische *Turingmaschinen mit*

mehreren Bändern. Durch die Benutzung mehrerer Bänder erhält man realistischere Aussagen als bei 1-Band-TM, da Rechenzeiten für bloßes Hin- und Herlaufen auf dem Turingband entfallen.

1.1 **Definition** : Seien $f : \mathbb{N} \to \mathbb{R}$ eine Funktion und τ eine nichtdeterministische Mehrband-TM mit dem Eingabealphabet Σ.

(i) τ hat die *Zeitkomplexität* $f(n)$, falls für jedes Wort $w \in \Sigma^*$ der Länge n gilt: τ angesetzt auf die Eingabe w hält bei jeder möglichen Berechnung in höchstens $f(n)$ Schritten an.

(ii) τ hat die *Platzkomplexität* $f(n)$, falls für jedes Wort $w \in \Sigma^*$ der Länge n gilt: τ angesetzt auf die Eingabe w benutzt bei jeder möglichen Berechnung auf jedem Band höchstens $f(n)$ Felder.

Diese Definition ist insbesondere auch auf deterministische Mehrband-TM anwendbar. Für solche TM gibt es natürlich für jede Eingabe w genau eine Berechnung.

Im Zusammenhang mit TM werden Probleme meist als Sprachen dargestellt, die akzeptiert werden sollen. Diese Darstellung haben wir bereits in Kapitel über „Nichtberechenbare Funktionen und Unentscheidbare Probleme “ benutzt. Z.B. wurde das Halteproblem für TM als die Sprache

$$H = \{bw_\tau 00u \in B^* \mid \tau \text{ angesetzt auf } u \text{ hält } \}$$

dargestellt. Wir fassen jetzt Probleme, also Sprachen, die mit gleicher Komplexität akzeptiert werden können, zu sogenannten *Komplexitätsklassen* zusammen.

1.2 **Definition** : Sei $f : \mathbb{N} \to \mathbb{R}$.

$\texttt{DTIME}\ (f(n)) = \{L \mid$ es gibt eine *deterministische* Mehrband-TM der Zeitkomplexität $f(n)$, die L akzeptiert $\}$

$\texttt{NTIME}\ (f(n)) = \{L \mid$ es gibt eine *nichtdeterministische* Mehrband-TM der Zeitkomplexität $f(n)$, die L akzeptiert $\}$

$\texttt{DSPACE}\ (f(n)) = \{L \mid$ es gibt eine *deterministische* Mehrband-TM der Platzkomplexität $f(n)$, die L akzeptiert $\}$

$\texttt{NSPACE}\ (f(n)) = \{L \mid$ es gibt eine *nichtdeterministische* Mehrband-TM der Platzkomplexität $f(n)$, die L akzeptiert $\}$

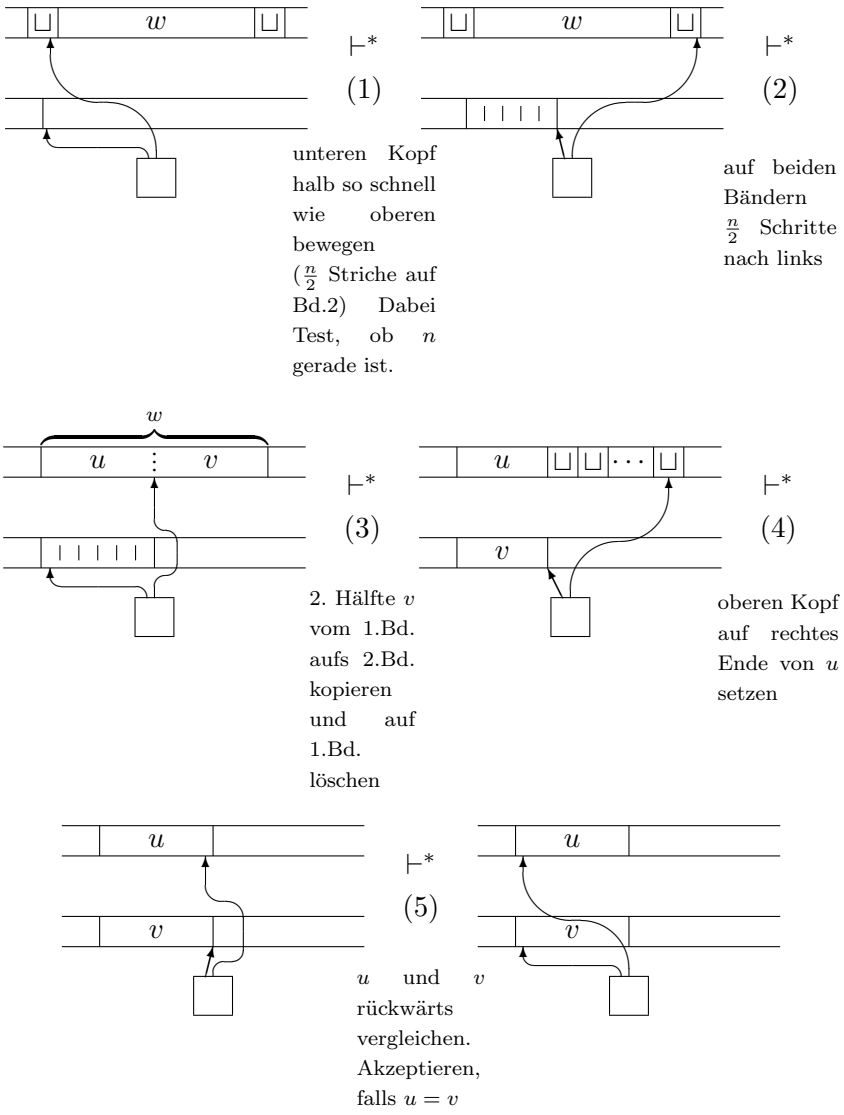
Beispiel : Betrachte $L = \{uu \mid u \in \{a, b\}^*\}$. Wir wollen eine möglichst effiziente Mehrband-TM konstruieren, die L akzeptiert.

Lösungsidee:

- Zu gegebenem Wort $w \in \{a, b\}^*$ stelle erst die Mitte von w fest und vergleiche dann die beiden Hälften.

- Zum Feststellen der Mitte wird ein 2.Band benutzt: also deterministische 2-Band TM.

Operationelle Phasen: Gegeben sei $w \in \{a, b\}^*$ der Länge n.



Zeitkomplexität der 5 Phasen berechnen:

$$(n+1)+(\frac{n}{2}+1)+(\frac{n}{2}+1)+(\frac{n}{2}+1)+(\frac{n}{2}+1) = 3n+5$$

Platzkomplexität: $n+2$

Also gilt: $L \in \texttt{DTIME}\ (3n+5)$,
$L \in \texttt{DSPACE}\ (n+2)$.

Nichtdeterministisch kann man folgendermaßen vorgehen:

Teil 1: Kopiere zeichenweise von Band 1 auf Band 2. Bricht diesen Prozess nichtdeterministisch irgendwann ab.

Teil 2: Kehre auf Band 2 an den Anfang zurück.

Teil 3: Vergleiche nun zeichenweise, ob ab der Position auf dem ersten Band das gleiche steht wie auf dem zweiten Band. Falls dies zutrifft und beide Lese-Schreibköpfe gleichzeitig am Ende auf Blank stoßen, dann akzeptiere.

Dieses Vorgehen benötigt im Erfolgsfall

$$\frac{n}{2} + \left(\frac{n}{2} + 1\right) + \left(\frac{n}{2} + 1\right) = \frac{3n}{2} + 2 \quad \text{Schritte,}$$

im ungünstigsten Fall, dass die nichtdeterministische Entscheidung erst beim letzten Zeichen des Eingabewortes fällt

$$n + (n + 1) + 1 = 2n + 2 \quad \text{Schritte.}$$

Also gilt: $L \in \texttt{NTIME}(2n + 2)$

In der Komplexitätstheorie vergleicht man das *asymptotische Verhalten* von Zeit- und Platzkomplexität, d.h. das Verhalten für „genügend große“ n. Dadurch lassen sich konstante Faktoren vernachlässigen. Dazu wird die *O-Notation* aus der Zahlentheorie benutzt.

1.3 **Definition** : Sei $g : \mathbb{N} \to \mathbb{R}$. Dann ist

$$O(g(n)) = \{f : \mathbb{N} \to \mathbb{R} \mid \exists n_0, k \in \mathbb{N} \, \forall n \geq n_0 : f(n) \leq k \cdot g(n)\}$$

d.h. $O(g(n))$ ist die Klasse aller Funktionen f, die für genügend große Argumente n durch ein Vielfaches von g *beschränkt* werden.

Beispiel : $(n \mapsto 3n + 4), (n \mapsto n + 2) \in O(n)$. Wir sagen deshalb auch: die Sprache L kann mit Zeit- und Platzkomplexität $O(n)$ akzeptiert werden, bzw. L kann mit *linearer* Zeit- und Platzkomplexität akzeptiert werden.

Es gelten folgende Rechenregeln für die O-Notation:

$$\begin{aligned}
f &\in O(f(n)) \\
O(c \cdot f(n)) &= O(f(n)) \text{ falls } c > 0 \\
O(f(n) + g(n)) &= O(max(f(n), g(n)) \\
O(f(n)) \cdot O(g(n)) &= O(f(n) \cdot g(n)) \\
O(f(n)) &\subseteq O(g(n)) \text{ falls } f \leq g \\
O(\Sigma_{i=0}^{k} \, a_i \cdot n^i) &= O(n^k) \text{ falls } a_k > 0
\end{aligned}$$

Da eine TM in jedem Rechenschritt höchstens ein neues Feld auf ihren Bändern besuchen kann, folgt:

$$\begin{array}{ccccc}
 & \subseteq & \texttt{DSPACE}(f(n)) & \subseteq & \\
\texttt{DTIME}(f(n)) & & & & \texttt{NSPACE}(f(n)) \\
 & \subseteq & \texttt{NTIME}(f(n)) & \subseteq &
\end{array}$$

§2 Die Klassen P und NP

Für praktisch brauchbare Algorithmen sollte die Komplexitätsfunktion ein *Polynom* $p(n)$ k-ten Grades sein, also von der Form

$$p(n) = a_k n^k + \cdots + a_1 n + a_0$$

mit $a_i \in \mathbb{N}$ für $i = 0, 1, \ldots, k$, $k \in \mathbb{N}$ und $a_k \neq 0$.

2.1 **Definition (Cobham, 1964)**:

$$\begin{aligned} P &= \bigcup_{p \text{ Polynom in } n} \texttt{DTIME}(p(n)) \\ NP &= \bigcup_{p \text{ Polynom in } n} \texttt{NTIME}(p(n)) \\ PSPACE &= \bigcup_{p \text{ Polynom in } n} \texttt{DSPACE}(p(n)) \\ NPSPACE &= \bigcup_{p\text{Polynom in } n} \texttt{NSPACE}(p(n)) \end{aligned}$$

2.2 **Satz (Savitch, 1970)**: Für alle Polynome p in n gilt:

$$NSPACE(p(n)) = DSPACE(p^2(n))$$

und damit

$$NPSPACE = PSPACE.$$

Ferner gilt

$$P \subseteq NP \subseteq PSPACE, \qquad (*)$$

da – wie bereits gesagt – eine TM in jedem Rechenschritt höchstens ein neues Feld besuchen kann.

Offenes Problem der Informatik: Sind die Inklusionen in $(*)$ echt oder gilt die Gleichheit ?

Die Klassen P und NP sind von großer Bedeutung, denn sie markieren den Übergang von praktisch durchführbarer Berechenbarkeit bzw. Entscheidbarkeit zu nur theoretisch interessanter Berechenbarkeit bzw. Entscheidbarkeit. Praktisch lösbar, d.h. praktisch berechenbar bzw. entscheidbar sind die Probleme in der Klasse P, deren Prinzip durch folgenden Merksatz charakterisiert werden kann:

P: *Konstruiere* in deterministischer Weise und
polynomieller Zeit die richtige Lösung.

Dabei sollten die zeitbeschränkenden Polynome einen möglichst kleinen Grad wie n, n^2 oder n^3 haben.

Praktisch unlösbar sind dagegen alle Probleme, für die der Zeitaufwand nachweislich exponentiell mit der Größe n der Eingabe wächst, jedenfalls wenn beliebige n zugelassen werden. Zwischen diesen beiden Extremen liegt eine große Klasse von praktisch wichtigen Problemen, für die im Augenblick nur exponentielle deterministische Algorithmen bekannt sind, die aber durch *nichtdeterministische* Algorithmen in polynomieller Zeit gelöst werden können. Dieses ist die Klasse NP, deren Prinzip im Vergleich zu P wie folgt formuliert werden kann:

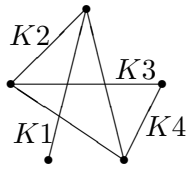
> NP: *Rate nichtdeterministisch* einen Lösungsvorschlag und *verifiziere* bzw. *prüfe* dann in deterministischer Weise und polynomieller Zeit, ob dieser Vorschlag eine richtige Lösung ist.

Auch diese Probleme sind bis heute *in voller Allgemeinheit* praktisch unlösbar. In der Praxis behilft man sich mit sogenannten *Heuristiken*, die den nichtdeterministischen Suchraum der möglichen Lösungsvorschläge stark einschränken. Durch diese Heuristiken versucht man, eine „Ideallösung“ zu approximieren.

Beispiele für Probleme aus der Klasse NP

(1) **Problem des Hamiltonschen Pfades**
Gegeben: Ein endlicher ungerichteter Graph mit n Knoten.
Frage: Gibt es einen Hamiltonschen Pfad in dem Graphen, d.h. einen Kantenzug, der jeden Knoten genau einmal trifft?

Betrachte z.B. den folgenden Graphen:



Dann ist der Kantenzug K1-K2-K3-K4 ein Hamiltonscher Pfad. Es ist leicht zu sehen, dass das Problem des Hamiltonschen Pfades in NP liegt: Man rate zunächst einen Pfad und prüfe dann, ob jeder Knoten genau einmal getroffen wird. Da es $n!$ Pfade im Graphen gibt, wäre dieses Verfahren nur mit exponentiellem Aufwand in deterministischer Weise anzuwenden.

(2) **Problem des Handlungsreisenden**
Gegeben: Ein endlicher ungerichteter Graph mit n Knoten und Längenangaben $\in \mathbb{N}$ für jede Kante sowie eine Zahl $k \in \mathbb{N}$.
Frage: Gibt es für den Handlungsreisenden eine Rundreise der Länge $\leq k$ oder mathematischer: gibt es einen Kreis, d.h. einen geschlossenen Kantenzug, der Länge $\leq k$ der jeden Knoten mindestens einmal trifft?

Auch dieses Problem liegt in NP: Man rate zunächst einen Kreis und berechne dann dessen Länge. Das Problem des Handlungsreisenden ist von praktischer Bedeutung z.B. beim Entwurf von Telefonnetzwerken oder integrierten Schaltungen.

(3) **Erfüllbarkeitsproblem für Boolesche Ausdrücke (abgekürzt SAT vom englischen Wort „satisfiability“)**
Gegeben: Ein Boolescher Ausdruck E, also ein Ausdruck, der nur aus den Variablen $v_1, v_2, \ldots, v_n$ besteht, die durch Operatoren $\neg$ (*not*), $\wedge$ (*and*) und $\vee$ (*or*), sowie durch Klammern miteinander verknüpft sind.
Frage: Ist E erfüllbar, d.h. gibt es eine Belegung der Booleschen Variablen $v_1, v_2, \ldots, v_n$ in E mit 0 und 1, so dass E insgesamt den Wert 1 liefert?

Zum Beispiel ist $E = (v_1 \wedge v_2) \vee \neg v_3$ erfüllbar durch die Werte $v_1 = v_2 = 1$ oder $v_3 = 0$. Wir werden das Problem SAT im Abschnitt 3 genauer untersuchen. □

Bei den Untersuchungen zur bislang offenen Frage, ob P = NP gilt, ist man auf eine erstaunliche Teilklasse von NP gestoßen, die Klasse NPC der sogenannten NP-*vollständigen* Probleme. Es gilt:

> Wenn *ein* Problem aus NPC in P liegt, so liegen bereits *alle* Probleme aus NP in P, d.h. dann gilt P = NP.

Die Klasse NPC wurde 1971 von S.A. Cook eingeführt. Das eben vorgestellte Problem SAT wurde von Cook als erstes als NP-vollständig nachgewiesen. Ab 1972 hat R. Karp viele weitere Probleme als NP-vollständig nachgewiesen. Heute kennt man weit über 1000 Beispiele aus der Klasse NPC.

Im folgenden wollen wir den Begriff NP-vollständig definieren. Dazu benötigen wir den Begriff der *polynomiellen Reduktion*, die Karp 1972 als Beweistechnik zum Nachweis von NP-Vollständigkeit einführte. Es handelt sich dabei um eine Verschärfung des Begriffs der Reduktion $L_1 \leq L_2$ aus Abschnitt 2.

2.3 **Definition** : Seien $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ Sprachen. Dann heißt L_1 *auf* L_2 *polynomiell reduzierbar*, abgekürzt

$$L_1 \leq_p L_2,$$

falls es eine totale und mit der Zeitkomplexität eines Polynoms berechenbare Funktion $f : \Sigma_1^* \to \Sigma_2^*$ gibt, so dass für alle $w \in \Sigma_1^*$ gilt:

$$w \in L_1 \Leftrightarrow f(w) \in L_2.$$

Wir sagen auch: $L_1 \leq_p L_2$ *mittels* f.

Anschaulich besagt $L_1 \leq_p L_2$, dass L_1 nicht aufwendiger oder „schwerer“ als L_2 ist. Man erkennt leicht, dass $\leq_p$ eine reflexive und transitive Relation auf Sprachen ist, da mit zwei Polynomen $p_1(n)$ und $p_2(n)$ auch $p_1(p_2(n))$ ein Polynom ist.

2.4 **Definition (Cook, 1971)**: Eine Sprache L_0 heißt *NP-vollständig*, falls Folgendes gilt:
(1) $L_0 \in NP$ und

(2) L_0 ist *NP-hart*, d.h. $\forall L \in NP : L \leq_p L_0$.

2.5 **Lemma (Polynomielle Reduktion)**: Sei $L_1 \leq_p L_2$. Dann gilt:

(i) Falls $L_2 \in P$ gilt, so auch $L_1 \in P$.

(ii) Falls $L_2 \in NP$ gilt, so auch $L_1 \in NP$.

(iii) Falls L_1 NP-vollständig ist und $L_2 \in NP$ gilt, so ist auch L_2 NP-vollständig.

Beweis : *zu (i) und (ii):* Es gelte $L_1 \leq_p L_2$ mittels einer Funktion f, die durch eine Turingmaschine τ_1 berechnet wird. Das Polynom p_1 begrenze die Rechenzeit von τ_1. Da $L_2 \in P$ (bzw. $L_2 \in NP$) ist, gibt es eine durch ein Polynom p_2 zeitbeschränkte (nichtdeterministische) Turingmaschine τ_2, die die charakteristische Funktion χ_{L_2} berechnet.

Wie bei der normalen Reduktion ist dann die charakteristische Funktion χ_{L_1} für alle $w \in \Sigma_1^*$ wie folgt berechenbar:

$$\chi_{L_1}(w) = \chi_{L_2}(f(w)).$$

Dieses geschieht durch Hintereinanderschalten der Turingmaschinen τ_1 und τ_2 . Sei jetzt $|w| = n$. Dann berechnet τ_1 das Wort $f(w)$ in $p_1(n)$ Schritten. Diese Zeitschranke beschränkt zugleich die Länge von $f(w)$, d.h. $|f(w)| \leq p_1(n)$. Damit wird die Berechnung von $\chi_{L_2}(f(w))$ in

$$p_1(n) + p_2(p_1(n))$$

Schritten durchgeführt. Also ist auch $L_1 \in P$ (bzw. $L_1 \in NP$).

zu (iii): Sei $L \in NP$. Da L_1 NP-vollständig ist, gilt $L \leq_p L_1$. Ferner gilt $L_1 \leq_p L_2$. Aus der Transitivität von $\leq_p$ folgt $L \leq_p L_2$, was zu zeigen war. □

2.6 **Korollar (Karp)**: Sei L_0 eine NP-vollständige Sprache. Dann gilt: $L_0 \in P \Leftrightarrow P = NP$.

Beweis : „⇒“: Aussage (i) des Lemmas.
„⇐“: klar. □

Beispiel : Wir zeigen folgende polynomielle Reduktion:

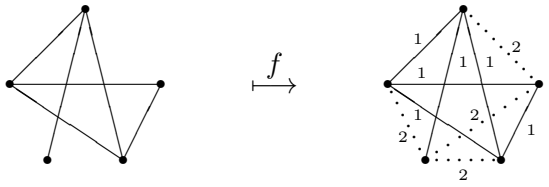
$$\text{Hamiltonscher Pfad } \leq_p \text{ Handlungsreisender.}$$

Gegeben sei ein Graph G mit n Knoten, dargestellt als Inzidenzmatrix der Größe n^2. Als Reduktionsfunktion betrachten wir folgende Konstruktion $f : G \mapsto (G', k)$ eines neuen Graphens G' mit Kantenlängen:

- G' übernimmt die n Knoten von G.
- Jede Kante von G wird Kante von G' der Länge 1.
- Jede „Nichtkante“ von G wird Kante von G' der Länge 2.

Damit ist G' ein vollständiger Graph, d.h. je zwei Knoten sind durch eine Kante verbunden. Ferner setzen wir als Schranke für die Länge der Rundreise $k = n + 1$.

Z.B. G: $n = 5$ $\quad f \longmapsto$ G': mit der Schranke $k = n + 1$



Die Konstruktion f erfolgt in poynomieller Zeit. (Für das Finden der Nichtkanten: die Inzidenzmatrix für G betrachten, also $O(n^2)$) Wir zeigen jetzt die Reduktionseigenschaft von f:

$$G \text{ hat Hamiltonschen Pfad} \Leftrightarrow G' \text{ hat Rundreise der Länge} \leq k = n + 1.$$

Beweis von „$\Rightarrow$“: Man braucht $n - 1$ Kanten der Länge 1 (d.h. „aus G“), um n verschiedene Knoten zu verbinden. Um diesen Hamiltonschen Pfad zu schließen, braucht man eine weitere Kante der Länge ≤ 2. Insgesamt hat die so konstruierte Rundreise die Länge $\leq n + 1$.

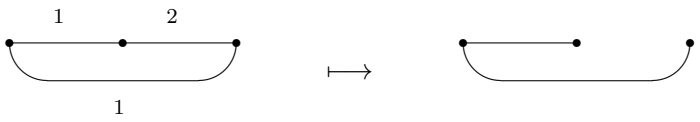
Beweis von „$\Leftarrow$“: In der Rundreise gibt es mindestens n Kanten (um n Knoten auf einem Kreis zu verbinden) und höchstens $n + 1$ Kanten (wegen der Kantenlängen ≥ 1 und der Wahl von $k = n + 1$).

Auf der Rundreise wird mindestens 1 Knoten zweimal erreicht (Start = Ende). Man muss also *auf jeden Fall eine Kante entfernen*, um einen Pfad mit lauter verschiedenen Knoten zu erhalten. Wir untersuchen, ob das reicht, und unterscheiden dazu folgende Fälle.

Fall 1: Es gibt in der Rundreise eine Kante der Länge 2.

Wir entfernen diese Kante. Der verbleibende Kantenzug hat die Länge $\leq n - 1$. In diesem Kantenzug gibt es dann $n - 1$ Kanten der Länge 1, die n verschiedene Knoten verbinden. Das ist der gesuchte Hamiltonsche Pfad. Hier ist ein Beispiel zu diesem Fall:

G':

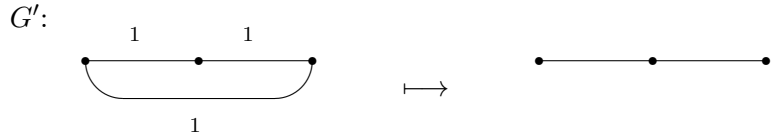


Rundreise der Länge 4 $\qquad$ Hamiltonscher Pfad

Fall 2: Alle Kanten in der Rundreise haben die Länge 1, sind also Kanten aus G.

Unterfall 2a: Die Rundreise hat n Kanten.

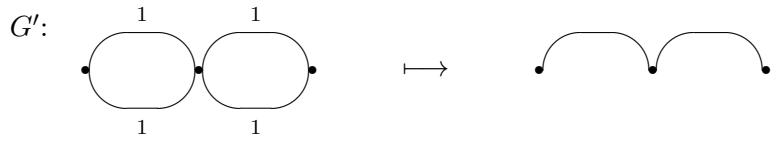
Wir entfernen eine Kante. Die verbleibenden $n - 1$ Kanten bilden (analog zu Fall 1) einen Hamiltonschen Pfad. Ein Beispiel dazu:



Rundreise der Länge 3 Hamiltonscher Pfad

Unterfall 2b: Die Rundreise hat $n + 1$ Kanten.

Wir entfernen zunächst eine Kante. Auf dem verbleibenden Kantenzug mit n Kanten gibt es noch einen Knoten, der zweimal besucht wird, also einen Teilkreis. Wir entfernen eine Kante, die an diesem Knoten liegt. Der verbleibende Kantenzug mit $n - 1$ Kanten bildet einen Hamiltonschen Pfad. Ein Beispiel dazu:



Rundreise der Länge 4 Hamiltonscher Pfad

§3 Das Erfüllbarkeitsproblem für Boolesche Ausdrücke

3.1 **Definition SAT**:

(i) Ein *Boolescher Ausdruck* (eine *Boolesche Formel*) ist ein Wort über {0,1} und Variablen mit den Operationen *Negation* ($\neg$), *Konjunktion* ($\wedge$) und *Disjunktion* ($\vee$). Im einzelnen (vgl. „Programmierung“ und „Rechnerstrukturen“): $V = \{v_1, v_2, \ldots\}$ sei eine unendliche Menge von Variablen. $B = \{0, 1\}$ sei die Menge der Wahrheitswerte („falsch“ und „wahr“).

(1) Jedes $v_i \in V$ ist ein Boolescher Ausdruck.

(2) Jedes $b \in B$ ist ein Boolescher Ausdruck.

(3) Wenn E ein Boolescher Ausdruck ist, dann auch $\neg E$.

(4) Wenn E_1, E_2 Boolesche Ausdrücke sind, dann auch $(E_1 \vee E_2)$ und $(E_1 \wedge E_2)$.

Wenn $v \in V$, dann schreibt man statt $\neg v$ meist $\overline{v}$.
Wenn $v \in V$, dann nennt man v und $\overline{v}$ *Literale.*

$\left(\begin{array}{l}\text{Prioritäten:}\\ \neg \text{ vor } \wedge \text{ vor } \vee\end{array}\right)$

(ii) Konjunktive Normalform:

(1) Wenn $y_1, y_2, \ldots, y_k$ Literale sind, dann ist $(y_1 \vee y_2 \vee \ldots \vee y_k)$ eine *Klausel* (*der Ordnung* k, d.h. mit k Literalen/Alternativen)

(2) Wenn $c_1, c_2, \ldots, c_r$ Klauseln (der Ordnung $\leq k$) sind, dann ist $c_1 \wedge c_2 \wedge \ldots \wedge c_r$ ein Boolescher Ausdruck in *konjunktiver Normalform* („KNF“) (der Ordnung $\leq k$). Wenn mindestens eine Klausel k Literale enthält, dann heißt dieser Ausdruck eine KNF der Ordnung k. Beispielsweise ist $E = (v_1 \vee v_2) \wedge (\neg v_1 \vee \neg v_2)$ in KNF der Ordnung 2. [Analog kann man „disjunktive Normalform“ definieren; siehe „Rechnerstrukturen“.]

(iii) Eine *Belegung* β ist eine Abbildung $\beta : V \to \{0, 1\}$, die folgendermaßen auf Boolesche Ausdrücke fortgesetzt wird (siehe (i),(2)–(4)):

$\beta(0) = 0, \beta(1) = 1,$
$\beta(\neg E) = 1 - \beta(E),$
$\beta(E_1 \wedge E_2) = \mathit{Min}(\beta(E_1), \beta(E_2)),$
$\beta(E_1 \vee E_2) = \mathit{Max}(\beta(E_1), \beta(E_2)).$

(iv) Ein Boolescher Ausdruck E heißt *erfüllbar* („satisfiable“), wenn es eine Belegung β gibt mit $\beta(E) = 1$.

(v) Das Erfüllbarkeitsproblem („satisfiability problem“) wird beschrieben durch die Sprache

$$\text{SAT} = \{E \mid E \text{ ist Boolescher Ausdruck, und} E \text{ ist erfüllbar } \}.$$

Das Alphabet ist in diesem Fall $V \cup \{0, 1, \neg, \vee, \wedge, (,) \}$ und daher wegen der Teilmenge V nicht endlich. Es lässt sich aber ein endliches Alphabet erreichen, indem die Variablen aus V wir folgt kodiert werden:

$$v_i \mapsto vj, \qquad \text{wobei } j \text{ die Dualdarstellung der Zahl } i \text{ ist}$$

also $v_1 \mapsto v1, v_2 \mapsto v10, v_3 \mapsto v11, v_4 \mapsto v100$ usw. für ein festes Symbol v. So ergibt sich

$$\text{SAT} \subseteq \{v, 0, 1, \neg, \wedge, \vee, (,)\}^*.$$

Die Elemente aus B können aus einem Booleschen Ausdruck E leicht durch Rechenregeln entfernt und es kann E in KNF vorgeben werden:

$$\text{KNF-SAT} := \{ E \in \text{SAT} \mid E \text{ ist Boolescher Ausdruck (ohne Elemente aus } B) \text{ in KNF} \},$$

$$\text{KNF-SAT}(k) := \{ E \in \text{KNF-SAT} \mid E \text{ ist von der Ordnung } k \}.$$

3.2 **Satz (Cook, 1971)**: SAT ist NP-vollständig.

Beweis :

- Zunächst erkennt man leicht, dass SAT in NP liegt:
  Man *rät* eine Belegung $\beta : \{v_1, \ldots, v_m\} \to \{0, 1\}$ zu einem Booleschen Ausdruck E, der genau m Variablen enthält. Dann setzt man für jede Variable v_i den Wert $\beta(v_i)$ ein und rechnet $\beta(E)$ nach den üblichen Rechenregeln (Def. 3.1(iii)) aus. Wenn $|E| = n$ gilt, dann besitzt E weniger als n Variable, d.h. $m \leq n$; das Raten einer Belegung β erfolgt in linearer Zeit (Tabelle anlegen, m Schritte), die Ersetzung in E dauert $|E| \cdot const$ Schritte und ebenso die Auswertung, d.h. SAT $\in$ NTIME $(c_1 n + c_2)$ für geeignete Konstanten c_1 und c_2. Insbesondere SAT $\in$ NP. [Beschreiben Sie, wie man die Berechnung $\beta(E)$ in linearer Zeit erreichen kann.]

- Der schwierigere Teil besteht darin zu zeigen, dass SAT NP-hart ist.

  Sei L eine beliebige Sprache aus NP. Dann existiert eine nichtdeterministische 1-Band TM $\tau = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$, die L in der Zeitkomplexität $p(n)$ akzeptiert, wobei p ein Polynom und n die Länge des Eingabewortes $w \in \Sigma^*$ ist. Dabei soll τ im Endzustand eingefroren sein:

  $$\forall q \in F, a \in \Gamma : \delta(q, a) = \{(q, a, S)\}.$$

  Zu zeigen ist: $L \leq_p$ SAT. Die Idee ist, zu gegebenem Eingabewort $w \in \Sigma^*$ einen Booleschen Ausdruck $E_{\tau,w}$ zu konstruieren, der das Verhalten von τ beschreibt und folgende Eigenschaft hat:

  $$\tau \text{ akzeptiert } w \Leftrightarrow E_{\tau,w} \text{ ist erfüllbar.}$$

  Wir nehmen o.B.d.A. an, dass $Q = \{q_0, \ldots, q_k\}, \Gamma = \{a_1, \ldots, a_l\}$ und $w = x_1 \ldots x_n$ gilt.

  Der Ausdruck $E_{\tau,w}$ enthält folgende Boolesche Variablen:

  - $zust_{t,q}$ für $t = 0, \ldots, p(n)$ und $q \in Q$.
    Interpretation: $zust_{t,q} = 1$ gdw. τ nach Schritt t im Zustand q ist.
  - $pos_{t,i}$ für $t = 0, \ldots, p(n)$ und $i = -p(n), \ldots, p(n)$.
    Interpretation: $pos_{t,i} = 1$ gdw. τ nach Schritt t auf Bandposition i ist.

– $band_{t,i,a}$ für $t = 0, \ldots, p(n)$ und $i = -p(n), \ldots, p(n)$ und $a \in \Gamma$.
  Interpretation: $band_{t,i,a} = 1$ gdw. nach Schritt t von τ auf Bandposition i das Zeichen a steht.

Wir konstruieren den Ausdruck $E_{\tau,w}$ aus mehreren Teilausdrücken. Dabei benutzen wir mehrfach einen Teilausdruck G, der für Variablen $v_1, \ldots, v_m$ Folgendes beschreibt:

$$G(v_1, \ldots, v_m) = 1 \text{ gdw. } \textit{genau eine} \text{ der Variablen } v_1, \ldots, v_m \text{ ist mit 1 belegt.}$$

Wir definieren

$$\begin{array}{lrcrcrcrcl} G(v_1, \ldots, v_m) := (& & v_1 & \wedge & \neg v_2 & \wedge & \ldots & \wedge & \neg v_m & \\ & \vee & \neg v_1 & \wedge & v_2 & \wedge & \ldots & \wedge & \neg v_m & \\ & & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \\ & \vee & \neg v_1 & \wedge & \neg v_2 & \wedge & \ldots & \wedge & v_m &). \end{array}$$

$G(v_1, \ldots, v_m)$ hat die Form einer $m \times m$-Matrix und daher die Größe $O(m^2)$.

Mit R beschreiben wir folgende *Randbedingungen* von τ:

– Die TM τ ist immer in genau einem Zustand.
– Der Kopf von τ ist immer in genau einer Position.
– An jeder Stelle des Bandes steht genau ein Zeichen.

Daher setzen wir

$$\begin{array}{rcl} R & := & \bigwedge_t \; G(zust_{t,q_0}, \ldots, zust_{t,q_k}) \\ & \wedge & \bigwedge_t \; G(pos_{t,-p(n)}, \ldots, pos_{t,p(n)}) \\ & \wedge & \bigwedge_{t,i} G(band_{t,i,a_1}, \ldots, band_{t,i,a_l}), \end{array}$$

wobei $t = 0, \ldots, p(n)$ und $i = -p(n), \ldots, p(n)$ gilt.

Mit A beschreiben wir die *Anfangsbedingungen* (zum Zeitpunkt $t = 0$) von τ:

– Die TM τ ist im Zustand q_0 und an der Position 1.
– Das Band ist leer bis auf die Positionen $1, \ldots, n$.
  Dort befindet sich das Wort $w = x_1 \ldots x_n$.

Daher setzen wir

$$\begin{array}{rcl} A & := & zust_{0,q_0} \wedge pos_{0,1} \\ & \wedge & \bigwedge_{i=-p(n)}^{0} band_{0,i,\sqcup} \wedge \bigwedge_{i=1}^{n} band_{0,i,x_i} \wedge \bigwedge_{i=n+1}^{p(n)} band_{0,i,\sqcup} \,. \end{array}$$

Mit T_1 und T_2 beschreiben wir die *Übergänge* (*Transitionen*) von τ: Ist τ im Schritt t im Zustand q an der Position i und liest a, dann können alle Transitionen aus $\delta(q, a)$ ausgeführt werden:

$$\begin{array}{rcll} T_1 & := & \bigwedge_{t,q,i,a} & (zust_{t,q} \wedge pos_{t,i} \wedge band_{t,i,a} \longrightarrow \\ & & & \bigvee_{(q',a',P) \in \delta(q,a)} \; zust_{t+1,q'} \wedge pos_{t+1,i+m(P)} \wedge band_{t+1,i,a'}) \,. \end{array}$$

wobei $t = 0, \ldots, p(n)$, $q \in Q$, $i = -p(n), \ldots, p(n)$ und $a \in \Gamma$ gilt. Ferner ist $m(L) = -1$, $m(S) = 0$ und $m(R) = 1$ und $E_1 \longrightarrow E_2$ Abkürzung für $\neg E_1 \vee \neg E_2$.

Alle anderen Bandinhalte bleiben gleich:

$$T_2 \;:=\; \bigwedge_{t,q,i,a} \;(\neg pos_{t,i} \wedge band_{t,i,a} \longrightarrow band_{t+1,i,a})\,,$$

wobei $t = 0, \ldots, p(n)$, $q \in Q$, $i = -p(n), \ldots, p(n)$ und $a \in \Gamma$ gilt.

Mit *End* beschreiben wir die *Endbedingung* von τ, dass zum Zeitpunkt $t = p(n)$ ein Endzustand eingenommen wird:

$$End \;:=\; \bigvee_{q \in F} zust_{p(n),q}\,.$$

Insgesamt ist der gesuchte Ausdruck dann

$$E_{\tau,w} \;:=\; R \wedge A \wedge T_1 \wedge T_2 \wedge End\,.$$

Aus der Konstruktion von $E_{\tau,w}$ ergibt sich sofort, dass

- aus einem akzpetierenden Lauf von τ für w eine Belegung der Variablen konstuiert werden kann, für die $E_{\tau,w}$ wahr wird, und
- aus einer $E_{\tau,w}$ erfüllenden Belegung der Variablen ein akzpetierender Lauf von τ für w konstruierbar ist.

Also gilt insgesamt:

$$w \in L \;\Leftrightarrow\; \tau \text{ akzeptiert } w \;\Leftrightarrow\; E_{\tau,w} \text{ ist erfüllbar} \;\Leftrightarrow\; E_{\tau,w} \in \text{SAT},$$

so dass bereits $L \leq \text{SAT}$ gilt. Es bleibt zu zeigen, dass $E_{\tau,w}$ für eine feste TM τ und ein beliebiges Eingabewort $w = x_1 \ldots x_n$ in *polynomieller Zeit* berechnet werden kann:

- Die obige Konstruktion ist unmittelbar durchführbar und enthält keine aufwendigen algorithmischen Ideen.
- $|E_{\tau,w}| \in O(p^3(n))$, wobei $p(n)$ die Zeitkomplexität von τ beim Akzeptieren von w mit $|w| = n$ ist, wie folgende Abschätzungen zeigen:

Für R haben wir

$$\begin{array}{rcll} R & := & \bigwedge_t \; G(zust_{t,q_0}, \ldots, zust_{t,q_k}) & // \in O(p(n)) \\ & \wedge & \bigwedge_t \; G(pos_{t,-p(n)}, \ldots, pos_{t,p(n)}) & // \in O(p^3(n)) \\ & \wedge & \bigwedge_{t,i} G(band_{t,i,a_1}, \ldots, band_{t,i,a_l}) & // \in O(p^2(n)) \end{array}$$

Bei der Komplexitätsabschätzung haben wir die Größe der Formel G berücksichtigt. Wir schätzen hier die Anzahl der Booleschen Variablen ab. Für A berechnen wir

$$\begin{array}{rcll} A & := & zust_{0,q_0} \wedge pos_{0,1} & // \in O(1) \\ & \wedge & \bigwedge_{i=-p(n)}^{0} band_{0,i,\sqcup} \wedge \bigwedge_{i=1}^{n} band_{0,i,x_i} \wedge \bigwedge_{i=n+1}^{p(n)} band_{0,i,\sqcup} & // \in O(p(n)) \end{array}$$

Für T_1 ergibt sich

$$T_1 \;=\; \bigwedge_{t,q,i,a} \;(zust_{t,q} \wedge pos_{t,i} \wedge band_{t,i,a} \longrightarrow \bigvee_{(q',a',P)\,\in\,\delta(q,a)} zust_{t+1,q'} \wedge pos_{t+1,i+m(P)} \wedge band_{t+1,i,a'}) \quad // \in O(p^2(n))$$

Für T_2 ergibt sich

$$T_2 \;:=\; \bigwedge_{t,q,i,a} \;(\neg pos_{t,i} \longrightarrow band_{t,i,a} = band_{t+1,i,a}) \quad // \in O(p^2(n))$$

Für *End* erhalten wir

$$End \;:=\; \bigvee_{q\,\in\,F} zust_{p(n),q} \quad // \in O(1)$$

Also ergibt sich für die Größe von

$$E_{\tau,w} \;:=\; R \wedge A \wedge T_1 \wedge T_2 \wedge End$$

die Summe

$$|E_{\tau,w}| \in O(p^3(n) + 3 \cdot p^2(n) + 2 \cdot p(n) + 2) = O(p^3(n)).$$

Also gilt $L \leq_p$ SAT. Damit ist der Satz 3.2 von Cook bewiesen: SAT ist NP-vollständig. □

Wir betrachten jetzt das Erfüllbarkeitsproblem KNF-SAT(3) für Boolesche Ausdrücke in KNF der Ordnung 3, das in der Literatur auch 3SAT genannt wird.

3.3 **Satz** : KNF-SAT(3) ist NP-vollständig.

Beweis : Wegen SAT ∈ NP gilt auch KNF-SAT(3) ∈ NP. Wir zeigen SAT $\leq_p$ KNF-SAT(3)[1]. Dazu beschreiben wir ein Verfahren mit polynomieller Zeitkomplexität, das einen beliebigen Booleschen Ausdruck E in einen Booleschen Ausdruck E' in KNF der Ordnung 3 transformiert, so dass gilt:

$$E \text{ ist erfüllbar} \;\; \text{gdw.} \;\; E' \text{ ist erfüllbar.}$$

Wir sagen dann auch, E ist *erfüllbarkeitsäquivalent* zu E'.

Das Verfahren wird an dem Beispiel $E = \neg(\neg(v_1 \vee \neg v_2) \vee v_3)$ erläutert. In mehreren Schritten wird E erfüllbarkeitsäquivalent in E' transformiert.

Schritt 1. Alle Negationen werden mit Hilfe der Regeln von DeMorgan an die Variablen gebracht: $E_1 = (v_1 \vee \neg v_2) \wedge \neg v_3$. Dieser Schritt ist von der Zeitkomplexität $O(n)$, wobei $n = |E|$ ist.

Schritt 2. Jedem Vorkommen eines Operators $\wedge$ oder $\vee$ wird eine neue Boolesche Variable x_i als Index zugeordnet: $E_2 = (v_1 \vee_{x_1} \neg v_2) \wedge_{x_2} \neg v_3$.

Schritt 3. E_2 wird in eine Konjunktion von Äquivalenzen mit jeweils drei Variablen transformiert und derjenigen Variablen, die am Operator auf oberster Ebene steht, hier x_2:
$E_3 = [x_1 \leftrightarrow (v_1 \vee \neg v_2)] \wedge [x_2 \leftrightarrow (x_1 \wedge \neg v_3)] \wedge x_2$.

[1] siehe *U. Schöning, Theoretische Informatik – kurzgefasst, Spektrum Akad. Verlag, 5. Auflage, 2008, S. 158 ff.*

Schritt 4. Die Äquivalenzen werden nach folgendem Schema in KNF transformiert:

$$[a \leftrightarrow (b \vee c)] \mapsto (a \vee \neg b) \wedge (\neg a \vee b \vee c) \wedge (a \vee \neg c)]$$
$$[a \leftrightarrow (b \wedge c)] \mapsto (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)]$$

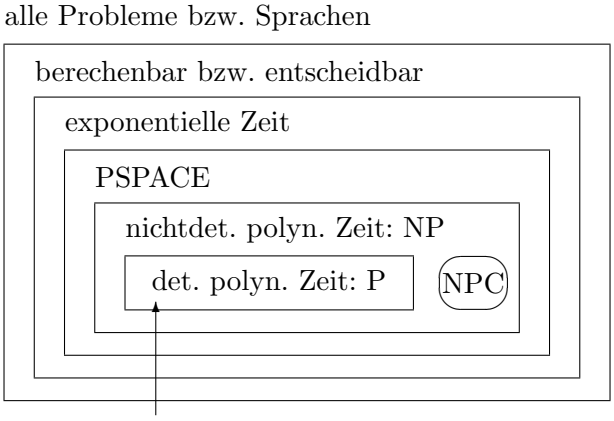
Angewandt auf E_3 ergibt sich als Endgebnis der Ausdruck

$$\begin{aligned} E' = \quad & (x_1 \vee \neg v_1) \wedge (\neg x_1 \vee v_1 \vee \neg v_2) \wedge (x_1 \vee v_2) \\ \wedge \quad & (\neg x_2 \vee x_1) \wedge (\neg x_2 \vee \neg v_3) \wedge (x_2 \vee \neg x_1 \vee v_3) \\ \wedge \quad & x_2 \,. \end{aligned}$$

in KNF der Ordnung 3. Alle Schritte sind von polynomieller Zeitkomplexität in Abhängigkeit von der Länge des gegebenen Ausdrucks E. Damit ist die Reduktion SAT $\leq_p$ KNF-SAT(3) gezeigt. Mit Lemma 2.5 (iii) folgt daraus die Behauptung des Satzes. □

Zwar sind E und E' *erfüllbarkeitsäquivalent*, was für den obigen Beweis genügt, aber *nicht aussagenlogisch äquivalent*, weil die neuen Variablen x_1 und x_2 von E' in E gar nicht vorkommen. Die Transformation eines beliebigen Booleschen Ausdrucks in eine aussagenlogisch äquivalente KNF ist von exponentieller Zeitkomplexität.

Weitere NP-vollständiger Probleme sind in der Literatur zu finden (Rucksackproblem, kürzeste Rundreise, Hamiltonsche Wege in Graphen, Cliquenproblem, Färbungsproblem von Graphen, ganzzahlige Programmierung, Stundenplanproblem, Zuteilungsproblem, Einbettungsproblem bei Graphen usw.). Abschließend seien die Enthaltenseinsbeziehungen der wichtigsten Komplexitätsklassen dargestellt:



Im Kern liegt die Klasse P der praktisch lösbaren Probleme. NPC bezeichnet die Klasse der NP-vollständigen (engl. NP complete) Probleme.

Index