Formale Systeme WS 2011/2012 Skript zur Vorlesung

Article

3 authors, including:

Christel Baier
Technische Universität Dresden
291 PUBLICATIONS **12,850** CITATIONS

SEE PROFILE

Formale Systeme
WS 2011/2012

Skript zur Vorlesung

Christel Baier, Manuela Berg, Walter Nauber
Lehrstuhl für Algebraische und Logische Grundlagen der Informatik
Fakultät für Informatik
TU Dresden

Inhaltsverzeichnis

Einleitung

Der erste Teil der Vorlesung behandelt Automatentheorie und formale Sprachen. Die behandelten Konzepte formaler Sprachen und der Automatentheorie bilden eine wichtige Grundlage z.B. für den Übersetzerbau, den Schaltkreisentwurf oder die Textverarbeitung. Im zweiten Teil beschäftigt sich die Vorlesung mit der Aussagenlogik.

Die Inhalte des ersten Teils der Vorlesung können zu großen Teilen in zahlreichen Lehrbüchern, die eine Einführung in die Theoretische Informatik anhand der Themenkomplexe formale Sprachen und Automatentheorie zum Ziel haben, nachgelesen werden. Eine Auswahl an Lehrbüchern, die den Inhalten der Vorlesung am nächsten kommen, ist:

1. A. Asteroth, C. Baier, *Theoretische Informatik. Eine Einführung in Berechenbarkeit, Komplexität und formale Sprachen mit 101 Beispielen*. Pearson Studium, 2002. (In der SLUB können einige Exemplare ausgeliehen werden.)
2. H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*. Prentice Hall, 1998.
3. J.E. Hopcroft, R. Motwani, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 3.Auflage, 2007.
4. U. Schöning, *Theoretische Informatik kurz gefaßt*. Spektrum Akademischer Verlag, 4. Auflage, 2000.
5. T. Sudkamp, *Languages and Machines. An Introduction to the Theory of Computer Science*. Addison Wesley, 3. Auflage, 2006.
6. I. Wegener, *Theoretische Informatik*. B.G. Teubner, 1993.

Es gibt ebenfalls zahlreiche Bücher, die sich mit mathematischer Logik und deren Anwendungen in der Informatik befassen und in denen die Inhalte der Vorlesung zum Themenkomplex Aussagenlogik wiederzufinden sind. Wir erwähnen hier (in alphabetischer Reihenfolge) zwei Standardwerke, in denen Teile der Vorlesung nachgelesen werden können.

1. Steffen Hölldobler: *Logik und Logikprogrammierung* Kolleg Synchron Publishers, 3. Auflage, 2003.
2. Uwe Schöning: *Logik für Informatiker* Spektrum Verlag, 5. Auflage, 2000.

Die Vorlesung setzt Kenntnisse über die Module Algorithmen und Datenstrukturen, Programmierung, sowie Mathematik voraus, wie sie in den betreffenden Lehrveranstaltungen für Studierende der Informatik, Medieninformatik und Informationssystemtechnik an der TUD vermittelt werden.

Das griechische Alphabet. Häufig werden griechische Buchstaben zur Bezeichnung mathematischer Objekte verwendet. Um eine Vorstellung zu vermitteln, was die einzelnen Zeichen bedeuten und wie sie ausgesprochen werden, ist in Abbildung 1 das griechische Alphabet aufgelistet. Die Symbole Γ, Δ, Θ, Λ, Ξ, Σ, Υ, Φ, Ψ und Ω sind griechische Großbuchstaben. Alle anderen Symbole sind Kleinbuchstaben.

α	alpha	ι	iota	ρ	rho
β	beta	κ	kappa	σ, Σ	sigma
γ, Γ	gamma	λ, Λ	lambda	τ	tau
δ, Δ	delta	μ	mu	υ, Υ	upsilon
ϵ, ε	epsilon	ν	nu	ϕ, φ, Φ	phi
ζ	zeta	ξ, Ξ	xi	χ	chi
η	eta	o	o	ψ, Ψ	psi
θ, ϑ, Θ	theta	π, ϖ, Π	pi	ω, Ω	omega

Abbildung 1: Das griechische Alphabet

1. Teil: Formale Sprachen und Automatentheorie

Formale Sprachen und Automatenmodelle

Formale Sprachen und Automatenmodelle wie endliche Automaten oder Kellerautomaten haben zahlreiche Anwendungen in der Informatik. Zu den wichtigsten zählt der Compilerbau. Das Grobschema der ersten Phasen des Übersetzungsvorgangs ist in Abbildung 2 skizziert.

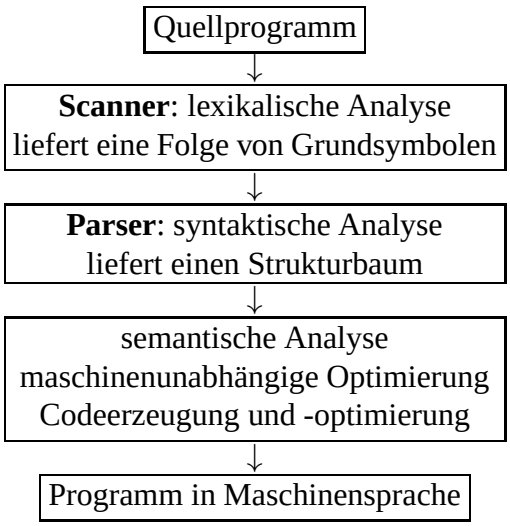


Abbildung 2: Grobschema des Übersetzungsvorgangs

Das Ziel der *lexikalischen Analyse* ist die Erkennung und Codierung der *Grundsymbole* sowie die Ausblendung bedeutungsloser Zeichen und Zeichenketten (z.B. Kommentare und Leerzeichen). Man unterscheidet vier Arten von Grundsymbolen:

Schlüsselwörter:	z.B. **IF**, **THEN**, **BEGIN** , etc.,
Spezialsymbole:	z.B. $<$, $+$, ;, etc.,
Identifier:	vom Benutzer gewählte Namen für Variablen, Programmodule, etc.
Literale:	vordefinierte Bezeichner für Werte gewisser Datentypen z.B. Zahlen.

Für Schlüsselwörter und Spezialsymbole ist keine spezielle Spezifikation notwendig, sie können explizit vorgegeben werden. Anders verhält es sich mit den Identifiern und Literalen. Diese werden durch *reguläre Sprachen* spezifiziert. Zu den unterschiedlichen Formalismen zur Darstellung regulärer Sprachen zählen Syntaxdiagramme, reguläre Ausdrücke, reguläre Grammatiken und endliche Automaten. Auf diese Formalismen werden wir in Kapitel 2 eingehen.

Das Programm geht als Folge von Grundsymbolen aus der lexikalischen Analyse hervor. In der *syntaktischen Analyse* geht es darum, die Struktur des Programms zu erkennen und eine abstrakte Baumstruktur des Quellprogramms zu erstellen. Dazu wird ein *Parser* verwendet, d.h. ein Analysealgorithmus, der die syntaktische Korrektheit des Programms (repräsentierende Folge von Grundsymbolen) bzgl. der syntaktischen Regeln, die der Programmiersprache zugrundeliegen, prüft. Die zulässige syntaktische Strukturierung von Programmen einer höheren Programmiersprache wird durch eine (deterministisch) *kontextfreie Grammatik* festgelegt.

Beispielsweise könnte man die Konstrukte einer an unsere Algorithmensprache anlehnenden imperativen Sprache durch Regeln wie in Abbildung 3 angeben. Im Anschluss an die syntaktische Analyse findet die *semantische Analyse* statt, deren Aufgaben sehr vielfältig sind (Erstellen eines Definitionsmoduls, Prüfung von Typverträglichkeit, Operatoridentifikation, etc.).

$$
\begin{array}{rcl}
\mathit{Stmts} & \to & \mathit{Assignment} \mid \mathit{CondStmt} \mid \mathit{Loop} \mid \mathit{Stmts}; \mathit{Stmts} \mid \ldots \\
\mathit{Assignment} & \to & \mathsf{Variable} := \mathit{Expr} \\
\mathit{CondStmt} & \to & \underline{\textbf{IF}}\ \mathit{BoolExpr}\ \underline{\textbf{THEN}}\ \mathit{Stmts}\ \underline{\textbf{FI}} \mid \ldots \\
\mathit{Loop} & \to & \underline{\textbf{WHILE}}\ \mathit{BoolExpr}\ \underline{\textbf{DO}}\ \mathit{Stmts}\ \underline{\textbf{OD}} \mid \\
 & & \underline{\textbf{REPEAT}}\ \mathit{Stmts}\ \underline{\textbf{UNTIL}}\ \mathit{BoolExpr} \mid \ldots \\
\mathit{BoolExpr} & \to & \mathit{Expr} < \mathit{Expr} \mid \mathit{Expr} \leqslant \mathit{Expr} \mid \ldots \\
 & & \mathit{BoolExpr} \wedge \mathit{BoolExpr} \mid \neg \mathit{BoolExpr} \mid \ldots \\
 & \vdots &
\end{array}
$$

Abbildung 3: Grammatik für Programme einer Pseudo-Programmiersprache

Die theoretischen Grundkonzepte von kontextfreien Sprachen werden wir in Kapitel 3 behandeln. Für deren Einsatz im Kontext des Compilerbaus sowie Details zur semantischen Analyse, Codegenerierung und Codeoptimierung verweisen wir auf Vorlesungen zum Thema Compilerbau.

Grammatiken versus Automaten. Ein wesentlicher Aspekt ist die Beziehung zwischen den verschiedenen Sprachtypen und Automatenmodellen (endliche Automaten, Kellerautomaten, Turingmaschinen). Während die Spezifikation einer formalen Sprache mit Hilfe einer Grammatik die Regeln festlegt, die ein Benutzer anwenden darf, um Wörter der Sprache zu bilden (z.B. den Programmcode zu formulieren), dienen die Automaten als Sprachakzeptoren. Die wesentliche Aufgabe des Automaten besteht darin, zu prüfen, ob ein gegebenes Eingabewort zu einer z.B. durch eine Grammatik gegebenen Sprache gehört. Diese Fragestellung taucht in natürlicherweise im Kontext von Übersetzern auf, da dort zu prüfen ist, ob der vom Benutzer verfasste Quellcode (das Eingabewort) den syntaktischen Regeln der betreffenden Programmiersprache (spezifiziert durch eine Grammatik oder ein Syntaxdiagramm) entspricht. Tatsächlich basiert die Funktionsweise eines Scanners auf einem *endlichen Automaten*; die eines Parsers auf einem *Kellerautomaten*.

Grundbegriffe für formale Sprachen

Bevor wir auf das formale Konzept von Grammatiken eingehen, fassen wir einige Notationen für Sprachen und Wörter zusammen, die teilweise bereits in der Mathematik-Vorlesung eingeführt wurden, und im Verlauf der Vorlesung häufig benutzt werden.

Alphabet. Ein *Alphabet* bezeichnet eine endliche nicht-leere Menge. Die Elemente eines Alphabets werden häufig *Zeichen* oder *Symbole* genannt. Alphabete bezeichnen wir meist mit griechischen Großbuchstaben wie Σ ("Sigma") oder Γ ("Gamma"). Wir schreiben $|\Sigma|$ für die Anzahl an Elementen in Σ.

Wort. Ein *endliches Wort* über Σ, kurz *Wort* genannt, ist eine endliche (eventuell leere) Folge von Zeichen in Σ, also von der Form $a_1 a_2 \dots a_k$, wobei $k \in \mathbb{N}$ und $\{a_1, \dots, a_k\} \subseteq \Sigma$. Für den Sonderfall $k = 0$ erhält man das *leere Wort*, welches mit dem griechischen Buchstaben ε ("epsilon") bezeichnet wird.

Die *Länge* eines Worts w ist die Anzahl an Zeichen, die in ihm vorkommen, und wird mit $|w|$ bezeichnet. Das leere Wort hat also die Länge 0 (d.h., $|\varepsilon| = 0$). Allgemein gilt $|a_1 a_2 \dots a_k| = k$.

Sind $w = a_1 \dots a_k$ und $u = b_1 \dots b_m$ Wörter über Σ, so bezeichnet $w \circ u$, oder kurz wu, das Wort $a_1 \dots a_k b_1 \dots b_m$, welches sich durch das Hintereinanderhängen von w und u ergibt. Man spricht auch von der *Konkatenation* von w und u.

Seien $w = a_1 \dots a_k$ und $u = b_1 \dots b_m$ Wörter über Σ.

- w wird *Präfix* von u genannt, falls es ein Wort v mit $u = wv$ gibt.
- w heißt *Suffix* von u, falls es ein Wort v mit $u = vw$ gibt.
- w heißt *Teilwort* von u, falls es Wörter v und x mit $u = vwx$ gibt.

Der Fall $v = \varepsilon$ (bzw. $x = \varepsilon$) ist dabei zugelassen. Also ist jedes Wort sowohl Präfix als auch Suffix (und auch Teilwort) von sich selbst. Ebenso ist ε zugleich Präfix und Suffix jedes Worts. Weiter ist klar, dass jedes Präfix und jedes Suffix von u zugleich ein Teilwort von u ist. Ein Wort w kann mehrfach als Teilwort von u vorkommen, z.B. hat $w = 01$ drei Vorkommen in $u = 11\underline{01}11\underline{0101}$. Wir sagen, dass w ein *echtes* Präfix von u ist, falls $w \neq u$ und w Präfix von u ist. In Analogie hierzu ist der Begriff "echtes Suffix" definiert. Ein Wort w wird echtes Teilwort von u genannt, falls w ein Teilwort von u ist und $w \neq u$.

Die Menge aller Wörter über einem Alphabet. Σ^* bezeichnet die Menge aller Wörter über Σ, einschliesslich dem leeren Wort ε. Die Bezeichnung Σ^+ wird für die Menge aller nichtleeren Wörter über Σ verwendet. Also:

$$\Sigma^+ \;=\; \Sigma^* \setminus \{\varepsilon\}$$

Ist $k \in \mathbb{N}$, so schreiben wir Σ^k für die Menge aller Wörter $w \in \Sigma^*$ der Länge k. Also $\Sigma^0 = \{\varepsilon\}$ und $\Sigma^k = \{a_1 \dots a_k : a_i \in \Sigma, i = 1, \dots, k\}$ für $k \geqslant 1$. Offenbar gilt:

$$\Sigma^* \;=\; \bigcup_{k \geqslant 0} \Sigma^k \quad \text{und} \quad \Sigma^+ \;=\; \bigcup_{k \geqslant 1} \Sigma^k$$

Mit Σ ist auch jede der Mengen Σ^k endlich. Genauer gilt: ist $|\Sigma| = n$, so ist $|\Sigma^k| = n^k$. Die Mengen Σ^* und Σ^+ sind jedoch unendlich, da es zu jedem $k \in \mathbb{N}$, $k \geqslant 1$, ein Wort der Länge k in Σ^+ (und Σ^*) gibt. (Beachte, dass Alphabete als nicht-leer vorausgesetzt werden.)

Sprache über einem Alphabet. Eine *formale Sprache*, kurz *Sprache* genannt, über einem Alphabet Σ ist eine beliebige Teilmenge L von Σ^*.

Man beachte, dass alle Sprachen abzählbar sind. Ist nämlich Σ ein Alphabet (also eine endliche nicht-leere Menge), so ist Σ^* als abzählbare Vereinigung der endlichen Mengen Σ^k, $k \geqslant 0$, abzählbar. Somit ist jede Teilmenge von Σ^*, also jede Sprache L über Σ, ebenfalls abzählbar.

Zur Erinnerung: eine Menge X heißt *abzählbar*, falls $X = \varnothing$ oder es eine surjektive Abbildung $h : \mathbb{N} \to X$ gibt. Ist $h : \mathbb{N} \to X$ surjektiv, so werden durch die unendliche Folge $h(0), h(1), h(2), \ldots$ alle Elemente aus X aufgezählt (eventuell manche Elemente mehrfach). Abzählbarkeit von X ist gleichbedeutend damit, dass X entweder endlich ist oder es eine bijektive Abbildung $f : X \to \mathbb{N}$ gibt. Aus den Mathematik-Vorlesungen ist bekannt, dass alle Teilmengen einer abzählbaren Menge abzählbar sind und abzählbare Vereinigungen abzählbarer Mengen wieder abzählbar sind. Beispiele für abzählbare Mengen sind alle endlichen Mengen, die Menge $\mathbb{N}$ der natürlichen Zahlen, aber auch die Menge $\mathbb{Q}$ der rationalen Zahlen. Beispiele für *überabzählbare* Mengen, d.h., Mengen, die nicht abzählbar sind, sind die Menge $\mathbb{R}$ der reellen Zahlen, die Potenzmenge $2^{\mathbb{N}}$ von $\mathbb{N}$ oder auch $\mathbb{N}^{\mathbb{N}}$. Hier sowie im Folgenden bezeichnet 2^X die Potenzmenge von X.

Operatoren für Sprachen. Zu den üblichen Verknüpfungsoperatoren für Sprachen L, L_1, L_2 über Σ zählen Vereinigung $L_1 \cup L_2$, Durchschnitt $L_1 \cap L_2$, Komplement $\overline{L}$ sowie Konkatenation $L_1 \circ L_2$ und Kleeneabschluss L^*. Die zuletzt genannten Sprachen sind wie folgt definiert. Seien L, L_1, L_2 Sprachen über Σ.

- Das *Komplement* von L ist durch $\overline{L} \stackrel{\text{def}}{=} \Sigma^* \setminus L$ gegeben.
- *Konkatenation:* $L_1 \circ L_2$, oder kurz $L_1 L_2$, bezeichnet die Menge aller Wörter $x_1 x_2$, welche sich durch Konkatenation eines Worts $x_1 \in L_1$ und eines Worts $x_2 \in L_2$ ergeben:

$$L_1 \circ L_2 \;=\; L_1 L_2 \;\stackrel{\text{def}}{=}\; \left\{ x_1 x_2 : x_1 \in L_1, x_2 \in L_2 \right\}$$

- Die Sprachen L^n für $n \in \mathbb{N}$ sind wie folgt definiert:

$$L^0 \stackrel{\text{def}}{=} \{\varepsilon\}, \quad L^{n+1} \stackrel{\text{def}}{=} L \circ L^n$$

  Für $n \geqslant 1$ gilt also $w \in L^n$ genau dann, wenn w die Form $w = x_1 x_2 \ldots x_n$ mit $\{x_1, \ldots, x_n\} \subseteq L$ hat.

- Der *Kleeneabschluss* von L ist durch

$$L^* \stackrel{\text{def}}{=} \bigcup_{n \geqslant 0} L^n$$

  definiert. Man spricht auch manchmal von dem *Sternoperator*. Die Elemente von L^* sind also genau diejenigen Wörter, die sich durch Aneinanderhängen von einer beliebigen endlichen Anzahl (eventuell 0) von Wörtern aus L ergeben. Der *Plusoperator* ist durch

$$L^+ \stackrel{\text{def}}{=} \bigcup_{n \geqslant 1} L^n$$

  definiert. Beachte, dass stets $\varepsilon \in L^*$ gilt. Dagegen gilt $\varepsilon \in L^+$ nur dann, wenn $\varepsilon \in L$ gilt. Daher gilt $L^+ = L^*$, falls $\varepsilon \in L$, und $L^+ = L^* \setminus \{\varepsilon\}$, falls $\varepsilon \notin L$.

Beispielsweise kann man die Menge von natürlichen Dezimalzahlen ohne führende Nullen durch die Sprache

$$L_{\mathit{Dezimalzahl}} = \{0\} \cup \{1,2,\ldots,9\} \circ \{0,1,2,\ldots,9\}^*$$

über dem Alphabet $\Sigma = \{0,1,2,\ldots,9\}$ charakterisieren.

Neben den üblichen Gesetzen der Mengenalgebra (z. B. Kommutativität und Assoziativität für Vereinigung und Durchschnitt) gelten z. B. folgende Gesetze für die Konkatenation. Dabei sind L_1, L_2, K beliebige Sprachen.

$$\begin{array}{rclcrcl} (L_1 \cup L_2)K & = & L_1K \cup L_2K & \qquad & \{\varepsilon\}K & = & K\{\varepsilon\} = K \\ K(L_1 \cup L_2) & = & KL_1 \cup KL_2 & & \varnothing K & = & K\varnothing = \varnothing \end{array}$$

Weiter gilt für den Kleeneabschluss und Plusoperator $K^* = (K \setminus \{\varepsilon\})^* = K^+ \cup \{\varepsilon\}$ und $K^+ = KK^* = K^*K$.

Abgeschlossenheit unter Verknüpfungsoperatoren. Sei $\mathfrak{K}$ eine Klasse von Sprachen. Die Klasse $\mathfrak{K}$ ist unter der Vereinigung abgeschlossen, falls für je zwei Sprachen L_1 und L_2 gilt:

$$\text{Falls } L_1 \in \mathfrak{K} \text{ und } L_2 \in \mathfrak{K}, \text{ so ist auch } L_1 \cup L_2 \in \mathfrak{K}.$$

Dabei kann man o. E. annehmen, dass L_1 und L_2 dasselbe Alphabet zugrundeliegt. Sind $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$, dann betrachten wir das Alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$ und fassen L_1, L_2 und $L_1 \cup L_2$ als Sprachen über dem Alphabet Σ auf. Entsprechende Bedeutung haben die Begriffe Abgeschlossenheit unter Durchschnitt, Komplement, Konkatenation oder Kleeneabschluss.

1 Grammatiken

Grammatiken geben Regeln vor, nach denen Wörter einer Sprache generiert werden können. Die Grundidee ist ein *Termersetzungsprozess*, bei dem einzelne Zeichen oder Zeichenketten – gebildet aus Terminalzeichen und Hilfsvariablen (Nichtterminalen) – durch andere Wörter ersetzt werden dürfen bis ein Wort entsteht, das ausschließlich Terminalzeichen enthält. Die Terminalzeichen sind die Symbole der Sprache. Beispielsweise wird man die Symbole "+", "−", $0, \ldots, 9$ als Terminalzeichen verwenden, um ganzzahlige Dezimalzahlen zu spezifizieren.

Beispiel 1.1 (Spezifikation für "Identifier"). Bevor wir die formale Definition einer Grammatik angeben, erläutern wir die Grundidee an einem einfachen Beispiel. Viele Programmiersprachen lassen als Identifier genau diejenigen Wörter zu, die aus einer beliebig langen Zeichenkette, gebildet aus Buchstaben $a, b, \ldots, z$ sowie den Ziffern $0, 1, \ldots, 9$ bestehen und mit einem Buchstaben beginnen.

Jeder *Identifier* besteht aus einem *Buchstaben*, gefolgt von
beliebig vielen *Buchstaben* oder *Ziffern*.

Mit den Operatoren $\circ$ (Konkatenation), Vereinigung und dem Kleeneabschluss können wir die Identifier durch die Sprache

$$L_{Idf} = \mathfrak{B} \circ (\mathfrak{B} \cup \mathfrak{Z})^*$$

charakterisieren. Dabei ist $\mathfrak{B} = \{a, b, \ldots, z\}$ die Menge (endliche Sprache) der Buchstaben und $\mathfrak{Z} = \{0, 1, \ldots, 9\}$. (Wir nehmen hier an, dass der Begriff "Buchstabe" im Sinn von "Kleinbuchstabe" verwendet wird.) Alternativ können wir L_{Idf} durch die in Abbildung 4 angegebenen *Regeln* darstellen.

$$\begin{array}{lcl} \mathit{Idf} & \to & BA \\ A & \to & \varepsilon \mid BA \mid ZA \\ B & \to & a \mid b \mid c \mid \ldots \mid z \\ Z & \to & 0 \mid 1 \mid 2 \mid \ldots \mid 9 \end{array}$$

Abbildung 4: Grammatik G_{Idf}

Intuitiv steht *Idf* für eine Variable, die über alle Identifier quantifiziert, während das Symbol A alle Zeichenketten repräsentiert, die aus Buchstaben und Ziffern gebildet werden (einschließlich der Zeichenkette der Länge 0, die wir durch das leere Wort ε repräsentieren). Die Symbole B und Z stehen intuitiv für alle Buchstaben bzw. Ziffern. Die erste Regel $\mathit{Idf} \to BA$ besagt, dass wir einen Identifier erhalten, indem wir zuerst einen Buchstaben wählen und an diesen ein aus dem Symbol A hergeleitetes Wort hängen. Die zweite Zeile ist eine Kurzschreibweise für die drei Regeln $A \to \varepsilon$, $A \to BA$ und $A \to ZA$. Diese drei Regeln erlauben es, das Symbol A durch eines der drei Wörter ε, BA oder ZA zu ersetzen. Dieser Vorgang kann beliebig oft wiederholt werden bis schließlich alle generierten B's und Z's durch konkrete Buchstaben bzw. Ziffern ersetzt werden. ■

Eine Grammatik besteht aus einer Menge von Variablen, auch Nichtterminale genannt, die repräsentativ für herleitbare Wörter stehen. Weiter gibt es eine Menge von Terminalsymbolen. Dies sind die Zeichen des Alphabets Σ, über dem wir eine Sprache definieren möchten. Die Regeln (auch Produktionen genannt) sind Paare (x, y), üblicherweise in Pfeilnotation $x \to y$ geschrieben, wobei x und y Wörter gebildet aus Terminalzeichen und Variablen sind. Die einzige Forderung ist, dass das Wort x auf der linken Seite mindestens ein Nichtterminal enthält.

Definition 1.2 (Grammatik). Eine Grammatik ist ein Tupel $G = (V, \Sigma, \mathcal{P}, S)$ bestehend aus

- einer endlichen Menge V von *Variablen* (auch *Nichtterminale* genannt),
- einem Alphabet Σ mit $V \cap \Sigma = \varnothing$,
- einem *Startsymbol* $S \in V$,
- einer endlichen Menge $\mathcal{P}$ bestehend aus Paaren (x, y) von Wörtern $x, y \in (V \cup \Sigma)^*$ mit $x \notin \Sigma^*$, d.h., x enthält wenigstens eine Variable.

Σ wird auch *Terminalalphabet* genannt. Die Elemente von Σ werden als *Terminalsymbole* oder *Terminalzeichen*, oder auch *Terminale*, bezeichnet. Die Elemente von $\mathcal{P}$ werden *Produktionen* oder *Regeln* genannt. Man nennt $\mathcal{P}$ daher auch *Produktionssystem* oder *Regelsystem*. ■

Das Produktionssystem wird üblicherweise nicht als Menge von Wortpaaren angegeben. Stattdessen wird eine Pfeilnotation verwendet. Üblich ist die Schreibweise $x \to y$ anstelle von $(x, y) \in \mathcal{P}$. Entsprechend schreibt man $x \not\to y$, falls $(x, y) \notin \mathcal{P}$. Regeln der Form $x \to \varepsilon$ werden *ε-Regeln* genannt. Regeln der Form $A \to B$, wobei A und B Nichtterminale sind, werden auch *Kettenregeln* genannt. *Terminalregeln* sind Regeln der Form $A \to a$, wobei A ein Nichtterminal und a ein Terminalzeichen ist. Ferner ist

$$x \to y_1 \,|\, y_2 \,|\, \ldots \,|\, y_n$$

eine Kurzschreibweise für die Regeln $x \to y_1$, $x \to y_2$, …, $x \to y_n$.[1] Wir werden im Folgenden häufig auf die explizite Angabe der Komponenten V, Σ und S einer Grammatik verzichten, sondern nur das zugehörige Produktionssystem angeben. Wir verwenden Großbuchstaben $S, A, B, \ldots$ für die Nichtterminale und Kleinbuchstaben $a, b, c, \ldots$ am Anfang des (gewöhnlichen) Alphabets für die Terminalzeichen. Werden keine weiteren Angaben gemacht, dann ist S das Startsymbol. Für Wörter über $V \cup \Sigma$ verwenden wir meist Kleinbuchstaben wie u, v, w, x, y, z am Ende des Alphabets.

[1] Die Backus-Naur-Form, kurz BNF, ist eine weit verbreitete syntaktische Notation, um Produktionssysteme für Grammatiken anzugeben. Neben der oben erläuterten Notation $x \to y_1 \,|\, y_2 \,|\, \ldots \,|\, y_n$ stellt die BNF noch weitere Kurzschreibweisen zur Verfügung. Eckige Klammern stehen für Optionen, geschweifte Klammern stehen für beliebig viele Wiederholungen, d.h.,

$$\begin{aligned} x &\to u[v]w \quad \text{steht für} \quad x \to uw \,|\, uvw \\ x &\to u\{v\}w \quad \text{steht für} \quad x \to uBw, \quad B \to \varepsilon \,|\, v \,|\, vB, \end{aligned}$$

wobei B eine neue Variable ist. Ferner ist in der BNF-Notation die Verwendung des Symbols "::=" anstelle des Pfeils "$\to$" gebräuchlich.

Beispiel 1.3 (Die Grammatik G_{Idf}). Zunächst machen wir uns die formale Definition einer Grammatik am Beispiel der Regeln für Identifier klar, siehe Abbildung 4 auf Seite 8. Die Variablenmenge ist $V = \{S, A, B, Z\}$, wobei $S = \mathit{Idf}$. Das Startsymbol ist $S = \mathit{Idf}$. Das Terminalalphabet ist die Menge der (Klein-)Buchstaben und Ziffern, also $\Sigma = \{a, b, \ldots, z, 0, 1, \ldots, 9\}$. Die oben angegebenen Regeln stehen für das Produktionssystem

$$\begin{aligned} \mathcal{P} \;=\; \{\; & (S, BA), \quad (A, \varepsilon), \quad (A, BA), \quad (A, ZA) \\ & (B, a), \quad (B, b), \quad \ldots, \quad (B, z) \\ & (Z, 0), \quad (Z, 1), \quad \ldots, \quad (Z, 9) \quad \} \end{aligned}$$

In Pfeilnotation entspricht dies den Regeln $S \to BA$, $A \to \varepsilon \,|\, BA \,|\, ZA$ und $B \to a \,|\, b \,|\, \ldots \,|\, z$ sowie $Z \to 0 \,|\, 1 \,|\, \ldots \,|\, 9$. ■

Das Anwenden einer Regel bedeutet, dass man in einem bereits hergeleiteten Wort das Teilwort x auf der linken Seite einer Regel $x \to y$ durch das Wort y auf der rechten Seite ersetzt. Dies wird durch eine Relation $\Rightarrow$ bestehend aus Wortpaaren über dem Alphabet $V \cup \Sigma$ formalisiert. Üblich ist die Schreibweise $u \Rightarrow v$ anstelle von $(u, v) \in \Rightarrow$.

Definition 1.4 (Her-/Ableitungsrelationen $\Rightarrow$ und $\Rightarrow^*$). Sei $G = (V, \Sigma, \mathcal{P}, S)$ eine Grammatik. Die Relation $\Rightarrow$ ist wie folgt definiert. Seien u und v zwei Wörter über dem Alphabet $V \cup \Sigma$. Dann gilt $u \Rightarrow v$ genau dann, wenn es Wörter x, y, w, $z \in (V \cup \Sigma)^*$ gibt, so dass

$$u = wxz, \quad v = wyz, \quad x \to y.$$

Die *Herleitungsrelation* von G (auch *Ableitungsrelation* genannt) $\Rightarrow^*$ bezeichnet die reflexive, transitive Hülle von $\Rightarrow$. Diese ist wie folgt definiert. Sind $u, v \in (V \cup \Sigma)^*$, so gilt

$$u \Rightarrow^* v$$

genau dann, wenn es eine Folge $u_0, u_1, \ldots, u_n$ von Wörtern $u_i \in (V \cup \Sigma)^*$ gibt, so dass

$$u = u_0 \;\Rightarrow\; u_1 \;\Rightarrow\; u_2 \;\Rightarrow\; \ldots \;\Rightarrow\; u_{n-1} \;\Rightarrow\; u_n = v.$$

Man spricht auch von einer Herleitung (oder Ableitung) von v aus u. Die *Länge* einer Herleitung ist durch die Anzahl an Regelanwendungen gegeben. Obige Herleitung hat also die Länge n. Der Spezialfall $n = 0$ ist zugelassen. Es gilt also $u \Rightarrow^* u$ für alle Wörter $u \in (V \cup \Sigma)^*$. Falls $S \Rightarrow^* v$ (wobei S das Startsymbol ist), so sagen wir, dass v in G herleitbar ist. ■

Es gilt also $u \Rightarrow v$ genau dann, wenn v aus u durch Anwendung genau einer Regel hervorgeht, d.h., wenn es eine Regel $x \to y$ gibt, so dass v aus u entsteht, indem ein Vorkommen von x in u durch das Wort y ersetzt wird. Weiter gilt $u \Rightarrow^* v$ genau dann, wenn sich v aus u durch Anwenden von beliebig vielen (null oder mehreren) Regeln aus u ergeben kann. Das Symbol $\Rightarrow$ bezeichnet also die "Ein-Schritt-Herleitungsrelation", während $\Rightarrow^*$ für die "Mehrschritt-Herleitungsrelation" steht.

Zur Verdeutlichung, welche Grammatik zugrundeliegt, indizieren wir manchmal die Symbole $\Rightarrow$ und $\Rightarrow^*$ mit der betreffenden Grammatik, d.h., wir schreiben an manchen Stellen auch $\Rightarrow_G$ statt $\Rightarrow$ und $\Rightarrow^*_G$ statt $\Rightarrow^*$. Vereinzelt verwenden wir auch die Schreibweise $x \to_G y$ statt $x \to y$, um zu verdeutlichen, dass es sich hierbei um eine Regel der Grammatik G handelt.

Beispielsweise hat die Grammatik G_{Idf} für "Identifier" in Abbildung 4 auf Seite 8 folgende Herleitung des Worts b46.

$$\begin{aligned} \mathit{Idf} &\Rightarrow BA \Rightarrow BZA \Rightarrow BZZA \\ &\Rightarrow BZZ \Rightarrow bZZ \Rightarrow b4Z \Rightarrow b46 \end{aligned}$$

Daher gilt $\mathit{Idf} \Rightarrow^* b46$.

Definition 1.5 (Erzeugte Sprache $\mathcal{L}(G)$). Sei $G = (V, \Sigma, \mathcal{P}, S)$ eine Grammatik. Die durch G erzeugte Sprache

$$\mathcal{L}(G) \stackrel{\text{def}}{=} \left\{ w \in \Sigma^* : S \Rightarrow^* w \right\}$$

besteht aus allen Wörtern $w \in \Sigma^*$, die sich aus dem Startsymbol S ableiten lassen. $\mathcal{L}(G)$ wird häufig auch die von G *definierte* oder *generierte* Sprache genannt. ■

Wir betrachten die Grammatik G_{Idf} in Abbildung 4 auf Seite 8. Die erzeugte Sprache $\mathcal{L}(G_{Idf})$ ist – wie erwartet – die Sprache bestehend aus allen Zeichenketten, die mit einem Buchstaben beginnen und dann mit einem beliebig langen (eventuell leeren) Wort – gebildet aus Buchstaben und Ziffern – enden. Aufgrund der einfachen Struktur der Grammatik G_{Idf} kann diese Aussage zwar als offensichtlich angesehen werden. Dennoch wollen wir exemplarisch am Beispiel einer vereinfachten Version G von G_{Idf} erläutern, wie der Nachweis, dass die erzeugte Sprache einer Grammatik mit einer gegebenen Sprache L übereinstimmt, erbracht werden kann. Grammatik G verwendet nur zwei Termimale b (anstelle aller Buchstaben) und C (anstelle aller Ziffern) und die Nichtterminalen S (statt *Idf*), A und B (wie in G_{Idf}) und C (statt Z). Die Regeln von G sind wie folgt:

$$\begin{array}{lcl@{\qquad}lcl} S & \to & BA & B & \to & b \\ A & \to & \varepsilon \mid BA \mid CA & C & \to & c \end{array}$$

Die Behauptung ist nun, dass $\mathcal{L}(G) = \left\{ bx : x \in \{b,c\}^* \right\}$. Es ist offensichtlich, dass ε in keiner der beiden Sprachen liegt.

"$\supseteq$" Zu zeigen ist, dass jedes nicht-leere Wort w über dem Terminalalphabet Σ, dessen erstes Zeichen ein Symbol b ist, eine Herleitung in G besitzt. Hierzu genügt es eine Herleitung von w in G anzugeben. Sei $w = bx$, wobei $x = d_1 \ldots d_k$ mit $d_i \in \{b,c\}$. Für $i \in \{1,\ldots,k\}$ sei

$$D_i = \begin{cases} B & : \text{ falls } d_i = b \\ C & : \text{ falls } d_i = c \end{cases}$$

Durch Anwenden der Regeln $S \to BA$ und $A \to D_iA$ für $i = 1,2,\ldots,k$ und schliesslich $A \to \varepsilon$ sowie die Regeln $B \to b$ und $D_i \to d_i$ für $i = 1,2,\ldots,k$ erhalten wir folgende Herleitung von w in G:

$$\begin{aligned} S &\Rightarrow BA \Rightarrow BD_1A \Rightarrow BD_1D_2A \Rightarrow \ldots \Rightarrow BD_1D_2\ldots D_kA \\ &\Rightarrow BD_1D_2\ldots D_k \Rightarrow bD_1D_2\ldots D_k \Rightarrow \ldots \Rightarrow bd_1D_2D_3\ldots D_k \\ &\Rightarrow bd_1d_2D_3\ldots D_k \Rightarrow \ldots \Rightarrow bd_1d_2\ldots d_k = w \end{aligned}$$

"$\subseteq$" Zu zeigen ist, dass alle aus G herleitbaren Wörter über dem Terminalalphabet $\{b,c\}$ nicht-leer sind und mit dem Symbol b beginnen. Der Beweis hierfür kann erbracht werden, indem man folgende Eigenschaft für alle in G herleitbaren Wörter über $\{A,B,C\} \cup \{b,c\}$ nachweist.

Aus $S \Rightarrow^* y \in (\{A,B,C\} \cup \{b,c\})^*$ folgt $y = u v_1 \dots v_{k-1} v_k$, wobei $k \geqslant 1$ und

(i) $u \in \{B\} \cup \{b\}$
(ii) $v_1, \dots, v_{k-1} \in \{B,C\} \cup \{b,c\}$
(iii) $v_k \in \{A,B,C\} \cup \{b,c\}$.

Diese Aussage kann durch Induktion nach der Länge einer Herleitung von y verifiziert werden. Ist nun $y \in \mathcal{L}(G)$, so gilt $S \Rightarrow^* y \in b\{b,c\}^*$ und somit $u = b$ und $v_1, \dots, v_{k-1}, v_k \in \{b,c\}$.

Definition 1.6 (Äquivalenz von Grammatiken). Seien $G_1 = (V_1, \Sigma, \mathcal{P}_1, S_1)$, $G_2 = (V_2, \Sigma, \mathcal{P}_2, S_2)$ zwei Grammatiken mit demselben Terminalalphabet Σ. G_1 und G_2 heißen *äquivalent*, falls $\mathcal{L}(G_1) = \mathcal{L}(G_2)$. ■

Beispiel 1.7 (Grammatik für Identifier). Die Grammatik G_{Idf} aus Abbildung 4 (Seite 8) ist äquivalent zu der Grammatik G'_{Idf} mit folgendem Produktionssystem.

$$\begin{aligned} S' &\to aA \mid bA \mid \dots \mid zA \\ A &\to \varepsilon \mid aA \mid bA \mid \dots \mid zA \mid 0A \mid \dots \mid 9A \end{aligned}$$

Dabei ist S' das Startsymbol von G'_{Idf}. ■

Grammatiken und die zugehörigen Sprachklassen werden wie in nachfolgender Definition klassifiziert. Diese Klassifizierung geht auf Chomsky (1956) zurück und wird daher als Chomsky Hierarchie bezeichnet.

Definition 1.8 (Grammatik-/Sprachtypen der Chomsky Hierarchie). Sei $G = (V, \Sigma, \mathcal{P}, S)$ eine Grammatik. G heißt

- *kontextsensitiv* oder *vom Typ 1*, wenn für alle Regeln $x \to y$ gilt: $|x| \leqslant |y|$.
- *kontextfrei* oder *vom Typ 2*, wenn alle Regeln die Form $A \to y$ haben, wobei A eine Variable und y ein beliebiges (eventuell leeres) Wort über $V \cup \Sigma$ ist.
- *regulär* oder *vom Typ 3*, wenn alle Regeln von der Form

  $$A \to \varepsilon \quad \text{oder} \quad A \to a \quad \text{oder} \quad A \to aB$$

  sind, wobei A, B Variablen (d.h., $A, B \in V$) und $a \in \Sigma$ ein Terminalzeichen sind.

Werden keine Einschränkungen gemacht, so spricht man auch von einer Grammatik *vom Typ 0*. Für Grammatiken, für die das leere Wort ableitbar ist, werden wir in Definition 1.12 einen Sonderfall angeben, der weitere Grammatiken zur Klasse der kontextsensitiven Grammatiken (Typ 1) zulässt. Im Folgenden verwenden wir häufig die Abkürzung CFG für die englische Bezeichung “contextfree grammar”.

Eine Sprache vom Typ i ist eine Sprache L, für die es eine Grammatik vom Typ i mit $\mathcal{L}(G) = L$ gibt. Dabei ist $i \in \{0,1,2,3\}$. Sprachen vom Typ 1 werden kontextsensitiv genannt. Sprachen vom Typ 2 heißen kontextfrei, während Sprachen vom Typ 3 regulär genannt werden. ■

Unmittelbar aus der Definition folgt, dass jede Grammatik vom Typ i ($i = 1,2,3$) zugleich eine Grammatik vom Typ 0 ist. Weiter ist klar, dass jede reguläre Grammatik zugleich kontextfrei ist.

Beispiel 1.9 (Grammatiktypen). Wir betrachten Grammatiken mit dem Terminalalphabet $\Sigma = \{a, b, c\}$ und der Variablenmenge $V = \{S, A, B\}$. Folgende Grammatik ist vom Typ 0, aber nicht vom Typ 1,2 oder 3.

$$S \to AS \mid ccSb, \quad AS \to Sbb, \quad cSb \to c, \quad cS \to a,$$

Folgende Grammatik ist kontextsensitiv, aber nicht kontextfrei.

$$S \to aAb, \quad aA \to aAbc, \quad Abc \to aacc$$

Ein Beispiel für eine kontextfreie Grammatik, die nicht regulär ist, ist

$$S \to AB, \quad A \to aABb \mid \varepsilon \mid bc, \quad B \to A.$$

Die Grammatik mit den Regeln

$$S \to b, \quad A \to \varepsilon \mid a \mid aA \mid aS$$

ist regulär. Die Grammatik G_{Idf} aus Abbildung 4 auf Seite 8 ist zwar nicht regulär; jedoch ist sie äquivalent zu der regulären Grammatik G'_{Idf} aus Beispiel 1.7. Daher ist die Sprache der "Identifier" regulär. ■

Eine informelle Erklärung für die Begriffe "kontextsensitiv" und "kontextfrei" ist wie folgt. Oftmals haben kontextsensitive Regeln die Form $uAw \to uvw$. Diese Regel erlaubt es, Variable A im Kontext von u und w durch v zu ersetzen. Man beachte jedoch, dass die Definition kontextsensitiver Grammatiken auch völlig andere Regeln zulässt, die dieser intuitiven Erklärung nicht standhalten. So ist z.B. auch die Regel $aAB \to cAd$ legitim in Grammatiken vom Typ 1. Mit der Regel $aAB \to cAd$ darf das Wort aAB durch cAd ersetzt werden, was einer Änderung des Kontexts (statt einer kontextabhängigen Änderung) entspricht. Der Begriff "kontextfrei" erklärt sich daraus, dass in Grammatiken vom Typ 2 die Nichtterminale völlig unabhängig vom Kontext, in dem sie auftauchen, ersetzt werden dürfen.

Beispiel 1.10 (CFG für arithmetische Ausdrücke). Arithmetische Ausdrücke über den natürlichen Zahlen werden aus Konstanten $k \in \mathbb{N}$, Variablen vom Typ Integer und den Basisoperatoren $+$ (Addition) und $*$ (Multiplikation) gebildet. Zur Vereinfachung gehen wir hier von einer festen endlichen Variablenmenge $\{x_1, \ldots, x_n\}$ aus. Die Syntax vollständig geklammerter arithmetischer Ausdrücke kann durch eine kontextfreie Grammatik G_{aA} beschrieben werden. Das Alphabet der Terminalzeichen ist

$$\Sigma = \{0,1,2,3,4,5,6,7,8,9\} \cup \{x_1, \ldots, x_n\} \cup \{+, *, (,)\}.$$

Wir verwenden drei Nichtterminale: das Startsymbol S und Nichtterminale A, B zur Erzeugung der Konstanten, dargestellt als Dezimalzahlen ohne führende Nullen. Die Regeln der Grammatik G_{aA} sind:

$$\begin{aligned} S &\to A \mid x_1 \mid \ldots \mid x_n \mid (S+S) \mid (S*S) \\ A &\to 0 \mid 1B \mid 2B \mid \ldots \mid 9B \\ B &\to \varepsilon \mid 0B \mid 1B \mid \ldots \mid 9B \end{aligned}$$

Beispielsweise ist

$$\begin{aligned} S &\Rightarrow (S+S) \Rightarrow ((S*S)+S) \\ &\Rightarrow ((x_1*S)+S) \\ &\Rightarrow ((x_1*x_2)+S) \\ &\Rightarrow ((x_1*x_2)+x_3) \end{aligned}$$

eine Ableitung der Formel $\alpha = ((x_1*x_2)+x_3)$ in G_{aA}. Aus G_{aA} sind nur vollständig geklammerte Ausdrücke herleitbar, nicht aber z.B. Ausdrücke wie $x_1*x_2+x_3$ (in der Bedeutung von $((x_1*x_2)+x_3)$) oder $x_1+x_2+x_3$ (was aufgrund des Assoziativgesetzes für die Addition eine sinnvolle Schreibweise ist). Für das Entfernen überflüssiger Klammern kann die oben angegebene Grammatik G_{aA} um Regeln vom Typ 0 erweitert werden, etwa $(S+(S+S)) \to S+S+S$. Statt bereits gesetzte Klammern zu entfernen, können korrekt (eventuell sparsam) geklammerte arithmetische Ausdrücke auch durch eine kontextfreie Grammatik erzeugt werden. Die CFG $G_{aA'}$ verwendet 6 Nichtterminale T, U, A, B, C und das Startsymbol S und folgende Regeln:

$$\begin{array}{lcl} S \to T \mid S+T & \qquad & A \to 0 \mid 1B \mid 2B \mid \ldots \mid 9B \\ T \to U \mid S*T & & B \to \varepsilon \mid 0B \mid 1B \mid \ldots \mid 9B \\ U \to (S) \mid A \mid C & & C \to x_1 \mid x_2 \mid \ldots \mid x_n \end{array}$$

Klammerlose Summen $\alpha_1+\alpha_2$ sind aus S herleitbar, während klammerlose Produkte aus T hergeleitet werden können. Das Nichtterminal U dient zum Setzen von Klammern und der Generierung der Terme: Konstanten (dargestellt als Dezimalzahlen ohne führende Nullen) können mittels der Nichtterminale A und B erzeugt werden. Das Nichtterminal C dient der Erzeugung der Variablen $x_1, \ldots, x_n$. ■

Beispiel 1.11 (Typ 1 Grammatik). Die Sprache $L = \left\{ a^n b^n c^n : n \in \mathbb{N}, n \geqslant 1 \right\}$ wird durch die kontextsensitive Grammatik G mit folgenden Regeln erzeugt:

$$\begin{array}{llllll} S \to aSBC \mid aBC & & aB \to ab & & bC \to bc \\ CB \to BC & & bB \to bb & & cC \to cc \end{array}$$

Dabei sind S, B, C Nichtterminale und S das Startsymbol. Beispielsweise ist

$$\begin{aligned} S &\Rightarrow aSBC \Rightarrow aaBCBC \\ &\Rightarrow aaBBCC \Rightarrow aabBCC \\ &\Rightarrow aabbCC \Rightarrow aabbcC \\ &\Rightarrow aabbcc \end{aligned}$$

eine Herleitung für das Wort $aabbcc = a^2b^2c^2$. Wir skizzieren nun, wie der Nachweis, dass $\mathcal{L}(G) = L$ gilt, erbracht werden kann. Zum Beweis der Inklusion "$\supseteq$" kann durch Induktion nach n gezeigt, dass alle Wörter $a^n b^n c^n$, $n \geqslant 1$, aus S abgeleitet werden können. (Exemplarisch wurde der Fall $n = 2$ oben betrachtet.) Etwas schwieriger ist der Nachweis der Inklusion "$\subseteq$". Zunächst betrachten wir ein Wort $w \in \{S, B, C, a, b, c\}^*$ gebildet aus Nichtterminalen und Terminalen. Ist $\sigma \subseteq \{a, b, c, B, C\}$, so schreiben wir $\#(\sigma, w)$ um die Anzahl an Vorkommen des Symbols σ in w zu bezeichnen. Durch Inspektion aller Regeln stellt man fest, dass für alle $w \in \{S, B, C, a, b, c\}^*$ mit $S \Rightarrow^* w$ folgende beiden Aussagen (1) und (2) gelten:

(1) $\#(a, w) = \#(b, w) + \#(B, w) = \#(c, w) + \#(C, w)$

(2) Ist $w = uav$, dann ist u von der Form a^n für ein $n \geqslant 0$. Ist $w = ubv$, so enthält u keines der Symbole c, S oder C. Besteht w nur aus Terminalzeichen, also $w \in \mathcal{L}(G) \subseteq \{a, b, c\}^*$, so hat w die Gestalt $a^n b^m c^k$, wobei $n, m, k \geqslant 1$.

Aus (1) folgt, dass jedes Wort $w \in \mathcal{L}(G) \subseteq \{a, b, c\}^*$ ebenso viele a's wie b's und c's enthält. Zusammen mit Aussage (2) folgt, dass jedes Wort $w \in \mathcal{L}(G)$ die Gestalt $a^n b^n c^n$ für ein $n \geqslant 1$ hat. ■

Der ε-Sonderfall für Typ 1 und ε-Freiheit. Intuitiv erwartet man, dass Sprachen vom Typ i zugleich Sprachen vom Typ $i-1$ sind. Für die Fälle $i = 1$ oder $i = 3$ ist diese Aussage klar. Mit der Forderung $|x| \leqslant |y|$ für alle Produktionen $x \to y$ einer kontextsensitiven Grammatik, kann das leere Wort niemals in der durch eine kontextsensitive Grammatik definierten Sprache $\mathcal{L}(G)$ liegen. Insofern ist jede kontextfreie Sprache, die ε enthält, nicht kontextsensitiv im Sinne von Definition 1.8 (Seite 12). Im Folgenden lockern wir die Definition kontextsensitiver Grammatiken/Sprachen etwas auf und berücksichtigen den Spezialfall von Sprachen, die das leere Wort enthalten.

Definition 1.12 (ε-Sonderfall für kontextsensitive Grammatiken). Eine Grammatik $G = (V, \Sigma, \mathcal{P}, S)$ mit $\varepsilon \in \mathcal{L}(G)$ heißt *kontextsensitiv* oder *vom Typ 1*, falls gilt:

(1) $S \to \varepsilon$

(2) Für alle Regeln $x \to y$ in $\mathcal{P} \setminus \{S \to \varepsilon\}$ ist $|x| \leqslant |y|$ und S kommt in y nicht vor.

Die zugehörige Sprache $\mathcal{L}(G)$ wird als Sprache vom Typ 1 oder kontextsensitive Sprache bezeichnet. ■

Z.B. ist die Grammatik mit dem Startsymbol S und den Regeln

$$S \to \varepsilon \mid Aba, \quad Ab \to bb$$

kontextsensitiv; nicht aber die Grammatik $S \to \varepsilon \mid Sba$.

Offenbar ist eine Sprache L genau dann kontextsensitiv, wenn die Sprache $L \setminus \{\varepsilon\}$ kontextsensitiv im Sinne von Definition 1.8 ist, also wenn es eine Grammatik G gibt, so dass $\mathcal{L}(G) = L \setminus \{\varepsilon\}$ und so dass $|x| \leqslant |y|$ für jede Regel $x \to y$ in G.

Wir zeigen nun, dass sich jede kontextfreie Grammatik in eine kontextsensitive Grammatik transformieren lässt, welche die selbe Sprache erzeugt. Problematisch sind die in CFG zulässigen ε-Regeln, d.h., Regeln der Form $A \to \varepsilon$. Im Folgenden zeigen wir, dass diese ε-Regeln in kontextfreien Grammatiken weitgehend entfernt werden können. Hierzu geben wir ein Verfahren an, das eine gegebene CFG G durch eine äquivalente kontextfreie Grammatik ersetzt, welche entweder gar keine ε-Regeln enthält oder gemäß des oben genannten ε-Sonderfalls kontextsensitiv ist. Man spricht dann auch von ε-Freiheit:

Definition 1.13 (ε-Freiheit kontextfreier Grammatiken). Sei $G = (V, \Sigma, \mathcal{P}, S)$ eine CFG. G heißt *ε-frei*, falls gilt:

(1) Aus $A \to \varepsilon$ folgt $A = S$.

(2) Falls $S \to \varepsilon$, dann gibt es *keine* Regel $A \to y$, so dass S in y vorkommt. ∎

Bis auf die Regel $S \to \varepsilon$ sind also keine ε-Regeln in ε-freien CFG zugelassen. Falls $S \to \varepsilon$, dann stellt Bedingung (2) sicher, dass diese nur zur Herleitung des leeren Worts angewandt, aber in keiner anderen Ableitung eingesetzt werden kann. Daher sind ε-freie CFG kontextsensitiv.

Algorithmus 1 CFG $G \rightsquigarrow$ äquivalente ε-freie CFG G'

Berechne $V_\varepsilon = \{A \in V : A \Rightarrow^* \varepsilon\}$ (* Markierungsalgorithmus (s.u.) *)
Entferne alle ε-Regeln aus G.
WHILE es gibt eine Regel $B \to xAy$ mit $A \in V_\varepsilon$, $|xy| \geqslant 1$ und $B \not\to xy$ **DO**
  wähle eine solche Regel und füge $B \to xy$ als neue Regel ein.
OD
IF $S \in V_\varepsilon$ **THEN**
  füge ein neues Startsymbol S' mit den Regeln $S' \to \varepsilon$ und $S' \to S$ ein.
FI

Wir zeigen nun, dass es zu jeder CFG eine äquivalente ε-freie CFG gibt. Algorithmus 1 skizziert ein Verfahren, das für gegebene CFG $G = (V, \Sigma, \mathcal{P}, S)$ eine äquivalente ε-freie CFG G' erstellt. Für alle Nichtterminale A mit $A \Rightarrow^* \varepsilon$ und für alle Regeln $B \to v$, in denen A auf der rechten Seite enthalten ist, etwa $v = xAy$, wird eine neue Regel $B \to xy$ eingefügt, vorausgesetzt, dass diese Regel noch nicht in der Grammatik enthalten ist und dass x und y nicht beide leer sind. Diese neuen Regeln dienen der Simulation von Herleitungsschritten in G, in denen Regeln zur Herleitung von ε aus A eingesetzt werden. Da in den neuen Regeln $B \to xy$ weitere Nichtterminale, aus denen sich das leere Wort ε herleiten lässt, auf der rechten Seite (also in x oder y) vorkommen können, ist dieser Vorgang solange zu wiederholen bis auf diese Weise keine neuen Regeln mehr eingefügt werden können. Die ursprünglichen ε-Regeln können dann entfernt werden. Falls $\varepsilon \notin \mathcal{L}(G)$, so enthält die resultierende CFG G' keinerlei ε-Regeln und ist zu G äquivalent. Falls ε in der von G erzeugten Sprache liegt, so benötigen wir in G' ein neues Startsymbol S' und fügen die Regel $S' \to \varepsilon$ sowie die Kettenregel $S' \to S$ ein. Die Regel $S' \to S$ dient als erste Regel in Herleitungen aller nicht-leeren Wörter von $\mathcal{L}(G)$, während die Regel $S' \to \varepsilon$ benötigt wird, um das leere Wort herzuleiten.

Im ersten Schritt von Algorithmus 1 wird ein algorithmisches Verfahren benötigt, das als Eingabe eine CFG G hat und dessen Aufgabe darin besteht, die Menge V_ε aller Nichtterminale A mit $A \Rightarrow^* \varepsilon$ zu berechnen. Dies lässt sich durch einen *Markierungsalgorithmus* realisieren, der sukzessive alle Nichtterminale, aus denen das leere Wort herleitbar ist, markiert.

1. Markiere alle Nichtterminale A mit $A \to \varepsilon$.
2. Solange es eine Regel $B \to A_1 \ldots A_n$ gibt, so dass $A_1, \ldots, A_n$ markierte Nichtterminale sind und B nicht markiert ist, wähle eine solche Regel und markiere B.
3. Gib alle markierten Nichtterminale aus.

Der Markierungsalgorithmus kann auch für den Test "gilt $\varepsilon \in \mathcal{L}(G)$?" eingesetzt werden. Hierzu ist lediglich zu prüfen, ob S nach Ausführung des Markierungsalgorithmus markiert ist. Genau in diesem Fall gilt $S \Rightarrow^* \varepsilon$.

Hiermit ist gezeigt, dass jede kontextfreie Sprache zugleich kontextsensitiv ist. Wir erhalten folgenden Satz:

Satz 1.14 (Chomsky Hierarchie). *Sprachen vom Typ* $\mathtt{i}$ *sind zugleich Sprachen vom Typ* $\mathtt{i}-1$ *(*$\mathtt{i} = 1, 2, 3$*).*

2 Reguläre Sprachen

Reguläre Sprachen wurden in Abschnitt 1 mit Hilfe regulärer Grammatiken definiert. In diesem Kapitel stellen wir äquivalente Formalismen für reguläre Sprachen vor (endliche Automaten und reguläre Ausdrücke) und diskutieren einige wichtige Eigenschaften regulärer Sprachen.

2.1 Endliche Automaten

Ein endlicher Automat fungiert als *Sprachakzeptor*, dessen Aufgabe darin besteht, das Wortproblem für eine gegebene Sprache L zu lösen, d.h., für gegebenes Eingabewort w zu entscheiden, ob w in L liegt ("der Automat akzeptiert") oder nicht ("der Automat verwirft"). Im Kontext eines Compilers ist es z.B. die Aufgabe eines Scanners zu entscheiden, ob eine gegebene Zeichenkette w den syntaktischen Regeln eines Identifiers genügt, also ob w zu der formalen Sprache, die durch die syntaktischen Regeln für Identifier gegeben ist, gehört.

Endliche Automaten können als eine sehr eingeschränkte Variante eines Rechnermodells angesehen werden, die mit einem Eingabeband ausgerüstet sind. Die Eingabe eines endlichen Automaten besteht aus einem endlichen Wort, welches zeichenweise auf dem Eingabeband abgelegt ist und auf welches mit Hilfe eines Lesekopfs zugegriffen werden kann. Der Lesekopf kann nur von links nach rechts bewegt werden (man spricht deshalb auch von "one-way automata"), um die Eingabezeichen nacheinander zu lesen. Auf das Eingabeband kann nicht geschrieben werden. Das Schema eines endlichen Automaten ist in Abbildung 5 skizziert. Ein endlicher Automat besteht im Wesentlichen nur aus einer endlichen Kontrolle (den Zuständen und der Übergangsfunktion) und einem Eingabeband. Das einzige zur Verfügung stehende Speichermedium sind die Zustände. Da es nur eine feste (von der Eingabe unabhängige) Anzahl von Zuständen gibt, können endliche Automaten zwar in den Zuständen speichern, ob z.B. bereits das Eingabezeichen a gelesen wurde, nicht aber die Anzahl der bereits gelesenen a's.

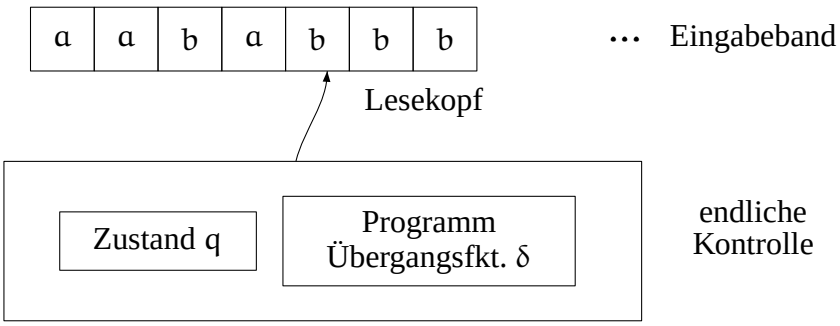


Abbildung 5: Endlicher Automat (Schema)

2.1.1 Deterministische endliche Automaten

Syntaktisch können endliche Automaten als eine besondere Form von gerichteten Graphen angesehen werden, in denen die Kanten mit Symbolen aus einem Alphabet beschriftet sind und

gewisse Zustände als Anfangs- bzw. Endzustände deklariert sind. Wir betrachten zunächst deterministische endliche Automaten, für die wir die Abkürzung DFA verwenden. Diese steht für die englische Bezeichnung “deterministic finite automaton”. In Abschnitt 2.1.2 werden wir die nichtdeterministische Variante untersuchen.

Definition 2.1 (Deterministischer endlicher Automat (DFA)). Ein DFA ist ein Tupel

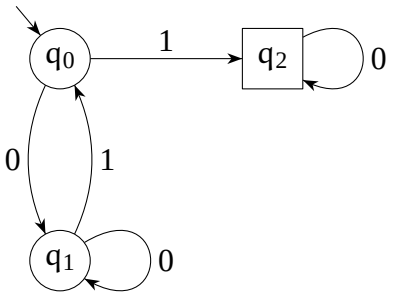
$$\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$$

bestehend aus

- einer endlichen Menge Q von *Zuständen*,
- einem endlichen Alphabet Σ,
- einem *Anfangszustand* $q_0 \in Q$,
- einer Menge $F \subseteq Q$ von *Endzuständen*
- einer partiellen[2] Funktion $\delta : Q \times \Sigma \to Q$.

δ wird auch *Übergangsfunktion* genannt. Der Anfangszustand q_0 wird auch als *Startzustand* oder *initialer Zustand* bezeichnet. Die Zustände in F werden auch *Akzeptanzzustände* oder *finale Zustände* genannt. ■

Für $p = \delta(q, a)$ schreiben wir auch $q \xrightarrow{a} p$ und nennen $q \xrightarrow{a} p$ eine Transition oder einen Übergang. Manchmal sprechen wir auch von einer a-Transition oder einem a-Übergang oder auch einer a-Kante. Für die graphische Darstellung eines DFA verwenden wir Kreise für die Zustände $q \in Q \setminus F$ und Quadrate oder Rechtecke für die Endzustände. Der Anfangszustand ist durch einen kleinen Pfeil markiert. Ein Beispiel ist in Abbildung 6 angegeben.



Zustandsraum: $Q = \{q_0, q_1, q_2\}$
Alphabet: $\Sigma = \{0, 1\}$
Anfangszustand q_0
Endzustandsmenge $F = \{q_2\}$

$\delta(q_0, 0) = q_1$ $\quad \delta(q_0, 1) = q_2$
$\delta(q_1, 0) = q_1$ $\quad \delta(q_1, 1) = q_0$
$\delta(q_2, 0) = q_2$ $\quad \delta(q_2, 1) = \perp$

Abbildung 6: DFA $\mathcal{M}$

Nun zu dem operationellen Verhalten eines endlichen Automaten. Initial steht das Eingabewort auf dem Eingabeband, der Lesekopf zeigt auf das erste Zeichen des Eingabeworts und der Automat befindet sich in seinem Anfangszustand q_0. In jedem Schritt findet in Abhängigkeit des aktuellen Zustands $q \in Q$ und des Zeichens a unter dem Lesekopf auf dem Eingabeband ein

[2] Eine partielle Funktion, ist eine Funktion $f\colon X \to Y$, in der nicht jedes Element des Definitionsbereichs $\mathrm{Def}(f)$ einem Element des Wertebereichs Y zugeordnet ist, das heißt $\mathrm{Def}(f) \subsetneq X$

Wechsel von Zustand q zu Zustand $p = \delta(q, a)$ statt und der Lesekopf wird um eine Position nach rechts verschoben. Ist $\delta(q, a) = \perp$ (undefiniert), so hält die Berechnung des Automaten an. Tritt dieser Fall nicht ein und kann das Eingabewort vollständig "gescannt" werden, dann hält die Berechnung an, nachdem das letzte Zeichen der Eingabe bearbeitet wurde. Eine Berechnung ist entweder *akzeptierend* (wenn das Eingabewort vollstängig gelesen wurde und der Automat einen Endzustand erreicht hat) oder *verwerfend* (sofern kein Endzustand erreicht wurde oder die Eingabe nicht zu Ende gelesen werden konnte).

Wir formalisieren nun das intuitiv erläuterte Akzeptanzverhalten eines DFA. Hierzu führen wir den Begriff eines Laufs für ein Eingabewort in einem DFA ein.

Definition 2.2 (Lauf, akzeptierend, verwerfend). Sei $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ ein DFA und $w = a_1 a_2 \dots a_n \in \Sigma^*$ ein Wort der Länge n. Der *Lauf* für w in $\mathcal{M}$ ist diejenige Zustandsfolge $p_0 p_1 \dots p_m$, so dass

(1) $p_0 = q_0$,

(2) $p_{i+1} = \delta(p_i, a_{i+1})$ für $i = 0, \dots, m-1$, und

(3) entweder $m = n$ oder $m < n$ und $\delta(p_m, a_{m+1}) = \perp$.

Falls $m = n$ und $p_n \in F$, dann wird $p_0 p_1 \dots p_n$ *akzeptierender Lauf* für w genannt. In diesem Fall sagen wir, dass w von $\mathcal{M}$ akzeptiert wird. Andernfalls, also wenn entweder $m < n$ oder $m = n$ und $p_n \notin F$, so wird $p_0 p_1 \dots p_m$ *verwerfender Lauf* für w genannt, und w wird von $\mathcal{M}$ verworfen. ■

Definition 2.3 (Akzeptierte Sprache eines DFA). Sei $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ ein DFA. Die von $\mathcal{M}$ akzeptierte Sprache $\mathcal{L}(\mathcal{M})$ ist die Menge aller Wörter $w \in \Sigma^*$, für die der zugehörige Lauf akzeptierend ist. Wir verwenden naheliegende Sprechweisen wie "$\mathcal{M}$ ist ein DFA für die Sprache L", falls $\mathcal{L}(\mathcal{M}) = L$. ■

Definition 2.4 (Erweiterte Übergangsfunktion eines DFA). Sei $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ ein DFA. Wir erweitern δ zu einer (ebenfalls mit δ bezeichneten) partiellen Abbildung

$$\delta : Q \times \Sigma^* \to Q.$$

Die Definition von $\delta(q, w)$ für $q \in Q$ und $w \in \Sigma^*$ erfolgt durch Induktion nach der Länge von w. Im Induktionsanfang ist das leere Wort zu betrachten:

$$\delta(q, \varepsilon) \stackrel{\text{def}}{=} q$$

Für Wörter $w = a$ der Länge 1 (also Wörtern, die aus einem einzelnen Zeichen in $a \in \Sigma$ bestehen) ist $\delta(q, w)$ bereits definiert. Wir nehmen nun an, dass $w = ax$, wobei $a \in \Sigma$ und x ein Wort der Länge $\geqslant 1$ über Σ ist, so dass $\delta(p, x)$ für alle Zustände p bereits definiert ist. Die Definition von $\delta(q, ax)$ ist dann wie folgt:

$$\delta(q, ax) \stackrel{\text{def}}{=} \begin{cases} \delta(\, \delta(q, a),\, x\,) & : \text{ falls } \delta(q, a) \neq \perp \\ \perp & : \text{ sonst.} \end{cases}$$

Ist $\delta(q, w) = p \in Q$, so schreiben wir auch $q \xrightarrow{w} p$. Für zwei Worte w, z aus Σ^+ gilt offensichtlich $\delta(q, wz) = \delta(\delta(q, w), z)$. ■

Ist also $q_0\,q_1 \ldots q_n \in Q^*$ der Lauf von $\mathcal{M}$ für das Wort $w = a_1\,a_2 \ldots a_n$, so ist $\delta(q_0, w) = q_n$. Kann das Wort $w = a_1\,a_2 \ldots a_n$ nicht vollständig gelesen werden (d.h., der Lauf für w in $\mathcal{M}$ hat die Form $q_0 \ldots q_m$, wobei $m < n$ und $\delta(q_m, a_{m+1}) = \perp$), so ist $\delta(q_0, w) = \perp$. Für die von einem DFA $\mathcal{M}$ akzeptierte Sprache $\mathcal{L}(\mathcal{M})$ gilt also:

$$\begin{aligned} \mathcal{L}(\mathcal{M}) &= \big\{ w \in \Sigma^* : \text{der Lauf für } w \text{ in } \mathcal{M} \text{ ist akzeptierend} \big\} \\ &= \big\{ w \in \Sigma^* : \delta(q_0, w) \in F \big\} \end{aligned}$$

Beispiel 2.5 (Akzeptierte Sprache). Die akzeptierte Sprache des DFA $\mathcal{M}$ aus Abbildung 6 ist $\mathcal{L}(\mathcal{M}) = L^*K$, wobei

$$L = \big\{ 0^n 1 : n \geqslant 1 \big\} \quad \text{und} \quad K = \big\{ 10^n : n \geqslant 0 \big\}.$$

Zunächst machen wir uns klar, dass tatsächlich alle Wörter in L^*K von $\mathcal{M}$ akzeptiert werden, also dass $L^*K \subseteq \mathcal{L}(\mathcal{M})$. Alle Wörter $0^n1 \in L$ haben einen Lauf der Form $q_0\,q_1\,q_1 \ldots q_1\,q_0$. Für die Wörter $x = x_1\,x_2 \ldots x_k \in L^*$ mit $x_1, \ldots, x_k \in L$ gilt daher:

$$q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} \ldots \xrightarrow{x_{k-1}} q_1 \xrightarrow{x_k} q_0$$

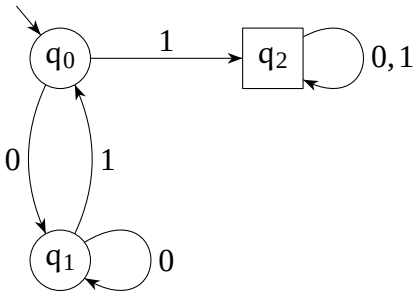
und somit $\delta(q_0, x) = q_0$. Die Läufe der Wörter $y = 10^m \in K$ haben die Form $q_0\,q_2\,q_2 \ldots q_2$. Für alle Wörter $w = xy$ mit $x \in L^*$ und $y \in K$ gilt daher

$$\delta\big(q_0, w\big) = \delta\big(q_0, xy\big) = \delta\big(\delta(q_0, x), y\big) = \delta(q_0, y) = q_2 \in F$$

und somit $w \in \mathcal{L}(\mathcal{M})$. Es bleibt zu zeigen, dass alle von $\mathcal{M}$ akzeptierten Wörter in L^*K liegen, also dass $\mathcal{L}(\mathcal{M}) \subseteq L^*K$. Aufgrund der topologischen Struktur von $\mathcal{M}$ ist klar, dass alle akzeptierten Wörter w die Form $w = xy$ haben müssen, wobei

$$q_0 \xrightarrow{x} q_0 \xrightarrow{y} q_2,$$

und x so gewählt ist, dass $\delta(q_0, x)$ der letzte in $\mathcal{M}$ besuchte Zustand q_0 ist. Dann aber muss x die Form $0^{n_1}10^{n_2}1 \ldots 0^{n_k}1$ haben, wobei $n_1, \ldots, n_k \geqslant 1$ und $k \geqslant 0$ (also in L^* liegen), und y mit einer 1 beginnen, gefolgt von beliebig vielen Nullen (also in K liegen). Es folgt $w \in L^*K$.



Wir betrachten nun den DFA $\mathcal{M}'$ im Bild oben, der sich von dem DFA $\mathcal{M}$ aus Abbildung 6 nur durch eine zusätzliche Schleife für das Terminalzeichen 1 an Zustand q_2 unterscheidet. Die akzeptierte Sprache $\mathcal{L}(\mathcal{M}')$ ist die Menge aller Wörter über dem Alphabet $\{0, 1\}$, die entweder mit einer 1 beginnen oder das Teilwort 011 enthalten. Dies ist die Menge aller Wörter in der Sprache L^*K', wobei L wie oben definiert ist und

$$K' = \big\{ 1z : z \in \{0, 1\}^* \big\}.$$

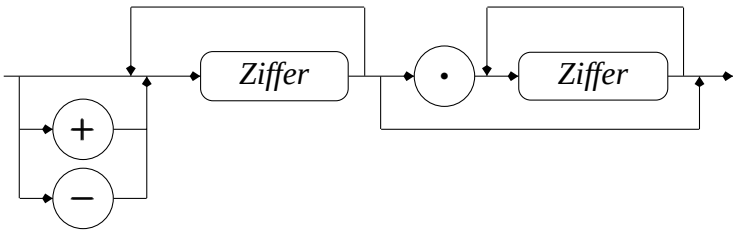
Offenbar haben alle Wörter, die mit einer 1 beginnen, einen akzeptierenden Lauf der Form $q_0\,q_2\ldots q_2$. Betrachten wir nun ein Wort w, das mit einer 0 beginnt und das Teilwort 0 1 1 enthält. Jedes solche Wort w kann zerlegt werden in $w = u011v$, wobei v ein beliebiges Wort über $\{0,1\}$ ist und das Präfix u entweder leer ist oder mit einer 0 beginnt und die Eigenschaft besitzt, dass vor jeder 1 eine 0 steht. Für jedes solche Wort gilt:

$$q_0 \xrightarrow{u} p \xrightarrow{0} q_1 \xrightarrow{1} q_0 \xrightarrow{1} q_2 \xrightarrow{v} q_2$$

Zustand p ist dabei entweder gleich q_1 (falls u mit einer 0 endet) oder gleich q_0 (falls $u = \varepsilon$ oder u mit einer 1 endet). Wir erhalten $\delta(q_0, w) = q_2 \in F$ und somit $w \in \mathcal{L}(\mathcal{M}')$.

Sei nun w ein von $\mathcal{M}'$ akzeptiertes Wort. Da q_0 nicht final ist, ist $w \neq \varepsilon$. Falls w mit einer 1 beginnt, liegt w in K' und somit in $L^* K'$. Falls $w = 0v \in \mathcal{L}(\mathcal{M}')$, dann muss der (akzeptierende) Lauf für w die Gestalt $q_0\,q_1\ldots q_2$ haben. Nach q_1 führen jedoch nur 0-Kanten und die einzige Möglichkeit q_2 von q_1 zu erreichen, sind die beiden 1-Kanten von q_1 nach q_0 und von q_0 nach q_2. Daher muss w das Teilwort 0 1 1 enthalten. ■

Beispiel 2.6 (DFA für Dezimalzahlen). Folgende Abbildung zeigt das Syntaxdiagramm für Dezimalzahlen mit optionalem Vorzeichen $+$ oder $-$, einer Ziffernfolge und einem optionalen Nachkommateil. Zur Vereinfachung schreiben wir kurz *Ziffer* um eine beliebige Ziffer $0,1,\ldots,9$ zu bezeichnen. Das zugrundeliegende Alphabet Σ besteht also aus allen Ziffern, den Vorzeichen $+$ und $-$ sowie einem Punkt $\bullet$, der den Vor- vom Nachkommateil trennt, also $\Sigma = \{+,-,\bullet,0,1,\ldots,9\}$.[3]



Ein DFA über Σ, dessen akzeptierte Sprache mit der Menge aller aus dem Syntaxdiagramm herleitbaren Dezimalzahlen übereinstimmt, ist in Abbildung 7 skizziert.

Wir betrachten das Eingabewort $w = +78\bullet 2$. Die zeichenweise Bearbeitung von w durch $\mathcal{M}$ wird durch folgende Gleichungen anhand der erweiterten Übergangsfunktion formalisiert:

$$\delta(q_0, +78\bullet 2) \;=\; \delta(q_4, 78\bullet 2) \;=\; \delta(q_1, 8\bullet 2) \;=\; \delta(q_1, \bullet 2) \;=\; \delta(q_2, 2) \;=\; q_3$$

Der zu $w = +78\bullet 2$ gehörende Lauf ist also die Zustandsfolge $q_0\,q_4\,q_1\,q_1\,q_2\,q_3$. Da q_3 ein Endzustand ist, wird das Wort $+78\bullet 2$ akzeptiert. Entsprechend ist

$$\delta(q_0, 78+9) \;=\; \delta(q_1, 8+9) \;=\; \delta(q_1, +9) \;=\; \perp$$

für den DFA aus Abbildung 7. Das Wort $78+9$ wird also verworfen, da das letzte Zeichen (die Ziffer 9) nicht gelesen werden kann. Der zugehörige Lauf ist $q_0\,q_1\,q_1$. Dieser ist verwerfend. ■

[3] Alternativ kann auch *Ziffer* als Symbol des Alphabets angesehen werden, in welchem Fall Σ aus den drei Symbolen $+$, $-$ und $\bullet$ und dem Symbol *Ziffer* besteht. In den unten angegebenen Wörtern sind dann die Dezimalziffern 7, 8 und 2 durch das Symbol *Ziffer* zu ersetzen. Diese Variante wurde in den Folien zur Vorlesung eingesetzt.

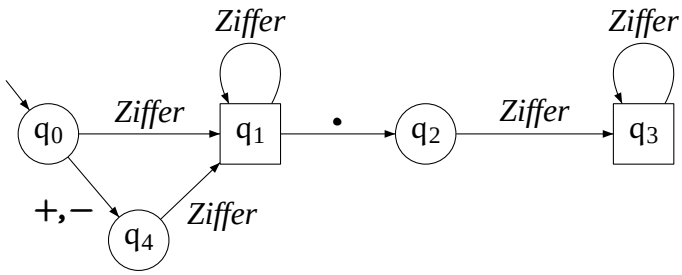


Abbildung 7: DFA für Dezimalzahlen

Totale Übergangsfunktion. Die Übergangsfunktion δ eines DFA wurde als *partielle* Funktion $\delta : Q \times \Sigma \to Q$ definiert. Undefiniertheitsstellen sind zunächst zugelassen, jedoch kann jeder DFA in einen totalen DFA $\mathcal{M}_{\text{tot}}$ überführt werden, der dasselbe Akzeptanzverhalten wie $\mathcal{M}$ hat. Unter einem totalen DFA verstehen wir einen DFA mit totaler Übergangsfunktion (d.h., ohne Undefiniertheitsstellen). Hierzu müssen wir lediglich $\mathcal{M}$ um einen neuen Zustand p erweitern und für alle "fehlenden" Übergänge eine entsprechende Transition zu Zustand p einfügen. Die präzise Definition des DFA $\mathcal{M}_{\text{tot}}$ ist wie folgt. Ist $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ ein DFA, dessen Übergangsfunktion mindestens eine Undefiniertheitsstelle hat, so ist der zugehörige totale DFA $\mathcal{M}_{\text{tot}}$ wie folgt definiert:

$$\mathcal{M}_{\text{tot}} \stackrel{\text{def}}{=} (Q \cup \{p\}, \Sigma, \delta_{\text{tot}}, q_0, F),$$

wobei p ein neuer Zustand ist (also nicht in Q liegt) und die Übergangsfunktion δ_{tot} wie folgt definiert ist:

$$\delta_{\text{tot}}(q, a) \stackrel{\text{def}}{=} \begin{cases} \delta(q, a) & : \text{ falls } q \in Q \text{ und } \delta(q, a) \neq \bot \\ p & : \text{ sonst} \end{cases}$$

Dabei ist $q \in Q \cup \{p\}$ und $a \in \Sigma$. Insbesondere enthält $\mathcal{M}_{\text{tot}}$ die Schleifen $p \xrightarrow{a} p$ für alle $a \in \Sigma$. D.h., sobald Zustand p betreten wird, kann dieser niemals mehr verlassen werden. Man spricht daher auch von einem *Fangzustand*. Da der neue Fangzustand p kein Akzeptanzzustand ist, sind alle Läufe, die in p enden, verwerfend. Offenbar ist $\mathcal{M}_{\text{tot}}$ total, d.h., die Übergangsfunktion δ_{tot} ist total. Ferner gilt $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}_{\text{tot}})$, da für jedes Wort $w \in \mathcal{L}(\mathcal{M})$ der Lauf für w in $\mathcal{M}$ akzeptierend und zugleich ein akzeptierender Lauf für w in $\mathcal{M}_{\text{tot}}$ ist. Für jedes Wort $w = a_1 a_2 \ldots a_n \in \Sigma^* \setminus \mathcal{L}(\mathcal{M})$ gilt:

- falls der Lauf für w in $\mathcal{M}$ die Form $q_0\, q_1 \ldots q_m$ für ein $m < n$ hat, so gilt $\delta(q_m, a_{m+1}) = \bot$ und somit $\delta_{\text{tot}}(q_m, a_{m+1}) = p$. In diesem Fall hat der Lauf für w in $\mathcal{M}_{\text{tot}}$ die Form $q_0\, q_1 \ldots q_m\, p\, p \ldots p$ und ist ebenfalls verwerfend. Also gilt $w \notin \mathcal{L}(\mathcal{M}_{\text{tot}})$.
- falls der Lauf für w in $\mathcal{M}$ vollständig ist (d.h., die Form $q_0\, q_1 \ldots q_n$ mit $q_n \notin F$ hat), so ist $q_0\, q_1 \ldots q_n$ zugleich der Lauf von w in $\mathcal{M}_{\text{tot}}$. Wegen $q_n \notin F$ ist dieser verwerfend. Es gilt also ebenfalls $w \notin \mathcal{L}(\mathcal{M}_{\text{tot}})$.

Aufgrund dieser Beobachtung ist es keine Einschränkung anzunehmen, dass ein vorliegender DFA total ist, was aus technischen Gründen oftmals hilfreich ist.

Der DFA $\mathcal{M}$ aus Abbildung 7 hat zunächst eine partielle Übergangsfunktion (z.B. ist $\delta(q_4, +) = \bot$). Er kann durch die Hinzunahme eines Fangzustands p zu einem totalen DFA $\mathcal{M}_{\text{tot}}$ modifiziert werden, siehe Abbildung 8. Beispielsweise gilt $\delta_{\text{tot}}(q_0, 78 + 9) = p$. Der Lauf für das Wort $78 + 9$ in $\mathcal{M}_{\text{tot}}$ ist $q_0\, q_1\, q_1\, p\, p$. Da p kein Endzustand ist, ist dieser Lauf verwerfend.

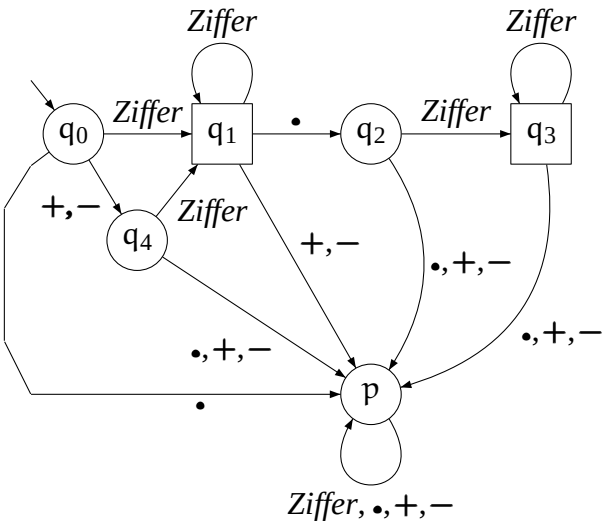


Abbildung 8: Totaler DFA für Dezimalzahlen

Endliche Automaten und reguläre Grammatiken (1. Teil)

Wir werden sehen, dass DFA und reguläre Grammatiken dieselbe Ausdrucksstärke haben. Im Beweis des folgenden Lemmas geben wir ein Verfahren an, wie man zu gegebenem DFA eine reguläre Grammatik konstruieren kann. Dieses belegt, dass DFA höchstens so ausdrucksstark wie reguläre Grammatiken sind.

Lemma 2.7 (DFA $\rightsquigarrow$ reguläre Grammatik). *Zu jedem DFA $\mathcal{M}$ gibt es eine reguläre Grammatik* G *mit* $\mathcal{L}(\mathcal{M}) = \mathcal{L}(G)$.

Beweis. Sei $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ ein DFA mit einer totalen Übergangsfunktion. Wir fassen die Zustände als Nichtterminale auf. Die Übergangsfunktion entspricht den Regeln einer regulären Grammatik. Jeden Übergang $q \xrightarrow{a} p$ in $\mathcal{M}$ mit $a \in \Sigma$ fassen wir als Regel $q \to ap$ auf. Ist p ein Endzustand, so fügen wir zusätzlich eine Terminalregel $q \to a$ ein. Zusätzlich ist der Sonderfall zu berücksichtigen, dass das leere Wort von $\mathcal{M}$ akzeptiert wird. Dies ist genau dann der Fall, wenn der Anfangszustand q_0 ein Endzustand ist. In diesem Fall fügen wir zusätzlich die Regel $q_0 \to \varepsilon$ ein. Die formale Definition der regulären Grammatik $G = (V, \Sigma, \mathcal{P}, S)$ ist wie folgt. Die Variablenmenge ist $V = Q$. Das Startsymbol ist $S = q_0$. Die Regeln von G sind durch folgende drei Bedingungen gegeben:

- Ist $\delta(q, a) = p$, dann ist $q \to ap$ eine Regel in G.
- Ist $\delta(q, a) = p \in F$, dann ist $q \to a$ eine Regel in G.
- Ist $q_0 \in F$, dann enthält G die ε-Regel $q_0 \to \varepsilon$.
- Es gibt keine weitere Regel als die oben genannten.

Offenbar ist G regulär. Es bleibt zu zeigen, dass $\mathcal{L}(G) = \mathcal{L}(\mathcal{M})$. Zunächst stellen wir fest, dass für das leere Wort gilt:

	$\varepsilon \in \mathcal{L}(\mathcal{M})$	
gdw	$q_0 \in F$	(gilt in jedem DFA)
gdw	$q_0 \to \varepsilon$ ist eine Regel in G	(Definition von G)
gdw	$\varepsilon \in \mathcal{L}(G)$	(da G keine weiteren ε-Regeln hat)

Im Folgenden sei $x = a_1 a_2 \dots a_n \in \Sigma^*$ ein nicht-leeres Wort, also $n \geqslant 1$. Zu zeigen ist, dass x genau dann von $\mathcal{M}$ akzeptiert wird, wenn x in G herleitbar ist.

"$\Longrightarrow$": Wir nehmen an, dass $x \in \mathcal{L}(\mathcal{M})$. Dann ist der zugehörende Lauf $q_0 q_1 \dots q_n$ in $\mathcal{M}$ akzeptierend. Daher ist $q_n \in F$ und $q_{i+1} = \delta(q_i, a_{i+1})$ für $0 \leqslant i < n$. Grammatik G enthält also die Regeln $q_i \to a_{i+1} q_{i+1}$ für $0 \leqslant i < n-1$ sowie die Regel $q_{n-1} \to a_n$. Also ist

$$\begin{array}{ccccccccccc} q_0 & \Rightarrow & a_1 q_1 & \Rightarrow & a_1 a_2 q_2 & \Rightarrow & \dots & \Rightarrow & a_1 a_2 \dots a_{n-1} q_{n-1} & \Rightarrow & a_1 a_2 \dots a_{n-1} a_n \\ \uparrow & & \uparrow & & & & & & \uparrow & & \\ \text{Regel} & & \text{Regel} & & & & & & \text{Regel} & & \\ q_0 \to a_1 q_1 & & q_1 \to a_2 q_2 & & & & & & q_{n-1} \to a_n & & \end{array}$$

eine Ableitung von x in G. Also gilt $x \in \mathcal{L}(G)$.

"$\Longleftarrow$": Wir setzen nun voraus, dass $x \in \mathcal{L}(G)$. Da G regulär ist, gibt es nur zwei Arten von potentiellen Herleitungen von x in G:

(1) $q_0 \Rightarrow a_1 q_1 \Rightarrow a_1 a_2 q_2 \Rightarrow \dots \Rightarrow a_1 a_2 \dots a_{n-1} q_{n-1} \Rightarrow a_1 a_2 \dots a_{n-1} a_n q_n \Rightarrow a_1 a_2 \dots a_{n-1} a_n$

(2) $q_0 \Rightarrow a_1 q_1 \Rightarrow a_1 a_2 q_2 \Rightarrow \dots \Rightarrow a_1 a_2 \dots a_{n-1} q_{n-1} \Rightarrow a_1 a_2 \dots a_{n-1} a_n$

In beiden Möglichkeiten wird in den ersten $n-1$ Herleitungsschritten eine Regel der Form $q_{i-1} \to a_i q_i$ eingesetzt und somit das Präfix $a_1 a_2 \dots a_{n-1}$ von x generiert. Aufgrund der Definition von G stimmt die Zustandsfolge $q_0 q_1 \dots q_{n-1}$ mit dem Lauf für $a_1 a_2 \dots a_{n-1}$ in $\mathcal{M}$ überein.

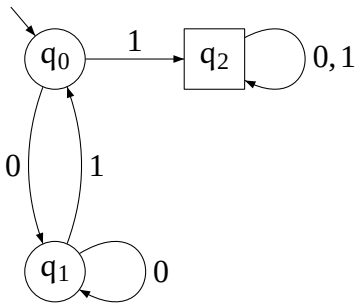
- In (1) wird die Regel $q_{n-1} \to a_n q_n$ im vorletzten Schritt eingesetzt, gefolgt von der ε-Regel $q_n \to \varepsilon$. Dies ist jedoch nur dann möglich, wenn $q_0 = q_n = \delta(q_{n-1}, a_n) \in F$.
- In (2) wird im n-ten Herleitungsschritt die Regel $q_{n-1} \to a_n$ eingesetzt. Dann aber ist $q_n \stackrel{\text{def}}{=} \delta(q_{n-1}, a_n)$ ein Endzustand.

In beiden Fällen ist $q_0 q_1 \dots q_{n-1} q_n$ der Lauf für x in $\mathcal{M}$. Dieser ist akzeptierend, da q_n ein Endzustand ist. Also gilt $x \in \mathcal{L}(\mathcal{M})$. □

Wir demonstrieren die im Beweis von Lemma 2.7 (Seite 24) angegebene Transformation am Beispiel des DFA in Abbildung 9. Die konstruierte reguläre Grammatik ist durch die Regeln

$$S \to 0B \mid 1A \mid 1 \qquad A \to 0A \mid 1A \mid 0 \mid 1 \qquad B \to 1S \mid 0B$$

gegeben. Dabei identifizieren wir S mit q_0, A mit q_2 und B mit q_1.



reguläre Grammatik mit den
Variablen $S = q_0$, $A = q_2$ und $B = q_1$

Transition $q_0 \xrightarrow{0} q_1$	$\widehat{=}$	Regel $S \to 0B$
Transition $q_0 \xrightarrow{1} q_2$	$\widehat{=}$	Regel $S \to 1A$
		Regel $S \to 1$
Transition $q_1 \xrightarrow{0} q_1$	$\widehat{=}$	Regel $B \to 0B$
Transition $q_1 \xrightarrow{1} q_0$	$\widehat{=}$	Regel $B \to 1S$
Transition $q_2 \xrightarrow{a} q_2$	$\widehat{=}$	Regel $A \to aA$
für $a \in \{0,1\}$		Regel $A \to a$

Abbildung 9: DFA $\mathcal{M} \rightsquigarrow$ reguläre Grammatik

2.1.2 Nichtdeterministische endliche Automaten

Wir betrachten nun die nichtdeterministische Variante von endlichen Automaten. Diese haben eine *Menge* von Anfangszuständen, unter denen eine nichtdeterministische Auswahl stattfindet. Analog ordnet die Übergangsfunktion jedem Paar $(q, a) \in Q \times \Sigma$ eine *Menge* von möglichen Folgezuständen zu. Nichtdeterministische endliche Automaten unterliegen der fiktiven Annahme, dass ein Orakel zur Verfügung steht, welches den Nichtdeterminismus – also die Wahl eines Anfangszustands und die Wahl der jeweiligen Folgezustände – so aufzulösen vermag, dass ein akzeptierender Lauf entsteht, sofern ein solcher existiert.

Definition 2.8 (Nichtdeterministischer endlicher Automat (NFA)). Ein NFA ist ein Tupel $\mathcal{M} = (Q, \Sigma, \delta, Q_0, F)$ bestehend aus einer endlichen Menge Q von Zuständen, einem endlichen Alphabet Σ, einer Menge $F \subseteq Q$ von Endzuständen und

- einer totalen *Übergangsfunktion* $\delta : Q \times \Sigma \to 2^Q$,
- einer Menge $Q_0 \in 2^Q$ von *Anfangszuständen*,

wobei 2^Q die Potenzmenge von Q bezeichnet. ■

Wie für DFA schreiben wir manchmal $q \xrightarrow{a} p$ für $p \in \delta(q, a)$. In Anlehnung an die visuelle Darstellung von NFA als gerichteten Graphen (eventuell mit parallelen Kanten) kann man die Übergangsfunktion eines NFA auch als Relation $\delta \subseteq Q \times \Sigma \times Q$ auffassen.

Definition 2.9 (Erweiterte Übergangsfunktion eines NFA). Sei $\mathcal{M} = (Q, \Sigma, \delta, Q_0, F)$ ein NFA. In Analogie zu Definition 2.4 (Seite 20) erweitern wir die Übergangsfunktion eines NFA zu einer (ebenfalls mit δ bezeichneten) Abbildung

$$\delta : 2^Q \times \Sigma^* \to 2^Q.$$

Intuitiv ist $\delta(P, x)$ die Menge aller Zustände, die man mit dem Wort x von einem Zustand $p \in P$ erreichen kann. Die formale Definition von $\delta(P, x)$ erfolgt durch Induktion nach der Länge von

x. Sei $P \subseteq Q$ und $a \in \Sigma$, $x \in \Sigma^+$. Dann ist

$$\delta(P,\varepsilon) \stackrel{\text{def}}{=} P, \;\; \delta(P,a) \stackrel{\text{def}}{=} \bigcup_{p\in P} \delta\big(p,a\big) \;\text{ und }\; \delta(P,ax) \stackrel{\text{def}}{=} \bigcup_{p\in P} \delta\big(\delta(p,a),x\big).$$

Ist $q \in Q$ und $w \in \Sigma^*$, so ist $\delta(q,w) \stackrel{\text{def}}{=} \delta(\{q\},w)$. Wie bei DFA gilt für zwei Worte w,z aus Σ^+ offensichtlich $\delta(q,wz) = \delta(\delta(q,w),z)$.

■

Definition 2.10 (Akzeptierte Sprache eines NFA). Sei $\mathcal{M} = (Q,\Sigma,\delta,Q_0,F)$ ein NFA. Die von $\mathcal{M}$ *akzeptierte Sprache* $\mathcal{L}(\mathcal{M})$ ist wie folgt definiert:

$$\mathcal{L}(\mathcal{M}) \stackrel{\text{def}}{=} \big\{ w \in \Sigma^* : \delta(Q_0,w) \cap F \neq \varnothing \big\}.$$ ■

Beispiel 2.11 (NFA). Wir betrachten den NFA $\mathcal{M}$ mit dem Alphabet $\Sigma = \{0,1\}$ in Abbildung 10. Dieser hat zwei Startzustände q_0 und q_1 zwischen denen eine nichtdeterministische Wahl stattfindet. Da $\delta(q_0,0) = \{q_0,q_1\}$ zweielementig ist, verhält sich $\mathcal{M}$ im Startzustand q_0 für das Eingabezeichen 0 nichtdeterministisch und entscheidet sich willkürlich für einen der beiden Folgezustände q_0 oder q_1.

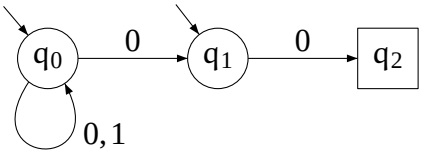


Abbildung 10: NFA $\mathcal{M}$

Da $\delta(q_0,1) = \{q_0\}$ einelementig ist, ist das Verhalten von $\mathcal{M}$ in Zustand q_0 bei Lesen einer 1 deterministisch. Weiter ist $\delta(q_1,1) = \varnothing$. Liest $\mathcal{M}$ also in Zustand q_1 das Eingabezeichen 1, so gibt es keinen entsprechenden Übergang und $\mathcal{M}$ verwirft. Analoges gilt für den Endzustand q_2, in dem weder eine 0 noch eine 1 gelesen werden kann, da $\delta(q_2,0) = \delta(q_2,1) = \varnothing$.

Für die erweiterte Übergangsfunktion gilt:

$$\begin{array}{rcllcl}
\delta(q_0,0100) &=& \delta(\delta(q_0,0),100) & = & \delta(\{q_0,q_1\},100) & \\
&=& \delta(q_0,100) \cup \delta(q_1,100) & = & \delta(\delta(q_0,1),00) & \\
&=& \delta(q_0,00) & = & \delta(\delta(q_0,0),0) & \\
&=& \delta(\{q_0,q_1\},0) & = & \delta(q_0,0) \cup \delta(q_1,0) & = \;\; \{q_0,q_1,q_2\}
\end{array}$$

und somit $\delta(Q_0,0100) = \delta(\{q_0,q_1\},0100) = \{q_0,q_1,q_2\}$. Wegen $\delta(Q_0,0100) \cap F = \{q_2\} \neq \varnothing$ gilt $0100 \in \mathcal{L}(\mathcal{M})$. ■

In Analogie zur Definition der Läufe in einem DFA (Seite 20) können wir $\mathcal{L}(\mathcal{M})$ über die Läufe charakterisieren. Sei $w = a_1 a_2 \ldots a_n \in \Sigma^*$. Ein Lauf für w in $\mathcal{M}$ ist eine Zustandsfolge $q_0\, q_1 \ldots q_m$ mit

$$q_0 \in Q_0 \;\text{ und }\; q_{i+1} \in \delta(q_i, a_{i+1}),\; i = 0,\ldots,m-1.$$

Weiter fordern wir, dass entweder $m = n$ oder $m < n$ und $\delta(q_m, a_{m+1}) = \varnothing$. Ist $m = n$ und $q_n \in F$, dann nennen wir $q_0\, q_1 \ldots q_n$ einen akzeptierenden Lauf für w in $\mathcal{M}$. Ist $m < n$ und $\delta(q_m, a_{m+1}) = \varnothing$ oder $m = n$ und $q_n \notin F$, dann wird $q_0\, q_1 \ldots q_m$ verwerfender Lauf für w in $\mathcal{M}$ genannt. Die von $\mathcal{M}$ akzeptierte Sprache ist also genau die Menge aller Wörter $w \in \Sigma^*$, für die es wenigstens einen akzeptierenden Lauf gibt:

$$\begin{aligned} \mathcal{L}(\mathcal{M}) &= \big\{ w \in \Sigma^* : \text{es gibt einen akzeptierenden Lauf für } w \text{ in } \mathcal{M} \big\} \\ &= \big\{ w \in \Sigma^* : \delta(Q_0, w) \cap F \neq \varnothing \big\} \end{aligned}$$

Im Gegensatz zu DFA kann ein Wort w viele Läufe in einem NFA haben. Für die Akzeptanz wird lediglich gefordert, dass einer der Läufe für w akzeptierend ist.

Beispiel 2.12 (Läufe und akzeptierte Sprache eines NFA). Wir betrachten nochmals den NFA aus Abbildung 10 auf Seite 27. Das Wort $w = 0100$ hat mehrere Läufe, nämlich:

$q_0\, q_0\, q_0\, q_0\, q_0$	ist nicht akzeptierend
$q_0\, q_0\, q_0\, q_0\, q_1$	ist nicht akzeptierend
$q_0\, q_0\, q_0\, q_1\, q_2$	ist akzeptierend
$q_0\, q_1$	ist nicht akzeptierend
$q_1\, q_2$	ist nicht akzeptierend

Da nur die Existenz eines akzeptierenden Laufs entscheidend ist, gilt $w = 0100 \in \mathcal{L}(\mathcal{M})$, obwohl es für w auch verwerfende Läufe gibt. Tatsächlich akzeptiert $\mathcal{M}$ genau diejenigen Wörter, die entweder 0 sind oder mit 00 enden:

$$\mathcal{L}(\mathcal{M}) \;=\; \big\{0\big\} \cup \big\{ x00 : x \in \{0,1\}^* \big\}.$$

Für den Nachweis der Inklusion "$\supseteq$" genügt es für jedes der Wörter 0 oder $x00$ mit $x \in \{0,1\}^*$ einen akzeptierenden Lauf in $\mathcal{M}$ anzugeben. Für das Wort 0 ist es der Lauf $q_1\, q_2$. Für die Wörter $x00$ mit $|x| = k$ ist

$$\underbrace{q_0\, q_0 \ldots q_0}_{\text{k+1-mal}}\, q_1\, q_2$$

ein akzeptierender Lauf, in dem $\mathcal{M}$ bei Scannen des Präfix x in Zustand q_0 verharrt und mit den letzten beiden Nullen von q_0 über q_1 in den Endzustand q_2 geht.

Die Inklusion "$\subseteq$" folgt aus der Beobachtung, dass erstens (1) nur das Wort 0 einen in q_1 beginnenden akzeptierenden Lauf besitzt und zweitens (2) alle Wörter, die einen in q_0 beginnenden akzeptierenden Lauf haben, mit 00 enden. Aussage (1) ist offensichtlich. Aussage (2) ergibt sich aus der Tatsache, dass alle Läufe, die in q_0 beginnen und in q_2 enden, die Gestalt $q_0\, q_0 \ldots q_0\, q_1\, q_2$ haben. Da Übergänge von q_0 nach q_1 und von q_1 nach q_2 nur durch Lesen des Zeichens 0 möglich sind, muss das betreffende Eingabewort die Form $x00$ haben. ■

Beispiel 2.13 (NFA für Mustererkennung). Wir betrachten das Problem der Mustererkennung, bei dem ein Muster $M = a_1 \ldots a_m$ und ein Text $T = b_1 \ldots b_t$ gegeben sind und gefragt ist, ob M in T vorkommt. Die Idee für einen nichtdeterministischen Mustererkennungsalgorithmus besteht darin, nichtdeterministisch eine Textposition i zu erraten, und dann zeichenweise zu prüfen, ob das Muster im Text ab Position i beginnt.

Für festes Muster kann dieses sehr simple Verfahren durch den in Abbildung 11 skizzierten NFA $\mathcal{M}$ realisiert werden. Wird ein Text $T = b_1 b_2 \ldots b_t$ über diesem NFA "gescannt" und stimmt das

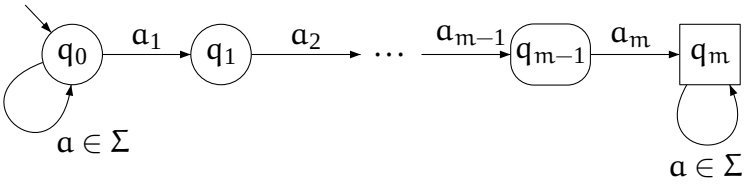


Abbildung 11: NFA für das Mustererkennungsproblem

aktuelle Eingabezeichen b_i mit dem ersten Zeichen a_1 des Musters überein, dann rät der NFA nichtdeterministisch, ob das Muster ab Position i beginnt. Wenn ja, wechselt der NFA von dem Anfangszustand q_0 in den Zustand q_1 und prüft nun deterministisch, ob die restlichen Zeichen $a_2 \dots a_m$ des Musters mit den folgenden Textzeichen $b_{i+1}\, b_{i+2} \dots b_{i+m-1}$ übereinstimmen. Die von $\mathcal{M}$ akzeptierte Sprache ist also die Menge aller Texte $T \in \Sigma^*$, so dass M ein Teilwort von T ist. ■

Äquivalenz von DFA und NFA

Jeder DFA kann als NFA aufgefasst werden. Dies ist wie folgt einsichtig. Ist $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ ein DFA, dann definieren wir einen NFA wie folgt. Sei $\mathcal{M}' = (Q, \Sigma, \delta', \{q_0\}, F)$, wobei

$$\delta'(q,a) \;=\; \begin{cases} \{\,\delta(q,a)\,\} & : \text{ falls } \delta(q,a) \neq \perp \\ \varnothing & : \text{ sonst.} \end{cases}$$

Offenbar ist jeder Lauf von $\mathcal{M}'$ für w zugleich ein Lauf von $\mathcal{M}$ in w und umgekehrt. Somit gilt $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$. Wir zeigen nun, dass es auch umgekehrt zu jedem NFA einen DFA gibt, welcher dieselbe Sprache akzeptiert.

Definition 2.14 (Äquivalenz von NFA). Seien $\mathcal{M}_1$ und $\mathcal{M}_2$ zwei NFA mit demselben Alphabet Σ. $\mathcal{M}_1$ und $\mathcal{M}_2$ heißen *äquivalent*, falls $\mathcal{L}(\mathcal{M}_1) = \mathcal{L}(\mathcal{M}_2)$.[4] ■

Satz 2.15 (NFA $\rightsquigarrow$ DFA, Potenzmengenkonstruktion). *Zu jedem NFA gibt es einen äquivalenten DFA.*

Beweis. Sei $\mathcal{M} = (Q, \Sigma, \delta, Q_0, F)$ ein NFA. Wir wenden die sog. *Potenzmengenkonstruktion* an und definieren einen DFA, dessen Zustände Mengen von Zuständen in $\mathcal{M}$ sind.

$$\mathcal{M}_{\text{det}} \;\stackrel{\text{def}}{=}\; \left(2^Q, \Sigma, \delta_{\text{det}}, Q_0, F_{\text{det}}\right),$$

wobei

$$F_{\text{det}} \;\stackrel{\text{def}}{=}\; \{\, P \subseteq Q : P \cap F \neq \varnothing \,\}$$

$$\delta_{\text{det}}(P,a) \;\stackrel{\text{def}}{=}\; \bigcup_{p \in P} \delta(p,a).$$

Insbesondere ist $\delta_{\text{det}}(\varnothing, a) = \varnothing$ für alle $a \in \Sigma$. Man beachte, dass die Übergangsfunktion δ_{det} von $\mathcal{M}_{\text{det}}$ total ist. Wir zeigen nun, dass $\mathcal{L}(\mathcal{M}) \;=\; \mathcal{L}(\mathcal{M}_{\text{det}})$.

[4] Wir fassen DFA als NFA auf, so dass der Äquivalenzbegriff zugleich für DFA definiert ist.

Die einfachste Argumentation benutzt die erweiterten Übergangsfunktionen in $\mathcal{M}$ und $\mathcal{M}_{\text{det}}$. Tatsächlich stimmt die deterministische Übergangsfunktion $\delta_{\text{det}} : 2^Q \times \Sigma^* \to 2^Q$ von $\mathcal{M}_{\text{det}}$ mit der in Definition 2.9 angegebenen Erweiterung der nichtdeterministischen Übergangsfunktion δ von $\mathcal{M}$ überein. Der Nachweis dieser Aussage wird erbracht, indem man durch Induktion nach der Wortlänge von w zeigt, dass $\delta_{\text{det}}(P, w) = \delta(P, w)$. Es gilt $\delta_{\text{det}}(P, \varepsilon) = \delta(P, \varepsilon)$ und $\delta_{\text{det}}(P, a) = \delta(P, a)$ (Definition von δ_{det}). Weiter ist für $a \in \Sigma$ und $w \in \Sigma^+$

$$\begin{aligned}
\delta_{\text{det}}(P, aw) &= \delta_{\text{det}}(\delta_{\text{det}}(P, a), w) \\
&= \delta_{\text{det}}(\delta(P, a), w) \\
&= \delta(\delta(P, a), w) \\
&= \delta(P, aw)
\end{aligned}$$

Nun folgt:

$$\begin{aligned}
w \in \mathcal{L}(\mathcal{M}_{\text{det}}) \quad & \text{gdw} \quad \delta_{\text{det}}(Q_0, w) \in F_{\text{det}} \\
& \text{gdw} \quad \delta(Q_0, w) \in F_{\text{det}} \\
& \text{gdw} \quad \delta(Q_0, w) \cap F \neq \varnothing \\
& \text{gdw} \quad w \in \mathcal{L}(\mathcal{M})
\end{aligned}$$

Wir geben nun einen expliziten Nachweis der Gleichung $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}_{\text{det}})$ an, der mit akzeptierenden Läufen in $\mathcal{M}$ bzw. $\mathcal{M}_{\text{det}}$ argumentiert und daher zugleich verdeutlicht, inwiefern $\mathcal{M}_{\text{det}}$ die möglichen Berechnungen von $\mathcal{M}$ simuliert.

"$\subseteq$": Sei $w = a_1 a_2 \dots a_n \in \mathcal{L}(\mathcal{M})$ und $q_0 q_1 \dots q_n$ ein akzeptierender Lauf von $\mathcal{M}$ für w. Dann ist $q_0 \in Q_0$, $q_{i+1} \in \delta(q_i, a_{i+1})$ für $i = 0, 1, \dots, n-1$, und $q_n \in F$. Sei $P_0 = Q_0$ und für $i = 1, \dots, n$ sei $P_i = \delta(P_{i-1}, a_i)$ und damit $P_i = \delta_{\text{det}}(P_{i-1}, a_i)$. Durch Induktion nach i kann man zeigen, dass

$$q_i \in P_i \text{ für } i = 0, 1, \dots, n.$$

Somit ist $P_0 P_1 \dots P_n$ der zu w gehörende Lauf in $\mathcal{M}_{\text{det}}$. Insbesondere ist $q_n \in P_n \cap F$ und daher $P_n \cap F \neq \varnothing$. Hieraus folgt $P_n \in F_{\text{det}}$. Also ist $P_0 P_1 \dots P_n$ ein akzeptierender Lauf. Somit ist $w \in \mathcal{L}(\mathcal{M}_{\text{det}})$.

"$\supseteq$": Sei $w \in \mathcal{L}(\mathcal{M}_{\text{det}})$ und $w = a_1 \dots a_n$. Weiter sei $P_0 P_1 \dots P_n$ der zu w gehörende Lauf in $\mathcal{M}_{\text{det}}$. Dann ist $P_0 = Q_0$, $P_{i+1} = \delta_{\text{det}}(P_i, a_{i+1})$, $i = 0, 1, \dots, n-1$, und P_n ein Endzustand in $\mathcal{M}_{\text{det}}$, also $P_n \cap F \neq \varnothing$. Wir konstruieren nun "rückwärts" einen akzeptierenden Lauf $q_0 q_1 \dots q_n$ für w in $\mathcal{M}$:

Wir wählen einen beliebigen Zustand $q_n \in P_n \cap F$.

Wähle $q_{n-1} \in P_{n-1}$ mit $q_n \in \delta(q_{n-1}, a_n)$.

Wähle $q_{n-2} \in P_{n-2}$ mit $q_{n-1} \in \delta(q_{n-2}, a_{n-1})$.

$\vdots$

Wähle $q_0 \in P_0$ mit $q_1 \in \delta(q_0, a_1)$.

Tatsächlich gibt es solche Zustände q_i, da $P_n \cap F \neq \varnothing$ und für $i = n-1, \ldots, 0$:

$$P_{i+1} \;=\; \delta_{\text{det}}(P_i, a_{i+1}) \;=\; \bigcup_{q \in P_i} \delta(q, a_{i+1}).$$

Daher gibt es zu jedem Zustand $q_{i+1} \in P_{i+1}$ einen Zustand $q_i \in P_i$ mit $q_{i+1} \in \delta(q_i, a_{i+1})$. Wegen $P_0 = Q_0$ ist q_0 ein Anfangszustand und $q_0\, q_1 \ldots q_n$ ein Lauf von $\mathcal{M}$ für w. Wegen $q_n \in F$ ist der Lauf $q_0\, q_1 \ldots q_n$ akzeptierend. Also ist $w \in \mathcal{L}(\mathcal{M})$. □

In Abbildung 12 betrachten wir ein einfaches Beispiel für die im Beweis von Satz 2.15 angegebene Potenzmengenkonstruktion. Links ist der NFA $\mathcal{M}$ skizziert, rechts dessen Potenzmengenkonstruktion. Beispielsweise kann der akzeptierende Lauf $q_0\, q_1\, q_0\, q_1\, q_0$ für das Wort 1010 in $\mathcal{M}$ durch den akzeptierenden Lauf $\{q_0\}\{q_0, q_1\}\{q_0\}\{q_0, q_1\}\{q_0\}$ im DFA $\mathcal{M}_{\text{det}}$ (Potenzmengenkonstruktion) simuliert werden. Zustand $\{q_1\}$ in der Potenzmengenkonstruktion ist unerreichbar. Er kann daher weggelassen werden, ohne die akzeptierte Sprache zu ändern. Auch auf den Zustand $\varnothing$ kann verzichtet werden, da von ihm kein Endzustand erreichbar ist.

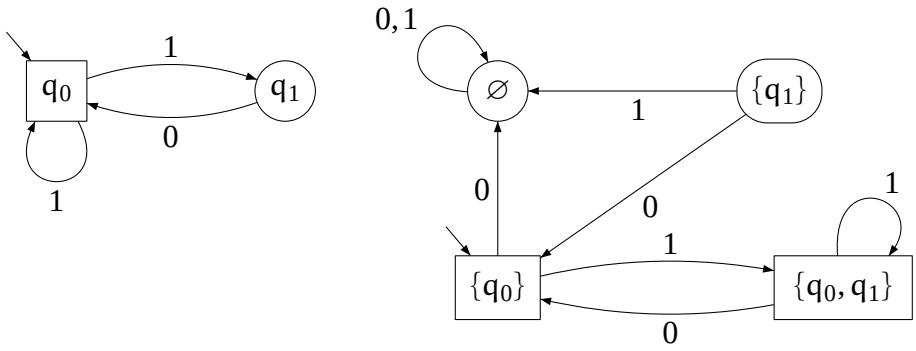


Abbildung 12: Beispiel zur Potenzmengenkonstruktion

Beispiel 2.16 (Das Teilsummenproblem). Das Teilsummenproblem bezeichnet folgende kombinatorische Fragestellung:

- Gegeben ist eine ganze Zahl $K \geqslant 2$ und eine nichtleere Folge $a_1\, a_2 \ldots a_n \in \{1, \ldots, K\}^*$.
- Gefragt ist, ob es eine Teilmenge I von $\{1, \ldots, n\}$ gibt, so dass $\sum_{i \in I} a_i = K$.

Ein einfacher nichtdeterministischer Algorithmus, der das Teilsummenproblem löst, rät nichtdeterministisch eine Teilmenge I von $\{1, \ldots, n\}$ und prüft dann, ob die Teilsumme der geratenen Indexmenge gleich K ist. Für festes K können wir diese Idee durch einen NFA realisieren. Intuitiv bearbeitet der Automat die Eingabewerte $a_1, a_2, \ldots, a_n$ der Reihe nach und entscheidet nichtdeterministisch, die i-te Zahl a_i an der Teilsumme zu beteiligen oder nicht. Wir betrachten folgenden NFA $\mathcal{M}^{(K)}$, dessen Zustände die Zahlen $q \in \{0, 1, \ldots, K\}$ sind, welche jeweils für den Wert der bereits erzielten Teilsummen stehen. Der Automat verfügt über einen eindeutigen Anfangs- und Endzustand. Der Anfangszustand ist 0; dies entspricht der initialen Teilsumme 0. Der Endzustand ist K (der gewünschte Wert der Teilsumme). Wir definieren den NFA $\mathcal{M}^{(K)}$ wie folgt:

$$\mathcal{M}^{(K)} \;\stackrel{\text{def}}{=}\; (\{0, 1, \ldots, K\}, \{1, 2, \ldots, K\}, \delta, \{0\}, \{K\}),$$

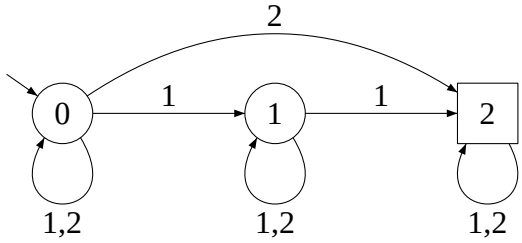


Abbildung 13: NFA $\mathcal{M}^{(K)}$ für das Teilsummenproblem ($K = 2$)

wobei für jeden Zustand $q \in \{0,1,\ldots,K\}$ und jedes Zeichen $a \in \{1,\ldots,K\}$:

$$\delta(q,a) \stackrel{\text{def}}{=} \{q, q+a\} \cap \{0,1,\ldots,K\} = \begin{cases} \{q\} & : \text{ falls } q+a > K \\ \{q, q+a\} & : \text{ falls } q+a \leqslant K \end{cases}$$

Für $K = 2$ hat der NFA $\mathcal{M}^{(K)}$ die in Abbildung 13 angegebene Gestalt. Man überlegt sich leicht, dass die von $\mathcal{M}^{(K)}$ akzeptierte Sprache genau die Menge aller Zahlenfolgen $a_1\, a_2 \ldots\, a_n \in \{1,2,\ldots,K\}^*$ ist, für die es eine Teilfolge $a_{i_1}\, a_{i_2} \ldots a_{i_\ell}$ gibt, deren Summe K ergibt.

Die Potenzmengenkonstruktion angewandt auf $\mathcal{M}^{(K)}$ liefert einen DFA $\mathcal{M}^{(K)}_{\text{det}}$, dessen Zustände Teilmengen von $\{0,1,\ldots,K\}$ sind:

$$\mathcal{M}^{(K)}_{\text{det}} = \big(\, 2^{\{0,1,\ldots,K\}},\ \{1,\ldots,K\},\ \delta_{\text{det}},\ \{0\},\ F_{\text{det}} \,\big),$$

wobei

$$\begin{aligned} F_{\text{det}} &= \big\{\, P \subseteq \{0,1,\ldots,K\} : K \in P \,\big\} \\ \delta_{\text{det}}(P,a) &= \big\{\, q \in \{0,1,\ldots,K\} : q \in P \text{ oder } q-a \in P \,\big\} \\ &= P \cup \big\{\, p+a : p \in P, p+a \leqslant K \,\big\} \end{aligned}$$

für $a \in \{1,\ldots,K\}$ und $P \subseteq \{0,1,\ldots,K\}$. Man überzeugt sich nun leicht davon, dass für jedes Eingabewort $w = a_1\, a_2 \ldots a_n$ über dem Alphabet $\{1,2,\ldots,K\}$ gilt:

$$\begin{gathered} w \text{ hat genau dann einen akzeptierenden Lauf in } \mathcal{M}^{(K)}_{\text{det}}, \\ \text{wenn } \sum_{i \in I} a_i = K \text{ für eine Teilmenge } I \text{ von } \{1,\ldots,n\}. \end{gathered}$$

Der DFA $\mathcal{M}^{(K)}_{\text{det}}$ kann also nun als Vorlage für einen deterministischen Algorithmus für das Teilsummenproblem (z.B. basierend auf Backtracking oder dynamischem Programmieren) dienen.
■

Endliche Automaten und reguläre Grammatiken (2. Teil)

In Lemma 2.7 auf Seite 24 haben wir gezeigt, dass die durch einen DFA akzeptierte Sprache durch eine reguläre Grammatik erzeugt wird. Die wesentliche Idee der Konstruktion einer regulären Grammatik bestand darin, die Übergänge $q \xrightarrow{a} p$ im DFA als Regeln $q \to ap$ einer Grammatik aufzufassen. Tatsächlich ist die im Beweis von Lemma 2.7 angegebene Transformation mit leichten Modifikationen auch für NFA einsetzbar, so dass die resultierende Grammatik sowohl regulär als auch ε-frei ist. Siehe Übungen.

Soweit zur Konstruktion regulärer Grammatiken für gegebenen endlichen Automaten. Umgekehrt können reguläre Grammatiken in NFA (und mit der Potenzmengenkonstruktion in DFA) überführt werden. Die Grundidee ist auch hier wieder eine Korrespondenz zwischen Übergängen $q \xrightarrow{a} p$ im NFA und den Regeln $q \to ap$ der Grammatik herzustellen.

Lemma 2.17 (Reguläre Grammatik $\rightsquigarrow$ NFA). *Zu jeder regulären Grammatik* G *gibt es einen NFA* $\mathcal{M}$ *mit* $\mathcal{L}(G) = \mathcal{L}(\mathcal{M})$.

Beweis. Sei $G = (V, \Sigma, \mathcal{P}, S)$ eine reguläre Grammatik. Wir definieren nun einen NFA

$$\mathcal{M} \stackrel{\text{def}}{=} (Q, \Sigma, \delta, Q_0, F),$$

der genau die Wörter der von G erzeugten Sprache akzeptiert. Die Zustandsmenge von $\mathcal{M}$ ist

$$Q \stackrel{\text{def}}{=} V \cup \{q_F\}, \text{ wobei } q_F \notin V.$$

Die Anfangszustandsmenge besteht aus dem Startsymbol von G, also $Q_0 \stackrel{\text{def}}{=} \{S\}$. Die Endzustandsmenge enthält den Spezialzustand q_F sowie alle Variablen A, aus denen das leere Wort herleitbar ist:

$$F \stackrel{\text{def}}{=} \{q_F\} \cup \{A \in V : A \to \varepsilon\}$$

Die Übergangsfunktion δ ist durch folgende beiden "Axiome" gegeben:

$$\begin{aligned} B \in \delta(A, a) \quad &\text{gdw} \quad A \to aB \\ q_F \in \delta(A, a) \quad &\text{gdw} \quad A \to a \end{aligned}$$

Dabei sind $A, B \in V$ und $a \in \Sigma$. Weiter ist $\delta(q_F, a) \stackrel{\text{def}}{=} \varnothing$ für alle $a \in \Sigma$.

Wir zeigen nun die Korrektheit der angegebenen Konstruktion. Hierzu ist $\mathcal{L}(\mathcal{M}) = \mathcal{L}(G)$ nachzuweisen. Zunächst stellen wir fest, dass das leere Wort entweder zu beiden Sprachen $\mathcal{L}(\mathcal{M})$ und $\mathcal{L}(G)$ oder zu keiner der beiden Sprachen gehört:

$$\varepsilon \in \mathcal{L}(G) \quad \text{gdw} \quad S \to \varepsilon \quad \text{gdw} \quad S \in F \quad \text{gdw} \quad \varepsilon \in \mathcal{L}(\mathcal{M}).$$

Sei nun $w = a_1 a_2 \ldots a_n \in \Sigma^+$. Zu zeigen ist, dass w genau dann in $\mathcal{L}(\mathcal{M})$ liegt, wenn w eine Herleitung in G besitzt.

Wir nehmen zunächst an, dass $w \in \mathcal{L}(G)$. Das Startsymbol ist per Definition der einzige Anfangszustand, also $Q_0 = \{S\}$. Da G eine reguläre Grammatik ist, gibt es zwei Arten von Ableitungen für das Wort w.
1. Fall: Sei

$$S \Rightarrow a_1 B_1 \Rightarrow a_1 a_2 B_2 \Rightarrow^* a_1 a_2 \ldots a_{n-1} B_{n-1} \Rightarrow a_1 a_2 \ldots a_{n-1} a_n$$

die Ableitung des Wortes $w$in G. Während die Regeln $B_{i-1} \to a_i B_i$ bzw. $S \to a_1 B_1$ in G Transitionen $B_{i-1} \xrightarrow{a_i} B_i$ bzw. $S \xrightarrow{a_1} B_1$ in $\mathcal{M}$ für $i = 2, \ldots, n-1$ entsprechen, korrespondiert die Regel $B_{n-1} \to a_n$ mit der Transition $B_{n-1} \xrightarrow{a_n} q_F$. Da q_F ein Endzustand ist, bildet die Zustandsfolge $SB_1B_2 \ldots B_{n-1}q_F$ einen akzeptierenden Lauf in $\mathcal{M}$. Damit ist $w \in \mathcal{L}(\mathcal{M})$.
2. Fall: Nun sei

$$S \Rightarrow a_1 B_1 \Rightarrow a_1 a_2 B_2 \Rightarrow^* a_1 a_2 \ldots a_{n-1} B_{n-1} \Rightarrow a_1 a_2 \ldots a_{n-1} a_n B_n \Rightarrow a_1 a_2 \ldots a_{n-1} a_n$$

die Ableitung des Wortes $w$in G. Wie im 1. Fall entsprechen die Regeln $B_{i-1} \to a_i B_i$ bzw. $S \to a_1 B_1$ in G den Transitionen $B_{i-1} \xrightarrow{a_i} B_i$ bzw. $S \xrightarrow{a_1} B_1$ in $\mathcal{M}$ für $i = 2, \ldots, n$. Wegen der Definition der Menge der Endzustände F impliziert die Regel $B_n \to \varepsilon$ in G, dass B_n ein Endzustand ist. Damit ist $SB_1B_2 \ldots B_{n-1}B_n$ ein akzeptierender Lauf in $\mathcal{M}$, und somit $w \in \mathcal{L}(\mathcal{M})$.

Es bleibt zu zeigen, dass jedes Wort, welches einen akzeptierenden Lauf in $\mathcal{M}$ hat, ein in G herleitbares Wort ist. Wir nehmen dazu nun an, dass $w \in \mathcal{L}(\mathcal{M})$. Dann gibt es einen akzeptierenden Lauf $q_0\, q_1 \ldots q_n$ für w in $\mathcal{M}$. Da Zustand q_F keine ausgehenden Transitionen hat und kein Startzustand ist, gibt es Variablen $B_0, B_1, \ldots, B_{n-1}$ in G, so dass $q_i = B_i$ für $0 \leqslant i < n$. Da der erste Zustand q_0 des akzeptierenden Laufs für w in $\mathcal{M}$ ein Anfangszustand von $\mathcal{M}$ ist, muss $B_0 = q_0 = S$ gelten. Weiter gilt

$$B_1 \in \delta(B_0, a_1), \quad B_2 \in \delta(B_1, a_2), \ \ldots, \ B_{n-1} \in \delta(B_{n-2}, a_{n-1}).$$

Also muss G die Regeln

$$B_0 \to a_1 B_1, \quad B_1 \to a_2 B_2, \ \ldots, \ B_{n-2} \to a_{n-1} B_{n-1}$$

enthalten. Wegen $B_0 = S$ ist

$$S \Rightarrow a_1 B_1 \Rightarrow a_1 a_2 B_2 \Rightarrow \ldots \Rightarrow a_1 a_2 \ldots a_{n-1} B_{n-1}$$

eine Herleitung in G. Der letzte Zustand q_n des akzeptierenden Laufs für w in $\mathcal{M}$ liegt in F. Also ist entweder $q_n = q_F$ oder $q_n = B$, wobei B ein Nichtterminal ist, für welches es in G die Regel $B \to \varepsilon$ gibt. Falls $q_n = q_F$, so enthält G die Regel $B_{n-1} \to a_n$. In diesem Fall ist w in G herleitbar, da

$$S \Rightarrow^* a_1 a_2 \ldots a_{n-1} B_{n-1} \Rightarrow a_1 a_2 \ldots a_{n-1} a_n$$

Ist $q_n = B$ für ein Nichtterminal B, so sind $B_{n-1} \to a_n B$ und $B \to \varepsilon$ Regeln in G. In diesem Fall ist w in G herleitbar, da

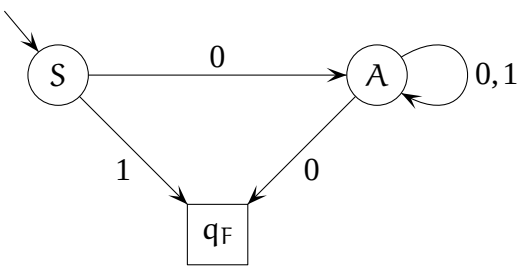
$$S \Rightarrow^* a_1 a_2 \ldots a_{n-1} B_{n-1} \Rightarrow a_1 a_2 \ldots a_{n-1} a_n B \Rightarrow a_1 a_2 \ldots a_{n-1} a_n \ = \ w$$

□

Beispiel 2.18 (Reguläre Grammatik ⇝ NFA). Für die reguläre Grammatik G mit den Regeln

$$S \ \to \ 1 \ \big| \ 0A, \quad A \ \to \ 0 \ \big| \ 0A \ \big| \ 1A$$

liefert die im Beweis von Lemma 2.17 angegebene Konstruktion folgenden NFA:



Fügt man in G die ε-Regel $A \to \varepsilon$ ein, so wird A zum Endzustand. ■

Corollar 2.19 (Äquivalenz von regulären Grammatiken und NFA/DFA). *Sei* $L \subseteq \Sigma^*$ *eine Sprache. Dann gilt:*

L *ist regulär, d.h.,* $L = \mathcal{L}(G)$ *für eine reguläre Grammatik* G

gdw *es gibt einen DFA* $\mathcal{M}$ *mit* $L = \mathcal{L}(\mathcal{M})$

gdw *es gibt einen NFA* $\mathcal{M}$ *mit* $L = \mathcal{L}(\mathcal{M})$

2.2 Eigenschaften regulärer Sprachen

Aus der Darstellbarkeit durch endliche Automaten lassen sich einige zentrale Eigenschaften regulärer Sprachen herleiten. Wir diskutieren Abschlusseigenschaften (Abschnitt 2.2.1) sowie Algorithmen für grundlegende Fragestellungen (Abschnitt 2.2.3) und stellen ein notwendiges Kriterium für reguläre Sprachen vor (Abschnitt 2.2.2).

2.2.1 Abschlusseigenschaften

Wir befassen uns zunächst mit Abschlusseigenschaften von regulären Sprachen unter den üblichen Operatoren für formale Sprachen. Wir werden sehen, dass reguläre Sprachen unter allen gängigen Verknüpfungsoperatoren (Vereinigung, Durchschnitt, Konkatenation, Kleeneabschluss und Komplementbildung) abgeschlossen sind. Hierzu erläutern wir, wie sich diese Operationen mit endlichen Automaten realisieren lassen.

Vereinigung. Seien $\mathcal{M}_1 = (Q_1, \Sigma, \delta_1, Q_{0,1}, F_1)$ und $\mathcal{M}_2 = (Q_2, \Sigma, \delta_2, Q_{0,2}, F_2)$ zwei NFA mit $Q_1 \cap Q_2 = \varnothing$. Die Grundidee zur Bildung des NFA für die Vereinigung besteht darin, für ein gegebenes Eingabewort w nichtdeterministisch zu entscheiden, mit welchem der beiden Automaten $\mathcal{M}_1$ oder $\mathcal{M}_2$ die Worterkennung durchgeführt wird. Wir definieren

$$\mathcal{M}_1 \uplus \mathcal{M}_2 \stackrel{\text{def}}{=} (Q_1 \cup Q_2, \Sigma, \delta, Q_{0,1} \cup Q_{0,2}, F_1 \cup F_2),$$

wobei

$$\delta(q,a) \stackrel{\text{def}}{=} \begin{cases} \delta_1(q,a) & : \text{ falls } q \in Q_1 \\ \delta_2(q,a) & : \text{ sonst.} \end{cases}$$

Offenbar gilt $\mathcal{L}(\mathcal{M}_1 \uplus \mathcal{M}_2) = \mathcal{L}(\mathcal{M}_1) \cup \mathcal{L}(\mathcal{M}_2)$. Man beachte, dass $\mathcal{M}_1 \uplus \mathcal{M}_2$ kein DFA ist; auch dann nicht, wenn $\mathcal{M}_1$ und $\mathcal{M}_2$ deterministisch sind.

Durchschnitt. Seien $\mathcal{M}_1 = (Q_1, \Sigma, \delta_1, Q_{0,1}, F_1)$ und $\mathcal{M}_2 = (Q_2, \Sigma, \delta_2, Q_{0,2}, F_2)$ zwei NFA. Einen NFA für die Schnittsprache $\mathcal{L}(\mathcal{M}_1) \cap \mathcal{L}(\mathcal{M}_2)$ erhalten wir durch eine Produktkonstruktion. Diese unterliegt der Vorstellung, dass $\mathcal{M}_1$ und $\mathcal{M}_2$ parallel geschaltet werden. Ist w das Eingabewort, dann starten wir die synchrone Bearbeitung des Worts w durch $\mathcal{M}_1$ und $\mathcal{M}_2$. Sobald einer der Automaten frühzeitig verwirft, dann auch der Produktautomat. Nur wenn beide Automaten akzeptieren, dann akzeptiert auch der Produktautomat. Wir definieren

$$\mathcal{M}_1 \otimes \mathcal{M}_2 \stackrel{\text{def}}{=} (Q_1 \times Q_2, \Sigma, \delta, Q_{0,1} \times Q_{0,2}, F_1 \times F_2),$$

wobei

$$\delta(\langle q_1, q_2\rangle, a) \stackrel{\text{def}}{=} \{\, \langle p_1, p_2\rangle : p_1 \in \delta_1(q_1, a), p_2 \in \delta_2(q_2, a) \,\}.$$

Für den Nachweis der Korrektheit ist zu zeigen, dass

$$\mathcal{L}(\mathcal{M}_1 \otimes \mathcal{M}_2) \;=\; \mathcal{L}(\mathcal{M}_1) \cap \mathcal{L}(\mathcal{M}_2).$$

Diese Aussage folgt aus der Beobachtung, dass die akzeptierenden Läufe im Produktautomaten $\mathcal{M}_1 \otimes \mathcal{M}_2$ die Form

$$\langle q_{0,1}, q_{0,2}\rangle \langle q_{1,1}, q_{1,2}\rangle \ldots \langle q_{n,1}, q_{n,2}\rangle$$

haben, wobei $q_{0,1}\, q_{1,1} \ldots q_{n,1}$ ein akzeptierender Lauf in $\mathcal{M}_1$ und $q_{0,2}\, q_{1,2} \ldots q_{n,2}$ ein akzeptierender Lauf in $\mathcal{M}_2$ sind. Daher wird jedes Wort in $\mathcal{L}(\mathcal{M}_1 \otimes \mathcal{M}_2)$ auch von $\mathcal{M}_1$ und $\mathcal{M}_2$ akzeptiert. Liegen umgekehrt akzeptierende Läufe $q_{0,1}\, q_{1,1} \ldots q_{n,1}$ und $q_{0,2}\, q_{1,2} \ldots q_{n,2}$ für ein Wort w in $\mathcal{M}_1$ bzw. $\mathcal{M}_2$ vor, so können diese zu einem akzeptierenden Lauf $\langle q_{0,1}, q_{0,2}\rangle \langle q_{1,1}, q_{1,2}\rangle \ldots \langle q_{n,1}, q_{n,2}\rangle$ im Produktautomaten zusammengesetzt werden. Also ist der Schnitt von $\mathcal{L}(\mathcal{M}_1)$ und $\mathcal{L}(\mathcal{M}_2)$ in der akzeptierten Sprache von $\mathcal{M}_1 \otimes \mathcal{M}_2$ enthalten.

Wird die Produktkonstruktion für zwei DFA $\mathcal{M}_1$, $\mathcal{M}_2$ durchgeführt, dann entsteht ein DFA, dessen Übergangsfunktion durch die Formel

$$\delta(\langle q_1, q_2\rangle, a) \stackrel{\text{def}}{=} \begin{cases} \langle \delta_1(q_1, a), \delta_2(q_2, a)\rangle & : \text{ falls } \delta_1(q_1, a) \neq \bot \text{ und } \delta_2(q_2, a) \neq \bot \\ \bot & : \text{ sonst.} \end{cases}$$

gegeben ist. Der Produktautomat von DFA ist also wieder ein DFA.

Beispiel 2.20 (Durchschnitt, Produkt-DFA). Als Beispiel betrachten wir die zwei DFA $\mathcal{M}_1$ und $\mathcal{M}_2$ links in Abbildung 14, welche die Sprachen

$$\begin{aligned} \mathcal{L}(\mathcal{M}_1) &= \{\, 1^n 00^m : m, n \geqslant 0 \,\} \\ \mathcal{L}(\mathcal{M}_2) &= \{\, 0x_1 0x_2 0 \ldots 0x_k 0 : k \geqslant 0, x_1, \ldots, x_k \in \{0,1\} \,\} \end{aligned}$$

akzeptieren. Der Produkt-DFA besteht aus vier Zuständen. Diese sind Paare bestehend aus je einem Zustand von $\mathcal{M}_1$ und $\mathcal{M}_2$. Der Anfangszustand ist $\langle q_0, p_0\rangle$. Der Endzustand ist $\langle q_1, p_1\rangle$. Man überzeugt sich leicht davon, dass die akzeptierte Sprache des Produkt-DFA genau aus den Wörtern 0^{2k+1}, $k \in \mathbb{N}$, besteht. Tatsächlich setzt sich die Durchschnittssprache $\mathcal{L}(\mathcal{M}_1) \cap \mathcal{L}(\mathcal{M}_2)$ aus genau denjenigen Wörtern w zusammen, die zugleich die Gestalt $1^n 00^m$ und $0x_1 0 \ldots 0x_k 0$, $x_i \in \{0,1\}$, haben. Also $n = 0$, $x_1 = \ldots = x_k = 0$ und $m = 2k$, und somit $1^n 00^m = 0^{2k+1}$. ■

Komplement. Für den Komplementoperator gehen wir von einem DFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ mit einer totalen Übergangsfunktion aus. Einen DFA $\overline{\mathcal{M}}$ für die Komplementsprache $\overline{\mathcal{L}(\mathcal{M})} = \Sigma^* \setminus \mathcal{L}(\mathcal{M})$ erhält man durch Komplementierung der Endzustandsmenge von $\mathcal{M}$:

$$\overline{\mathcal{M}} \;\stackrel{\text{def}}{=}\; (Q, \Sigma, \delta, q_0, Q \setminus F)$$

Da $\mathcal{M}$ total ist, besitzt jedes Wort $w = a_1 a_2 \ldots a_n \in \Sigma^*$ einen "vollständigen" Lauf $q_0\, q_1 \ldots q_n$ in $\mathcal{M}$. Dies ist zugleich der Lauf für w in $\overline{\mathcal{M}}$. Da die Endzustandsmengen in $\mathcal{M}$ und $\overline{\mathcal{M}}$ komplementär sind, ist $q_0\, q_1 \ldots q_n$ genau dann in $\mathcal{M}$ akzeptierend, wenn $q_0\, q_1 \ldots q_n$ in $\overline{\mathcal{M}}$ verwerfend ist:

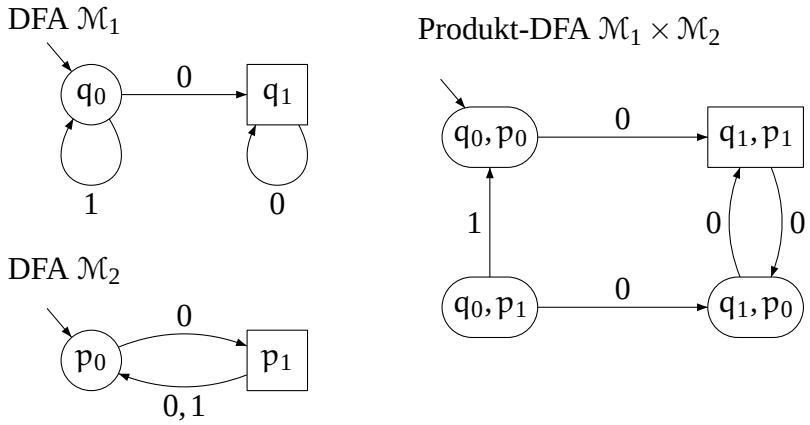


Abbildung 14: Beispiel für Produktkonstruktion $\mathcal{M}_1 \otimes \mathcal{M}_2$

$$w \in \mathcal{L}(\mathcal{M}) \quad \text{gdw} \quad q_n \in F \quad \text{gdw} \quad q_n \notin Q \setminus F \quad \text{gdw} \quad w \in \Sigma^* \setminus \mathcal{L}(\overline{\mathcal{M}})$$

Also ist $\overline{\mathcal{M}}$ ein DFA mit $\mathcal{L}(\overline{\mathcal{M}}) = \overline{\mathcal{L}(\mathcal{M})}$. Die Konstruktion des Komplementautomaten erfordert lediglich $\mathcal{O}(|Q|)$ Schritte.

Die Annahme, dass der vorliegende DFA eine totale Übergangsfunktion hat, ist wesentlich, da alle Wörter w von $\overline{\mathcal{M}}$ akzeptiert werden müssen, für die ein DFA $\mathcal{M}$ mit partieller Übergangsfunktion eine vorzeitig abbrechende (verwerfende) Berechnung hat. Die entsprechende Konstruktion schlägt auch für NFA fehl. Dieser Sachverhalt ist wegen der Asymmetrie von Akzeptanz und Verwurf in nichtdeterministischen Automaten nicht verwunderlich: ein NFA akzeptiert genau dann, wenn es wenigstens einen akzeptierenden Lauf gibt; es kann jedoch auch verwerfende Läufe für das betreffende Eingabewort geben. Andererseits wird ein Eingabewort w nur dann von einem NFA verworfen, wenn *alle* Läufe für w verwerfend sind. Liegt ein NFA vor, dann kann mit der Potenzmengenkonstruktion (siehe Beweis von Satz 2.15 auf Seite 29) ein äquivalenter DFA konstruiert und auf diesen der Komplementoperator angewendet werden.

Vereinigung für DFA. Wir haben erwähnt, dass die oben angegebene Konstruktion für die Vereinigung zunächst einen NFA liefert; auch wenn zwei DFA verknüpft werden. Liegen zwei DFA $\mathcal{M}_1$, $\mathcal{M}_2$ mit disjunkten Zustandsmengen vor, für die ein DFA für die Sprache $\mathcal{L}(\mathcal{M}_1) \cup \mathcal{L}(\mathcal{M}_2)$ erstellt werden soll, so kann man den Operator $\uplus$ anwenden und dann die Potenzmengenkonstruktion durchführen. Wenn $\mathcal{M}_1$ und $\mathcal{M}_2$ deterministisch sind, dann sind die erreichbaren Zustände der Potenzmengenkonstruktion zweielementige Mengen $\{q, p\}$ bestehend aus aus einem Zustand q in $\mathcal{M}_1$ und einem Zustand p von $\mathcal{M}_2$. Diese zweielementigen Mengen $\{q, p\}$ kann man als Paare $\langle q, p\rangle$ und Zustände einer für die Vereinigung modifizierten Produktkonstruktion auffassen. Diese modifizierte Produktkonstruktion kann man mittels der de Morganschen Regel

$$\mathcal{L}(\mathcal{M}_1) \cup \mathcal{L}(\mathcal{M}_2) \;=\; \overline{\overline{\mathcal{L}(\mathcal{M}_1)} \cap \overline{\mathcal{L}(\mathcal{M}_2)}},$$

welche die Vereinigung auf die Komplement- und Durchschnittsbildung zurückführt, aus der gewöhnlichen Produktkonstruktion für den Durchschnitt herleiten. Die Aussage, dass der DFA

$$\mathcal{M} \;=\; \overline{\overline{\mathcal{M}_1} \otimes \overline{\mathcal{M}_2}}$$

genau die Vereinigungssprache akzeptiert, kann man sich intuitiv wie folgt klarmachen. Die Endzustände von $\mathcal{M}$ sind genau diejenigen Zustandspaare $\langle q_1, q_2\rangle$, für die wenigstens einer der beiden Zustände q_1 oder q_2 ein Endzustand ist. Somit akzeptiert $\mathcal{M}$ genau dann, wenn wenigstens einer der Läufe in $\mathcal{M}_1$ oder $\mathcal{M}_2$ akzeptierend ist. Tatsächlich ist der Umweg über die Komplementierung unnötig, da es genügt, den Produktautomaten von $\mathcal{M}_1$ und $\mathcal{M}_2$ zu bilden und diesen mit der Endzustandsmenge

$$F = \{\, \langle q_1, q_2\rangle : q_1 \in F_1 \text{ oder } q_2 \in F_2 \,\}$$

zu versehen. Dabei müssen wir voraussetzen, dass $\mathcal{M}_1$ und $\mathcal{M}_2$ jeweils mit einer totalen Übergangsfunktion ausgestattet sind. Alternativ kann man den Zustandsraum des Produkts auch um die Zustandsräume von $\mathcal{M}_1$ und $\mathcal{M}_2$ erweitern (wobei $Q_1 \cap Q_2 = \varnothing$ unterstellt wird) und z.B. von Zustand $\langle q_1, q_1\rangle$ in Zustand $\delta_1(q_1, a)$ von $\mathcal{M}_1$ übergehen, falls das Zeichen a gelesen wird und $\delta_2(q_2, a) = \perp$. Die Menge F der Endzustände des Produkts ist dann um die Endzustände von $\mathcal{M}_1$ und $\mathcal{M}_2$ zu erweitern.

ε-NFA

Als technisches Hilfsmittel zur Behandlung von Konkatenation und Kleeneabschluss betrachten wir eine Erweiterung von NFA, in denen ε-Transitionen möglich sind, welche im Folgenden kurz ε-NFA genannt werden. ε-Transitionen sind *spontane Zustandsveränderungen*, welche unabhängig vom Zeichen unter dem Lesekopf stattfinden können und welche die Position des Lesekopfs unverändert lassen. Es können beliebig viele ε-Transitionen hintereinander stattfinden.

Definition 2.21 (ε-NFA). Ein *ε-NFA* ist ein Tupel $\mathcal{M} = (Q, \Sigma, \delta, Q_0, F)$, dessen Komponenten Q, Σ, Q_0 und F wie in einem NFA definiert sind und dessen Übergangsfunktion eine Funktion des Typs

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

ist. Ein akzeptierender Lauf für ein Wort $w \in \Sigma^*$ in $\mathcal{M}$ ist eine Zustandsfolge $q_0\, q_1 \ldots q_m$, so dass folgende Eigenschaften gelten:

(1) $q_0 \in Q_0$

(2) $m \geqslant |w|$ und es gibt Elemente $b_1, b_2, \ldots, b_m \in \Sigma \cup \{\varepsilon\}$, so dass $w = b_1\, b_2 \ldots b_m$ und $q_{i+1} \in \delta(q_i, b_{i+1})$ für $0 \leqslant i \leqslant m-1$.

(3) $q_m \in F$

Die akzeptierte Sprache von $\mathcal{M}$ ist dann wie für gewöhnliche NFA durch

$$\mathcal{L}(\mathcal{M}) \stackrel{\text{def}}{=} \{\, w \in \Sigma^* : \text{es gibt einen akzeptierenden Lauf für } w \text{ in } \mathcal{M} \,\}$$

definiert. ■

Als Beispiel betrachten wir den ε-NFA in Abbildung 15. Dieser akzeptiert die Sprache $L = \{b\} \cup \{b^n a : n \in \mathbb{N}\}$.

Offenbar kann jeder NFA (und damit auch jeder DFA) als ε-NFA interpretiert werden. Umgekehrt kann der Effekt von ε-Transitionen durch zusätzliche Transitionen mit Beschriftungen des Eingabealphabets und geeignete Wahl der Anfangs- oder Endzustandsmenge simuliert werden.

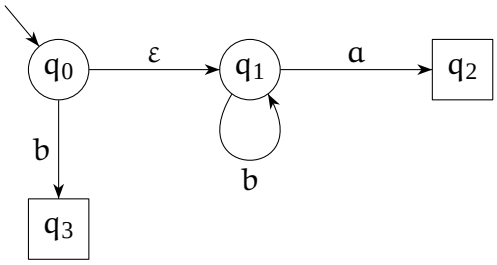


Abbildung 15: ε-NFA

Lemma 2.22 (ε-NFA ⇝ NFA). *Zu jedem ε-NFA $\mathcal{M}$ gibt es einen NFA $\mathcal{M}'$ mit $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$.*

Beweis. Sei $\mathcal{M} = (Q, \Sigma, \delta, Q_0, F)$ ein ε-NFA. Ein NFA $\mathcal{M}'$ ohne ε-Transitionen wird nun konstruiert, indem ein Zustand q eine Transition zu einem Zustand p hat ($q \xrightarrow{a} p$), wenn q in $\mathcal{M}$ eine a-Transition hat und anschließend durch beliebig viele (0 oder mehrere) ε-Transitionen p erreicht wird.

Die reflexive, transitive Hülle von $\xrightarrow{\varepsilon}$ wird mit $\overset{\varepsilon}{\Longrightarrow}$ bezeichnet, d. h., sind $q, p \in Q$, so gilt:

$$p \overset{\varepsilon}{\Longrightarrow} q \quad \text{gdw} \quad q \text{ ist von } p \text{ über 0 oder mehrere } \varepsilon\text{-Transitionen erreichbar}$$

Es gilt also $p \overset{\varepsilon}{\Longrightarrow} q$ genau dann, wenn es eine Zustandsfolge $p_0 p_1 \dots p_n$ der Länge $n \geqslant 0$ gibt, so dass $p_0 = p$, $p_n = q$ und $p_{i+1} \in \delta(p_i, \varepsilon)$ für $i = 0, 1, \dots, n-1$, d.h.,

$$p = p_0 \xrightarrow{\varepsilon} p_1 \xrightarrow{\varepsilon} \dots \xrightarrow{\varepsilon} p_n = q.$$

Wir konstruieren nun den NFA $\mathcal{M}'$, dessen Zustände, Alphabet und Endzustände wie im ε-NFA $\mathcal{M}$ definiert sind:

$$\mathcal{M}' \overset{\text{def}}{=} (Q, \Sigma, \delta', Q_0', F).$$

Weiter sind

$$\begin{aligned} Q_0' &= \{p \in Q \colon q_0 \overset{\varepsilon}{\Longrightarrow} p \text{ für ein } q_0 \in Q_0\} \qquad \text{und} \\ \delta'(q, a) &= \left\{p \in Q \colon q \xrightarrow{a} r \overset{\varepsilon}{\Longrightarrow} p \text{ für ein } r \in Q\right\}. \end{aligned}$$

Es bleibt zu zeigen, dass $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$.

"$\subseteq$": Sei $w = a_1 a_2 \dots a_n \in \Sigma^*$ ein Wort, wobei die a_i's Elemente des Alphabets Σ sind, so gilt $w \in \mathcal{L}(\mathcal{M})$ genau dann, wenn es eine Zustandsfolge $r_0\, q_0\, r_1\, q_1\, r_2 \dots q_{n-1}\, r_n\, q_n$ mit $r_0 \in Q_0$, $q_n \in F$ und

$$\begin{array}{ccccccccccccc} r_0 & \overset{\varepsilon}{\Longrightarrow} & q_0 & \xrightarrow{a_1} & r_1 & \overset{\varepsilon}{\Longrightarrow} & q_1 & \xrightarrow{a_2} & r_2 & \overset{\varepsilon}{\Longrightarrow} & \dots & \xrightarrow{a_{n-1}} & r_{n-1} \overset{\varepsilon}{\Longrightarrow} q_{n-1} \xrightarrow{a_n} r_n \overset{\varepsilon}{\Longrightarrow} q_n \\ \uparrow & & & & & & & & & & & & \uparrow \\ \in Q_0 & & & & & & & & & & & & \in F \end{array}$$

gibt. Es gilt also $q_0 \in Q_0'$ und $q_{i+1} \in \delta'(q_i, a_{i+1})$ für $i = 0, 1, \dots, n-1$, und somit $q_n \in \delta'(q_0, w)$. Die Zustandsfolge $q_0 q_1 \dots q_n$ ist also ein akzeptierender Lauf von $\mathcal{M}'$.

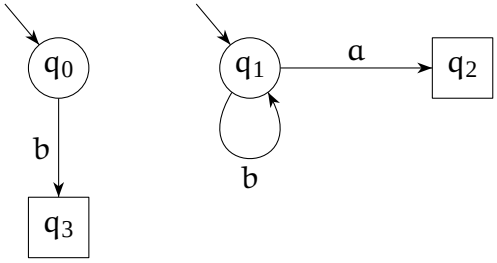
"$\supseteq$": Sei $w = a_1 a_2 \dots a_n \in \Sigma^*$ und sei $p_0 p_1 \dots p_n$ ein akzeptierender Lauf für w in $\mathcal{M}'$, d. h.,

$$p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p_n.$$

Dann ist $p_0 \in Q_0'$ und damit existiert ein $q_0 \in Q_0$, so dass $q_0 \stackrel{\varepsilon}{\Longrightarrow} p_0$. Außerdem sind $p_n \in F$. Für alle $i = 1, \ldots, n$ ist entweder $p_{i-1} \xrightarrow{a_i} p_i$ eine Transition in $\mathcal{M}$ oder es existiert ein Zustand $q_i \in Q$, so dass $p_{i-1} \xrightarrow{a_i} q_i$ und $q_i \stackrel{\varepsilon}{\Longrightarrow} p_i$. Damit hat $\mathcal{M}$ einen akzeptierenden Lauf der Form $q_0 \ldots p_0 q_1 \ldots p_1 q_2 \ldots p_{n-1} q_n \ldots p_n$, woraus $w \in \mathcal{L}(\mathcal{M})$ folgt. □

Statt eine Art ε-Abschluss den Transitionen der Übergangsfunktion anzuschließen wie im vorangegangenen Beweis, können bei der Konstrukion eines NFA aus einem ε-NFA auch alternative Definitionen von δ' genutzt werden mit entsprechenden Anpassungen der Anfangs- und Endzustandsmengen Q_0' und F'.

Dem in Abbildung 15 angegebenen ε-NFA $\mathcal{M}$ entspricht der gewöhnliche NFA, der aus $\mathcal{M}$ entsteht, indem die ε-Transition gelöscht wird und stattdessen Zustand q_1 als weiterer Anfangszustand deklariert wird. Es ergibt sich folgender NFA:



Ein weiteres Beispiel für die Transformation "NFA $\mathcal{M}$ mit ε-Transitionen $\rightsquigarrow$ NFA $\mathcal{M}'$" ist in Abbildung 16 angegeben. Hier ist die Menge der Anfangszustände im NFA $\mathcal{M}'$ durch die Menge von Anfangszuständen $\{q_0, q_1, q_2\}$ gegeben. Weiter gibt es in $\mathcal{M}'$ z.B. einen a-Übergang von q_1 nach q_2. Dieser ergibt sich aus den Transitionen

$$q_1 \xrightarrow{a} q_4 \xrightarrow{\varepsilon} q_2$$

in $\mathcal{M}$. Daher gilt $q_2 \in \delta'(q_1, a)$.

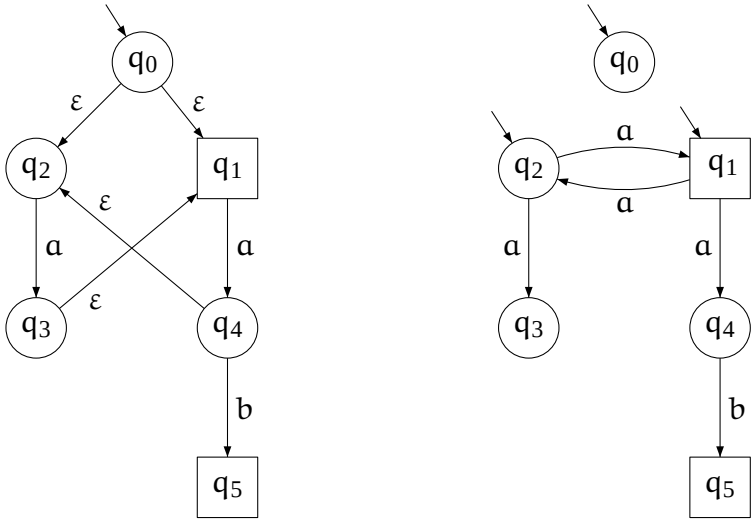


Abbildung 16: ε-NFA $\mathcal{M} \rightsquigarrow$ NFA $\mathcal{M}'$

Die Erweiterung von ε-NFA ermöglicht recht einfache Realisierungen von Konkatenation und Kleeneabschluss. In beiden Fällen gehen wir von ε-NFA $\mathcal{M}_1$, $\mathcal{M}_2$ bzw. $\mathcal{M}$ für gegebene reguläre Sprachen L_1 und L_2 bzw. L aus und konstruieren einen ε-NFA für $L_1 \circ L_2$ bzw. L^*. Der resultierende ε-NFA kann dann mit der im Beweis von Lemma 2.22 (Seite 39) angegebenen Methode zu einem gewöhnlichen NFA transformiert werden.

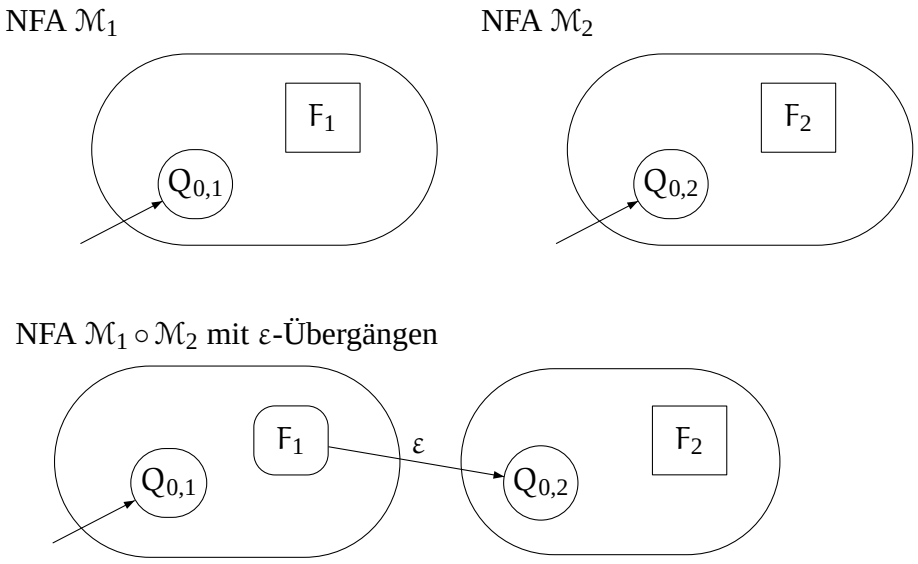


Abbildung 17: Idee für den Konkatenationsoperator für NFA

Konkatenation. Seien $\mathcal{M}_1 = (Q_1, \Sigma, \delta_1, Q_{0,1}, F_1)$ und $\mathcal{M}_2 = (Q_2, \Sigma, \delta_2, Q_{0,2}, F_2)$ zwei ε-NFA. Wir können ohne Einschränkung annehmen, dass $Q_1 \cap Q_2 = \varnothing$. Der ε-NFA

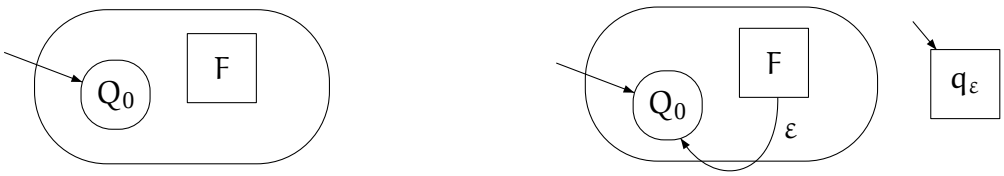
$$\mathcal{M}_1 \circ \mathcal{M}_2 \;=\; (Q_1 \cup Q_2, \Sigma, \delta, Q_{0,1}, F_2)$$

verbindet die Endzustände von $\mathcal{M}_1$ mit den Anfangszuständen von $\mathcal{M}_2$ durch einen ε-Übergang. Siehe Abildung 17. Wir formalisieren dies durch folgende Definition der Übergangsfunktion δ:

$$\delta(q,a) \;=\; \begin{cases} \delta_1(q,a) & : \text{ falls } q \in Q_1 \text{ und } a \in \Sigma \\ \delta_2(q,a) & : \text{ falls } q \in Q_2 \text{ und } a \in \Sigma \\ Q_{0,2} \cup \delta_1(q,\varepsilon) & : \text{ falls } q \in F_1 \text{ und } a = \varepsilon \\ \delta_1(q,\varepsilon) & : \text{ falls } q \in Q_1 \setminus F_1 \text{ und } a = \varepsilon \\ \delta_2(q,\varepsilon) & : \text{ falls } q \in Q_2 \text{ und } a = \varepsilon \end{cases}$$

Es ist leicht zu sehen, dass tatsächlich $\mathcal{L}(\mathcal{M}_1 \circ \mathcal{M}_2)$ mit der Sprache $\mathcal{L}(\mathcal{M}_1)\mathcal{L}(\mathcal{M}_2)$ übereinstimmt.

Kleeneabschluss. Sei $\mathcal{M} = (Q, \Sigma, \delta, Q_0, F)$ ein NFA mit oder ohne ε-Transitionen. Wir definieren einen ε-NFA $\mathcal{M}^*$ für die Sprache $\mathcal{L}(\mathcal{M})^*$ basierend auf der Idee, dass wir in allen Endzuständen von $\mathcal{M}$ mit einer ε-Transition zu einem Anfangszustand von $\mathcal{M}$ zurücksetzen können.



NFA $\mathcal{M}$ NFA $\mathcal{M}^*$ mit ε-Übergängen

Abbildung 18: Idee für den Kleeneoperator für NFA

Siehe Abbildung 18. Die formale Definition von $\mathcal{M}^*$ ist wie folgt:

$$\mathcal{M}^* \stackrel{\text{def}}{=} (Q \cup \{q_\varepsilon\}, \Sigma, \delta^*, Q_0 \cup \{q_\varepsilon\}, F \cup \{q_\varepsilon\}),$$

wobei $q_\varepsilon \notin Q$ ein neuer Zustand ist, der zugleich als Anfangs- und Endzustand von $\mathcal{M}^*$ deklariert wird. Die Übergangsfunktion δ^* von $\mathcal{M}^*$ verbindet die Endzustände von $\mathcal{M}$ über einen ε-Übergang mit den Anfangszuständen von $\mathcal{M}$. Falls $a \in \Sigma$ und $q \in Q$, so ist $\delta^*(q,a) = \delta(q,a)$. Die ε-Transitionen der Zustände $q \in Q$ sind durch

$$\delta^*(q,\varepsilon) \;=\; \begin{cases} Q_0 \cup \delta(q,\varepsilon) & : \text{ falls } q \in F \\ \delta(q,\varepsilon) & : \text{ falls } q \in Q \setminus F \end{cases}$$

gegeben. Zustand q_ε hat keine ausgehenden Transitionen, d.h., $\delta^*(q_\varepsilon, a) = \varnothing$ für alle $a \in \Sigma \cup \{\varepsilon\}$. Der Spezialzustand q_ε wird benötigt, um sicherzustellen, dass das leere Wort akzeptiert wird. Man kann auf q_ε verzichten, falls $\varepsilon \in \mathcal{L}(\mathcal{M})$. Es ist leicht zu sehen, dass $\mathcal{L}(\mathcal{M}^*) = \mathcal{L}(\mathcal{M})^*$.

Für den Plusoperator kann man die Gleichung $L^+ = L \circ L^*$ verwenden und $+$ mittels Konkatenation und Kleeneabschluss für ε-NFA realisieren. Alternativ kann man auch einen ε-NFA $\mathcal{M}^+$ genau wie für den Kleeneabschluss konstruieren und lediglich auf den zusätzlichen Anfangs- und Endzustand q_ε verzichten. $\mathcal{M}^+$ geht also aus $\mathcal{M}$ hervor, indem ε-Transitionen von allen End- zu allen Anfangszuständen eingefügt werden.

Satz 2.23. *Die Klasse der regulären Sprachen ist unter Vereinigung, Konkatenation, Kleeneabschluss, Komplementbildung und Durchschnitt abgeschlossen.*

Größe der konstruierten endlichen Automaten

Die oben angegebenen Operatoren für endliche Automaten stellen zugleich algorithmische Verfahren zur Realisierung der fünf Kompositionsoperatoren für reguläre Sprachen dar, die durch endliche Automaten gegeben sind. Wir diskutieren nun die Größe der konstruierten endlichen Automaten gemessen an der Anzahl an Zuständen. Ist $\mathcal{M} = (Q, \Sigma, \delta, Q_0, F)$ ein ε-NFA (oder NFA oder DFA), so bezeichnet $|\mathcal{M}|$ die Größe des Zustandsraums von $\mathcal{M}$, also

$$|\mathcal{M}| \stackrel{\text{def}}{=} |Q| \;=\; \text{Anzahl an Zustände in } \mathcal{M}$$

Die Effizienz von Algorithmen, die mit endlichen Automaten operieren, wird gemessen an der Anzahl an Zuständen plus Anzahl an Transitionen. Hierfür verwenden wir auch die Bezeichnung *size*$(\mathcal{M})$. Der exakte Wert von *size*$(\mathcal{M})$ ist also $|Q|$ plus die Anzahl an Tripeln

$(q, a, p) \in Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ mit $p \in \delta(q, a)$. Dies entspricht der Größe von $\mathcal{M}$ in der für Graphen bekannten Adjazenzlistendarstellung. Offenbar gilt

$$|\mathcal{M}| \leqslant \mathit{size}(\mathcal{M}) \leqslant |Q| + |Q|^2 \cdot (|\Sigma| + 1) = \mathcal{O}(|\mathcal{M}|^2),$$

wobei sich die Angabe der Größenordnung mit dem Operator $\mathcal{O}$ auf ein *festes* Alphabet bezieht. Je nach Kontext bezeichnen wir $|\mathcal{M}|$ oder $\mathit{size}(\mathcal{M})$ als Größe des ε-NFA $\mathcal{M}$.

Wir diskutieren nun die Größe der endlichen Automaten, die durch den Einsatz der oben erläuterten Operatoren entstehen. Wir konzentrieren uns hier auf die Größe des Zustandsraums. Analoge Betrachtungen können für den Größenoperator $\mathit{size}(\mathcal{M})$ durchgeführt werden. Die beschriebene Transformation eines ε-NFA $\mathcal{M}$ in einen äquivalenten NFA $\mathcal{M}'$ hat keinen Einfluss auf die Zustände. Es gilt also $|\mathcal{M}| = |\mathcal{M}'|$. Die Potenzmengenkonstruktion, die einen NFA $\mathcal{M}$ in einen äquivalenten DFA $\mathcal{M}_{\text{det}}$ überführt, verursacht ein exponentielles Blowup. Mit der im Beweis von Satz 2.15 auf Seite 29 angegebenen Definition von $\mathcal{M}_{\text{det}}$ gilt

$$\left|\mathcal{M}_{\text{det}}\right| = 2^{|\mathcal{M}|}.$$

Oftmals sind zwar nicht alle Teilmengen P der Zustandsmenge Q von $\mathcal{M}$ in $\mathcal{M}_{\text{det}}$ erreichbar, jedoch werden wir später sehen, dass das exponentielle Wachstum für den Übergang von NFA zu äquivalenten DFA unvermeidbar sein kann.

Nun zu den Kompositionsoperatoren. Die Größe des Zustandsraums des NFA $\mathcal{M}_1 \uplus \mathcal{M}_2$ für die Vereinigung ist $|\mathcal{M}_1| + |\mathcal{M}_2|$. Durch die Realisierung des Durchschnitts mittels des Produktoperators $\otimes$ ergibt sich ein NFA mit $|\mathcal{M}_1| \cdot |\mathcal{M}_2|$ Zuständen. Wendet man eine on-the-fly Konstruktion an, die nur den erreichbaren Teil des Produktautomaten konstruiert, so kann der Zustandsraum des resultierenden Automaten für die Durchschnittssprache selbstverständlich kleiner als $|\mathcal{M}_1| \cdot |\mathcal{M}_2|$ sein. Dasselbe gilt für den modifizierten Produkt-Operator als Vereingungsoperator für DFA. Der Komplementoperator für DFA lässt den Zustandsraum unverändert. Für jeden DFA $\mathcal{M}$ gilt also $|\overline{\mathcal{M}}| = |\mathcal{M}|$. Startet man mit einem NFA $\mathcal{M}$ und wendet zunächst die Potenzmengenkonstruktion an, um dann den Komplementbildung einzusetzen, so ist ein exponentielles Blowup möglich. Die Operationen Konkatenation und Kleeneabschluss sind für NFA oder ε-NFA effizient. Es gilt $|\mathcal{M}_1 \circ \mathcal{M}_2| = |\mathcal{M}_1| + |\mathcal{M}_2|$ und $|\mathcal{M}^*| = |\mathcal{M}| + 1$.

2.2.2 Das Pumping Lemma für reguläre Sprachen

Der folgende Satz (in der Literatur als Pumping Lemma bekannt) präsentiert ein notwendiges Kriterium für reguläre Sprachen und kann für den Nachweis genutzt werden, dass eine Sprache *nicht* regulär ist.

Satz 2.24 (Pumping Lemma für reguläre Sprachen). *Sei $L \subseteq \Sigma^*$ eine reguläre Sprache. Dann gibt es eine ganze Zahl $n \geqslant 1$, so dass jedes Wort $z \in L$ mit $|z| \geqslant n$ wie folgt zerlegt werden kann: $z = uvw$ mit Wörtern $u, v, w \in \Sigma^*$, so dass*

(1) $uv^k w \in L$ *für alle* $k \in \mathbb{N}$

(2) $|v| \geqslant 1$

(3) $|uv| \leqslant n$.

Beweis. Da L regulär ist, gibt es einen NFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ mit $\mathcal{L}(\mathcal{M}) = L$ (Corollar 2.19 auf Seite 35). Sei $|Q| = n$. Wir zeigen nun, dass jedes Wort in $\mathcal{L}(\mathcal{M})$ der Länge $\geqslant n$ in Teilwörter u, v, w zerlegt werden kann, so dass die oben genannten Bedingungen (1), (2) und (3) erfüllt sind. Sei

$$z \;=\; a_1 a_2 \ldots a_m \in \mathcal{L}(\mathcal{M}) \;\text{ mit }\; |z| = m \geqslant n$$

und sei $q_0\, q_1 \ldots q_m$ ein akzeptierender Lauf für z in $\mathcal{M}$. Dann ist $q_m \in F$. Wir betrachten nur die ersten $n+1$ Zustände dieses Laufs. Da $\mathcal{M}$ genau n Zustände hat, gibt es Indizes i, j mit $0 \leqslant i < j \leqslant n$, so dass $q_i = q_j$. Wir zerlegen z wie folgt:

$$u = a_1 a_2 \ldots a_i, \quad v = a_{i+1} a_{i+2} \ldots a_j, \quad w = a_{j+1} a_{j+2} \ldots a_m.$$

Wir weisen nun die geforderten Eigenschaften (1), (2) und (3) nach.

(2) Wegen $i < j$ gilt $|v| \geqslant 1$.

(3) Wegen $j \leqslant n$ gilt $|uv| = |a_1 \ldots a_j| = j \leqslant n$.

(1) Sei $k \in \mathbb{N}$. Wir betrachten das Wort

$$uv^k w \;=\; \underbrace{a_1 \ldots a_i}_{=u}\underbrace{a_{i+1} \ldots a_j}_{=v}\underbrace{a_{i+1} \ldots a_j}_{=v} \;\ldots\; \underbrace{a_{i+1} \ldots a_j}_{=v}\underbrace{a_{j+1} \ldots a_m}_{=w}$$

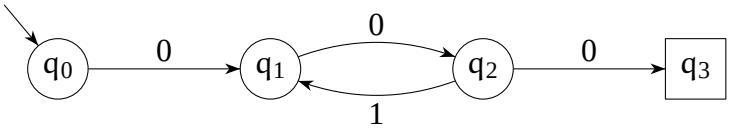
Das Wort $uv^k w$ hat einen akzeptierenden Lauf der Form

$$\begin{aligned} & q_0 \ldots q_i\, q_{i+1} \ldots q_j\, q_{i+1} \ldots q_j \ldots q_{i+1} \ldots q_j\, q_{j+1} \ldots q_m \\ = \;& q_0 \ldots q_i\, (q_{i+1}, \ldots, q_j)^k\, q_{j+1} \ldots q_m \end{aligned}$$

Dabei benutzen wir die Tatsache, dass $q_i = q_j$. Wegen $q_m \in F$ folgt $uv^k w \in \mathcal{L}(\mathcal{M}) = L$.

□

Wir machen uns die Aussage von Satz 2.24 an einem einfachen Beispiel klar. Wir betrachten folgenden DFA $\mathcal{M}$ für die Sprache $L = \left\{ 0(01)^m 00 : m \in \mathbb{N} \right\}$.



Die Konstante n aus dem Pumping Lemma kann hier $n = 4$ (Anzahl an Zustände) gewählt werden. Jedes Wort $x \in \mathcal{L}(\mathcal{M})$ der Länge $\geqslant 4$ hat die Form $x = uvw$, wobei

$$u = 0, \quad v = 01 \quad \text{und} \quad w = (01)^m 00$$

und die Pump-Eigenschaft (1) $uv^k w = 0(01)^k w \in \mathcal{L}(\mathcal{M}) = L$ für alle $k \in \mathbb{N}$ (2) $v \neq \varepsilon$, sowie (3) $|uv| = 3 < 4 = n$.

Nicht-reguläre Sprachen. Das Pumping Lemma kann für den Nachweis verwendet werden, dass eine Sprache nicht regulär ist. Als Beispiel betrachten wir

$$L = \{ a^n b^n : n \in \mathbb{N}, n \geqslant 1 \}.$$

Eine informelle Erklärung, warum L nicht regulär ist, ergibt sich daraus, dass das Modell endlicher Automaten nicht mächtig genug ist, um L darzustellen. Dies liegt im Wesentlichen daran, dass endliche Automaten keinen unbeschränkten Speicher haben; sondern lediglich die Zustände zur Speicherung von Daten verwenden können. Endliche Automaten können so konzipiert werden, dass sie bis 2, 3 oder bis zu einem anderen *festen* (von der Eingabe unabhängigen) Wert k "zählen" können; jedoch sind sie nicht in der Lage, von der Eingabe abhängige Werte zu speichern. Beim Einlesen der Eingabe kann ein endlicher Automat zwar prüfen, ob das Eingabewort von der Form $a^n b^m$ ist. Ein endlicher Automat kann jedoch nicht die präzise Anzahl der gelesenen a's speichern, um dann zu prüfen, ob ebensoviele b's folgen.

Wir weisen nun formal nach, dass L nicht regulär ist. Hierzu zeigen wir, dass die im Pumping Lemma angegebene Bedingung verletzt ist. Wir nehmen an, L wäre regulär, und führen diese Annahme zu einem Widerspruch. Sei n die Zahl aus dem Pumping-Lemma und sei $x = a^n b^n$. Weiter seien u, v und w Teilworte von x mit $x = uvw$ und den Eigenschaften (1), (2) und (3) aus dem Pumping Lemma.

Da $|uv| \leqslant n$ ist, besteht v nur aus a's.

Etwa $v = a^l$. Dann ist $l = |v| \geqslant 1$. Somit ist

$$uv^2w \;=\; uvvw \;=\; a^{n+l}b^n \;\notin L.$$

Widerspruch zu Eigenschaft (1).

Ebenfalls mit Hilfe des Pumping Lemmas kann man zeigen, dass auch die Sprache L' aller Wörter $w \in \{a, b\}^*$, die ebenso viele a's wie b's enthalten, *nicht* regulär ist. Eine andere Methode, um den Nachweis zu erbringen, dass L' nicht regulär ist, geht wie folgt. Offenbar ist

$$L \;=\; L' \cap L'', \;\text{ wobei } L'' = \{ a^n b^m : n, m \geqslant 0 \}.$$

Die Sprache L'' ist regulär. Dies kann man z.B. durch die Angabe eines endlichen Automaten $\mathcal{M}$ für L'' belegen.

> Hierzu genügt ein DFA mit zwei Zuständen q_a und q_b, die beide als Endzustände deklariert sind. Der Anfangszustand ist q_a. Weiter ist $\delta(q_a, a) = q_a$, $\delta(q_a, b) = \delta(q_b, b) = q_b$ und $\delta(q_b, a) = \bot$.

Wäre nun L' regulär, dann wäre (gemäß der Aussage von Satz 2.23, Seite 42) auch die Sprache $L = L' \cap L''$ regulär. Dies ist nicht der Fall, wie wir oben gesehen haben.

Die Aussage des Pumping Lemmas stellt eine notwendige Bedingung für reguläre Sprachen dar. Wir erwähnen ohne Beweis, dass es nicht-reguläre Sprachen gibt, welche das im Pumping Lemma angegebene Kriterium erfüllen.

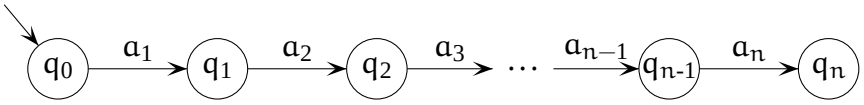
2.2.3 Algorithmen für endliche Automaten

In engem Zusammenhang zur Konstruktion von endlichen Automaten für gegebene reguläre Sprachen stehen Analysealgorithmen, wovon Verfahren für das Wortproblem und der Äquivalenztest zu den wichtigsten Fragestellungen zählen.

Das Wortproblem. Ist $\mathcal{M}$ ein DFA mit dem Alphabet Σ und w ein Wort über Σ, dann kann die Frage nach dem Wortproblem der akzeptierten Sprache von $\mathcal{M}$, d.h., "gilt $w \in \mathcal{L}(\mathcal{M})$?" in linearer Zeit $\mathcal{O}(|w|)$ beantwortet werden. Wir müssen lediglich den DFA $\mathcal{M}$ bei Eingabe w simulieren. Für NFA kann der für $w = a_1 a_2 \ldots a_n$ relevante Teil des Potenzmengen-DFA $\mathcal{M}_{\text{det}}$ konstruiert werden, was einer Berechnung von $\delta(Q_0, w)$ für die erweiterte Übergangsfunktion im NFA mit der Berechnungsvorschrift

$$P_0 := Q_0 \text{ und } P_{i+1} := \bigcup_{p \in P_i} \delta(p, a_{i+1}) \text{ für } i = 0, 1, \ldots, n-1.$$

Anschliessend ist zu prüfen, ob $P_n = \delta(Q_0, w)$ wenigstens einen Endzustand von $\mathcal{M}$ enthält. Alternativ kann man einen DFA $\mathcal{M}_w$ für die einelementige Sprache $\{w\}$ konstruieren (siehe Skizze unten), dann das Produkt $\mathcal{M}_w \otimes \mathcal{M}$ bilden und anschliessend einen Nichtleerheitstest (siehe unten) auf den Produkt-NFA anwenden. Dieses Verfahren beruht auf der Beobachtung, dass $w \in \mathcal{L}(\mathcal{M})$ genau dann gilt, wenn $\mathcal{L}(\mathcal{M}_w \otimes \mathcal{M}) = \mathcal{L}(\mathcal{M}_w) \cap \mathcal{L}(\mathcal{M}) = \{w\} \cap \mathcal{L}(\mathcal{M}) = \{w\}$.



Leerheitstest. Die Frage "gilt $\mathcal{L}(\mathcal{M}) = \varnothing$?" für einen NFA (oder DFA) $\mathcal{M}$ lässt sich mit einer Erreichbarkeitsanalyse realisieren. Wir müssen lediglich prüfen, ob einer der Endzustände von einem der Anfangszustände erreichbar ist. Dazu können wir eine Tiefen- oder Breitensuche anwenden, wahlweise "vorwärts" (d.h., mit den Anfangszuständen beginnend nach einem erreichbaren Zustand $p \in F$ suchend entlang der Kanten, die duch die Übergangsfunktion induziert werden) oder "rückwärts" (d.h., mit den Endzuständen beginnend und nach einem Anfangszustand suchend entlang der Kanten (p, q), falls $p \in \bigcup_{a \in \Sigma} \delta(q, a)$). In beiden Fällen sind die Kosten für den Leerheitstest linear in $\mathit{size}(\mathcal{M})$.

Universalität. Dual zu dem Leerheitsproblem ist die Frage nach der Universalität eines endlichen Automaten. Hiermit ist die Frage "gilt $\mathcal{L}(\mathcal{M}) = \Sigma^*$?" gemeint, wobei Σ für das Alphabet von $\mathcal{M}$ steht. Offenbar gilt $\mathcal{L}(\mathcal{M}) = \Sigma^*$ genau dann, wenn $\mathcal{L}(\overline{\mathcal{M}}) = \varnothing$. Daher kann der Universalitätstest für DFA durch Komplementierung der Endzustandsmenge (und eventuell vorangegangene Totalisierung) auf das Leerheitsproblem zurückgeführt werden.

Inklusions- und Äquivalenztest. Die Frage, ob zwei DFA $\mathcal{M}_1$ und $\mathcal{M}_2$ äquivalent sind, d.h. "gilt $\mathcal{L}(\mathcal{M}_1) = \mathcal{L}(\mathcal{M}_2)$?" lässt sich auf das Inklusionsproblem reduzieren, da:

$$\mathcal{L}(\mathcal{M}_1) = \mathcal{L}(\mathcal{M}_2) \quad \text{gdw} \quad \mathcal{L}(\mathcal{M}_1) \subseteq \mathcal{L}(\mathcal{M}_2) \text{ und } \mathcal{L}(\mathcal{M}_2) \subseteq \mathcal{L}(\mathcal{M}_1)$$

Für den Inklusionstest können wir folgende Beobachtung ausnutzen:

$$\begin{aligned} \mathcal{L}(\mathcal{M}_1) \subseteq \mathcal{L}(\mathcal{M}_2) \quad & \text{gdw} \quad \mathcal{L}(\mathcal{M}_1) \cap \overline{\mathcal{L}(\mathcal{M}_2)} = \varnothing \\ & \text{gdw} \quad \mathcal{L}\left(\mathcal{M}_1 \otimes \overline{\mathcal{M}_2}\right) = \varnothing \end{aligned}$$

Zur Beantwortung der Frage "gilt $\mathcal{L}(\mathcal{M}_1) \subseteq \mathcal{L}(\mathcal{M}_2)$?" können wir also wie folgt verfahren. Wir bilden den Produktautomaten aus $\mathcal{M}_1$ und dem Komplementautomaten von $\mathcal{M}_2$ und führen für diesen den Leerheitstest durch. Die Laufzeit für den Inklusions- und Äquivalenztest für gegebene DFA $\mathcal{M}_1$, $\mathcal{M}_2$ ist linear in der Größe der Produkt-DFA $\mathcal{M}_1 \otimes \overline{\mathcal{M}_2}$ bzw. $\overline{\mathcal{M}_1} \otimes \mathcal{M}_2$, und kann daher durch $\Theta(\mathit{size}(\mathcal{M}_1) \cdot \mathit{size}(\mathcal{M}_2))$ angebenen werden. Hierbei steht $\mathit{size}(\mathcal{M})$ für die Anzahl an Zuständen und Transitionen in $\mathcal{M}$. Für gegebene NFA kann der Inklusions- und Äquivalenztest mit der beschriebenen Methode und vorangegangener Determinisierung gelöst werden. Die Kosten werden dann jedoch durch die Determinisierung dominiert und sind im schlimmsten Fall exponentiell.

Endlichkeitstest. Das Endlichkeitsproblem für endliche Automaten fragt, ob die akzeptierte Sprache endlich ist. Das Endlichkeitsproblem kann für NFA dank folgender Beobachtung gelöst werden. Die durch einen NFA $\mathcal{M}$ akzeptierte Sprache ist genau dann *unendlich*, wenn es einen Anfangszustand q_0, einen Endzustand p und einen Zustand r gibt, so dass r von q_0 und p von r erreichbar ist und r auf einem Zyklus liegt. Diese Bedingung kann durch den Einsatz von Graphalgorithmen, welche hier nicht erläutert werden, in linearer Zeit gelöst werden.

2.3 Reguläre Ausdrücke

In Satz 2.23 auf Seite 42 haben wir gesehen, dass die Klasse der regulären Sprachen unter allen gängigen Verknüpfungsoperatoren (Vereinigung, Durchschnitt, Komplement, Konkatenation und Kleeneabschluss) abgeschlossen ist. Satz 2.23 kann dahingehend verschärft werden, dass die Klasse der regulären Sprachen über Σ die kleinste Klasse ist, die unter den Operationen Vereinigung, Konkatenation und Kleeneabschluss abgeschlossen ist und welche die Sprachen $\varnothing$, $\{\varepsilon\}$ und $\{a\}$, $a \in \Sigma$ enthält. Um diese Aussage zu belegen, betrachten wir einen weiteren Formalismus, der sehr intuitive Schreibweisen für reguläre Sprachen ermöglicht.

Definition 2.25 (Syntax regulärer Ausdrücke). Sei Σ ein Alphabet. Die Menge der regulären Ausdrücke über Σ ist durch folgende induktive Definition gegeben.

1. $\varnothing$ und ε sind reguläre Ausdrücke.
2. Für jedes $a \in \Sigma$ ist a ein regulärer Ausdruck.
3. Mit α und β sind auch $(\alpha\beta)$, $(\alpha + \beta)$ und (α^*) reguläre Ausdrücke.
4. Nichts sonst ist ein regulärer Ausdruck.

Die Klammern werden oftmals weggelassen, wobei der Verknüpfungsoperator $+$ die schwächste Priorität hat und der Sternoperator am stärksten bindet. Z.B. steht $a\varepsilon + bc^*$ für $((a\varepsilon) + (b(c^*)))$.[5] ■

[5] Ähnlich wie in Beispiel 1.10 auf Seite 13 kann anstelle der induktiven Definition eine kontextfreie Grammatik für die Syntax vollständig oder sparsam geklammerter regulärer Ausdrücke angegeben werden.

Anstelle der oben angegebenen induktiven Definition verwendet man auch häufig eine saloppe BNF-ähnliche Kurzschreibweise (*abstrakte Syntax* genannt), die die Option für das Setzen von Klammern als Selbstverständlichkeit unterstellt und α, β als Metasymbole für reguläre Ausdrücke and a für die Elemente des zugrundeliegenden Alphabets verwendet.

$$\alpha ::= \varnothing \mid \varepsilon \mid a \mid \alpha\beta \mid \alpha+\beta \mid \alpha^*$$

Reguläre Ausdrücke stehen für Sprachen. Intuitiv sind ε und a Kurzschreibweisen für die jeweils einelementigen Sprachen $\{\varepsilon\}$ und $\{a\}$. Der Ausdruck $\alpha\beta$ steht für die Sprache, die sich durch Konkatenation der Sprachen für α und β ergibt. Manchmal verwenden wir hierfür auch das Konkatenationssymbol $\circ$ und schreiben $\alpha\circ\beta$ statt $\alpha\beta$. (In der Literatur wird manchmal auch ein Strichpunkt ";" anstelle von $\circ$ als Symbol für die Konkatenation verwendet.) Das Symbol $+$ steht für die Vereinigung, der Sternoperator $*$ für den Kleeneabschluss. Der Plusoperator ist durch $\alpha^+ = \alpha^*\alpha$ definiert. Anstelle von $\alpha+\beta$ wird in der Literatur auch oftmals $\alpha|\beta$ geschrieben.

Die *Länge* eines regulären Ausdrucks α wird mit $|\alpha|$ bezeichnet. Sie ist definiert als die Länge von α aufgefasst als Wort über dem Alphabet $\Sigma \cup \{+, *, (,), \varepsilon, \varnothing\}$. Für unsere Zwecke wird nur die asymptotische Länge eine Rolle spielen. Diese ist durch die Anzahl an Operatoren (Vereinigung $+$, Konkatenation $\circ$ und Kleeneabschluss $*$) in α gegeben. So enthält z.B. der reguläre Ausdruck $a\varepsilon + bc^*$ insgesamt vier Operatoren (zwei Konkatenationen und je einmal "$+$" und "$*$").

Definition 2.26 (Semantik regulärer Ausdrücke). Die zu einem regulären Ausdruck α gehörende Sprache $\mathcal{L}(\alpha)$ ist wie folgt definiert.

$$\begin{array}{rclcrcl} \mathcal{L}(\varnothing) & = & \varnothing & & \mathcal{L}(\varepsilon) & = & \{\varepsilon\} \\ \mathcal{L}(a) & = & \{a\} & & \mathcal{L}(\alpha\beta) & = & \mathcal{L}(\alpha)\mathcal{L}(\beta) \\ \mathcal{L}(\alpha+\beta) & = & \mathcal{L}(\alpha)\cup\mathcal{L}(\beta) & & \mathcal{L}(\alpha^*) & = & \mathcal{L}(\alpha)^* \end{array}$$

Zwei reguläre Ausdrücke α_1, α_2 heißen *äquivalent* (i. Z. $\alpha_1 \equiv \alpha_2$), wenn $\mathcal{L}(\alpha_1) = \mathcal{L}(\alpha_2)$. Wir nennen α nichtleer, wenn $\alpha \not\equiv \varnothing$; also wenn $\mathcal{L}(\alpha) \neq \varnothing$. ■

Alle Rechengesetze für formale Sprachen und den Vereinigungs-, Konkatenations- und Sternoperator (siehe Seite 7) übertragen sich auf reguläre Ausdrücke. So gilt z.B. das Assoziativgesetz $(\alpha+\beta)+\gamma \equiv \alpha+(\beta+\gamma)$ und Kommutativgesetz $\alpha+\beta \equiv \beta+\alpha$ sowie die beiden Distributivgesetze:

$$\alpha(\beta+\gamma) \equiv \alpha\beta+\alpha\gamma \qquad (\beta+\gamma)\alpha \equiv \beta\alpha+\gamma\alpha$$

Im Allgemeinen gilt jedoch $(\alpha+\beta)^* \not\equiv \alpha^*+\beta^*$, da z. B. das Wort ab in der zu $(a+b)^*$ gehörenden Sprache liegt, jedoch aber nicht in der zu a^*+b^* gehörenden Sprache. Weitere Beispiele sind die Neutralität von ε für den Konkatenationsoperator $\varepsilon\alpha \equiv \alpha\varepsilon \equiv \alpha$ oder $\varnothing\alpha \equiv \alpha\varnothing \equiv \varnothing$.

Reguläre Ausdrücke bilden neben regulären Grammatiken und den bereits besprochenen Varianten endlicher Automaten einen weiteren Formalismus für reguläre Sprachen. Wir werden sehen, dass die Klasse der regulären Sprachen genau mit der Klasse der Sprachen $\mathcal{L}(\alpha)$ für einen

regulären Ausdruck α übereinstimmt. Diese Aussage weisen wir nach, indem wir spracherhaltende Transformationen von reguläre Ausdrücken zu endlichen Automaten und umgekehrt angeben.

Wir geben zwei Verfahren an, wie man zu gegebenem regulären Ausdruck α einen ε-NFA konstruieren kann, der genau die Sprache $\mathcal{L}(\alpha)$ akzeptiert. Diese Verfahren liefern den Beweis für folgendes Lemma.

Lemma 2.27 (Regulärer Ausdruck $\rightsquigarrow$ ε-NFA). *Zu jedem regulären Ausdruck α gibt es einen ε-NFA $\mathcal{M}$ mit $\mathcal{L}(\alpha) = \mathcal{L}(\mathcal{M})$.*

1. Verfahren "Regulärer Ausdruck $\rightsquigarrow$ NFA". Das erste Verfahren beruht auf einem kompositionellen Ansatz, der die Operatoren Vereinigung ($+$), Konkatenation und Kleeneabschluss regulärer Ausdrücke durch die entsprechenden Operatoren auf ε-NFA realisiert. Für $\alpha = \varnothing$, ε oder $a \in \Sigma$ geben wir explizit einen NFA $\mathcal{M}_\alpha$ mit $\mathcal{L}(\mathcal{M}_\alpha) = \mathcal{L}(\alpha)$ an.

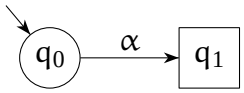
- Für $\alpha = \varnothing$ können wir einen beliebigen NFA mit leerer Endzustandsmenge verwenden, etwa bestehend aus einem Anfangszustand, der keine Transitionen hat und nicht final ist. (Alternativ könnte man sogar den NFA mit leerer Zustandsmenge verwenden.)
- Für $\alpha = \varepsilon$ verwenden wir einen NFA, der aus einem Zustand q_0 besteht. Dieser ist zugleich Anfangs- und Endzustand und hat keine ausgehenden Transitionen.
- Ist $\alpha = a \in \Sigma$, so verwenden wir einen NFA $\mathcal{M}_a$ mit zwei Zuständen q_0, q_1 und der Transition
$$q_0 \xrightarrow{a} q_1.$$
Es gibt keine weiteren Transitionen in $\mathcal{M}_a$. Zustand q_0 wird als Anfangszustand, q_1 als Endzustand deklariert.

Ist α von der Form $\beta\gamma$ oder $\beta + \gamma$ oder β^*, dann konstruieren wir zuerst (rekursiv) ε-NFA $\mathcal{M}_\beta$ bzw. $\mathcal{M}_\gamma$ für die Teilausdrücke β und γ und wenden dann den entsprechenden Operator für ε-NFA an. Siehe Abschnitt 2.2 auf Seite 35 ff.

Die Größe des resultierenden ε-NFA $\mathcal{M}_\alpha$ gemessen an der Anzahl an Zuständen ist offenbar linear in der Länge des gegebenen regulären Ausdrucks α. Beachte, dass die NFA für atomare reguläre Ausdrücke $\varnothing$, ε oder $a \in \Sigma$ höchstens zwei Zustände haben und dass die Operatoren Konkatenation $\mathcal{M}_1 \circ \mathcal{M}_2$ und Vereinigung $\mathcal{M}_1 \uplus \mathcal{M}_2$ einen ε-NFA generieren, dessen Anzahl an Zuständen gleich $|\mathcal{M}_1| + |\mathcal{M}_2|$ ist. Für den Kleeneabschluss gilt $|\mathcal{M}^*| = |\mathcal{M}| + 1$. Durch strukturelle Induktion kann dann gezeigt werden, dass $|\mathcal{M}_\alpha| \leqslant |\alpha| + 1$.

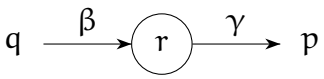
2. Verfahren "Regulärer Ausdruck $\rightsquigarrow$ NFA". Eine weitere Konstruktionsmöglichkeit für die Konstruktion eines ε-NFA besteht darin, den Automaten schrittweise durch sukzessives Einfügen von Zuständen und Kanten aufzubauen. Den Spezialfall $\alpha \equiv \varnothing$ können wir ausklammern, da die akzeptierte Sprache eines NFA ohne Endzustände (d.h. mit leerer Endzustandsmenge) offenbar leer ist. Im Folgenden setzen wir also $\alpha \not\equiv \varnothing$ voraus. Beginnend mit einem gerichteten Graphen bestehend aus zwei Zuständen, die über eine mit α beschriftete Kante miteinander verbunden sind, fügen wir sukzessive neue Zustände ein. Die Transitionen sind zunächst mit

Initialisierung:

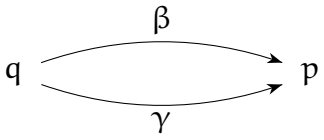


Iteration: solange möglich, wende eine der drei Regeln an:

Regel 1: generiere neuen Zustand r und ersetze $q \xrightarrow{\beta\gamma} p$ durch



Regel 2: ersetze $q \xrightarrow{\beta+\gamma} p$ durch



Regel 3: generiere neuen Zustand r und ersetze $q \xrightarrow{\beta^*} p$ durch

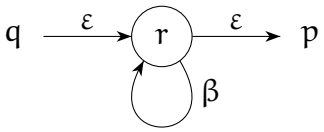


Abbildung 19: Zweites Verfahren "regulärer Ausdruck $\rightsquigarrow$ ε-NFA"

regulären Ausdrücken beschriftet. Diese werden sukzessive durch echte Teilausdrücke ersetzt bis alle Kanten mit ε oder einem Terminalzeichen $a \in \Sigma$ beschriftet sind. Die präzise Vorgehensweise ist in Abbildung 19 angegeben. Der resultierende Graph ist ein ε-NFA, der genau die Sprache $\mathcal{L}(\alpha)$ akzeptiert. Wie für das erste Verfahren kann auch hier garantiert werden, dass die Größe des konstruierten NFA linear in der Länge des gegebenen regulären Ausdrucks ist.

Die bisherigen Ergebnisse belegen noch nicht, dass jede reguläre Sprache durch einen regulären Ausdruck beschrieben werden kann. Wir zeigen nun, dass zu jedem endlichen Automaten $\mathcal{M}$ ein regulärer Ausdruck α konstruiert werden kann, so dass $\mathcal{L}(\alpha) = \mathcal{L}(\mathcal{M})$.

Lemma 2.28 (NFA $\rightsquigarrow$ regulärer Ausdruck). *Zu jedem NFA $\mathcal{M}$ gibt es einen regulären Ausdruck α mit $\mathcal{L}(\alpha) = \mathcal{L}(\mathcal{M})$.*

Auch hierzu diskutieren wir zwei Verfahren. Das erste Verfahren beruht auf einem Gleichungssystem für reguläre Ausdrücke, welche die Sprachen $L_q = \{x \in \Sigma^* : \delta(q,x) \cap F \neq \varnothing\}$ repräsentieren, und ist unter dem Stichwort "Ersetzungsmethode" bekannt. Das zweite Verfahren beruht auf der Methodik des dynamischen Programmierens zur Berechnung regulärer Ausdrücke für die Sprachen $L_{q,p}$ aller Wörter x, so dass $\delta(q,x) = p$.

Ungeachtet, welche Methode eingesetzt wird, können zunächst einige Vereinfachungen an dem gegebenen NFA vorgenommen werden, die die akzeptierte Sprache unverändert lassen, aber den Automaten verkleinern. Z.B. können alle Zustände q entfernt werden, von denen kein Endzustand erreichbar ist. Eine derartige Vereinfachung des NFA lässt sich mit einer Tiefen- oder Breiten-Rückwärtssuche, gestartet mit den F-Zuständen, realisieren. Mit Rückwärtssuche ist hier gemeint, dass mit den Endzuständen beginnend nach einem Anfangszustand entlang der

Kanten (p,q), falls $p \in \bigcup_{a\in\Sigma} \delta(q,a)$, gesucht wird. Ebenso können alle unerreichbaren Zustände (also Zustände, die von keinem Anfangszustand erreichbar sind) eliminiert werden. Im Folgenden können wir also davon ausgehen, dass die Endzustandsmenge F von jedem Zustand erreichbar ist und dass alle Zustände in Q erreichbar sind.

1. Verfahren "NFA $\rightsquigarrow$ regulärer Ausdruck": Ersetzungsmethode. Im Folgenden sei $\mathcal{M} = (Q, \Sigma, \delta, Q_0, F)$ ein NFA, von dem vorausgesetzt wird, dass alle Zustände von wenigstens einem Anfangszustand $q_0 \in Q_0$ erreichbar sind und dass die Endzustandsmenge F von jedem Zustand erreichbar ist. Das Ziel ist die Konstruktion eines regulären Ausdrucks für die durch $\mathcal{M}$ akzeptierte Sprache $\mathcal{L}(\mathcal{M}) \subseteq \Sigma^*$. Die Ersetzungsmethode beruht auf der Idee, jedem Zustand q einen regulären Ausdruck α_q zuzuordnen, welcher für die Sprache

$$L_q \stackrel{\text{def}}{=} \{w \in \Sigma^* : \delta(q,w) \cap F \neq \varnothing\}$$

aller Wörter $w \in \Sigma^*$ steht, die einen in Zustand q beginnenden, akzeptierenden Lauf im Automaten haben. Das Ziel ist also die Konstruktion regulärer Ausdrücke α_q für alle Zustände $q \in Q$, so dass $\mathcal{L}(\alpha_q) = L_q$. Wegen $\mathcal{L}(\mathcal{M}) = \bigcup_{q\in Q_0} L_q$ erhält man dann einen gesuchten regulären Ausdruck für $\mathcal{M}$, indem man die Summe der regulären Ausdrücke α_q für alle Anfangszustände betrachtet. Ist also etwa $Q_0 = \{q_1, \ldots, q_k\}$, so ist

$$\alpha \;=\; \alpha_{q_1} + \ldots + \alpha_{q_k}$$

ein regulärer Ausdruck mit $\mathcal{L}(\alpha) = \mathcal{L}(\mathcal{M})$.

Zunächst kann man ein Gleichungssystem für die α_q's hinschreiben, welches sich aus der Übergangsrelation δ ergibt. Im Folgenden gehen wir von einer beliebigen, aber festen Nummerierung der Zustände aus, etwa $Q = \{q_1, \ldots, q_n\}$, und schreiben kurz α_i für α_{q_i}.

Das *Gleichungssystem* für die regulären Ausdrücke ist nun wie folgt:

- Ist $q_i \notin F$, so ist
$$\alpha_i \;\equiv\; \sum_{a\in\Sigma} \sum_{r\in\delta(q_i,a)} a\alpha_r$$
- Ist q ein Endzustand, so ist der oben angegebene Ausdruck um "$+\varepsilon$" zu erweitern. D.h., für $q_i \in F$ ist
$$\alpha_i \;\equiv\; \sum_{a\in\Sigma} \sum_{r\in\delta(q_i,a)} a\alpha_r \;+\; \varepsilon$$

Dabei ist $\sum_{a\in\Sigma} \sum_{r\in\delta(q_i,a)} a\alpha_r \;=\; \varnothing$, falls $\delta(q_i,a) = \varnothing$ für alle $a \in \Sigma$.

Das Gleichungssystem für die α_i's kann nun gelöst werden, indem man sukzessive folgende *Ersetzungregel* einsetzt:

$$\text{"Aus } \alpha \;\equiv\; \beta\alpha + \gamma \text{ und } \varepsilon \notin \mathcal{L}(\beta) \text{ folgt } \alpha \equiv \beta^*\gamma\text{."}$$

Zusätzlich benötigt man einige Äquivalenzregeln für reguläre Ausdrücke, etwa die beiden Äquivalenzgesetze

$$\beta_1\gamma+\ldots+\beta_k\gamma \equiv (\beta_1+\ldots+\beta_k)\gamma \quad \text{und} \quad \gamma\beta_1+\ldots+\gamma\beta_k \equiv \gamma(\beta_1+\ldots+\beta_k)$$

und einige Äquivalenzgesetze für ε und $\varnothing$ (wie z.B. $\beta+\varnothing \equiv \beta$ oder $\beta\varnothing \equiv \varnothing$). Man beachte den Sonderfall $\gamma \equiv \varnothing$, für den sich die angegebene Regel durch

$$\text{“Aus } \alpha \equiv \beta\alpha \text{ und } \varepsilon \notin \mathcal{L}(\beta) \text{ folgt } \alpha \equiv \varnothing\text{”}$$

ersetzen lässt, da $\beta^*\varnothing \equiv \varnothing$. Für den Sonderfall $\gamma = \varepsilon$ ergibt sich

$$\text{“Aus } \alpha \equiv \beta\alpha + \varepsilon \text{ und } \varepsilon \notin \mathcal{L}(\beta) \text{ folgt } \alpha \equiv \beta^*\text{”}$$

Zunächst zur Korrektheit der angegebenen Regel. Hierzu formulieren wir die Regel auf Ebene formaler Sprachen:

Lemma 2.29 (Korrektheit der Ersetzungsregel). *Seien* $L, K, H \subseteq \Sigma^*$*, so dass* $L = KL \cup H$ *und* $\varepsilon \notin K$*. Dann gilt* $L = K^*H$.

Beweis. “$\subseteq$”: Durch Induktion nach n zeigen wir, dass

$$\left\{w \in L : |w| \leqslant n\right\} \subseteq K^*H.$$

Induktionsanfang $n = 0$: falls $\varepsilon \in L = KL \cup H$, so ist $\varepsilon \in H$ (da $\varepsilon \notin K$) und somit $\varepsilon \in K^*H$.

Induktionsschritt $n-1 \Longrightarrow n$ $(n \geqslant 1)$: Sei $w \in L$ ein Wort der Länge n. Wegen $L = KL \cup H$ gilt $w \in KL$ oder $w \in H$. Falls $w \in H$, so ist $w = \varepsilon w \in K^*H$. Wir nehmen nun an, dass $w \in KL$, etwa

$$w = xy, \text{ wobei } x \in K \text{ und } y \in L.$$

Da $\varepsilon \notin K$, ist $x \neq \varepsilon$ und somit

$$|y| = |w| - |x| \leqslant n-1.$$

Nach Induktionsvoraussetzung gilt $y \in K^*H$. Wegen $x \in K$ ist $w = xy \in K^*H$.

“$\supseteq$”: sei $w \in K^*H$, etwa $w = x_1 \ldots x_m y$, wobei $m \geqslant 0$, $x_1, \ldots, x_m \in K$ und $y \in H$. Wegen $L = KL \cup H$ gilt $H \subseteq L$ und $KL \subseteq L$. Wir erhalten:

$$\begin{array}{rr} & y \in L \quad (\text{da } y \in H) \\ \Longrightarrow & x_m y \in L \quad (\text{da } x_m \in K \text{ und } y \in L) \\ \Longrightarrow & x_{m-1}x_m y \in L \quad (\text{da } x_{m-1} \in K \text{ und } x_m y \in L) \\ \vdots & \\ \Longrightarrow & x_1 \ldots x_{m-1}x_m y \in L \quad (\text{da } x_1 \in K \text{ und } x_2 \ldots x_{m-1}x_m y \in L) \end{array}$$

und somit $w = x_1 \ldots x_{m-1}x_m y \in L$. □

Beispiel 2.30 (Ersetzungsmethode). Wir betrachten den DFA $\mathcal{M} = (\{q_0, q_1, q_2\}, \{a, b, c\}, \delta, q_0, \{q_1\})$ aus Abbildung 20. Das Ziel ist es, jedem der drei Zustände q_i einen regulären Ausdruck α_i für die Sprache L_i zuzuordnen, wobei L_i für die Sprache bestehend aus allen Wörtern w mit $\delta(q_i, w) = q_1$ steht. Für die akzeptierte Sprache von $\mathcal{M}$ gilt dann $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\alpha_0)$.

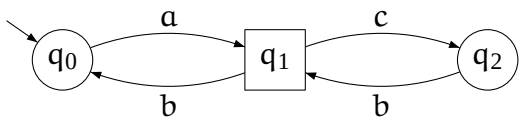


Abbildung 20: DFA $\mathcal{M}$ aus Beispiel 2.30

Zunächst erstellen wir ein Gleichungssystem für die α_i's:

$$\begin{aligned}\alpha_0 &\equiv a\alpha_1\\ \alpha_1 &\equiv b\alpha_0 + c\alpha_2 + \varepsilon\\ \alpha_2 &\equiv b\alpha_1\end{aligned}$$

Wir setzen nun die Gleichung für α_0 in die Gleichung für α_1 ein und erhalten:

$$\alpha_1 \equiv \underbrace{ba}_{\beta}\alpha_1 + \underbrace{c\alpha_2 + \varepsilon}_{\gamma}$$

Anwenden der oben genannten Regel ergibt:

$$\alpha_1 \equiv (ba)^*(c\alpha_2 + \varepsilon)$$

Einsetzen in die Gleichung für α_2 liefert:

$$\alpha_2 \equiv b(ba)^*(c\alpha_2 + \varepsilon) \equiv b(ba)^*c\alpha_2 + b(ba)^*$$

Durch Anwenden der genannten Regel mit $\beta = b(ba)^*c$ und $\gamma = b(ba)^*$ erhalten wir:

$$\alpha_2 \equiv (b(ba)^*c)^* b(ba)^*$$

Einsetzen der Lösung für α_2 in die Gleichung für α_1 ergibt.

$$\alpha_1 \equiv b\alpha_0 + c(b(ba)^*c)^* b(ba)^* + \varepsilon$$

Durch Einsetzen von α_1 in α_0 und Anwendung der Ersetzungsregel erhält man den gesuchten regulären Ausdruck für α_0

$$\alpha_0 \equiv (ab)^*\big(ac((b(ba)^*c)^* b(ba)^*) + a\big)$$

Selbstverständlich gibt es hier für das Ersetzen viele Möglichkeiten, die letztendlich zu verschiedenen (aber äquivalenten) Ausdrücken führen können. ■

Beispiel 2.31 (Ersetzungsmethode). Als weiteres Beispiel betrachten wir den NFA $\mathcal{M}$ aus Abbildung 21 mit dem Alphabet $\Sigma = \{a, b, c, d\}$. Wie zuvor suchen wir reguläre Ausdrücke α_i für die Sprache bestehend aus allen Wörtern w, so dass $q_1 \in \delta(q_i, w)$. Die akzeptierte Sprache von $\mathcal{M}$ ist dann durch den regulären Ausdruck $\alpha_1 + \alpha_2$ gegeben, da q_1 und q_2 Anfangszustände von $\mathcal{M}$ sind.

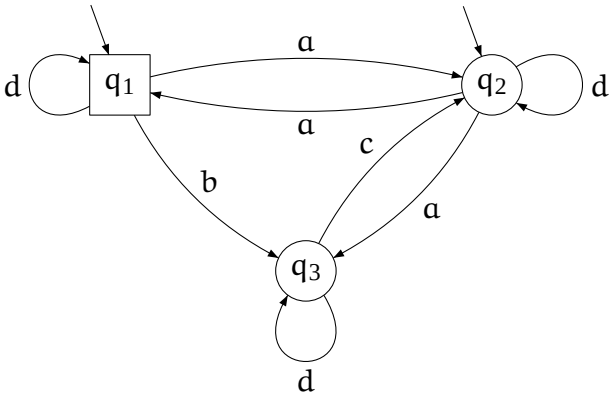


Abbildung 21: NFA $\mathcal{M}$ aus Beispiel 2.31

Das induzierte Gleichungssystem für die α_i's lautet:

$$\begin{aligned}
\alpha_1 &\equiv d\alpha_1 + a\alpha_2 + b\alpha_3 + \varepsilon \\
\alpha_2 &\equiv a\alpha_1 + d\alpha_2 + a\alpha_3 \\
\alpha_3 &\equiv c\alpha_2 + d\alpha_3
\end{aligned}$$

Aufgrund der rekursiven Gestalt des Gleichungssystems führt das Einsetzen der Geichungen für α_i in die Gleichungen der beiden anderen α_j's hier nicht zur Elimination von α_i. Dennoch können wir sukzessive die Regel "aus $\alpha \equiv \beta\alpha + \gamma$, $\varepsilon \notin \mathcal{L}(\beta)$ folgt $\alpha \equiv \beta^*\gamma$" anwenden. Z.B. kann man aus $\alpha_3 \equiv c\alpha_2 + d\alpha_3$ auf

$$\alpha_3 \equiv d^*c\alpha_2$$

schließen. Einsetzen in die Gleichung für α_1 liefert:

$$\begin{aligned}
\alpha_1 &\equiv d\alpha_1 + a\alpha_2 + b\alpha_3 + \varepsilon \\
&\equiv d\alpha_1 + a\alpha_2 + bd^*c\alpha_2 + \varepsilon \\
&\equiv d\alpha_1 + (a + bd^*c)\alpha_2 + \varepsilon
\end{aligned}$$

und somit

$$\begin{aligned}
\alpha_1 &\equiv d^*\big((a + bd^*c)\alpha_2 + \varepsilon\big) \\
&\equiv d^*(a + bd^*c)\alpha_2 + d^*
\end{aligned}$$

Diese Darstellungen von α_1 und α_3 in Abhängigkeit von α_2 können nun in die Gleichung für α_2 eingesetzt werden. Wir erhalten:

$$\begin{aligned}
\alpha_2 &\equiv a\alpha_1 + d\alpha_2 + a\alpha_3 \\
&\equiv a(d^*(a + bd^*c)\alpha_2 + d^*) + d\alpha_2 + ad^*c\alpha_2 \\
&\equiv ad^*(a + bd^*c)\alpha_2 + d\alpha_2 + ad^*c\alpha_2 + ad^* \\
&\equiv (\,ad^*(a + bd^*c) + d + ad^*c\,)\alpha_2 + ad^*
\end{aligned}$$

und somit eine Lösung für α_2:

$$\alpha_2 \equiv (\,ad^*(a + bd^*c) + d + ad^*c\,)^*ad^*$$

Lösungen für α_1 und α_3 ergeben sich nun durch Einsetzen dieser Lösung für α_2 in die oben genannten Darstellungen von α_1 und α_3 in Abhängigkeit von α_2. ■

2. Verfahren "NFA $\rightsquigarrow$ regulärer Ausdruck": dynamisches Programmieren. Wie zuvor sei $\mathcal{M} = (Q, \Sigma, \delta, Q_0, F)$ ein NFA. Für $q, p \in Q$ definieren wir

$$L_{q,p} \stackrel{\text{def}}{=} \{ w \in \Sigma^* : p \in \delta(q, w) \} = \{ w \in \Sigma^* : q \xrightarrow{w} p \}$$

Offenbar gilt $L_{q,p} = \mathcal{L}(\mathcal{M}_{q,p})$, wobei $\mathcal{M}_{q,p}$ der NFA ist, der aus $\mathcal{M}$ entsteht, indem q als einziger Anfangszustand und p als einziger Endzustand deklariert wird, d.h., $\mathcal{M}_{q,p} = (Q, \Sigma, \delta, \{q\}, \{p\})$. Daher ist die Sprache $L_{q,p}$ regulär. Das Ziel ist es nun, reguläre Ausdrücke $\alpha[q,p]$ für alle Zustände $q, p \in Q$ anzugeben, so dass $\mathcal{L}(\alpha[q,p]) = L_{q,p}$. Liegen diese regulären Ausdrücke $\alpha[q,p]$ vor, dann erhält man einen regulären Ausdruck α für die Sprache $\mathcal{L}(\mathcal{M})$ wie folgt:

$$\alpha \stackrel{\text{def}}{=} \sum_{q \in Q_0} \sum_{p \in F} \alpha[q,p]$$

Tatsächlich ist α dann ein regulärer Ausdruck für die von $\mathcal{M}$ akzeptierte Sprache, da nämlich:

$$\mathcal{L}(\alpha) = \bigcup_{q \in Q_0} \bigcup_{p \in F} \mathcal{L}(\alpha[q,p]) = \bigcup_{q \in Q_0} \bigcup_{p \in F} L_{q,p} = \mathcal{L}(\mathcal{M})$$

Im Folgenden gehen wir von einer beliebigen, aber festen Nummerierung der Zustände von $\mathcal{M}$ aus, etwa

$$Q = \{ q_1, \ldots, q_n \},$$

wobei $q_1, \ldots, q_n$ paarweise verschieden sind. Zur Vereinfachung identifizieren wir im Folgenden die Zustände mit ihren Nummern und schreiben $L_{i,j}$ für L_{q_i,q_j} und $\alpha[i,j]$ statt $\alpha[q_i, q_j]$.

Für $k \in \{0, 1, \ldots, n\}$ und $i, j \in \{1, \ldots, n\}$ sei

$$L^k_{i,j} \stackrel{\text{def}}{=} \{ w \in \Sigma^* : \begin{array}{l} \text{es gibt einen Lauf für } w \text{ der Form } q_i\, p_1 \ldots p_m\, q_j \\ \text{mit } m \geqslant 0 \text{ und } \{p_1, p_2, \ldots, p_m\} \subseteq \{q_1, \ldots, q_k\} \end{array} \}$$

Der Sonderfall $m = 0$ bedarf einer Erklärung. Ist $i = j$, so sind für $m = 0$ alle Läufe der Länge 0 und 1 zugelassen. Hierfür gibt es nur zwei Kandidaten, nämlich den Lauf q_i für das leere Wort und den Lauf $q_i\, q_i$ für alle Wörter a, für die es eine a-Schleife an Zustand q_i gibt. Im Sonderfall $m = 0$ und $i \neq j$ ist nur der Lauf $q_i\, q_j$ der Länge 1 zugelassen. Dieser ist ein Lauf für alle Wörter a, für die es eine a-Transition von q_i nach q_j gibt. [6]

Die Sprache $L^k_{i,j}$ besteht also aus allen Wörtern w, für die es einen Lauf gibt, der das Wort w vollständig liest und in Zustand q_i beginnt, in Zustand q_j endet und der – eventuell abgesehen vom ersten Zustand q_i und letzten Zustand q_j – höchstens die ersten k Zustände $q_1, \ldots, q_k$ besucht. Ist nun $q = q_i$ und $p = q_j$, so ist

$$L_{q_i,q_j} = L_{i,j} = L^n_{i,j},$$

da in der Definition von $L^k_{i,j}$ für $k = n$ alle Läufe von $q = q_i$ nach $p = q_j$ betrachtet werden.

[6] Eine technische Bemerkung zu dem Begriff "Lauf". Läufe für NFA wurden als Zustandsfolgen definiert, die in einem Anfangszustand beginnen. Der Begriff eines Laufs bezieht sich hier also streng genommen nicht auf $\mathcal{M}$, sondern auf den NFA $\mathcal{M}' = (Q, \Sigma, \delta, \{q_i\}, F)$ mit eindeutigem Anfangszustand q_i.

Die Idee ist nun, durch Induktion nach k reguläre Ausdrücke für die Sprachen $L^k_{i,j}$ anzugeben. Wir betrachten zuerst den Fall $k = 0$ und $i \neq j$. Die Sprache $L^0_{i,j}$ besteht nur aus Wörtern der Länge 1, nämlich

$$L^0_{i,j} \;=\; \big\{a \in \Sigma : q_i \xrightarrow{a} q_j\big\}.$$

Ist also $\{a \in \Sigma : q_j \in \delta(q_i, a)\} = \{a_1, a_2, \ldots, a_r\}$, so ist

$$\alpha^0[i,j] \;\stackrel{\text{def}}{=}\; a_1 + a_2 + \ldots + a_r$$

ein regulärer Ausdruck mit $\mathcal{L}(\alpha^0[i,j]) = L^0_{i,j}$, wobei $i \neq j$. Falls $r = 0$, d.h., falls $\{a \in \Sigma : q_j \in \delta(q_i, a)\} = \varnothing$, so steht $a_1 + \ldots + a_r$ für den regulären Ausdruck $\varnothing$. Für den Sonderfall $i = j$ ist das leere Wort zu berücksichtigen. Dieses liegt stets in $L^0_{i,i}$. Ist also $\{a \in \Sigma : q_i \in \delta(q_i, a)\} = \{a_1, a_2, \ldots, a_r\}$, so ist

$$\alpha^0[i,i] \;\stackrel{\text{def}}{=}\; a_1 + a_2 + \ldots + a_r + \varepsilon$$

ein regulärer Ausdruck mit $\mathcal{L}(\alpha^0[i,i]) = L^0_{i,i}$.

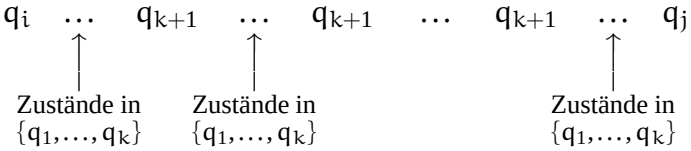
Wir nehmen nun an, dass $k \in \{0, 1, \ldots, n-1\}$ und dass reguläre Ausdrücke $\alpha^k[i,j]$ mit $\mathcal{L}(\alpha^k[i,j]) = L^k_{i,j}$ für alle i, j bereits vorliegen. Wir leiten nun aus den Ausdrücken $\alpha^k[\ldots]$ reguläre Ausdrücke $\alpha^{k+1}[i,j]$ für alle Zustandsnummern i, j her. Gedanklich teilen wir die in der Definition von $L^{k+1}_{i,j}$ vorkommenden Läufe $q_i\, p_1 \ldots p_m\, q_j$ in zwei Gruppen ein:

- diejenigen Läufe, die nicht durch den $(k+1)$-ten Zustand q_{k+1} laufen (eventuell abgesehen von dem ersten bzw. letzten Zustand, sofern $k+1 = i$ oder $k+1 = j$), d.h.,

  $$\{p_1, \ldots, p_m\} \subseteq \{q_1, \ldots, q_k\}.$$

  Diese Läufe $q_i\, p_1 \ldots p_m\, q_j$ sind bereits in der Definition von $L^k_{i,j}$ erfasst.

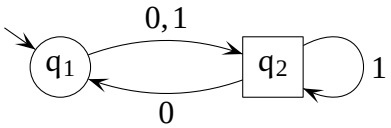
- diejenigen Läufe, die mindestens einmal durch den Zustand q_{k+1} führen. In diesem Fall kann $q_i\, p_1 \ldots p_m\, q_j$ in Fragmente zerlegt werden:



Daher gilt $L^{k+1}_{i,j} \;=\; L^k_{i,j} \cup L^k_{i,k+1}\big(L^k_{k+1,k+1}\big)^* L^k_{k+1,j}$ und wir können $\alpha^{k+1}[i,j]$ wie folgt definieren:

$$\alpha^{k+1}[i,j] \;\stackrel{\text{def}}{=}\; \alpha^k[i,j] \;+\; \alpha^k[i,k+1]\big(\alpha^k[k+1,k+1]\big)^*\alpha^k[k+1,j].$$

Beispiel 2.32 (NFA $\rightsquigarrow$ regulärer Ausdruck). Wir veranschaulichen die angegebene Methode exemplarisch an folgendem Automaten $\mathcal{M}$.



Da $\mathcal{M}$ nur einen Anfangszustand (nämlich q_1) und einen Endzustand (nämlich q_2) hat, ist $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\alpha^2[1,2])$. Wir behandeln zunächst den Fall $k = 0$:

$$\begin{array}{lcllcl} \alpha^0[1,1] & = & \varepsilon & \alpha^0[1,2] & = & 0+1 \\ \alpha^0[2,1] & = & 0 & \alpha^0[2,2] & = & 1+\varepsilon \end{array}$$

Für $k = 1$ erhalten wir z.B.

$$\alpha^1[1,1] \;=\; \alpha^0[1,1] \,+\, \alpha^0[1,1]\,\big(\alpha^0[1,1]\big)^*\,\alpha^0[1,1] \;=\; \varepsilon \,+\, \varepsilon\,\varepsilon^*\,\varepsilon$$

und somit $\alpha^1[1,1] \equiv \varepsilon$. Insgesamt erhalten wir für $k = 1$ folgende Ausdrücke, die zu den Ausdrücken $\alpha^1[i,j]$ äquivalent sind:

$$\begin{array}{lcllcl} \alpha^1[1,1] & \equiv & \varepsilon & \alpha^1[1,2] & \equiv & 0+1 \\ \alpha^1[2,1] & \equiv & 0 & \alpha^1[2,2] & \equiv & 1+\varepsilon+00+01 \end{array}$$

Hieraus ergeben sich die regulären Ausdrücke für $k = 2$:

$$\begin{array}{rcl} \alpha^2[1,1] & = & \alpha^1[1,1] \,+\, \alpha^1[1,2]\,\big(\alpha^1[2,2]\big)^*\,\alpha^1[2,1] \\ & \equiv & \varepsilon \,+\, (0+1)(\varepsilon+1+00+01)^*0 \\ \alpha^2[1,2] & = & \alpha^1[1,2] \,+\, \alpha^1[1,2]\,\big(\alpha^1[2,2]\big)^*\,\alpha^1[2,2] \\ & \equiv & 0+1+(0+1)(\varepsilon+1+00+01)^*(\varepsilon+1+00+01) \end{array}$$

Analog können $\alpha^2[2,1]$ und $\alpha^2[2,2]$ (bzw. reguläre Ausdrücke, die zu $\alpha^2[2,1]$ und $\alpha^2[2,2]$ äquivalent sind) berechnet werden. ■

Wie das obige Beispiel zeigt, können die regulären Ausdrücke $\alpha^k[i,j]$ extrem lang werden. Tatsächlich ist die Länge $\alpha^k[i,j]$ exponentiell in k, sofern keine syntaktischen Vereinfachungen an den Zwischenergebnissen vorgenommen werden. Dies erklärt sich daraus, dass die Ausdrücke $\alpha^k[\ldots]$ jeweils vier Teilausdrücke $\alpha^{k-1}[\ldots]$ enthalten. Daher ist eine untere Schranke für die minimale Länge $T(k)$ der Ausdrücke $\alpha^k[i,j]$ durch die Rekurrenz

$$T(0) \geqslant 1, \qquad T(k+1) \;\geqslant\; 4T(k)+1 \;\text{ für } k \geqslant 0$$

gegeben. Durch Induktion nach k kann nun gezeigt werden, dass $T(k) \geqslant 4^k$. Wir zeigen hier kurz, wie man – ohne Raten – zu der Lösung $T(k) \geqslant 4^k$ der obigen Rekurrenz gelangt. Hierzu betrachten wir ein hinreichend großes k und setzen sukzessive die Ungleichungen für $T(k)$, $T(k-1)$, etc. ein:

$$\begin{array}{rcll} T(k) & \geqslant & 4\;T(k-1)+1 & \geqslant\; 4\;\cdot(4T(k-2)+1)+1 \\ & \geqslant & 4^2T(k-2)+4+1 & \geqslant\; 4^2\cdot(4T(k-3)+1)+4+1 \\ & \geqslant & 4^3T(k-3)+4^2+4+1 & \geqslant\; 4^4\cdot(4T(k-4)+1)+4^2+4+1 \\ & \vdots & & \\ & \geqslant & 4^\ell T(k-\ell)+4^{\ell-1}+4^{\ell-2}+\ldots+4^2+4+1 & \end{array}$$

Mit $k = \ell$ und der Ungleichung $T(0) \geqslant 1$ erhalten wir eine Teilsumme der geometrischen Reihe als untere Schranke für $T(k)$:

$$T(k) \geqslant 4^k+4^{k-1}+\ldots+4^2+4+1,$$

also ist $T(k) \geqslant 4^k$. In analoger Weise kann man eine obere Schranke für die Länge der regulären Ausdrücke $\alpha^k[i,j]$ berechnen. Sei $S(k)$ die maximale Länge, die die Ausdrücke $\alpha^k[i,j]$ haben können. Man kann nun die Rekurrenz $S(0) = \mathcal{O}(|\Sigma|)$ und $S(k) \leqslant 4S(k-1) + C$ verwenden, wobei C eine Konstante ist, deren exakter Wert für eine asymptotische obere Schranke von $S(k)$ unerheblich ist. Die obere Schranke $\mathcal{O}(|\Sigma|)$ für die Länge der Ausdrücke $\alpha^0[i,j]$ ergibt sich aus der Tatsache, dass der längste mögliche Ausdruck für $\alpha^0[i,j]$ die Form $a_1 + a_2 + \ldots + a_r + \varepsilon$ hat, wobei $\Sigma = \{a_1, \ldots, a_r\}$ und $r = |\Sigma|$. Die präzise Länge dieses Ausdrucks ist $(r+1) + 3r = 4r + 1$, da für jedes Summenzeichen drei Längeneinheiten zu veranschlagen sind, von denen zwei auf die unterdrückten Klammern und eine Längeneinheit auf das Zeichen $+$ entfallen. Entscheidend ist hier jedoch lediglich, dass die asymptotische Länge durch ein Vielfaches von $r = |\Sigma|$ nach oben beschränkt ist. Die Rekurrenz $S(0) = \mathcal{O}(|\Sigma|)$ und $S(k) \leqslant 4 \cdot S(k-1) + C$ hat die Lösung $S(k) = \mathcal{O}(|\Sigma| 4^k)$. Betrachtet man Σ als festes Alphabet, so kann $|\Sigma|$ wie eine Konstante behandelt werden und es folgt $S(k) = \mathcal{O}(4^k)$. Die berechneten regulären Ausdrücke haben also im besten und im schlimmsten Fall exponentielle Länge. Die Kosten für die Laufzeit des Verfahrens können daher mit

$$\sum_{k=0}^{n-1} n^2 \cdot 4^k \;=\; n^2 \cdot \sum_{k=0}^{n-1} 4^k \;=\; n^2 \cdot \left(\frac{4^n}{3} - \frac{1}{3} \right)$$

angesetzt werden. Die Laufzeit ist also asymptotisch durch $n^2 \cdot 4^n$ nach oben und unten beschränkt. Dasselbe gilt für den Platzbedarf.

2.4 Minimierung von endlichen Automaten

Die Komplementierung eines endlichen Automaten, die u.a. in einigen Algorithmen wie dem Inklusionstest benötigt wird, setzt einen deterministischen Automaten voraus. Ist der Ausgangspunkt ein nichtdeterministischer endlicher Automat, so kann dieser zwar durch die Potenzmengenkonstruktion determinisiert werden, jedoch ist der resultierende DFA dann oftmals sehr viel größer als der ursprüngliche NFA, selbst dann, wenn man alle unerreichbaren Zustände des Potenzmengen-DFA entfernt. In diesem Abschnitt stellen wir ein Verfahren vor, wie man aus einem gegebenen DFA einen äquivalenten DFA minimaler Größe konstruieren kann. Der Algorithmus beruht auf dem Satz von Myhill & Nerode, der eine weitere Charakterisierung regulärer Sprachen angibt und der – abgesehen von der Minimierung von DFA – auch für andere Zwecke nützlich ist. So kann der Satz von Myhill & Nerode z.B. genutzt werden, um den Nachweis zu erbringen, dass eine gegebene Sprache nicht regulär ist, oder auch um untere Schranken für kleinste DFA für gegebene reguläre Sprache anzugeben.

Zunächst erinnern wir kurz an den Begriff einer *Äquivalenzrelation*. Sei X eine Menge. Eine Äquivalenzrelation auf X ist eine binäre Relation $\mathcal{R} \subseteq X \times X$, die folgende drei Eigenschaften besitzt:

Reflexivität:	$(x,x) \in \mathcal{R}$ für alle $x \in X$
Transitivität:	aus $(x,y) \in \mathcal{R}$ und $(y,z) \in \mathcal{R}$ folgt $(x,z) \in \mathcal{R}$
Symmetrie:	aus $(x,y) \in \mathcal{R}$ folgt $(y,x) \in \mathcal{R}$.

Für Äquivalenzrelationen wird oftmals eine Infixschreibweise benutzt, also $x \mathcal{R} y$ für $(x,y) \in \mathcal{R}$. Die Äquivalenzklasse eines Elements $x \in X$ ist $[x]_{\mathcal{R}} = \{y \in X : x \mathcal{R} y\}$. Es gilt $x \mathcal{R} y$ genau dann,

wenn $[x]_{\mathcal{R}} = [y]_{\mathcal{R}}$ und genau dann, wenn $[x]_{\mathcal{R}} \cap [y]_{\mathcal{R}} \neq \varnothing$. Aus der Definition ist ersichtlich, dass eine Äquivalenzklasse immer nicht-leer ist. $X/\mathcal{R} = \big\{[x]_{\mathcal{R}} : x \in X\big\}$ bezeichnet den Quotientenraum, d.h., die Menge aller Äquivalenzklassen. Der *Index* von $\mathcal{R}$ bezeichnet die Anzahl an Äquivalenzklassen (möglicherweise ∞). Sind $\mathcal{R}_1$ und $\mathcal{R}_2$ Äquivalenzrelationen auf X mit $\mathcal{R}_1 \subseteq \mathcal{R}_2$, so wird $\mathcal{R}_1$ feiner als $\mathcal{R}_2$ und $\mathcal{R}_2$ gröber als $\mathcal{R}_1$ genannt.

Definition 2.33 (Nerode-Äquivalenz $\sim_L$, Nerode-Index). Sei $L \subseteq \Sigma^*$ eine Sprache. Die Nerode-äquivalenz für L bezeichnet diejenige Äquivalenzrelation $\sim_L$ auf Σ^*, so dass für alle Wörter x, $y \in \Sigma^*$ gilt:

$$x \sim_L y \quad \text{gdw} \quad \begin{cases} \text{für alle } z \in \Sigma^* \text{ gilt:} \\ xz \in L \iff yz \in L \end{cases}$$

Wir schreiben $[x]_L$ für die Äquivalenzklasse von x bzgl. $\sim_L$. Es gilt also $[x]_L = \{y \in \Sigma^* : x \sim_L y\}$. Σ^*/L bezeichnet den Quotientenraum unter $\sim_L$. $[x]_L$ und Σ^*/L sind also lediglich eine Kurzschreibweisen für $[x]_{\sim_L}$ bzw. $\Sigma^*/\sim_L$. Der *Nerode-Index* von L bezeichnet den Index von $\sim_L$, also die Anzahl an Elementen im Quotientenraum Σ^*/L. ■

Man überzeugt sich leicht davon, dass $\sim_L$ tasächlich eine Äquivalenzrelation ist, also reflexiv, transitiv und symmetrisch.

Beispiel 2.34 (Nerode-Äquivalenz $\sim_L$). Wir betrachten die reguläre Sprache $L = \mathcal{L}(0^*1^*)$. Die Wörter 0^n mit $n \geqslant 0$ sind paarweise $\sim_L$-äquivalent, da $0^n z \in L$ genau dann, wenn $z \in \mathcal{L}(0^*1^*)$:

$$\varepsilon \sim_L 0 \sim_L 00 \sim_L 000 \sim_L \ldots,$$

Sämtliche Wörter $x \in \mathcal{L}(0^*)$ liegen also in derselben Äquivalenzklasse. Diese ist $\mathcal{L}(0^*)$. Entsprechend sind alle Wörter der Form 0^n1^m mit $n \geqslant 0$ und $m \geqslant 1$ paarweise $\sim_L$-äquivalent, da da $0^n1^m z \in L$ genau dann gilt, wenn $z \in \mathcal{L}(1^*)$. Also:

$$1 \sim_L 01 \sim_L 001 \sim_L \ldots,$$

Alle anderen Wörter $x \in \mathcal{L}(0^*1^+0(0+1)^*)$, also alle Wörter, in denen eine Eins vor einer Null steht, sind paarweise Nerode-äquivalent, da $xz \notin L$ für alle Wörter $z \in \{0,1\}^*$. Der Quotientenraum bzgl. $\sim_L$ ist also

$$\Sigma^*/L = \big\{\mathcal{L}(0^*), \mathcal{L}(0^*1^+), \mathcal{L}(0^*1^+0(0+1)^*)\big\}.$$

Der Nerode-Index von L ist somit 3. ■

Satz 2.35 (Satz von Myhill & Nerode). *Sei L eine Sprache. L ist genau dann regulär, wenn der Nerode-Index von L endlich ist.*

Der Beweis von Satz 2.35 beruht auf mehreren Hilfsaussagen, die wir im Folgenden diskutieren werden. Der eigentliche Beweis wird dann auf Seite 61 geführt. Bevor wir auf die für den Satz von Myhill & Nerode zentralen Hilfsaussagen eingehen, überzeugen wir uns von der Aussage an zwei Beispielen. Für die reguläre Sprache $L = \mathcal{L}(0^*1^*)$ besteht Σ^*/L aus drei Elementen, siehe Beispiel 2.34 auf Seite 59. Der Nerode-Index von L ist also endlich (nämlich 3). Die Sprache

$$K = \{0^n1^n : n \geqslant 0\}$$

ist nicht regulär (siehe Seite 44). In der Tat sind die Wörter 0^n1^ℓ und 0^m1^k mit $0 \leqslant n \leqslant \ell$ und $0 \leqslant m \leqslant k$ und $n-\ell \neq m-k$ paarweise nicht Nerode-äquivalent bezüglich $\sim_K$. Es gilt nämlich

$$0^n1^\ell1^{n-\ell} = 0^n1^n \in K, \text{ aber } 0^m1^k1^{n-\ell} = 0^m1^{m+n-\ell} \notin K.$$

Somit ist der Quotientenraum $\{0,1\}^*/K$ und damit der Nerode-Index von K unendlich.

Der Beweis von Satz 2.35 benutzt einige Aussagen über die wie folgt definierte Äquivalenzrelation $\sim_\mathcal{M}$ für gegebenen DFA $\mathcal{M}$.

Definition 2.36 (DFA-Äquivalenz $\sim_\mathcal{M}$). Sei $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ ein DFA. Die Äquivalenzrelation $\sim_\mathcal{M} \subseteq \Sigma^* \times \Sigma^*$ ist wie folgt definiert:

$$x \sim_\mathcal{M} y \quad \text{gdw} \quad \delta(q_0, x) = \delta(q_0, y)$$

Dabei sind x, y beliebige Wörter über dem Alphabet Σ. $\Sigma^*/\mathcal{M}$ und $[x]_\mathcal{M}$ sind Kurzschreibweisen für den Quotientenraum $\Sigma^*/\sim_\mathcal{M}$ bzw. die Äquivalenzklasse $[x]_\mathcal{M}$. ■

Wir betrachten den DFA $\mathcal{M}$ aus Abbildung 22 auf Seite 60, der die Sprache $\mathcal{L}(0^*1^*)$ akzeptiert. Der Quotientenraum $\Sigma^*/\sim_\mathcal{M}$ besteht aus drei Äquivalenzklassen.

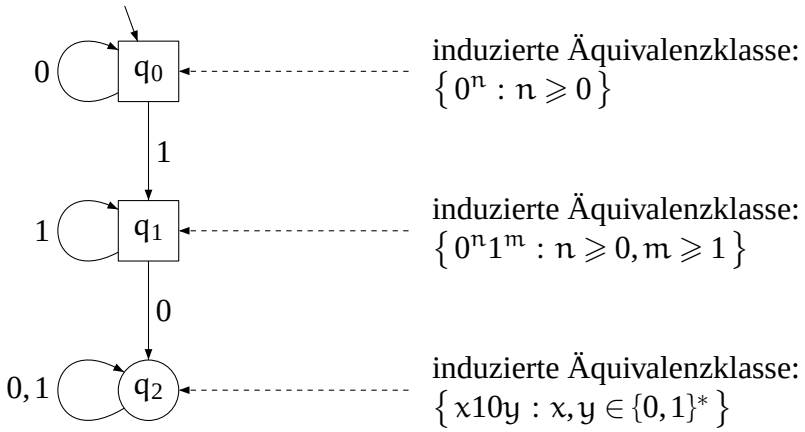


Abbildung 22: DFA $\mathcal{M}$ und die induzierten $\sim_\mathcal{M}$-Äquivalenzklassen

Lemma 2.37 (DFA-Äquivalenz hat endlichen Index). *Für jeden DFA $\mathcal{M}$ ist der Quotientenraum $\Sigma^*/\mathcal{M}$ endlich.*

Beweis. Wir betrachten zunächst einen DFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ mit totaler Übergangsfunktion. Für jeden Zustand q sind die Wörter aus der Sprache

$$K_q \stackrel{\text{def}}{=} \{x \in \Sigma^* : \delta(q_0, x) = q\}$$

paarweise äquivalent bzgl. $\sim_\mathcal{M}$. Andererseits gilt $x \not\sim_\mathcal{M} y$ für $q \neq p$, $x \in K_q$, $y \in K_p$. Somit gilt $[x]_\mathcal{M} = K_q$, falls $\delta(q_0, x) = q$. Der Quotientenraum bzgl. $\sim_\mathcal{M}$ ist daher

$$\Sigma^*/\mathcal{M} = \{K_q : q \in Q\} \setminus \{\varnothing\}.$$

Also ist $|\Sigma^*/\mathcal{M}| \leqslant |Q| < \infty$. Insbesondere ist $|\Sigma^*/\mathcal{M}| = |Q'|$, wobei Q' die Menge der von q_0 erreichbaren Zustände ist.

Für DFA mit partieller Übergangsfunktion ist die Argumentation analog, jedoch muss (gedanklich) ein zusätzlicher Fangzustand eingefügt werden, der alle Wörter x mit $\delta(q_0, x) = \perp$ repräsentiert. Es ergibt sich dann $|\Sigma^*/\mathcal{M}| \leqslant |Q|+1 < \infty$. Insbesondere ist in diesem Falle $|\Sigma^*/\mathcal{M}| = |Q'|+1$, wenn Q' die Menge der von q_0 erreichbaren Zustände ist. □

Lemma 2.38 (Zusammenhang $\sim_L$ und $\sim_{\mathcal{M}}$). *Sei $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ ein totaler DFA und $L = \mathcal{L}(\mathcal{M})$. Dann gilt:*

(a) *$\sim_{\mathcal{M}}$ ist feiner als $\sim_L$, d.h., für alle $x, y \in \Sigma^*$ mit $x \sim_{\mathcal{M}} y$ gilt $x \sim_L y$.*

(b) *$|Q| \geqslant |\Sigma^*/\mathcal{M}| \geqslant |\Sigma^*/L|$.*

Beweis. Wir zeigen zunächst Aussage (a). Es gelte $x \sim_{\mathcal{M}} y$. Weiter sei $z \in \Sigma^*$. Dann gilt:

$$\delta(q_0, xz) \;=\; \delta(\,\delta(q_0, x),\, z\,) \;=\; \delta(\,\delta(q_0, y),\, z\,) \;=\; \delta(q_0, yz)$$

Hieraus folgt:

$$\begin{aligned} xz \in L = \mathcal{L}(\mathcal{M}) \quad &\text{gdw} \quad \delta(q_0, xz) \in F \\ &\text{gdw} \quad \delta(q_0, yz) \in F \\ &\text{gdw} \quad yz \in \mathcal{L}(\mathcal{M}) = L \end{aligned}$$

Somit gilt $x \sim_L y$.

Nun zum Nachweis von Aussage (b). Da die Relation $\sim_{\mathcal{M}}$ stets eine Verfeinerung von $\sim_L$ (Aussage (a)) ist jede DFA-Äquivalenzklasse in (genau) einer Nerode-Äquivalenzklasse enthalten. Sei nun Q' die Menge aller vom Anfangszustand q_0 erreichbaren Zustände. Für $q \in Q'$ sei L_q diejenige Nerode-Äquivalenzklasse, welche die durch q induzierte DFA-Äquivalenzklasse

$$K_q \;=\; \{x \in \Sigma^* : \delta(q_0, x) = q\}$$

enthält. Also $K_q \subseteq L_q \in \Sigma^*/L$. Man beachte, dass $K_q \neq \varnothing$ für $q \in Q'$ und dass $L_q = L_p$ für $q \neq p$ möglich ist. Dann gilt:

$$\Sigma^*/L \;=\; \{L_q : q \in Q'\}.$$

Also ist $|\Sigma^*/L| \leqslant |Q'| = |\Sigma^*/\mathcal{M}| < \infty$ (siehe Lemma 2.37 auf Seite 60). □

Beweis von Satz 2.35 (Satz von Myhill & Nerode, Seite 59). Zu zeigen ist, dass eine gegebene Sprache L genau dann regulär ist, wenn der Nerode-Index von L (also die Anzahl an Äquivalenzklassen unter der Nerode-Äquivalenz $\sim_L$) endlich ist. Sei L regulär und $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ ein totaler DFA mit $\mathcal{L}(\mathcal{M}) = L$. Aus Lemma 2.38 (siehe Seite 61). folgt:

$$|\Sigma^*/L| \leqslant |\Sigma^*/\mathcal{M}| \leqslant |Q| < \infty$$

Wir nehmen nun an, dass L eine Sprache mit endlichem Nerode-Index ist. Wir zeigen, dass L regulär ist. Zunächst stellen wir fest, dass für alle $x, y \in \Sigma^*$ und $a \in \Sigma$ gilt:

(1) Aus $x \sim_L y$ folgt $xa \sim_L ya$.

Insbesondere ist $[xa]_L = [ya]_L$. Wir definieren einen DFA

$$\mathcal{M}_L \stackrel{\text{def}}{=} (Q_L, \Sigma, \delta_L, q_{0,L}, F_L),$$

dessen Zustände die Äquivalenzklassen bzgl. $\sim_L$ sind.

$$\begin{array}{lcll} Q_L & = & \Sigma^*/L & F_L = \{[x]_L : x \in L\} \\ q_{0,L} & = & [\varepsilon]_L & \delta_L([x]_L, a) = [xa]_L \end{array}$$

Aussage (1) stellt die Wohldefiniertheit der Übergangsfunktion δ_L sicher. Ferner gilt für alle $x \in \Sigma^*$:

(2) $\delta_L([\varepsilon]_L, x) = [x]_L$

(3) $x \in L$ gdw $[x]_L \in F_L$

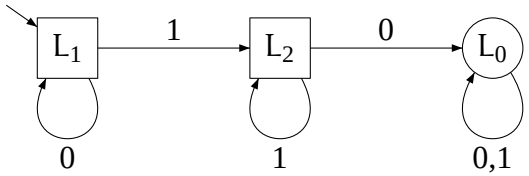
Aussage (2) kann durch Induktion nach der Länge von x gezeigt werden. Nun zum Nachweis von Aussage (3). Die Implikation "$\Longrightarrow$" ist klar, da F_L aus allen Äquivalenzklassen $[x]_L$ mit $x \in L$ besteht. Für den Nachweis der Implikation "$\Longleftarrow$" müssen wir die spezielle Form der Relation $\sim_L$ benutzen. Ist nämlich $[x]_L \in F_L$, dann gibt es ein $y \in L$, so dass $[y]_L = [x]_L$; also $x \sim_L y$. Wir betrachten nun das Wort $z = \varepsilon$ und erhalten $x = x\varepsilon = xz \in L$, da $yz = y\varepsilon = y \in L$. Damit ist Aussage (3) bewiesen. Aus (3) folgt:

$$\begin{array}{lll} x \in \mathcal{L}(\mathcal{M}_L) & \text{gdw} & \delta_L([\varepsilon]_L, x) \in F_L \\ & \text{gdw} & [x]_L \in F_L \\ & \text{gdw} & x \in L \end{array}$$

Also ist $\mathcal{L}(\mathcal{M}_L) = L$ und L regulär. ■

Definition 2.39 (Minimalautomat $\mathcal{M}_L$). Sei L eine reguläre Sprache. Der im Beweis von Satz 2.35 konstruierte DFA für L wird *Minimalautomat* von L genannt und mit $\mathcal{M}_L$ bezeichnet. ■

Beispiel 2.40 (Minimalautomat). Der Minimalautomat für die Sprache $L = \mathcal{L}(0^*1^*) = \left\{ 0^n 1^m : n, m \geqslant 0 \right\}$ hat folgende Gestalt:



Dabei ist $L_1 = \mathcal{L}(0^*)$, $L_2 = \mathcal{L}(0^*11^*)$ und $L_0 = \{0,1\}^* \setminus (L_1 \cup L_2)$. ■

Teil (b) des folgenden Satzes 2.43 zeigt, dass die Anzahl der Zustände im Minimalautomaten minimal unter allen DFA mit totaler Übergangsfunktion für die betreffende Sprache ist (was die Bezeichnung „Minimalautomat" rechtfertigt). In Teil (a) zeigen wir die Eindeutigkeit des Minimalautomaten als DFA mit totaler Übergangsfunktion, für den die induzierte Äquivalenzrelation $\sim_{\mathcal{M}}$ mit $\sim_L$ übereinstimmt. Tatsächlich haben wir nur Eindeutigkeit "bis auf Isomorphie", d.h., Gleichheit bis auf eventuelles Umbenennen der Zustandsnamen. Auch wenn der Begriff der Isomorphie intuitiv klar ist, geben wir die präzise Definition von Isomorphie an.

Definition 2.41 (Isomorphie). Zwei DFA $\mathcal{M}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, $\mathcal{M}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ heißen *isomorph*, wenn es eine bijektive Abbildung $\varphi : Q_1 \to Q_2$ gibt, so dass $q_2 = \varphi(q_1)$, $F_2 = \varphi(F_1)$ und

$$\delta_2(\varphi(q), a) = \varphi(\delta_1(q, a)) \text{ für alle } q \in Q_1 \text{ und } a \in \Sigma$$

Die Abbildung φ wird *Isomorphismus* genannt. ■

DFA $\mathcal{M}_1$ und $\mathcal{M}_2$ sind also genau dann isomorph, wenn $\mathcal{M}_1$ und $\mathcal{M}_2$ bis auf die Namen der erreichbaren Zustände übereinstimmen. Intuitiv identifizieren wir isomorphe DFA, da die Zustandsnamen völlig irrelevant für die akzeptierte Sprache sind. Offensichtlich gilt folgendes Lemma:

Lemma 2.42 (Isomorphismus erhält die akzeptierte Sprache). *Sind $\mathcal{M}_1$ und $\mathcal{M}_2$ isomorphe DFA, so gilt $\mathcal{L}(\mathcal{M}_1) = \mathcal{L}(\mathcal{M}_2)$.*

Satz 2.43 (Eindeutigkeit und Minimalität des Minimalautomaten). *Für jede reguläre Sprache L gilt:*

(a) Der Minimalautomat $\mathcal{M}_L$ für L ist der bis auf Isomorphie eindeutig bestimmte DFA mit totaler Übergangsfunktion, der folgende drei Eigenschaften besitzt:

- $\mathcal{L}(\mathcal{M}_L) = L$
- $\sim_{\mathcal{M}_L} = \sim_L$
- *Jeder Zustand ist vom Anfangszustand erreichbar.*

(b) Ist $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ ein totaler DFA mit $\mathcal{L}(\mathcal{M}) = L$, dann ist $|Q| \geqslant |\Sigma^*/L|$.

Beweis. Teil (b) folgt sofort aus Lemma 2.38 (Seite 61). Wir zeigen Aussage (a). Zunächst ist klar, dass jeder Zustand in $\mathcal{M}_L$ vom Anfangszustand erreichbar ist. Dies folgt aus der Beobachtung:

$$\text{Ist } x \in \Sigma^*, \text{ so ist } [x]_L = \delta_L([\varepsilon]_L, x).$$

Also ist jeder Zustand $[x]_L$ in $\mathcal{M}_L$ vom Anfangszustand $[\varepsilon]_L$ erreichbar. Die Aussage $\mathcal{L}(\mathcal{M}_L) = L$ wurde im Beweis von Satz 2.35 nachgewiesen. Nun zeigen wir, dass die Relationen $\sim_{\mathcal{M}_L}$ und $\sim_L$ übereinstimmen. Wegen Teil (a) von Lemma 2.38 (Seite 61) genügt es zu zeigen, dass $\sim_L$ eine Verfeinerung von $\sim_{\mathcal{M}_L}$ ist. Seien $x, y \in \Sigma^*$ und $x \sim_L y$. Es folgt:

$$\delta_L([\varepsilon]_L, x) = [x]_L = [y]_L = \delta_L([\varepsilon]_L, y)$$

Also ist $x \sim_{\mathcal{M}_L} y$. Es folgt, dass $\sim_L$ und $\sim_{\mathcal{M}_L}$ übereinstimmen.

Damit erfüllt $\mathcal{M}_L$ die drei genannten Eigenschaften.

Wir zeigen nun, dass jeder weitere DFA mit totaler Übergangsfunktion mit den genannten drei Eigenschaften zu $\mathcal{M}_L$ isomorph ist. Sei nun $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ ein weiterer totaler DFA mit $\mathcal{L}(\mathcal{M}) = L$, $\sim_{\mathcal{M}} = \sim_L$ und für den alle Zustände $q \in Q$ vom Anfangszustand q_0 erreichbar sind. Für jeden Zustand $q \in Q$ definieren wir die Sprache

$$K_q \stackrel{\text{def}}{=} \{x \in \Sigma^* : \delta(q_0, x) = q\}.$$

Dann ist K_q eine Äquivalenzklasse unter $\sim_{\mathcal{M}} = \sim_L$.[7] Also ist $K_q \in \Sigma^*/L$ ein Zustand im Minimalautomaten $\mathcal{M}_L$.

Zu zeigen ist noch, dass die Abbildung $\varphi : Q \to \Sigma^*/L$, $\varphi(q) = K_q$, ein Isomorphismus ist.

1. Bijektivität von φ.

   1.1 Surjektivität: Für jede Nerode-Äquivalenzklasse $[x]_L \in Q_L$ gilt

   $$\varphi(\delta(q_0, x)) = K_{\delta(q_0,x)} = [x]_L$$

   1.1 Injektivität: Seien $p, q \in Q$ mit $\varphi(q) = \varphi(p)$, also $K_q = K_p$. Wähle beliebiges Wort x mit $K_q = K_p = [x]_L$. Dann gilt $q = \delta(q_0, x) = p$.

2. φ ist struktur-erhaltend.

   2.1 Für den Anfangszustand gilt:

   $$\varphi(q_0) = K_{q_0} = \{x \in \Sigma^* : \delta(q_0, x) = q_0\} = [\varepsilon]_{\mathcal{M}} = [\varepsilon]_L$$

   2.2 Für jeden Zustand q in $\mathcal{M}$ gilt:

   $$q \in F \quad \text{gdw} \quad K_q \cap L \neq \varnothing \quad \text{gdw} \quad K_q \in F_L$$

   In der ersten Äquivalenz benutzen wir die Tatsache, dass für jedes Wort x mit $K_q = [x]_L$ gilt $q \in F$ genau dann, wenn $x \in L$. In der zweiten Äquivalenz benutzen wir die in der VL bewiesene Aussage, dass $[x]_L \in F_L$ genau dann, wenn $x \in L$.

   2.3 Seien $q \in Q$ und $a \in \Sigma$. Sei $p = \delta(q, a)$ und $x \in \Sigma^*$ ein Wort mit $K_q = [x]_L$. Zu zeigen ist, dass $\delta_L(\varphi(q), a) = \varphi(p)$. Dies ist korrekt, da

   $$\delta(q_0, xa) = \delta\big(\delta(q_0, x), a\big) = \delta(q, a) = p$$

   und somit $xa \in K_p$. Es folgt:

   $$\varphi(p) = K_p = [xa]_L = \delta_L([x]_L, a) = \delta_L(\varphi(q), a)$$

□

Bemerkung 2.44 (Totale versus partielle Übergangsfunktion im Minimalautomaten). Die Übergangsfunktion des Minimalautomaten ist total. Unter allen DFA mit totaler Übergangsfunktion ist $\mathcal{M}_L$ minimal. Jedoch ist eine weitere Zustandsreduktion möglich, wenn man partielle Übergangsfunktionen zulässt. Sei

$$L_0 \stackrel{\text{def}}{=} \big\{ x \in \Sigma^* : xz \notin L \text{ für alle } z \in \Sigma^* \big\}.$$

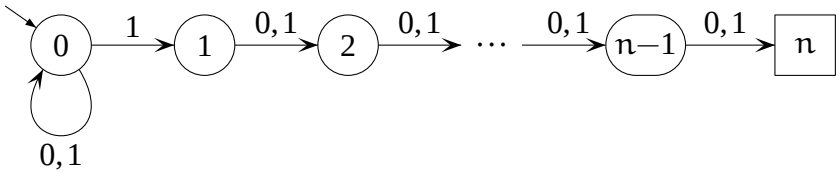
Falls $L_0 \neq \varnothing$, so ist L_0 ein Zustand im Minimalautomaten, von dem aus kein Endzustand erreichbar ist. Daher können L_0 und alle zugehörigen Kanten entfernt werden. Der resultierende Automat ist nun minimal unter allen DFA $\mathcal{M}$ mit $\mathcal{L}(\mathcal{M}) = L$. Beispielsweise kann L_0 im Beispiel 2.40 aus dem Minimalautomaten $\mathcal{M}_L$ für die Sprache $L = \mathcal{L}(0^*1^*)$ eliminiert werden. ■

[7] Beachte: Äquivalenzklassen sind (per Definition) nicht leer. Für die Zustände q in $\mathcal{M}$ ist $K_q \neq \varnothing$, da q vom Anfangszustand q_0 erreichbar ist.

In Satz 2.15 (Seite 29) haben wir die Äquivalenz von DFA und NFA nachgewiesen. Durch die Potenzmengenkonstruktion angewandt auf gegebenen NFA $\mathcal{M}$ kann ein DFA $\mathcal{M}_{\text{det}}$ entstehen, der exponentiell größer als $\mathcal{M}$ ist, selbst dann, wenn man alle unerreichbaren Zustände entfernt. Insbesondere ist auch der zeitliche Aufwand für die Konstruktion exponentiell in der Größe des NFA. Das exponentielle Blowup beim Übergang von einem NFA zu einem äquivalenten DFA lässt sich jedoch im allgemeinen nicht verhindern, da es Sprachen gibt, für die die Darstellung durch einen NFA sehr viel effizienter als durch einen DFA ist. Als Beispiel hierzu betrachten wir die Sprachfamilie $(L_n)_{n\geqslant 1}$, wobei

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

Die Sprache L_n wird durch folgenden NFA mit $n+1$ Zuständen der folgenden Form akzeptiert:



Wir zeigen nun mit Hilfe des Satzes von Myhill & Nerode, dass jeder DFA für L_n mindestens exponentiell viele Zustände hat.

Lemma 2.45. *Jeder DFA für* L_n *hat mindestens* 2^n *Zustände.*

Beweis. Es genügt zu zeigen, dass der Nerode-Index der Sprachen L_n mindestens exponentiell ist. Genauer gilt:

Je zwei Wörter $x \neq y \in \{0,1\}^n$ sind *nicht* $\sim_{L_n}$-äquivalent.

Hieraus folgt dann, dass $\left|\Sigma^*/L_n\right| \geqslant \left|\{0,1\}^n\right| = 2^n$.

Wir zeigen nun, dass für je zwei Wörter $x,y \in \{0,1\}^n$ mit $x \neq y$ gilt $x \not\sim_L y$. Sei $x = a_1a_2\ldots a_n$ und $y = b_1b_2\ldots b_n$ und $x \neq y$. Dann gibt es einen Index i mit $a_i \neq b_i$; etwa $a_i = 1$ und $b_i = 0$. Dann ist $x0^{i-1} \in L_n$ und $y0^{i-1} \notin L_n$, da a_i bzw. b_i das n-letzte Zeichen von $x0^{i-1}$ bzw. $y0^{i-1}$ ist. Also ist $x \not\sim_{L_n} y$. □

Der Minimierungsalgorithmus. Der Ausgangspunkt ist ein DFA $\mathcal{M} = (Q,\Sigma,\delta,q_0,F)$. Zur Vereinfachung nehmen wir an, dass die Übergangsfunktion total ist und dass alle Zustände $q \in Q$ vom Anfangszustand erreichbar sind. Gesucht ist ein äquivalenter DFA mit minimaler Anzahl an Zuständen. Aus Satz 2.43 (Seite 63) folgt, dass wir einen zum Minimalautomaten $\mathcal{M}_L$ isomorphen DFA konstruieren müssen. Die Idee des Minimierungsalgorithmus besteht darin, sukzessive Zustände in $\mathcal{M}$ zu identifizieren, die mit genau denselben Wörtern einen Endzustand erreichen. D.h., wir bilden den Quotienten-DFA unter der wie folgt definierten Äquivalenzrelation $\equiv$ auf den Zuständen von $\mathcal{M}$.

Definition 2.46 (Quotienten-DFA $\mathcal{M}/\equiv$). Sei $\mathcal{M} = (Q,\Sigma,\delta,q_0,F)$ ein totaler DFA. Sei $\equiv$ die wie folgt definierte Äquivalenzrelation auf dem Zustandsraum Q von $\mathcal{M}$:

$$q \equiv p \quad \text{gdw} \quad \begin{cases} \text{für alle Wörter } z \in \Sigma^* \text{ gilt:} \\ \delta(q,z) \in F \Leftrightarrow \delta(p,z) \in F. \end{cases}$$

Der Quotienten-DFA von $\mathcal{M}$ unter $\equiv$ ist wie folgt definiert:

$$\mathcal{M}/\equiv \stackrel{\text{def}}{=} \big(Q/\equiv, \Sigma, \delta', [q_0]_\equiv, F'\big),$$

wobei $F' \stackrel{\text{def}}{=} \big\{ [q]_\equiv : q \in F \big\}$ und $\delta'\big([q]_\equiv, a\big) \stackrel{\text{def}}{=} [\delta(q,a)]_\equiv$ mit $q \in Q$ und $a \in \Sigma$. ■

Bevor wir fortfahren, müssen wir uns überlegen, dass $\equiv$ tatsächlich eine Äquivalenzrelation ist und dass die Übergangsfunktion des Quotienten-DFA *wohldefiniert* ist. Ersteres ist offensichtlich. Für die Wohldefiniertheit der Übergangsfunktion δ' ist zu zeigen, dass für alle $q, p \in Q$ und $a \in \Sigma$ gilt:

$$\text{Aus } q \equiv p \text{ folgt } \delta(q,a) \equiv \delta(p,a).$$

Dies ist wie folgt einsichtig. Wir setzen $q \equiv p$ voraus. Zu zeigen ist nun, dass die Zustände $q' = \delta(q,a)$ und $p' = \delta(p,a)$ unter $\equiv$ zueinander äquivalent sind. Hierzu betrachten wir ein beliebiges Wort $z \in \Sigma^*$. Dann gilt $\delta(q',z) = \delta(q,az)$ und $\delta(p',z) = \delta(p,az)$. Hieraus folgt:

$$\begin{aligned} \delta(q',z) \in F \quad &\text{gdw} \quad \delta(q,az) \in F && (\text{da } \delta(q',z) = \delta(q,az)) \\ &\text{gdw} \quad \delta(p,az) \in F && (\text{da } q \equiv p) \\ &\text{gdw} \quad \delta(p',z) \in F && (\text{da } \delta(p',z) = \delta(p,az)) \end{aligned}$$

Lemma 2.47 (Korrektheit des Quotienten-DFA). *$\mathcal{M}/\equiv$ und $\mathcal{M}$ sind äquivalent.*

Beweis. Zunächst zeigen wir, dass der Übergang von $\mathcal{M}$ zum Quotienten-DFA die akzeptierte Sprache unverändert lässt. D.h., $\mathcal{L}(\mathcal{M}/\equiv) = \mathcal{L}(\mathcal{M})$. Im Folgenden schreiben wir kurz $[q]$ anstelle von $[q]_\equiv$. Zunächst überlegen wir uns, dass

$$q \in F \quad \text{gdw} \quad [q] \in F'$$

Die Richtung "$\Longrightarrow$" folgt sofort aus der Definition von F'. Für die Richtung "$\Longleftarrow$" nehmen wir an, dass $[p] \in F'$. Dann gibt es einen Endzustand $q \in F$ mit $[q] = [p]$; also $q \equiv p$. Wir betrachten das Wort $z = \varepsilon$ und erhalten $p = \delta(p,\varepsilon) \in F$, da $\delta(q,\varepsilon) = q \in F$.

Sei $z = a_1 \ldots a_n \in \Sigma^*$ und $q_0\, q_1 \ldots q_n$ der zugehörige Lauf in $\mathcal{M}$. $[q_0]\,[q_1] \ldots [q_n]$ ist dann der Lauf für z im Quotienten-DFA $\mathcal{M}/\equiv$. Dies kann durch Induktion nach der Wortlänge n gezeigt werden. Hieraus folgt nun die gewünschte Eigenschaft:

$$\begin{aligned} x \in \mathcal{L}(\mathcal{M}) \quad &\text{gdw} \quad q_0\, q_1 \ldots q_n \text{ ist akzeptierend} \\ &\text{gdw} \quad q_n \in F \\ &\text{gdw} \quad [q_n] \in F_\equiv \\ &\text{gdw} \quad [q_0]\,[q_1] \ldots [q_n] \text{ ist akzeptierend} \\ &\text{gdw} \quad x \in \mathcal{L}(\mathcal{M}/\equiv). \end{aligned}$$

□

Im Beweis des vorangegangenen Lemmas haben wir die Beobachtung gemacht, dass Läufe in $\mathcal{M}$ zu Läufen im Quotienten-DFA geliftet werden können, indem man von der Zustandsfolge $q_0\,q_1 \ldots q_n$ zu der Folge der Äquivalenzklassen $[q_0]_\equiv [q_1]_\equiv \ldots [q_n]_\equiv$ übergeht. Für die erweiterte Übergangsfunktion im Quotienten-DFA folgt hieraus:

$$\delta'([q_0]_\equiv, z) \;=\; [\delta(q_0, z)]_\equiv \qquad \text{für alle Wörter } z \in \Sigma^*$$

Mit der Annahme, dass alle Zustände q in $\mathcal{M}$ vom Anfangszustand q_0 erreichbar sind, also die Form $q = \delta(q_0, z)$ für ein Wort z haben, gilt nun auch, dass alle Zustände des Quotienten-DFA vom Anfangszustand $[q_0]_\equiv$ erreichbar sind.

Satz 2.48 (Isomorphie des Quotienten-DFA und Minimalautomaten). *$\mathcal{M}/\equiv$ und der Minimalautomat von $\mathcal{L}(\mathcal{M})$ sind isomorph.*

Beweis. Sei $L = \mathcal{L}(\mathcal{M})$. Wir zeigen, dass die Nerode-Äquivalenz $\sim_L$ mit der DFA-Äquivalenz $\sim_{\mathcal{M}/\equiv}$ des Quotienten-DFA übereinstimmt. Hierzu ist die gegenseitige Verfeinerung beider Äquivalenzrelationen zu zeigen:

(1) $\sim_{\mathcal{M}/\equiv}$ ist feiner als $\sim_L$

(2) $\sim_L$ ist feiner als $\sim_{\mathcal{M}/\equiv}$

Aussage (1) ist eine Folgerung der vorangegangenen Ergebnisse. Da $\mathcal{M}/\equiv$ die Sprache L akzeptiert (Lemma 2.47, Seite 66), ist somit die induzierte DFA-Äquivalenz $\sim_{\mathcal{M}/\equiv}$ eine Verfeinerung von $\sim_L$ (Lemma 2.38, Seite 61).

Nun zum Beweis von Aussage (2). Zu zeigen ist, dass die Nerode-Äquivalenz $\sim_L$ eine Verfeinerung der DFA-Äquivalenz $\sim_{\mathcal{M}/\equiv}$ ist. Wie zuvor schreiben wir $[q]$ anstelle von $[q]_\equiv$. Seien x, $y \in \Sigma^*$ mit $x \sim_L y$. Dann gilt für alle Wörter $z \in \Sigma^*$:

$$\begin{array}{rcccc} & xz \in L & \text{gdw} & yz \in L \\ \text{und somit:} & \delta(q_0, xz) \in F & \text{gdw} & \delta(q_0, yz) \in F \\ & \uparrow & & \uparrow \\ & \delta(\delta(q_0, x), z) & & \delta(\delta(q_0, y), z) \end{array}$$

Mit $p_x = \delta(q_0, x)$ und $p_y = \delta(q_0, y)$ gilt also:

$$\delta(p_x, z) \in F \;\text{ gdw }\; \delta(q_0, xz) \in F \;\text{ gdw }\; \delta(q_0, yz) \in F \;\text{ gdw }\; \delta(p_y, z) \in F$$

Da diese Aussage für beliebiges z gilt, folgt $p_x = \delta(q_0, x) \;\equiv\; \delta(q_0, y) = p_y$. Für die erweiterte Übergangsfunktion im Quotienten-DFA ergibt sich hieraus:

$$\delta'([q_0], x) \;=\; [\delta(q_0, x)] \;=\; [\delta(q_0, y)] \;=\; \delta'([q_0], y)$$

Also sind x und y DFA-äquivalent bezogen auf den Quotienten-DFA, d.h., $x \sim_{\mathcal{M}/\equiv} y$.

Die Voraussetzung, dass die Übergangsfunktion von $\mathcal{M}$ total ist und dass alle Zustände $q \in Q$ vom Anfangszustand q_0 erreichbar sind (d.h., die Form $q = \delta(q_0, z)$ für ein $z \in \Sigma^*$ haben), überträgt sich auf den Quotienten-DFA (siehe oben). Die Isomorphie von $\mathcal{M}/\equiv$ und $\mathcal{M}_L$ ist daher eine Folgerungs aus der Eindeutigkeit des Minimalautomaten, siehe Teil (a) von Satz 2.43 auf Seite 63. □

Der Minimierungsalgorithmus startet mit einem totalen DFA $\mathcal{M}$ ohne unerreichbare Zustände und berechnet nun die Äquivalenzrelation $\equiv$. Durch Zustandsaggregation (d.h., Zusammenfassen aller Zustände einer $\equiv$-Äquivalenzklasse) erhält man dann den Quotienten-DFA, also einen zum Minimalautomaten isomorphen DFA. Die Grobidee für die Berechnung von $\equiv$ besteht darin, sukzessive zweistellige Relationen

$$\mathcal{R}_0 \supseteq \mathcal{R}_1 \supseteq \mathcal{R}_2 \supseteq \ldots \supseteq \equiv$$

zu berechnen. Die Relationen $\mathcal{R}_i$ werden also zunehmend feiner und erfüllen die Eigenschaft:

$$\text{Aus } \langle q, q' \rangle \in Q^2 \setminus \mathcal{R}_i \text{ folgt } q \not\equiv q'.$$

Die initiale Relation $\mathcal{R}_0$ ist eine Äquivalenzrelation, die alle Endzustände und alle Zustände in $Q \setminus F$ identifiziert. Hier wird die Tatsache ausgenutzt, dass Endzutände niemals zu einem Zustand äquivalent sein können, der kein Endzustand ist. Im $(i+1)$-ten Iterationsschritt entfernen wir aus der Relation $\mathcal{R}_i$ ein Zustandspaar $\langle q, q' \rangle$, so dass

$$\langle \delta(q, a), \delta(q', a) \rangle \notin \mathcal{R}_i \text{ für ein } a \in \Sigma,$$

und erhalten somit die Relation $\mathcal{R}_{i+1}$. In diesem Fall gilt $\delta(q, a) \not\equiv \delta(q', a)$. Also gibt es ein Wort $z \in \Sigma^*$, so dass

$$\delta(q, az) = \delta(\delta(q, a), z) \in F \text{ und } \delta(q', az) = \delta(\delta(q', a), z) \notin F,$$

oder umgekehrt. In jedem Fall gilt dann $q \not\equiv q'$. Dies rechtfertigt es, das Zustandspaar $\langle q, q' \rangle$ aus $\mathcal{R}_i$ zu entfernen. Falls es kein solches Paar $\langle q, q' \rangle \in \mathcal{R}_i$ und $a \in \Sigma$ mit $(\delta(q, a), \delta(q', a)) \notin \mathcal{R}_i$ gibt, dann hält das Verfahren an. Für die finale Relation $\mathcal{R}_i$ gilt nun tatsächlich $\mathcal{R}_i = \equiv$. Dies ist wie folgt einsichtig. Die finale Relation $\mathcal{R}_i$ hat die Eigenschaft, dass für alle Zustandspaare $\langle q, q' \rangle \in \mathcal{R}_i$ gilt:

$$\langle \delta(q, a), \delta(q', a) \rangle \in \mathcal{R}_i \text{ für alle } a \in \Sigma \text{ und entweder } \{q, q'\} \cap F = \varnothing \text{ oder } \{q, q'\} \subseteq F.$$

Durch Induktion nach n kann nun gezeigt werden, dass für alle $\langle q, q' \rangle \in \mathcal{R}_i$ und alle Wörter z mit $|z| = n$ gilt:

$$\delta(q, z) \in F \quad \text{gdw} \quad \delta(q', z) \in F$$

Also ist $q \equiv q'$ für alle $\langle q, q' \rangle \in \mathcal{R}_i$. Die finale Relation $\mathcal{R}_j$ ist also feiner als die Äquivalenz $\equiv$. Wie oben erwähnt gilt andererseits, dass $\equiv$ feiner als $\mathcal{R}_i$ ist. Daher stimmt die finale Relation $\mathcal{R}_i$ mit $\equiv$ überein.

Wir formulieren den Algorithmus (der in der Literatur häufig als *table filling algorithm* bezeichnet wird) mit einer zweidimensionalen Tabelle, die die *ungeordneten* Zustandspaare verwaltet. Siehe Algorithmus 2. Für eine Implementierung kann man eine feste Nummerierung $q_0, q_1, \ldots, q_n$ der Zustände zugrundelegen und die Tabelle so erstellen, dass genau die geordneten Zustandspaare $\langle q_i, q_k \rangle$ mit $i > k$ dargestellt sind. Für alle ungeordneten Zustandspaare $\langle q_i, q_k \rangle$ ist in der Tabelle entweder eine Markierung oder kein Eintrag. Die Markierungen deuten an, für welche Zustandspaare $\langle q_i, q_k \rangle$ *bereits nachgewiesen* ist, dass $q_i \not\equiv q_k$. In der j-ten Iteration besteht die Relation $\mathcal{R}_j$ genau aus allen Paaren $\langle q_i, q_k \rangle$, für die *kein* Eintrag in der Tabelle ist.

Es ist klar, dass die Anzahl an Iterationen durch $\mathcal{O}(|Q|^2)$ beschränkt ist. Mit einigen Tricks, die hier nicht besprochen werden, lässt sich das Verfahren jedoch effizienter implementieren.

Algorithmus 2 Minimierungsalgorithmus für DFA (table filling algorithm)

Erstelle eine Tabelle für alle ungeordneten Zustandspaare $\langle q_i, q_k \rangle$.

Markiere in der Tabelle alle Paare $\langle q_i, q_k \rangle$ mit $\{q_i, q_k\} \cap F \neq \varnothing$ und $\{q_i, q_k\} \setminus F \neq \varnothing$.

WHILE es gibt ein unmarkiertes Paar $\langle q_i, q_k \rangle$ und ein $a \in \Sigma$,
so dass $\delta(q_i, a) \neq \delta(q_k, a)$ und das Paar $\langle \delta(q_i, a), \delta(q_k, a) \rangle$ markiert ist **DO**
markiere $\langle q_i, q_k \rangle$

OD

Bilde maximale Mengen paarweise unmarkierter Zustände.

Beispiel 2.49 (Minimierungsalgorithmus). Der Minimierungsalgorithmus angewandt auf den DFA $\mathcal{M}$ links in Abbildung 23 markiert zunächst die Paare $\langle q_4, q_0 \rangle$, $\langle q_4, q_1 \rangle$, $\langle q_4, q_2 \rangle$ und $\langle q_4, q_3 \rangle$, da nur q_4 ein Endzustand ist. Dann werden – in irgendeiner Reihenfolge – die Paare $\langle q_1, q_0 \rangle$, $\langle q_2, q_1 \rangle$, $\langle q_3, q_0 \rangle$ und $\langle q_3, q_2 \rangle$ markiert, da jeweils nur ein Zustand jedes Paares einen Übergang zum Endzustand besitzt. Folgende Tabelle fasst das Ergebnis der Markierungsschritte zusammen.

q_1	X			
q_2		X		
q_3	X		X	
q_4	X	X	X	X
	q_0	q_1	q_2	q_3

Letztendlich sind also q_0 und q_2 äquivalente Zustände, sowie auch q_1 und q_3. Der Quotienten-DFA ergibt sich nun, indem q_0, q_2 bzw. q_1, q_3 zu je einem Zustand zusammengefasst werden. Man erhält den Automaten $\mathcal{M}/\equiv$ rechts in Abbildung 23. ■

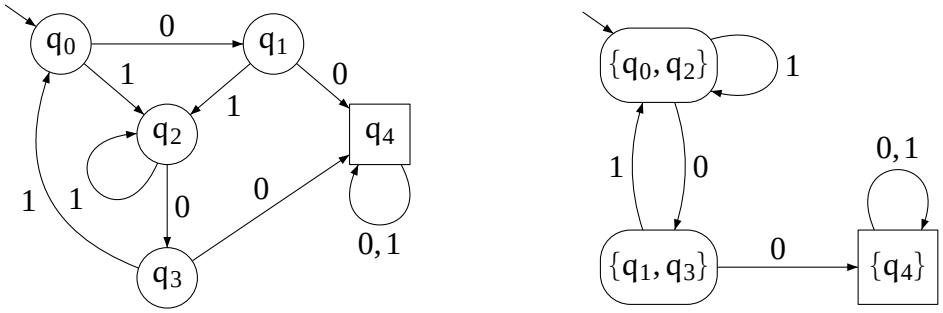


Abbildung 23: DFA $\mathcal{M}$ und der Quotienten-DFA $\mathcal{M}/\equiv$

Zusammenfassung

Folgender Satz fasst die Charakterisierungen regulärer Sprachen, die wir in diesem Abschnitt kennengelernt haben, zusammen.

Satz 2.50 (Charakterisierung regulärer Sprachen). *Sei* L *eine Sprache. Dann sind folgende Aussagen äquivalent:*

(a) L *ist regulär, d.h.* $L = \mathcal{L}(G)$ *für eine reguläre Grammatik* G.

(b) $L = \mathcal{L}(\mathcal{M})$ *für einen DFA* $\mathcal{M}$.

(c) $L = \mathcal{L}(\mathcal{M})$ *für einen NFA* $\mathcal{M}$.

(d) $L = \mathcal{L}(\mathcal{M})$ *für einen* ε*-NFA* $\mathcal{M}$.

(e) $L = \mathcal{L}(\alpha)$ *für einen regulären Ausdruck* α.

(f) Der Nerode-Index von L *ist endlich.*

In diesem Sinn sind die genannten Formalismen reguläre Grammatiken, DFA, NFA, ε-NFA und reguläre Ausdrücke gleichwertig. Darüberhinaus haben wir *konstruktive* Methoden angegeben, wie man die Formalismen ineinander überführen kann. Die meisten der Konstruktionen sind effizient. Ausnahmen hiervon ist die Determinisierung (und Komplementierung) von NFA mit der Potenzmengenkonstruktion sowie die Konstruktion regulärer Ausdrücke für gegebenen endlichen Automaten. Weiter hatten wir gesehen, dass die Klasse der regulären Sprachen unter allen gängigen Operatoren für formale Sprachen abgeschlossen ist und dass einige grundlegende algorithmische Problemstellungen recht einfach mit endlichen Automaten gelöst werden können.

3 Kontextfreie Sprachen

Sprachen wie $\{a^n b^n : n \geqslant 1\}$ oder die Sprache bestehend aus allen Wörtern über dem Alphabet $\{a, b\}$, in denen die Anzahl an a's und b's übereinstimmt, sind nicht regulär. Sprachen dieses Typs werden jedoch benötigt, um beispielsweise die Syntax korrekt geklammerter Ausdrücke (etwa arithmetischer Ausdrücke) festzulegen. Wir denken uns a als das Symbol für "Klammer auf" und b für "Klammer zu". Mit dieser Interpretation von a und b besteht die Sprache der korrekt geklammerten Ausdrücke aus allen Wörtern $w \in \{a, b\}^*$, so dass jedes Präfix w' von w mindestens so viele a's wie b's enthält und für w die Anzahl an a's und b's gleich ist. Z.B. steht $w = aababb$ für einen korrekten Ausdruck der Form $(()())$; während die Wörter abb oder $baba$ für die unsinnig geklammerten Ausdrücke $())$ bzw. $)()($ stehen. Anstelle der Klammern kann man sich auch andere für höhere Programmsprachen typischen Konstrukte wie "begin … end" oder "if … then … else …" vorstellen. Aus diesem Grund scheiden reguläre Sprachen als Formalismus für Programmiersprachen und die Syntaxanalyse im Compilerbau aus. Kontextsensitive Sprachen sind zwar mächtig genug, um geklammerte Ausdrücke zu spezifizieren, jedoch sind für das Wortproblem keine effizienten Algorithmen bekannt. Somit bleibt von der Chomsky Hierarchie nur die Klasse der kontextfreien Sprachen, die für die Syntaxanalyse in Frage zu kommen scheint. Tatsächlich haben wir bereits gesehen, dass z.B. die Syntax korrekt geklammerter arithmetischer Ausdrücke durch eine kontextfreie Grammatik darstellbar ist. Siehe Beispiel 1.10 auf Seite 13. Die oben erwähnte nicht-reguläre Sprache $L = \{a^n b^n : n \geqslant 1\}$ wird durch die CFG mit den beiden Regeln $S \to ab$ und $S \to aSb$ generiert.

3.1 Grundbegriffe

Wir verwenden im Folgenden häufig die englische Abkürzung CFG für "contextfree grammar" und manchmal auch CFL für "contextfree language". Zunächst erinnern wir an einige Grundbegriffe und Bezeichnungen, die in Abschnitt 1 eingeführt wurden. Eine CFG ist eine Grammatik, in der sämtliche Regeln die Gestalt $A \to x$ haben, wobei A ein Nichtterminal und x das leere Wort oder ein beliebiges Wort bestehend aus Variablen und Terminalzeichen ist. Siehe Definition 1.8 auf Seite 12 ff.

Wie zuvor verwenden wir Großbuchstaben $(S, A, B, C, \ldots)$ für die Variablen. Werden keine anderen Angaben gemacht, dann steht S für das Startsymbol. Kleinbuchstaben am Anfang des Alphabets $(a, b, c, \ldots)$ stehen für Terminalzeichen. Wörter bezeichnen wir mit Kleinbuchstaben am Ende des Alphabets (etwa u, v, w, x, y, z).

Größe einer CFG. Die Größe einer Grammatik G wird an der Anzahl an Nichtterminalen und der Summe der Längen aller Regeln gemessen und mit $|G|$ bezeichnet. Hierbei ist die Länge einer Regel $x \to y$ durch $|x| + |y|$ definiert. Das Terminalalphabet Σ wird als konstant angenommen und geht daher nicht in die Größe der Grammatik ein. Ist also $G = (V, \Sigma, \mathcal{P}, S)$, so ist

$$|G| \;=\; |V| \;+\; \sum_{(x,y)\in\mathcal{P}} (|x|+|y|).$$

Diese Definition ist für Grammatiken beliebigen Typs anwendbar, wird in dieser Vorlesung jedoch nur für den kontextfreien Fall verwendet.

ε-Freiheit. In Kapitel 1 wurde der Begriff der ε-Freiheit für kontextfreie Grammatiken eingeführt (siehe Seite 15). In ε-freien CFG ist höchstens die ε-Regel $S \to \varepsilon$ zulässig, wobei S das Startsymbol bezeichnet. Gehört die Regel $S \to \varepsilon$ zu dem Produktionssystem einer ε-freien CFG, dann wird gefordert, dass S in keiner Regel auf der rechten Seite erscheint. Dies stellt sicher, dass die ε-Regel $S \to \varepsilon$ niemals in einer Ableitung der Länge > 1 angewandt werden kann. In Abschnitt 1 wurde ein Algorithmus vorgestellt, der gegebene CFG in eine äquivalente ε-freie CFG überführt; also eine CFG, die im Wesentlichen keine ε-Regeln enthält. Das skizzierte Verfahren kann jedoch ein exponentielles Blowup verursachen. Wir illustrieren diese Aussage an einem einfachen Beispiel. Wir betrachten die Familie $(G_n)_{n\geqslant 1}$ von kontextfreien Grammatiken $G_n = (V_n, \Sigma_n, \mathcal{P}_n, S)$, wobei $V_n = \{S, A, B_1, \ldots, B_n\}$, $\Sigma_n = \{a, b_1, \ldots, b_n\}$ und $\mathcal{P}_n$ aus den Regeln

$$\begin{array}{lcl} S & \to & aB_1B_2\ldots B_n \\ B_1 & \to & \varepsilon \mid b_1 \\ B_2 & \to & \varepsilon \mid b_2 \\ & \vdots & \\ B_n & \to & \varepsilon \mid b_n \end{array}$$

besteht. Dann ist $|G_n| = \mathcal{O}(n)$. Das in Abschnitt 1 vorgestellte Verfahren angewandt auf G_n liefert eine ε-freie CFG G'_n mit den Regeln

$$S \to aB_{j_1}B_{j_2}\ldots B_{j_k} \text{ für jede Teilmenge } \{j_1, j_2, \ldots, j_k\} \text{ von } \{1, 2, \ldots, n\}$$

und den Regeln $B_i \to b_i$ für $1 \leqslant i \leqslant n$. Die Größe von G'_n ist also exponentiell in n. In diesem Beispiel variiert zwar das Alphabet, jedoch kann man das Beispiel für festes Alphabet $\Sigma = \{a, b\}$ abändern, z.B. indem man b_i durch b oder durch $a^i b$ ersetzt. Im ersten Fall (d.h., $b_i = b$) entsteht eine CFG ohne ε-Regeln exponentieller Größe, die allerdings zu der ε-freien CFG $\tilde{G}_n$ bestehend aus den Regeln $S \to aB^i$ für $0 \leqslant i \leqslant n$ und $B \to b$ äquivalent ist. Diese hat lineare Größe. Im zweiten Fall (d.h., $a^i b$ statt b_i) ist die Größe der ursprünglichen Grammatiken G_n zwar quadratisch in n, jedoch ändert dies nichts an dem Phänomen des exponentiellen Größenwachstums.

Das exponentielle Blowup beim Übergang von einer CFG zu einer äquivalenten ε-freien CFG ist jedoch vermeidbar. Es gibt ein effizienteres Verfahren, das eine gegebene CFG G in eine äquivalente ε-freie CFG derselben asymptotischen Größe überführt. Dieses benutzt einen Trick, den wir bei der Behandlung von Normalformen kennenlernen werden und der später in den Übungen besprochen wird. Es ist daher keine Einschränkung, den Sonderfall von ε-Regeln auszuklammern, und zur Vereinfachung anzunehmen, dass eine CFG vorliegt, die keinerlei ε-Regeln enthält.

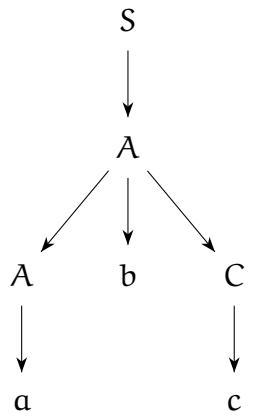
Ableitungsbaum. Jede Ableitung $A \Rightarrow x_1 \Rightarrow x_2 \Rightarrow \ldots \Rightarrow x_{n-1} \Rightarrow x_n = x$ eines Wortes $x \in (V \cup \Sigma)^*$ kann graphisch durch einen Baum, der *Ableitungsbaum* oder auch *Syntaxbaum* genannt wird, dargestellt werden. Der Ableitungsbaum ist ein gerichteter Baum, dessen Knoten mit Symbolen aus $V \cup \Sigma$ beschriftet sind. Die Wurzel ist mit dem Nichtterminal A markiert. Jeder innere Knoten v ist mit einem Nichtterminal beschriftet. Eine Verzweigung in v stellt eine der in der Ableitung $A \Rightarrow x_1 \Rightarrow \ldots \Rightarrow x_n = x$ angewandten Regeln dar. Die Markierungen der Blätter ergeben – von links nach rechts gelesen – das Wort x. (Dazu setzen wir eine Ordnung der Söhne eines inneren Knotens voraus, die es erlaubt, von "links" und "rechts" zu sprechen.)

Die Höhe des Ableitungsbaums ist die Anzahl innerer Knoten eines längsten Pfades (also eines längsten Weges von der Wurzel bis zu einem Nichtterminal) des Ableitungsbaums. Wir sprechen im Folgenden von einem Ableitungsbaum für ein Wort $x \in (V \cup \Sigma)^*$, um den Ableitungsbaum von x bzgl. einer Herleitung $S \Rightarrow x_0 \ldots \Rightarrow x_n = x$ zu bezeichnen. Selbstverständlich gibt es nur zu den Wörtern x mit $S \Rightarrow^* x$ einen Ableitungsbaum.

Beispiel 3.1 (Ableitungsbaum). Wir betrachten die CFG mit den Regeln $S \to A$, $A \to AbC$, $A \to a$, $C \to c$ und die beiden Ableitungen

$$(L) \qquad S \Rightarrow A \Rightarrow AbC \Rightarrow abC \Rightarrow abc$$
$$(R) \qquad S \Rightarrow A \Rightarrow AbC \Rightarrow Abc \Rightarrow abc$$

Beide Ableitungen induzieren denselben Ableitungsbaum, der unten skizziert ist.



Die Höhe dieses Ableitungsbaums beträgt 3. ■

Beispiel 3.1 zeigt, dass mehrere Ableitungen denselben Ableitungsbaum haben können. Allerdings unterscheiden sich Ableitungen mit demselben Ableitungsbaum stets höchstens in der Reihenfolge, in der die Regeln angewandt werden. Da sich die aus einer CFG ableitbaren Wörter aus den Ableitungsbäumen mit Wurzel S ergeben, kann man die Reihenfolge, in der die Nichtterminale ersetzt werden, festlegen, ohne die durch die CFG definierte Sprache zu ändern. Naheliegend ist es, die Nichtterminale von links nach rechts oder umgekehrt zu ersetzen.

Definition 3.2 (Rechtsableitung, Linksableitung). Sei $G = (V, \Sigma, \mathcal{P}, S)$ eine CFG. Die Rechtsableitungsrelation $\Rightarrow_R$ ist wie folgt definiert. Seien $v, w \in (V \cup \Sigma)^*$. Dann gilt

$$v \Rightarrow_R w$$

genau dann, wenn $v = xAz$, wobei $z \in \Sigma^*$, $A \in V$ und $x \in (V \cup \Sigma)^*$ und $w = xyz$ für eine Regel $A \to y$ von G. Die reflexive transitive Hülle von $\Rightarrow_R$ wird mit $\Rightarrow_R^*$ bezeichnet. In analoger Weise sind *Linksableitungen* und die Relationen $\Rightarrow_L$ und $\Rightarrow_L^*$ definiert. ■

Eine Ableitung $A \Rightarrow x_1 \Rightarrow \ldots \Rightarrow x_n$ ist also genau dann eine *Rechtsableitung*, falls in jedem Ableitungsschritt $x_{i-1} \Rightarrow x_i$ das rechteste Nichtterminal in x_{i-1} ersetzt wird. In Beispiel 3.1 (Seite 73) ist die erste (mit (L) markierte) Ableitung eine Linksableitung; während die zweite (mit (R) markierte) Ableitung eine Rechtsableitung ist.

Da man jeder Ableitung einen Ableitungsbaum zuordnen kann und aus diesem eine Rechtsableitung konstruieren kann, gilt

$$A \Rightarrow^* u \quad \text{gdw} \quad A \Rightarrow_R^* u$$

für alle $u \in (V \cup \Sigma)^*$. Daher stellt die Einschränkung auf Rechtsableitungen keinen Verlust für die durch eine CFG definierte Sprache dar. Entsprechendes gilt für Linksableitungen.

Nutzlose Variablen. Sei $G = (V, \Sigma, \mathcal{P}, S)$ eine CFG und $A \in V$. A heißt *nutzlos*, wenn es *kein* Wort $w \in \Sigma^*$ mit $A \Rightarrow^* w$ gibt. Nutzlose Variablen sind also solche Variablen, aus denen sich kein Wort bestehend aus Terminalzeichen ableiten lässt. Offenbar tragen nutzlose Variablen nicht zur erzeugten Sprache bei. Nutzlose Variablen – zusammen mit allen Regeln, in denen nutzlose Variablen vorkommen – können daher eliminiert werden, ohne die erzeugte Sprache zu ändern. Einzige Ausnahme ist der Sonderfall, dass das Startsymbol S nutzlos ist. (Aus formalen Gründen darf das Startsymbol nicht entfernt werden.) In diesem Fall ist die erzeugte Sprache $\mathcal{L}(G)$ leer. Z.B. sind die Variablen A und B der folgenden Grammatik nutzlos.

$$S \to A \mid B \mid C \qquad A \to aA \qquad B \to aAB \mid A \mid Bb \qquad C \to aSb \mid ab$$

Eine äquivalente CFG ohne nutzlose Variablen ist durch die Regeln $S \to C$, $C \to aSb$ und $C \to ab$ gegeben. Die Menge der nutzlosen Variablen lässt sich leicht mit einem *Markierungsalgorithmus* berechnen, der sukzessive alle Variablen markiert, aus denen wenigstens ein Wort gebildet aus Terminalzeichen herleitbar ist.

1. Markiere alle Variablen A mit $A \to w$ für ein $w \in \Sigma^*$.
2. Solange es Regeln der Form $B \to x_1 A_1 x_2 A_2 x_3 \ldots x_{n-1} A_n x_n$ in G gibt, so dass B nicht markiert ist, $A_1, \ldots, A_n$ markiert sind und $x_1, \ldots, x_n \in \Sigma^*$, wähle eine solche Regel und markiere B.
3. Gib alle nicht-markierten Variablen aus. Dies sind die nutzlosen Variablen von G.

Erzeugbare Variablen, Variablengraph. Ebenso wie nutzlose Variablen können auch alle Variablen entfernt werden, die in keinem aus dem Startsymbol S ableitbaren Wort über $V \cup \Sigma$ vorkommen. Variablen, für die es Wörter $u, v \in (V \cup \Sigma)^*$ mit $S \Rightarrow^* uAv$ gibt, nennen wir *erzeugbar*. Die erzeugbaren Variablen können durch eine Analyse des *Variablengraphen* von G gefunden werden. Der Variablengraph einer CFG $G = (V, \Sigma, \mathcal{P}, S)$ ist der gerichtete Graph $\mathcal{V}_G = (V, \hookrightarrow)$, dessen Knotenmenge mit der Variablenmenge V von G übereinstimmt und dessen Kantenrelation $\hookrightarrow \subseteq V \times V$ wie folgt definiert ist:

$$A \hookrightarrow B \overset{\text{def}}{\iff} \begin{cases} \text{es gibt eine Regel } A \to uBv \\ \text{in } G \text{ mit } u, v \in (V \cup \Sigma)^* \end{cases}$$

Mit einer Tiefen- oder Breitensuche im Variablengraph können alle erzeugbaren Variablen bestimmt werden. Entfernt man die nicht-erzeugbaren Variablen (nebst ihrer Regeln), so ändert sich die erzeugte Sprache von G nicht.

Bereinigte CFG. Wir nennen eine kontextfreie Grammatik $G = (V, \Sigma, \mathcal{P}, S)$ *bereinigt*, falls G keine nutzlosen Variablen hat und alle Variablen $A \in V$ erzeugbar sind. Das Bereinigen einer gegebenen CFG kann mit Hilfe des oben skizzierten Markierungsalgorithmus zur Bestimmung der nicht nutzlosen Variablen und einer Analyse des Variablengraphen vorgenommen werden. Wann immer es relevant ist, werden wir daher im Folgenden ohne Einschränkung annehmen, dass die vorliegende kontextfreie Grammatik bereinigt ist.

3.2 Die Chomsky Normalform

Für algorithmische Problemstellungen (z.B. das Wortproblem) – aber auch für den Nachweis von Eigenschaften kontextfreier Sprachen – ist es angenehm, von CFG in einer Normalform auszugehen. Eine der wichtigsten Normalformen für CFG ist die Chomsky Normalform.

Definition 3.3 (Chomsky Normalform (CNF)). Sei $G = (V, \Sigma, \mathcal{P}, S)$ eine CFG. G heißt in CNF, falls alle Regeln in G von der Form

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

sind, wobei $A, B, C \in V$ und $a \in \Sigma$. Wir sprechen von einer CNF-Grammatik, um eine CFG in CNF zu bezeichnen. ■

Z.B. ist die CFG mit den Regeln $S \to AD \,|\, AB$, $D \to SD \,|\, d$, $A \to a$ und $B \to b$ in CNF; nicht aber die CFG mit den Regeln $S \to aSb \,|\, ab$.

Eigenschaften von CNF-Grammatiken. Die Ableitungsbäume von CNF-Grammatiken sind stets Binärbäume. Innere Knoten, die für die Anwendung einer Regel der Form $A \to BC$ stehen, haben genau zwei Söhne. Innere Knoten mit nur einem Sohn stehen für die Anwendung einer Terminalregel $A \to a$. Diese sind nur auf der untersten Ebene möglich. Ist G eine CNF-Grammatik und $w \in \mathcal{L}(G)$ ein Wort der Länge n, so ist $n \geqslant 1$ und jede Ableitung von w in G besteht aus genau $2n-1$ Regelanwendungen, nämlich genau n Terminalregeln des Typs $A \to a$, mit denen die Zeichen von w generiert werden, und genau $n-1$ Regeln des Typs $A \to BC$. Für die Höhe zugehöriger Ableitungsbäume gilt:

Lemma 3.4 (Höhe von Ableitungsbäumen in CNF-Grammatiken). *Sei G eine CNF-Grammatik und $w \in \mathcal{L}(G)$ ein Wort der Länge n. Ist T ein Ableitungsbaum für w in G, so gilt:*

$$\log n + 1 \leqslant \textit{Höhe von } T \leqslant n$$

Beweis. Die obere Schranke n ergibt sich aus der Tatsache, dass jeder Pfad in T von der Wurzel zu einem Blatt nur durch genau einen inneren Knoten führt, der für die Anwendung einer Terminalregel $A \to a$ steht. Da jede Ableitung von w genau $n-1$ Regeln des Typs $A \to BC$ einsetzt, ist die Höhe von T durch $(n-1)+1 = n$ beschränkt.

Der Nachweis der unteren Schranke $\log n + 1$ kann durch Induktion nach n erfolgen. Genauer zeigen wir, dass die Ableitungsbäume aller Ableitungen $A \Rightarrow x_1 \Rightarrow \ldots \Rightarrow x_{2n-1} = w$ mit beliebiger Variable A und einem Wort w gebildet aus Terminalzeichen mit $|w| = n$ mindestens die Höhe $\log n + 1$ haben. Der Induktionsanfang $n = 1$ ist klar, da Ableitungen von Wörtern der

Länge 1 in CNF-Grammatiken nur aus der Anwendung einer Terminalregel $A \to a$ bestehen. Zugehörige Ableitungsbäume haben die Höhe 1. Nun zum Induktionsschritt $n-1 \Longrightarrow n$, wobei $n \geqslant 2$. Sei $w = a_1 a_2 \dots a_n$ und T ein Ableitungsbaum von w, dessen Wurzel mit A beschriftet ist. In der zugehörigen Ableitung muss im ersten Schritt eine Regel $A \to BC$ eingesetzt werden. Die weiteren Ableitungsschritte sind aus den Schritten von Ableitungen $B \Rightarrow^* a_1 \dots a_m$ und $C \Rightarrow^* a_{m+1} \dots a_n$ zusammengesetzt, wobei $1 \leqslant m \leqslant n-1$. Diese beiden Ableitungen sind in T durch die Teilbäume der beiden mit B und C beschrifteten Söhne der Wurzel dargestellt. Eines der beiden Wörter $a_1 \dots a_m$ oder $a_{m+1} \dots a_n$ muss mindestens die Länge $n/2$ haben. Etwa $m \geqslant n/2$. Nach Induktionsvoraussetzung hat jeder Ableitungsbaum für $B \Rightarrow^* a_1 \dots a_m$ mindestens die Höhe $\log m + 1 \geqslant \log n$. Die Höhe von T ist daher mindestens $\log n + 1$. □

Satz 3.5. *Zu jeder CFG G mit $\varepsilon \notin \mathcal{L}(G)$ gibt es eine äquivalente CFG in CNF.*

Beweis. Sei $G = (V, \Sigma, \mathcal{P}, S)$ eine CFG mit $\varepsilon \notin \mathcal{L}(G)$. Wir geben nun ein Verfahren an, mit dem G in eine äquivalente CFG G' in CNF überführt werden kann.

1. Schritt: Elimination von ε-Regeln. Zunächst erstellen wir eine zu G äquivalente ε-freie CFG G_1. Wegen $\varepsilon \notin \mathcal{L}(G)$, enthält diese keine ε-Regeln.

2. Schritt: Elimination von Kettenregeln. Im zweiten Schritt führen wir G_1 in eine äquivalente CFG G_2 *ohne Kettenregeln*, d.h. ohne Regeln der Form $A \to B$, über.

3. Schritt. G_2 wird durch eine äquivalente CFG G_3 ersetzt, in der sämtliche Regeln die Form $A \to a$ oder $A \to B_1 B_2 \dots B_k$ mit $k \geqslant 2$ haben, wobei $A, B_1, \dots, B_k$ Variablen sind und a ein Terminal.

4. Schritt. Im letzten Schritt ersetzen wir G_3 durch eine äquivalente CNF-Grammatik G_4.

Ein Verfahren zur Konstruktion einer ε-freien kontextfreien Grammatik G_1, die zu G äquivalent ist (1. Schritt), wurde bereits besprochen. Wir erläutern nun die restlichen drei Schritte.

2. Schritt. Elimination von Kettenregeln. Das Resultat des ersten Schritts ist eine zu G äquivalente CFG, die keine ε-Regeln enthält. Das Ziel ist es nun, Kettenregeln zu entfernen. Hierzu arbeiten wir in zwei Phasen.

1. Phase 1 entfernt zyklische Kettenregeln. Hierzu identifizieren wir alle Variablen, die auf einem Zyklus liegen. Sind etwa $B_0, B_1, \dots B_n$ Variablen, so dass G_1 die Kettenregeln $B_0 \to B_1$, $B_1 \to B_2$, …, $B_{n-1} \to B_n$ und $B_n \to B_0$ enthält, so wählen wir eine dieser Variablen, etwa $A = B_0$, und ersetzen nun in jeder Regel jedes Vorkommen von B_i für $1 \leqslant i \leqslant n$ durch A und entfernen $B_1, \dots, B_n$ aus der Variablenmenge.[8] Dies ist gerechtfertigt, da $B_i \Rightarrow^* A$ und $A \Rightarrow^* B_i$ und somit aus A und B_i genau dieselben Wörter ableitbar sind. Auf diese Weise entstehen nutzlose Kettenregeln der Form $A \to A$. Diese können ersatzlos gestrichen werden.

   Für die verbliebenen Variablen lässt sich nun eine Nummerierung $A_1, A_2, \dots, A_k$ bestimmen, so dass $i < j$ für jede der (verbliebenen) Kettenregeln $A_i \to A_j$. Algorithmisch ergibt

[8] Ist eine der Variablen B_j das Startsymbol S von G, sind alle anderen B_i's in S umzubenennen.

sich diese Nummerierung mittels Algorithmen zur Bestimmung einer topologischen Sortierung des gerichteten Graph bestehend aus den verbliebenen Variablen (Knoten) und den Kettenregeln als Kanten. Dieser Graph ist azyklisch, da die Variablen, die auf einem Zyklus liegen, identifiziert wurden. Er besitzt somit eine topologische Sortierung $A_1, \ldots, A_k$ und diese garantiert, dass $i < j$ für jede Kante/Kettenregel $A_i \to A_j$.

2. Phase 2 entfernt nun die verbliebenen azyklischen Kettenregeln. Sie benutzt hierzu die am Ende von Phase 1 bestimmte Nummerierung $A_1, \ldots, A_k$ der Variablen und wendet dann das in Algorithmus 3 auf Seite 77 angegebene Verfahren an.

Phase 1 hat offensichtlich keinen Einfluss auf die erzeugte Sprache. Die Korrektheit von Phase 2 ergibt sich aus der Beobachtung, dass Ableitungen der Form $A_i \Rightarrow A_j \Rightarrow x$ in der aus Phase 1 hervorgegangenen Grammatik durch Anwenden der neuen Regel $A_i \to x$ in G_2 simuliert werden können und dass alle "neuen" Regeln in G_2 Ableitungen in G_1 entsprechen.

Algorithmus 3 Elimination azyklischer Kettenregeln

```
(* Ausgangspunkt: Nummerierung A_1,...,A_k der Variablen, so dass  *)
(*                für jede Kettenregel A_i → A_j gilt: i < j       *)

FOR i = k−1,...,1 DO
  FOR j = i+1,...,k DO
    IF A_i → A_j THEN
      streiche die Regel A_i → A_j
      FOR ALL Regeln A_j → x DO
        füge die Regel A_i → x hinzu.
      OD
    FI
  OD
OD
```

3. Schritt. Nach dem zweiten Schritt liegt eine CFG G_2 vor, in der alle Regeln, die Gestalt $A \to a$ mit $a \in \Sigma$ oder $A \to x$ mit $x \in (\Sigma \cup V)^*$ haben, wobei $|x| \geqslant 2$. Regeln des Typs $A \to a$ mit $a \in \Sigma$ sind bereits in CNF-Gestalt. Regeln des Typs $A \to x$, wobei $|x| \geqslant 2$ und x wenigstens ein Terminal enthält, werden nun durch Regeln ersetzt, in denen A durch ein Wort bestehend aus zwei oder mehr Variablen ersetzt wird. Hierzu führen wir für jedes Terminal $a \in \Sigma$ eine neue Variable A_a mit der Regel $A_a \to a$ ein. (Aus A_a ist also nur das Wort a ableitbar.) Jede Regel $A \to yaz$ mit $y, z \in (\Sigma \cup V)^*$ und $|yz| \geqslant 1$ in G_2 wird nun durch $A \to yA_az$ ersetzt.

Es genügt nur für solche Terminalzeichen a ein neues Nichtterminal A_a einzuführen, für die es eine Regel $A \to x$ gibt, so dass $|x| \geqslant 2$ und a in x vorkommt. Weiter kann man $A_a = A$ setzen, falls A eine bereits existente Variable ist, für die es genau die Regel $A \to a$ gibt.

4. Schritt. Nach dem dritten Schritt liegt nun eine CFG G_3 vor, in der alle Regeln die Form $A \to a$ mit $a \in \Sigma$ oder $A \to B_1B_2 \ldots B_k$ haben, wobei die B_i's Variablen sind und $k \geqslant 2$. Um eine CNF-Grammatik zu erhalten, müssen also nur noch die Regeln $A \to B_1B_2 \ldots B_k$ mit $k \geqslant 3$ in geeigneter Weise ersetzt werden. Die Vorgehensweise ist wie folgt. Für jede Regel $A \to$

$B_1B_2\dots B_k$ in G_3 mit $k \geqslant 3$ fügen wir neue, paarweise verschiedene Variablen $C_1, C_2, \dots C_{k-2}$ ein und ersetzen die Regel $A \to B_1B_2\dots B_k$ durch folgende Regeln:

$$\begin{array}{lcl} A & \to & B_1C_1 \\ C_1 & \to & B_2C_2 \\ & \vdots & \\ C_{k-3} & \to & B_{k-2}C_{k-2} \\ C_{k-2} & \to & B_{k-1}B_k \end{array}$$

Die neuen Variablen $C_1, C_2, \dots C_{k-2}$ tauchen in keinen weiteren Regeln auf. Die Motivation für die Ersetzung der Regel $A \to B_1B_2\dots B_k$ durch die angegebenen CNF-Regeln besteht darin, jede Anwendung der Regel $A \to B_1\dots B_k$ in G_3 durch die Rechtsableitung

$$\begin{array}{lcl} A & \Rightarrow & B_1C_1 \\ & \Rightarrow & B_1B_2C_2 \\ & \Rightarrow & B_1B_2B_3C_3 \\ & \Rightarrow & \dots \\ & \Rightarrow & B_1B_2\dots B_{k-2}C_{k-2} \\ & \Rightarrow & B_1B_2\dots B_{k-2}B_{k-1}B_k \end{array}$$

in G_4 zu simulieren. Man macht sich leicht klar, dass alle vier Schritte spracherhaltend sind, d. h.,$\mathcal{L}(G) = \mathcal{L}(G_1) = \mathcal{L}(G_2) = \mathcal{L}(G_3) = \mathcal{L}(G_4)$ und dass G_4 offenbar eine CNF-Grammatik ist. Hiermit ist die Aussage von Satz 3.5 bewiesen. □

Beispiel 3.6 (Erstellung der CNF). Für die CFG G mit den Regeln

$$S \to C \text{ und } C \to aCb \mid ab$$

arbeitet der im Beweis von Satz 3.5 angegebene Algorithmus zur Erstellung einer äquivalenten CNF-Grammatik wie folgt. Der erste Schritt entfällt, da G keine ε-Regeln enthält. Da es keine Zyklen zyklischen Kettenregeln in G gibt, entfällt im Wesentlichen auch die erste Phase des zweiten Schritts. Die für die zweite Phase benötigte Nummerierung ist $A_1 = S$ und $A_2 = C$. Die zweite Phase des zweiten Schritts (Algorithmus 3) ersetzt nun die Kettenregel $S \to C$ durch die beiden Regeln

$$S \to aCb \quad \text{und} \quad S \to ab.$$

Im dritten Schritt werden neue Nichtterminale $A_a = A$ und $A_b = B$ mit den Regeln $A \to a$ und $B \to b$ hinzugefügt. Anschließend werden sämtliche Vorkommen von a bzw. b auf der rechten Seite einer Regel durch A bzw. B ersetzt. Wir erhalten das Produktionssystem

$$S \to ACB \mid AB \qquad C \to ACB \mid AB \qquad A \to a \qquad B \to b$$

Im vierten Schritt werden die beiden Regeln $S \to ACB$ und $C \to ACB$ durch

$$S \to AD,\ D \to CB,\ C \to AE,\ E \to CB$$

ersetzt. ■

Beispiel 3.7 (Erstellung der CNF, Elimination von Kettenregeln). Wir betrachten die CFG mit den folgenden Regeln.

$$\begin{array}{lll} S \to A \mid aB \mid aC & B \to S \mid Ba & C \to D \mid c \\ A \to B \mid C \mid cAdD & & D \to d \mid dA \end{array}$$

Der erste Schritt (Elimination von ε-Regeln) entfällt. Phase 1 des zweiten Schritts entfernt die zyklischen Kettenregeln $S \to A$, $A \to B$ und $B \to S$. Wir ersetzen A und B durch S und streichen die nun entstandene Kettenregel $S \to S$. Nach der ersten Phase von Schritt 2 liegt also eine Grammatik mit den Variablen S, C und D und folgenden Regeln vor.

$$\begin{array}{ll} S \to aS \mid aC \mid C \mid cSdD \mid Sa & C \to D \mid c \\ & D \to d \mid dS \end{array}$$

Die verbliebenen Kettenregeln sind also $S \to C$ und $C \to D$. Die für Phase 2 des zweiten Schritts benötigte Nummerierung ist $S = A_1$, $C = A_2$ und $D = A_3$. Algorithmus 3 ersetzt zuerst $C \to D$ durch $C \to d$ und $C \to dS$, dann $S \to C$ durch die drei Regeln $S \to d$, $S \to dS$ und $S \to c$. Man erhält die Grammatik G_2 mit folgenden Regeln:

$$\begin{array}{l} S \to aS \mid aC \mid cSdD \mid Sa \mid c \mid d \mid dS \\ C \to c \mid d \mid dS \\ D \to d \mid dS \end{array}$$

Im dritten Schritt werden die neuen Variablen A_a, A_c und A_d mit den zugehörigen Terminalregeln $A_a \to a$, $A_c \to c$ und $A_d \to d$ eingefügt. Wir erhalten die Grammatik G_3:

$$\begin{array}{ll} S \to A_aS \mid A_aC \mid A_cSA_dD \mid SA_a \mid c \mid d \mid A_dS & \\ & A_a \to a \\ C \to c \mid d \mid A_dS & A_c \to c \\ D \to d \mid A_dS & A_d \to d \end{array}$$

Nun sind fast alle Regeln in CNF-Gestalt. Einzige Ausnahme ist die Regel $S \to A_cSA_dD$. Diese wird nun im vierten Schritt durch die Regeln

$$S \to A_cC_1, \quad C_1 \to SC_2, \quad C_2 \to A_dD$$

ersetzt. ■

Bemerkung 3.8 (Elimination von Kettenregeln). In Algorithmus 3 ist die Reihenfolge, in der die Regeln $A_i \to A_j$ entfernt werden, wesentlich. Z.B. für die Grammatik G mit den Regeln

$$S \to A, \qquad A \to B \quad \text{und} \quad B \to b$$

wird zuerst $A \to B$ durch $A \to b$ ersetzt und dann $S \to A$ durch $S \to b$. Würde man zuerst S, dann A behandeln, dann würde man die Grammatik bestehend aus den Regeln

$$S \to B, \qquad A \to b, \qquad B \to b$$

erhalten, welche die Kettenregel $S \to B$ enthält. Selbst wenn man eventuell neu entstehende Kettenregeln ignoriert, kann das Verfahren fehlschlagen, falls die im Algorithmus beschriebene Reihenfolge nicht eingehalten wird. In obigem Beispiel würde man die Grammatik mit den Regeln $A \to b$ und $B \to b$ erhalten, in der es keine Regel für das Startsymbol S gibt. Die erzeugte Sprache dieser Grammatik ist leer und stimmt somit *nicht* mit der durch G definierten Sprache $\mathcal{L}(G) = \{b\}$ überein. ■

Größe der konstruierten CNF-Grammatik. Mit der beschriebenen Methode zur Erstellung einer äquivalenten CNF-Grammatik G' gilt $|G'| = \mathcal{O}(|G|^2)$, sofern im ersten Schritt $G \rightsquigarrow G_1$ ein Verfahren eingesetzt wird, welches gegebene CFG G durch eine ε-freie CFG G_1 derselben asymptotischen Größe ersetzt.[9] Für den vierten Schritt $G_3 \rightsquigarrow G_4$ gilt $|G_4| = \mathcal{O}(|G_3|)$, da lediglich Regeln $A \to B_1 B_2 \dots B_k$ der Länge $k+1$ durch $k-1$ Regeln $C_i \to C_{i+1} C_{i+2}$ der Länge 3 ersetzt werden. Dabei ist $C_0 = A$, $C_k = B_{k-1}$ und $C_{k+1} = B_k$. Setzt man voraus, dass jedes Zeichen des Terminalalphabets von G in wenigstens einer Regel vorkommt, dann hat auch der dritte Schritt (das Einfügen neuer Variablen A_a und zugehöriger Regeln $A_a \to a$) keinen Einfluss auf die asymptotische Größe. Es gilt also $|G_3| = \mathcal{O}(|G_2|)$. Die Elimination der Kettenregeln (Schritt 2) kann jedoch die Größe der Grammatik quadratisch vergrößern. Die Behandlung zyklischer Kettenregeln in Phase 1 hat zwar keine Auswirkungen auf die asymptotische Größe einer Grammatik, jedoch kann der Kopiervorgang in Phase 2 zur Behandlung azyklischer Kettenregeln (Algorithmus 3) ein quadratisches Blowup bewirken. Ein einfaches Beispiel hierfür ist die Grammatik $G = G_1$ bestehend aus den Regeln $A_i \to A_{i+1}$ und $A_i \to a_i$ für $1 \leqslant i < n$ und $A_n \to a_n$. (Etwa $S = A_1$.) Diese hat lineare Größe. Algorithmus 3 erzeugt eine Grammatik quadratischer Größe:

$$\begin{aligned}
A_1 &\to a_1 \mid a_2 \mid a_3 \mid \dots \mid a_n \\
A_2 &\to a_2 \mid a_3 \mid \dots \mid a_n \\
A_3 &\to a_3 \mid \dots \mid a_n \\
&\vdots \\
A_n &\to a_n
\end{aligned}$$

3.3 Der Cocke-Younger-Kasami Algorithmus

Eine der zentralen algorithmischen Problemstellungen für Grammatiken ist das *Wortproblem*. Dieses hat als Eingabe eine Grammatik G und ein Wort w über das Terminalalphabet von G und fragt, ob w aus G hergeleitet werden kann, also ob $w \in \mathcal{L}(G)$. Eine wichtige Instanz ist das Parsingproblem des Compilers, bei dem w für den vom Benutzer geschriebenen Programmtext steht und der Parser zu prüfen hat, ob die syntaktischen Regeln der betreffenden Programmiersprache eingehalten wurden. Im Compilerbau werden hierfür sehr ausgefeilte Algorithmen für spezielle Arten kontextfreier Grammatiken eingesetzt. Wir behandeln hier einen konzeptionell einfachen Algorithmus, der von einer kontextfreien Grammatik in Chomsky Normalform ausgeht.

Im Folgenden sei $G = (V, \Sigma, \mathcal{P}, S)$ eine CNF-Grammatik und $w = a_1 a_2 \dots a_n \in \Sigma^+$. Wir schreiben $w_{i,j}$ für das Teilwort der Länge j, das an Position i beginnt, also $w_{i,j} = a_i a_{i+1} \dots a_{i+j-1}$. Dabei ist $1 \leqslant i \leqslant n$ und $1 \leqslant j \leqslant n+1-i$. Für den Test, ob $w \in \mathcal{L}(G)$ liegt, wenden wir die Methode des dynamischen Programmierens zur Bestimmung der Variablenmengen

$$V[i,j] = \left\{ A \in V : A \Rightarrow^* w_{i,j} \right\}$$

an. Der Bereich von i und j ist wie zuvor, also $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant n+1-i$. Die Menge $V[i,j]$ besteht also aus allen Variablen A, aus denen sich das Teilwort $w_{i,j}$ von w ableiten lässt. Die

[9] Wie bereits erwähnt kann der in Abschnitt 1 (siehe Seite 15) beschriebene Algorithmus, der eine gegebene CFG in eine äquivalente ε-freie CFG überführt, zu einem exponentiellen Blowup führen. Es gibt jedoch ein effizientes Verfahren, welches die asymptotische Größe der CFG unverändert lässt (siehe Übungen).

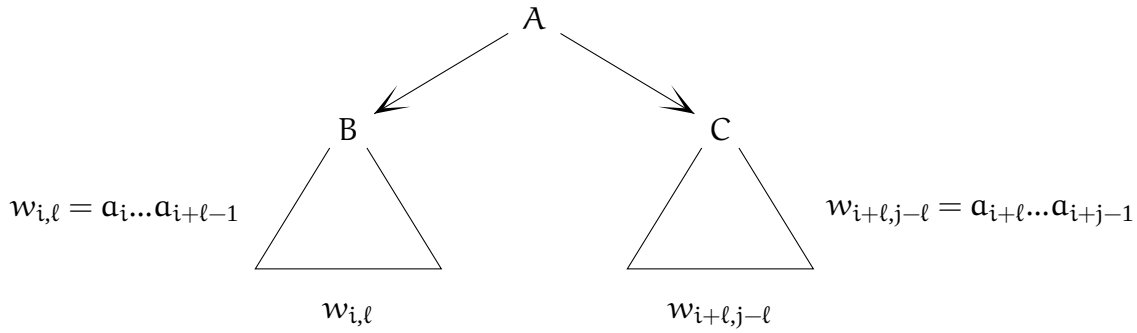


Abbildung 24: Ableitungsbaum für $w_{i,j} = a_i a_{i+1} \dots a_{i+j-1}$

Antwort auf das Wortproblem für G und $w = w_{1,n}$ ergibt sich durch Betrachtung der Variablenmenge $V[i,j]$ für die Indizes $i = 1$ und $j = n$:

$$w \in \mathcal{L}(G) \quad \text{gdw} \quad S \in V[1,n].$$

Die Berechnung der Variablenmengen $V[i,j]$ erfolgt in Bottom-Up Manier mit zunehmender Länge j der betrachteten Teilwörter von w. Die Initialisierungsphase betrachten wir Teilwörter der Länge 1, also die einzelnen Zeichen von w. Da G in CNF ist, sind diese nur durch die Anwendung einer Terminalregel herleitbar. Also:

$$V[i,1] \;=\; \big\{ A \in V : A \to a_i \big\}$$

Die Ableitungen, die mit einer Variablen A starten, und eines der Teilwörter von w der Länge $j \geqslant 2$ erzeugen, haben mindestens die Länge 2. Im ersten Ableitungsschritt muss also eine Regel der Form $A \to BC$ eingesetzt werden. Für $A \in V$ gilt daher $A \Rightarrow^* w_{i,j}$ genau dann, wenn es eine Regel $A \to BC$ gibt, so dass aus B und C ein nicht-leeres Präfix der Länge ℓ bzw. Suffix der Länge $j-\ell$ von $w_{i,j}$ herleitbar ist. Die Struktur eines zugehörigen Ableitungsbaums ist in Abbildung 24 dargestellt. Es gilt also:

$$A \Rightarrow^* w_{i,j} \quad \text{gdw} \quad \begin{cases} \text{es gibt eine Regel } A \to BC \text{ und eine Zahl } \ell \in \{1,\dots,j-1\} \\ \text{mit } B \Rightarrow^* w_{i,\ell} \text{ und } C \Rightarrow^* w_{i+\ell,j-\ell} \end{cases}$$

Wir erhalten folgende rekursive Charakterisierung der Variablenmengen $V[i,j]$.

$$V[i,j] \;=\; \bigcup_{\ell=1}^{j-1} \big\{ A \in V : \text{ es gibt eine Regel } A \to BC \text{ mit } B \in V[i,\ell] \text{ und } C \in V[i+\ell, j-\ell] \big\}$$

Diese Beobachtung wird im Algorithmus von Cocke-Younger-Kasami, kurz CYK-Algorithmus genannt, angewandt. Siehe Algorithmus 4 auf Seite 82. Dieser berechnet in Bottom-Up-Manier die Zeilen der folgenden Tabelle.

$V[1,n]$						
$V[1,n-1]$	$V[2,n-1]$					
$V[1,n-2]$	$V[2,n-2]$	$V[3,n-2]$				
	$\vdots$					
$V[1,2]$	$V[2,2]$	$V[3,2]$	$V[4,2]$	$\dots$	$V[n-1,2]$	
$V[1,1]$	$V[2,1]$	$V[3,1]$	$V[4,1]$	$\dots$	$V[n-1,1]$	$V[n,1]$
a_1	a_2	a_3	a_4	$\dots$	a_{n-1}	a_n

Algorithmus 4 Der CYK-Algorithmus

(* Eingabe: CNF-Grammatik $G = (V, \Sigma, \mathcal{P}, S)$ und Wort $w = a_1 a_2 \dots a_n \in \Sigma^+$ *)
(* Aufgabe: prüft, ob $w \in \mathcal{L}(G)$ *)

FOR $i = 1, \dots, n$ **DO**
  $V[i,1] := \{A \in V : A \to a_i\}$;
OD
FOR $j = 2, \dots, n$ **DO**
  FOR $i = 1, \dots, n+1-j$ **DO**
    $V[i,j] := \varnothing$;
    FOR $\ell = 1, \dots, j-1$ **DO**
      $V[i,j] := V[i,j] \cup \{A \in V :$ es gibt eine Regel $A \to BC$ mit $B \in V[i,\ell]$ und $C \in V[i+\ell, j-\ell]\}$
    OD
  OD
OD
IF $S \in V[1,n]$ **THEN**
  Return "ja, $w \in \mathcal{L}(G)$."
ELSE
  Return "nein, $w \notin \mathcal{L}(G)$."
FI

Es ist offensichtlich, dass die Laufzeit des CYK-Algorithmus kubisch in der Länge des Eingabeworts ist und dass der Platzbedarf quadratisch ist. Die Kosten beziehen sich auf den Fall, dass die kontextfreie Sprache durch eine feste CNF-Grammatik dargestellt ist. Die Größe der CNF-Grammatik wird daher als konstant angesehen. Wir erhalten folgenden Satz:

Satz 3.9. *Das Wortproblem für CNF-Grammatiken lässt sich mit dem CYK-Algorithmus in Zeit* $\mathcal{O}(n^3)$ *und Platz* $\mathcal{O}(n^2)$ *lösen. Dabei ist* n *die Länge des Eingabeworts.*

Beispiel 3.10 (CYK-Algorithmus). Die Arbeitsweise des CYK-Algorithmus für die Sprache $L = \{a^n b^n : n \geqslant 1\}$, dargestellt durch die CNF-Grammatik

$$S \to AC \mid AB \qquad C \to SB \qquad A \to a \qquad B \to b,$$

und das Wort $w = aabb$ ist wie folgt. Im Initialisierungsschritt wird

$$V[1,1] = V[2,1] = \{A\} \quad \text{und} \quad V[3,1] = V[4,1] = \{B\}$$

gesetzt. Daraus ergibt sich $V[1,2] = V[3,2] = \varnothing$ und $V[2,2] = \{S\}$. Im zweiten Schleifendurchlauf erhalten wir

$$V[1,3] = \varnothing \quad \text{und} \quad V[2,3] = \{C\},$$

da $S \in V[2,2]$, $B \in V[3,1]$ und $C \to SB$. Wegen $A \in V[1,1]$ und $C \in V[2,3]$ ergibt sich $V[1,4] =$

$\{S\}$. Die oben erwähnte Tabelle hat also die Gestalt:

$V[1,4]$	$\{S\}$			
$V[i,3]$	$\varnothing$	$\{C\}$		
$V[i,2]$	$\varnothing$	$\{S\}$	$\varnothing$	
$V[i,1]$	$\{A\}$	$\{A\}$	$\{B\}$	$\{B\}$
	a	a	b	b

Der Algorithmus terminiert also mit der Antwort "ja". ■

Weitere algorithmische Probleme für CFG. Das Leerheitsproblem "ist $\mathcal{L}(G) = \varnothing$ für gegebene CFG G?" lässt sich mit dem auf Seite 74 skizzierten Markierungsalgorithmus zur Bestimmung nutzloser Variablen lösen. Das Startsymbol S ist genau dann nutzlos, wenn $\mathcal{L}(G) = \varnothing$. Auch das Endlichkeitsproblem "ist $\mathcal{L}(G)$ endlich?" ist für CFG effizient lösbar. Darauf gehen wir später ein. Viele andere algorithmische Fragestellungen für CFG, z.B. das Äquivalenzproblem "gilt $\mathcal{L}(G_1) = \mathcal{L}(G_2)$?" oder die Frage, ob $\mathcal{L}(G)$ regulär ist, sind jedoch unentscheidbar, d.h., nicht algorithmisch lösbar.

3.4 Eigenschaften kontextfreier Sprachen

Wir zeigen zunächst, dass jede kontextfreie Sprache eine Pumpeigenschaft besitzt, die für den Nachweis der Nicht-Kontextfreiheit nützlich sein kann. Später diskutieren wir Abschlusseigenschaften für die Klasse der kontextfreien Sprachen unter den fünf gängigen Kompositionsoperatoren für formale Sprachen.

3.4.1 Das Pumping Lemma für kontextfreie Sprachen

In Analogie zum Pumping Lemma für reguläre Sprachen gibt es ein notwendiges Kriterium für kontextfreie Sprachen. Wir werden dieses mit Hilfe der Ableitungsbäume für CNF-Grammatiken beweisen und später für den Entwurf eines Endlichkeitstests für kontextfreie Grammatiken benutzen.

Satz 3.11 (Pumping Lemma für kontextfreie Sprachen). *Sei L eine kontextfreie Sprache. Dann gibt es eine natürliche Zahl n, so dass sich jedes Wort $z \in L$ der Länge $\geqslant n$ wie folgt zerlegen lässt: $z = uvwxy$, wobei*

(1) $uv^k wx^k y \in L$ *für alle* $k \in \mathbb{N}$

(2) $|vx| \geqslant 1$

(3) $|vwx| \leqslant n$.

Beweis. Wir können o.E. annehmen, dass $\varepsilon \notin L$. (Ist $\varepsilon \in L$, dann betrachten wir die Sprache $L \setminus \{\varepsilon\}$ anstelle von L.) Sei G eine CNF-Grammatik mit $L = \mathcal{L}(G)$ (siehe Satz 3.5, Seite 76). Weiter sei $N = |V|$ die Anzahl an Variablen in G. Die im Pumping Lemma genannte Konstante n ist wie folgt:

$$n = 2^N.$$

Sei $z \in L$ mit $|z| \geqslant n$. Zu zeigen ist, dass sich z in fünf Teilwörter u, v, w, x, y zerlegen lässt, so dass Eigenschaften (1), (2) und (3) erfüllt sind. Als Hilfsmittel für die Definition solcher Teilwörter u, v, w, x, y verwenden wir einen Ableitungsbaum T für z. Dieser ist ein Binärbaum mit $|z|$ Blättern. Innere Knoten mit genau einem Sohn repräsentieren eine Regel der Form $A \to a$. Sie haben die Höhe 1. Alle anderen inneren Knoten stehen für eine Regel der Form $A \to BC$. Sie haben genau zwei Söhne. Die Höhe von T sei $h{+}1$. Wegen Lemma 3.4 (Seite 75) gilt:

$$h \;\geqslant\; \log|z| \;\geqslant\; \log n \;=\; N$$

Sei $v_0 v_1 \ldots v_{h+1}$ ein Pfad in T der Länge $h{+}1$ von der Wurzel v_0 zu einem Blatt v_{h+1}. Also ist v_{h+1} ein Zeichen $a \in \Sigma$ von z. Weiter sei A_i die Markierung des Knotens v_i, $i = 0, 1, \ldots, h$. Dann ist $A_0 = S, A_1, \ldots, A_h$ eine Folge von Variablen der Länge $h{+}1 \geqslant N{+}1$. Also gibt es eine Variable A und Indizes $i, j \in \{0, 1, \ldots, h\}$ mit $i < j$, so dass $A = A_i = A_j$ und so dass die Variablen $A_{i+1}, \ldots, A_h$ paarweise verschieden sind.

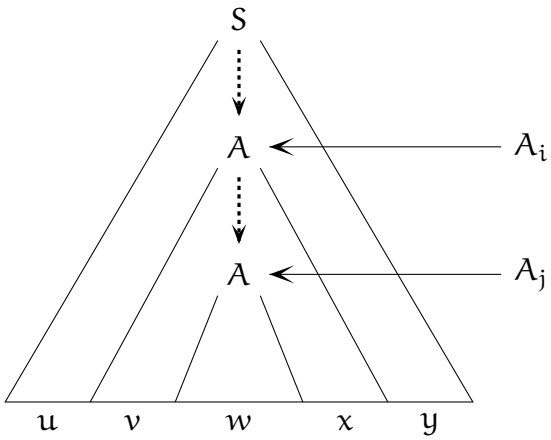


Abbildung 25: Zerlegung von z in $uvwxy$

Wir zerlegen z in $z = uvwxy$ gemäß der Skizze in Abbildung 25 auf Seite 84 und zeigen, dass die geforderten Bedingungen erfüllt sind.

ad (1). Ableitungen der Wörter $uv^k wx^k y$ ergeben sich durch Kombinieren der Ableitungen

$$S \;\Rightarrow^*\; uA_i y \;=\; uAy, \qquad A = A_j \;\Rightarrow^*\; w, \qquad A = A_i \;\Rightarrow^*\; vA_j x \;=\; vAx.$$

Es folgt $S \Rightarrow^* uv^k wx^k y$ (also $uv^k wx^k y \in L$) für alle $k \in \mathbb{N}$. Abbildung 26 skizziert die Ableitungsbäume für $k = 0$ und $k = 2$.

ad (2). Da G in CNF ist, ist $v \neq \varepsilon$ oder $x \neq \varepsilon$. (Da v_i ein innerer Knoten der Höhe > 1 ist, steht v_i für das Anwenden einer Regel der Gestalt $A_i \to BC$. Ist z.B. v_j ein Nachfolger des mit B markierten Sohns von v_i, so ist das aus C hergeleitete Wort ein Suffix von x mit $|x| \geqslant 1$.) Es folgt $|vx| \geqslant 1$.

ad (3). Wir betrachten den Teilbaum T' mit Wurzel v_i. T' ist ein Ableitungsbaum für die Ableitung $A_i \Rightarrow^* vwx$. Die Höhe von T' ist $h{-}i{+}1$, da $v_i, v_{i+1}, \ldots, v_{h+1}$ ein längster Pfad in T' ist. Da $A_{i+1}, \ldots, A_h$ paarweise verschieden sind, gilt $h{-}i \leqslant N$. Die Höhe von T' ist daher $\leqslant N{+}1$. Da in jeder CNF-Grammatik die Höhe eines Ableitungsbaum für ein Wort der Länge ℓ mindestens

$\log \ell + 1$ ist (siehe Lemma 3.4 auf Seite 75), folgt hieraus:

$$|vwx| \leqslant 2^{h-i} \leqslant 2^N = n$$

Die Zerlegung $z = uvwxy$ erfüllt also die drei geforderten Bedingungen. □

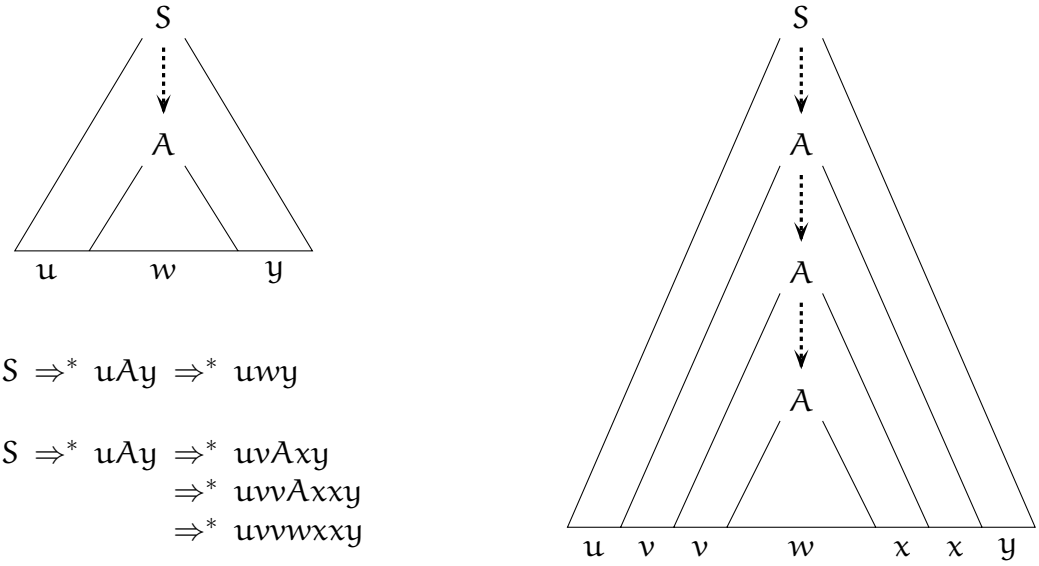


Abbildung 26: Pumpeigenschaft für $k = 0$ und $k = 2$

Nicht-kontextfreie Sprachen. Das Pumping Lemma kann oftmals hilfreich sein, um nachzuweisen, dass eine Sprache *nicht* kontextfrei ist. Als Beispiel betrachten wir die Sprache

$$L = \{a^n b^n c^n : n \in \mathbb{N}\},$$

und zeigen mit Hilfe des Pumping Lemmas, dass L nicht kontextfrei ist. Wir nehmen an, dass L kontextfrei ist und führen diese Annahme zu einem Widerspruch. Sei n die Zahl aus dem Pumping Lemma und $z = a^n b^n c^n$. Das Wort z liegt in L und hat die Länge $3n > n$. Es gibt also eine Zerlegung $z = uvwxy$, so dass die Eigenschaften (1), (2) und (3) aus dem Pumping Lemma erfüllt sind. Wegen $|vwx| \leqslant n$ (Bedingung (3)) ist vwx entweder ein Teilwort von $a^n b^n$ oder von $b^n c^n$. Wegen (2) ist wenigstens eines der Wörter v oder x nicht leer. Daher ist es nicht möglich, dass die Anzahl der Vorkommen der drei Symbole a, b und c in uv^2wx^2y gleich ist. Daher ist $uv^2wx^2y \notin L$. Dies steht im Widerspruch zur Pumpeigenschaft (1).

Endlichkeitsproblem für CFG. Das Endlichkeitsproblem für CFG hat eine kontextfreie Grammatik G als Eingabe und fragt, ob die erzeugte Sprache endlich ist. Dies lässt sich algorithmisch wie folgt lösen. Zunächst wenden wir die bereits besprochenen Verfahren an, um G zu bereinigen (siehe Seite 75) und in CNF zu bringen (siehe Beweis von Satz 3.5). Falls G das leere Wort akzeptiert, so erzeugt die konstruierte CNF-Grammatik G' die Sprache $\mathcal{L}(G) \setminus \{\varepsilon\}$. Diese ist genau dann endlich, wenn $\mathcal{L}(G)$ endlich ist. Wir nehmen nun an, dass eine bereinigte CNF-Grammatik G vorliegt. Für diese führen wir einen Zyklentest im Variablengraphen

$\mathcal{V}_G$ durch. (Siehe Seite 74.) Hierfür gibt es Linearzeitalgorithmen, die hier nicht besprochen werden. Ist $\mathcal{V}_G$ zyklisch, so ist $\mathcal{L}(G)$ unendlich. Andernfalls ist $\mathcal{L}(G)$ endlich.

Die Korrektheit des Verfahrens kann wie folgt bewiesen werden. Zunächst nehmen wir an, dass $\mathcal{V}_G$ zyklisch ist und zeigen, dass aus G unendlich viele Wörter über dem Terminalalphabet Σ herleitbar sind. Sei etwa $A_0 A_1 \dots A_m$ ein Zyklus in $\mathcal{V}_G$ und und $A = A_0 = A_m$. Da G bereinigt ist, gibt es Wörter $u, w, y \in \Sigma^*$ mit $S \Rightarrow^* uAy$ (da A erzeugbar ist und keine Variable nutzlos ist) und $A \Rightarrow^* w$ (da A nicht nutzlos ist). Da A auf einem Zyklus liegt, G keine ε-Regeln hat und keine Variable von G nutzlos ist, gibt es Wörter $v, x \in \Sigma^*$ mit $A \Rightarrow^* vAx$ und $|vx| \geqslant 1$. Ab hier können wir wie im Pumping-Lemma argumentieren und erhalten, dass für jedes $k \geqslant 0$ das Wort $uv^k wx^k y$ aus G herleitbar ist:

$$S \;\Rightarrow^*\; uAy \;\Rightarrow^*\; uv^k Ax^k y \;\Rightarrow^*\; uv^k wx^k y.$$

Da wenigstens eines der Wörter v oder x nicht-leer ist, sind die Wörter $uv^k wx^k y$ paarweise verschieden. Wir nehmen nun umgekehrt an, dass $\mathcal{L}(G)$ unendlich ist und zeigen, dass $\mathcal{V}_G$ einen Zyklus enthält. Sei n die Anzahl an Variablen in G. Da es nur endlich viele Wörter der Länge $\leqslant 2^n$ gibt, muss es ein Wort $w \in \mathcal{L}(G)$ mit $|w| > 2^n$ geben. Die weitere Argumentation ist nun wie im Beweis des Pumping-Lemmas. Wir betrachten einen Ableitungsbaum T für w und einen längsten Pfad in T. Auf diesem Pfad muss wenigstens eine Variable mehrfach vorkommen, also auf einem Zyklus in $\mathcal{V}_G$ liegen.

3.4.2 Abschlusseigenschaften kontextfreier Sprachen

Wir betrachten hier zunächst nur die drei Operatoren Vereinigung, Konkatenation und Kleeneabschluss, die sich recht einfach mit kontexfreien Grammatiken realisieren lassen.

Vereinigung. Seien $G_1 = (V_1, \Sigma, \mathcal{P}_1, S_1)$ und $G_2 = (V_2, \Sigma, \mathcal{P}_2, S_2)$ kontextfreie Grammatiken mit demselben Terminalalphabet Σ. Wir können o.E. annehmen, dass $V_1 \cap V_2 = \varnothing$. Andernfalls können die Nichtterminale entsprechend umbenannt werden. $G_1 \uplus G_2$ bezeichnet diejenige Grammtik, die man erhält, wenn man alle Regeln in G_1 und G_2 zulässt und ein neues Startsymbol S hinzufügt, aus dem die Startsymbole von G_1 und G_2 in einem Schritt herleitbar sind. Formal ist $G_1 \uplus G_2 \stackrel{\text{def}}{=} (V, \Sigma, \mathcal{P}, S)$ wie folgt definiert. Die Variablenmenge ist $V = V_1 \cup V_2 \cup \{S\}$ mit einem neuen Startsymbol $S \notin V_1 \cup V_2$. Die Regeln von $G_1 \uplus G_2$ sind die Regeln aus G_1 und G_2 sowie die beiden neuen Startregeln $S \to S_1$ und $S \to S_2$. Offenbar ist $G_1 \uplus G_2$ kontextfrei und $\mathcal{L}(G_1 \uplus G_2) = \mathcal{L}(G_1) \cup \mathcal{L}(G_2)$.

Die angegebene Konstruktion einer Grammatik für die Vereinigungssprache ist auch für beliebige Grammatiken (Typ 0 der Chomsky Hierarchie) und kontextsensitive Grammatiken geeignet. Genauer gilt:

Lemma 3.12 (Vereinigungsoperator für Grammatiken (Typ 0,1,2)). *Sind* G_1 *und* G_2 *vom Typ* $i \in \{0,1,2\}$*, dann ist auch* $G_1 \uplus G_2$ *vom Typ* i *und es gilt* $\mathcal{L}(G_1 \uplus G_2) = \mathcal{L}(G_1) \cup \mathcal{L}(G_2)$.

Konkatenation. Wie zuvor seien G_1, G_2 kontextfreie Grammatiken mit demselben Terminalalphabet Σ und disjunkten Variablenmengen. Das Ziel ist die Definition einer kontextfreien Grammatik $G_1 \circ G_2$, deren erzeugte Sprache gleich $\mathcal{L}(G_1) \circ \mathcal{L}(G_2)$ ist. Diese Grammatik

$G_1 \circ G_2$ erhält man, indem man alle Regeln von G_1 und G_2 zulässt und ein neues Startsymbol S einfügt, aus dem sich das Wort S_1S_2 ableiten lässt. Formal ist die Grammatik

$$G_1 \circ G_2 \stackrel{\text{def}}{=} (V, \Sigma, \mathcal{P}, S)$$

wie folgt definiert. Die Variablenmenge ist $V = V_1 \cup V_2 \cup \{S\}$ mit einem neuen Startsymbol $S \notin V_1 \cup V_2$. Das Produktionssystem setzt sich aus den Regeln in G_1 und in G_2 und der neuen Startregel $S \to S_1S_2$ zusammen. Offenbar ist mit G_1, G_2 auch die Grammatik $G_1 \circ G_2$ kontextfrei. Die Korrektheit der angegebenen Definition von $G_1 \circ G_2$ im Sinne von "$\mathcal{L}(G_1 \circ G_2) = \mathcal{L}(G_1) \circ \mathcal{L}(G_2)$" ist wie folgt einsichtig.

"$\supseteq$": Sei $w = w_1w_2$, wobei $w_1 \in \mathcal{L}(G_1)$ und $w_2 \in \mathcal{L}(G_2)$. Dann gibt es Herleitungen $S_1 \Rightarrow^* w_1$ in G_1 und $S_2 \Rightarrow^* w_2$ in G_2. Diese können zu einer Herleitung von w in $G_1 \circ G_2$ kombiniert werden:

$$S \Rightarrow S_1S_2 \Rightarrow^* w_1S_2 \Rightarrow^* w_1w_2 = w$$

Also ist $w \in \mathcal{L}(G_1 \circ G_2)$.

"$\subseteq$": Durch Induktion nach der Länge n einer Herleitung $S \Rightarrow y_1 \Rightarrow y_2 \Rightarrow \ldots \Rightarrow y_n$ in $G_1 \circ G_2$ kann gezeigt werden, dass für $n \geqslant 1$ das hergeleitete Wort $y_n \in (V_1 \cup V_2 \cup \Sigma)^*$ die Gestalt $y_n = w_1w_2$ hat, wobei w_1 und w_2 Wörter mit folgenden Eigenschaften sind:

- $w_1 \in (V_1 \cup \Sigma)^*$ und $S_1 \Rightarrow^* w_1$ in G_1
- $w_2 \in (V_2 \cup \Sigma)^*$ und $S_2 \Rightarrow^* w_2$ in G_2

Ist nun $y_n \in \Sigma^*$, so folgt $w_1 \in \mathcal{L}(G_1)$ und $w_2 \in \mathcal{L}(G_2)$ und somit $y_n \in \mathcal{L}(G_1) \circ \mathcal{L}(G_2)$. ■

Die angegebene Konstruktion $G_1 \circ G_2$ liefert zwar auch für Grammatiken vom Typ 0 oder 1 eine Grammatik des entsprechenden Typs, jedoch ist nicht garantiert, dass die akzeptierte Sprache mit $\mathcal{L}(G_1) \circ \mathcal{L}(G_2)$ übereinstimmt. Lediglich die Inklusion $\mathcal{L}(G_1) \circ \mathcal{L}(G_2) \subseteq \mathcal{L}(G_1 \circ G_2)$ kann für beliebige Grammatiken G_1 und G_2 garantiert werden, nicht aber die umgekehrte Inklusion. Besteht z.B. Grammatik G_1 aus der Regel $S_1 \to a$ und G_2 aus der Regel $aS_2 \to aa$, dann ist die erzeugte Sprache von G_2 leer und somit $\mathcal{L}(G_1) \circ \mathcal{L}(G_2) = \varnothing$. Andererseits ist

$$S \Rightarrow S_1S_2 \Rightarrow aS_2 \Rightarrow aa$$

eine Herleitung in $G_1 \circ G_2$. Also ist $aa \in \mathcal{L}(G_1 \circ G_2) \setminus (\mathcal{L}(G_1) \circ \mathcal{L}(G_2))$.

Kleeneabschluss. Sei $G = (V, \Sigma, \mathcal{P}, S)$ eine kontextfreie Grammatik. Eine kontextfreie Grammatik G^* für die Sprache $\mathcal{L}(G)^*$ erhält man, indem man G um ein neues Startsymbol S' erweitert, aus dem sich ε und SS' herleiten lassen. Formal ist

$$G^* \stackrel{\text{def}}{=} (V', \Sigma, \mathcal{P}_*, S_*),$$

wobei $V' = V \cup \{S_*\}$ mit $S_* \notin V$ und das Produktionssystem $\mathcal{P}_*$ aus den Regeln in $\mathcal{P}$ und den beiden neuen Startregeln $S_* \to \varepsilon$ und $S_* \to SS_*$ besteht. Es ist klar, dass mit G auch G^* kontextfrei ist. Ähnlich wie für den Konkatenationsoperator kann gezeigt werden, dass $\mathcal{L}(G^*) = \mathcal{L}(G)^*$, sofern G kontextfrei ist. Für Grammatiken vom Typ 0 oder Typ 1 ist der angegebene Operator für den Kleeneabschluss jedoch falsch.

Corollar 3.13. *Die Klasse der kontextfreien Sprachen ist unter Vereinigung, Konkatenation und Kleeneabschluss abgeschlossen.*

Die Klasse der kontextfreien Sprachen ist jedoch *nicht* abgeschlossen unter der Durchschnitts- und Komplementbildung. Dies lässt sich wie folgt begründen. Die Sprachen

$$L_1 = \{a^n b^n c^m : n, m \geqslant 1\}, \quad L_2 = \{a^n b^m c^m : n, m \geqslant 1\}$$

sind kontextfrei. Z.B. wird L_1 durch die CFG mit den Regeln

$$S \to DC \quad C \to c \mid Cc \quad D \to aDb \mid ab$$

erzeugt. Eine ähnliche Grammatik kann für L_2 angegeben werden. Die Durchschnittssprache

$$L_1 \cap L_2 = \{a^n b^n c^n : n \geqslant 1\}$$

ist jedoch *nicht* kontextfrei (wie zuvor gezeigt). Wäre die Klasse der kontextfreien Sprachen abgeschlossen unter der Komplementbildung, dann wäre sie auch unter der Durchschnittsbildung abgeschlossen. Dies folgt aus dem de Morganschen Gesetz

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

und der Abgeschlossenheit kontextfreier Sprachen unter der Vereinigung.

Satz 3.14. *Die Klasse der kontextfreien Sprachen ist nicht abgeschlossen unter Durchschnitt und Komplement.*

3.5 Kellerautomaten

Das Automatenmodell für reguläre Sprachen sind endliche Automaten. Diese haben keinen unbegrenzten Speicher und können höchstens Daten eines festen endlichen Bereichs in ihren Zuständen speichern. Für Sprachen wie $L = \{a^n b^n : n \geqslant 1\}$ ist dies jedoch nicht ausreichend, da ein Automat für L die Information über die genaue Anzahl an gelesenen a's speichern muss. Kellerautomaten erweitern endliche Automaten um ein unbegrenztes Speichermedium, nämlich einen Keller. Wir machen uns zunächst klar, warum für kontextfreie Sprachen genau Keller als Speichermedium geeignet sind und betrachten dazu eine weitere Normalform kontextfreier Grammatiken.

3.5.1 Die Greibach Normalform

Neben der Chomsky Normalform gibt es eine Reihe weiterer Normalformen für kontextfreie Grammatiken. Wir erwähnen hier kurz die Greibach Normalform, die wir dann als Ausgangspunkt für die Diskussion von Kellerautomaten und deren Äquivalenz zu CFG nutzen werden.

Definition 3.15 (Greibach Normalform). Sei G eine CFG. G ist in Greibach Normalform, falls alle Produktionen in G von der Form

$$A \to aB_1B_2 \dots B_k$$

sind, wobei $a \in \Sigma$, $B_1, B_2, \dots, B_k \in V$ und $k \in \mathbb{N}$. Der Fall $k = 0$, also Regeln der Form $A \to a$, ist zugelassen. ■

Z.B. ist durch $S \to aB \mid aSB$, $B \to b$ eine CFG in Greibach Normalform für die Sprache $\{a^n b^n : n \geqslant 1\}$ gegeben. Weiter ist jede reguläre Grammatik, die keine ε-Regeln enthält, in Greibach Normalform. Wie die Chomsky Normalform ist auch die Greibach Normalform universell für kontextfreie Sprachen, die das leere Wort nicht enthalten. Wir zitieren dieses Ergebnis ohne Beweis:

Satz 3.16. *Zu jeder CFG mit $\varepsilon \notin \mathcal{L}(G)$ gibt es eine äquivalente CFG in Greibach Normalform.* (ohne Beweis)

Linksableitungen für Grammatiken in Greibach Normalform haben in i Schritten ein Wort der Form $x_i = a_1 \dots a_i A_1 A_2 \dots A_k$ erzeugt. Dabei sind $a_1, \dots, a_i$ Terminalzeichen. Für den jeweils nächsten Ableitungsschritt ist nur die Kenntnis der ersten Variablen A_1 entscheidend. Diese wird durch ein Wort $a_{i+1} B_1 B_2 \dots B_\ell$ ersetzt. Das hergeleitete Wort hat nun die Form

$$x_{i+1} = a_1 \dots a_i a_{i+1} B_1 B_2 \dots B_\ell A_2 \dots A_k.$$

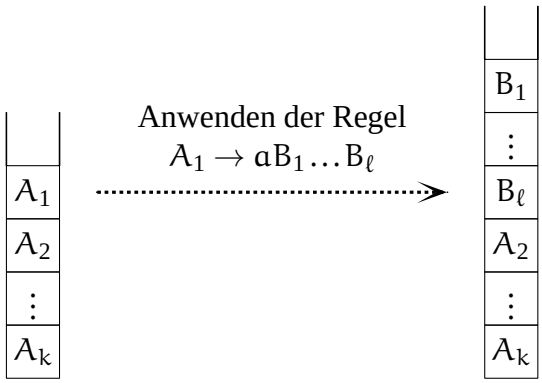
Der nächste Linksableitungsschritt muss nun eine Regel für Variable B_1 einsetzen, etwa $B_1 \to a_{i+2} C_1 C_2 \dots C_m$. Man erhält das Wort

$$x_{i+2} = a_1 \dots a_i a_{i+1} a_{i+2} C_1 C_2 \dots C_m B_2 \dots B_\ell A_2 \dots A_k,$$

für das nun C_1 durch Anwenden einer Regel zu ersetzen ist. Das Suffix bestehend aus den noch zu ersetzenden Variablen kann durch Anwenden von Terminalregeln, also Regeln des Typs $A \to a$, verkürzt werden. Wird etwa für x_{i+2} die Terminalregel $C_1 \to a_{i+3}$ eingesetzt, so hat das nun hergeleitete Wort die Form

$$x_{i+3} = a_1 \dots a_i a_{i+1} a_{i+2} a_{i+3} C_2 \dots C_m B_2 \dots B_\ell A_2 \dots A_k,$$

so dass als nächstes eine Regel für Variable C_2 eingesetzt werden muss. Linksableitungen behandeln also die Variablen im LIFO-Prinzip.[10]



Zur Verwaltung der jeweils hergeleiteten Teilwörter, bestehend aus den noch zu ersetzenden Variablen, scheint daher ein Keller geeignet zu sein. Für die Auswahl, welche der Regeln $A_1 \to a_{i+1} B_1 \dots B_\ell$ angewandt werden, setzen wir das Konzept von Nichtdeterminismus ein. Diese Überlegungen führen zu den nichtdeterministischen Kellerautomaten.

[10] Die Abkürzung LIFO steht für "last-in, first-out".

3.5.2 Nichtdeterministische Kellerautomaten

Ein Kellerautomat besteht aus einer endlichen Kontrolle, einem Eingabeband, auf das nur lesende Zugriffe (von links nach rechts) zulässig sind sowie einem Keller. Siehe Abbildung 27 auf Seite 91. Der Keller ist unbeschränkt und kann beliebig viele Zeichen eines festen Kelleralphabets speichern. Die endliche Kontrolle besteht aus (Kontroll-)Zuständen eines endlichen Zustandsraums. Ähnlich wie in ε-NFA kann in jedem Schritt entweder ein Eingabezeichen gelesen werden oder ein spontaner ε-Übergang stattfinden. Im Gegensatz zu endlichen Automaten hängt die Übergangsfuktion nicht nur vom aktuellen Zustand und Zeichen unter dem Lesekopf ab, sondern auch von dem obersten Kellersymbol. Verbunden mit jedem Übergang ist eine Pop-Operation (Entfernen des obersten Kellersymbols) sowie eine (eventuell leere) Folge von Push-Operationen.

Definition 3.17 (Nichtdeterministischer Kellerautomat (NKA)). Ein NKA ist ein Tupel $\mathcal{K} = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ bestehend aus

- einer endlichen Menge Q von Zuständen,
- einem Eingabealphabet Σ,
- einem Kelleralphabet Γ,
- einem Anfangszustand $q_0 \in Q$,
- einem Kellerstartsymbol $\# \in \Gamma$,
- einer Menge $F \subseteq Q$ von Endzuständen,
- einer totalen Übergangsfunktion $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$, so dass die Mengen $\delta(q, a, A)$ und $\delta(q, \varepsilon, A)$ endlich sind für alle $q \in Q$, $a \in \Sigma$ und $A \in \Gamma$.

Wie zuvor verwenden wir Kleinbuchstanben $a, b, c, \ldots$ am Anfang des Alphabets für Symbole aus Σ. Grossbuchstaben $A, B, C, \ldots$ dienen als Bezeichner für Kellersymbole, d.h., Elemente aus dem Kelleralphabet Γ. Ausnahmen sind Beispiele für NKA, in denen manche Kellersymbole direkten Bezug zu den Zeichen in Σ haben, für die wir dann dasselbe Symbol verwenden. ■

Die intuitive Arbeitsweise eines Kellerautomaten ist wie folgt. Initial steht das Eingabewort auf dem Eingabeband; der Keller enthält nur das Symbol #. Der Automat startet die Berechnung in dem Anfangszustand q_0. Das weitere Verhalten ist durch die Übergangsfunktion bestimmt. Ist q der aktuelle Zustand, a das Eingabezeichen unter dem Lesekopf und A das oberste Kellersymbol, dann wählt der Automat nichtdeterministisch ein Paar

$$(p, z) \in \delta(q, a, A) \cup \delta(q, \varepsilon, A).$$

Die Paare $(p, z) \in \delta(q, a, A)$ stehen für die Bearbeitung des nächsten Eingabezeichens; in diesem Fall wird der Lesekopf um eine Position nach rechts verschoben. Die Paare $(p, z) \in \delta(q, \varepsilon, A)$ repräsentieren ε-Transitionen, in denen kein Eingabezeichen gelesen wird; in diesem Fall bleibt die Position des Lesekopfs unverändert. Ist $(p, z) \in \delta(q, a, A) \cup \delta(q, \varepsilon, A)$ die gewählte Alternative, dann wechselt der Kellerautomat in Zustand p und ersetzt das oberste Kellersymbol A durch das Wort z. Ist $z = \varepsilon$, dann wird das oberste Kellersymbol lediglich entfernt; es wird jedoch nichts auf den Keller gelegt. In jedem Schritt werden also genau eine Pop-Operation und 0 oder mehrere Push-Operationen ausgeführt.

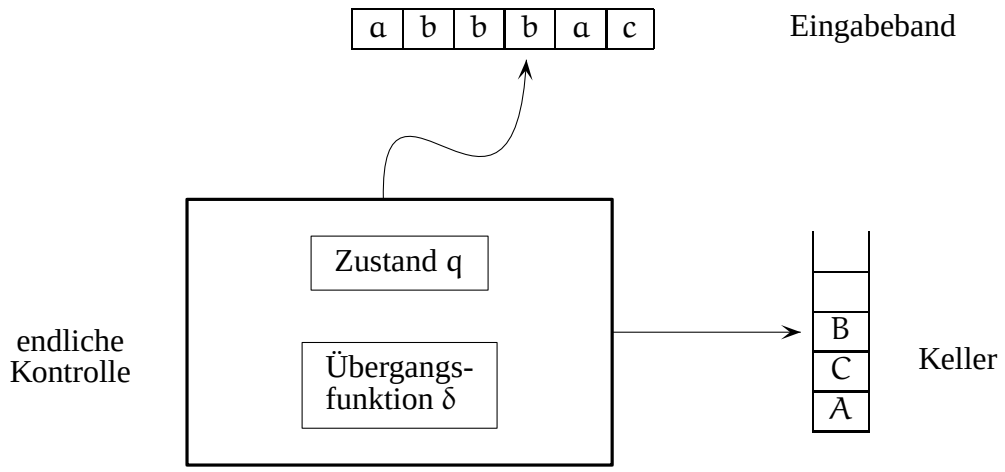


Abbildung 27: Schema eines NKA

Konfigurationen und Konfigurationswechsel. Für eine Formalisierung des schrittweisen Verhaltens benötigen wir den Begriff einer Konfiguration. Diese setzt sich aus dem aktuellen (Kontroll-)Zustand, dem aktuellen Kellerinhalt und der Information über die Position des Lesekopfs zusammen. Letzteres wird durch das noch zu lesende Suffix der Eingabe dargestellt. Sei $\mathcal{K}$ wie oben. Eine *Konfiguration* für $\mathcal{K}$ ist ein Tripel (q, u, y) bestehend aus einem Zustand $q \in Q$, einem Wort $u \in \Sigma^*$ über dem Eingabealphabet und einem Wort $y \in \Gamma^*$ über dem Kelleralphabet. Intuitiv steht q für den aktuellen Zustand, u für den noch nicht gelesenen Teil der Eingabe und y für den Kellerinhalt. Das erste Zeichen von y ist das oberste Kellersymbol. Die *Anfangskonfiguration* für das Eingabewort $w \in \Sigma^*$ ist das Tripel $(q_0, w, \#)$. Der NKA aus Abbildung 27 auf Seite 91 befindet sich in der Konfiguration (q, bac, BCA).

Sei $\mathcal{K}$ ein NKA, wie zuvor. *Konf*$(\mathcal{K}) = Q \times \Sigma^* \times \Gamma^*$ bezeichnet die Menge aller Konfigurationen. Die Konfigurationsrelation $\vdash_{\mathcal{K}}$ (oder kurz $\vdash$) formalisiert den Übergang von einer zur nächsten Konfiguration. Sie ist also eine binäre Relation auf *Konf*$(\mathcal{K})$ und wird als die kleinste Relation (bezüglich der Inklusion) definiert, so dass folgende Eigenschaften (1) und (2) für alle Zustände $q, p \in Q$, Kellersymbole $A \in \Gamma$, und Wörter $u \in \Sigma^*$, $y, z \in \Gamma^*$ gelten:

(1) Aus $(p, z) \in \delta(q, a, A)$ und $a \in \Sigma$ folgt $(q, au, Ay) \vdash (p, u, zy)$.

(2) Aus $(p, z) \in \delta(q, \varepsilon, A)$ folgt $(q, u, Ay) \vdash (p, u, zy)$.

Bedingung (1) steht für das Lesen eines Eingabezeichens und Bedingung (2) für eine ε-Transition, die – wie in ε-NFA – unabhängig vom Zeichen unter dem Lesekopf stattfinden kann. In beiden Fällen wird vorausgesetzt, dass A das oberste Kellersymbol der aktuellen Konfiguration ist. Ist der Keller leer, d.h., ist die aktuelle Konfiguration von der Form (q, u, ε), so ist weder (1) noch (2) anwendbar. Daher gilt

$$(q, u, \varepsilon) \nvdash \text{ für alle } q \in Q \text{ und } u \in \Sigma^*.$$

Kellerautomaten halten also stets an, wenn der Keller leer ist. Weiter hält ein Kellerautomat stets dann an, wenn $\delta(q, a, A) = \delta(q, \varepsilon, A) = \varnothing$ für den aktuellen Zustand q, das Zeichen a unter dem Lesekopf und das Topelement A des Kellers.

Das Symbol $\vdash_{\mathcal{K}}^{*}$ (oder kurz $\vdash^{*}$) bezeichnet die transitive, reflexive Hülle von $\vdash$. Es gilt also $(q,u,y) \vdash^{*} (q',u',y')$ gilt genau dann, wenn die Konfiguration (q,u,y) durch 0 oder mehrere Schritte in (q',u',y') überführt werden kann, wobei jeder Konfigurationswechsel gemäß $\vdash$ als "Schritt" zählt. Jede Konfigurationsfolge

$$(q_0,w,\#) \vdash (q_1,u_1,y_1) \vdash (q_2,u_2,y_2) \vdash \ldots$$

wird ein *Lauf* von $\mathcal{K}$ für w genannt, sofern die Folge entweder maximal ist (also unendlich ist oder in einer Konfiguration endet, die keine Folgekonfiguration hat) oder in einer Konfiguration (q_m,u_m,y_m) endet, für die $u_m = \varepsilon$ gilt (d.h., das Eingabewort wurde komplett gelesen). Manchmal sprechen wir auch von einer *Berechnung* von $\mathcal{K}$ für w.

Bemerkung 3.18 (Läufe in NFA vs. Läufe in NKA.). Für NFA konnten wir auf das Konzept von Konfigurationen verzichten und Läufe für ein Eingabewort $a_1\,a_2 \ldots a_n$ als Zustandsfolgen $q_0\,q_1\,\ldots$ bestehend aus höchstens $n+1$ Zuständen definieren. Wenn keine ε-Transitionen vorhanden sind, dann ist klar, dass im $(i+1)$-ten Zustand q_i die ersten i Zeichen $a_1,\ldots,a_i$ bereits gelesen wurden und $a_{i+1}\,a_{i+2} \ldots a_n$ das noch zu lesende Suffix der Eingabe ist. Dies gilt zwar auch für NKA ohne ε-Transitionen, also NKA mit $\delta(q,\varepsilon,A) = \varnothing$ für alle Zustände $q \in Q$ und Kellersymbole $A \in \Gamma$, jedoch hängt das Verhalten des NKA ab Zustand $q = q_i$ nicht nur von den restlichen Eingabezeichen $a_{i+1}\,a_{i+2} \ldots a_n$ ab, sondern auch von dem Kellerinhalt.

Die in Definition 2.21 auf Seite 38 angegebene Definition eines Laufs für $w = a_1 a_2 \ldots a_n \in \Sigma^*$ in einem ε-NFA $\mathcal{M}$ kann in Analogie zu der oben angegebenen Definition eines Laufs in einem NKA umformuliert werden. Die Konfigurationen eines ε-NFA $\mathcal{M} = (Q,\Sigma,\delta,Q_0,F)$ sind nun Paare $(q,u) \in Q \times \Sigma^*$. Die Konfigurationsrelation $\vdash_{\mathcal{M}}$ ist die kleinste binäre Relation auf $Q \times \Sigma^*$, so dass (1) $(q,au) \vdash (p,u)$, falls $p \in \delta(q,a)$ und (2) $(q,u) \vdash (p,u)$, falls $p \in \delta(q,\varepsilon)$. Als Lauf für $w = a_1 a_2 \ldots a_n \in \Sigma^*$ in $\mathcal{M}$ kann nun jede Folge

$$(q_0,u_0) \vdash (q_1,u_1) \vdash (q_2,u_2) \vdash \ldots$$

von Konfigurationen, die in einer Konfiguration (q_0,u_0) mit $q_0 \in Q_0$ und $u_0 = w$ beginnt, bezeichnet werden, sofern die betreffende Folge entweder unendlich ist oder in einer Konfiguration (q_m,u_m) endet, die entweder keine Folgekonfiguration besitzt oder für die $u_m = \varepsilon$ gilt. Die in Definition 2.21 angegebene Definition eines Laufs ergibt sich dann durch die Projektionen auf die Zustände in allen solchen endlichen Konfigurationsfolgen. Die von $\mathcal{M}$ akzeptierte Sprache $\mathcal{L}(\mathcal{M})$ besteht aus genau solchen Wörtern $w \in \Sigma^*$, für die es einen Anfangszustand $q_0 \in Q_0$ und einen Zustand $p \in F$ gibt, so dass $(q_0,w) \vdash^{*} (p,\varepsilon)$. Wie für NKA bezeichnet $\vdash^{*}$ die reflexive, transitive Hülle von $\vdash$. ■

Akzeptanzverhalten. Für Kellerautomaten unterscheidet man zwei Varianten der akzeptierten Sprache. Diese basieren entweder auf der Akzeptanz *über Endzustände* oder *bei leerem Keller*. Für die Akzeptanz über Endzustände fordert man, dass nach dem Lesen der kompletten Eingabe (und nach eventuellen ε-Transitionen) ein Endzustand erreicht ist; unabhängig vom Kellerinhalt. In diesem Fall sind genau die Konfigurationen der Form (q,ε,y) mit $q \in F$ akzeptierend. Dagegen stellt die Akzeptanz bei leerem Keller die Forderung, dass der Keller leer und die Eingabe zu Ende gelesen ist. In diesem Fall sind genau die Konfigurationen $(q,\varepsilon,\varepsilon)$ akzeptierend, wobei q ein beliebiger Zustand ist. In beiden Akzeptanzvarianten sind sämtliche Berechnungen, welche enden bevor die Eingabe zu Ende gelesen ist, verwerfend.

Definition 3.19 (Die akzeptierten Sprachen eines NKA). Sei $\mathcal{K} = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ ein NKA. Die *bei leerem Keller akzeptierte* Sprache ist

$$\mathcal{L}_\varepsilon(\mathcal{K}) \;=\; \big\{w \in \Sigma^* : (q_0, w, \#) \vdash^* (q, \varepsilon, \varepsilon) \text{ für ein } q \in Q\big\}.$$

Die *über Endzustände akzeptierte Sprache* ist

$$\mathcal{L}(\mathcal{K}) \;=\; \big\{w \in \Sigma^* : (q_0, w, \#) \;\vdash^*\; (q, \varepsilon, y) \text{ für ein } y \in \Gamma^*, q \in F\big\}.$$

■

Für die Akzeptanz bei leerem Keller ist die Endzustandsmenge völlig unerheblich. Für NKA mit Akzeptanz bei leerem Keller kann man daher auf die Komponente F völlig verzichten. Wir schreiben sie als Tupel $\mathcal{K} = (Q, \Sigma, \Gamma, \delta, q_0, \#)$.

Eine Charakterisierung der akzeptierten Sprache eines Kellerautomaten durch *akzeptierender Läufe* ist wie für NFA möglich. Unter dem Akzeptanzkriterium "über Endzustände" wird ein Lauf für das Eingabewort w

$$(q_0, w, \#) \;\vdash\; (q_1, u_1, y_1) \;\vdash\; (q_2, u_2, y_2) \;\vdash\; \ldots \vdash (q_m, u_m, y_m)$$

akzeptierend genannt, falls $q_m \in F$ (ein Endzustand wurde erreicht) und $u_m = \varepsilon$ (die Eingabe wurde komplett gelesen). Für NKA mit Akzeptanz bei leerem Keller sind genau solche Läufe für w

$$(q_0, w, \#) \;\vdash\; (q_1, u_1, y_1) \;\vdash\; (q_2, u_2, y_2) \;\vdash\; \ldots \vdash (q_m, u_m, y_m)$$

akzeptierend, für die gilt: $u_m = \varepsilon$ (die Eingabe wurde komplett gelesen) und $y_m = \varepsilon$ (der Keller ist leer).

Beispiel 3.20 (NKA). Die Sprache $L = \{a^n b^n : n \geqslant 1\}$ wird durch einen Kellerautomaten $\mathcal{K}$ mit zwei Zuständen q_a und q_b akzeptiert, dessen Arbeitsweise für das Eingabewort $w \in \{a, b\}^*$ informell wie folgt beschrieben werden kann. $\mathcal{K}$ startet in Zustand q_a. Ist die Eingabe leer oder das erste Eingabezeichen ein b, dann verwirft $\mathcal{K}$ sofort. Andernfalls liest $\mathcal{K}$ die führenden a's von w und legt diese im Keller ab. Mit Lesen des ersten a's wird das Anfangskellersymbol # überschrieben. Sobald das erste b gelesen wird, wechselt $\mathcal{K}$ in den Zustand q_b. In Zustand q_b wird für jedes gelesene b ein a aus dem Keller entfernt. Befindet sich $\mathcal{K}$ in Zustand q_b, so verwirft $\mathcal{K}$ das Eingabewort genau in folgenden Fällen:

- In Zustand q_b wird ein a gelesen. Dann ist w von der Gestalt $a^n b^m a x$, wobei $1 \leqslant m < n$ und $x \in \{a, b\}^*$, und somit $w \notin L$.
- Der Keller ist leer und die Eingabe ist noch nicht zu Ende gelesen. Dann ist w von der Form $a^n b^n u$, wobei $u \in \{a, b\}^+$, und somit $w \notin L$.

Die präzise Darstellung von $\mathcal{K}$ ist wie folgt:

$$\mathcal{K} \;=\; (\{q_a, q_b\}, \{a, b\}, \{a, b, \#\}, \delta, q_a, \#, \{q_b\}),$$

wobei

$$\begin{array}{lcllcl}
\delta(q_a, a, \#) & = & \{(q_a, a)\} & \qquad \delta(q_a, a, a) & = & \{(q_a, aa)\} \\
\delta(q_a, b, a) & = & \{(q_b, \varepsilon)\} & \qquad \delta(q_b, b, a) & = & \{(q_b, \varepsilon)\}
\end{array}$$

und $\delta(\cdot) = \varnothing$ in allen verbleibenden Fällen. Beispielsweise gilt

$$\begin{aligned}
(q_a, aabb, \#) &\vdash (q_a, abb, a) \vdash (q_a, bb, aa) \vdash (q_b, b, a) \vdash (q_b, \varepsilon, \varepsilon) \\
(q_a, aaba, \#) &\vdash (q_a, aba, a) \vdash (q_a, ba, aa) \vdash (q_b, a, a) \\
(q_a, aabbb, \#) &\vdash (q_a, abbb, a) \vdash (q_a, bbb, aa) \vdash (q_b, bb, a) \vdash (q_b, b, \varepsilon)
\end{aligned}$$

und $(q_a, bbaa, \#) \nvdash$ bzw. $(q_a, \varepsilon, \#) \nvdash$. Für den formalen Nachweis, dass $L = \mathcal{L}_\varepsilon(\mathcal{K})$ ist einerseits zu zeigen, dass alle nicht-leeren Wörter der Form $a^n b^n$ von $\mathcal{K}$ akzeptiert werden. Dies erfolgt durch Angabe akzeptierender Läufe:

$$(q_a, a^n b^n, \#) \vdash^* (q_a, b^n, a^n) \vdash (q_b, b^{n-1}, a^{n-1}) \vdash^* (q_b, \varepsilon, \varepsilon)$$

Umgekehrt ist zu zeigen, dass keine anderen Wörter von $\mathcal{K}$ akzeptiert werden. Zunächst betrachten wir das leere Wort. NKA $\mathcal{K}$ verwirft das Eingabewort ε, da es in Zustand q_a keine ε-Transitionen gibt und daher $(q_a, \varepsilon, \#) \nvdash$ gilt. Alle Wörter, die mit b beginnen, werden ebenfalls von $\mathcal{K}$ sofort verworfen, da es für Zustand q_a weder eine ε-Transitionen noch eine b-Transition gibt, sofern # das oberste Kellersymbol ist. D.h., es gilt $(q_a, bx, \#) \nvdash$ für alle $x \in \{a, b\}^*$. Letztendlich müssen noch Wörter der Form $a^n b^m a x$ mit $1 \leqslant m < n$ und $x \in \{a, b\}^*$ sowie Wörter der Form $a^n b^n u$ mit $n \geqslant 1$ und $u \in \{a, b\}^+$. Für Wörter des ersten Typs verwirft $\mathcal{K}$, da das Verhalten von $\mathcal{K}$ durch Läufe der Form

$$(q_a, a^n b^m a x, \#) \vdash^* (q_a, b^m a x, a^n) \vdash (q_b, b^{m-1} a x, a^{n-1}) \vdash^* (q_b, a x, a^{n-m}) \nvdash$$

bestimmt ist. Im zweiten Fall sind die Läufe ebenfalls verwerfend:

$$(q_a, a^n b^n u, \#) \vdash^* (q_a, b^n u, a^n) \vdash (q_b, b^{n-1} u, a^{n-1}) \vdash^* (q_b, u, \varepsilon) \nvdash,$$

da der Keller in der letzten Konfiguration leer ist, aber die Eingabe noch nicht komplett gelesen ist. Die von $\mathcal{K}$ mit dem Endzustand q_b akzeptierte Sprache stimmt jedoch nicht mit L überein. Beispielsweise ist dann

$$(q_a, aab, \#) \vdash (q_a, ab, a) \vdash (q_a, b, aa) \vdash (q_b, \varepsilon, a)$$

ein akzeptierender Lauf für aab in $\mathcal{K}$, da dieser in einer Konfiguration endet, in welcher der aktuelle Zustand (nämlich q_b) ein Endzustand ist und die Eingabe komplett gelesen wurde. Daher gilt

$$aab \in \mathcal{L}(\mathcal{K}) \setminus \mathcal{L}_\varepsilon(\mathcal{K}).$$

Man kann zeigen, dass $\mathcal{L}(\mathcal{K}) = \{a^n b^m : n \geqslant m \geqslant 1\}$. ∎

Beispiel 3.21 (NKA). Der NKA in dem vorangegangenen Beispiel ist deterministisch, da er keine ε-Transitionen hat und die Übergangsfunktion für jedes Tripel $(q, a, A) \in Q \times \Sigma \times \Gamma$ als höchstens einelementige Menge definiert ist. Insbesondere ist in jeder Konfiguration höchstens ein Übergang möglich. Wir geben nun ein Beispiel für einen NKA mit "echtem Nichtdeterminismus" für die Sprache

$$L = \{ ww^R : w \in \{0,1\}^* \}.$$

Dabei steht w^R für das inverse (gespiegelte) Wort von w. Ist also $w = a_1 a_2 \dots a_n$, so ist $w^R = a_n \dots a_2 a_1$. (Der Buchstabe "R" steht für "reverse".)

Die Arbeitsweise des NKA für gegebenes Eingabewort $x \in \{0,1\}^*$, für welches zu prüfen ist, ob x die Form ww^R für ein $w \in \{0,1\}^*$ hat, basiert auf folgenden Ideen. Wir verwenden zwei Zustände q^+ und q^-. In Zustand q^+ lesen wir ein Präfix des Eingabeworts x und legen die gelesenen Symbole zeichenweise im Keller ab. Zu jedem Zeitpunkt steht also das Wort v^R im Keller, wenn v das bereits gelesene Präfix der Eingabe x ist. Zustand q^- dient dazu, zu prüfen, ob der Rest der Eingabe mit dem im Keller abgelegten Wort v^R übereinstimmt. Stimmt das gelesene Eingabezeichen mit dem obersten Kellerzeichen überein, dann wird die Entscheidung, ob von Zustand q^+ in den Zustand q^- zu wechseln ist, *nichtdeterministisch* gefällt. Wir definieren den NKA mit Akzeptanz bei leerem Keller wie folgt:

$$\mathcal{K} \;=\; (\{q^+, q^-\}, \{0,1\}, \{0,1,\#\}, \delta, q^+, \#).$$

Die Übergangsfunktion δ ist wie folgt definiert. Seien a, $b \in \{0,1\}$, $a \neq b$.

$$\begin{array}{lcllcl}
\delta(q^+, a, a) &=& \{(q^+, aa), (q^-, \varepsilon)\} & & & \\
\delta(q^+, a, b) &=& \{(q^+, ab)\} & & \delta(q^-, a, a) &= \{(q^-, \varepsilon)\} \\
\delta(q^+, a, \#) &=& \{(q^+, a\#)\} & & \delta(q^-, \varepsilon, \#) &= \{(q^-, \varepsilon)\} \\
\delta(q^+, \varepsilon, \#) &=& \{(q^-, \varepsilon)\} & &
\end{array}$$

und $\delta(\cdot) = \varnothing$ in allen anderen Fällen. Mögliche Berechnungen für das Wort $w = 0110$ sind:

$$\begin{array}{lllllll}
(q^+, 0110, \#) & \vdash & (q^+, 110, 0\#) & \vdash & (q^+, 10, 10\#) & \vdash & (q^-, 0, 0\#) \\
 & & & \vdash & (q^-, \varepsilon, \#) & \vdash & (q^-, \varepsilon, \varepsilon)
\end{array}$$

$$\begin{array}{lllllll}
(q^+, 0110, \#) & \vdash & (q^+, 110, 0\#) & \vdash & (q^+, 10, 10\#) & \vdash & (q^+, 0, 110\#) \\
 & & & & & \vdash & (q^+, \varepsilon, 0110\#)
\end{array}$$

$$(q^+, 0110, \#) \;\vdash\; (q^-, 0110, \varepsilon)$$

Die erste Berechnung ist akzeptierend; die zweite und dritte nicht.

Nun zur Korrektheit. Für den Nachweis, dass $L \subseteq \mathcal{L}_\varepsilon(\mathcal{K})$ ist zu zeigen, dass alle Wörter der Form ww^R einen akzeptierenden Lauf in $\mathcal{K}$ haben. Für $\varepsilon = \varepsilon\varepsilon^R$ stellt der Übergang $\delta(q^+, \varepsilon, \#) = \{(q^-, \varepsilon)\}$ sicher, dass für das leere Eingabewort der Keller in einem Schritt geleert wird:

$$(q^+, \varepsilon, \#) \;\vdash\; (q^-, \varepsilon, \varepsilon)$$

Sei nun $w = va$, wobei $a \in \Sigma$ und $v \in \Sigma^*$. Dann gibt es einen akzeptierenden Lauf für ww^R folgender Form:

$$\begin{array}{lllllll}
(q^+, ww^R, \#) & \vdash^* & (q^+, w^R, w^R\#) & = & (q^+, av^R, av^R\#) & & \\
 & \vdash & (q^-, v^R, v^R\#) & \vdash^* & (q^-, \varepsilon, \#) & \vdash & (q^-, \varepsilon, \varepsilon)
\end{array}$$

Für die Inklusion $\mathcal{L}_\varepsilon(\mathcal{K}) \subseteq L$ ist zu zeigen, dass alle Wörter x, die einen akzeptierenden Lauf in $\mathcal{K}$ haben, von der Gestalt ww^R sind. Wegen $\varepsilon \in L$ ist nur der Fall $x \neq \varepsilon$ von Interesse. Wir benutzen folgende Hilfsaussagen:

(1) Aus $(q^-, u, v\#) \vdash^* (q, \varepsilon, \varepsilon)$ folgt $q^- = q$ und $u = v$. Dabei sind $u, v \in \{0,1\}^*$.

(2) Aus $(q^+, x, \#) \vdash^* (q^-, \varepsilon, \varepsilon)$ mit $x \neq \varepsilon$ folgt, dass die Gestalt $x = zaau$ hat, wobei $z, u \in \{0,1\}^*$ und $a \in \{0,1\}$.

Hilfsaussage (1) ergibt sich durch Inspektion der Übergänge für Zustand q^-. Hilfsaussage (2) ist wie folgt einsichtig. Mit allen Transitionen von q^+, bei denen $\mathcal{K}$ im Anfangszustand q^+ bleibt, werden Push-Operationen durchgeführt, bei denen das gelesene Zeichen a auf den Keller gelegt wird. Einzige Ausnahme ist die ε-Transition für Zustand q^+ und oberstes Kellersymbol #. Diese ε-Transition ist zwar in der Anfangskonfiguration ausführbar, führt dann jedoch zu der terminalen, verwerfenden Konfiguration (q^+, x, ε). In akzeptierenden Läufen für ein nichtleeres Eingabewort x kann diese ε-Transitionen nicht eingesetzt werden. Eine Konfiguration, in der der Keller leer und die Eingabe komplett gelesen ist, ist nur dann von der Startkonfiguration $(q^+, x, \#)$ erreichbar, wenn $\mathcal{K}$ nach Einlesen eines Präfixes von x in Zustand q^- wechselt. Dies ist aber nur dann möglich, wenn x die Form $zaau$ hat, wobei $z, u \in \{0,1\}^*$ und

$$(q^+, x, \#) \vdash^* (q^+, au, az^R\#) \vdash (q^-, u, z^R\#) \vdash^* (q^-, \varepsilon, \varepsilon)$$

Aus Aussage (1) folgt nun $u = z^R$ und somit $x = ww^R$, wobei $w = za$. ∎

Jeder ε-NFA kann als Sonderfall eines NKA mit Akzeptanz über Endzustände aufgefasst werden. Ist nämlich $\mathcal{M} = (Q, \Sigma, \delta, \{q_0\}, F)$ ein ε-NFA mit eindeutigem Anfangszustand q_0, so erhält man einen NKA $\mathcal{K}_\mathcal{M}$ mit $\mathcal{L}(\mathcal{K}_\mathcal{M}) = \mathcal{L}(\mathcal{M})$ wie folgt:

$$\mathcal{K}_\mathcal{M} \stackrel{\text{def}}{=} (Q, \Sigma, \{\#\}, \delta', q_0, \#, F), \quad \text{wobei} \quad \delta'(q, a, \#) = \big\{(p, \#) : p \in \delta(q, a)\big\}$$

Die Annahme, dass der vorliegende ε-NFA einen eindeutigen Anfangszustand hat, ist keine Einschränkung, da jeder ε-NFA in einen äquivalenten ε-NFA mit genau einem Anfangszustand transformiert werden kann. Ebenso hätten wir in der Definition von Kellerautomaten eine Menge von Anfangszuständen zulassen können. Hinsichtlich der Ausdrucksstärke von NKA ist dies irrelevant.

Äquivalenz der Akzeptanzbedingungen

Es ist klar, dass die Sprachen $\mathcal{L}(\mathcal{K})$ und $\mathcal{L}_\varepsilon(\mathcal{K})$ i.A. nicht übereinstimmen. Z.B. wenn $F = \varnothing$, dann ist stets $\mathcal{L}(\mathcal{K}) = \varnothing$; während $\mathcal{L}_\varepsilon(\mathcal{K}) \neq \varnothing$ möglich ist. Dennoch sind die beiden Arten von Akzeptanzbedingungen äquivalent. Wir zeigen dies durch gegenseitige Simulationen.

Satz 3.22 (Von $\mathcal{K}_\varepsilon$ zu $\mathcal{K}_F$). *Zu jedem NKA $\mathcal{K}_\varepsilon$ mit Akzeptanz bei leerem Keller gibt es einen NKA $\mathcal{K}_F$ mit Akzeptanz über Endzustände, so dass $\mathcal{L}(\mathcal{K}_\varepsilon) = \mathcal{L}(\mathcal{K}_F)$.*

Beweis. Sei $\mathcal{K}_\varepsilon = (Q, \Sigma, \Gamma, \delta, q_0, \#)$ ein NKA mit Akzeptanz bei leerem Keller. Wir entwerfen einen NKA $\mathcal{K}_F$ so, dass $\mathcal{K}_F$ sich schrittweise wie $\mathcal{K}_\varepsilon$ verhält und – sobald $\mathcal{K}_\varepsilon$ eine akzeptierende Konfiguration $(q, \varepsilon, \varepsilon)$ erreicht – in einen speziellen Zustand q_F überwechselt, in dem $\mathcal{K}_F$ akzeptierend anhält. Auch hier verwenden wir ein zusätzliches Kellersymbol \$, das während der Simulation ganz unten im Keller von $\mathcal{K}_F$ liegt. Dieses benötigen wir, um zu verhindern, dass $\mathcal{K}_F$ verwerfend anhält, wenn $\mathcal{K}_\varepsilon$ akzeptiert. Wir definieren

$$\mathcal{K}_F \stackrel{\text{def}}{=} (Q \cup \{q_0', q_F\}, \Sigma, \Gamma \cup \{\$\}, \delta', q_0', \#, \{q_F\}),$$

wobei $\$ \notin \Gamma$, $q_0', q_F \notin Q$ und $q_0' \neq q_F$. Die Übergangsfunktion δ' ist wie folgt definiert. Sei $a \in \Sigma$, $A \in \Gamma$ und $q \in Q$.

$$\begin{array}{lcllcl} \delta'(q_0', \varepsilon, \#) & = & \{(q_0, \#\$)\} & \qquad \delta'(q, a, A) & = & \delta(q, a, A) \\ \delta'(q, \varepsilon, \$) & = & \{(q_F, \varepsilon)\} & \qquad \delta'(q, \varepsilon, A) & = & \delta(q, \varepsilon, A) \end{array}$$

In allen verbleibenden Fällen ist $\delta'(\cdot) = \varnothing$. Man beachte, dass $\mathcal{K}_F$ genau dann in den Endzustand q_F gelangen kann, wenn das Symbol \$ das oberste Kellersymbol ist; also wenn der Keller von $\mathcal{K}_\varepsilon$ leer ist. Ist in diesem Fall die Eingabe zu Ende gelesen, akzeptieren $\mathcal{K}_\varepsilon$ und $\mathcal{K}_F$. Andernfalls verwerfen $\mathcal{K}_\varepsilon$ und $\mathcal{K}_F$ das Eingabewort. Die Entscheidung, wann $\mathcal{K}_F$ in den Zustand q_F übergeht, wird *deterministisch* gefällt. Diese Beobachtung wird später von Bedeutung sein, wenn wir deterministische Kellerautomaten betrachten. Es bleibt noch $\mathcal{L}(\mathcal{K}_F) = \mathcal{L}_\varepsilon(\mathcal{K}_\varepsilon)$ nachzuweisen. Wir zeigen zuerst $\mathcal{L}(\mathcal{K}_F) \supseteq \mathcal{L}_\varepsilon(\mathcal{K}_\varepsilon)$. Sei also $w \in \mathcal{L}_\varepsilon(\mathcal{K}_\varepsilon)$. Dann gilt für ein $q \in Q$, dass $(q_0, w, \#) \vdash_{\mathcal{K}_\varepsilon}^* (q, \varepsilon, \varepsilon)$. Dann ist $(q_0', w, \#) \vdash_{\mathcal{K}_F} (q_0, w, \#\$) \vdash_{\mathcal{K}_F}^* (q, \varepsilon, \$) \vdash_{\mathcal{K}_F} (q_F, \varepsilon, \varepsilon)$ ein akzeptierender Lauf für w in $\mathcal{K}_F$, also $w \in \mathcal{L}(\mathcal{K}_F)$. Sei nun $w \in \mathcal{L}(\mathcal{K}_F)$. Da der Endzustand q_F in $\mathcal{K}_F$ nur beim Übergang $\delta'(q, \varepsilon, \$)$ erreicht werden kann, ist der akzeptierender Lauf für w in $\mathcal{K}_F$ von der Gestalt $(q_0', w, \#) \vdash_{\mathcal{K}_F} (q_0, w, \#\$) \vdash_{\mathcal{K}_F}^* (q, \varepsilon, \$) \vdash_{\mathcal{K}_F} (q_F, \varepsilon, \varepsilon)$. Wegen der Konstruktion des Automaten mit Akzeptanz bei leerem Keller ist der induzierte Lauf in $\mathcal{K}_\varepsilon$: $(q_0, w, \#) \vdash_{\mathcal{K}_\varepsilon}^* (q, \varepsilon, \varepsilon)$. Also ist $w \in \mathcal{L}_\varepsilon(\mathcal{K}_\varepsilon)$. □

Satz 3.23 (Von $\mathcal{K}_F$ zu $\mathcal{K}_\varepsilon$). *Zu jedem NKA $\mathcal{K}_F$ mit Akzeptanz über Endzustände gibt es einen NKA $\mathcal{K}_\varepsilon$ mit Akzeptanz bei leerem Keller, so dass $\mathcal{L}(\mathcal{K}_F) = \mathcal{L}(\mathcal{K}_\varepsilon)$.*

Beweis. Sei $\mathcal{K}_F = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ ein NKA mit Akzeptanz über Endzustände. Wir verwenden eine ähnliche Simulationsstrategie wie im Beweis von Satz 3.22 und konzipieren einen NKA $\mathcal{K}_\varepsilon$ mit Akzeptanz bei leerem Keller, so dass $\mathcal{K}_\varepsilon$ eine schrittweise Simulation von $\mathcal{K}_F$ durchführt. Sobald der simulierte NKA $\mathcal{K}_F$ eine akzeptierende Konfiguration (p, ε, x), $p \in F$, erreicht, entleert $\mathcal{K}_\varepsilon$ seinen Keller und akzeptiert ebenfalls. Um zu verhindern, dass nicht-akzeptierende Berechnungen

$$(q_0, w, \#) \vdash_{\mathcal{K}_F}^* (q, \varepsilon, \varepsilon), \quad \text{wobei } q \notin F,$$

zur Akzeptanz von $\mathcal{K}_\varepsilon$ führen, verwenden wir ein zusätzliches Kellersymbol \$, das stets ganz unten im Keller von $\mathcal{K}_\varepsilon$ liegt und das nur dann entfernt wird, wenn $\mathcal{K}_F$ eine akzeptierende Konfiguration (p, ε, x), $p \in F$, erreicht. Wir definieren die Komponenten von $\mathcal{K}_\varepsilon$ wie folgt. Der Zustandsraum von $\mathcal{K}_\varepsilon$ besteht aus den Zuständen von $\mathcal{K}_F$ und zwei weiteren Zuständen q_0' und q_F. Der Zustand q_0' ist der Anfangszustand und dient lediglich dazu, $\mathcal{K}_\varepsilon$ in eine Konfiguration zu bringen, die \$ als unterstes Element auf den Stack schreibt. Der Zustand q_F dient der Entleerung des Kellers sobald $\mathcal{K}_F$ eine akzeptierende Endkonfiguration erreicht hat. Die formale Konstruktion von $\mathcal{K}_\varepsilon$ erfolgt so, dass jeder akzeptierende Lauf

$$(q_0, w, \#) \vdash_{\mathcal{K}_F}^* (p, \varepsilon, A_1 \dots A_n), \quad \text{wobei } p \in F,$$

von $\mathcal{K}_F$ einen akzeptierenden Lauf in $\mathcal{K}_\varepsilon$ folgender Form induziert:

$$\begin{array}{lll} (q_0', w, \#) & \vdash_{\mathcal{K}_\varepsilon} & (q_0, w, \#\$) \\ & \vdash_{\mathcal{K}_\varepsilon}^* & (p, \varepsilon, A_1 A_2 \dots A_n \$) \\ & \vdash_{\mathcal{K}_\varepsilon} & (q_F, \varepsilon, A_2 \dots A_n \$) \\ & \vdash_{\mathcal{K}_\varepsilon}^* & (q_F, \varepsilon, \$) \\ & \vdash_{\mathcal{K}_\varepsilon} & (q_F, \varepsilon, \varepsilon). \end{array}$$

Die formale Definition von $\mathcal{K}_\varepsilon$ ist nun wie folgt:

$$\mathcal{K}_\varepsilon \stackrel{\text{def}}{=} (Q \cup \{q_0', q_F\}, \Sigma, \Gamma \cup \{\$\}, \delta', q_0', \#, \varnothing)$$

wobei $\$ \notin \Gamma$, $q_0', q_F \notin Q$, $q_0' \neq q_F$. Die Übergangsfunktion δ' ist wie folgt definiert. Seien $a \in \Sigma$, $A \in \Gamma$ und q, p Zustände in $\mathcal{K}$, so dass $q \notin F$ und $p \in F$.

$$\begin{aligned}
\delta'(q_0', \varepsilon, \#) &= \{(q_0, \#\$)\} \\
\\
\delta'(q_F, \varepsilon, A) &= \{(q_F, \varepsilon)\} \\
\delta'(q_F, \varepsilon, \$) &= \{(q_F, \varepsilon)\}
\end{aligned}
\qquad
\begin{aligned}
\delta'(q, a, A) &= \delta(q, a, A) \\
\delta'(q, \varepsilon, A) &= \delta(q, \varepsilon, A) \\
\\
\delta'(p, a, A) &= \delta(p, a, A) \\
\delta'(p, \varepsilon, A) &= \{(q_F, \varepsilon)\} \cup \delta(p, \varepsilon, A) \\
\delta'(p, \varepsilon, \$) &= \{(q_F, \varepsilon)\}
\end{aligned}$$

In allen verbleibenden Fällen ist $\delta'(\cdot) = \varnothing$. Sobald sich $\mathcal{K}_F$ in einem Endzustand $p \in F$ befindet und der Keller von $\mathcal{K}_F$ nicht leer ist, dann entscheidet $\mathcal{K}_\varepsilon$ *nichtdeterministisch*, ob die Simulation fortgeführt wird oder ob $\mathcal{K}_\varepsilon$ in den Zustand q_F wechselt und dort (nach Entleerung des Kellers) anhält. Findet dieser ε-Übergang zu Zustand q_F zu früh (noch bevor die Eingabe vollständig gelesen wurde) statt, dann liegt eine verwerfende Berechnung von $\mathcal{K}_\varepsilon$ vor. Ist die Eingabe bereits zu Ende gelesen, dann akzeptieren $\mathcal{K}_F$ und $\mathcal{K}_\varepsilon$. Dies liegt daran, dass in Zustand q_F ausschließlich ε-Transitionen möglich sind. Tatsächlich gilt nun $\mathcal{L}(\mathcal{K}_F) = \mathcal{L}_\varepsilon(\mathcal{K}_\varepsilon)$, da es zu jedem akzeptierenden Lauf

$$(q_0, w, \#) \vdash_{\mathcal{K}_F}^* (p, \varepsilon, x),\ p \in F$$

von $\mathcal{K}_F$ einen "entsprechenden" akzeptierenden Lauf $(q_0', w, \#) \vdash_{\mathcal{K}_\varepsilon}^* (q', \varepsilon, \varepsilon)$ von $\mathcal{K}_\varepsilon$ gibt, wobei $q' \in Q \cup \{q_0', q_F\}$. Umgekehrt gibt es zu jedem akzeptierenden Lauf in $\mathcal{K}_\varepsilon$ einen "entsprechenden" akzeptierenden Lauf in $\mathcal{K}_F$. □

Äquivalenz von NKA und kontextfreien Grammatiken

Die Formalismen der CFG und des Automatenmodells NKA sind äquivalent. Für den Nachweis wird gezeigt, dass für jeden NKA mit Akzeptanz bei leerem Keller eine CFG konstruiert wird, so dass die erzeugte Sprache der CFG und die akzeptierte Sprache des NKA gleich sind. Aus einer gegebenen CFG wird ein NKA konstruiert, indem der Herleitungsprozess, der durch die Linksableitungen einer CFG in Greibach-Normalform gegeben ist, in naheliegender Weise durch einen NKA vorgenommen wird (vergleiche Kapitel 3.5.1).

Satz 3.24 (Von Greibach NF zu NKA). *Zu jeder CFG G in Greibach Normalform gibt es einen NKA $\mathcal{K}$ mit $\mathcal{L}(G) = \mathcal{L}_\varepsilon(\mathcal{K})$.*

Beweis. Sei $G = (V, \Sigma, \mathcal{P}, S)$ eine CFG in Greibach Normalform. Wir definieren einen NKA $\mathcal{K}$ mit nur einem Zustand q_0 und ohne ε-Transitionen. Die Idee dabei ist, dass jede Regel $A \to aB_1 \ldots B_k$ von G einem Übergang von $\mathcal{K}$ entspricht.

$$\mathcal{K} \stackrel{\text{def}}{=} (\{q_0\}, \Sigma, V, \delta, q_0, S),$$

wobei

$$\delta(q_0, a, A) = \left\{ (q_0, B_1 \ldots B_k) : A \to aB_1 \ldots B_k \right\}$$

für alle $a \in \Sigma$ und $A \in V$. In allen verbleibenden Fällen ist $\delta(\cdot) = \varnothing$. Die Behandlung einer Terminalregel $A \to a$ ergibt sich aus k=0. Sie induziert also das Paar $(q_0, \varepsilon) \in \delta(q_0, a, A)$.

Der Nachweis für $\mathcal{L}_\varepsilon(\mathcal{K}) = \mathcal{L}(G)$ kann erbracht werden, indem man zunächst durch Induktion nach n zeigt, dass für alle $a_1, \ldots, a_n \in \Sigma$, $x \in \Sigma^*$, $A_1, \ldots, A_\ell \in V$ gilt:

$$(q_0, a_1 \ldots a_n x, S) \vdash^n (q_0, x, A_1 \ldots A_\ell) \quad \text{gdw} \quad S \Rightarrow_L^n a_1 \ldots a_n A_1 \ldots A_\ell$$

Dabei steht $\Rightarrow_L^n$ für die Linksherleitbarkeit mit n Herleitungsschritten. Entsprechend steht $\vdash^n$ für die "n-Schritt-Konfigurationsrelation". Da $\mathcal{K}$ keine ε-Transitionen hat, gilt für alle Wörter $w \in \Sigma^*$ der Länge n:

$$(q_0, w, S) \vdash^* (q_0, \varepsilon, y) \quad \text{gdw} \quad (q_0, w, S) \vdash^n (q_0, \varepsilon, y).$$

Da G in Greibach Normalform ist, bestehen alle Herleitungen eines Worts $w \in \mathcal{L}(G)$ mit $|w| = n$ aus genau n Regelanwendungen. Daher gilt $S \Rightarrow_L^n w$ genau dann, wenn $S \Rightarrow^* w$ für alle Wörter $w \in \Sigma^*$ der Länge n. Mit $x = \varepsilon$, $w = a_1 \ldots a_n$, $\ell = 0$ folgt daher aus dem oben angegebenen Zusammenhang zwischen $\Rightarrow_L^n$ und $\vdash^n$:

$$\begin{array}{lll} w \in \mathcal{L}_\varepsilon(\mathcal{K}) & \text{gdw} & (q_0, w, S) \vdash^* (q_0, \varepsilon, \varepsilon) \\ & \text{gdw} & (q_0, w, S) \vdash^n (q_0, \varepsilon, \varepsilon) \\ & \text{gdw} & S \Rightarrow_L^n w \\ & \text{gdw} & S \Rightarrow^* w \\ & \text{gdw} & w \in \mathcal{L}(G) \end{array}$$

Also stimmen $\mathcal{L}_\varepsilon(\mathcal{K})$ und $\mathcal{L}(G)$ überein. □

Der im Beweis von Satz 3.24 angegebene NKA hat nur einen Zustand und keine ε-Transitionen. Der nichtdeterministische Ableitungsprozess jeder kontextfreien Grammatik lässt sich also mit einem NKA mit nur *einem* Zustand simulieren, der in jedem Schritt ein Zeichen des Eingabeworts "konsumiert".

Beispiel 3.25 (Von Greibach NF zu NKA). Die CFG mit den Regeln

$$S \to aSA \mid bSB \mid aB \mid bAAA, \quad A \to a, \quad B \to b$$

ist in Greibach Normalform. Der gemäß Satz 3.24 konstruierte NKA hat folgende Gestalt: $\mathcal{K} = (\{q_0\}, \{a, b\}, \{A, S, B\}, \delta, q_0, S)$, wobei

$$\begin{array}{lll} \delta(q_0, a, S) = \{(q_0, SA), (q_0, B)\} & & \delta(q_0, b, S) = \{(q_0, SB), (q_0, AAA)\} \\ \delta(q_0, a, A) = \{(q_0, \varepsilon)\} & & \delta(q_0, b, B) = \{(q_0, \varepsilon)\} \end{array}$$

Z.B. entspricht die Linksableitung $S \Rightarrow_L aSA \Rightarrow_L aaBA \Rightarrow_L aabA \Rightarrow_L aaba$ der akzeptierenden Berechnung

$$(q_0, aaba, S) \vdash (q_0, aba, SA) \vdash (q_0, ba, BA) \vdash (q_0, a, A) \vdash (q_0, \varepsilon, \varepsilon)$$ ■

Wir zeigen nun, dass auch umgekehrt zu jedem Kellerautomaten $\mathcal{K}$ mit Akzeptanz bei leerem Keller eine kontextfreie Grammatik konstruiert werden kann, die die Sprache $\mathcal{L}_\varepsilon(\mathcal{K})$ erzeugt. Aufgrund der Äquivalenz der Akzeptanzbedingungen "über Endzustände" und "bei leerem Keller" gilt dieselbe Aussage auch für die NKA-Sprachen $\mathcal{L}(\mathcal{K})$.

Satz 3.26 (Von NKA zu CFG). *Zu jedem NKA $\mathcal{K}$ gibt es eine CFG G mit $\mathcal{L}_\varepsilon(\mathcal{K}) = \mathcal{L}(G)$.*

Beweis. Sei $\mathcal{K} = (Q, \Sigma, \Gamma, \delta, q_0, \#)$ ein NKA mit Akzeptanz bei leerem Keller. Die Idee für die Konstruktion einer äquivalenten kontextfreien Grammatik G besteht darin, alle Tripel $\langle q, A, p\rangle \in Q \times \Gamma \times Q$ als Nichtterminale zu verwenden und das Produktionssystem so zu definieren, dass für alle Wörter $w \in \Sigma^*$, Kellersymbole $A \in \Gamma$ und Zustände $q, p \in Q$ gilt:

$$\underset{\substack{\uparrow\\ \text{Herleitung}\\ \text{in G}}}{\langle q, A, p\rangle \Rightarrow^* w} \quad \text{gdw} \quad \underset{\substack{\uparrow\\ \text{Konfigurations-}\\ \text{übergänge in } \mathcal{K}}}{(q, w, A) \vdash^* (p, \varepsilon, \varepsilon)} \qquad (*)$$

Die Bedingung rechts in Aussage (*), also $(q, w, A) \vdash^* (p, \varepsilon, \varepsilon)$, ist äquivalent zu der Aussage, dass der NKA $\mathcal{K}$ die Konfiguration (q, wu, Az) in (p, u, z) überführen kann, wobei mit Ausnahme des letzten Konfigurationswechsels kein Zugriff auf die Kellerzellen, in denen z ablegt ist, erfolgt. Dabei sind $u \in \Sigma^*$ und $z \in \Gamma^*$ beliebige Wörter über dem Eingabe- bzw. Kelleralphabet. Die Definition von G unterliegt nun der Idee, akzeptierende Läufe von $\mathcal{K}$ in Lauffragmente $(p_{i-1}, u_{i-1}, z_{i-1}) \vdash^* (p_i, u_i, z_i)$ zu zerlegen, wobei $z_{i-1} = B_i z_i$ mit $B_i \in \Gamma$ und u_i ein Suffix von u_{i-1} ist, in denen – abgesehen vom letzten Konfigurationswechsel – die Kellerzellen, in denen die Symbole von z_i liegen, unberührt bleiben. Die Grammatik G kann solche Lauffragmente gemäß (*) durch Ableitungen der Form $\langle p_{i-1}, B_i, p_i\rangle \Longrightarrow^* w_i$ simulieren, wobei w_i dasjenige Präfix von u_{i-1} ist, für welches $u_{i-1} = w_i u_i$ gilt.

Die formale Definition der Komponenten von G ist wie folgt. Die Variablenmenge ist $V = \{S\} \cup Q \times \Gamma \times Q$, wobei S als Startsymbol fungiert. Das Terminalalphabet von G ist das Eingabealphabet Σ von $\mathcal{K}$. Das Produktionssystem von G besteht aus folgenden Regeln:

- Startregeln: $S \to \langle q_0, \#, p\rangle$ für jeden Zustand $p \in Q$
- Regeln zur Simulation der Transitionen von $\mathcal{K}$:
  1. Lesen eines Eingabezeichens mit Push-Operationen:
     für jeden Übergang $(r, B_1 B_2 \ldots B_n) \in \delta(q, a, A)$, wobei $n \geqslant 1$, $B_1, \ldots, B_n \in \Gamma$ und $a \in \Sigma$, enthält G die Regeln
     $$\langle q, A, p\rangle \to a \langle r, B_1, p_1\rangle \langle p_1, B_2, p_2\rangle \ldots \langle p_{n-1}, B_n, p\rangle,$$
     wobei $p, p_1, \ldots, p_{n-1} \in Q$.
  2. Lesen eines Eingabezeichens ohne Push-Operationen:
     für jeden Übergang $(p, \varepsilon) \in \delta(q, a, A)$ und $a \in \Sigma$, enthält G die Regel $\langle q, A, p\rangle \to a$.
  3. ε-Transition mit Push-Operationen:
     für alle $(r, B_1 B_2 \ldots B_n) \in \delta(q, \varepsilon, A)$, wobei $n \geqslant 1$, $B_1, \ldots, B_n \in \Gamma$ und $a \in \Sigma$, enthält G die Regeln
     $$\langle q, A, p\rangle \to \langle r, B_1, p_1\rangle \langle p_1, B_2, p_2\rangle \ldots \langle p_{n-1}, B_n, p\rangle,$$
     wobei $p, p_1, \ldots, p_{n-1} \in Q$.

4. ε-Transition ohne Push-Operationen:
   für alle $(p, \varepsilon) \in \delta(q, \varepsilon, A)$ und $a \in \Sigma$, enthält G die ε-Regel $\langle q, A, p\rangle \to \varepsilon$.

In 1., 2., 3. und 4. sind $q, r \in Q$ beliebige Zustände und $A \in \Gamma$ ein beliebiges Kellersymbol. Die Korrektheit dieser Grammatik kann wie folgt nachgewiesen werden. Aussage (*) kann durch Induktion über die Länge einer Herleitung in G (Implikation "$\Longrightarrow$") bzw. Länge eines akzeptierenden Laufs in $\mathcal{K}$ (Implikation "$\Longleftarrow$") nachgewiesen werden. Die weitere Argumentation für den Nachweis, dass $\mathcal{L}(G) = \mathcal{L}_\varepsilon(\mathcal{K})$, ist dann wie folgt.

Wir beginnen mit der Inklusion "$\subseteq$". Wir nehmen zunächst an, dass $w \in \mathcal{L}(G)$. Dann gilt $S \Rightarrow^* w$. Also muss es eine Startregel $S \to \langle q_0, \#, p\rangle$ geben, so dass $\langle q_0, \#, p\rangle \Rightarrow^* w$. Wegen Aussage (*) gilt dann $(q_0, w, \#) \vdash^* (p, \varepsilon, \varepsilon)$ und somit $w \in \mathcal{L}_\varepsilon(\mathcal{K})$.

Die Inklusion "$\supseteq$" kann mit analogen Argumenten nachgewiesen werden. Ist $w \in \mathcal{L}_\varepsilon(\mathcal{K})$, so gibt es einen Zustand $p \in Q$ mit $(q_0, w, \#) \vdash^* (p, \varepsilon, \varepsilon)$. Wegen Aussage (*) gilt dann $\langle q_0, \#, p\rangle \Rightarrow^* w$. Also ist durch $S \Rightarrow \langle q_0, \#, p\rangle \Rightarrow^* w$ eine Herleitung von w in G gegeben.

Nun zum induktiven Nachweis von Aussage (*). Exemplarisch zeigen wir die Implikation "$\Longrightarrow$". Im Induktionsanfang nehmen wir an, dass $\langle q, A, p\rangle \Rightarrow w$ (Ableitung der Länge $k = 1$). Dies ist nur dann möglich, wenn

- entweder $w = a \in \Sigma$ und die Regel $\langle q, A, p\rangle \to a$ zur Simulation des Lesens eines Eingabezeichens ohne Push-Operation eingesetzt wurde
- oder wenn $w = \varepsilon$ und die Regel $\langle q, A, p\rangle \to \varepsilon$ zur Simulation einer ε-Transition ohne Push-Operation angewandt wurde.

Im ersten Fall gilt $(p, \varepsilon) \in \delta(q, a, A)$ und somit $(q, a, A) \vdash (p, \varepsilon, \varepsilon)$. Im zweiten Fall gilt $(p, \varepsilon) \in \delta(q, \varepsilon, A)$ und $(q, \varepsilon, A) \vdash (p, \varepsilon, \varepsilon)$. Im Induktionsschritt nehmen wir an, dass eine Ableitung von w mit $k+1$ Regelanwendungen vorliegt, wobei $k \geqslant 1$. Etwa:

$$\langle q, A, p\rangle \Rightarrow x \Rightarrow^k w, \quad \text{wobei } x \in (V \cup \Sigma)^+ \text{ wenigstens eine Variable enthält}^{11}$$

Wegen $k \geqslant 1$ muss sich der erste Herleitungsschritt auf die Anwendung einer Regel zur Simulation eines Konfigurationswechsels von $\mathcal{K}$ mit Push-Operationen beziehen. Etwa die Regel

$$\langle q, A, p\rangle \to \underbrace{a \langle r, B_1, p_1\rangle \langle p_1, B_2, p_2\rangle \dots \langle p_{n-1}, B_n, p\rangle}_{=x}$$

Dann hat w die Form $a w_1 w_2 \dots w_n$, wobei $\langle p_{i-1}, B_i, p_i\rangle \Rightarrow^{k_i} w_i$ für $1 \leqslant i \leqslant n$. und $k = k_1 + \dots + k_n$. Dabei ist $r = p_0$ und $p_n = p$. (Die Existenz einer solchen Zerlegung von w folgt z.B. durch die Betrachtung eines Ableitungsbaums. Die Wörter w_i sind durch die Blätter im Teilbaum von B_i gegeben.) Wegen $k_i \leqslant k$ können wir nun die Induktionsvoraussetzung anwenden und erhalten

$$(p_{i-1}, w_i, B_i) \vdash^* (p_i, \varepsilon, \varepsilon) \quad \text{für } 1 \leqslant i \leqslant n.$$

[11] Das Symbol $\Rightarrow^k$ steht für die Ableitbarkeit mit genau k Ableitungsschritten.

Diese Konfigurationswechsel in $\mathcal{K}$ können nun kombiniert werden:

$$\begin{aligned}(q, w, A) = (q, a w_1 w_2 \dots w_n, A) &\vdash (r, w_1 w_2 \dots w_n, B_1 B_2 \dots B_n)\\ &\vdash^* (p_1, w_2 \dots w_n, B_2 \dots B_n)\\ &\vdots\\ &\vdash^* (p_{n-1}, w_n, B_n)\\ &\vdash^* (p, \varepsilon, \varepsilon)\end{aligned}$$

Die Beweisrichtung "$\Longleftarrow$" folgt analogen Argumenten. □

Beispiel 3.27 (Von NKA zu CFG). Die kontextfreie Sprache $L = \{a^n b^n : n \geqslant 1\}$ ist die vom Kellerautomaten $\mathcal{K} = (\{q_a, q_b\}, \{a, b\}, \{B, \#\}, \delta, q_a, \#)$ akzeptierte Sprache, wobei

$$\begin{aligned}\delta(q_a, a, \#) &= \{(q_a, B)\} & \qquad \delta(q_a, b, B) &= \{(q_b, \varepsilon)\}\\ \delta(q_a, a, a) &= \{(q_a, BB)\} & \qquad \delta(q_b, b, B) &= \{(q_b, \varepsilon)\},\end{aligned}$$

d. h. , $L = \mathcal{L}(\mathcal{K})$ (vergleiche 3.20 auf Seite 93). Aus der Konstruktion im Satz 3.26 erhalten wir die Grammatik $G = (V, \{a, b\}, \mathcal{P}, S)$ mit $V = \{S\} \cup \{q_a, q_b\} \times \{B, \#\} \times \{q_a, q_b\}$ (damit gibt es 9 Variablen in der Grammatik G) und den Regeln

$$\begin{aligned}S &\to \langle q_a, \#, q_a\rangle \mid \langle q_a, \#, q_b\rangle\\ \langle q_a, \#, q_a\rangle &\to a\langle q_a, B, q_a\rangle & \qquad \langle q_a, \#, q_b\rangle &\to a\langle q_a, B, q_b\rangle\\ \langle q_a, B, p\rangle &\to a\langle q_a, B, p_1\rangle\langle p_1, B, p\rangle, \text{ wobei } p, p_1 \in \{q_a, q_b\}\\ \langle q_a, B, q_b\rangle &\to b & \qquad \langle q_b, B, q_b\rangle &\to b\end{aligned}$$

Beispielsweise entspricht der akzeptierende Lauf

$$(q_a, ab, \#) \vdash (q_a, b, B) \vdash (q_b, \varepsilon, \varepsilon)$$

der simulierten Linksableitung in G:

$$S \Rightarrow \langle q_a, \#, q_b\rangle \Rightarrow a\langle q_a, B, q_b\rangle \Rightarrow ab.$$

■

Das folgende Corollar beruht auf einer Kombination der vorangegangenen Ergebnisse:

Corollar 3.28 (Charakterisierung kontextfreier Sprachen). *Sei* L *eine Sprache. Dann sind folgende Aussagen äquivalent:*

(a) L *ist kontextfrei, d.h.* $L = \mathcal{L}(G)$ *für eine CFG* G.

(b) $L = \mathcal{L}(\mathcal{K})$ *für einen NKA* $\mathcal{K}$.

(c) $L = \mathcal{L}_\varepsilon(\mathcal{K}_\varepsilon)$ *für einen NKA* $\mathcal{K}_\varepsilon$.

Da für die Konstruktion eines NKA für gegebene kontextfreie Grammatik in Greibach Normalform ein NKA mit nur einem Zustand und ohne ε-Transitionen ausreichend ist, ist jeder Kellerautomat $\mathcal{K}$ äquivalent zu einem Kellerautomaten mit Akzeptanz bei leerem Keller, der nur einen Zustand hat. Auch auf ε-Transitionen kann verzichtet werden, sofern das leere Wort nicht in der akzeptierten Sprache von $\mathcal{K}$ liegt.

In Abschnitt 3.4.1 (Seite 85) haben wir mit Hilfe des Pumping Lemmas nachgewiesen, dass die Sprache $L = \{a^n b^n c^n : n \geqslant 0\}$ nicht kontextfrei ist. Mit Corollar 3.28 können wir hierfür eine intuitive Erklärung geben. Liegt einem NKA ein Eingabewort der Form $a^n b^n c^n$ vor, dann kann er die führenden a's in seinem Keller ablegen und diese beim Lesen der b's entfernen. Nach dem Lesen der b's ist der Keller jedoch leer. Die Information über die Anzahl n der gelesenen a's bzw. b's steht bei der Bearbeitung der c's nicht mehr zur Verfügung. Derartig intuitive Argumentationen helfen oftmals einzuschätzen, ob eine Sprache kontextfrei ist oder nicht. Selbstverständlich ersetzen sie keinen formalen Beweis.

Durchschnitt zwischen kontextfreien und regulären Sprachen. In Abschnitt 3.4.2 (Seite 86 ff) haben wir gesehen, dass die Klasse der kontextfreien Sprachen *nicht* unter der Durchschnittsbildung abgeschlossen ist. Dennoch ist der Durchschnitt einer kontextfreien Sprache mit einer regulären Sprache stets kontextfrei. Wir können hierfür ähnlich wie für reguläre Sprachen verfahren und das *Produkt* eines NKA mit einem NFA bilden; dieses ist ein NKA für die Durchschnittssprache.

Lemma 3.29. *Ist* L_1 *eine kontextfreie und* L_2 *eine reguläre Sprache, so ist* $L_1 \cap L_2$ *kontextfrei.*

Beweis. Sei $\mathcal{K} = (Q_1, \Sigma, \Gamma, \delta_1, q_{0,1}, \#, F_1)$ ein NKA mit $\mathcal{L}(\mathcal{K}) = L_1$ und $\mathcal{M} = (Q_2, \Sigma, \delta_2, \{q_{0,2}\}, F_2)$ ein NFA mit $\mathcal{L}(\mathcal{M}) = L_2$, der genau einen Anfangszustand hat. Wir definieren das Produkt aus $\mathcal{K}$ und $\mathcal{M}$ als NKA:

$$\mathcal{K} \otimes \mathcal{M} \stackrel{\text{def}}{=} (Q_1 \times Q_2, \Sigma, \Gamma, \delta, \langle q_{0,1}, q_{0,2}\rangle, \#, F_1 \times F_2),$$

wobei die Übergangsfunktion δ des Produkt-NKA wie folgt definiert ist:

$$\begin{aligned}
\delta(\langle q_1, q_2\rangle, a, A) &= \left\{(\langle q_1', q_2'\rangle, x) : (q_1', x) \in \delta_1(q_1, a, A) \text{ und } q_2' \in \delta_2(q_2, a)\right\} \\
\delta(\langle q_1, q_2\rangle, \varepsilon, A) &= \left\{(\langle q_1', q_2\rangle, x) : (q_1', x) \in \delta_1(q_1, \varepsilon, A)\right\}
\end{aligned}$$

Ähnlich wie für die Korrektheit der Produktkonstruktion für endliche Automaten kann nun gezeigt werden, dass tatsächlich $\mathcal{L}(\mathcal{K} \otimes \mathcal{M}) = \mathcal{L}(\mathcal{K}) \cap \mathcal{L}(\mathcal{M})$ gilt. □

3.5.3 Deterministische Kellerautomaten

Deterministische Kellerautomaten (DKA) sind NKA, in denen der jeweils nächste Schritt eindeutig festgelegt ist. Da wir die ε-Transitionen beibehalten möchten, müssen wir fordern, dass für jeden Zustand q, jedes Zeichen $a \in \Sigma$ und jedes Bandsymbol A die Menge $\delta(q, a, A) \cup \delta(q, \varepsilon, A)$ höchstens einelementig ist. Dies stellt sicher, dass jede Konfiguration höchstens eine Folgekonfiguration hat.

Definition 3.30 (Deterministischer Kellerautomat (DKA)). Ein DKA ist ein NKA $\mathcal{K} = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$, so dass für alle $q \in Q$, $a \in \Sigma$ und $A \in \Gamma$ gilt:

(1) $|\delta(q, a, A)| \leqslant 1$

(2) $|\delta(q, \varepsilon, A)| \leqslant 1$

(3) Aus $\delta(q, \varepsilon, A) \neq \varnothing$ folgt $\delta(q, a, A) = \varnothing$.

Eine Sprache L heißt *deterministisch kontextfrei*, falls es einen DKA $\mathcal{K}$ mit $L = \mathcal{L}(\mathcal{K})$ gibt. Wir verwenden die englische Abkürzung DCFL für deterministisch kontextfreie Sprachen. ■

Die Übergangsfunktion δ eines DKA kann als partielle Funktion $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma^*$ angesehen werden. Es ist daher üblich, auf die Mengenschreibweise zu verzichten und $\delta(q, a, A) = (p, z)$ statt $\delta(q, a, A) = \{(p, z)\}$ zu schreiben. Entsprechend ist die Schreibweise $\delta(q, a, A) = \bot$ anstelle von $\delta(q, a, A) = \varnothing$ gebräuchlich.

Beispiel 3.31 (Deterministisch kontextfreie Sprachen). Die Sprache $L = \{a^n b^n : n \geqslant 1\}$ ist deterministisch kontextfrei. Intuitiv ist klar, dass wir einen DKA so entwerfen können, der führende a's des Eingabeworts in seinem Keller ablegt. Sobald das erste b gelesen wird, versucht der DKA seinen Keller zu entleeren, indem für jedes gelesene b ein a aus dem Keller genommen wird. Tatsächlich arbeitet der in Beispiel 3.20 auf Seite 93 angegebene NKA $\mathcal{K}$ für die Sprache L nach diesem Prinzip. Durch Inspektion der Übergangsfunktion überzeugt man sich davon, dass $\mathcal{K}$ tatsächlich deterministisch ist. Die Sprachen

$$L_1 = \left\{ww^R : w \in \{0,1\}^*\right\} \quad \text{und} \quad L_2 = \left\{w\$w^R : w \in \{0,1\}^*\right\}$$

sind kontextfrei. (Wie zuvor steht w^R für das gespiegelte Wort von w). Jedoch ist nur L_2 deterministisch kontextfrei. Wir begnügen uns mit informellen Argumenten. Für die Sprache L_2 kann man einen DKA entwerfen, der zunächst sämtliche gelesenen Zeichen in seinem Keller ablegt. Sobald er das Symbol \$ liest, weiß der DKA, dass das Eingabewort von der Form $w\$y$ mit $w \in \{0,1\}^*$ und $y \in \{0,1,\$\}^*$ ist. Zu prüfen ist nun, ob der Kellerinhalt mit dem noch zu lesenden Teilwort y übereinstimmt. Hierzu ist offenbar kein Nichtdeterminismus notwendig. Für die Sprache L_1 dagegen kann man zwar einen ähnlich arbeitenden NKA entwerfen (siehe Beispiel 3.21, Seite 94), jedoch muss dieser NKA *nichtdeterministisch* entscheiden, wann er beginnt, seinen Keller zu entleeren. ■

Offenbar ist die Klasse der deterministisch kontextfreien Sprachen eine Teilklasse der Sprachen vom Typ 2. Andererseits umfasst die Klasse der deterministisch kontextfreien Sprachen die regulären Sprachen. Dies folgt aus der Beobachtung, dass man jeden DFA als DKA auffassen kann, der niemals das Kellerstartsymbol # durch andere Symbole ersetzt.

Satz 3.32 (Abschlusseigenschaften von DCFL). *Die Klasse der deterministisch kontextfreien Sprachen ist unter der Komplementbildung abgeschlossen; nicht aber unter Vereinigung, Konkatenation, Kleeneabschluss und Durchschnittsbildung.* (teilweise ohne Beweis)

Wir erläutern nur, wie man für den Durchschnitt und die Vereinigung argumentieren kann, wenn man das Abschlussresultat für den Komplementoperator als bekannt voraussetzt. Die Sprachen

$$L_1 = \{a^n b^m c^m : n, m \in \mathbb{N}\}, \quad L_2 = \{a^n b^n c^m : n, m \in \mathbb{N}\}$$

sind deterministisch kontextfrei. (Für L_1 und L_2 kann man einen ähnlichen DKA wie für die Sprache $\{a^n b^n : n \geqslant 1\}$ entwerfen. Siehe Beispiel 3.20 auf Seite 93.) Andererseits ist die Durchschnittssprache

$$L_1 \cap L_2 = \{a^n b^n c^n : n \in \mathbb{N}\}$$

nicht (deterministisch) kontextfrei (siehe Seite 85). Damit ist die Klasse der deterministisch kontextfreien Sprachen nicht unter der Durchschnittsbildung abgeschlossen. Das oben zitierte

Ergebnis, dass mit L auch $\overline{L}$ deterministisch kontextfrei ist, zusammen mit der de Morganschen Regel

$$L_1 \cap L_2 \;=\; \overline{\overline{L_1} \cup \overline{L_2}},$$

ergibt, dass die Klasse der deterministischen kontextfreien Sprachen bzgl. der Vereinigung *nicht* abgeschlossen sein kann.

In Lemma 3.29 (Seite 103) haben wir gesehen, dass das Produkt $\mathcal{K} \otimes \mathcal{M}$ eines NKA $\mathcal{K}$ und NFA $\mathcal{M}$ genau die Durchschnittsprache akzeptiert, wenn die Akzeptanz durch Endzustände zugrundegelegt wird. Durch Inspektion der Übergangsfunktion ergibt sich, dass $\mathcal{K} \otimes \mathcal{M}$ ein DKA ist, falls sowohl $\mathcal{K}$ als auch $\mathcal{M}$ deterministisch sind. Somit ist $L_1 \cap L_2$ deterministisch kontextfrei, wenn L_1 deterministisch kontextfrei und L_2 regulär ist. Diese Überlegungen liefern folgendes Lemma.

Lemma 3.33 (Schnitt von DCFL und regulärer Sprache). *Ist L_1 deterministisch kontextfrei und L_2 regulär, so ist $L_1 \cap L_2$ deterministisch kontextfrei.*

In der Definition deterministisch kontextfreier Sprachen (Definition 3.30, Seite 103) haben wir die Akzeptanz durch Endzustände gefordert. Im Gegensatz zu nichtdeterministischen Kellerautomaten sind die beiden Akzeptanzvarianten "Akzeptanz durch Endzustände" und "Akzeptanz bei leerem Keller" für deterministische Kellerautomaten *nicht* gleichwertig. Die Akzeptanz bei leerem Keller ist für DKA schwächer als die Akzeptanz durch Endzustände. Dies liegt vorallem daran, dass ein Kellerautomat stets dann anhält, wenn sein Keller leer ist. Für die Akzeptanz ist es jedoch entscheidend, dass die Eingabe komplett gelesen wurde. Liegt ein DKA $\mathcal{K}$ und ein Eingabewort $w = xy$, wobei $y \neq \varepsilon$, vor, so dass $\mathcal{K}$ für das Präfix x bei leerem Keller anhält, dann führt die Berechnung von $\mathcal{K}$ für das Eingabewort w zu einer Konfiguration, in der der Keller leer ist, aber das Teilwort y noch nicht gelesen ist. $\mathcal{K}$ verwirft also das Wort w. Wir präzisieren diese Beobachtungen.

Definition 3.34 (Präfixeigenschaft). Sei $L \subseteq \Sigma^*$. L hat die *Präfixeigenschaft*, wenn für alle Wörter $w \in L$ gilt: Ist x ein echtes Präfix von w, so ist $x \notin L$. ■

Die Sprache $\{a^n b^n : n \geqslant 1\}$ hat die Präfixeigenschaft, da für jedes Wort $w = a^n b^n$ alle echten Präfixe die Form $a^n b^i$ oder a^i mit $n > i \geqslant 1$ haben. Die reguläre Sprache $L = \mathcal{L}(a^* b^*)$ hat die Präfixeigenschaft *nicht*. Z.B. ist $w = aaab \in L$ und $x = aa \in L$.

Wir zeigen nun, dass DKA mit der Akzeptanz bei leerem Keller genau diejenigen deterministisch kontextfreien Sprachen erkennen können, welche die Präfixeigenschaft haben.

Lemma 3.35 (DKA mit Akzeptanz bei leerem Keller). *Sei $\mathcal{K}$ ein DKA.*

(a) $\mathcal{L}_\varepsilon(\mathcal{K})$ ist deterministisch kontextfrei, d.h. $L = \mathcal{L}(\mathcal{K}')$ für einen DKA $\mathcal{K}'$.

(b) $\mathcal{L}_\varepsilon(\mathcal{K})$ hat die Präfixeigenschaft.

Beweis. ad (a). Der im Beweis von Satz 3.22 (Seite 96) konstruierte NKA $\mathcal{K}_F$ mit Akzeptanz durch Endzustände ist deterministisch, falls $\mathcal{K} = \mathcal{K}_\varepsilon$ deterministisch ist.

ad (b). Sei $\mathcal{K}_\varepsilon = (Q, \Sigma, \Gamma, \delta, q_0, \#)$ und $w \in \mathcal{L}_\varepsilon(\mathcal{K})$, $w = xy$, $y \neq \varepsilon$. Angenommen $x \in \mathcal{L}_\varepsilon(\mathcal{K})$. Dann gilt

$$(q_0, x, \#) \;\vdash^*\; (q, \varepsilon, \varepsilon)$$

für einen Zustand $q \in Q$. Da $\mathcal{K}$ deterministisch ist, stimmen die ersten Schritte der Berechnung von $\mathcal{K}$ für w genau mit der Berechnung von $\mathcal{K}$ für x überein. Daher gilt:

$$(q_0, w, \#) = (q_0, xy, \#) \vdash^* (q, y, \varepsilon).$$

Wegen $y \neq \varepsilon$ liegt ein verwerfender Lauf für w vor. Widerspruch. □

Die Sprache $L = \mathcal{L}(0^*1^*)$ ist regulär und somit deterministisch kontextfrei. Es gibt jedoch *keinen* DKA $\mathcal{K}$ mit $L = \mathcal{L}_\varepsilon(\mathcal{K})$, da L die Präfixeigenschaft nicht hat.

Im Beweis von Satz 3.23 (Seite 97) haben wir zu gegebenem NKA $\mathcal{K} = \mathcal{K}_F$ mit Akzeptanz durch Endzustände einen äquivalenten NKA $\mathcal{K}_\varepsilon$ mit Akzeptanz bei leerem Keller konstruiert. Der konstruierte NKA $\mathcal{K}_\varepsilon$ ist jedoch auch dann kein DKA, wenn $\mathcal{K}_F$ deterministisch ist. Dies untermauert das Resultat, dass für DKA die Akzeptanz bei leerem Keller schwächer als die Akzeptanz durch Endzustände ist. Hat jedoch die durch $\mathcal{K}_F$ akzeptierte Sprache $\mathcal{L}(\mathcal{K}_F)$ die Präfixeigenschaft, dann können wir die angegebene Definition des Kellerautomaten $\mathcal{K}_\varepsilon$ so modifizieren, dass ein DKA entsteht, für den die bei leerem Keller akzeptierte Sprache mit $\mathcal{L}(\mathcal{K}_F)$ übereinstimmt. Die durchzuführende Veränderung besteht lediglich darin, dass – sobald $\mathcal{K}_F$ einen Endzustand erreicht – der simulierende DKA $\mathcal{K}_\varepsilon$ seinen Keller entleert.

Lemma 3.36. *Sei L eine deterministische kontextfreie Sprache mit der Präfixeigenschaft. Dann gibt es einen DKA $\mathcal{K}_\varepsilon$ mit $L = \mathcal{L}_\varepsilon(\mathcal{K}_\varepsilon)$.*

Beweis. Sei $\mathcal{K}_F = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ ein DKA für L mit Akzeptanz durch Endzustände, also $\mathcal{L}(\mathcal{K}_F) = L$. Da L die Präfixeigenschaft hat, kann man o.E. annehmen, dass $\delta(p, \varepsilon, A) = \delta(p, a, A) = \bot$ für alle Endzustände $p \in F$ und alle Zeichen $a \in \Sigma$, $A \in \Gamma$. Wir wenden nun die im Beweis von Satz 3.23 (Seite 97) angegebene Konstruktion eines äquivalenten Kellerautomaten $\mathcal{K}_\varepsilon$ mit Akzeptanz bei leerem Keller an. Da $\mathcal{K}$ anhält, sobald ein Zustand $p \in F$ betreten wird, ist $\mathcal{K}_\varepsilon$ deterministisch. □

Lemma 3.35 (Seite 105) und Lemma 3.36 (Seite 106) implizieren folgenden Satz.

Satz 3.37 (DKA-Sprachen für Akzeptanz bei leerem Keller). *Sei L eine Sprache. Dann gilt: L ist genau dann deterministisch kontextfrei mit der Präfixeigenschaft, wenn $L = \mathcal{L}_\varepsilon(\mathcal{K}_\varepsilon)$ für einen DKA $\mathcal{K}_\varepsilon$.*

Abschliessend erwähnen wir, dass es einen einfachen Trick gibt, um eine gegebene deterministisch-kontextfreie Sprache L in eine "ähnliche" Sprache zu überführen, die durch einen DKA mit Akzeptanz bei leerem Keller akzeptiert wird. Hierzu verwenden wir eine Endemarkierung \$, die an alle Wörter aus L angehängt wird und somit die Präfixeigenschaft erzwingt.

Für $L \subseteq \Sigma^*$ sei $L\$ = \{w\$: w \in L\}$, wobei \$ ein Zeichen ist, das nicht in Σ liegt. Tatsächlich gilt nun:

Lemma 3.38 (DCFL mit Endemarkierung). *Ist L deterministisch kontextfrei, so gibt es einen DKA $\mathcal{K}_\varepsilon$ mit $\mathcal{L}_\varepsilon(\mathcal{K}_\varepsilon) = L\$$.*

Beweis. Sei $\mathcal{K} = (Q, \Sigma, \Gamma, \delta, q_0, \#, F)$ ein DKA mit $L = \mathcal{L}(\mathcal{K})$. Wir konstruieren nun einen DKA $\mathcal{K}_\varepsilon$ mit Akzeptanz bei leerem Keller, der $\mathcal{K}$ simuliert und den Keller leert, sobald $\mathcal{K}$ einen

Endzustand erreicht und \$ gelesen wird. Die formale Definition von $\mathcal{K}_\varepsilon$ ist wie folgt:

$$\mathcal{K}_\varepsilon = (Q \cup \{q_0', q_F\}, \Sigma \cup \{\$\}, \Gamma', \delta', q_0', \#)$$

Das Kelleralphabet ist $\Gamma' = \Gamma \cup \{\$\}$, wobei wir $\$ \notin \Gamma$ annehmen. Das zusätzliche Symbol \$ (als Kellersymbol) dient zur Markierung des Kellerbodens während der Simulation. Die Übergangsfunktion δ' ist wie folgt:

$$\begin{array}{lcll} \delta'(q_0', \varepsilon, \#) & = & (q_0, \#\$) & \\ \delta'(p, \$, B) & = & (q_F, B) & \text{für } p \in F \\ \delta'(q_F, \varepsilon, B) & = & (q_F, \varepsilon) & \end{array} \qquad \begin{array}{lcl} \delta'(q, a, A) & = & \delta(q, a, A) \\ \delta'(q, \varepsilon, A) & = & \delta(q, \varepsilon, A) \end{array}$$

für alle $q \in Q$, $a \in \Sigma$, $A \in \Gamma$ und $B \in \Gamma \cup \{\$\}$. Zu zeigen ist nun, dass $\mathcal{L}_\varepsilon(\mathcal{K}_\varepsilon) = L\$$.

"$\supseteq$": Sei $x \in L\$$, also $x = w\$$ für ein Wort $w \in L$. Wegen $L = \mathcal{L}(\mathcal{K})$ gilt:

$$(q_0, w, \#) \vdash_{\mathcal{K}}^* (p, \varepsilon, x)$$

für einen Endzustand p von $\mathcal{K}$ und ein Wort $x \in \Gamma^*$. Dann gilt:

$$(q_0', w\$, \#) \vdash_{\mathcal{K}_\varepsilon} (q_0, w\$, \#\$) \vdash_{\mathcal{K}_\varepsilon}^* (p, \$, x\$) \vdash_{\mathcal{K}_\varepsilon} (q_F, \varepsilon, x\$) \vdash_{\mathcal{K}_\varepsilon}^* (q_F, \varepsilon, \varepsilon)$$

Also $x = w\$ \in \mathcal{L}_\varepsilon(\mathcal{K}_\varepsilon)$.

"$\subseteq$": Sei nun $x \in \mathcal{L}_\varepsilon(\mathcal{K}_\varepsilon)$. Da q_0' eine ε-Transition hat, gilt dann:

$$(q_0', x, \#) \vdash_{\mathcal{K}_\varepsilon} (q_0, x, \#\$) \vdash_{\mathcal{K}_\varepsilon}^* (q, \varepsilon, \varepsilon)$$

für einen Zustand q von $\mathcal{K}_\varepsilon$. Die einzige Möglichkeit das Kellersymbol \$ aus dem Keller zu entfernen sind die Transitionen $\delta'(p, \varepsilon, \$) = (q_F, \varepsilon)$ für die Endzustände p von $\mathcal{K}$. Weiter ist q_F absorbierend, d.h., ein Zustand, von dem kein anderer erreichbar ist, und q_F hat eine ε-Transition. Ferner wird \$ niemals auf den Keller gelegt, abgesehen von der ε-Transition vom Anfangszustand q_0'. Daher muss $q = q_F$ gelten und es einen Zustand $p \in F$ und Wörter $w \in \Sigma^*$ und $y \in \Gamma^*$ geben, so dass $x = w\$$ und

$$(q_0, w\$, \#\$) \vdash_{\mathcal{K}_\varepsilon}^* (p, \$, y\$) \vdash_{\mathcal{K}_\varepsilon} (q_F, \varepsilon, y\$) \vdash_{\mathcal{K}_\varepsilon}^* (q_F, \varepsilon, \varepsilon)$$

Da δ und δ' auf $Q \times \Sigma \times \Gamma$ übereinstimmen, folgt $(q_0, x, \#) \vdash_{\mathcal{K}}^* (p, \varepsilon, y\$)$ und somit $x \in \mathcal{L}(\mathcal{K}) = L$. Also ist $w = x\$ \in L\$$. □

Zusammenfassung der Konzepte aus Kapitel 3

Eine zentrale Rolle bei der Untersuchung von kontextfreien Sprachen spielen Normalformen kontextfreier Grammatiken, wie Chomsky oder Greibach Normalform. Die Chomsky Normalform haben wir als Ausgangspunkt für einen Algorithmus für das Wort- und Endlichkeitsproblem genutzt. Das Leerheitsproblem ist für kontextfreie Grammatiken ebenfalls effizient lösbar. Die meisten anderen algorithmischen Fragestellungen für kontextfreie Grammatiken (z.B. das Äquivalenz- oder Inklusionsproblem oder die Frage, ob die erzeugte Sprache einer kontextfreien Grammatik regulär ist) sind jedoch nicht algorithmisch lösbar. Weiter diente uns die Chomsky

Normalform für den Nachweis des Pumping Lemmas, das eine notwendige Bedingung für kontextfreie Sprachen liefert und somit hilfreich sein kann zu zeigen, dass eine gegebene Sprache nicht kontextfrei ist.

Das zu den kontextfreien Sprachen gehörende Automatenmodell ist der Kellerautomat. Im Wesentlichen ist dies eine Erweiterung vom ε-NFA um ein im LIFO-Prinzip organisiertes Speichermedium. Im Gegensatz zu endlichen Automaten sind die nichtdeterministische und deterministische Version von Kellerautomaten nicht gleichmächtig. Die beiden Akzeptanzkriterien "Akzeptanz über Endzustände" und "Akzeptanz bei leerem Keller" sind für die nichtdeterministische Version äquivalent. Für deterministische Kellerautomaten ist Akzeptanz über Endzustände jedoch mächtiger. Die Klasse der Sprachen, die durch einen DKA mit Akzeptanz bei leerem Keller akzeptiert werden, ist genau die Klasse der DKA-Sprachen mit der Präfixeigenschaft.

Die Klasse der kontextfreien Sprachen ist unter Vereinigung, Kleeneabschluss und Konkatenation abgeschlossen, jedoch nicht unter Durchschnitt und Komplement. Die Klasse der deterministisch kontextfreien Sprachen ist nur unter Komplement abgeschlossen, aber unter keinem der anderen vier gängigen Verknüpfungsoperatoren (Vereinigung, Schnitt, Konkatenation, Kleeneabschluss).

Im Kontext des Übersetzerbaus spielen spezielle Grammatiktypen für deterministisch kontextfreie Spachen eine zentrale Rolle. Der Parser kann dann das Wortproblem durch Algorithmen lösen, die ähnlich wie ein DKA arbeiten und deutlich effizienter sind als der CYK-Algorithmus oder andere Algorithmen für die vollständige Klasse kontextfreier Sprachen, dargestellt durch (Normalformen von) Typ 2 Grammatiken.

Sprachen vom Typ 0 und Typ 1

In den letzten beiden Kapiteln haben wir uns ausführlich mit den beiden unteren Ebenen (Typ 3 und Typ 2) der Chomsky-Hierarchie befasst. Bevor wir den ersten Teil zu formalen Sprachen und Automaten beenden erwähnen wir hier kurz einige Ergebnisse zu Automatenmodellen, Abschlusseigenschaften und algorithmischen Fragestellungen für die Sprachenklassen der beiden oberen Ebenen der Chomsky-Hierarchie (Typ 0 und Typ 1).

Das zur Klasse der Sprachen vom Typ 0 gehörende Automatenmodell sind *Turingmaschinen*. Diese werden ausführlich im Modul "Theoretische Informatik und Logik" besprochen. Sie bestehen aus einer endlichen Kontrolle und nutzen ein unbeschränktes Band als Eingabemedium und Arbeitsspeicher. Auf die Zellen des Bandes kann sequentiell über einen Lese-/Schreibkopf zugegriffen werden. Hierin liegt der wesentliche Unterschied zu Kellerautomaten, bei denen die Zugriffe auf den Keller nur mittels der üblichen Kelleroperationen push und pop zulässig sind. Das Modell von Turingmaschinen ist nicht nur im Kontext formaler Sprachen von Bedeutung, sondern liefert zugleich ein universelles Rechnermodell, welches zu modernen Rechnern hinsichtlich der Berechenbarkeit äquivalent ist. Der Preis für den Zugewinn an Ausdrucksstärke verglichen mit Kellerautomaten ist jedoch hoch. Alle interessanten algorithmischen Fragestellungen (darunter das Wort-, Leerheits- und Äquivalenzproblem u.v.m.) sind für Turingmaschinen unentscheidbar, d.h., nicht algorithmisch lösbar.

Das zu kontextsensitiven Sprachen gehörende Automatenmodell sind *linear beschränkte Automaten* (LBA), eine Variante von Turingmaschinen, die für die Bearbeitung eines Eingabeworts im Wesentlichen nur diejenigen Bandzellen, in denen initial ein Eingabezeichen steht, benutzen können. Für festes Eingabewort sind also die relevanten Bandzellen a-priori festgelegt. Dies ermöglicht den Entwurf von Algorithmen, um das Wortproblem für kontextsensitive Sprachen zu lösen.

Während die nichtdeterministische und deterministische Variante von Turingmaschinen äquivalent sind, ist die entsprechende Frage für LBA noch ungelöst. Die Klasse der kontextsensitiven Sprachen ist – wie die Klasse der regulären Sprachen – unter den fünf wichtigsten Kompositionsoperatoren Vereinigung, Schnitt, Komplement, Konkatenation und Kleeneabschluss abgeschlossen. Die Klasse der Sprachen vom Typ 0 ist abgeschlossen unter Vereinigung, Schnitt, Konkatenation und Kleeneabschluss, jedoch nicht unter Komplement.

2. Teil: Aussagenlogik

Die Logik ist eine sehr alte Wissenschaftsdisziplin, die sich mit der Lehre vom vernünftigen Schlussfolgern ("Beweisen") befasst. Klassische Problemstellungen der Logik sind die Frage nach Formalisierungen schlüssigen Denkens, oder nach Beweisregeln, welche maschinell realisierbar sind. Tief verwurzelt in der Informatik spielt die Logik in vielen Teilgebieten der Informatik traditionell eine zentrale Rolle. Beispiele sind die Logiksynthese der technischen Informatik, maschinelle Beweiser, die in der künstlichen Intelligenz oder Robotik eingesetzt werden, logische Kalküle beruhend auf Vor- und Nachbedingungen für sequentielle Programme (z.B. Hoare Logik), modale und temporale Logiken, die zur Spezifikation und Verifikation von parallelen Systemen dienen, sowie deduktive Datenbanken oder Logikprogrammiersprachen wie PROLOG oder DATALOG.

4 Aussagenlogik

In dieser Vorlesung befassen wir uns nur mit der Aussagenlogik. Trotz ihrer sehr einfachen Struktur spielt die Aussagenlogik eine wichtige Rolle in der Informatik und anderen Wissenschaftsdisziplinen. Zugleich bildet die Aussagenlogik das Grundgerüst reichhaltigerer Logiken wie der Prädikatenlogik, das es erlaubt über Relationen zwischen Objekten oder den Eigenschaften der Individuen einer Objektmenge zu sprechen.

4.1 Grundbegriffe der Aussagenlogik

Aussagenlogische Konzepte sind bereits aus mehreren anderen Vorlesungen bekannt. So sind z.B. Boolesche Ausdrücke, wie sie als Schleifenabbruchkriterium in imperativen Programmiersprachen vorkommen können, aussagenlogische Formeln. Die aus der technischen Informatik bekannten Schaltnetze sind graphische Darstellungen aussagenlogischer Formeln und Schaltfunktionen deren Semantik.

In diesem Abschnitt gehen wir auf die formale Syntax und Semantik aussagenlogischer Formeln und zugehörigen Grundbegriffen sowie Normalformen ein. Die folgenden Abschnitte befassen sich dann mit dem Sonderfall von Hornformeln, denen in der Logikprogrammierung eine zentrale Rolle zukommt, und algorithmischen Fragestellungen.

4.1.1 Syntax und Semantik

Die Grundbausteine aussagenlogischer Formeln sind atomare Aussagensymbole, die je nach Kontext wahr oder falsch sein können und durch logische Operatoren wie "und", "oder", "nicht" zu komplexen Aussagen verknüpft werden können.

Syntax der Aussagenlogik. Im Folgenden bezeichnet *AP* eine nicht-leere Menge von paarweise verschiedenen Booleschen Variablen, die in diesem Kontext *Aussagensymbole* oder *Atome* oder *atomare Formeln* genannt werden. Die Abkürzung *AP* steht für die englische Bezeichnung "atomic propositions". Intuitiv stehen die Aussagensymbole für Aussagen, die entweder wahr

oder falsch (aber nicht beides) sein können. Beispielsweise können sie die intuitive Bedeutung "Programmvariable X hat den aktuellen Wert 5" oder "NFA $\mathcal{M}$ befindet sich in Zustand q" haben. Zur Bildung komplexer Aussagen verwenden wir die Booleschen Operatoren $\neg$ (Negation) und $\wedge$ (Konjunktion, das logische "und"), aus denen alle anderen Booleschen Verknüpfer wie u.a. $\vee$ (Disjunktion, das logische "oder"), $\rightarrow$ (Implikation), $\leftrightarrow$ (Äquivalenz) herleitbar sind.

Wir verwenden hier meist Kleinbuchstaben am Ende des Alphabets wie x, y, z als Aussagensymbole, also Elemente von *AP*, und griechische Buchstaben am Anfang des Alphabets $\alpha, \beta, \gamma, \ldots$ für aussagenlogische Formeln.

Zur Angabe der Syntax aussagenlogischer Formeln verfahren wir ähnlich wie für reguläre Ausdrücke (Abschnitt 2.3). Wir geben zunächst eine induktive Definition aussagenlogischer Formeln und dann die abstrakte Syntax in BNF-ähnlicher Notation an. Die Menge aller aussagenlogischen Formeln über *AP* ist induktiv durch folgende vier Regeln definiert.

1. *true* ist eine aussagenlogische Formel über *AP*.
2. Jedes Aussagensymbol $x \in AP$ ist eine aussagenlogische Formel über *AP*.
3. Sind α und β aussagenlogische Formeln über *AP*, dann sind auch $(\neg\alpha)$ und $(\alpha \wedge \beta)$ aussagenlogische Formeln über *AP*.
4. Nichts sonst ist eine aussagenlogische Formel über *AP*.

Unterstellt man die Regeln für das Setzen von Klammern als Selbstverständlichkeit, so kann obige induktive Definition in kompakter Form in BNF-ähnlicher Notation geschrieben werden (abstrakte Syntax), wobei α und β als Metasymbole für Formeln und x als Metasymbol für die Atome in *AP* fungieren.

$$\alpha \ ::= \ \mathit{true} \ \mid \ x \ \mid \ \neg\alpha \ \mid \ \alpha \wedge \beta$$

Abgeleitete Operatoren. Neben dem Verzicht auf überflüssige Klammern sind vereinfachende Schreibweisen, die sich durch die Verwendung zusätzlicher Symbole ergeben, üblich. Diese werden aus der Konstanten *true* und den beiden Basisoperatoren $\wedge$ und $\neg$ hergeleitet. Die wichtigsten abgeleiteten Operatoren sind:

$$\begin{aligned}
\mathit{false} &\stackrel{\text{def}}{=} \neg\mathit{true} && \\
\alpha \vee \beta &\stackrel{\text{def}}{=} \neg(\neg\alpha \wedge \neg\beta) && \text{(Disjunktion)} \\
\alpha \rightarrow \beta &\stackrel{\text{def}}{=} \neg\alpha \vee \beta && \text{(Implikation)} \\
\alpha \leftrightarrow \beta &\stackrel{\text{def}}{=} (\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta) && \text{(Äquivalenz)}
\end{aligned}$$

Prioritätsregeln zur Vermeidung von Klammern. Zur Einsparung von Klammern treffen wir die Vereinbarung, dass die Negation am stärksten bindet, dann die Operatoren $\wedge$, $\vee$, $\rightarrow$ und $\leftrightarrow$ (in dieser Reihenfolge). Z.B. steht $\neg x \wedge y \vee z$ für $((\neg x) \wedge y) \vee z$ oder $x \wedge y \leftrightarrow \neg y \vee z$ für $(x \wedge y) \leftrightarrow ((\neg y) \vee z)$.

Semantik der Aussagenlogik. Bis hier haben wir nur syntaktische Konstrukte der Aussagenlogik behandelt. Die Interpretation der Konstanten *true* als eine Aussage, die immer wahr ist, oder der Symbole $\neg$ und $\wedge$ als Negation bzw. Konjunktion wird erst durch die Angabe einer entsprechenden Semantik gerechtfertigt. Ebenso ist die Vereinfachung von Formeln durch bekannte "Gesetze" (etwa doppelte Verneinung $\neg\neg\alpha \rightsquigarrow \alpha$) oder der Übergang zu Normalformen erst auf semantischer Ebene sinnvoll, auf der den Symbolen der Logik eine Bedeutung zugewiesen wird. Intuitiv stehen die Atome in *AP* für Aussagen, die je nach Kontext wahr oder falsch sein können. Der Kontext wird durch eine Abbildung formalisiert, die jedem Aussagensymbol einen Wahrheitswert 0 ("falsch") oder 1 ("wahr") zuordnet.

Definition 4.1 (Belegung, Interpretation, Wahrheitswert einer Formel). Eine *Interpretation* oder *Belegung* für die Aussagenmenge *AP* ist eine Abbildung $I : AP \to \{0,1\}$. Die Belegungen für $AP = \{x_1,\ldots,x_n\}$ geben wir oftmals in der Form $[x_1 = b_1,\ldots,x_n = b_n]$ oder $x_1^I = b_1$, …, $x_n^I = b_n$ an und meinen damit die Abbildung $I : \{x_1,\ldots,x_n\} \to \{0,1\}$, deren Funktionswerte durch $I(x_i) = b_i$, $i = 1,\ldots,n$, gegeben sind. Der durch eine Belegung I induzierte Wahrheitswert $\alpha^I \in \{0,1\}$ für gegebene aussagenlogische Formel α über *AP* ist durch strukturelle Induktion definiert, siehe Abbildung 28. ■

$$
\begin{array}{lcll}
\mathit{true}^I & \stackrel{\text{def}}{=} & 1 & \\
x^I & \stackrel{\text{def}}{=} & I(x) & \\
(\neg\alpha)^I & \stackrel{\text{def}}{=} & 1-\alpha^I & = \begin{cases} 1 & : \text{ falls } \alpha^I = 0 \\ 0 & : \text{ sonst} \end{cases} \\
(\alpha\wedge\beta)^I & \stackrel{\text{def}}{=} & \min\{\alpha^I,\beta^I\} & = \begin{cases} 1 & : \text{ falls } \alpha^I = 1 \text{ und } \beta^I = 1 \\ 0 & : \text{ sonst} \end{cases}
\end{array}
$$

Abbildung 28: Definition der Wahrheitswerte aussagenlogischer Formeln unter Belegung I

Die formale Definition entspricht genau der intuitiven Bedeutung des Negations- und Konjunktionsoperators. Dies kann man sich klarmachen, indem man *Wertetafeln* betrachtet und die oben angegebene Definition von α^I einsetzt.

α^I	$(\neg\alpha)^I$
0	1
1	0

α^I	β^I	$(\alpha\wedge\beta)^I$
0	0	0
0	1	0
1	0	0
1	1	1

Für die abgeleiteten Operatoren ergibt sich die erwartete Semantik, siehe Abbildung 29.

Offenbar spielen für den Wahrheitswert α^I nur die Werte $I(x) = x^I$ der in α vorkommenden Atome $x \in AP$ eine Rolle. Für festes α genügt es daher, die Werte x^I der in α vorkommenden Atome vorzugeben, um den Wahrheitswert von α unter I zu bestimmen.

Wir betrachten die Formel $\alpha = (x\vee y)\wedge(\neg x\vee\neg z)$. Die Wahrheitswerte von α unter den acht möglichen Belegungen für x, y, z ergeben sich aus folgender Wertetafel:

$$
\begin{aligned}
\mathit{false}^{\mathrm{I}} &= 0 \\
(\alpha \vee \beta)^{\mathrm{I}} &= \max\{\alpha^{\mathrm{I}}, \beta^{\mathrm{I}}\} = \begin{cases} 1 & : \text{ falls } \alpha^{\mathrm{I}} = 1 \text{ oder } \beta^{\mathrm{I}} = 1 \\ 0 & : \text{ sonst} \end{cases} \\
(\alpha \to \beta)^{\mathrm{I}} &= \begin{cases} 1 & : \text{ falls } \alpha^{\mathrm{I}} \leqslant \beta^{\mathrm{I}} \\ 0 & : \text{ sonst} \end{cases} \\
(\alpha \leftrightarrow \beta)^{\mathrm{I}} &= \begin{cases} 1 & : \text{ falls } \alpha^{\mathrm{I}} = \beta^{\mathrm{I}} \\ 0 & : \text{ sonst} \end{cases}
\end{aligned}
$$

Abbildung 29: Semantik einiger abgeleiteter Operatoren

x^{I}	y^{I}	z^{I}	$(x \vee y)^{\mathrm{I}}$	$(\neg x \vee \neg z)^{\mathrm{I}}$	α^{I}
0	0	0	0	1	0
0	0	1	0	1	0
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	0

Erfüllende Belegung, Modell. Falls $\alpha^{\mathrm{I}} = 1$, so nennen wir I eine *erfüllende Belegung* oder ein *Modell* für α. Z.B. ist $\mathrm{I} = [x = 1, y = 0, z = 0]$ ein Modell für $(x \vee y) \wedge (\neg x \vee \neg z)$, jedoch ist I nicht erfüllend für $(x \wedge y) \vee (\neg x \wedge \neg z)$. Wir verwenden oftmals lässige Sprechweisen wie "α ist falsch unter I", falls $\alpha^{\mathrm{I}} = 0$. Entsprechend machen wir von Formulierungen der Art "α ist unter I wahr" oder "α ist unter I erfüllt" Gebrauch, falls I ein Modell für α ist.

Definition 4.2 (Erfüllbarkeit, Gültigkeit). Sei α eine aussagenlogische Formel über *AP*. α heißt *erfüllbar*, wenn es eine erfüllende Belegung für α gibt, also wenn eine Belegung I für die in α vorkommenden Atome existiert, so dass $\alpha^{\mathrm{I}} = 1$, Formel α heißt *unerfüllbar*, falls α nicht erfüllbar ist, also wenn $\alpha^{\mathrm{I}} = 0$ für alle Belegungen I, Formel α wird *Tautologie* oder *gültig* genannt, wenn $\alpha^{\mathrm{I}} = 1$ für alle Belegungen I. ■

Beispielsweise sind $\alpha = (x \vee y) \wedge \neg(\neg y \to x)$ oder $\beta = x \leftrightarrow \neg x$ unerfüllbar. Das formale Argument für die Unerfüllbarkeit von β ist, dass unter jeder Belegung I die atomare Formel x und deren Negat $\neg x$ unterschiedliche Wahrheitswerte haben, also $x^{\mathrm{I}} \neq (\neg x)^{\mathrm{I}}$ und somit $\beta^{\mathrm{I}} = (x \leftrightarrow \neg x)^{\mathrm{I}} = 0$. Zum Nachweis der Unerfüllbarkeit von α betrachten wir eine beliebige Interpretation I und zeigen, dass α unter I falsch ist.

- Falls $x^{\mathrm{I}} = 1$, so ist $(\neg y \to x)^{\mathrm{I}} = 1$ und somit $\neg(\neg y \to x)^{\mathrm{I}} = 0$ und daher $\alpha^{\mathrm{I}} = 0$.
- Wir nehmen nun $x^{\mathrm{I}} = 0$ an. Falls $y^{\mathrm{I}} = 0$, so ist $(x \vee y)^{\mathrm{I}} = 0$ und somit $\alpha^{\mathrm{I}} = 0$. Falls $y^{\mathrm{I}} = 1$, so ist $(\neg y \to x)^{\mathrm{I}} = (\neg\neg y \vee x)^{\mathrm{I}} = 1$ und somit $\neg(\neg y \to x)^{\mathrm{I}} = 0$. Wir erhalten ebenfalls $\alpha^{\mathrm{I}} = 0$.

Die Formel $\gamma = x \vee \neg(x \wedge y)$ ist eine Tautologie. Dies folgt aus der Tatsache, dass jede Belegung I für x und y entweder erfüllend für die linke Teilformel x ist (nämlich dann, wenn $x^I = 1$) oder ein Modell der rechten Teilformel $\neg(x \wedge y)$ ist (nämlich dann, wenn $x^I = 0$).

Die Formeln $\alpha = x \vee \neg y$ und $\beta = x \wedge \neg y$ sind erfüllbar (da $\alpha^I = \beta^I = 1$ für $x^I = 1$ und $y^I = 0$), aber nicht gültig (da $\alpha^I = \beta^I = 0$ für $x^I = 0$ und $y^I = 1$).

Wir fassen zusammen. Der Nachweis der Erfüllbarkeit oder Nicht-Gültigkeit kann durch Angabe eines Modells bzw. einer nicht-erfüllenden Belegung erfolgen. Hingegen erfordert der Nachweis der Unerfüllbarkeit oder Gültigkeit die Betrachtung aller Interpretationen, für die dann zu zeigen ist, dass die betreffende Formel stets falsch (Unerfüllbarkeit) bzw. stets wahr (Gültigkeit) ist. Weitere Beweisoptionen für den Nachweis der Unerfüllbarkeit bzw. Gültigkeit ergeben sich durch die semantischen Begriffe der Äquivalenz von Formeln oder der logischen Konsequenz. Darauf gehen wir später ein. Zunächst stellen wir folgenden offensichtlichen Zusammenhang zwischen Gültigkeit und Unerfüllbarkeit fest:

$$\begin{array}{lll} \alpha \text{ ist unerfüllbar} & \text{gdw} & \alpha^I = 0 \text{ für alle Belegungen I} \\ & \text{gdw} & (\neg\alpha)^I = 1 \text{ für alle Belegungen I} \\ & \text{gdw} & \neg\alpha \text{ ist gültig} \end{array}$$

Beispiel 4.3 (n-Damen-Problem). Das n-Damen-Problem fragt nach einer Platzierung von n Damen auf einem $n \times n$-Schachbrett, so dass sich die Damen nicht gegenseitig bedrohen. Abweichend von der sonst üblichen Nummerierung der Felder durch Buchstaben und Zahlen denken wir uns das Schachbrett als $n \times n$-Matrix und verwenden Nummern $1, \ldots, n$ für die Zeilen und Spalten.

	1	2	3	4	5
1		×			
2					×
3			×		
4	×				
5				×	

mögliche Lösung für $n = 5$:

je eine Dame in den Feldern
$(1,2)$, $(2,5)$, $(3,3)$, $(4,1)$, $(5,4)$

Das Ziel ist nun eine aussagenlogische Formel α_n anzugeben, deren Modelle genau den Lösungen, also den legitimen Plazierungen der n Damen, entsprechen. Hierzu verwenden wir die Aussagensymbole $x_{i,j}$ für $1 \leqslant i, j \leqslant n$ mit der intuitiven Bedeutung, dass $x_{i,j}$ für die Aussage "auf Feld (i,j) befindet sich eine Dame" steht. Die Formel α_n codiert nun die Eigenschaften, dass in jeder Zeile eine Dame steht (damit ist sichergestellt, dass tatsächlich n Damen plaziert werden) und dass keine Dame auf einem Feld steht, das durch eine andere Dame in einem Spielzug erreichbar ist: [12]

$$\alpha_n = \underbrace{\bigwedge_{1 \leqslant i \leqslant n} \bigvee_{1 \leqslant j \leqslant n} x_{i,j}}_{\substack{\text{in jeder Zeile } i \text{ steht} \\ \text{wenigstens eine Dame}}} \quad \wedge \quad \underbrace{\bigwedge_{1 \leqslant i,j \leqslant n} \big(x_{i,j} \rightarrow \bigwedge_{(k,\ell) \in H(i,j)} \neg x_{k,\ell} \big)}_{\substack{\text{die Damen bedrohen sich} \\ \text{nicht gegenseitig}}}.$$

[12] Wir verwenden hier mehrstellige Konjunktions- und Disjunktionsoperatoren. Diese können durch beliebige Klammerung auf den zweistelligen Fall zurückgeführt werden. Eine formale Rechtfertigung hierfür wird später durch die Äquivalenzgesetze (Assoziativität und Kommutativität) für Konjunktionen und Disjunktionen gegeben. Siehe Abbildung 30.

Für jedes Feld (i,j) definieren wir die Menge $H(i,j)$ als diejenige Teilmenge von $\{1,\ldots,n\}^2$, die genau aus solchen Feldern (k,ℓ) besteht, die von einer Dame in Feld (i,j) erreichbar sind. Es gilt also $(k,\ell) \in H(i,j)$ genau dann, wenn sich Damen an den Feldpositionen (i,j) und (k,ℓ) gegenseitig bedrohen:

$$\begin{aligned} H(i,j) \;=\; & \{(i,\ell) : 1 \leqslant \ell \leqslant n,\, \ell \neq j\} \quad \cup \\ & \{(k,j) : 1 \leqslant k \leqslant n,\, k \neq i\} \quad \cup \\ & \{(k,\ell) : 1 \leqslant k,\ell \leqslant n,\, (k,\ell) \neq (i,j),\, k-\ell = i-j\,\} \cup \\ & \{(k,\ell) : 1 \leqslant k,\ell \leqslant n,\, (k,\ell) \neq (i,j),\, k+\ell = i+j\,\} \end{aligned}$$

Die erfüllenden Belegungen von α_n stehen dann für die Lösungen des n-Damen-Problems. Insbesondere ist α_n genau dann erfüllbar, wenn das n-Damen-Problem lösbar ist. Z.B. entspricht die oben angegebene Lösung des 5-Damen-Problems mit je einer Dame in den Feldern $(1,2)$, $(2,5)$, $(3,3)$, $(4,1)$ und $(5,4)$ der Interpretation I, die durch

$$x_{1,2}^I = x_{2,5}^I = x_{3,3}^I = x_{4,1}^I = x_{5,4}^I = 1 \;\text{ und }\; x_{k,\ell}^I = 0 \text{ für alle anderen Felder}$$

gegeben ist. Man kann sich nun davon überzeugen, dass I tatsächlich ein Modell für α_5 ist. ■

Beispiel 4.4 (Pigeonhole-Formeln). Die Unlösbarkeit des aus der Mathematik bekannten Taubenschlagprinzips, das auch unter dem Stichwort Schubfachprinzip bekannt ist, lässt sich mittels der Aussagenlogik nachvollziehen. Hierbei geht es um das Problem, n Tauben auf $n-1$ Löcher (Taubenschläge) zu verteilen, so dass in jedem Loch höchstens eine Taube sitzt und jede Taube sich tatsächlich in einem der Löcher befindet. Wir nummerieren die Tauben von 1 bis n und die Löcher von 1 bis $n-1$ und verwenden das Atom $x_{i,j}$ mit $1 \leqslant i < n-1$ und $1 \leqslant j \leqslant n$ für die atomare Aussage, dass sich Taube j in Loch i befindet. Das Taubenschlagprinzip kann nun durch folgende aussagenlogische Formel dargestellt werden:

$$\alpha_n \;=\; \bigwedge_{1\leqslant i<n} \bigwedge_{1\leqslant j<k\leqslant n} \underbrace{(\neg x_{i,j} \vee \neg x_{i,k})}_{\substack{\text{Taube j oder Taube k}\\ \text{ist nicht in Loch i}}} \;\wedge\; \bigwedge_{1\leqslant j\leqslant n} \underbrace{(x_{1,j} \vee \ldots \vee x_{n-1,j})}_{\substack{\text{Taube j sitzt}\\ \text{in einem Loch}}}.$$

Die linke Teilformel ist genau dann wahr, wenn in jedem Loch maximal eine Taube ist, während die rechte Teilformel genau dann wahr ist, wenn sich jede Taube in einem Loch befindet.

Wir zeigen nun, dass α_n unerfüllbar ist. Angenommen I ist eine Belegung mit $\alpha_n^I = 1$. Für jedes $j \in \{1,\ldots,n\}$ ist dann $(x_{1,j} \vee \ldots \vee x_{n-1,j})^I = 1$, d.h., für jede Taube j gibt es ein Loch i_j mit $x_{i_j,j}^I = 1$. Wegen $\{i_1,\ldots,i_n\} \subseteq \{1,\ldots,n-1\}$ gibt es ein $i \in \{1,\ldots,n-1\}$ und Tauben j und k mit $j < k$, die beide in Loch i sitzen. Also $i_j = i_k = i$. Dann aber ist

$$x_{i,j}^I = x_{i,k}^I = 1 \;\text{ und somit }\; (\neg x_{i,j} \vee \neg x_{i,k})^I = 0.$$

Dies ist nicht möglich, da $\alpha_n^I = 1$. Widerspruch. ■

Definition 4.5 (Äquivalenz $\equiv$ von Formeln). Zwei Formeln α und β, die unter jeder Interpretation denselben Wahrheitswert haben, werden *semantisch äquivalent* oder kurz *äquivalent*, i.Z. $\alpha \equiv \beta$, genannt. D.h.:

$$\alpha \equiv \beta \quad \text{gdw} \quad \text{für alle Belegungen I gilt: } \alpha^I = \beta^I \qquad \blacksquare$$

Beispielsweise sind die Formeln $x \wedge \neg\neg y$ und $x \wedge y$ äquivalent, da die Modelle beider Formeln genau solche Belegungen I sind, für die $x^I = y^I = 1$ gilt. Für beliebige Formeln α und β sind $\alpha \leftrightarrow \beta$ und $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$ äquivalent, da für jede Interpretation I gilt:

$$\begin{aligned}
(\alpha \leftrightarrow \beta)^I = 1 \quad & \text{gdw} \quad \alpha^I = \beta^I \\
& \text{gdw} \quad \alpha^I \leqslant \beta^I \text{ und } \beta^I \leqslant \alpha^I \\
& \text{gdw} \quad (\alpha \rightarrow \beta)^I = 1 \text{ und } (\beta \rightarrow \alpha)^I = 1 \\
& \text{gdw} \quad \big((\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)\big)^I = 1
\end{aligned}$$

Für komplexere Formeln ist es oftmals nicht ganz so einfach zu belegen, dass sie dieselben Modelle haben. Für den formalen Nachweis, dass $\alpha \equiv \beta$ stehen verschiedene Beweisoptionen zur Verfügung. Zum einen kann man zeigen, dass jedes Modell für α ein Modell für β ist, und umgekehrt. Alternativ kann man eine beliebige Interpretation betrachten und dann zeigen, dass $\alpha^I = \beta^I$, indem man zwischen den Fällen $\alpha^I = 1$ und $\alpha^I = 0$ unterscheidet. Als Beispiel für die erste Beweisoption betrachten wir die Formeln:

$$\alpha = x \wedge (\neg y \vee \neg x) \quad \text{und} \quad \beta = x \wedge \neg y$$

Der Nachweis für $\alpha \equiv \beta$ besteht nun aus folgenden beiden Teilen:

1. Jedes Modell für α ist ein Modell für β.

   Beweis: Ist I eine Interpretation mit $\alpha^I = 1$, so ist $x^I = 1$ und $\gamma^I = 1$, wobei $\gamma = \neg y \vee \neg x$. Wegen $(\neg x)^I = 0$ und $\gamma^I = 1$ ist $(\neg y)^I = 1$, also $y^I = 0$. Nun folgt $\beta^I = (x \wedge \neg y)^I = 1$.

2. Jedes Modell für β ist ein Modell für α.

   Beweis: Ist I eine Interpretation mit $\beta^I = 1$, so ist $x^I = (\neg y)^I = 1$, also $y^I = 0$. Dann aber ist $(\neg y \vee \neg x)^I = 1$ und $\alpha^I = 1$.

Die zweite Beweisoption machen wir uns am Beispiel der Formeln $\gamma = \alpha \wedge (\beta \vee \alpha)$ und α klar, wobei α und β beliebige Formeln sind. Wir zeigen $\alpha \equiv \gamma$, indem wir nachweisen, dass $\alpha^I = \gamma^I$ für jede Belegung I und unterscheiden dabei zwischen den möglichen beiden Wahrheitswerten für α unter I.

1. Ist $\alpha^I = 1$, so ist $\gamma^I = 1$.

   Beweis: Ist I eine Interpretation mit $\alpha^I = 1$, so ist $(\beta \vee \alpha)^I = 1$ und somit $\gamma^I = 1$.

2. Ist $\alpha^I = 0$, so ist $\gamma^I = 0$.

   Beweis: Für jede Belegung I mit $\alpha^I = 0$ ist $(\alpha \wedge \sigma)^I = 0$ für jede Formel σ. Insbesondere gilt $\gamma^I = 0$.

Eine andere Beweisoption für die Äquivalenz zweier Formeln α und β ist der Einsatz grundlegender Äquivalenzregeln (siehe unten). Hierzu führt man α durch sukzessives Anwenden von

Idempotenz

$$\alpha \wedge \alpha \equiv \alpha \qquad \alpha \vee \alpha \equiv \alpha$$

Kommutativität

$$\alpha \wedge \beta \equiv \beta \wedge \alpha \qquad \alpha \vee \beta \equiv \beta \vee \alpha$$

Assoziativität

$$\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma \qquad \alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$$

Absorption

$$\alpha \wedge (\alpha \vee \beta) \equiv \alpha \qquad \alpha \vee (\alpha \wedge \beta) \equiv \alpha$$

Doppelte Verneinung

$$\neg\neg\alpha \equiv \alpha$$

Dualität der Konstanten

$$\neg \mathit{false} \equiv \mathit{true} \qquad \neg \mathit{true} \equiv \mathit{false}$$

Dualität von $\wedge$ und $\vee$
(De Morgansche Gesetze)

$$\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta \qquad \neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$$

Distributivität

$$\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma) \qquad (\alpha \wedge \beta) \vee \gamma \equiv (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$$

$$\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma) \qquad (\alpha \vee \beta) \wedge \gamma \equiv (\alpha \wedge \gamma) \vee (\beta \wedge \gamma)$$

Abbildung 30: Einige Äquivalenzgesetze der Aussagenlogik

Äquivalenzregeln in die Formel β über. Bevor wir auf die wichtigsten Äquivalenzgesetze eingehen, erwähnen wir, dass der formale Nachweis für die Inäquivalenz zweier Formeln α oder β meist deutlich einfacher ist. Um $\alpha \not\equiv \beta$ zu zeigen, genügt es nämlich *eine* Belegung I anzugeben, unter der eine der beiden Formeln wahr und die andere falsch ist. Z.B. gilt

$$(x \to y) \wedge (y \to z) \not\equiv x \to z,$$

da die linke Formel unter der Belegung I mit $x^I = z^I = 1$ und $y^I = 0$ falsch ist (da nämlich $(x \to y)^I = 0$), die rechte Formel aber unter I wahr ist. Ebenso gilt

$$(x \to y) \wedge (y \leftrightarrow z) \not\equiv x \to z,$$

da die Interpretation I mit $x^I = y^I = 0$ und $z^I = 1$ ein Modell für $x \to z$ ist, aber nicht-erfüllend für die linke Formel (da $(y \leftrightarrow z)^I = 0$).

In Abbildung 30 auf Seite 117 sind einige Äquivalenzregeln angegeben. Dabei sind α, β, γ beliebige aussagenlogische Formeln.[13] Das Assoziativ- und Kommutativgesetz für die Disjunktion $\wedge$ und die Konjunktion $\vee$ rechtfertigen den Verzicht auf Klammern und Schreibweisen wie

$$\bigwedge_{1 \leqslant i \leqslant n} \alpha_i \text{ oder } \alpha_1 \wedge \ldots \wedge \alpha_n.$$

[13] Da wir in der Syntax aussagenlogischer Formeln nur die Konstante *true* und die Basisoperatoren $\neg$ und $\wedge$ verwendet haben, und *false* durch $\neg$*true* bzw. den Disjunktionsoperator $\vee$ durch $\alpha \vee \beta = \neg(\neg\alpha \wedge \neg\beta)$ definiert haben, gelten die Äquivalenzgesetze $\neg$*false* $\equiv$ *true* und das De Morgan'sche Gesetz $\alpha \vee \beta \equiv \neg(\neg\alpha \wedge \neg\beta)$ trivialerweise. Wir haben sie zur Vollständigkeit hier dennoch aufgeführt.

Darüber hinaus verwendet man häufig Formeln des Typs

$$\alpha \;=\; \bigwedge_{i\in J} \alpha_i.$$

Dabei ist J eine beliebige endliche Indexmenge. Ist J nichtleer, dann steht α für eine Formel $\alpha_{i_1} \wedge \ldots \wedge \alpha_{i_k}$, wobei $J = \{i_1, \ldots, i_k\}$ und $i_1, \ldots, i_k$ paarweise verschieden sind. Für $J = \varnothing$ ist die Vereinbarung

$$\bigwedge_{i\in\varnothing} \alpha_i \stackrel{\text{def}}{=} \textit{true}, \qquad \bigvee_{i\in\varnothing} \alpha_i \stackrel{\text{def}}{=} \textit{false}.$$

Weitere Äquivalenzgesetze sind:

- Ist α eine Tautologie, so gilt: $\alpha \vee \beta \;\equiv\; \textit{true}$ und $\alpha \wedge \beta \;\equiv\; \beta$
- Ist α unerfüllbar, so gilt: $\alpha \vee \beta \;\equiv\; \beta$ und $\alpha \wedge \beta \;\equiv\; \textit{false}$

Aus den beiden zuletzt genannten Äquivalenzgesetzen zusammen mit den Distributivgesetzen ergibt sich:

$$\alpha \wedge (\beta \vee \neg\alpha) \;\equiv\; (\alpha \wedge \beta) \vee \underbrace{(\alpha \wedge \neg\alpha)}_{\text{unerfüllbar}} \;\equiv\; \alpha \wedge \beta$$

und $\alpha \vee (\beta \wedge \neg\alpha) \equiv \alpha \vee \beta$.

Zur Illustration, wie anhand der Äquivalenzgesetze Formeln vereinfacht oder die Äquivalenz zweier Formeln nachgewiesen werden kann, betrachten wir zwei Beispiele.

$$\begin{aligned}
& (x \vee y) \wedge (\neg x \vee y) \;\wedge\; (x \vee \neg y) \wedge (\neg x \vee \neg y) \\
\equiv\;& (\underbrace{(x \wedge \neg x)}_{\text{unerfüllbar}} \vee y)) \;\wedge\; ((\underbrace{x \wedge \neg x}_{\text{unerfüllbar}}) \vee \neg y) \\
\equiv\;& y \wedge \neg y \;\equiv\; \textit{false}
\end{aligned}$$

Die erste Umformung beruht auf den Distrubutivgesetzen. Da *false* unerfüllbar ist, ist damit auch die Ausgangsformel $(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$ unerfüllbar. In diesem Sinn können die Äquivalenzregeln auch für den Nachweis der Unerfüllbarkeit einer Formel dienen. Analog kann der Nachweis der Gültigkeit einer Formel erbracht werden, indem man α durch Äquivalenzumformungen in *true* überführt. Erfüllbare, nicht-tautologische Formeln können zwar nicht in *true* oder *false* überführt werden, jedoch können die Äquivalenzregeln eingesetzt werden, um Formeln zu vereinfachen. Wir zeigen nun wie durch Einsatz der Äquivalenzgesetze die Äquivalenz der Formeln $\alpha = y \wedge (x \to (y \leftrightarrow \neg x))$ und $\beta = y \wedge \neg x$ nachgewiesen werden kann.

$$\begin{aligned}
& y \wedge \big(x \to (y \leftrightarrow \neg x)\big) && \\
=\;& y \wedge \big(\neg x \vee (y \wedge \neg x) \vee (\neg y \wedge \neg\neg x)\big) && \text{Definition von } \to \text{ und } \leftrightarrow \\
\equiv\;& y \wedge \big(\neg x \vee (y \wedge \neg x) \vee (\neg y \wedge x)\big) && \text{Regel zur doppelten Verneinung} \\
\equiv\;& y \wedge \big(\neg x \vee (\neg y \wedge x)\big) && \text{Absorptionsgesetz} \\
\equiv\;& y \wedge (\neg x \vee \neg y) && \text{o.g. Folgerung aus den Distributivgesetzen} \\
\equiv\;& y \wedge \neg x &&
\end{aligned}$$

Folgerungen, Konsequenzen. Ist $\mathfrak{F}$ eine Menge von Formeln, so sagen wir, dass α eine logische Folgerung von $\mathfrak{F}$ ist, falls in jedem Kontext, in dem alle Aussagen, die durch Formeln in $\mathfrak{F}$ dargestellt sind, wahr sind, auch die durch α dargestellte Aussage wahr ist. Hierzu benötigen wir erst den Begriff eines Modells für Formelmengen.

Definition 4.6 (Modell, Erfüllbarkeit für Formelmenge). Sei $\mathfrak{F}$ eine Menge von aussagenlogischen Formeln. Eine Belegung I heißt *Modell für* $\mathfrak{F}$ oder auch *erfüllende Belegung für* $\mathfrak{F}$, falls I ein Modell für alle Formeln $\alpha \in \mathfrak{F}$ ist, d.h., falls $\alpha^I = 1$ für alle $\alpha \in \mathfrak{F}$. Formelmenge $\mathfrak{F}$ heißt *erfüllbar*, wenn es wenigstens ein Modell für $\mathfrak{F}$ gibt. Eine Formelmenge $\mathfrak{F}$ heißt *unerfüllbar*, wenn sie nicht erfüllbar und *gültig*, wenn jede Belegung erfüllend für $\mathfrak{F}$ ist. ■

Zunächst betrachten wir zwei einfache Beispiele. Die Formelmenge $\mathfrak{F}_1 = \{y \wedge \neg z, \neg x \vee y, x\}$ ist erfüllbar, da die Belegung I mit $x^I = y^I = 1$ und $z^I = 0$ ein gemeinsames Modell der drei Formeln in $\mathfrak{F}_1$ ist. Die Formelmenge $\mathfrak{F}_2 = \{\neg x \vee \neg y, x, y\}$ ist unerfüllbar, da die erste Formel $\neg x \vee \neg y$ in $\mathfrak{F}_2$ nur unter Belegungen I wahr ist mit $x^I = 0$ oder $y^I = 0$. Jede solche Belegung I ist aber nicht erfüllend für die restlichen Formeln x und y in $\mathfrak{F}_2$. Betrachten wir nun die unendliche Formelmenge

$$\mathfrak{F} \quad = \quad \{x_i \vee \neg x_{i+1} : i \in \mathbb{N}\},$$

der die unendlichen Menge $AP = \{x_i : i \in \mathbb{N}\}$ von Aussagensymbolen zugrundeliegt. Beispiele für Modelle von $\mathfrak{F}$ sind die Belegung, unter der alle Atome x_i wahr sind (d.h., $x_i^I = 1$ für alle $i \in \mathbb{N}$), oder die Belegungen I_n mit $x_i^{I_n} = 1$ für $i < n$, $x_i^{I_n} = 0$ für $i > n$ und $x_n^{I_n}$ beliebig.

Der Begriff der Erfüllbarkeit endlicher Formelmengen kann auf den Begriff der Erfüllbarkeit für Formeln zurückgeführt werden. Ist nämlich $\mathfrak{F}$ endlich, etwa $\mathfrak{F} = \{\alpha_1, \ldots, \alpha_n\}$, so ist $\mathfrak{F}$ genau dann erfüllbar, wenn die Formel $\alpha_1 \wedge \ldots \wedge \alpha_n$ erfüllbar ist.

Definition 4.7 (Logische Folgerung, Konsequenz, Relation $\models$). Eine aussagenlogische Formel α ist eine *logische Folgerung* oder *Konsequenz* von einer aussagenlogischen Formelmenge $\mathfrak{F}$, falls alle Modelle für $\mathfrak{F}$ zugleich Modelle für α sind. Hierfür schreiben wir $\mathfrak{F} \models \alpha$. ■

Also:

$$\begin{array}{lll} \mathfrak{F} \models \alpha & \text{gdw} & \text{jedes Modell für } \mathfrak{F} \text{ ist ein Modell für } \alpha \\ & \text{gdw} & \text{für alle Belegungen I gilt: ist I ein Modell für } \mathfrak{F}, \text{ dann ist } \alpha^I = 1 \end{array}$$

Beispielsweise gilt $\{x \vee y, x \vee \neg y\} \models x$, da $x^I = 1$ für jede gemeinsame erfüllende Belegung I für $x \vee y$ und $x \vee \neg y$ gelten muss. Ein anderes Beispiel ist

$$\{x, \neg x \vee y, \neg y \vee z\} \models y \wedge z.$$

Zum Nachweis dieser Aussage betrachten wir eine erfüllende Belegung I für die Formelmenge links. Dann gilt $x^I = 1$. Wegen $(\neg x \vee y)^I = 1$ folgt daher $y^I = 1$. Aus $(\neg y \vee z)^I$ erhalten wir somit $z^I = 1$. Dann aber ist $(y \wedge z)^I = 1$.

Sind α und β Formeln, so schreiben wir $\beta \models \alpha$ statt $\{\beta\} \models \alpha$. Offenbar gilt:

$$\begin{array}{lll} \beta \models \alpha & \text{gdw} & \text{jedes Modell für } \beta \text{ ist ein Modell für } \alpha \\ & \text{gdw} & \beta^I \leqslant \alpha^I \text{ für alle Belegungen I} \\ & \text{gdw} & \text{die Formel } \beta \to \alpha \text{ ist gültig} \end{array}$$

Z.B. gilt $x \wedge y \Vdash y$ da für jede erfüllende Belegung I für $\alpha = x \wedge y$ beide Atome x und y wahr sein müssen (d.h., $x^I = y^I = 1$). Insbesondere ist dann die atomare Formel y unter I wahr. Andere einfache Beispiele sind $x \Vdash x \vee y$ oder $x \wedge (x \to y) \Vdash y$. Beachte, dass unter jeder erfüllenden Belegung I für $\alpha = x \wedge (x \to y)$ das Atom x unter I mit 1 belegt ist und auch $(x \to y)^I = 1$ gelten muss. Dann aber ist $y^I = 1$. Also ist jede erfüllende Belegung für $\alpha = x \wedge (x \to y)$ zugleich erfüllend für die atomare Formel $\beta = y$.

Beispiel 4.8 (Affen-Bananen-Problem). In einem geschlossenen Raum befinden sich ein Affe und ein Stuhl. An der Decke hängt eine Banane. Die Frage ist, ob der Affe die Banane erreichen kann. Nun ja, Affen schaffen es immer irgendwie an die Bananen ranzukommen. Aber damit möchten wir uns nicht zufrieden geben und stellen die Erreichbarkeit der Banane als logische Folgerung einer Formelmenge dar, welche die Ausgangssituation und die Handlungsmöglichkeiten des Affen beschreibt.

(1) Wenn der Affe nicht auf dem Stuhl steht, dann kann er den Stuhl unter die Banane schieben.
(2) Der Affe kann auf den Stuhl klettern, sofern er nicht bereits auf dem Stuhl steht.
(3) Wenn der Affe auf dem Stuhl und der Stuhl unter der Banane steht, dann kann der Affe die Banane erreichen.
(4) Der Affe steht zunächst nicht auf dem Stuhl.

Wir verwenden folgende vier Aussagensymbole mit der jeweils angebenen intuitiven Bedeutung:

down	"der Affe steht nicht auf dem Stuhl"
up	"der Affe kann auf den Stuhl klettern"
below	"der Stuhl kann unter die Banane geschoben werden"
banana	"der Affe kann die Banane erreichen"

Die Formelmenge $\mathfrak{F} = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ besteht aus Formeln, die den Aussagen (1), (2), (3), (4) entsprechen:

$$\alpha_1 = \mathit{down} \to \mathit{below} \qquad \alpha_3 = \mathit{up} \wedge \mathit{below} \to \mathit{banana}$$
$$\alpha_2 = \mathit{down} \to \mathit{up} \qquad \alpha_4 = \mathit{down}$$

Die Frage, ob der Affe die Bananen erreichen kann, ist nun gleichwertig zur Frage, ob die atomare Formel *banana* logisch aus $\mathfrak{F}$ folgt. Hierzu betrachten wir ein Modell I für $\mathfrak{F}$. Dann gilt $\alpha_1^I = \alpha_2^I = \alpha_3^I = \alpha_4^I = 1$. Hieraus folgt:

$$\begin{aligned} \mathit{down}^I &= 1, && \text{da } \alpha_4^I = 1 \\ \mathit{up}^I &= 1, && \text{da } \alpha_2^I = 1 \\ \mathit{below}^I &= 1, && \text{da } \alpha_1^I = 1 \\ \mathit{banana}^I &= 1, && \text{da } \alpha_3^I = 1 \end{aligned}$$

Also gilt $\mathfrak{F} \Vdash \mathit{banana}$. ■

Beispiel 4.9 (Streichholzspiel). Ein Beispiel für eine Instanz des Folgerungsproblems ist die Frage nach einer Gewinnstrategie für folgendes Streichholzspiel. Gegeben sind n Streichhölzer, wobei n eine ganze Zahl $\geqslant 3$ ist. Spielerin A und Spieler B ziehen abwechselnd, beginnend mit Spielerin A. Jeder Spielzug besteht darin, entweder ein oder zwei Streichhölzer zu entfernen. Verlierer ist, wer das letzte Streichholz entfernt. Gefragt ist, ob Spielerin A eine Gewinnstrategie hat, d.h., ob es Zugmöglichkeiten für die initiale Konfiguration (mit n Streichhölzern) und alle folgenden Konfigurationen, in denen Spielerin A ziehen muss, gibt, so dass Spielerin A gewinnt, unabhängig davon, wie sich ihr Gegenspieler B verhält. Wir verwenden Aussagensymbole $x_1, \ldots, x_n$, wobei x_i die intuitive Bedeutung hat, dass Spielerin A eine Gewinnstrategie für die Konfiguration mit genau i Streichhölzern hat. Sei α_n folgende Formel:

$$\alpha_n \stackrel{\text{def}}{=} \neg x_1 \wedge x_2 \wedge \bigwedge_{3 \leqslant i \leqslant n} (x_i \leftrightarrow \neg x_{i-1} \vee \neg x_{i-2}).$$

Intuitiv steht das negative Literal $\neg x_1$ für die Tatsache, dass Spielerin A verliert, wenn nur noch ein Streichholz vorhanden ist und sie den nächsten Zug machen muss. Für zwei Streichhölzer hat sie jedoch eine Gewinnstrategie: sie entfernt ein Streichholz und versetzt Spieler B damit in die Lage, das letzte Streichholz entfernen zu müssen. Für $i \geqslant 3$ Streichhölzer liegt für Spielerin A genau dann eine Gewinnstrategie vor, wenn durch die Wegnahme von einem oder zwei Streichhölzern eine Konfiguration mit $i-1$ bzw. $i-2$ Streichhölzern erreicht werden kann, für welche es keine Gewinnstrategie gibt. Dies erklärt die Teilformel $x_i \leftrightarrow \neg x_{i-1} \vee \neg x_{i-2}$. Es gilt nun:

$$\left.\begin{array}{r}\text{Spielerin } A \text{ hat eine Gewinnstrategie} \\ \text{für die Konfiguration mit } n \text{ Streichhölzern}\end{array}\right\} \text{ gdw } \alpha_n \Vdash x_n$$

Durch Induktion nach n kann nun gezeigt werden, dass $\alpha_n \Vdash x_n$ genau dann, wenn $n \mod 3 \neq 1$, also wenn

$$n \in \{2,3,5,6,8,9,11,12,\ldots\} = \{3k{+}2, 3k{+}3 : k \in \mathbb{N}\}.$$

Die Gewinnstrategie sieht wie folgt aus. Liegen $3k{+}3$ Streichhölzer auf dem Tisch, so entferne zwei Streichhölzer. Damit ergibt sich die Konfiguration mit $3k{+}1$ Streichhölzern, in der Spieler B keine Gewinnstrategie hat. Liegen $3k{+}2$ Streichhölzer auf dem Tisch, so entferne ein Streichholz, was ebenfalls zur Konfiguration mit $3k{+}1$ Streichhölzern führt. ■

Ist $\mathfrak{F}$ eine Formelmenge und α eine Formel, so gilt:

$$\begin{array}{lll}\mathfrak{F} \Vdash \alpha & \text{gdw} & \text{jedes Modell für } \mathfrak{F} \text{ ist ein Modell für } \alpha \\ & \text{gdw} & \text{für jede Belegung I gilt: ist } \beta^I = 1 \text{ für alle } \beta \in \mathfrak{F}, \text{ so ist } (\neg\alpha)^I = 0 \\ & \text{gdw} & \mathfrak{F} \cup \{\neg\alpha\} \text{ hat kein Modell} \\ & \text{gdw} & \mathfrak{F} \cup \{\neg\alpha\} \text{ ist unerfüllbar}\end{array}$$

Damit erhalten wir folgendes Lemma, welches zeigt, dass logische Folgerbarkeit auf einen Unerfüllbarkeitstest zurückgeführt werden kann. Von dieser Beobachtung machen die Interpreter von Logikprogrammiersprachen Gebrauch. In diesem Zusammenhang steht $\mathfrak{F}$ für das Logikprogramm und α für die Frage. Statt eines expliziten Tests, ob α eine Konsequenz von $\mathfrak{F}$ ist, wird die Unerfüllbarkeit der Formelmenge geprüft, die aus dem Logikprogramm und der Negation der Frage besteht.

Lemma 4.10 (Folgerungsrelation und Unerfüllbarkeit). *Sei $\mathfrak{F}$ eine Formelmenge und α eine Formel. Dann gilt $\mathfrak{F} \models \alpha$ genau dann, wenn die Formelmenge $\mathfrak{F} \cup \{\neg\alpha\}$ unerfüllbar ist.*

Ist nun $\mathfrak{F}$ eine unerfüllbare Formelmenge, dann ist auch $\mathfrak{F} \cup \{\neg\alpha\}$ für jede Formel α unerfüllbar. Aus Lemma 4.10 folgt daher:

Ist $\mathfrak{F}$ unerfüllbar, so gilt $\mathfrak{F} \models \alpha$ für jede Formel α.

So gilt z.B. $\{x \vee \neg y, \neg x, y \wedge z\} \models \neg z$. Für einelementige Formelmengen, etwa $\mathfrak{F} = \{\beta\}$, kann die Aussage von Lemma 4.10 wie folgt umformuliert werden:

$$\beta \models \alpha \quad \text{gdw} \quad \beta \wedge \neg\alpha \text{ ist unerfüllbar}$$

Zusammenhänge zwischen Erfüllbarkeit, Folgerbarkeit, Äquivalenz und Gültigkeit. Es besteht ein enger Zusammenhang zwischen semantischer Äquivalenz, Gültigkeit, logischer Folgerbarkeit als binäre Relation über Formeln sowie Erfüllbarkeit (bzw. Unerfüllbarkeit). Jeder der vier Begriffe lässt sich auf die anderen drei Begriffe zurückführen. Z.B. gilt für die Konsequenzrelation $\models$ als binäre Relation über Formeln:

$$\begin{array}{lll} \alpha \models \beta & \text{gdw} & \alpha \to \beta \text{ ist gültig} \\ & \text{gdw} & \alpha \wedge \neg\beta \text{ ist unerfüllbar} \\ & \text{gdw} & \alpha \to \beta \equiv \mathit{true} \end{array}$$

Für endliche Formelmengen gilt:

$$\begin{array}{lll} \{\beta_1, \ldots, \beta_n\} \models \alpha & \text{gdw} & \beta_1 \wedge \ldots \wedge \beta_n \models \alpha \\ & \text{gdw} & \beta_1 \wedge \ldots \wedge \beta_n \to \alpha \text{ ist gültig} \\ & \text{gdw} & \beta_1 \wedge \ldots \wedge \beta_n \wedge \neg\alpha \text{ ist unerfüllbar} \end{array}$$

Die Darstellung der semantischen Äquivalenz durch die Konzepte Folgerbarkeit, Gültigkeit oder Unerfüllbarkeit ist wie folgt:

$$\begin{array}{lll} \alpha \equiv \beta & \text{gdw} & \alpha \models \beta \text{ und } \beta \models \alpha \\ & \text{gdw} & \alpha \leftrightarrow \beta \text{ ist gültig} \\ & \text{gdw} & \alpha \oplus \beta \text{ ist unerfüllbar} \end{array}$$

Dabei ist $\oplus$ der sogenannte Paritätsoperator, auch "exklusives oder" oder XOR genannt. Dieser ist durch

$$\alpha \oplus \beta \stackrel{\text{def}}{=} (\neg\alpha \wedge \beta) \vee (\alpha \wedge \neg\beta)$$

definiert. Offenbar ist $\alpha \oplus \beta$ äquivalent zu $\neg(\alpha \leftrightarrow \beta)$.

Diese Beobachtung ist für algorithmische Zwecke interessant, da z.B. jeder Algorithmus für das Erfüllbarkeitsproblem zugleich als Gültigkeitstest, Folgerbarkeitstest oder Äquivalenztest eingesetzt werden kann.

Das Model Checking Problem der Ausagenlogik. Wir werden im Verlauf der Vorlesung sehen, dass das Erfüllbarkeitsproblem, das Gültigkeitsproblem, das Äquivalenzproblem und das Folgerbarkeitsproblem der Aussagenlogik in exponentieller Zeit lösbar sind. Sie zählen zu "algorithmisch schwierigen" Problemen, für die es vermutlich keine effizienten Algorithmen gibt. Das Model Checking Problem ist jedoch gänzlich anders geartet, da dieses eine Formel α und eine Belegung I als gegeben voraussetzt und lediglich die Berechnung des Wahrheitswerts $\alpha^I \in \{0,1\}$ erfordert, statt eine Aussage über alle Belegungen zu machen. Im Falle der Aussagenlogik ist das Model Checking Problem sehr einfach (bei geeigneter Implementierung in linearer Zeit) zu lösen. Beispielsweise kann die Formel in Postfixnotation gebracht werden, die dann "von links nach rechts" ausgewertet werden kann. Dies entspricht im Wesentlichen der Erstellung des *Syntaxbaums* der Formel, dessen Knoten im Postorder-Prinzip generiert und in Bottom-Up-Manier bewertet werden. In Abbildung 31 ist der Syntaxbaum für die Formel

$$\alpha \;=\; (\neg x \wedge \neg(y \wedge z)) \wedge \neg(\neg y \wedge x)$$

skizziert. Die Beschriftungen in dieser Abbildung deuten an, wie die Wahrheitswerte der Teilformeln – dargestellt durch die Knoten des Syntaxbaums – von unten nach oben propagiert werden. Ausgangspunkt ist dabei die Belegung I mit $x^I = 0$ und $y^I = z^I = 1$, welche Wahrheitswerte für die Blätter (die mit je einem der Atome x, y, z beschriftet sind) vorgibt.

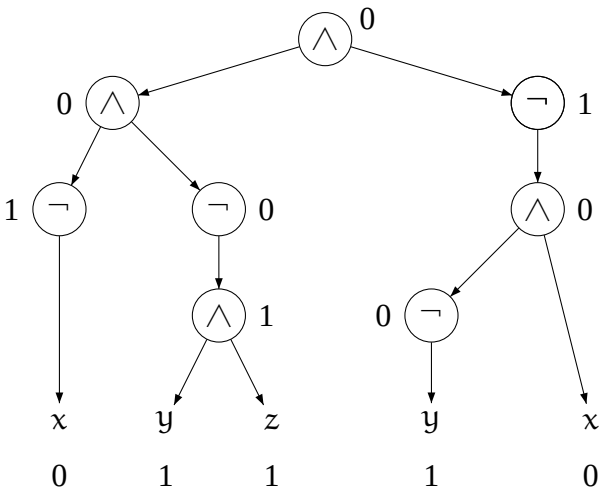


Abbildung 31: Bottom-Up-Bewertung des Syntaxbaums für aussagenlogische Formel

Länge von Formeln. Die Kostenfunktionen für Algorithmen, die als Eingabe eine aussagenlogische Formel haben, werden üblicherweise in Abhängigkeit der Formellänge erstellt, oftmals auch als zweistellige Funktion $T(n, m)$, wobei n für die Anzahl an Atomen in der Eingabeformel und m für die Länge der Eingabeformel steht.

Ist α eine aussagenlogische Formel, so bezeichnet $|\alpha|$ die *Länge* von α. Diese ist durch die Anzahl an in α vorkommenden Operatoren definiert. Für unsere Wahl der Syntax der Aussagenlogik mit den Basisoperatoren $\neg$ und $\wedge$ bezeichnet $|\alpha|$ die Anzahl an Vorkommen von $\neg$ und $\wedge$ in α. Dies entspricht folgenden Rekursionsformeln für die Länge von aussagenlogischen

Formeln. Wir setzen $|\mathit{true}| = |x| = 0$ für alle $x \in \mathit{AP}$ und

$$\begin{aligned} |\neg\alpha| &= |\alpha| + 1 \\ |\alpha \wedge \beta| &= |\alpha| + |\beta| + 1 \end{aligned}$$

Die abgeleitete Konstante *false* (die durch $\neg$*true* definiert wurde) hat also die Länge 1, während $\neg(x \wedge \neg y)$ die Länge 3 hat (2 Negationen, 1 Konjunktion). Man beachte, dass die Länge einer Formel des Typs $\bigwedge_{1 \leqslant i \leqslant n} \alpha_i$ gleich $n-1$ ist (und nicht etwa 1). Für den Disjunktionsoperator ergibt sich

$$|\alpha \vee \beta| = |\neg(\neg\alpha \wedge \neg\beta)| = |\alpha| + |\beta| + 4,$$

wobei die Konstante "4" für "3 Negationen plus 1 Konjunktion" steht. Damit ergibt sich für den Implikationsoperator:

$$|\alpha \to \beta| = |\neg\alpha \vee \beta| = (|\alpha| + 1) + |\beta| + 4 = |\alpha| + |\beta| + 5$$

Die so definierte Länge ist nützlich für induktive Argumente. Das Prinzip der strukturellen Induktion für aussagenlogische Formeln entspricht genau einer gewöhnlichen Induktion nach der Länge der Formeln. Für algorithmische Betrachtungen sind wir jedoch nur an der *asymptotischen Länge* von Formeln interessiert. Die konkreten Werte der additiven Konstanten (z.B. "1" für die Konjunktion, "4" für Disjunktion, "5" für die Implikation) spielen keine Rolle. Daher ist es für die asymptotische Länge irrelevant, ob man eine Darstellung von α mit den Basisoperatoren $\wedge$ und $\neg$ fordert und den beiden Basisoperatoren jeweils eine Längeneinheit zuordnet und für die abgeleiteten Operatoren $\vee$ und $\to$ 4 bzw. 5 Längeneinheiten veranschlagt werden, *oder* ob man Darstellungen mit den Operatoren $\wedge, \vee, \neg, \to$ zulässt und jedem dieser Operatoren nur eine Längeneinheit zuordnet. Ebenso ist es für die asymptotische Länge unerheblich, ob man der Konstanten *true* und den atomaren Formeln $x \in \mathit{AP}$ null Längeneinheiten oder eine Längeneinheit zuweist. Vorsicht ist jedoch mit anderen abgeleiteten Operatoren wie $\leftrightarrow$ oder $\oplus$ geboten. Wir betrachten hier exemplarisch den Paritätsoperator. Für diesen gilt:

$$|\alpha \oplus \beta| = |(\neg\alpha \wedge \beta) \vee (\alpha \wedge \neg\beta)| = 2|\alpha| + 2|\beta| + \text{const},$$

wobei "const" gemäß der oben angegebenen Definition der Formellänge für die Zahl 8 steht (jeweils 1 Kosteneinheit für die beiden Negationen und die beiden Konjunktionen und 4 Kosteneinheiten für die Disjunktion). Veranschlagt man eine Längeneinheit für jeden vorkommenden Operator, so ergibt sich der Wert 5 für "const" (2 Negationen, 2 Konjunktionen, 1 Disjunktion). In beiden Fällen gilt:

$$|\alpha \oplus \beta| > 2|\alpha| + 2|\beta|$$

Betrachte nun die Formeln $\alpha_n = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ für $n \geqslant 2$.[14]

Die Modelle für α_n genau diejenigen Belegungen I sind, für die die Anzahl an Indizes $i \in \{1, \ldots, n\}$ mit $x_i^I = 1$ ungerade ist.Diese Aussage kann durch Induktion nach n bewiesen werden.

- Der Induktionsanfang $n = 2$ ist klar, da die Modelle von $\alpha_2 = x_1 \oplus x_2$ genau diejenigen Belegungen I sind, die genau eines der Aussagensymbole x_1 oder x_2 mit 1 bewerten.

[14] Der Paritätsoperator ist assoziativ, was die klammerlose Schreibweise rechtfertigt. Formal können die Formeln α_n induktiv definiert werden, etwa $\alpha_2 = x_1 \oplus x_2$ und $\alpha_n = \alpha_{n-1} \oplus x_n$ für $n \geqslant 3$.

- Im Induktionsschritt $n-1 \Longrightarrow n$ setzen wir voraus, dass die Modelle von α_{n-1} genau diejenigen Belegungen I sind, für die die Anzahl an Indizes $i \in \{1,\ldots,n-1\}$ mit $x_i^I = 1$ ungerade ist. Sei nun I eine Belegung für $x_1,\ldots,x_n$. Dann ist I genau dann ein Modell für α_n, wenn

  $$\text{entweder } \alpha_{n-1}^I = 1 \;\&\; x_n^I = 0 \quad \text{oder } \alpha_{n-1}^I = 0 \;\&\; x_n^I = 1 \qquad (*)$$

  Die Induktionsvoraussetzung liefert in beiden Fällen, dass Aussage (*) äquivalent ist zur Forderung, dass $\left|\left\{i \in \{1,\ldots,n\} : x_i^I = 1\right\}\right|$ ungerade ist.

Für die Formellänge der α_n's gilt:

$$|\alpha_n| \geqslant 2|\alpha_{n-1}| \geqslant 4|\alpha_{n-2}| \geqslant \ldots \geqslant 2^{n-2}|\alpha_2| \geqslant 2^{n-2}$$

Die Länge von α_n wächst also exponentiell, obwohl mit der Schreibweise $x_1 \oplus x_2 \oplus \ldots \oplus x_n$ nur $2n-1$ Zeichen verwendet werden und man daher lineare Formellänge vermuten könnte.

4.1.2 Normalformen

Für jede Art von Logik spielen Normalformen eine wichtige Rolle. Diese machen gewisse syntaktische Einschränkungen und dienen oftmals als Ausgangspunkt für Algorithmen oder können auch Beweise vereinfachen. Wir betrachten hier die positive Normalform, die nur eine eingeschränkte Nutzung der Negation erlaubt sowie zweistufige Normalformen (konjunktive und disjunktive Normalform). Im Kontext dieser Normalformen sind die Konjunktionen und Disjunktionen "gleichberechtigt", d.h., wir betrachten beide $\wedge$ und $\vee$ als Basisoperatoren.

Literal. Ein *Literal* ist eine Formel der Art x oder $\neg x$, wobei x ein Atom ist. Literale der Form x heißen *positiv*; Literale der Form $\neg x$ werden *negativ* genannt. Die Literale x und $\neg x$ sind *komplementär* zueinander. Ist L ein Literal, so schreiben wir $\overline{L}$ für die Komplementierung des Literals:

$$\overline{L} \stackrel{\text{def}}{=} \begin{cases} \neg x & : \text{ falls } L = x \text{ ein positives Literal ist} \\ x & : \text{ falls } L = \neg x \text{ ein negatives Literal ist.} \end{cases}$$

Offenbar gilt $\overline{L} \equiv \neg L$.

Positive Normalform (PNF)

In der positiven Normalform (kurz PNF), manchmal auch *Negationsnormalform* genannt, ist nur eine eingeschränkte Nutzung des Negationsoperators zulässig ist. Formeln in PNF verwenden den Negationsoperator nur auf der Ebene der Literale. Um dieselbe Ausdrucksstärke wie die Aussagenlogik zu garantieren, wird neben dem Konjunktionsoperator $\wedge$ der hierzu duale Operator $\vee$ (der Disjunktionsoperator) benötigt. D.h., aussagenlogische Formeln in positiver Normalform (PNF) über der Aussagenmenge *AP* werden durch Konjunktionen und Disjunktionen aus den Konstanten *true* und *false*, sowie den Literalen x und $\neg x$ für $x \in AP$ gebildet. Die abstrakte Syntax von PNF-Formeln ist wie folgt:

$$\alpha \quad ::= \quad \mathit{true} \;\mid\; \mathit{false} \;\mid\; x \;\mid\; \neg x \;\mid\; \alpha_1 \wedge \alpha_2 \;\mid\; \alpha_1 \vee \alpha_2 \qquad \text{(Formeln in PNF)}$$

Beispielsweise ist $z \wedge (\neg x \vee (\neg y \wedge z)) \wedge (x \vee \neg z)$ in PNF, während $\neg(x \vee \neg y)$ oder $\neg x \vee \neg\neg y$ nicht in PNF sind. Man beachte, dass abgeleitete Operatoren, wie die Implikation, Parität oder Äquivalenz, nicht ohne weiteres in PNF-Formeln eingesetzt werden können. Z.B. ist $(x \wedge z) \to y$ nicht in PNF, obwohl auf den ersten Blick nur positive Literale "sichtbar" sind. Hierzu muss man sich klarmachen, dass $(x \wedge z) \to y$ lediglich eine abkürzende Schreibweise für $\neg(x \wedge z) \vee y$ ist; eine Formel, die offenbar nicht in PNF ist. Für die asymptotische Länge einer PNF-Formel reicht es in α die Anzahl an Vorkommen der Operatoren $\wedge$ und $\vee$ zu zählen.

Jede aussagenlogische Formel α kann in eine äquivalente PNF-Formel derselben asymptotischen Länge transformiert werden. Hierzu sind lediglich die Dualität zwischen Konjunktion und Disjunktion (die De Morgan'schen Regeln), die Dualität der Konstanten *true* und *false* sowie das Gesetz der doppelten Verneinung auszunutzen, um alle Negationen "nach innen" zu ziehen. Geht man von einer aussagenlogischen Formel α aus, die nur die in der primären Syntax angegebenen Operatoren $\neg$ und $\wedge$ (aber keine abgeleiteten Operatoren) verwendet, so erhält man durch sukzessives Anwenden der nachstehenden Transformationsregeln ("von außen nach innen") eine zu α äquivalente PNF-Formel:

$$\begin{array}{lcl} \neg \mathit{true} & \rightsquigarrow & \mathit{false} \\ \neg\neg\gamma & \rightsquigarrow & \gamma \\ \neg(\gamma_1 \wedge \gamma_2) & \rightsquigarrow & \neg\gamma_1 \vee \neg\gamma_2 \end{array}$$

Die Hinzunahme des Disjunktionsoperators als Basisoperator ist problemlos, da dann lediglich $\neg(\gamma_1 \vee \gamma_2) \rightsquigarrow \neg\gamma_1 \wedge \neg\gamma_2$ als zusätzliche Transformationsregel hinzugefügt werden muss. Werden obige Regeln solange wie möglich eingesetzt, so ist klar, dass der Negationsoperator nur noch auf Ebene der Literale vorkommt und die resultierende Formel β somit in PNF ist. Wendet man die obigen Transformationsregeln stets "von außen nach innen" an, also nur auf solche Teilformeln $\neg\gamma$, die nicht Teilformel einer anderen Teilformel $\neg\delta$ sind, so kann die Anzahl an Transformationsschritten durch die Anzahl an Teilformeln von α nach oben abgeschätzt werden. (Genauer gilt: die Anzahl an vorgenommenen Transformationsschritten ist höchstens $|\alpha|$ plus Anzahl an Vorkommen der Konstanten *true* in α.) Mit der Anzahl an Transformationsschritten ist auch die Länge von β in $\mathcal{O}(|\alpha|)$. Dies ist wie folgt einsichtig. Wie oben erwähnt reicht es für die asymptotische Länge einer PNF-Formel die Anzahl an Disjunktionen und Konjunktionen zu zählen. Die ersten beiden Transformationsregeln entfernen eine bzw. zwei Negationen, die dritte Regel ersetzt eine Negation und eine Konjunktion durch zwei Negationen und eine Disjunktion. In allen drei Fällen bleibt die Anzahl an Konjunktionen und Disjunktionen unverändert. Also ist die asymptotische Länge von β gleich der Anzahl an Konjunktionen plus Disjunktionen von α. Diese ist durch $|\alpha|$ beschränkt. Somit gilt $|\beta| = \mathcal{O}(|\alpha|)$.

Die Erstellung einer äquivalenten PNF-Formel durch sukzessives Anwenden der Transformationsregeln illustrieren wir am Beispiel der aussagenlogischen Formel

$$\alpha \;=\; \neg\big((x \vee \neg z) \wedge \neg(x \wedge \neg y)\big).$$

Wir wenden sukzessive die De Morganschen Regeln und die Regel zur doppelten Verneinung an, um die Negationen vollständig nach innen zu ziehen und redundante Negationen aufzulösen:

$$\begin{aligned}
\alpha &= \neg\big((x \vee \neg z) \wedge \neg(x \wedge \neg y)\big) \\
&\equiv \neg(x \vee \neg z) \vee \neg\neg(x \wedge \neg y) \\
&\equiv \neg(x \vee \neg z) \vee (x \wedge \neg y) \\
&\equiv (\neg x \wedge \neg\neg z) \vee (x \wedge \neg y) \\
&\equiv (\neg x \wedge z) \vee (x \wedge \neg y)
\end{aligned}$$

Mit dem angegebenen Verfahren und den obigen Überlegungen zur Länge der konstruierten PNF-Formel erhalten wir folgenden Satz:

Satz 4.11 (Universalität der PNF). *Zu jeder aussagenlogischen Formel α gibt es eine PNF-Formel β mit $\alpha \equiv \beta$ und $|\beta| = \mathcal{O}(|\alpha|)$.*

Zweistufige Normalformen (KNF, DNF)

Traditionell spielen zweistufige Normalformen, in denen die Formeln aus Konjunktionen über Disjunktionen (konjunktive Normalform), oder umgekehrt, Disjunktionen über Konjunktionen (disjunktive Normalform), aufgebaut sind und auf unterster Ebene nur Literale zugelassen sind, eine wichtige Rolle in der Informatik. In der technischen Informatik sind sie für die Logiksynthese zentral, da dort zweistufige Normalformen (insbesondere die disjunktive Normalform) als Ausgangspunkt für den Entwurf von PLA (programmable logic arrays) dienen. In der Algorithmik und Komplexitätstheorie kommt der konjunktiven Normalform eine herausragende Bedeutung zu.

Einstufige Formeln (Klauseln, Monome). Zunächst klären wir einige Grundbegriffe. Beginnen wir zunächst mit einstufigen Formeln, welche Literale $L_1, ..., L_k$ mit Junktoren desselben Typs (entweder Konjunktion oder Disjunktion) verknüpfen. Ein *Monom* oder *Konjunktionsterm* ist eine Formel des Typs $L_1 \wedge L_2 \wedge \ldots \wedge L_k$, wobei $L_1, \ldots, L_k$ Literale sind. Formeln des Typs $L_1 \vee L_2 \vee \ldots \vee L_k$ werden *Klauseln* oder auch *Disjunktionsterme* genannt. Ein Monom heißt *widersprüchlich*, falls es zueinander komplementäre Literale x und $\neg x$ enthält. Widersprüchliche Monome sind offenbar unerfüllbar. Klauseln mit zueinander komplementären Literalen sind dagegen tautologisch:

$$\ldots \vee x \vee \neg x \vee \ldots \equiv \textit{true}.$$

Zur Bezeichnung von Literalen verwenden wir üblicherweise den Buchstaben L, während wir für Klauseln griechische Kleinbuchstaben wie κ ("kappa"), λ ("lambda"), σ ("sigma") oder τ ("tau") benutzen, und μ ("mu") oder ν ("nu") für Monome; jeweils mit oder ohne Index.

Für eine Klausel κ ist $\kappa = \bigvee_{i \in \varnothing} \ldots = \textit{false}$ die Klausel bestehend aus 0 Literalen. In diesem Fall spricht man auch von der *leeren Klausel*. Sie wird auch mit dem Symbol $\sqcup$ bezeichnet. Bei einer Klausel, die aus genau einem Literal besteht, spricht man auch von einer *Einheitsklausel*. Z.B. ist $\kappa = \neg x \vee y \vee z$ eine Klausel mit drei Literalen und $\kappa' = \neg x$ eine Einheitsklausel. Eine *negative Klausel* bezeichnet eine Klausel, die keine positiven Literale enthält, aber mindestens ein negatives Literal; also eine Klausel der Form $\neg x_1 \vee \ldots \vee \neg x_k$ mit $k \geqslant 1$. Entsprechend wird eine nicht-leere Klausel *positiv* genannt, wenn sie keine negativen Literale enthält; also wenn sie die Gestalt $x_1 \vee \ldots \vee x_k$ mit $k \geqslant 1$ hat.

Konjunktive Normalform (KNF). Eine Formel in konjunktiver Normalform, kurz KNF-Formel genannt, ist eine Konjunktion von Klauseln; also von der Form

$$\alpha = \bigwedge_{1 \leqslant i \leqslant m} \underbrace{(L_{i,1} \vee L_{i,2} \vee \ldots \vee L_{i,k_i})}_{\text{i-te Klausel}},$$

wobei die $L_{i,j}$ Literale sind und $m \geqslant 0$, $k_1, \ldots, k_i \geqslant 0$. Z.B. ist $\alpha = (\neg x \vee y) \wedge (y \vee z)$ eine zu $\beta = y \vee (\neg x \wedge z)$ äquivalente Formel in KNF. Offenbar ist eine KNF-Formel α genau dann unter einer gegebenen Belegung I wahr, wenn es in jeder Klausel von α wenigstens ein Literal gibt, das unter I wahr ist. Ist also α wie oben, dann gilt $\alpha^I = 1$ genau dann, wenn es zu jedem $i \in \{1, \ldots, m\}$ einen Index $j \in \{1, \ldots, k_i\}$ mit $L_{i,j}^I = 1$ gibt. Im Sonderfall $m = 0$ ist α eine KNF-Formel bestehend aus 0 Klauseln. In diesem Fall ist

$$\alpha = \bigwedge_{i \in \varnothing} \ldots = \mathit{true}.$$

Falls eine der Klauseln von α leer ist, dann hat α die Form $\ldots \wedge \sqcup \wedge \ldots = \ldots \wedge \mathit{false} \wedge \ldots$, und ist somit unerfüllbar. Die asymptotische Länge einer KNF-Formel α ist durch die totale Anzahl an Literalen in α gegeben. Liegt z.B. die KNF-Formel

$$\alpha = \bigwedge_{1 \leqslant i \leqslant m} \bigvee_{1 \leqslant j \leqslant k_i} L_{i,j}$$

vor, so ist die asymptotische Länge von $|\alpha|$ gleich $k_1 + \ldots + k_m$.

Disjunktive Normalform (DNF). Formeln in disjunktiver Normalform, auch DNF-Formeln oder Polynome genannt, sind Disjunktionen von 0 oder mehreren Monomen. Beispielsweise ist $(x \wedge \neg y \wedge \neg z) \vee (\neg y) \vee (x \wedge z)$ eine Formel in DNF. Die allgemeine Gestalt von DNF-Formeln ist also

$$\alpha = \bigvee_{1 \leqslant i \leqslant m} \bigwedge_{1 \leqslant j \leqslant k_i} L_{i,j},$$

wobei $L_{i,j}$'s Literale sind und $m \geqslant 0$, $k_1, \ldots, k_m \geqslant 0$. Auch hier sind die Sonderfälle $m = 0$ oder $k_i = 0$ möglich. Für den Sonderfall $m = 0$ ist α eine Disjunktion von 0 Monomen, also $\alpha = \bigvee_{i \in \varnothing} \ldots = \mathit{false}$, Ist $k_i = 0$, so ist das i-te Monom leer (d.h., eine Konjunktion von 0 Literalen). Das i-te Monom steht somit für die Konstante $\bigwedge_{j \in \varnothing} \ldots = \mathit{true}$. Wie im Falle von KNF-Formeln ist die asymptotische Länge einer DNF-Formel α durch die totale Anzahl an Literalen in α gegeben.

Die beiden zweistufigen Normalformen sind dual zueinander, da das Vertauschen von Konjunktionen und Disjunktionen und das Ersetzen jedes Literals L durch das komplementäre Literal $\overline{L}$ jede KNF-Formel α in eine DNF-Formel β mit $\beta \equiv \neg\alpha$ überführt. Ist also

$$\alpha = \bigvee_{1 \leqslant i \leqslant m} \bigwedge_{1 \leqslant j \leqslant k_i} L_{i,j}$$

eine DNF-Formel, so ist

$$\beta = \bigwedge_{1 \leqslant i \leqslant m} \bigvee_{1 \leqslant j \leqslant k_i} \overline{L}_{i,j}$$

eine zu $\neg\alpha$ äquivalente KNF-Formel.

Satz 4.12 (Universalität von KNF und DNF). *Zu jeder aussagenlogischen Formel α gibt es eine äquivalente KNF-Formel und eine äquivalente DNF-Formel.*

Für den Beweis von Satz 4.12 geben wir zwei alternative Verfahren zur Konstruktion äquivalenter KNF- bzw. DNF-Formeln an.

KNF/DNF via Wertetafel. Das einfachste Verfahren zur Erstellung einer DNF oder KNF geht von einer Wertetafel aus und verknüpft die sogenannten *Minterme* disjunktiv (DNF) bzw. *Maxterme* konjunktiv (KNF).

Definition 4.13 (Min-/Maxterme). Ist $AP = \{x_1, \ldots, x_n\}$, so wird jedes Monom der Form $L_1 \wedge L_2 \wedge \ldots \wedge L_n$ mit $L_i \in \{x_i, \neg x_i\}$ Minterm und jede Klausel der Form $L_1 \vee L_2 \vee \ldots \vee L_n$ mit $L_i \in \{x_i, \neg x_i\}$ Maxterm genannt. ■

Min- und Maxterme stehen in Eins-zu-Eins-Beziehung zu den Belegungen für *AP*. Hierzu wird jedem Minterm $\mu = L_1 \wedge L_2 \wedge \ldots \wedge L_n$ genau diejenige Belegung I zugeordnet, unter der alle Literale L_i von μ wahr sind, also $x_i^I = 1$, falls $L_i = x_i$ und $x_i^I = 0$, falls $L_i = \neg x_i$. Dual hierzu wird jedem Maxterm $\kappa = L_1 \vee L_2 \vee \ldots \vee L_n$ diejenige Belegung I zugeordnet, unter der alle Literale in κ falsch sind, also $x_i^I = 1$, falls $L_i = \neg x_i$ und $x_i^I = 0$, falls $L_i = x_i$. Ein Minterm für eine aussagenlogische Formel α ist ein Minterm, für den die zugeordnete Belegung ein Modell für α ist. Offenbar ist α dann äquivalent zu derjenigen DNF-Formel, die man durch Disjunktion aller Minterme für α erhält. Ein Maxterm für α ist ein Maxterm, für den die induzierte Belegung kein Modell für α ist. Eine zu α äquivalente KNF-Formel erhält man durch Konjunktion aller Maxterme für α. Durch Inspektion der Wertetafeln für eine aussagenlogische Formel α erhält man die Min- bzw. Maxterme und kann daraus wie eben beschrieben eine äquivalente DNF- bzw. KNF-Formel generieren. Die hier vorgestellte Methode zur Generierung einer DNF- bzw. KNF-Formel zeigt auch, dass es zu jeder Formel eine äquivalente Formel in DNF oder KNF gibt. Jedoch führt diese stets zu einer exponentiellen Laufzeit. Ferner sind die resultierenden Normalformen oftmals sehr lang.

Wir betrachten nun als Beispiel die aussagenlogische Formel

$$\alpha \;=\; \neg\big((x \vee \neg z) \wedge \neg(x \wedge \neg y)\big),$$

welche wir schon im vorigen Abschnitt über die PNF verwendet haben. In der folgenden Tabelle ist für jede mögliche Belegung I der Wahrheitswert von α aufgelistet. Auf der rechten Seite sind die korrespondierenden Maxterme angegeben:

x^I	y^I	z^I	α^I	
0	0	0	0	$\longleftarrow$ Maxterm $x \vee y \vee z$
0	0	1	1	
0	1	0	0	$\longleftarrow$ Maxterm $x \vee \neg y \vee z$
0	1	1	1	
1	0	0	1	
1	0	1	1	
1	1	0	0	$\longleftarrow$ Maxterm $\neg x \vee \neg y \vee z$
1	1	1	0	$\longleftarrow$ Maxterm $\neg x \vee \neg y \vee \neg z$

Verknüpft man die Maxterme konjunktiv, so erhält man folgende zu α äquivalente KNF-Formel:

$$(x \vee y \vee z) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$$

In analoger Weise erhalten wir eine zu α äquivalente DNF-Formel. Hierzu benötigen wir die Minterme von α, die ebenfalls aus der Wertetafel abgelesen werden können:

x^I	y^I	z^I	α^I	
0	0	0	0	
0	0	1	1	$\longleftarrow$ Minterm $\neg x \wedge \neg y \wedge z$
0	1	0	0	
0	1	1	1	$\longleftarrow$ Minterm $\neg x \wedge y \wedge z$
1	0	0	1	$\longleftarrow$ Minterm $x \wedge \neg y \wedge \neg z$
1	0	1	1	$\longleftarrow$ Minterm $x \wedge \neg y \wedge z$
1	1	0	0	
1	1	1	0	

Eine zu α äquivalente DNF-Formel erhält man durch Disjunktion der Minterme:

$$(\neg x \wedge \neg y \wedge z) \vee (\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge \neg z) \vee (x \wedge \neg y \wedge z)$$

KNF/DNF via PNF und Äquivalenzgesetze. Ein alternatives Verfahren, zu einer aussagenlogischen Formel eine äquivalente Formel in KNF oder DNF zu generieren, besteht darin, zunächst eine äquivalente PNF-Formel zu erstellen, die dann mit Hilfe der Distributivgesetze für die Disjunktion und Konjunktion in die gewünschte zweistufige Normalform gebracht wird. Sei α eine beliebige aussagenlogische Formel, dann kann diese in eine äquivalente Formel α' in PNF transformiert werden, indem man vorkommende Negationen "nach innen zieht" (wie oben im Abschnitt über PNF beschrieben). Durch sukzessives Anwenden der Distributivgesetze kann α' in eine äquivalente aussagenlogische KNF- oder DNF-Formel umgewandelt werden. Wir betrachten hier nur exemplarisch die Konstruktion einer äquivalenten KNF-Formel, bei der folgende Transformationsregeln auf die PNF-Formel α' angewendet werden, um die Disjunktionen nach innen zu ziehen:

$$\begin{aligned} \beta \vee (\gamma_1 \wedge \gamma_2) &\quad \rightsquigarrow \quad (\beta \vee \gamma_1) \wedge (\beta \vee \gamma_2) \\ (\gamma_1 \wedge \gamma_2) \vee \beta &\quad \rightsquigarrow \quad (\gamma_1 \vee \beta) \wedge (\gamma_2 \vee \beta) \end{aligned}$$

Die Transformation zur DNF erfolgt analog mit entsprechend dualen Distributivgesetzen.

Als Beispiel nehmen wir die aussagenlogische Formel α, die schon im letzten Abschnitt betrachtet wurde:

$$\alpha = \neg\big((x \vee \neg z) \wedge \neg(x \wedge \neg y)\big)$$

Eine zu α äquivalente PNF-Formel α' erhält man wie im Abschnitt über die PNF mit dem gleichen Beispiel für α beschrieben:

$$\alpha' = (\neg x \wedge z) \vee (x \wedge \neg y)$$

Die Transformation zu einer äquivalenten KNF-Formel mittels der oben aufgelisteten Distributivgesetze erfolgt dann schrittweise wie folgt:

$$\begin{aligned}\alpha \;\equiv\; \alpha' \;&=\; (\neg x \wedge z) \vee (x \wedge \neg y)\\ &\equiv\; \big((\neg x \wedge z) \vee x\big) \wedge \big((\neg x \wedge z) \vee \neg y\big)\\ &\equiv\; \big((\neg x \vee x) \wedge (z \vee x)\big) \wedge \big((\neg x \vee \neg y) \wedge (z \vee \neg y)\big)\\ &\equiv\; (z \vee x) \wedge (\neg x \vee \neg y) \wedge (z \vee \neg y)\end{aligned}$$

Offenbar ist die resultierende Formel in der Tat eine KNF-Formel. Das Verfahren beruhend auf den Äquivalenzregeln ist effizienter als das Verfahren via Wertetabellen, welches wir im letzten Abschnitt betrachtet haben (vergleiche z.B. die Länge der entstandenen KNF-Formel). Jedoch ist ein exponentielles Blowup möglich und für manche Formeln sogar unvermeidbar, wie der folgende Satz belegt.

Satz 4.14 (Exponentielles Blowup durch den Übergang zu KNF und DNF).

(a) Es gibt DNF-Formeln α_n der Länge $|\alpha_n| = \mathcal{O}(n)$, so dass jede zu α_n äquivalente KNF-Formel mindestens 2^n Klauseln hat.

(b) Es gibt KNF-Formeln γ_n der Länge $|\gamma_n| = \mathcal{O}(n)$, so dass jede zu γ_n äquivalente DNF-Formel mindestens 2^n Monome hat.

Beweis. Zunächst beweisen wir Aussage (a). Die DNF-Formeln

$$\alpha_n \;\stackrel{\text{def}}{=}\; (x_1 \wedge y_1) \vee \ldots \vee (x_n \wedge y_n)$$

haben lineare Länge, da die Anzahl an Konjunktionen und Disjunktionen in α_n gleich $2n-1$ ist und somit $|\alpha_n| = \mathcal{O}(n)$. Äquivalente KNF-Formeln ergeben sich durch Anwenden des Distributivgesetzes. Man erhält folgende KNF-Formel mit genau 2^n Klauseln:

$$\alpha'_n \;\stackrel{\text{def}}{=}\; \bigwedge_{\substack{(z_1,\ldots,z_n)\\ z_j \in \{x_j, y_j\} \text{ für } 1 \leqslant j \leqslant n}} (z_1 \vee \ldots \vee z_n).$$

Beispielsweise gilt für $n = 2$:

$$\alpha'_2 \;=\; (x_1 \vee x_2) \wedge (x_1 \vee y_2) \wedge (y_1 \vee x_2) \wedge (y_1 \vee y_2).$$

Formal kann die Äquivalenz von α_n und α'_n entweder durch Argumentation mit den Distributivgesetzen nachgewiesen werden oder auch explizit durch den Nachweis, dass α_n und α'_n unter allen Belegungen denselben Wahrheitswert haben:

- Ist I eine erfüllende Belegung für α_n, so gibt es ein $j \in \{1,\ldots,n\}$, so dass $x_j^I = y_j^I = 1$. Da jede Klausel von α'_n entweder x_j oder y_j enthält, ist I zugleich erfüllend für α'_n.
- Ist I eine Belegung mit $\alpha_n^I = 0$, so gilt für jedes $j \in \{1,\ldots,n\}$, dass wenigstens eines der Atome x_j oder y_j mit 0 belegt ist. Sei nun

  $$z_j \;\stackrel{\text{def}}{=}\; \begin{cases} x_j & : \text{ falls } x_j^I = 0\\ y_j & : \text{ sonst}\end{cases}$$

  für $1 \leqslant j \leqslant n$. Dann ist $z_j^I = 0$ für $1 \leqslant j \leqslant n$ und somit $z_1 \vee \ldots \vee z_n$ eine Klausel von α'_n, welche unter I den Wahrheitswert 0 hat. Also ist I auch für α'_n nicht erfüllend.

Wir zeigen nun, dass jede zu α_n äquivalente KNF-Formel 2^n oder mehr Klauseln hat. Hierzu betrachten wir eine *kürzeste* zu α_n äquivalente KNF-Formel

$$\beta_n = \lambda_1 \wedge \ldots \wedge \lambda_k.$$

und zeigen, dass $k \geqslant 2^n$. Es ist klar, dass in β_n höchstens die Atome $x_1, \ldots, x_n, y_1, \ldots, y_n$ vorkommen und dass kein Literal mehrfach in einer der Klauseln von β_n vorkommen kann. Andernfalls könnte β_n verkürzt werden.

Behauptung 1. β_n enthält keine negativen Literale.

Intuitiv ist diese Aussage klar, da sonst β_n verkürzt werden kann. Eine formale Beweismöglichkeit besteht darin, anzunehmen, dass β_n ein negatives Literal $\neg z$ mit $z \in \{x_1, \ldots, x_n, y_1, \ldots, y_n\}$ enthält. Man betrachtet dann diejenige Formel, welche aus β_n entsteht, indem ein (oder alle) Vorkommen von $\neg z$ in β_n gestrichen werden und zeigt dann, dass die resultierende Formel immer noch zu α_n äquivalent ist. Dies widerspricht dann der Annahme, dass β_n eine *kürzeste* zu α_n äquivalente KNF-Formel ist.

Aus der ersten Behauptung folgt, dass alle Klauseln λ_i von β_n Disjunktionen von Atomen in $\{x_1, \ldots, x_n, y_1, \ldots, y_n\}$ sind.

Behauptung 2. Für jede Klausel λ_i von β_n und jedes $j \in \{1, \ldots, n\}$ gilt: in λ_i kommt wenigstens eines der Literale x_j oder y_j vor.

Beweis von Behauptung 2. Wir nehmen an, dass weder x_j noch y_j in λ_i vorkommt, und führen diese Annahme zu einem Widerspruch. Hierzu betrachten wir die Belegung I, welche genau die in λ_i vorkommenden Atome mit 0 belegt, alle anderen mit 1. Da x_j und y_j nicht in λ_i vorkommen, gilt:

$$x_j^I = y_j^I = 1$$

Offenbar gilt $\alpha_n^I = 1$. Wegen $\alpha_n \equiv \alpha_n' \equiv \beta_n$ gilt daher:

$$\beta_n^I = \alpha_n^I = 1$$

Andererseits ist $\lambda_i^I = 0$, da λ_i aufgrund der ersten Behauptung nur aus positiven Literalen besteht und diese nach Wahl von I mit 0 belegt sind. Hieraus folgt $\beta_n^I = 0$. Widerspruch. ⌋

In der dritten Behauptung zeigen wir nun, dass tatsächlich jede der 2^n Klauseln $\kappa = z_1 \vee \ldots \vee z_n$ mit $z_j \in \{x_j, y_j\}$ für $1 \leqslant j \leqslant n$ der oben angegebenen zu α_n äquivalenten KNF-Formel α_n' in β_n vorkommen muss.

Behauptung 3. Jede der Klauseln κ von α_n' kommt (modulo Permutation der Literale) in $\beta_n = \lambda_1 \wedge \ldots \wedge \lambda_k$ vor, d.h., es gibt einen Index $i \in \{1, \ldots, k\}$, so dass $\lambda_i \equiv \kappa$.

Beweis von Behauptung 3. Sei κ eine Klausel von α_n'. Sei I diejenige Belegung, unter der genau solche Atome, welche in κ vorkommen, zu 0 evaluieren. D.h.,

$$z^I = \begin{cases} 0 & : \text{ falls } z \text{ in } \kappa \text{ vorkommt} \\ 1 & : \text{ sonst} \end{cases}$$

Ist z.B. $\kappa = x_1 \vee \ldots \vee x_m \vee y_{m+1} \vee \ldots \vee y_n$, so betrachten wir die Belegung I mit

$$x_1^I = \ldots x_m^I = y_{m+1}^I = \ldots = y_n^I = 0 \quad \text{und} \quad y_1^I = \ldots y_m^I = x_{m+1}^I = \ldots = x_n^I = 1.$$

Offenbar ist α'_n unter I falsch, d.h., $(\alpha'_n)^I = 0$. Da α'_n zu α_n und β_n äquivalent ist, erhalten wir:

$$\beta_n^I \;=\; \alpha_n^I \;=\; 0$$

Also gibt es eine Klausel λ_i von β_n, so dass $\lambda_i^I = 0$. Da aber alle Atome, die nicht in κ vorkommen, mit 1 belegt sind, muss λ_i aus Atomen bestehen, die auch in κ vorkommen. Da aufgrund der zweiten Behauptung jede Klausel von β_n aus mindestens n Literale besteht (nämlich je eines der Literale x_j oder y_j enthält) folgt $\lambda_i \equiv \kappa$. ⌋

Aus der dritten Behauptung folgt, dass β_n aus 2^n Klauseln besteht. Damit ist Aussage (a) bewiesen. Aussage (b) ergibt sich nun aus der Dualität von KNF und DNF. Die KNF-Formeln

$$\gamma_n \stackrel{\text{def}}{=} (x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge \ldots \wedge (x_n \vee y_n),$$

haben lineare Länge, aber jede äquivalente DNF-Formel hat 2^n oder mehr Monome. □

In den folgenden Abschnitten befassen wir uns mit dem *Erfüllbarkeitsproblem* der Aussagenlogik. Dieses ist auch unter der Abkürzung SAT für die englische Bezeichnung satisfiability problem bekannt. Dieses zählt zu den algorithmisch schwierigen Problemen, für die es vermutlich keine effizienten Lösungsverfahren gibt. (Mehr hierzu in dem Modul "Theoretische Informatik und Logik".)

Zunächst stellen wir fest, dass es Formeltypen gibt, für das Erfüllbarkeitsproblem sehr einfach zu lösen ist. Hierzu zählen DNF-Formeln. Eine DNF-Formel ist nämlich genau dann erfüllbar, wenn wenigstens eines ihrer Monome erfüllbar ist. Und Monome, also Konjunktionen von Literalen, sind genau dann erfüllbar, wenn sie keine zueinander komplementären Literale enthalten. Mit dieser Beobachtung ergibt sich ein DNF-SAT-Beweiser (damit ist ein Verfahren für den Erfüllbarkeitstest für DNF-Formeln gemeint) wie folgt. Sei α die Eingabeformel, also eine DNF-Formel, für welche die Erfüllbarkeit zu prüfen ist. In einem vorbereitenden Schritt können zunächst alle widersprüchlichen Monome (d.h. Monome mit zueinander komplementären Literalen) gestrichen werden. Es entsteht eine DNF-Formel, die genau dann unerfüllbar ist, wenn sie kein Monom enthält, also wenn $\alpha = \mathit{false}$. Damit ergibt sich ein Verfahren für DNF-SAT, welches die Laufzeit $\mathcal{O}(|\alpha|)$ hat. Dennoch ist die Erstellung einer äquivalenten DNF kein Allheilmittel, da kürzeste äquivalente DNF-Formeln exponentiell länger als die ursprüngliche Formel sein können.

Eine weitere Klasse, für die ein einfacher und effizienter Erfüllbarkeitstest möglich ist, ist die Klasse der Hornformeln, die in dem folgenden Abschnitt betrachtet wird.

4.2 Hornformeln

Eine wichtige Formelklasse bilden die sogenannten Hornformeln, denen in der Logikprogrammierung eine zentrale Rolle zukommt, und für die ein sehr einfacher und effizienter Erfüllbarkeitstest möglich ist. Hornformeln sind spezielle KNF-Formeln, nämlich:

Definition 4.15 (Hornformel, Hornklausel). Eine *Hornklausel* ist eine Klausel, in der höchstens ein positives Literal vorkommt (aber beliebig viele negative Literale). Eine KNF-Formel heißt *Hornformel*, falls alle ihre Klauseln Hornklauseln sind. ■

Die KNF-Formeln $(x \vee \neg y) \wedge \neg z$, $(\neg x \vee \neg y) \wedge \neg z$, $(\neg x \vee y) \wedge z$ sind Hornformeln. Auch $x \wedge y$ ist eine Hornformel, da sie aus zwei Einheitsklauseln besteht. Die KNF-Formel $x \vee y$ (die nur aus einer Klausel besteht) ist dagegen keine Hornformel, da sie eine Klausel mit zwei positiven Literalen enthält. Es gibt drei Arten von Hornformeln:

- negative Klauseln, d.h. Klauseln, welche nur aus negativen Literalen bestehen. Im Kontext der Logikprogrammierung werden sie meist *Zielklauseln* genannt. Ein Sonderfall hiervon ist die leere Klausel.
- positive Einheitsklauseln, auch *Fakten* genannt.
- Hornformeln mit (genau) einem positiven und wenigstens einem negativen Literal. Diese werden auch *Regeln* genannt.

Anlehnend an die Syntax von Logikprogrammiersprachen wie PROLOG oder DATALOG sind für Hornformeln Implikationsschreibweisen "*Prämisse* $\rightarrow$ *Folgerung*" gebräuchlich. So steht $x_1 \wedge \ldots \wedge x_m \rightarrow y$ für die Hornklausel $\neg x_1 \vee \ldots \vee \neg x_n \vee y$. Die Prämisse $x_1 \wedge \ldots \wedge x_m$ wird der *Rumpf* und das positive Literal y der *Kopf* der Regel genannt. Positive Einheitsklauseln (Fakten) werden oftmals in der Form *true* $\rightarrow x$ geschrieben, Zielklauseln in der Form $x_1 \wedge \ldots \wedge x_m \rightarrow$ *false*. Beachte, dass

$$\textit{true} \rightarrow x = \textit{false} \vee x \equiv x \quad \text{und} \quad x_1 \wedge \ldots \wedge x_m \rightarrow \textit{false} \equiv \neg x_1 \vee \ldots \vee \neg x_m.$$

Beispiel 4.16 (Affen-Bananen-Problem). Die vier Formeln $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ der Formelmenge $\mathfrak{F}$ aus Beispiel 4.8 zum Affen-Bananen-Problem (siehe Seite 120) sind Hornklauseln. Die letzte Klausel $\alpha_4 = \textit{banana}$ ist ein Faktum. Die anderen drei Formeln sind Regeln. In den ersten beiden Regeln

$$\begin{aligned} \alpha_1 &= \textit{down} \rightarrow \textit{below} \equiv \neg \textit{down} \vee \textit{below} \\ \alpha_2 &= \textit{down} \rightarrow \textit{up} \equiv \neg \textit{down} \vee \textit{up} \end{aligned}$$

besteht der Rumpf jeweils nur aus einem Literal (nämlich *down*). Die Atome *up* bzw. *below* sind die Köpfe der beiden Regeln. Die dritte Regel

$$\alpha_3 = \textit{up} \wedge \textit{below} \rightarrow \textit{banana} \equiv \neg \textit{up} \vee \neg \textit{below} \vee \textit{banana}$$

hat zwei Literale im Rumpf und die Banane im Kopf. Wie bereits in Beispiel 4.8 nachgewiesen wurde, gilt $\alpha \models \textit{banana}$, wobei α diejenige Hornformel ist, die sich durch Konjunktion der vier Hornklauseln ergibt, also $\alpha = \alpha_1 \wedge \alpha_2 \wedge \alpha_3 \wedge \alpha_4$. Äquivalent zur Folgerbarkeit von *banana* aus α ist die Unerfüllbarkeit der Hornformel

$$\alpha \wedge \neg \textit{banana} \equiv \alpha \wedge (\textit{banana} \rightarrow \textit{false}),$$

die sich durch die Hinzunahme der Negation der Frage als Zielklausel ergibt. ■

Beispiel 4.17 ("Wissensbasiertes System" für ein Chemielabor). Gegeben ist eine Datenbank, in der Informationen über die vorhandenen chemischen Stoffe und durchführbaren chemischen Reaktionen

$$x_1 + x_2 + \ldots + x_k \rightarrow y_1 + y_2 + \ldots + y_\ell$$

eines Chemielabors verwaltet werden.[15] Gefragt ist, ob ein gewisser chemischer Stoff z (über 0 oder mehr chemische Reaktionen) erzeugt werden kann. Jeden Stoff fassen wir als Aussagensymbol auf. Das Vorhandensein von vorhandenen Stoffe und durchführbaren Reaktionen wird durch eine Hornformel α dargestellt. Die vorhandenen Stoffe stellen wir durch Fakten dar. Jede der durchführbaren Reaktionen $x_1+x_2+\ldots+x_k \;\rightarrow\; y_1+y_2+\ldots+y_\ell$ zerlegen wir in ℓ Regeln

$$\begin{array}{rcl} x_1\wedge x_2\wedge\ldots\wedge x_k & \rightarrow & y_1 \\ x_1\wedge x_2\wedge\ldots\wedge x_k & \rightarrow & y_2 \\ \vdots & & \vdots \\ x_1\wedge x_2\wedge\ldots\wedge x_k & \rightarrow & y_\ell \end{array}$$

welche die Herstellbarkeit von jedem der Stoffe y_i unter der Voraussetzung, dass die Stoffe $x_1, x_2, \ldots, x_k$ verfügbar sind, beschreiben. Die Frage, ob ein Stoff z in dem Labor erzeugt werden kann, ist gleichbedeutend mit der Frage, ob $\alpha \Vdash z$ gilt (Begründung siehe unten). Zur Beantwortung der Frage ist also die Unerfüllbarkeit der Formel $\alpha\wedge\neg z$ zu prüfen. Wir illustrieren diese Aussage an einem Beispiel.

$$\begin{array}{rcl} MgO+H_2 & \rightarrow & Mg+H_2O \\ C+O_2 & \rightarrow & CO_2 \\ H_2O+CO_2 & \rightarrow & H_2CO_3 \end{array}$$

seien die durchführbaren chemischen Reaktionen, wobei die Grundstoffe MgO, H_2, O_2 und C verfügbar sind. Wir fragen, ob $z = H_2CO_3$ (Kohlensäure) herstellbar ist. Hierzu ist die Unerfüllbarkeit folgender Hornformel nachzuweisen:

$$\begin{array}{ll} MgO \wedge H_2 \wedge O_2 \wedge C & \wedge \\ (H_2\wedge MgO \rightarrow Mg) & \wedge \\ (H_2\wedge MgO \rightarrow H_2O) & \wedge \\ (C\wedge O_2 \rightarrow CO_2) & \wedge \\ (H_2O\wedge CO_2 \rightarrow H_2CO_3) & \wedge \\ (H_2CO_3 \rightarrow \mathit{false}). & \end{array}$$

Wir erläutern nun, warum die Herstellbarkeit von Stoff z gleichbedeutend mit $\alpha \Vdash z$ ist. Zunächst nehmen wir an, dass $\alpha \Vdash z$. Wir betrachten nun die Belegung I, welche genau denjenigen Aussagensymbolen (Stoffen) den Wahrheitswert 1 zuweist, die in dem vorliegenden Labor über 0 oder mehrere Schritte herstellbar sind. Dann gilt $\alpha^I = 1$ und somit $z^I = 1$. Also ist Stoff z herstellbar. Wir setzen nun umgekehrt die Herstellbarkeit von z voraus und zeigen, dass z eine logische Folgerung von α ist. Hierzu zeigen wir durch Induktion nach m, dass jeder Stoff, der in dem betreffenden Labor durch m Reaktionen hergestellt werden kann, eine logische Folgerung aus α ist. Im Induktionsanfang $m = 0$ betrachten wir Stoffe, die durch 0 Reaktionen hergestellt werden können. Dies sind initial vorhandene Stoffe, die als Fakten in α vorkommen. Diese Stoffe sind offenbar logische Folgerungen aus α. Nun zum Induktionsschritt $m-1 \Longrightarrow m$, wobei

[15] Die Durchführbarkeit einer Reaktion ist so zu verstehen, dass das Labor über die Ausstattung verfügt, um die betreffende Reaktion durchzuführen, sofern die Stoffe der linken Seite vorliegen (initial vorhanden sind oder durch vorangegangene Reaktionen erzeugt wurden).

$m \geqslant 1$. Sei z ein Stoff, der mittels m Reaktionen in dem betreffenden Labor hergestellt werden kann. Ferner sei

$$x_1 + \ldots + x_k \;\rightarrow\; \ldots + z + \ldots$$

die letzte dieser m Reaktionen. Dann ist jeder der Stoffe x_j mit höchstens $m-1$ Reaktionen in dem Labor herstellbar. Nach Induktionsvoraussetzung gilt $\alpha \Vdash x_j$ für $j = 1, \ldots, k$. Da $x_1 \wedge \ldots \wedge x_k \rightarrow z$ eine Klausel von α ist, gilt $\alpha \Vdash z$. ■

Markierungsalgorithmus für HORN-SAT. Für Hornformeln ist das Erfüllbarkeitsproblem HORN-SAT (als Kurzform für die englische Bezeichnung "satisfiability problem for Horn formulas") recht leicht zu beantworten. Dieses hat als Eingabe eine Hornformel α. Die Aufgabe ist zu entscheiden, ob α erfüllbar ist. Hierzu kann ein *Markierungsalgorithmus* eingesetzt werden, der sukzessive alle Aussagensymbole markiert, die unter einer erfüllenden Belegung (falls es eine solche gibt) zwingend mit 1 bewertet sein müssen. In der Initialisierungsphase werden alle Aussagensymbole, die als Fakten in α vorkommen, markiert. Der Algorithmus markiert nun solange wie möglich weitere Aussagensymbole, die als Kopf y in einer Regel $x_1 \wedge \ldots \wedge x_m \rightarrow y$ erscheinen, wobei $x_1, \ldots, x_m$ bereits markiert sind, oder bricht mit der Antwort "nein" ab, sofern es eine Zielklausel $x_1 \wedge \ldots \wedge x_m \rightarrow \mathit{false}$ gibt, für die $x_1, \ldots, x_m$ markiert sind. Siehe Algorithmus 5.

Algorithmus 5 Markierungsalgorithmus für HORN-SAT

(* Eingabe: Hornformel α *)
(* Aufgabe: prüft, ob α erfüllbar ist *)

Markiere alle Aussagensymbole x, die in α als Fakten vorkommen.
REPEAT
  IF es gibt eine Zielklausel $x_1 \wedge \ldots \wedge x_m \rightarrow \mathit{false}$ in α, so dass $x_1, \ldots, x_m$ markiert sind
  THEN gib "nein, α ist unerfüllbar" aus und halte an
  FI
  IF es gibt eine Regel $x_1 \wedge \ldots \wedge x_m \rightarrow y$ in α, so dass $x_1, \ldots, x_m$ markiert sind und y nicht markiert ist,
  THEN markiere y
  FI
UNTIL keine Veränderungen in der letzten Iteration
gib "ja, α ist erfüllbar" aus

Die Terminierung des Verfahrens ergibt sich aus der Beobachtung, dass die Anzahl an Markierungsschritten, die in der Initialisierungsphase oder Markierungsschritte innerhalb der Schleife durchgeführt werden, durch die Anzahl an Aussagensymbolen, die in α vorkommen, nach oben beschränkt ist. Wir zeigen nun die Korrektheit der ausgegebenen Antworten.

Satz 4.18 (Korrektheit des Markierungsalgorithmus). *Ist α eine Hornformel, so terminiert der beschriebene Markierungsalgorithmus für HORN-SAT (Algorithmus 5) mit der korrekten Antwort "nein" im Falle der Unerfüllbarkeit von α und "ja" im Falle der Erfüllbarkeit von α.*

Ist α erfüllbar, so ist diejenige Belegung I*, die genau den markierten Aussagensymbolen den Wahrheitswert* 1 *zuordnet, das kleinste Modell für α. D.h., es gilt $\alpha^I = 1$ und für jedes Modell* J *für α gilt $x^I \leqslant x^J$ für alle Aussagensymbole x, die in α vorkommen.*

Beweis. Zunächst beweisen wir folgende Hilfsaussage, die als Schleifeninvariante angesehen werden kann:

(*) Ist J ein Modell für α, so ist $y^J = 1$ für alle markierten Aussagensymbole y.

Der Nachweis von (*) erfolgt durch Induktion nach der Anzahl an Markierungsschritten in der Initialisierungsphase oder innerhalb der REPEAT-Schleife. Der Induktionsanfang bezieht sich auf die Initialisierungsphase. In dieser werden genau die Aussagensymbole x, die als Einheitsklauseln in α vorkommen, markiert. Unter jedem Modell J für $\alpha = \ldots \wedge x \wedge \ldots$ müssen diese den Wahrheitswert 1 haben. Im Induktionsschritt nehmen wir nun an, dass y innerhalb der REPEAT-Schleife markiert wurde. Dann gibt es eine Regel $\kappa = x_1 \wedge \ldots \wedge x_m \to y$ von α, so dass die Aussagensymbole $x_1, \ldots, x_m$ vor y markiert wurden. Die Induktionsvoraussetzung liefert $x_1^J = \ldots = x_m^J = 1$ für jede erfüllende Belegung J für α. Wegen $\alpha^J = 1$ gilt $\kappa^J = 1$ und somit auch $y^J = 1$. Damit ist (*) bewiesen.

1. Wir setzen zunächst voraus, dass Algorithmus 5 "nein" ausgibt und zeigen, dass α unerfüllbar ist. Die Antwort "nein" ist nur möglich, wenn es eine Zielklausel $x_1 \wedge \ldots \wedge x_n \to$ *false* von α gibt, so dass $x_1, \ldots, x_m$ markiert sind. Wir nehmen nun an, dass α eine erfüllende Belegung J besitzt und führen diese Annahme zu einem Widerspruch. Aufgrund von Aussage (*) müsste dann $x_1^J = \ldots = x_m^J = 1$ gelten. Dann aber wäre

$$(x_1 \wedge \ldots \wedge x_m \to \mathit{false})^J \;=\; (\neg x_1 \vee \ldots \vee \neg x_m)^J \;=\; 0$$

   und somit $\alpha^J = 0$. Widerspruch zur Annahme, dass J erfüllend für α ist.

2. Wir nehmen nun an, dass der Algorithmus mit der Antwort "ja" terminiert und zeigen, dass die Belegung I, welche durch

$$x^I \stackrel{\text{def}}{=} \begin{cases} 1 & : \text{ falls } x \text{ nach der Terminierung markiert ist} \\ 0 & : \text{ sonst} \end{cases}$$

   definiert ist, erfüllend für α ist. Sei κ eine Klausel von α. Wir zeigen, dass $\kappa^I = 1$ ist.

   1. Fall: κ ist ein Faktum (positive Einheitsklausel); etwa $\kappa = x$. Dann wird x in der Initialisierungsphase markiert. Daher gilt $\kappa^I = x^I = 1$.

   2. Fall: κ ist eine Regel der Form $x_1 \wedge \ldots \wedge x_m \to y$.

   - Falls $x_1^I = \ldots = x_m^I = 1$, so ist $y^I = 1$ und somit $\kappa^I = 1$.
   - Falls $x_i^I = 0$ für ein $i \in \{1, \ldots, m\}$, so ist $\kappa^I = 1$.

   3. Fall: κ ist eine Zielklausel der Form $x_1 \wedge \ldots \wedge x_m \to$ *false*. Der Fall $x_1^I = \ldots = x_m^I = 1$ ist nicht möglich, da Algorithmus 5 sonst mit der Antwort "nein" terminiert hätte. Also gilt $x_i^I = 0$ für ein $i \in \{1, \ldots, m\}$ und somit $\kappa^I = 1$.

Da alle Klauseln von κ unter I den Wahrheitswert 1 haben, ist $\alpha^I = 1$. Aus Hilfsaussage (*) folgt, dass $x^I \leqslant x^J$ für jede erfüllende Belegung J für α. Also ist I das kleinste Modell für α.

□

Da Algorithmus 5 für eine Hornformel mit n Aussagensymbolen höchstens n Iterationen (Markierungsschritte) ausführt, ist klar, dass eine Implementierung möglich ist, deren Laufzeit polynomiell in $|\alpha|$ beschränkt ist.

Beispiel 4.19 (Markierungsalgorithmus für HORN-SAT). Wir betrachten nochmals das Chemielabor aus Beispiel 4.17 auf Seite 134. Wie bereits erwähnt, ist die Herstellbarkeit von Kohlensäure H_2CO_3 gleichbedeutend mit der Unerfüllbarkeit nachstehender Hornformel:

$$\begin{aligned}\beta \;=\; & MgO \wedge H_2 \wedge O_2 \wedge C \wedge \\ & (H_2 \wedge MgO \rightarrow Mg) \wedge (H_2 \wedge MgO \rightarrow H_2O) \wedge \\ & (C \wedge O_2 \rightarrow CO_2) \wedge (H_2O \wedge CO_2 \rightarrow H_2CO_3) \wedge \\ & (H_2CO_3 \rightarrow \textit{false})\end{aligned}$$

Algorithmus 5 angewandt auf β beginnt mit der Markierung der vier Fakten MgO, H_2, O_2 und C (Initialisierungsphase). Dann sind die Rümpfe der ersten drei Regeln markiert, so dass nacheinander deren Köpfe (also die Aussagensymbole Mg, H_2O, CO_2) markiert werden (Markierungsschritte innerhalb der REPEAT-Schleife). Da nun auch der Rumpf der letzten Regel markiert ist, wird deren Kopf (das Aussagensymbol H_2CO_3) markiert. Damit ist der Rumpf der Zielklausel $H_2CO_3 \rightarrow$ *false* markiert, so dass Algorithmus mit der Antwort "nein" terminiert. Hornformel β ist also unerfüllbar und somit Stoff H_2CO_3 herstellbar.

Wir fügen nun zusätzlich das Faktum $NaCl$ und die Regeln ein, die für die Durchführbarkeit der Reaktion

$$NaCl + H_2SO_4 \rightarrow HCl + NaHSO_4$$

stehen, die aus $NaCl$ (Natriumchlorid) und H_2SO_4 (Schwefelsäure) die beiden Stoffe HCl (Chlorwasserstoff) und $NaHSO_4$ (Natriumbisulfat) erzeugen. Die Herstellbarkeit des Stoffes $NaHSO_4$ ist nun gleichbedeutend mit der Unerfüllbarkeit folgender Hornformel:

$$\begin{aligned}\gamma \;=\; & MgO \wedge H_2 \wedge O_2 \wedge C \wedge NaCl \wedge \\ & (H_2 \wedge MgO \rightarrow Mg) \wedge (H_2 \wedge MgO \rightarrow H_2O) \wedge \\ & (C \wedge O_2 \rightarrow CO_2) \wedge (H_2O \wedge CO_2 \rightarrow H_2CO_3) \wedge \\ & (NaCl \wedge H_2SO_4 \rightarrow HCl) \wedge (NaCl \wedge H_2SO_4 \rightarrow NaHSO_4) \wedge \\ & (NaHSO4 \rightarrow \textit{false})\end{aligned}$$

Der Markierungsalgorithmus für HORN-SAT angewandt auf γ verhält sich zunächst wie für β. In der Initialisierungsphase werden die fünf Fakten MgO, H_2, O_2, C und $NaCl$ markiert. Dann werden der Reihe nach die Stoffe Mg, H_2O, CO_2 und H_2CO_3 markiert. Danach hält Algorithmus 5 mit der Antwort "ja" an, da keine weiteren Markierungen und kein Abbruch möglich sind. Also ist γ erfüllbar. Unter den genannten Bedingungen ist also $NaHSO_4$ nicht herstellbar ist. Dieser Sachverhalt ist nicht verwunderlich, da der Stoff H_2SO_4 "vergessen" wurde. ■

Wir beenden diesen Abschnitt mit einigen einfachen Aussagen über Hornformeln. Zunächst stellen wir fest, dass Hornformeln *ohne* Zielklauseln niemals widersprüchlich sein können. Solche Hornformeln nennt man auch *definit*. Erst durch die Hinzunahme von Zielklauseln können unerfüllbare Formeln entstehen. Die Begründung hierfür ist sehr einfach. In definiten Hornformeln hat jede Klausel genau ein positives Literal. Betrachtet man nun die Interpretation, die alle Aussagensymbole mit dem Wahrheitswert 1 belegt, so erhält man offenbar eine erfüllende Belegung. Eine andere mögliche Begründung orientiert sich am Markierungsalgorithmus, der offenbar niemals die Antwort "nein" ausgeben kann, wenn keine Zielklauseln vorhanden sind. Die Regeln und Fakten eines Logikprogramms, die zu einer Hornformel ohne Zielklauseln zusammengefasst werden können, stellen also stets eine erfüllbare Formel dar.

Umgekehrt ist auch jede Hornformel, die keine Fakten und keine leeren Klauseln enthält, erfüllbar. Die Begründung hierfür ist, dass die Belegung I, die allen Aussagensymbolen den Wert 0 zuweist, alle Klauseln wahr macht, da jede Klausel nun mindestens ein negatives Literal enthält. Auch hier ist eine alternative Argumentation mit dem Markierungsalgorithmus möglich: Wenn keine Fakten vorliegen, so wird kein Aussagensymbol markiert. Die Antwort "nein" könnte deshalb nur dann ausgegeben werden, wenn eine Zielklausel mit leerem Rumpf (also eine leere Klausel) vorliegt. Wir erhalten die Aussage des folgenden Lemmas:

Lemma 4.20 (Hinreichende Kriterien für die Erfüllbarkeit von Hornformeln).

(a) Jede definite Hornformel (d.h., Hornformel ohne Zielklauseln) ist erfüllbar.

(b) Jede Hornformel ohne Fakten und leere Klauseln ist erfüllbar.

Bemerkung 4.21 (Hornformeln sind nicht universell). Es gibt aussagenlogische Formeln, zu denen es keine äquivalenten Hornformeln gibt. Ein derartiges Beispiel ist die Formel $x \vee y$. Wir nehmen an, dass es eine zu $x \vee y$ äquivalente Hornformel α gibt. Dann haben $x \vee y$ und α dieselben Modelle. Da α eine erfüllbare Hornformel ist, haben α und somit auch $x \vee y$ ein *kleinstes* Modell. Dies steht jedoch im Widerspruch zur Beobachtung, dass $x \vee y$ kein kleinstes Modell besitzt, da $[x = 0, y = 0]$ kein Modell für $x \vee y$ ist und die "nächst größeren" Modelle $[x = 1, y = 0]$ und $[x = 0, y = 1]$ unvergleichbar sind. ■

Eine weitere Beobachtung ist, dass aus Hornformeln ohne Zielklauseln keine negativen Literale logisch gefolgert werden können. Diese Beobachtung ist für die Logikprogrammierung sehr bedeutsam, da sie einen vorsichtigen Umgang mit Fragen des Typs "gilt Aussage … nicht?" erfordert.

Lemma 4.22 (Hornformeln ohne Zielklauseln lassen keine negativen Schlüsse zu!). *Ist α eine Hornformel, in der jede Klausel genau ein positives Literal hat, und x ein Aussagensymbol, so gilt $\alpha \not\models \neg x$.*

Beweis. Angenommen $\alpha \models \neg x$. Dann ist $\alpha \wedge \neg\neg x \equiv \alpha \wedge x$ unerfüllbar. Andererseits ist $\alpha \wedge x$ eine Hornformel, die keine Zielklauseln enthält, und somit erfüllbar (Teil (a) von Lemma 4.20). Widerspruch. □

4.3 SAT-Beweiser

Neben dem Spezialfall von DNF- und Hornformeln gibt es weitere Klassen von aussagenlogischen Formeln, für die das Erfüllbarkeitsproblem algorithmisch effizient lösbar ist. Werden keine syntaktischen Einschränkungen für die Eingabeformel gemacht, so zählt das Erfüllbarkeitsproblem der Aussagenlogik (SAT für "satisfiability problem") zu den algorithmisch schwierigen Problemen. Wir stellen hier nun einige Verfahren vor, wie man die Erfüllbarkeit für beliebige aussagenlogische Formeln und KNF-Formeln algorithmisch testen kann. Solche Verfahren werden auch SAT-Beweiser bzw. KNF-SAT-Beweiser genannt.

Naiver SAT-Beweiser. Im Folgenden sei α eine aussagenlogische Formel, deren Erfüllbarkeit zu prüfen ist, und x_1, ..., x_n seien die in α vorkommenden Atome. Der naive Algorithmus für SAT berechnet die Wahrheitswerte α^I unter allen Belegungen I für $\{x_1,\ldots,x_n\}$, und gibt "ja" zurück, genau dann, wenn $\alpha^I = 1$ für wenigstens eine der Belegungen. Diese "alle-Belegungen-ausprobieren"-Methode kann dadurch realisiert werden, indem man den *Entscheidungsbaum* für α mit einer Tiefensuche (Preorder) analysiert. Der Entscheidungsbaum ist lediglich eine anschauliche Darstellung aller Belegungen für $x_1,\ldots,x_n$ und deren Wahrheitswerten für α. Der Entscheidungsbaum für α ist ein vollständiger Binärbaum der Höhe n. Die beiden ausgehenden Kanten eines inneren Knotens der Tiefe i, wobei $0 \leqslant i < n$, stehen für die Belegungen $x_{i+1} = 0$ bzw. $x_{i+1} = 1$. Jedem inneren Knoten v der Tiefe i kann diejenige (Teil-)Belegung $[x_1 = b_1,\ldots x_i = b_i]$ für die ersten i Atome $x_1,\ldots,x_i$ zugeordnet werden, die sich aus den Beschriftungen der Kanten des Pfads von der Wurzel zu Knoten v ergibt. Jedem Pfad von der Wurzel zu einem Blatt (Knoten der Tiefe n) entspricht eine Belegung I für $x_1,\ldots,x_n$. Das betreffende Blatt ist mit dem Wahrheitswert von α unter dieser Belegung I beschriftet. Ein Beispiel für einen Entscheidungsbaum ist in Abbildung 32 angegeben, wobei stets die Kante von einem inneren Knoten der Tiefe i zu dem linken Sohn für den Fall $x_{i+1} = 0$ steht und die Kante zu dem rechten Sohn für $x_{i+1} = 1$.

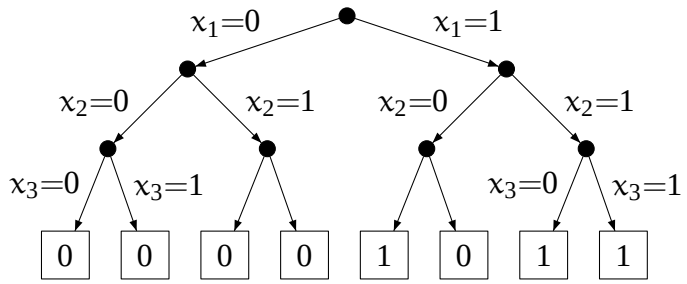


Abbildung 32: Entscheidungsbaum für $x_1 \wedge (x_2 \vee \neg x_3)$

Im Kontext des Erfüllbarkeitsproblems sind wir lediglich daran interessiert, ob der Entscheidungsbaum ein mit dem Wahrheitswert 1 beschriftetes Blatt enthält. Daher können wir die Wahrheitswerte der Blätter bis hoch zur Wurzel propagieren, indem wir jeden inneren Knoten v mit 1 beschriften, sofern wenigstens einer seiner beiden Söhne mit 1 beschriftet ist. Andernfalls beschriften wir Knoten v mit 0. An der Beschriftung der Knoten mit den Werten 1 oder 0 kann man nun ablesen, ob der betreffende Teilbaum ein mit 1 beschriftetes Blatt enthält oder nicht.

Die gegebene Formel α ist also genau dann erfüllbar, wenn die Wurzel des Entscheidungsbaums für α mit 1 beschriftet ist.

Algorithmus 6 SAT(α, i)

IF $i = n$ **THEN**
  berechne den Wahrheitswert α^I von α unter $I = [x_1 = b_1, \ldots, x_n = b_n]$
  IF $\alpha^I = 1$ **THEN** return *true* **ELSE** return *false* **FI**
ELSE
  $b_{i+1} := 0$
  IF SAT$(\alpha, i+1)$ **THEN** return *true* **FI** ;
  $b_{i+1} := 1$
  IF SAT$(\alpha, i+1)$ **THEN** return *true* **ELSE** return *false* **FI**
FI

Die Erzeugung und Bewertung des Entscheidungsbaums nach dem Preorder-Prinzip kann leicht mit einem *Backtracking*-Algorithmus implementiert werden, der mit n globalen Booleschen Variablen $b_1, \ldots, b_n$ arbeitet, welche für die Belegungen der Atome $x_1, \ldots, x_n$ stehen. Das Verfahren ist in Algorithmus 6 skizziert, welches initial mit SAT$(\alpha, 0)$ aufzurufen ist. Die Terminierung des Verfahrens ist klar, da die Rekursionstiefe durch n (Anzahl an Atomen) beschränkt ist. Falls SAT$(\alpha, 0)$ mit der Rückgabe *true* terminiert, so stehen die aktuellen Werte $b_1, \ldots, b_n$ für eine erfüllende Belegung für α. In diesem Fall ist α also erfüllbar. Andernfalls, also falls SAT$(\alpha, 0)$ mit der Rückgabe *false* terminiert, so enthält der Entscheidungsbaum kein mit 1 bewertetes Blatt. Die Eingabeformel α ist daher unerfüllbar. Damit ist die partielle Korrektheit des Verfahrens klar. Da für eine unerfüllbare Eingabeformel α alle 2^n Belegungen zu betrachten sind, ist die worst-case Laufzeit exponentiell. Aufgrund der exponentiellen Laufzeit ist nur für Formeln mit hinreichend kleiner Anzahl n an Atomen in zumutbarer Zeit eine Antwort des naiven SAT-Beweisers zu erwarten. Ein kleines Rechenbeispiel illustriert diese Aussage. Nimmt man an, dass α unerfüllbar ist und $n = 80$ Atome hat und dass in 1 Millisekunde die Wahrheitswerte von α unter 1.000 Belegungen berechnet werden können, so ist die benötigte Rechenzeit $2^{80}/1.000.000$ Sekunden > 2^{34} Jahre.

Wie bereits erwähnt zählt SAT zu den algorithmisch schwierigen Problemen, für die es vermutlich keine effiziente Lösung gibt. Dennoch gibt es eine Reihe von ausgefeilten Algorithmen, die zwar ebenfalls exponentielle worst-case Laufzeit haben, jedoch für viele praxisrelevante Formeln recht schnell ein Ergebnis liefern. Im Folgenden stellen wir Algorithmen vor, welche als SAT-Beweiser für KNF-Formeln dienen. Der erste Algorithmus geht von einer beliebigen KNF-Formel aus und ist somit durch eine vorangeschaltete Transformation, welche eine gegebene aussagenlogische Formel in eine äquivalente KNF-Formel überführt, für alle aussagenlogischen Formeln einsetzbar. Mit einem Trick, den wir später erläutern, kann das etwaige exponentielle Längenwachstum durch die Erstellung äquivalenter KNF-Formeln verhindert werden.

KNF-SAT-Beweiser

Wir beginnen mit einer Reihe von einfachen Beobachtungen, die im Kontext von KNF-SAT-Beweisern nützlich sind. Wir gehen dabei stets von einer KNF-Formel $\alpha = \kappa_1 \wedge \ldots \wedge \kappa_m$ aus, wobei $m \geqslant 0$ und $\kappa_1, \ldots, \kappa_m$ Klauseln sind. Zunächst halten wir zwei triviale Sonderfälle fest, in denen die Unerfüllbarkeit bzw. Erfüllbarkeit von α sofort entschieden werden kann:

1. Sonderfall: Keine Klausel. Ist α eine KNF-Formel mit 0 Klauseln, so ist $\alpha = \mathit{true}$. In diesem Fall ist α erfüllbar (sogar gültig).

2. Sonderfall: Leere Klausel. Falls eine der Klauseln von α leer ist, so ist α von der Form

$$\alpha \;=\; \ldots \wedge \sqcup \wedge \ldots \;=\; \ldots \wedge \mathit{false} \wedge \ldots$$

und somit unerfüllbar. (Wir erinnern daran, dass das Symbol $\sqcup$ als Schreibweise für die leere Klausel eingeführt wurde. Es entspricht der Formel *false*.)

Lemma 4.23 (Einfache Sonderfälle von KNF-SAT).

(a) Falls jede Klausel von α wenigstens ein positives Literal enthält, so ist α erfüllbar.

(b) Falls jede Klausel von α wenigstens ein negatives Literal enthält, so ist α erfüllbar.

(c) Jede KNF-Formel α ohne leere Klauseln, in welcher jedes Atom $x \in AP$ entweder nur positiv oder nur negativ vorkommt, ist erfüllbar.

Beweis. ad (a). Liegt eine KNF-Formel vor, in der jede Klausel wenigstens ein positives Literal enthält, so ist die Belegung, die jedem Aussagensymbol x den Wahrheitswert 1 zuordnet, eine erfüllende Belegung für α. Insbesondere ist α erfüllbar.

Die Argumentation für (b) ist analog. Falls jede Klausel von α wenigstens ein negatives Literal enthält, so ist die Belegung, die jedem Aussagensymbol x den Wahrheitswert 0 zuordnet, eine erfüllende Belegung für α.

ad (c). Sei α eine KNF-Formel ohne leere Klauseln, in welcher jedes Atom $x \in AP$ entweder nur positiv oder nur negativ vorkommt. Offenbar ist die Belegung I, welche jedes Atom x, das nur positiv in α vorkommt, mit 1 bewertet und jedes andere Atom mit 0, erfüllend für α. Z.B. ist

$$(x \vee \neg y) \wedge (x \vee \neg z) \wedge (\neg y \vee \neg z)$$

eine solche KNF-Formel, da x nur positiv und y und z nur negativ vorkommen. Die Belegung $I = [x = 1, y = 0, z = 0]$ ist offenbar erfüllend. □

Eine für mehrere Algorithmen bedeutsame Beobachtung ist, dass α genau dann erfüllbar ist, wenn wenigstens eine jener beiden Formeln erfüllbar ist, welche sich aus α ergeben, wenn ein fester Wahrheitswert für eines der Atome x unterstellt wird. Für KNF-Formeln kann die Vorgabe von Wahrheitswerten für Aussagensymbole sehr leicht durch folgende syntaktische Transformationen "simuliert" werden.

Da das Streichen tautologischer Klauseln, also von Klauseln der Form $\ldots \vee \neg x \vee x \vee \ldots$, die zueinander komplementäre Literale enthalten, keinen Einfluss auf die Wahrheitswerte einer KNF-Formel hat, können wir im Folgenden o.E. voraussetzen, dass die gegebene KNF-Formel α keine tautologischen Klauseln enthält. Wegen der Idempotenzgesetze kann man weiter annehmen, dass die Literale in den Klauseln paarweise verschieden sind.

Definition 4.24 (Ersetzung von Atomen durch Wahrheitswerte). Sei α eine KNF-Formel und x ein Atom. Die Formel $\alpha[x/0]$ entsteht aus α, indem folgende Transformationen durchgeführt werden:

(1) In jeder Klausel von α, welche das positive Literal x enthält, wird das Literal x gestrichen.

(2) Jede der verbliebenen Klausel, welche das negative Literal $\neg x$ enthält, wird (ersatzlos) gestrichen.

Abbildung 33 veranschaulicht die Definition. In Analogie hierzu sind die Formeln $\alpha[x/1]$ definiert. Im ersten Schritt ist jede Klausel, die das negative Literal $\neg x$ enthält, um $\neg x$ zu verkürzen. Der zweite Schritt eliminiert alle Klauseln, welche das positive Literal x enthalten. ■

$$\kappa_1 \wedge \ldots \kappa_{i-1} \wedge (L_1 \vee \ldots \vee L_{j-1} \vee x \vee L_{j+1} \vee \ldots \vee L_k) \wedge \kappa_{i+1} \wedge \ldots \wedge \kappa_m$$

$$(1) \Big\Downarrow \text{ wird ersetzt durch}$$

$$\kappa_1 \wedge \ldots \kappa_{i-1} \wedge (L_1 \vee \ldots \vee L_{j-1} \vee L_{j+1} \vee \ldots \vee L_k) \wedge \kappa_{i+1} \wedge \ldots \wedge \kappa_m$$

$$\kappa_1 \wedge \ldots \kappa_{i-1} \wedge (L_1 \vee \ldots \vee L_{j-1} \vee \neg x \vee L_{j+1} \vee \ldots \vee L_k) \wedge \kappa_{i+1} \wedge \ldots \wedge \kappa_m$$

$$(2) \Big\Downarrow \text{ wird ersetzt durch}$$

$$\kappa_1 \wedge \ldots \kappa_{i-1} \wedge \kappa_{i+1} \wedge \ldots \wedge \kappa_m$$

Abbildung 33: Ersetzen von x durch Wahrheitswert 0 in α

Die Rechtfertigung für die beiden Schritte der Transformation ist wie folgt. Zunächst stellen wir fest, dass die Vorgabe des Wahrheitswerts 0 für x einer syntaktischen Ersetzung von x durch *false* entspricht. Klauseln, welche das positive Literal x enthalten, etwa

$$\kappa \;=\; x \vee L_2 \vee \ldots \vee L_k,$$

werden durch die Ersetzung von x durch *false* zu:

$$\textit{false} \vee L_2 \vee \ldots \vee L_k \;\equiv\; \underbrace{L_2 \vee \ldots \vee L_k}_{\substack{\kappa \text{ nach Streichen} \\ \text{des positiven Literals } x}}$$

Dies erklärt den ersten Schritt der Transformation, in dem alle Klauseln um das positive Literal x verkürzt werden. Schritt 2 der Transformation beruht auf der Beobachtung, dass alle Klauseln von α, die das negative Literal $\neg x$ enthalten, durch die Ersetzung von x durch *false* zu Tautologien werden und daher gestrichen werden können. Beachte:

$$\underbrace{(\ldots \vee \neg \textit{false} \vee \ldots)}_{\equiv \textit{true}} \wedge \kappa_2 \wedge \ldots \wedge \kappa_m \;\equiv\; \textit{true} \wedge \kappa_2 \wedge \ldots \wedge \kappa_m$$

$$\equiv\; \kappa_2 \wedge \ldots \wedge \kappa_m$$

Die Formeln $\alpha[x/0]$ und $\alpha[x/\textit{false}]$ sind also äquivalent, wobei $\alpha[x/\textit{false}]$ durch die syntaktische Ersetzung aller Vorkommen des Atoms x in α durch die Formel *false* entsteht. Ebenso sind $\alpha[x/1]$ und $\alpha[x/\textit{true}]$ äquivalent. Hieraus ergibt sich, dass die Formeln $\alpha[x/b]$ folgende Eigenschaft haben, von der wir im Folgenden sehr häufig Gebrauch machen werden:

Lemma 4.25. *Sei α eine KNF-Formel über AP und $x \in AP$. Dann gilt für jede Belegung* I*:*

$$\alpha^{I} \;=\; \begin{cases} \alpha[x/0]^{I} & : \; \textit{falls } x^{I} = 0 \\ \alpha[x/1]^{I} & : \; \textit{falls } x^{I} = 1 \end{cases}$$

Beispiel 4.26 (Ersetzung von Atomen durch Wahrheitswerte in KNF-Formeln). Wir betrachten folgende Formel

$$\alpha \;=\; (x \vee \neg y \vee \neg z) \wedge (y \vee z \vee \neg w \vee \neg v) \wedge (\neg x \vee w) \wedge (\neg x \vee \neg z).$$

Zur Bildung von $\alpha[x/0]$ werden x aus der ersten Klausel entfernt und die dritte und vierte Klausel ersatzlos gestrichen. Die zweite Klausel wird unverändert übernommen, da sie weder x noch $\neg x$ als Literal enthält. Man erhält also:

$$\alpha[x/0] \;=\; (\neg y \vee \neg z) \wedge (y \vee z \vee \neg w \vee \neg v)$$

Zur Bildung von $\alpha[x/1]$ wird die erste Klausel gestrichen (da sie das positive Literal x enthält) und die zweite Klausel bleibt unverändert. Aus der dritten und vierten Klausel wird jeweils $\neg x$ gestrichen, also:

$$\alpha[x/1] \;=\; (y \vee z \vee \neg w \vee \neg v) \wedge w \wedge \neg z.$$

Durch das Streichen von Literalen aus Einheitsklauseln (im ersten Schritt der Transformation $\alpha \rightsquigarrow \alpha[x/\ldots]$) entsteht stets die leere Klausel. Ist z.B.

$$\gamma \;=\; (\neg y \vee \neg z) \wedge x,$$

so ist

$$\gamma[x/0] \;=\; (\neg y \vee \neg z) \wedge \sqcup.$$

Für die resultierende Formel $\gamma[x/0]$ kann das Erfüllbarkeitsproblem daher sofort mit "nein" beantwortet werden. (Siehe zweiter Sonderfall "leere Klausel".) Für eine KNF-Formel β, in der jede Klausel das negative Literal $\neg x$ enthält, entsteht beim Übergang zu $\beta[x/0]$ die triviale KNF-Formel *true* (die keine Klauseln enthält). Z.B. werden bei der Erstellung von $\beta[x/0]$ für

$$\beta \;=\; (\neg x \vee y \vee z) \wedge (\neg x \vee \neg z)$$

beide Klauseln gestrichen. Also ist $\beta[x/0] = \textit{true}$. ■

Offenbar kommt das Atom x in $\alpha[x/b]$ nicht mehr vor. Weiter ist klar, dass die Formeln $\alpha[x/0]$ und $\alpha[x/1]$ höchstens so viele Klauseln wie α enthalten und in Zeit $\mathcal{O}(|\alpha|)$ erstellt werden können. Ferner enthält jede der Klauseln von $\alpha[x/0]$ und $\alpha[x/1]$ höchstens so viele Literale wie die entsprechende Klausel von α.

Lemma 4.27 (Splitting-Regel). *Sei α eine KNF-Formel, x ein Atom. Dann gilt: α ist genau dann erfüllbar, wenn wenigstens eine der beiden Formeln $\alpha[x/0]$ oder $\alpha[x/1]$ erfüllbar ist.*

Beweis. "$\Longrightarrow$": Sei α erfüllbar und I eine erfüllende Belegung für α. Aus Lemma 4.25 folgt:

- Falls $x^{I} = 1$, so ist I ein Modell für $\alpha[x/1]$. Also ist $\alpha[x/1]$ erfüllbar.

- Falls $x^I = 0$, so ist I ein Modell für $\alpha[x/0]$. Also ist $\alpha[x/0]$ erfüllbar.

"$\Longleftarrow$": Ist etwa $\alpha[x/0]$ erfüllbar und I ein Modell für $\alpha[x/0]$, so betrachten wir die Belegung $I[x := 0]$, welche auf allen Atomen $y \neq x$ mit I übereinstimmt und x den Wahrheitswert 0 zuordnet. Also:

$$y^{I[x:=0]} \stackrel{\text{def}}{=} \begin{cases} y^I & : \text{ falls } y \in AP \setminus \{x\} \\ 0 & : \text{ falls } y = x \end{cases}$$

Wegen $x^{I[x:=0]} = 0$ und da x nicht in $\alpha[x/0]$ vorkommt, gilt:

$$\alpha^{I[x:=0]} \;=\; \alpha[x/0]^{I[x:=0]} = \alpha[x/0]^I = 1,$$

wobei in der ersten Gleichheit Lemma 4.25 für die Belegung $I[x := 0]$ benutzt wurde. □

Beruhend auf der Splitting-Regel kann das naive Backtracking für beliebige aussagenlogische Formeln (Algorithmus 6) für KNF-Formeln umformuliert werden. Anstelle einer expliziten Darstellung der Belegungen (mit Hilfe der booleschen Variablen $b_1, \ldots, b_n$) verkürzen wir hier die auf Erfüllbarkeit zu testende KNF-Formel α, indem wir zu $\alpha[x/1]$ bzw. $\alpha[x/0]$ übergehen. Dies ist in Algorithmus 7 formuliert.

Algorithmus 7 Naiver KNF-SAT-Beweiser: rekursiver Algorithmus SAT(α)

```
(*  Eingabe:   KNF-Formel α                              *)
(*  Rückgabe:  true, falls α erfüllbar. Andernfalls false. *)
-------------------------------------------------------------
IF α enthält die leere Klausel THEN return false FI
IF α = true THEN return true FI
wähle ein Atom x, das in α vorkommt          (* Splitting-Regel *)
IF SAT(α[x/0])
   THEN return true
   ELSE return SAT(α[x/1])
FI
```

Die Rekursionsstruktur liefert im Wesentlichen den Entscheidungsbaum, jedoch besteht die Chance, dass $\alpha[x/\ldots]$ deutlich kürzer als α ist und neben x auch andere Atome, die in α vorkommen, keine Vorkommen in $\alpha[x/\ldots]$ haben. Da hier nur nach der Erfüllbarkeit gefragt ist, kann das Verfahren abgebrochen werden, sobald ein mit *true* beschriftetes Blatt erreicht wird, da dann $\alpha[\ldots] = \textit{true}$ und somit auf die Erfüllbarkeit von α geschlossen werden kann.

Der Davis-Putnam-Algorithmus

Der Davis-Putnam-Algorithmus ist ein KNF-SAT-Beweiser, der sich in der Praxis sehr gut bewährt hat und auf dem naiven Backtracking-Algorithmus beruht. Die Idee besteht darin, durch den Einsatz gewisser Regeln das binäre Verzweigen gemäß der Splitting-Regel zu verhindern.

Im Folgenden verwenden wir häufig die Abkürzung “DP-Algorithmus”. In der Literatur wird dieser auch häufig DPLL-Algorithmus genannt, wobei die vier Buchstaben DPLL für die Namen der vier Wissenschaftler Davis, Putnam, Logeman und Loveland stehen, auf die die wesentlichen Konzepte des Algorithmus zurückgehen.

Zunächst stellen wir fest, dass die Reihenfolge, in welcher die Atome bei dem naiven Backtracking-Verfahren betrachtet werden, entscheidenden Einfluss auf die Anzahl der Rekursionsaufrufe haben kann. Liegt z.B. die unerfüllbare KNF-Formel

$$\alpha \;=\; (x \vee y \vee z) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z) \wedge \neg z$$

vor, so generiert der naive Backtracking-Algorithmus für die Reihenfolge x, y, z den vollständigen Entscheidungsbaum für α. Wesentlich günstiger ist die Reihenfolge z, y, x, da die Verzweigung $z/1$ sofort zu einer leeren Klausel führt und damit den rechten Teilbaum des Entscheidungsbaums für α nicht generiert. Diese Beobachtung lässt sich für beliebige KNF-Formel verallgemeinern, in denen eine Einheitsklausel vorkommt.

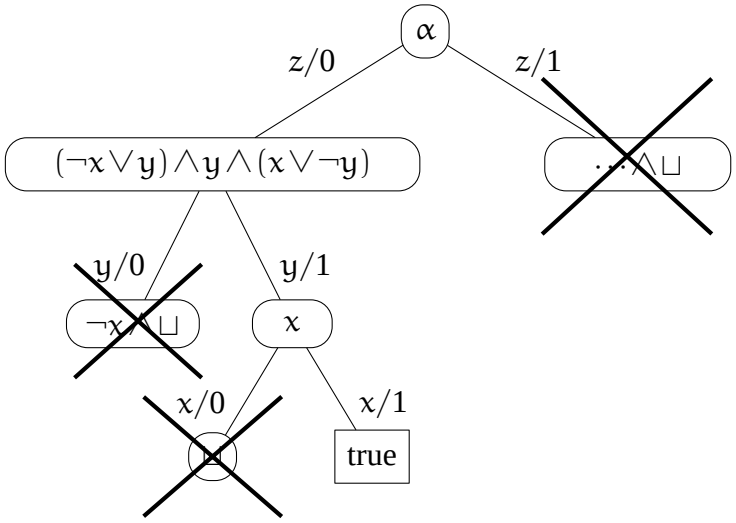


Abbildung 34: Beispiel zur Unit-Regel

Abbildung 34 illustriert diese Beobachtung am Beispiel der KNF-Formel

$$\alpha \;=\; (\neg x \vee y) \wedge (z \vee y) \wedge (z \vee x \vee \neg y) \wedge \neg z,$$

in der am Ende die Einheitsklausel $\neg z$ vorkommt. Die Formel

$$\alpha[z/0] \;=\; (\neg x \vee y) \wedge y \wedge (x \vee \neg y)$$

enthält die Einheitsklausel y. Daher ist nur $y/1$ relevant, während die Verzweigung $y/0$ eingespart werden kann, da $\alpha[z/0, y/0]$ die leere Klausel enthält.

Nehmen wir nun an, dass α eine KNF-Formel ist, in der eine Einheitsklausel vorkommt. Etwa $\alpha = \ldots \wedge x \wedge \ldots$. Ferner sei I eine Interpretation. Offenbar kann I höchstens dann für α erfüllend sein, wenn $x^I = 1$. In diesem Fall gilt $\alpha^I = \alpha[x/1]^I$ (siehe Lemma 4.25). Ist I also ein Modell für α, so ist I zugleich ein Modell für $\alpha[x/1]$. Ferner kann für jede erfüllende Belegung I für $\alpha[x/1]$ angenommen werden, dass x mit 1 belegt ist, da x keine Vorkommen in $\alpha[x/1]$ hat. Damit ergibt sich Teil (a) von Lemma 4.28. Die Aussage in Teil (b) ist symmetrisch und betrifft negative Einheitsklauseln.

Lemma 4.28 (Unit-Regel). *Sei α eine KNF-Formel und x ein Atom. Dann gilt:*

(a) Enthält α die Einheitsklausel x, so ist α genau dann erfüllbar, wenn $\alpha[x/1]$ erfüllbar ist.

(b) Enthält α die negative Einheitsklausel $\neg x$, so ist α genau dann erfüllbar, wenn $\alpha[x/0]$ erfüllbar ist.

Die Unit-Regel erlaubt es also, auf das binäre Verzweigen mit der Splitting-Regel zu verzichten und stattdessen nur eine der Formeln $\alpha[x/1]$ oder $\alpha[x/0]$ auf Erfüllbarkeit zu untersuchen (je nachdem ob die negative Einheitsklausel $\neg x$ oder die positive Einheitsklausel x in α vorkommt). Dies führt zur ersten Verfeinerung des naiven Backtracking-Algorithmus mit Hilfe der Unit-Regel, welche eingesetzt wird, wann immer sich eine KNF-Formel mit Einheitsklauseln ergibt.

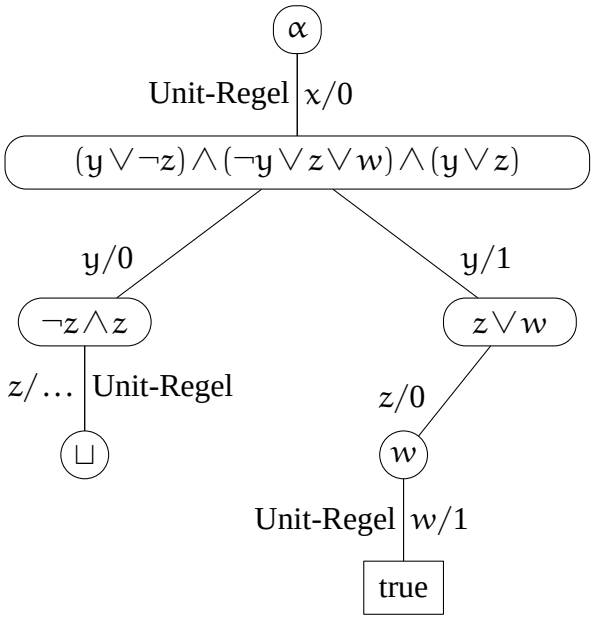


Abbildung 35: Beispiel zum Einsatz der Unit-Regel im DP-Algorithmus

Abbildung 35 zeigt den Rekursionsbaum, der sich für die erfüllbare Formel

$$\alpha \;=\; (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z \vee w) \wedge (x \vee y \vee z) \wedge \neg x$$

ergibt, wenn die Splitting-Regel nur dann eingesetzt wird, wenn die Unit-Regel nicht anwendbar ist, also eine KNF-Formel ohne Einheitsklauseln vorliegt. Durch den Einsatz der Unit-Regel finden also nur sieben Rekursionsaufrufe statt, bis das "linkeste" mit *true* beschriftete Blatt gefunden wird. Für die Formel

$$\begin{aligned} \alpha &= (x \vee y \vee \neg z) \wedge (y \vee z) \wedge \beta, && \text{wobei} \\ \beta &= (z \vee w) \wedge (z \vee \neg w) \wedge (\neg z \vee w) \wedge (\neg z \vee \neg w) \end{aligned}$$

greift die Unit-Regel erst auf den unteren Ebenen, sofern zuerst die Variablen x und y betrachtet werden. Es gilt jedoch $\alpha[x/0] = \alpha[x/1] \wedge (y \vee \neg z)$. Aus diesem Grund wäre es ausreichend, die kürzere Formel $\alpha[x/1]$ auf Erfüllbarkeit zu testen, was mit der nun folgenden Pure-Literal-Regel erreicht wird.

Wir sagen, dass das Atom x in α *nur positiv* vorkommt, wenn keine der Klauseln von α das negative Literal $\neg x$ enthält und das positive Literal x wenigstens in einer Klausel von α enthalten ist. Entsprechend bedeutet die Formulierung, "x kommt in α *nur negativ* vor", dass keine der Klauseln von α das positive Literal x enthält und das negative Literal $\neg x$ wenigstens in einer Klausel von α enthalten ist.

Lemma 4.29 (Die Pure-Literal-Regel). *Sei α eine KNF-Formel und x ein Atom, das in α vorkommt. Dann gilt:*

(a) Kommt x nur positiv in α vor, so ist α genau dann erfüllbar, wenn $\alpha[x/1]$ erfüllbar ist.

(b) Kommt x nur negativ in α vor, so ist α genau dann erfüllbar, wenn $\alpha[x/0]$ erfüllbar ist.

Beweis. Wir weisen nur Teil (a) nach. Aussage (b) folgt mit analogen Argumenten. Aufgrund der Splitting-Regel impliziert die Erfüllbarkeit von $\alpha[x/1]$ die Erfüllbarkeit von α. Wir nehmen nun an, dass α erfüllbar ist und dass x nur positiv in α vorkommt. Zu zeigen ist die Erfüllbarkeit von $\alpha[x/1]$. Sei I ein Modell für α.

- Ist $x^I = 1$, so ist $\alpha[x/1]^I = \alpha^I = 1$ und somit I ein Modell für $\alpha[x/1]$.
- Sei $x^I = 0$ und κ eine Klausel von $\alpha[x/1]$. Dann ist κ entweder eine Klausel von α oder $\kappa \vee x$ ist eine Klausel von α. Da I eine erfüllende Belegung für α ist, gilt $\kappa^I = 1$ im ersten Fall. Im zweiten Fall gilt $(\kappa \vee x)^I = 1$. Wegen $x^I = 0$ folgt ebenfalls $\kappa^I = 1$.

In beiden Fällen ist I ein Modell für $\alpha[x/1]$.

Eine alternative Argumentation für den Beweis von Aussage (a) in Lemma 4.29 benutzt die Beobachtung, dass wenn x nur positiv in α vorkommt, so ergibt sich $\alpha[x/0]$ durch Streichen von x aus allen Klauseln, welche das positive Literal x enthalten, während $\alpha[x/1]$ aus α durch Streichen dieser Klauseln entsteht. Die KNF-Formel $\alpha[x/1]$ besteht also genau aus denjenigen Klauseln von α, in denen x nicht vorkommt. Somit gilt:

$$\alpha[x/0] \equiv \alpha[x/1] \wedge \gamma,$$

wobei γ für diejenige KNF-Formel steht, die man erhält, indem man die Konjunktion aller Klauseln von α bildet, welche x enthalten und anschließend aus jeder dieser Klauseln x streicht. Die KNF-Formel $\alpha[x/0]$ ist daher höchstens dann erfüllbar, wenn $\alpha[x/1]$ erfüllbar ist. Aus der Splitting-Regel folgt daher, dass α genau dann erfüllbar, wenn $\alpha[x/1]$ erfüllbar ist. □

Erweitert man Algorithmus 7 um die Pure-Literal-Regel, so erhält man für die Eingabeformel

$$\alpha = (x \vee y \vee \neg z) \wedge (y \vee z) \wedge (\neg z \vee w)$$

die erfüllende Belegung $[x = 1, y = 1, z = 0, w = 1]$ auf direktem Weg, da x und y in α nur positiv vorkommen und z in

$$\alpha[x/1, y/1] = \neg z \vee w$$

nur negativ vorkommt. Siehe Abbildung 36 auf Seite 149. Eine weitere Verbesserung des naiven Backtracking-Algorithmus beruht auf folgender Subsumierungsregel. Diese befasst sich mit dem Fall, dass eine Klausel κ' eine andere Klausel κ um zusätzliche Literale erweitert.

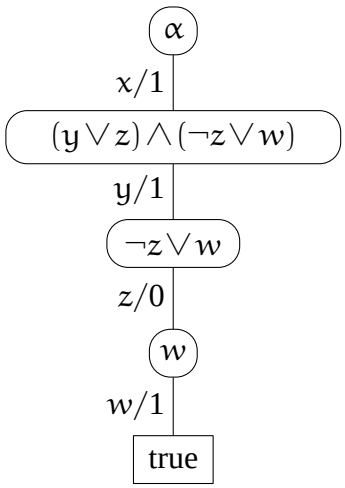


Abbildung 36: Beispiel zur Pure-Literal-Regel

Bezeichnung 4.30 (Teilklausel). Seien $\kappa = L_1 \vee \ldots \vee L_k$ und $\lambda = L'_1 \vee \ldots \vee L'_m$ zwei Klauseln. Wir sagen, dass κ Teilklausel von λ ist, i.Z. $\kappa \subseteq \lambda$, falls jedes Literal von κ in λ enthalten ist:

$$\kappa \subseteq \lambda \quad \text{gdw} \quad \underbrace{\{L_1, \ldots, L_k\}}_{\substack{\text{Menge der Literale} \\ \text{von } \kappa}} \subseteq \underbrace{\{L'_1, \ldots, L'_m\}}_{\substack{\text{Menge der Literale} \\ \text{von } \lambda}}$$

Der Begriff "Teilklausel" bezieht sich also auf die Sichtweise von Klauseln als Mengen von Literalen. Z.B. ist $\kappa = y \vee z$ Teilklausel von $\lambda = z \vee y \vee x$. ■

Ist κ Teilklausel von λ, so ist jede erfüllende Belegung von κ zugleich eine erfüllende Belegung von λ. Es gilt also $\kappa \Vdash \lambda \equiv \kappa \vee \tau$ für eine geeignete Klausel τ. Die Folgerungsrelation $\Vdash$ bezogen auf Klauseln wird auch *Subsumierungsrelation* genannt. Kommen beide Klauseln κ und λ in einer KNF-Formel α vor, so ist Klausel λ redundant, da der "Effekt" von λ durch κ subsumiert wird:

$$\text{wegen } \kappa \Vdash \lambda \text{ gilt:} \quad \kappa \wedge \lambda \equiv \kappa$$

Die gegebene KNF-Formel α ist also zu derjenigen KNF-Formel äquivalent, die aus α entsteht, indem die längere Klausel λ aus α gestrichen wird. Diese KNF-Formel bezeichnen wir mit $\alpha \setminus \{\lambda\}$. Hat etwa α die Form $\kappa \wedge \lambda \wedge \beta$, so gilt:

$$\underbrace{\kappa \wedge \lambda \wedge \beta}_{\alpha} \equiv \underbrace{\kappa \wedge \beta}_{\alpha \setminus \{\lambda\}}.$$

Für den Spezialfall, dass α zwei oder mehr Vorkommen der Klausel κ enthält (d.h., $\kappa = \lambda$), so ist die Schreibweise $\alpha \setminus \{\kappa\}$ so zu lesen, dass alle bis auf *eines* der Vorkommen von κ gestrichen werden. Beispielsweise gilt:

$$\underbrace{(x \vee \neg y)}_{=\kappa} \wedge \underbrace{(x \vee \neg y \vee z)}_{=\kappa'} \wedge (\neg x \vee \neg y \vee w) \quad \equiv \quad (x \vee \neg y) \wedge (\neg x \vee \neg y \vee w)$$

Lemma 4.31 (Subsumierungsregel). *Sei α eine KNF-Formel und κ und λ zwei Klauseln von α, so dass κ Teilklausel von λ ist. Dann sind α und $\alpha \setminus \{\lambda\}$ äquivalent.*

Im Kontext des DP-Algorithmus ist nur relevant, ob die Formeln α und $\alpha \setminus \{\lambda\}$ beide erfüllbar oder beide unerfüllbar sind (später wird hierfür der Begriff erfüllbarkeitsäquivalent eingeführt). Auch wenn initial eine KNF-Formel vorliegt, die keine Klauseln enthält, die Teilklausel anderer Klauseln sind, so können durch den rekursiven Abstieg des Backtracking-Algorithmus subsumierte Klauseln entstehen. Ist z.B.

$$\alpha \;=\; (\neg x \vee y \vee z) \wedge (w \vee u \vee y \vee z) \wedge \ldots,$$

so enthält $\alpha[x/1]$ u.a. die beiden Klauseln $y \vee z$ und $w \vee u \vee y \vee z$.

Der Pseudo-Code des Davis-Putnam-Algorithmus mit Unit-, Pure-Literal- und Subsumierungsregel ist in Algorithmus 8 auf Seite 151 angegeben. Die Wahl zuerst die Pure-Literal-Regel, dann die Unit-Regel auf Anwendbarkeit zu prüfen, ist willkürlich. Beide Regeln haben zum Ziel die Splitting-Regel zu verzögern, um so den Rekursionsbaum klein zu halten. Da die Vereinfachung einer KNF-Formel durch die Subsumierungsregel im Allgemeinen sehr aufwendig ist, kann es empfehlenswert sein, auf die Subsumierungsregel zu verzichten, und stattdessen nur die Pure-Literal-, Unit- und Splitting-Regel anzuwenden. Statt im zweiten Schritt nur den Trivialfall der KNF-Formel *true* zu behandeln, können auch weitere syntaktische Kriterien für die Erfüllbarkeit eingesetzt werden. So kann etwa die Aussage von Lemma 4.23 auf Seite 142 benutzt werden, um einen weiteren Terminalfall zu integrieren.

Beispiel 4.32 (Worst-case für den Davis-Putnam-Algorithmus). Trotz der eingesetzten Regeln, die das naive Backtracking verfeinern und oftmals das Erfüllbarkeitsproblem sehr schnell lösen können, ist die worst-case Anzahl an Rekursionsaufrufen des Davis-Putnam-Algorithmus (Algorithmus 8 auf Seite 151) exponentiell in der Anzahl an Aussagensymbolen der vorliegenden Formel. Ein Beispiel für KNF-Formeln, für welche der Davis-Putnam-Algorithmus auch unter Anwendung aller Regeln exponentielle Kosten verursacht, sind die wie folgt rekursiv definierten Formeln α_n:

$$\alpha_0 \;\stackrel{\text{def}}{=}\; (x_0 \vee y_0) \wedge (\neg x_0 \vee y_0) \wedge (x_0 \vee \neg y_0) \wedge (\neg x_0 \vee \neg y_0)$$

und

$$\alpha_n \;\stackrel{\text{def}}{=}\; \alpha_{n-1} \wedge (x_n \vee y_n) \wedge (\neg x_n \vee \neg y_n)$$

für $n = 1, 2, 3, \ldots$. Da bereits α_0 unerfüllbar ist, ist jede der Formeln α_n unerfüllbar. Wir diskutieren nun die Kosten von Algorithmus 8 für die Eingabeformel α_n, wobei wir die Reihenfolge $x_n, y_n, x_{n-1}, y_{n-1}, \ldots, x_0, y_0$ annehmen, in der die Aussagensymbole als Kandidatinnen für die Pure-Literal-, Unit- oder Splitting-Regel herangezogen werden. Die folgenden Betrachtungen gelten aus Symmetriegründen auch für jede andere Reihenfolge, in der x_0 und y_0 am Ende stehen. Der durch den Aufruf von SAT(α_n) erzeugte Baum hat die in Abbildung 37 angegebene Gestalt.

Zunächst stellen wir fest, dass in den α_i-Ebenen die Splitting-Regel angewandt wird. Diese liefert Formeln der Gestalt $\alpha_{i-1} \wedge y_i$ und $\alpha_{i-1} \wedge \neg y_i$, für welche die Pure-Literal-Regel (oder auch Unit-Regel) greift. Die Subsumierungsregel kommt überhaupt nicht zum Zuge. Der Rekursionsbaum enthält 2^i Knoten für die Formel α_{n-i}. Daher erhält man insgesamt asymptotisch 2^n Rekursionsaufrufe. Die worst-case-Laufzeit ist also bereits ohne Berücksichtigung der Kosten für die Regelanwendungen exponentiell. ∎

Algorithmus 8 Davis-Putnam-Algorithmus für KNF-Formeln: rekursiver Algorithmus SAT(α)

(* Eingabe: KNF-Formel α *)
(* Rückgabe: *true*, falls α erfüllbar. Andernfalls *false*. *)

(* Terminalfälle *)

IF α enthält die leere Klausel $\sqcup$
THEN return *false* **FI**

IF $\alpha = true$
THEN return *true* **FI**

(* Pure-Literal-Regel *)

IF es gibt ein Atom x, das in α nur positiv vorkommt
THEN return SAT($\alpha[x/1]$) **FI**

IF es gibt ein Atom x, das in α nur negativ vorkommt
THEN return SAT($\alpha[x/0]$) **FI**

(* Unit-Regel *)

IF α enthält eine Einheitsklausel **THEN**
wähle eine Einheitsklausel κ
IF $\kappa = x$ ist eine positive Einheitsklausel
THEN return SAT($\alpha[x/1]$) **FI**
IF $\kappa = \neg x$ ist eine negative Einheitsklausel
THEN return SAT($\alpha[x/0]$) **FI**
FI

(* Subsumierungsregel *)

WHILE α enthält zwei Klauseln κ und λ mit $\kappa \subseteq \lambda$ **DO**
wähle solche Klauseln κ und λ und streiche λ
OD

(* Splitting-Regel *)

wähle ein Atom x, das in α vorkommt
IF SAT($\alpha[x/0]$)
THEN return *true*
ELSE return SAT($\alpha[x/1]$)
FI

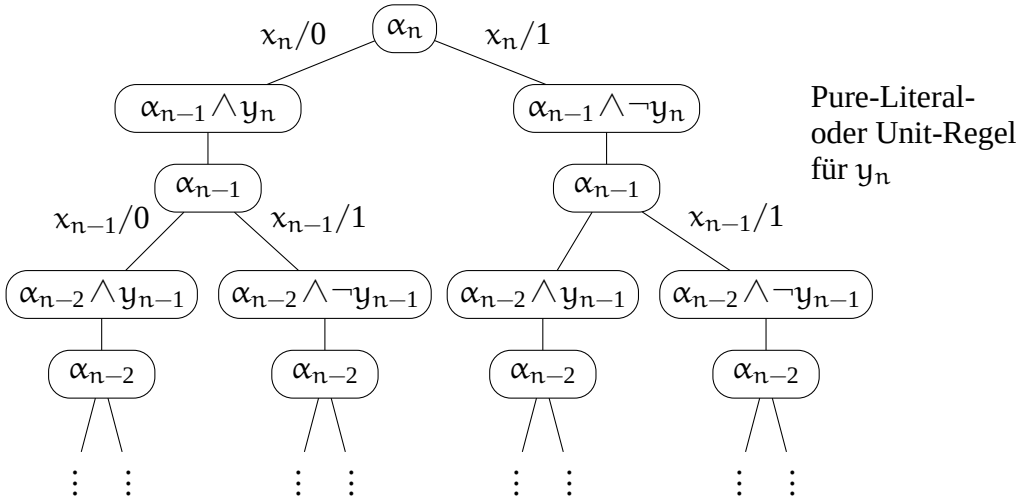


Abbildung 37: Beispiel zum worst-case für den Davis-Putnam-Algorithmus

Die exponentielle worst-case Laufzeit bezieht sich auf "extreme" KNF-Formeln. Für Formeln mit zahlreichen erfüllenden Belegungen ist eine weitaus geringere Laufzeit zu erwarten. Z.B. finden für gültige Eingabeformeln mit n Atomen höchstens $n+1$ Rekursionsaufrufe statt, da dann ein mit *true* beschriftetes Blatt erreicht wird. (Dies gilt bereits für das naive Backtracking.) Aber auch für unerfüllbare Eingabeformeln zeigen Erfahrungswerte, dass der Davis-Putnam-Algorithmus oftmals sehr viel bessere (als exponentielle) Laufzeit hat; insbesondere dann, wenn geeignete Heuristiken eingesetzt werden, mit denen ein Aussagensymbol für die Anwendung der Splitting-Regel ausgewählt wird, z.B.:

- Wähle ein Atom, das am häufigsten in α vorkommt.
- Wähle ein Atom x, für welches die Summe der Längen aller Klauseln, welche x oder $\neg x$ als Literal enthalten, maximal ist.
- Wähle ein Atom, das in den kürzesten Klauseln am häufigsten vorkommt.
- Wähle ein Atom x, für welches die Differenz der positiven und negativen Vorkommen von x in den kürzesten Klauseln (betragsmäßig) maximal ist.

$\vdots$

Die ersten beiden Heuristiken haben zum Ziel, möglichst kurze Formeln $\alpha[x/\ldots]$ zu generieren. Die Motivation für die dritte Heuristik besteht darin, möglichst schnell die Unit-Regel anwenden zu können. Entsprechend zielt die vierte Heuristik auf die Pure-Literal-Regel ab.

Für eine Implementierung des Davis-Putnam-Algorithmus empfiehlt sich die Verwendung von geeigneten Datenstrukturen, welche die Generierung der Formeln $\alpha[x/\ldots]$ unterstützen (z.B. Listen mit Verweisen auf alle Klauseln, die x bzw. $\neg x$ als Literal enthalten) sowie Datenstrukturen, die alle Atome, auf welche die Pure-Literal- oder Unit-Regel anwendbar sind, geeignet verwalten.

Abschließend bemerken wir, dass der Platzbedarf des Davis-Putnam-Algorithmus nur linear in der Formellänge ist, da der zusätzliche Platz durch den Rekursionsstack dominiert wird und die Rekursionstiefe durch $\mathcal{O}(n)$ beschränkt ist. Dasselbe gilt auch für das naive Backtracking.

Die Annahme, dass der Davis-Putnam Algorithmus mit einer KNF-Formel gestartet wird, ist zwar keine Einschränkung, da jede aussagenlogische Formel in eine äquivalente KNF-Formel transformiert werden kann, jedoch kann die konstruierte KNF-Formel exponentiell länger als die ursprüngliche Formel sein. In Satz 4.14 auf Seite 131 hatten wir nachgewiesen, dass dieses Phänomen unabhängig von dem gewählten Verfahren zur Konstruktion einer äquivalenten KNF-Formel ist.

Von SAT zu 3SAT

Das etwaige exponentielle Blowup durch den Übergang von einer aussagenlogischen Formel α zu einer äquivalenten KNF-Formel kann im Kontext des Erfüllbarkeitsproblems verhindert werden, indem auf die Erstellung einer äquivalenten KNF-Formel verzichtet und stattdessen eine *erfüllbarkeitsäquivalente* KNF-Formel derselben asymptotischen Länge konstruiert wird. Tatsächlich ist eine solche algorithmische Transformation sogar dann möglich, wenn höchstens drei Literale pro Klausel erlaubt sind. Damit kann SAT auf 3SAT (das Erfüllbarkeitsproblem für KNF-Formeln mit höchstens drei Literalen pro Klausel) zurückgeführt werden, ohne ein exponentielles Blowup in Kauf nehmen zu müssen.

Definition 4.33 (Erfüllbarkeitsäquivalenz). *Zwei aussagenlogische Formeln α_1, α_2 heißen erfüllbarkeitsäquivalent, i.Z.*

$$\alpha_1 \equiv^{\text{SAT}} \alpha_2,$$

wenn sie entweder beide erfüllbar oder beide unerfüllbar sind. ■

Erfüllbarkeitsäquivalente Formeln können völlig verschiedene erfüllende Belegungen haben und sind im Allgemeinen nicht semantisch äquivalent. Beispielsweise sind die atomaren Formeln x, wobei $x \in AP$, paarweise *nicht* semantisch äquivalent, jedoch sind sie erfüllbarkeitsäquivalent. Die Literale x und $\neg x$ bilden ein Beispiel für erfüllbarkeitsäquivalente Formeln, die *keine* gemeinsame erfüllende Belegung haben. Man beachte, dass es lediglich zwei Äquivalenzklassen hinsichtlich Erfüllbarkeitsäquivalenz gibt: die Klasse aller erfüllbaren Formeln und die Klasse aller unerfüllbaren Formeln.

Bezeichnung 4.34 (3KNF). *Eine aussagenlogische Formel α ist in 3KNF, falls sie in KNF ist und jede Klausel aus höchstens drei Literalen besteht. Analoge Bedeutung hat die Bezeichnung 2KNF oder k-KNF für ganze Zahlen $k \geqslant 1$.* ■

Z.B. ist $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3 \vee \neg x_3)$ eine 3KNF-Formel. Durch Wiederholen von Literalen kann man zu jeder 3KNF-Formel eine äquivalente 3KNF-Formel mit genau drei Literalen pro Klausel erstellen. Z.B. ist $(x_1 \vee x_1 \vee x_1) \wedge (\neg x_2 \vee \neg x_2 \vee x_3)$ eine zu $x_1 \wedge (\neg x_2 \vee x_3)$ äquivalente 3KNF-Formel mit genau drei Literalen pro Klausel. Die Existenz von äquivalenten 3KNF-Formeln mit genau drei Literalen pro Klausel gilt auch für 3KNF-Formeln mit der leeren Klausel. Hier kann die leere Klausel durch $(x \vee x \vee x) \wedge (\neg x \vee \neg x \vee \neg x)$ für ein beliebiges Atom x ersetzt werden.

Ist α eine k-KNF-Formel, so entsteht durch den Übergang von α zu $\alpha[x/1]$ oder $\alpha[x/0]$ wieder eine KNF-Formel mit höchstens k Literalen pro Klausel. Dies ist klar, da $\alpha[x/1]$ und $\alpha[x/0]$ durch Streichen von Klauseln und Streichen von Literalen aus Klauseln aus α hervorgehen. Mit α sind also auch $\alpha[x/1]$ und $\alpha[x/0]$ k-KNF-Formeln.

Satz 4.35 (Linearzeit-Transformation: Formel $\rightsquigarrow$ erfüllbarkeitsäquivalente 3KNF). *Zu jeder aussagenlogischen Formel α kann eine erfüllbarkeitsäquivalente 3KNF-Formel (d.h. eine KNF-Formel mit höchstens 3 Literalen pro Klausel) in linearer Zeit $\mathcal{O}(|\alpha|)$ konstruiert werden.*

Da in linearer Zeit nur polynomiell viele Daten verarbeitet werden können, ist auch die Länge der konstruierten 3KNF-Formel durch $\mathcal{O}(|\alpha|)$ beschränkt.

Beweis. Die wesentliche Idee der Transformation ist, die Formel in PNF zu überführen und dann den inneren Knoten des Syntaxbaums 3KNF-Formeln konstanter Länge zuzuordnen. Im Folgenden sei α die Eingabeformel mit den Atomen $x_1, \ldots, x_n$. Der erste Schritt der Transformation führt α in eine äquivalente Formel α_{PNF} in PNF derselben asymptotischen Länge über.

- Durch Ersetzen jeder Teilformel der Form $\beta \wedge \mathit{true}$ oder $\mathit{true} \wedge \beta$ durch β und jeder Teilformel $\beta \vee \mathit{true}$ oder $\mathit{true} \vee \beta$ durch *true*, sowie analoge Transformationen für jedes Vorkommen von *false*, entsteht eine zu α und α_{PNF} äquivalente Formel α', so dass entweder $\alpha' = \mathit{true}$ oder $\alpha' = \mathit{false}$ oder α' ist eine PNF-Formel, welche die Konstanten *true* und *false* nicht enthält.
- Falls $\alpha' = \mathit{true}$, so ist α' eine 3KNF-Formel mit 0 Klauseln. Falls $\alpha' = \mathit{false}$, so ist α' eine 3KNF-Formel bestehend aus der leeren Klausel. In beiden Fällen ist die Transformation abgeschlossen.

Im Folgenden nehmen wir an, dass $\alpha_{\text{PNF}} = \alpha'$ eine PNF-Formel der Länge $|\alpha_{\text{PNF}}| = \mathcal{O}(|\alpha|)$ ist, welche zu α äquivalent ist und in welcher die Konstanten *true* oder *false* nicht vorkommen. Wir betrachten nun den *Syntaxbaum* T von α_{PNF}, wobei wir die in α vorkommenden Literale wie atomare Formeln behandeln. Die inneren Knoten des Syntaxbaums für α_{PNF} stehen für die Teilformeln von α_{PNF}, deren Top-Level-Operator eine Konjunktion oder Disjunktion ist. Die inneren Knoten werden entsprechend mit $\wedge$ oder $\vee$ beschriftet. Die Blätter des Syntaxbaums stehen für die Vorkommen der Literale in α_{PNF}. Wir schreiben β_v für die durch Knoten v des Syntaxbaums dargestellte Teilformel von α_{PNF}. Ist also v ein innerer mit *op* beschrifteter Knoten von T, wobei $\mathit{op} \in \{\wedge, \vee\}$, und sind w und u die beiden Söhne von v, so ist

$$\beta_v \;=\; \beta_w \,\mathit{op}\, \beta_u.$$

Jedes Blatt v identifizieren wir mit dem durch v repräsentierten Literal, also

$$\beta_v \in \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}, \;\; \text{falls } v \text{ ein Blatt ist.}$$

Für die inneren Knoten verwenden wir beliebige Knotennamen und setzen voraus, dass diese paarweise verschieden sind und nicht in $\{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$ vorkommen. Im Folgenden bezeichne v_0 den Wurzelknoten von T und $\text{In}(T)$ die Menge (der Knotennamen) aller inneren Knoten von T.

Die nun konstruierte 3KNF-Formel α_{3KNF} verwendet die in α vorkommenden Atome $x_1, \ldots, x_n$ und zusätzlich die Namen der inneren Knoten. Die zugrundeliegende Menge an Aussagensymbolen ist also

$$AP \;\stackrel{\text{def}}{=}\; \{x_1, \ldots, x_n\} \cup \text{In}(T).$$

Die 3KNF-Formel α_{3KNF} entsteht nun durch die Konjunktion der Einheitsklausel v_0 und 3KNF-Formeln σ_v für jeden inneren Knoten v des Syntaxbaums. Die 3KNF-Formeln σ_v ergeben sich durch geeignete Umformungen aus Formeln der Gestalt

$$v \leftrightarrow (w \vee u), \quad \text{falls } v \text{ ein innerer Knoten mit den Söhnen } w \text{ und } u \text{ und mit } \vee \text{ beschriftet ist,}$$

$$v \leftrightarrow (w \wedge u), \quad \text{falls } v \text{ ein innerer Knoten mit den Söhnen } w \text{ und } u \text{ und mit } \wedge \text{ beschriftet ist.}$$

Zur Erinnerung: die Söhne w und u von v sind entweder innere Knoten (also Atome in AP) oder Blätter, also Literale in $\{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$.

Die Formeln $v \leftrightarrow (w \mathit{op} u)$ werden nun in 3KNF mit jeweils genau drei Klauseln gebracht. Wie zuvor ist $\mathit{op} \in \{\wedge, \vee\}$ die Beschriftung von Knoten v. Ist v ein mit $\wedge$ beschrifteter Knoten und sind w und u die beiden Söhne von v, so definieren wir:

$$\sigma_v \stackrel{\text{def}}{=} (v \vee \neg w \vee \neg u) \wedge (\neg v \vee w) \wedge (\neg v \vee u)$$

Man überzeugt sich nun leicht davon, dass die 3KNF-Formel σ_v zur Formel $v \leftrightarrow (w \wedge u)$ äquivalent ist. Es gilt nämlich:

$$\begin{aligned} \sigma_v[v/0] &= \neg w \vee \neg u \equiv \neg(w \wedge u) \\ \sigma_v[v/1] &= w \wedge u \end{aligned}$$

und somit $\sigma_v \equiv v \leftrightarrow (w \wedge u)$.

Jedem AND-Knoten v im Syntaxbaum kann also eine 3KNF-Formel σ_v mit drei Klauseln zugeordnet werden. Analoges gilt für die OR-Knoten. Ist v ein mit $\vee$ beschrifteter Knoten mit den Söhnen w und u, so definieren wir:

$$\sigma_v \stackrel{\text{def}}{=} (v \vee \neg w) \wedge (v \vee \neg u) \wedge (\neg v \vee w \vee u)$$

Offenbar sind auch in diesem Fall $v \leftrightarrow (w \vee u)$ und σ_v äquivalent, da

$$\begin{aligned} \sigma_v[v/0] &= \neg w \wedge \neg u \equiv \neg(w \vee u) \\ \sigma_v[v/1] &= w \vee u \end{aligned}$$

Die resultierende zu α und α_{PNF} äquivalente 3KNF-Formel ist nun wie folgt definiert:

$$\alpha_{\text{3KNF}} \stackrel{\text{def}}{=} v_0 \wedge \bigwedge_{v \in \text{In}(T)} \sigma_v,$$

wobei – wie zuvor – v_0 für den Wurzelknoten des Syntaxbaums T steht und $\text{In}(T)$ die Menge aller inneren Knoten von T bezeichnet.

Korrektheit der Transformation. Hierzu ist zu zeigen, dass die Kosten für die Konstruktion von α_{3KNF} linear in $|\alpha|$ sind und dass α_{3KNF} zu α erfüllbarkeitsäquivalent ist.

Zunächst zur Länge von α_{3KNF}. Offenbar enthält α_{3KNF} höchstens

$$3 \cdot |\alpha_{\text{PNF}}| + 1 = \mathcal{O}(|\alpha|)$$

Klauseln. Beachte, dass die Anzahl an inneren Knoten des Syntaxbaums mit der Anzahl an Konjunktionen und Disjunktionen in α_{PNF} übereinstimmt. Die Anzahl an $\wedge$- und $\vee$-Teilformeln

von α_{PNF} ist durch $|\alpha_{\text{PNF}}|$ nach oben beschränkt. Da für jeden inneren Knoten v die Formel σ_v aus drei Klauseln besteht, ist die Klauselanzahl von α_{3KNF} durch $3 \cdot |\alpha_{\text{PNF}}| + 1$ beschränkt, wobei der Summand "+1" für die Einheitsklausel v_0 steht.

Da α_{PNF} sowie der Syntaxbaum T von α_{PNF} in linearer Zeit aus α konstruiert werden können und sich α_{3KNF} aus T im Wesentlichen durch Hinschreiben der oben explizit angegebenen 3KNF-Formeln σ_v ergibt, kann α_{3KNF} aus α in linearer Zeit konstruiert werden.

Nun zur Erfüllbarkeitsäquivalenz von α und α_{3KNF}. Wie oben erwähnt, bezeichnet β_v die durch Knoten v dargestellte Teilformel von α_{PNF}. Es gilt also $\beta_v = v$ für jedes Blatt v. Ist v ein innerer Knoten, der mit dem Operator *op* beschriftet ist und dessen Söhne w und u sind, so ist $\beta_v = \beta_w \mathit{op} \beta_u$. Ferner ist $\alpha_{\text{PNF}} = \beta_{v_0}$ für den Wurzelknoten v_0.

1. Wir nehmen an, dass α erfüllbar ist und zeigen die Erfüllbarkeit von α_{3KNF}. Sei I eine erfüllende Belegung für α. Wir erweitern nun I zu einer erfüllenden Belegung J für α_{3KNF} wie folgt:

   - $x_i^J \stackrel{\text{def}}{=} x_i^I$ für $1 \leqslant i \leqslant n$,
   - $v^J \stackrel{\text{def}}{=} \beta_v^I$ für jeden inneren Knoten v des Syntaxbaums von α_{PNF}.

   Offenbar gilt $v^J = \beta_v^I$ für jeden Knoten v des Syntaxbaums. (Für die inneren Knoten gilt dies nach Definition von J. Für die Blätter ist dies offensichtlich, da diese für Literale x_i oder $\neg x_i$ stehen und die Belegungen I und J auf $\{x_1, \ldots, x_n\}$ übereinstimmen.) Zu zeigen ist nun, dass $\sigma_v^J = 1$ für jeden inneren Knoten v und dass $v_0^J = 1$.

   Zunächst zum Nachweis, dass $\sigma_v^J = 1$ für alle inneren Knoten v. Wir erläutern dies am Beispiel eines AND-Knotens v, d.h., v ist mit dem Operator $\wedge$ beschriftet. Seien w und u die beiden Söhne von v. Dann gilt:

   $$\begin{aligned} & v^J = \beta_v^I = 1 \\ \text{gdw} \quad & (\beta_w \wedge \beta_u)^I = 1 \\ \text{gdw} \quad & \beta_w^I = \beta_u^I = 1 \\ \text{gdw} \quad & w^J = u^J = 1 \\ \text{gdw} \quad & (w \wedge u)^J = 1 \end{aligned}$$

   und somit $\sigma_v^J = (v \leftrightarrow (w \wedge u))^J = 1$. Die Argumentation für die mit $\vee$ beschrifteten Knoten ist analog. Weiter ist

   $$\begin{aligned} v_0^J &= \beta_{v_0}^I && \text{(nach Definition von J)} \\ &= \alpha_{\text{PNF}}^I && \text{(da } \alpha_{\text{PNF}} = \beta_{v_0}\text{)} \\ &= \alpha^I && \text{(da } \alpha \equiv \alpha_{\text{PNF}}\text{)} \\ &= 1 && \text{(da I erfüllend für } \alpha\text{)} \end{aligned}$$

   Also ist J eine erfüllende Belegung für $\alpha_{\text{3KNF}} = v_0 \wedge \bigwedge_{v \in \text{In}(T)} \sigma_v$.

2. Wir setzen nun die Erfüllbarkeit von α_{3KNF} voraus und weisen nach, dass auch α erfüllbar ist. Sei J eine erfüllende Belegung für α_{3KNF}. Wir zeigen, dass J zugleich eine erfüllende Belegung für α ist. Hierzu benutzen wir ein induktives Argument, um folgende Aussage (*) nachzuweisen:

$$v^J = \beta_v^J \quad \text{für jeden Knoten } v \text{ von } T \tag{*}$$

Aussage (*) weisen wir durch Induktion nach der Höhe des Teilbaums von v nach.

- Im Induktionsanfang sind die Blätter von T zu betrachten, da genau deren Teilbäume die Höhe 0 haben. Für die Blätter ist die Aussage $v^J = \beta_v^J$ klar, da das Blatt v mit dem Literal β_v identifiziert wird.
- Im Induktionsschritt betrachten wir einen inneren Knoten v, so dass dessen Teilbaum die Höhe h hat. Dann ist $h \geqslant 1$ und die Teilbäume der beiden Söhne w und u von v haben die Höhe $\leqslant h-1$. Die Induktionsvoraussetzung kann also auf w und u angewandt werden. Dies liefert

  $$w^J = \beta_w^J \text{ und } u^J = \beta_u^J.$$

  Wir nehmen an, dass Knoten v die Beschriftung *op* hat, wobei $op \in \{\wedge, \vee\}$. Für die durch Knoten v dargestellte Teilformel β_v von α_{PNF} gilt also:

  $$\beta_v = \beta_w \, op \, \beta_u$$

  Wegen $\alpha_{3KNF}^J = 1$, sind alle Klauseln von α_{3KNF} unter J wahr. Daher ist auch σ_v unter J wahr. Da σ_v zu $v \leftrightarrow (w \, op \, u)$ äquivalent ist, gilt:

  $$(v \leftrightarrow (w \, op \, u))^J = \sigma_v^J = 1$$

  Wegen $\beta_v = \beta_w \, op \, \beta_u$ und da $w^J = \beta_w^J$ und $u^J = \beta_u^J$ (nach Induktionsvoraussetzung, siehe oben) folgt hieraus:

  $$v^J = (w \, op \, u)^J \overset{IV}{=} (\beta_w \, op \, \beta_u)^J = \beta_v^J$$

Mit Hilfe von (*) angewandt auf den Wurzelknoten $v = v_0$ erhalten wir:

$$\begin{aligned} \alpha^J &= \alpha_{PNF}^J && (\text{da } \alpha \equiv \alpha_{PNF}) \\ &= \beta_{v_0}^J && (\text{da } \beta_{v_0} = \alpha_{PNF}) \\ &= v_0^J && (\text{Aussage (*)}) \\ &= 1 && (\text{da } v_0 \text{ Klausel von } \alpha_{3KNF} \text{ und } \alpha_{3KNF}^J = 1) \end{aligned}$$

Damit ist die Erfüllbarkeit von α gezeigt.

□

Beispiel 4.36 (Formel $\rightsquigarrow$ erfüllbarkeitsäquivalente 3KNF). Unser Ausgangspunkt ist die PNF-Formel

$$\alpha = \alpha_{PNF} = ((\neg x \vee y) \wedge z) \vee (x \wedge \neg y).$$

Wir betrachten nun den Syntaxbaum für α, wobei die Blätter für die in α vorkommenden Literale stehen und die inneren Knoten für je eine der Teilformeln

$$\begin{array}{lcl}
\beta_{v_0} & = & ((\neg x \vee y) \wedge z) \vee (x \wedge \neg y) \quad = \quad \beta_u \vee \beta_r \quad = \quad \alpha \\
\beta_r & = & x \wedge \neg y \\
\beta_u & = & (\neg x \vee y) \wedge z \qquad\qquad = \quad \beta_w \wedge z \\
\beta_w & = & \neg x \vee y
\end{array}$$

Der Wurzelknoten ist also v_0 und stellt die gesamte Formel α dar. Der Syntaxbaum ist in Abbildung 38 skizziert, wobei die Knotenbeschriftung AND bzw. OR für $\wedge$ bzw. $\vee$ steht. Die für α konstruierte erfüllbarkeitsäquivalente 3KNF-Formel α_{3KNF} verwendet die ursprünglichen Atome x, y, z sowie zusätzlich die Atome v_0, w, u, r und hat die Gestalt:

$$\alpha_{3KNF} \;=\; v_0 \wedge \sigma_{v_0} \wedge \sigma_w \wedge \sigma_u \wedge \sigma_r,$$

wobei $\sigma_{v_0}, \sigma_w, \sigma_u$ und σ_r 3KNF-Formeln bestehend aus je drei Klauseln sind, die Semantik der Knoten v_0, w, u, r widerspiegeln, also $\sigma_{v_0} \equiv v_0 \leftrightarrow (u \vee r)$, $\sigma_u \equiv u \leftrightarrow (w \wedge z)$, $\sigma_w \equiv w \leftrightarrow (\neg x \vee y)$ und $\sigma_w \equiv r \leftrightarrow (x \wedge \neg y)$. ■

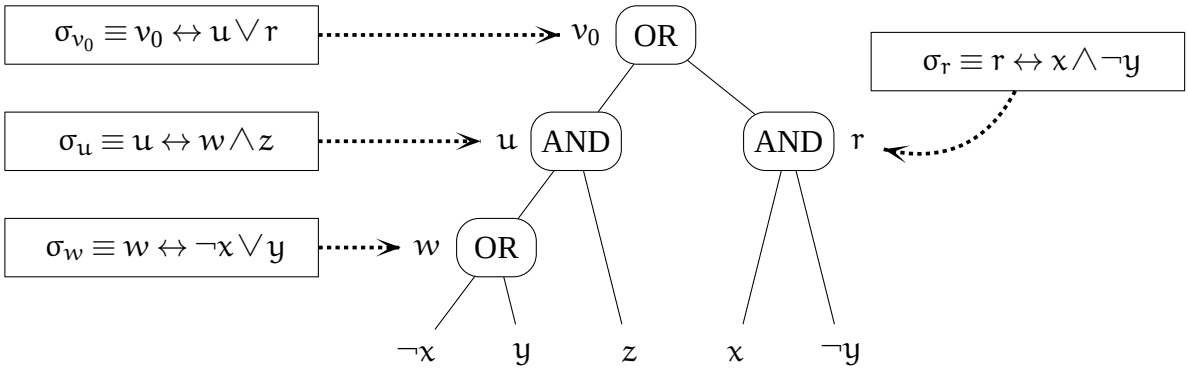


Abbildung 38: Syntaxbaum für $((\neg x \vee y) \wedge z) \vee (x \wedge \neg y)$

4.4 Resolution

Der Resolutionskalkül leistet einen entscheidenden Beitrag für die Logikprogrammierung und andere Bereiche der Informatik (KI, regelbasierte Systeme, etc.). Im Gegensatz zu den SAT-Beweisern, die wir in den vorangegangenen Abschnitten diskutiert haben, zielt Resolution auf den Nachweis der Unerfüllbarkeit einer KNF-Formel (oder allgemeiner Klauselmenge) ab. Wesentliche Idee der Resolution ist es, die logische Folgerbarkeit der leeren Klausel *false* $= \sqcup$ durch eine Kette von elementaren Folgerungsschritten zu belegen. Man spricht in diesem Kontext auch von einer *Widerlegung* für die Eingabeformel. Logische Folgerbarkeit $\alpha \models \beta$ für beliebige Formeln α kann mittels Resolution über den Umweg einer Zerlegung von $\neg\beta$ in Klauseln $\kappa_1, \ldots, \kappa_m$ durch eine Widerlegung für $\alpha \wedge \kappa_1 \wedge \ldots \wedge \kappa_m$ nachgewiesen werden.

Mengenschreibweise von Klauseln und KNF-Formeln. Im Folgenden identifizieren wir Klauseln mit der betreffenden Literalmenge. Ist also $\kappa = L_1 \vee \ldots \vee L_k$ eine Klausel, so wird

κ als Literalmenge $\{L_1, \ldots, L_k\}$ aufgefasst. Damit werden Klauseln, in denen ein Literal mehrfach vorkommt, durch äquivalente Klauseln ohne mehrfach vorkommende Literale ersetzt. Etwa $x \vee x \vee \neg y$ wird durch $\{x, x, \neg y\} = \{x, \neg y\}$ dargestellt. Ferner abstrahiert die Mengenschreibweise von der Reihenfolge der Literale. Dies ist aufgrund der Äquivalenzregeln für die Assoziativität und Kommutativität von Disjunktionen gerechtfertigt. Die leere Klausel $\sqcup$ entspricht nun der leeren Menge. Analog verfahren wir mit aussagenlogischen KNF-Formeln, die wir als Klauselmengen auffassen. Ist etwa $\alpha = \kappa_1 \wedge \ldots \wedge \kappa_m$, so wird α mit der Klauselmenge $\{\kappa_1, \ldots, \kappa_m\}$ identifiziert.

Man beachte, dass die Mengenschreibweise für Klauseln einer Disjunktion entspricht, die Mengenschreibweise für KNF-Formeln einer Konjunktion. Daher erhalten wir gegensätzliche Interpretationen der leeren Menge:[16]

- Die leere Klausel (also die leere Literalmenge) $\sqcup$ steht für *false* und ist daher unerfüllbar.
- Die leere Klauselmenge $\varnothing$ steht für *true* und ist daher gültig.

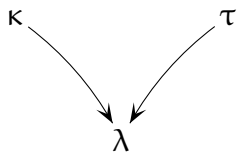
Damit ergibt sich auch der Unterschied zwischen der leeren Klauselmenge $\varnothing$ und der einelementigen Klauselmenge $\{\sqcup\}$. Während die leere Klauselmenge $\varnothing$ stets gültig ist, ist $\{\sqcup\}$ stets unerfüllbar. Etwas gewöhnungsbedürftig sind Einheitsklauseln, für die nun die Schreibweisen $\{x\}$ und x bzw. $\{\neg x\}$ und $\neg x$ gleichwertig sind. Analoges gilt für KNF-Formeln mit nur einer Klausel. Für diese ist die Schreibweise κ (ohne Mengenklammer) gleichbedeutend mit der Mengenschreibweise $\{\kappa\}$. Im Folgenden verwenden wir die Formel- oder Mengennotation; je nachdem, welche Sichtweise einfacher ist. (An vielen Stellen ist die Wahl jedoch willkürlich.)

Die Idee des Resolutionskalküls beruht darauf, durch syntaktische Transformationen logische Schlussfolgerungen zu ziehen. Hierzu wird folgende Beobachtung eingesetzt: Sind $\kappa = x \vee \lambda_1$ und $\tau = \neg x \vee \lambda_2$ zwei Klauseln der als wahr angenommenen Grundmenge, so kann auf die logische Folgerbarkeit der Klausel $\lambda_1 \vee \lambda_2$ geschlossen werden.

Definition 4.37 (Resolvent). Seien κ und τ Klauseln und L ein Literal, so dass $L \in \kappa$ und das negierte Literal $\overline{L}$ in τ enthalten sind.[17] Dann heißt die Klausel

$$\lambda \;=\; \kappa \setminus \{L\} \;\cup\; \tau \setminus \{\overline{L}\}$$

der Resolvent von κ und τ über L. Man spricht auch von dem L-Resolventen oder kurz einem Resolventen, von κ und τ. Die Klauseln κ und τ werden in diesem Kontext auch *Elternklauseln* genannt.

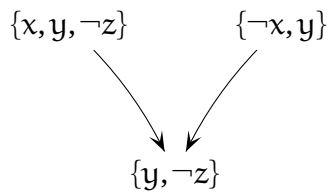


Oftmals verwendet man eine graphische Notation wie in dem Bild oben, um anzudeuten, dass die Klausel λ ein Resolvent der beiden Klauseln κ und τ ist. In Darstellungen durch gerichtete Graphen sind die Knoten mit Klauseln beschriftet. Kanten stehen für die Resolventenbildung. ■

[16] Aus diesem Grund ist es üblich ein Sondersymbol wie $\sqcup$ für die leere Klausel – anstelle des sonst üblichen Symbols $\varnothing$ für die leere Menge – zu verwenden.

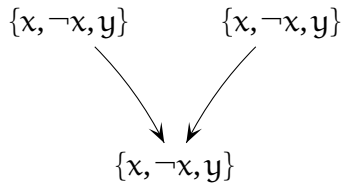
[17] Zur Erinnerung: $\overline{x} = \neg x$ und $\overline{\neg x} = x$. $\overline{L}$ ist also dasjenige Literal, welches zu $\neg L$ äquivalent ist.

Beispiel 4.38 (Resolventen). Z.B. ist $y \vee \neg z$ ein Resolvent aus $x \vee y \vee \neg z$ und $\neg x \vee y$.



Die Klausel $y \vee \neg z$ ergibt sich durch Streichen des positiven Literals x aus der Klausel $x \vee y \vee \neg z$ und des negativen Literals $\neg x$ aus $\neg x \vee y$ und anschließende disjunktive Verknüpfung der verkürzten Klauseln. Die Klauseln $x \vee y \vee \neg z$ und $x \vee \neg z \vee w$ haben jedoch keine Resolventen, da sie keine zueinander komplementäre Literale enthalten. ■

Durch Resolventenbildung kann die leere Klausel $\sqcup$ entstehen; nämlich dann, wenn der Resolvent der beiden Einheitsklauseln x und $\neg x$ gebildet wird. Zwei Klauseln können durchaus auch zwei oder mehrere Resolventen haben; nämlich dann, wenn sie mehrere zueinander komplementäre Literale enthalten. Z.B. sind $x \vee \neg x \vee z$ und $y \vee \neg y \vee z$ Resolventen der Klauseln $x \vee y \vee z$ und $\neg x \vee \neg y$. In solchen Fällen sind jedoch alle Resolventen tautologisch. Resolventen einer tautologischen Klausel sind zwar möglich, aber langweilig. Z.B.:



Das folgende Lemma zeigt, dass Resolventen einer KNF-Formel logische Folgerungen sind.

Lemma 4.39 (Resolventenlemma (1. Teil)). *Sei α eine KNF-Formel und λ ein Resolvent zweier Klauseln von α. Dann gilt $\alpha \Vdash \lambda$.*

Beweis. Sei $\lambda = \kappa \setminus \{x\} \cup \tau \setminus \{\neg x\}$, wobei κ, τ Klauseln von α sind und $x \in \kappa$, $\neg x \in \tau$. Sei I ein Modell für α. Zu zeigen ist, dass $\lambda^I = 1$; also dass eines der Literale von λ den Wahrheitswert 1 unter I hat.

1. Fall: $x^I = 1$. Wegen $\neg x \in \tau$ und $\tau^I = 1$ gibt es ein Literal $L \in \tau \setminus \{\neg x\}$, welches unter I den Wahrheitswert 1 hat. Dieses Literal L ist auch in λ enthalten.

2. Fall: $x^I = 0$. In diesem Fall liegt eine zum ersten Fall analoge Situation für κ vor. Wegen $\kappa^I = 1$ gibt es ein anderes in κ vorkommendes Literal $L \neq x$, das unter I den Wahrheitswert 1 hat. Dieses Literal L liegt in λ. □

Lemma 4.40. *Ist α eine KNF-Formel, so gilt $\alpha \wedge \lambda \equiv \alpha$ für jeden Resolventen λ zweier Klauseln von α.*

Beweis. Sei λ ein Resolvent von α. Es ist klar, dass jedes Modell für $\alpha \wedge \lambda$ zugleich ein Modell für α ist. Umgekehrt gilt $\lambda^I = 1$ für jedes Modell I für α, da $\alpha \Vdash \lambda$ (Resolventenlemma). Also ist I ein Modell für $\alpha \wedge \lambda$. □

Bezeichnung 4.41 (Resolventenmenge, Resolutionsabschluss). Sei α eine KNF-Formel. Dann bezeichnet die Resolventenmenge

$$Res(\alpha) \;=\; \big\{\, \lambda : \lambda \text{ ist Resolvent zweier Klauseln in } \alpha \,\big\}$$

diejenige Menge bestehend aus allen Resolventen, die aus den Klauseln von α gebildet werden können. Der Resolutionsabschluss $Res^*(\alpha)$ von α ergibt sich durch sukzessive Resolventenbildung:

$$Res^*(\alpha) \;=\; \bigcup_{i \geqslant 0} Res^i(\alpha),$$

wobei $Res^0(\alpha) = \alpha$ und $Res^{i+1}(\alpha) = Res^i(\alpha) \cup Res\big(Res^i(\alpha)\big)$ für $i = 0, 1, 2, \ldots$. ■

Beispiel 4.42 (Resolutionsabschluss). Für $\alpha \;=\; (x \vee \neg y \vee z) \wedge (y \vee z) \wedge (\neg x \vee z) \wedge \neg z$ erhalten wir (wobei die Klauseln als Literalmengen angegeben werden):

$$\begin{aligned}
Res^0(\alpha) &= \big\{\, \{x, \neg y, z\},\ \{y, z\},\ \{\neg x, z\},\ \{\neg z\} \,\big\} \\
Res^1(\alpha) &= Res^0(\alpha) \cup \big\{\, \{x, z\},\ \{\neg y, z\},\ \{x, \neg y\},\ \{y\},\ \{\neg x\} \,\big\} \\
Res^2(\alpha) &= Res^1(\alpha) \cup \big\{\, \{z\},\ \{x\},\ \{\neg y\} \,\big\} \\
Res^3(\alpha) &= Res^2(\alpha) \cup \big\{\, \sqcup \,\big\}
\end{aligned}$$

und $Res^k(\alpha) = Res^3(\alpha) = Res^*(\alpha)$ für alle $k \geqslant 4$. ■

Offenbar gilt $Res^0(\alpha) \subseteq Res^1(\alpha) \subseteq Res^2(\alpha) \subseteq \ldots \subseteq Res^*(\alpha) \subseteq 2^{\mathfrak{L}}$, wobei $\mathfrak{L}$ die Menge aller Literale bezeichnet, die in wenigstens einer der Klauseln von α vorkommen. Da $\mathfrak{L}$ (und somit auch die Potenzmenge $2^{\mathfrak{L}}$) endlich ist, ist die Folge der Resolventenmengen $Res^i(\alpha)$ ab einem gewissen Index stationär. Wir halten diese Beobachtung in folgendem Lemma fest:

Lemma 4.43 (Endlichkeit des Resolutionsabschlusses). *Für jede KNF-Formel α gibt es eine natürliche Zahl k, so dass*

$$Res^k(\alpha) \;=\; Res^{k+1}(\alpha) \;=\; Res^{k+2}(\alpha) \;=\; \ldots \;=\; Res^*(\alpha).$$

Eine sehr grobe obere Schranke für die in Lemma 4.43 genannte Zahl k ist $|2^{\mathfrak{L}}| = 2^{2n} = 4^n$, wobei n für die Anzahl an Aussagensymbolen steht, die in α vorkommen.

Durch Induktion nach i kann man mit Hilfe des Resolventenlemmas zeigen, dass alle Klauseln in $Res^i(\alpha)$ logische Folgerungen von α sind. Hieraus folgt die Korrektheit der Resolution als Kalkül für logische Folgerbarkeit:

Lemma 4.44 (Resolventenlemma (2. Teil)). *Sei α eine KNF-Formel. Dann gilt $\alpha \Vdash \lambda$ für alle $\lambda \in Res^*(\alpha)$.*

Insbesondere ist die Resolution als Verfahren für den Nachweis der Unerfüllbarkeit einer KNF-Formel korrekt. Man spricht auch von Widerlegungskorrektheit:

Corollar 4.45 (Widerlegungskorrektheit). *Falls $\sqcup \in Res^*(\alpha)$, so ist α unerfüllbar.*

Beweis. Falls $\sqcup \in \mathit{Res}^*(\alpha)$, so gilt $\alpha \Vdash \sqcup = \mathit{false}$ (zweiter Teil des Resolventenlemmas). Hieraus folgt die Unerfüllbarkeit von α. □

Wir werden später sehen, dass auch die Umkehrung gilt. Zunächst geben wir jedoch eine alternative Charakterisierung des Resolutionsabschlusses an.

Definition 4.46 ((Resolutions-)Herleitung). Sei α eine KNF-Formel, aufgefasst als Klauselmenge, und λ eine Klausel. Eine *Resolutionsherleitung* (kurz *Herleitung*) von λ aus α ist eine Folge von Klauseltripeln

$$\langle \kappa_1, \tau_1, \lambda_1 \rangle \ \langle \kappa_2, \tau_2, \lambda_2 \rangle \ \ldots \ \langle \kappa_m, \tau_m, \lambda_m \rangle,$$

so dass $\lambda_m = \lambda$ und so dass für alle $i \in \{1, 2, \ldots, m\}$ gilt:

$$\lambda_i \text{ ist Resolvent von } \kappa_i \text{ und } \tau_i, \text{ und } \kappa_i,\ \tau_i \in \alpha \cup \{\lambda_1, \lambda_2, \ldots, \lambda_{i-1}\}.$$

m wird die Länge der Herleitung genannt. Da die jeweils ersten beiden Klauseln κ_i, τ_i der Herleitungstripel lediglich die Funktion haben anzugeben, woraus die hergeleitete Klausel λ_i entstanden ist, verzichtet man oft auf deren Angabe und nennt die Klauselfolge $\lambda_1 \lambda_2 \ldots \lambda_m$ eine Herleitung von $\lambda_m = \lambda$ aus α. Eine Klausel λ ist aus α herleitbar, wenn entweder $\lambda \in \alpha$ oder wenn es eine Herleitung der Länge $m \geqslant 1$ für λ aus α gibt. Alle Klauseln aus α werden als in Herleitungen der Länge 0 erzeugbar angesehen. ■

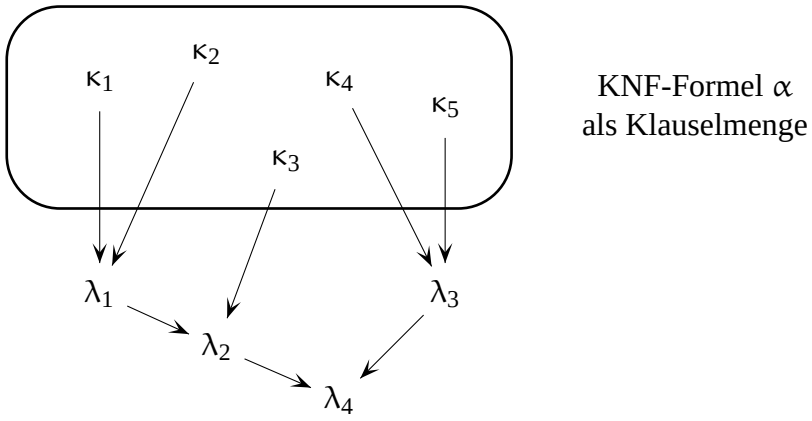


Abbildung 39: Graphische Darstellung einer Resolutionsherleitung

Abbildung 39 zeigt wie Herleitungen als azyklische Digraphen skizziert werden können. Die in einer Herleitung benötigten Klauseln aus α haben keine Vorgänger. Alle über ein oder mehrere Resolutionsschritte hergeleiteten Klauseln haben genau zwei Vorgängerknoten; nämlich die beiden Elternklauseln. Das Bild in Abbildung 39 könnte etwa für folgende Herleitung stehen:

$$\langle \kappa_1, \kappa_2, \lambda_1 \rangle \ \ \langle \lambda_1, \kappa_3, \lambda_2 \rangle \ \ \langle \kappa_4, \kappa_5, \lambda_3 \rangle \ \ \langle \lambda_2, \lambda_3, \lambda_4 \rangle$$

Beispiel 4.47 (Resolutionsherleitung). Ist $\alpha = (x \vee y) \wedge (\neg x \vee \neg z) \wedge (y \vee z)$, so ist

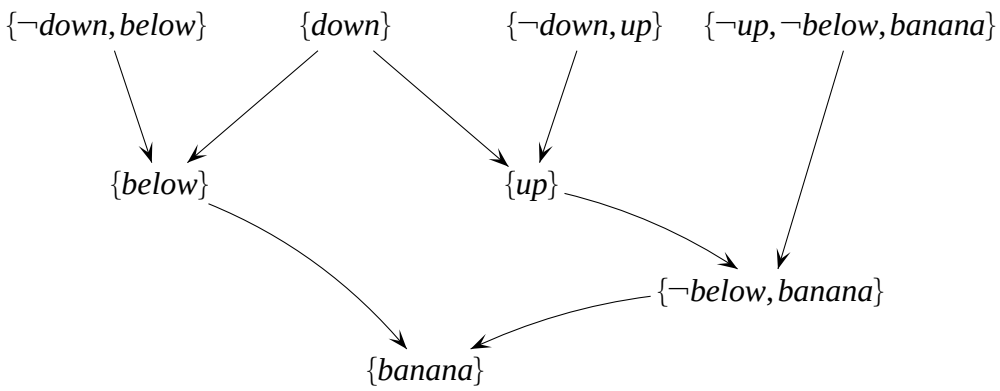
$$\langle \{x, y\}, \{\neg x, \neg z\}, \{y, \neg z\} \rangle \ \ \langle \{y, z\}, \{y, \neg z\}, \{y\} \rangle$$

eine Herleitung der Einheitsklausel $\{y\}$ aus α. Diese hat die Länge 2. ■

Beispiel 4.48 (Resolutionsherleitung (Affen-Bananen-Problem)). Wir kehren zu dem Affen-Bananen-Problem zurück. Ausgangspunkt ist die KNF-Formel $\alpha = \kappa_1 \wedge \kappa_2 \wedge \kappa_3 \wedge \kappa_4$ mit den Hornklauseln $\kappa_1 = \neg \mathit{down} \vee \mathit{below}$, $\kappa_2 = \neg \mathit{down} \vee \mathit{up}$, $\kappa_3 = \neg \mathit{up} \vee \neg \mathit{below} \vee \mathit{banana}$ und der Einheitsklausel $\kappa_4 = \mathit{down}$. Die folgende Skizze stellt die Resolutionsherleitung

$$\langle \kappa_2, \kappa_4, \lambda_1 \rangle \quad \langle \kappa_1, \kappa_4, \lambda_2 \rangle \quad \langle \kappa_3, \lambda_1, \lambda_3 \rangle \quad \langle \lambda_2, \lambda_3, \lambda_4 \rangle$$

dar, wobei $\lambda_1 = \{\mathit{up}\}$, $\lambda_2 = \{\mathit{below}\}$, $\lambda_3 = \{\neg \mathit{below}, \mathit{banana}\}$ und $\lambda_4 = \{\mathit{banana}\}$. Aus dem zweiten Teil des Resolventenlemmas folgt die für den Affen erfreuliche Aussage, dass er die Bananen erreichen kann. ■



Bemerkung 4.49 (Irrelevante Resolutionsschritte). Es ist klar, dass in einer Herleitung alle Resolutionsschritte weggelassen werden können, die für die Generierung der letzten Klausel λ_m nicht benötigt werden. Z.B. ist

$$\langle \{x, y\}, \{\neg x\}, \{y\} \rangle \quad \langle \{z, \neg w\}, \{w\}, \{z\} \rangle \quad \langle \{y\}, \{\neg y\}, \sqcup \rangle$$

eine Herleitung aus $\alpha = (x \vee y) \wedge \neg x \wedge \neg y \wedge (z \vee \neg w) \wedge w$, die zu

$$\langle \{x, y\}, \{\neg x\}, \{y\} \rangle \quad \langle \{y\}, \{\neg y\}, \sqcup \rangle$$

verkürzt werden kann, da zur Herleitung der leeren Klausel die im zweiten Resolutionsschritt generierte Klausel $\{z\}$ nicht benötigt wird. In diesem Sinn kann man stets annehmen, dass Herleitungen unverkürzbar sind. ■

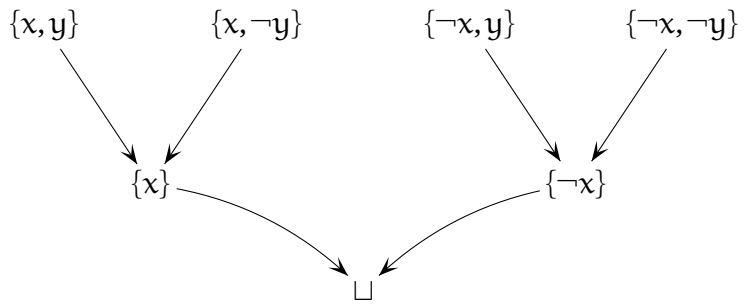
Durch Induktion über die Länge m einer Herleitung kann gezeigt werden, dass alle herleitbaren Klauseln in $\mathit{Res}^*(\alpha)$ liegen. Umgekehrt hat jede Klausel in $\mathit{Res}^*(\alpha)$ eine Herleitung aus α. Diese Aussage kann nachgewiesen werden, indem man durch Induktion nach i zeigt, dass es zu jeder Klausel in $\mathit{Res}^i(\alpha)$ eine Herleitung aus α gibt. Wir erhalten damit folgende Aussage:

Lemma 4.50 (Resolutionsabschluss und Herleitbarkeit). *$\mathit{Res}^*(\alpha)$ ist die Menge aller Klauseln, für die es eine Herleitung aus α gibt.*

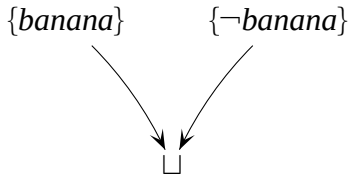
Aus Corollar 4.45 auf Seite 161 und Lemma 4.50 ergibt sich, dass jede Klauselmenge α, aus welcher die leere Klausel herleitbar ist, widersprüchlich (d.h. unerfüllbar) ist. Dies erklärt folgende Bezeichnung:

Definition 4.51 ((Resolutions-)Widerlegung). Eine *Resolutionswiderlegung* (kurz Widerlegung) für α bezeichnet eine Resolutionsherleitung der leeren Klausel $\sqcup$ aus α. ■

Folgende Abbildung zeigt eine Widerlegung für die unerfüllbare KNF-Formel $(x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y)$.



Beispiel 4.52 (Resolutionswiderlegung (Affen-Bananen-Problem)). Wir betrachten nun die Klauselmenge $\alpha \cup \{\neg \mathit{banana}\}$, wobei α wie in Beispiel 4.48 gegeben ist. Die negative Einheitsklausel $\{\neg \mathit{banana}\}$ steht für die Negation der Frage nach der Erreichbarkeit der Banane. Die in Beispiel 4.48 angegebene Herleitung kann nun um den Resolutionsschritt



erweitert werden. Man erhält somit eine Resolutionswiderlegung für $\alpha \cup \{\neg \mathit{banana}\}$. Hieraus folgt zunächst die Unerfüllbarkeit von $\alpha \cup \{\neg \mathit{banana}\}$ und somit $\alpha \models \mathit{banana}$. ■

Da aus der Existenz einer Widerlegung (also einer Herleitung von $\sqcup$) auf die Unerfüllbarkeit der betreffenden Klauselmenge geschlossen werden kann, ist Resolution als Kalkül für den Nachweis der Unerfüllbarkeit von KNF-Formeln *korrekt*. Der nun folgende Hauptsatz dieses Abschnitts sichert die *Vollständigkeit* der Resolution für den Nachweis der Unerfüllbarkeit zu, indem er belegt, dass auch die Umkehrung gilt; also, dass die leere Klausel aus α herleitbar ist, sofern α unerfüllbar ist. Man spricht daher auch von *Widerlegungsvollständigkeit*.

Satz 4.53 (Resolutionssatz der Aussagenlogik). *Sei α eine KNF-Formel. Dann ist α genau dann unerfüllbar, wenn $\sqcup \in \mathit{Res}^*(\alpha)$.*

Beweis. Die Korrektheit der Resolution (die Implikation "$\Longleftarrow$") wird durch Corollar 4.45 (Seite 161) belegt. Wir zeigen nun die Widerlegungsvollständigkeit des Resolutionskalküls. Wir nehmen die Unerfüllbarkeit von α an und zeigen, dass die leere Klausel $\sqcup$ über 0 oder mehrere Resolutionsschritte aus α resolviert werden kann. Diese Aussage beweisen wir durch Induktion nach der Anzahl n an Aussagensymbolen, die in α vorkommen.

Der Induktionsanfang ist klar, da für $n = 0$ eine KNF-Formel vorliegt, die keine Aussagensymbole enthält und daher entweder zu *false* oder zu *true* äquivalent ist. Da die Unerfüllbarkeit von α vorausgesetzt wird, ist nur der erste Fall möglich. Daher gilt $\sqcup \in \alpha \in \mathit{Res}^*(\alpha)$. Nun zum

Induktionsschritt $n \Longrightarrow n+1$. Wir nehmen an, dass eine unerfüllbare KNF-Formel α mit $n+1$ Aussagensymbolen vorliegt. Sei x eines der in α vorkommenden Aussagensymbole. Die wie in Bezeichnung 4.24 auf Seite 142 definierten KNF-Formeln

$$\alpha_0 \;=\; \alpha[x/0], \quad \alpha_1 \;=\; \alpha[x/1]$$

sind aus höchstens n Aussagensymbolen aufgebaut. Mit α sind auch α_0 und α_1 unerfüllbar (Splitting-Regel, siehe Lemma 4.27, Seite 144). Daher liefert die Induktionsvoraussetzung angewandt auf α_0 und α_1, dass $\sqcup \in \mathit{Res}^*(\alpha_0)$ und $\sqcup \in \mathit{Res}^*(\alpha_1)$. Wir zeigen nun, dass

$$\big\{\sqcup, \{x\}\big\} \;\cap\; \mathit{Res}^*(\alpha) \;\neq\; \varnothing \qquad (*)$$

Die Idee für den Nachweis dieser Aussage beruht darauf, die Resolutionsschritte, die zur Herleitung von $\sqcup$ aus α_0 führten, zu einer Herleitung der leeren Klausel $\sqcup$ oder der positiven Einheitsklausel $\{x\}$ aus α zu liften. Hierzu verfahren wir wie folgt. Falls $\sqcup \in \alpha_0$, so gilt entweder $\sqcup \in \alpha$ oder $\{x\} \in \alpha$. In diesem Fall ist Aussage (*) offenbar erfüllt. Im Folgenden nehmen wir $\sqcup \notin \alpha_0$ an und weisen Aussage (*) nach. Wegen $\sqcup \in \mathit{Res}^*(\alpha_0)$ gibt es eine Resolutionswiderlegung

$$\langle \kappa_1, \tau_1, \lambda_1\rangle \; \langle \kappa_2, \tau_2, \lambda_2\rangle \; \ldots \; \langle \kappa_m, \tau_m, \lambda_m\rangle$$

der Länge $m \geqslant 1$. Jede der Klauseln κ von α_0 ist entweder eine Klausel von α oder ist durch Streichen des positiven Literals x aus einer Klausel von α entstanden. Dies erlaubt es, eine Resolutionsherleitung

$$\langle \kappa'_1, \tau'_1, \lambda'_1\rangle \; \langle \kappa'_2, \tau'_2, \lambda'_2\rangle \; \ldots \; \langle \kappa'_m, \tau'_m, \lambda'_m\rangle$$

aus α wie folgt zu definieren. Für $i = 1, \ldots, m$ sind die Klauseln κ'_i wie folgt definiert.

- Ist κ_i eine Klausel von α_0 und α, so setzen wir $\kappa'_i = \kappa_i$.
- Ist $\kappa_i \in \alpha_0 \setminus \alpha$, so ist κ_i aus einer Klausel $\kappa \in \alpha$, welche das positive Literal x enthält, durch Streichen von x entstanden. Wir kehren nun – durch Wiedereinfügen von x – zur ursprünglichen Klausel von α zurück. Hierzu setzen wir $\kappa'_i \;=\; \kappa \;=\; \kappa_i \cup \{x\}$.
- Ist $\kappa_i \notin \alpha_0$, so ist $\kappa_i = \lambda_j$ für ein $j \in \{1, \ldots, i-1\}$. In diesem Fall setzen wir $\kappa'_i \;=\; \lambda'_j$.

In analoger Weise definieren wir die Klausel τ'_i. Die Klausel λ'_i ist als Resolvent von κ'_i und τ'_i definiert. Ist etwa

$$\lambda_i \;=\; \kappa_i \setminus \{y\} \;\cup\; \tau_i \setminus \{\neg y\}$$

mit $y \in \kappa_i$ und $\neg y \in \tau_i$, so setzen wir

$$\lambda'_i \;=\; \kappa'_i \setminus \{y\} \;\cup\; \tau'_i \setminus \{\neg y\}.$$

Es ist nun leicht zu sehen, dass $\langle \kappa'_1, \tau'_1, \lambda'_1\rangle \;\ldots,\; \langle \kappa'_m, \tau'_m, \lambda'_m\rangle$ eine Herleitung aus α ist und dass

$$\lambda'_i \;=\; \lambda_i \;\text{ oder }\; \lambda'_i \;=\; \lambda_i \cup \{x\}, \;\; i = 1, \ldots, m.$$

Mit $i = m$ folgt, dass λ'_m entweder die leere Klausel ist oder die Einheitsklausel $\{x\}$. Eine analoge Argumentation zeigt, dass

$$\{\sqcup, \{\neg x\}\} \cap \mathit{Res}^*(\alpha) \neq \varnothing. \qquad (**)$$

Da die leere Klausel $\sqcup$ ein Resolvent der Einheitsklauseln $\{x\}$ und $\{\neg x\}$ ist, folgt aus (*) und (**) die Herleitbarkeit der leeren Klausel. Also $\sqcup \in \mathit{Res}^*(\alpha)$. □

Beispiel 4.54 (Lifting der Herleitung). Wir illustrieren die im Beweis des Resolutionssatzes beschriebene Technik zum Liften von Herleitungen aus $\alpha[x/b]$ zu Herleitungen aus α an zwei einfachen Beispielen. Sei

$$\alpha \;=\; \underbrace{(x \vee \neg y \vee z)}_{=\kappa_1'} \wedge \underbrace{(y \vee z)}_{=\tau_1'=\tau_1} \wedge (\neg x \vee z) \wedge \underbrace{\neg z}_{=\kappa_2'=\kappa_2}$$

Dann ist $\alpha_0 \;=\; \alpha[x/0] \;=\; \underbrace{(\neg y \vee z)}_{=\kappa_1} \wedge \underbrace{(y \vee z)}_{=\tau_1} \wedge \underbrace{\neg z}_{=\kappa_2}$. Z.B. ist

$$\begin{aligned} \langle \kappa_1, \tau_1, \lambda_1 \rangle &= \big\langle \{\neg y, z\}, \{y, z\}, \{z\} \big\rangle \\ \langle \kappa_2, \tau_2, \lambda_2 \rangle &= \big\langle \{\neg z\}, \underbrace{\{z\}}_{=\lambda_1}, \sqcup \big\rangle \end{aligned}$$

eine Widerlegung für α_0. Diese wird zu einer Herleitung der Einheitsklausel $\{x\}$ aus α geliftet:

$$\begin{aligned} \langle \kappa_1', \tau_1', \lambda_1' \rangle &= \big\langle \underbrace{\{x, \neg y, z\}}_{=\kappa_1 \cup \{x\}}, \underbrace{\{y, z\}}_{=\tau_1 \in \alpha}, \underbrace{\{x, z\}}_{=\lambda_1 \cup \{x\}} \big\rangle \\ \langle \kappa_2', \tau_2', \lambda_2' \rangle &= \big\langle \underbrace{\{\neg z\}}_{=\kappa_2 \in \alpha}, \underbrace{\{x, z\}}_{=\lambda_1'}, \underbrace{\{x\}}_{=\lambda_2 \cup \{x\}} \big\rangle. \end{aligned}$$

Ebenso kann für $\alpha_1 \;=\; \alpha[x/1] \;=\; (y \vee z) \wedge z \wedge \neg z$ verfahren werden. Für α_1 haben wir eine Widerlegung der Länge 1 bestehend aus dem Tripel $\langle \{z\}, \{\neg z\}, \sqcup \rangle$. Wiedereinfügen des gestrichenen Literals $\neg x$ in die erste Elternklausel liefert die Herleitung

$$\big\langle \{\neg x, z\}, \{\neg z\}, \{\neg x\} \big\rangle$$

der Einheitsklausel $\{\neg x\}$ aus α. Wir setzen nun die beiden Herleitungen für $\{x\}$ und $\{\neg x\}$ aus α zu einer Widerlegung für α zusammen:

$$\big\langle \{x, \neg y, z\}, \{y, z\}, \{x, z\} \big\rangle \;\; \big\langle \{\neg z\}, \{x, z\}, \{x\} \big\rangle \;\; \big\langle \{\neg x, z\}, \{\neg z\}, \{\neg x\} \big\rangle \;\; \big\langle \{x\}, \{\neg x\}, \sqcup \big\rangle$$

Man beachte, dass es möglich ist, durch das Liften von Widerlegungen für $\alpha[x/b]$ sofort eine Widerlegung für α zu erhalten. Z.B. trifft dies auf die Formel

$$\alpha \;=\; (\neg x \vee \neg y) \wedge (y \vee z) \wedge (\neg y \vee z) \wedge \neg z$$

zu. Für diese besteht

$$\alpha_0 \;=\; \alpha[x/0] \;=\; (y \vee z) \wedge (\neg y \vee z) \wedge \neg z$$

nur aus Klauseln, die bereits in α enthalten sind. Liften der Widerlegung

$$\big\langle \{y, z\}, \{\neg z\}, \{y\} \big\rangle \;\; \big\langle \{\neg y, z\}, \{\neg z\}, \{\neg y\} \big\rangle \;\; \big\langle \{y\}, \{\neg y\}, \sqcup \big\rangle$$

für α_0 zu einer Herleitung für α bringt keinerlei Veränderungen, da die Elternklauseln der ersten beiden Resolutionsschritte in α liegen. ■

Resolutionsalgorithmus (Erfüllbarkeitstest). In Algorithmus 9 (Seite 167) ist ein naives Verfahren für den Nachweis der (Un-)Erfüllbarkeit einer KNF-Formel angegeben, das auf unseren bisherigen Ergebnissen beruht. Die Terminierung des Verfahrens folgt aus Lemma 4.43 (Seite 161); die Korrektheit der Antworten aus dem Resolutionssatz (Satz 4.53 Seite 164).

Algorithmus 9 Resolutionsalgorithmus

```
(*  Eingabe:   KNF-Formel α            *)
(*  Aufgabe:   prüft, ob α erfüllbar ist  *)

α_0 = α;
i := 0;                          (* Berechne die Mengen α_i = Res^i(α), i = 1,2,3,... *)
REPEAT
    i := i + 1;
    α_i := α_{i-1} ∪ Res(α_{i-1})
UNTIL ⊔ ∈ α_i  oder  α_i = α_{i-1};
IF ⊔ ∈ α_i
      THEN  return "nein, α ist unerfüllbar"
      ELSE  return "ja, α ist erfüllbar"
FI
```

Vollständigkeit versus Widerlegungsvollständigkeit. Die Vollständigkeitsaussage des Resolutionskalküls in Satz 4.53 bezieht sich lediglich auf die Herleitbarkeit der leeren Klausel aus unerfüllbaren Klauselmengen. Es gilt jedoch *nicht*, dass alle logisch folgerbaren Klauseln herleitbar sein müssen. Die Umkehrung des zweiten Teils des Resolventenlemmas gilt also *nicht* für beliebige Klauseln. In diesem Sinn ist der Resolutionskalkül *unvollständig*. Ein Beispiel, das die Unvollständigkeit belegt, ist die unerfüllbare KNF-Formel $\alpha = y \wedge \neg y \wedge x$. Die Einheitsklausel $\{\neg x\}$ ist zwar eine logische Folgerung aus α, jedoch ist $\{\neg x\}$ nicht aus α durch Resolution herleitbar. Dennoch gilt die Vollständigkeit der Resolution als Kalkül für logische Folgerbarkeit in folgendem Sinn:

Satz 4.55 (Resolutionsherleitbarkeit von Klauseln). *Für jede nicht-tautologische KNF-Formel α und jede nicht-tautologische Klausel λ gilt:*

$$\alpha \models\vdash \lambda \quad \textit{gdw} \quad \lambda' \in \textit{Res}^*(\alpha) \textit{ für eine Klausel } \lambda' \textit{ mit } \lambda' \subseteq \lambda \qquad \text{(ohne Beweis)}$$

Mit $\lambda = \sqcup$ ergibt sich erneut die Widerlegungskorrektheit und -vollständigkeit, die im Resolutionssatz nachgewiesen wurde. Eine weitere Konsequenz aus Satz 4.55 ist die Vollständigkeit der Resolution als Kalkül für die Folgerbarkeit von Einheitsklauseln aus erfüllbaren KNF-Formeln. Ist nämlich α eine erfüllbare KNF-Formel und L ein Literal, so gilt aufgrund der Aussage von Satz 4.55:

$$\alpha \models\vdash L \quad \text{gdw} \quad L \in \textit{Res}^*(\alpha).$$

Resolutionsalgorithmus für logische Folgerungen. Soll anstelle der Erfüllbarkeit von α geprüft werden, ob eine Klausel λ logisch aus α folgt (also ob $\alpha \models \lambda$), so ist das Schleifenabbruchkriterium im Resolutionsalgorithmus (Algorithmus 9) wie folgt zu ändern:

"**UNTIL** es gibt eine Klausel $\lambda' \in \alpha_i$ mit $\lambda' \subseteq \lambda$ oder $\alpha_i = \alpha_{i-1}$;"

Alternativ kann man den Resolutionsalgorithmus unverändert lassen und ihn mit der Klauselmenge bestehend aus den Klauseln von α und den Einheitsklauseln $\overline{L_1}, \ldots, \overline{L_k}$ starten, wobei $L_1, \ldots, L_k$ die Literale von λ sind. Wir erinnern daran, dass $\alpha \models \lambda$ genau dann gilt, wenn $\alpha \wedge \neg\lambda$ unerfüllbar ist. Ist nun $\lambda = L_1 \vee \ldots \vee L_k$, so ist

$$\begin{aligned} \alpha \wedge \neg\lambda &= \alpha \wedge \neg(L_1 \vee \ldots \vee L_k) \\ &\equiv \alpha \wedge \neg L_1 \wedge \ldots \wedge \neg L_k \\ &\equiv \alpha \wedge \overline{L_1} \wedge \ldots \wedge \overline{L_k} \end{aligned}$$

Die Formel $\alpha \wedge \neg\lambda$ kann daher als KNF-Formel mit der induzierten Klauselmenge

$$\cup \left\{ \{\overline{L_1}\}, \ldots, \{\overline{L_k}\} \right\}$$

aufgefasst werden. Diese Vorgehensweise hat den Nachteil, dass man die Anzahl an Klauseln und damit die Anzahl an potentiellen Resolventen erhöht.

Die einfache Formulierung des Resolutionsalgorithmus bildet alle möglichen Resolventen und ist daher hoffnungslos ineffizient, da es im Allgemeinen sehr viele Resolventen gibt. Falls n Aussagensymbole in der Eingabeformel α vorkommen, so wächst die maximale Anzahl an Schleifendurchläufen des Resolutionsalgorithmus exponentiell in n. Ein Beispiel für Formeln, in denen der Resolutionsalgorithmus tatsächlich exponentiell viele Schleifendurchläufe benötigt, sind die *Pigeonhole-Formeln*, die bereits in Beispiel 4.4 auf Seite 115 betrachtet wurden. Diese beschreiben das unlösbare Problem, n Tauben auf $n-1$ Löcher zu verteilen, so dass jede Taube in einem Loch sitzt, aber keine zwei Tauben sich in demselben Loch befinden:

$$\alpha_n = \bigwedge_{1 \leqslant i < n} \bigwedge_{1 \leqslant j < k \leqslant n} \underbrace{(\neg x_{i,j} \vee \neg x_{i,k})}_{\substack{\text{Taube } j \text{ oder Taube } k \\ \text{ist nicht in Loch } i}} \wedge \bigwedge_{1 \leqslant j \leqslant n} \underbrace{(x_{1,j} \vee \ldots \vee x_{n-1,j})}_{\substack{\text{Taube } j \text{ sitzt} \\ \text{in einem Loch}}}$$

Man kann zeigen, dass es eine Konstante $c > 1$ gibt, so dass die kürzeste Resolutionswiderlegung für α_n mindestens c^n Resolutionsschritte benötigt. Wir verzichten auf den Nachweis dieser Aussage.

Resolutionsalgorithmus für 2KNF. Trotz des schlechten worst-case-Verhaltens gibt es mehrere Teilklassen von KNF-Formeln, für die der Resolutionsalgorithmus (selbst in der naiven Version) recht effizient ist. Dazu zählt die Klasse der 2KNF-Formeln (d.h., KNF-Formeln mit höchstens zwei Literalen pro Klausel), für die nachgewiesen werden kann, dass selbst im schlimmsten Fall nur quadratisch viele Iterationen ausgeführt werden. Beachte, dass erstens Resolventen von 2KNF-Formeln stets höchstens zwei Literale haben und dass zweitens die Anzahl an Klauseln mit maximal zwei Literalen über einer n-elementigen Atommenge $\{x_1, \ldots, x_n\}$ durch

$$2n \cdot (2n-1) + 2n + 1 = \mathcal{O}\left(n^2\right)$$

beschränkt ist. Diese Schranke ergibt sich wie folgt: der Summand 1 steht für die leere Klausel, der Summand $2n$ für die Anzahl an einelementigen Klauseln, und der Summand $2n * (2n-1)$ steht für die Anzahl an zweielementigen Klauseln.

N-Resolution und andere Resolutionsstrategien

In vielen Fällen ist es nicht so dramatisch wie im Falle der Pigeonhole-Formeln, da oftmals "relativ kurze" Widerlegungen unerfüllbarer Formeln existieren, obwohl es "sehr viele" Resolventen gibt. Für praktische Anwendungen der Resolutionsidee sind daher Strategien von großer Bedeutung, welche die Anzahl an relevanten Resolventen einschränken und gezielt nach Resolventen suchen, die tatsächlich zu einer Widerlegung beitragen können.

N- und P-Resolution zählen zu den Resolutionsstrategien, welche die Anzahl an relevanten Resolventen einschränken, die für den Nachweis der Unerfüllbarkeit einer Klauselmenge benötigt werden. Wir beginnen mit Erläuterungen der N-Resolution, in der nur solche Resolutionsschritte zugelassen werden, in denen eine der Elternklauseln aus lauter negativen Literalen besteht. Nicht-leere Klauseln, welche nur aus negativen Literalen bestehen, werden auch *negative Klauseln* genannt.

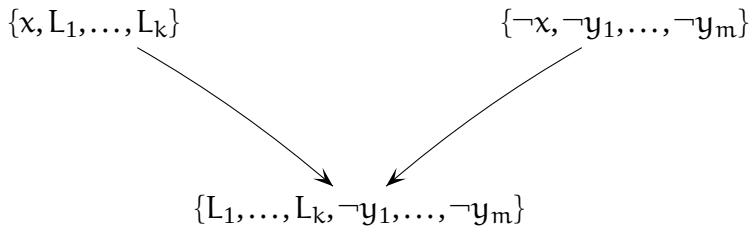


Abbildung 40: Schema der N-Resolution

Zunächst machen wir uns intuitiv klar, warum die Beschränkung auf derartige N-Resolventen sinnvoll ist. Der letzte Resolutionsschritt einer Widerlegung (der Länge $\geqslant 1$) resolviert stets zwei zueinander komplementäre Einheitsklauseln $\{x\}$ und $\{\neg x\}$. Insbesondere besteht eine der Elternklauseln, nämlich $\{\neg x\}$, nur aus negativen Literalen. Die Klausel $\{\neg x\}$ kann nur durch Resolution

- *entweder* einer Klausel der Form $\{\neg y, \neg x\}$ und einer der beiden Klauseln $\{y\}$ oder $\{y, \neg x\}$
- *oder* einer negativen Einheitsklausel $\{\neg y\}$ und der Klausel $\{y, \neg x\}$

entstanden sein. In jedem Fall ist eine negative Klausel beteiligt. Die positive Einheitsklausel $\{x\}$ kann zwar zunächst durch die Resolvierung zweier nicht-negativer Klauseln, etwa $\{\neg z, x\}$ und entweder $\{z, x\}$, entstanden sein; jedoch kann man durch eine vorangestellte Resolvierung mit der negativen Einheitsklausel $\{\neg x\}$ die Elternklausel $\{\neg z, x\}$ zu $\{\neg z\}$ verkürzen.

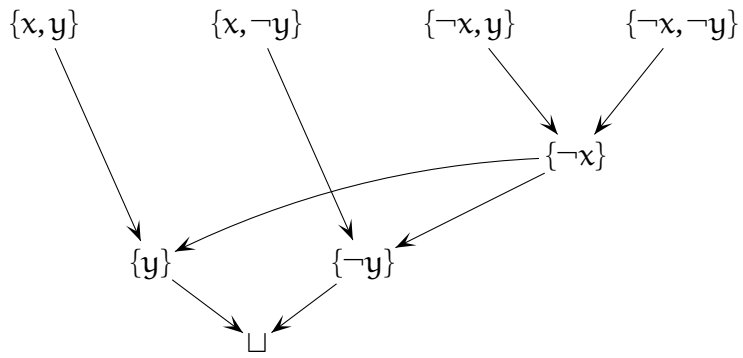
Diese Überlegungen motivieren die sogenannte *N-Resolution*, bei der ausschließlich solche Resolventen zugelassen sind, welche aus einer beliebigen Klausel und einer negativen Klausel gebildet werden können. Das allgemeine Schema ist in Abbildung 40 angegeben, wobei $L_1, \ldots, L_k$ beliebige Literale mit $x \notin \{L_1, \ldots, L_k\}$ und $y_1, \ldots, y_m$ Aussagensymbole mit $x \notin \{y_1, \ldots, y_m\}$ sind.

Definition 4.56 (N-Herleitung). Eine N-Herleitung von λ aus einer KNF-Formel α bezeichnet eine Herleitung

$$\langle \kappa_1, \tau_1, \lambda_1 \rangle \langle \kappa_2, \tau_2, \lambda_2 \rangle \ldots \langle \kappa_m, \tau_m, \lambda_m \rangle$$

von λ aus α, so dass $\kappa_1, \ldots, \kappa_m$ negative Klauseln sind. ■

Andere Begriffe und Bezeichnungen, die für die gewöhnliche Resolution eingeführt wurden, sind analog definiert. Der N-Resolutionsabschluss von α besteht aus allen Klauseln, für die es eine N-Herleitung aus α gibt. Er wird mit $NRes^*(\alpha)$ bezeichnet. Eine N-Widerlegung für α bezeichnet eine N-Herleitung der leeren Klausel aus α.



Die Abbildung oben zeigt eine N-Widerlegung für die unerfüllbare KNF-Formel

$$(x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y).$$

Tatsächlich schränkt die N-Resolution oftmals die Anzahl an möglichen Resolventen erheblich ein. Liegt z.B. die KNF-Formel

$$\alpha \;=\; (x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee \neg w \vee \neg y) \wedge (x \vee w \vee \neg z) \wedge (\neg w \vee \neg z)$$

vor, so können unter der gewöhnlichen Resolution sieben Resolventen gebildet werden. Die erste Klausel kann mit der zweiten (auf zwei Arten), vierten und fünften Klausel resolviert werden. Weiter können Resolventen aus der zweiten und dritten Klausel, aus der dritten und vierten und aus der vierten und fünften Klausel generiert werden. Man erhält:

$$\begin{aligned} Res^1(\alpha) \;&=\; \alpha \cup \\ &\big\{ \{x, z, \neg z\},\ \{x, y, \neg y\},\ \{x, \neg y, w\},\ \{x, \neg y, \neg w\},\ \{x, \neg z, \neg w\}\ \{x, \neg y, \neg z\},\ \{x, \neg z\} \big\} \end{aligned}$$

Nun können weitere Resolventen, z.B. aus $\{x, z, \neg z\}$ und der ersten, zweiten, dritten und vierten Klausel von α sowie aus $\{x, z, \neg z\}$ und $\{x, \neg z, \neg w\}$, gebildet werden. Mit der N-Resolution ist dagegen nur die Resolventenbildung zwischen der ersten und fünften Klausel sowie der vierten und der fünften zulässig, da nur die fünfte Klausel $\neg w \vee \neg z$ aus lauter negativen Literalen besteht und diese nur mit der ersten und vierten Klausel resolvierbar ist. Man erhält die beiden N-Resolventen $\{x, \neg y, \neg w\}$ und $\{x, \neg z\}$. Weitere N-Resolventen können nicht gebildet werden, da diese beiden Klauseln das positive Literal x enthalten. Aus der Widerlegungsvollständigkeit der N-Resolution, (siehe Satz 4.57 unten) ergibt sich damit die Erfüllbarkeit von α mit der Bildung von nur zwei N-Resolventen.

Die Korrektheit der N-Resolution als Kalkül für den Nachweis der Unerfüllbarkeit ist klar, da aus der Existenz einer N-Widerlegung die Aussage $\square \in Res^*(\alpha)$ folgt. Der Resolutionssatz impliziert daher die Unerfüllbarkeit von α. Interessanterweise ist die N-Resolution auch vollständig für den Nachweis der Unerfüllbarkeit. Wir zitieren das Ergebnis ohne Beweis:

Satz 4.57 (Korrektheit und Widerlegungsvollständigkeit der N-Resolution). *Sei α eine KNF-Formel. Dann gilt: α ist genau dann unerfüllbar, wenn es eine N-Widerlegung für α gibt. (ohne Beweis)*

Bemerkung 4.58. Die Widerlegungsvollständigkeit der N-Resolution belegt nochmals die Erfüllbarkeit von KNF-Formeln, in denen jede Klausel wenigstens ein positives Literal besitzt. Siehe Teil (a) von Lemma 4.23 auf Seite 142. Aus den Klauseln solcher KNF-Formeln können keine N-Resolventen gebildet werden. Daher ist eine N-Widerlegung ausgeschlossen. ■

Der Nachteil der N-Resolution ist, dass N-Widerlegungen oftmals länger als kürzeste Widerlegungen sind. Wir erwähnen ohne Beweis, dass es unerfüllbare KNF-Formeln α_n gibt, deren kürzeste Widerlegungen polynomielle Länge haben, während alle N-Widerlegungen mindestens exponentiell lang sind.

P-Resolution. In analoger Weise ist die P-Resolution definiert, bei der nur Resolventen aus einer beliebigen Klausel und einer Klausel mit nur positiven Literalen gebildet werden dürfen. Die Begriffe P-Herleitung und P-Widerlegung sind wie für die N-Resolution definiert. Die Korrektheit und Widerlegungsvollständigkeit der P-Resolution kann wie für die N-Resolution bewiesen werden. Gegebene KNF-Formel α ist also genau dann unerfüllbar, wenn es eine P-Widerlegung für α gibt. Ein Beispiel für eine P-Widerlegung für die unerfüllbare KNF-Formel $(x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y)$ ist in Abbildung 41 angegeben.

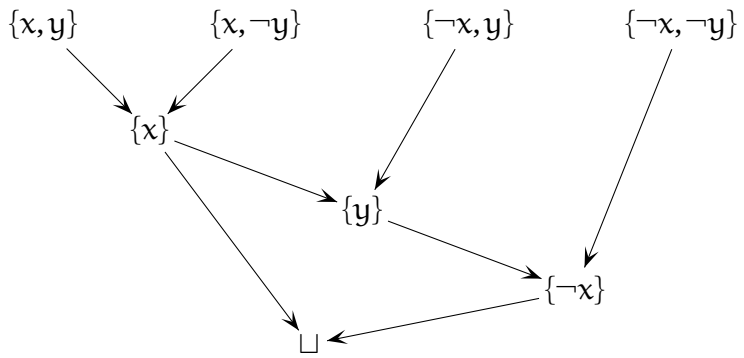


Abbildung 41: Beispiel für eine P-Widerlegung

Lineare Resolution. Die N-Resolution schränkt zwar die Anzahl an Resolventen ein, so dass unter Einsatz der N-Strategie oftmals eine deutliche Verbesserung im Resolutions-Algorithmus zu verzeichnen ist; jedoch können die zu N-Widerlegungen gehörenden Graphen sehr komplexe Struktur haben. Eine gezielte Suche nach Widerlegungen ist unter der Einschränkung der N-Resolution nicht wirklich gewährleistet. Analoges gilt für die P-Resolution.

Lineare Resolution löst sich von dem Schema des Resolutions-Algorithmus und sucht stattdessen gezielt nach Herleitungen der leeren Klausel. Die Idee besteht darin, eine "lineare" Folge von Klauseln $\lambda_1, \ldots, \lambda_m$ zu generieren, so dass die i-te Klausel λ_i ein Resolvent von λ_{i-1} und einer Klausel τ_i in $\alpha \cup \{\lambda_1, \ldots, \lambda_{i-2}\}$ ist. τ_i wird in diesem Kontext auch *Seitenklausel* genannt. Sei λ eine Klausel und α eine KNF-Formel. Eine Herleitung von λ aus α wird *linear* genannt, wenn sie die Gestalt

$$\langle \lambda_0, \tau_1, \lambda_1 \rangle \ \langle \lambda_1, \tau_2, \lambda_2 \rangle \ \ldots \ \langle \lambda_{m-1}, \tau_m, \lambda_m \rangle, \quad \text{wobei } \lambda_m = \lambda$$

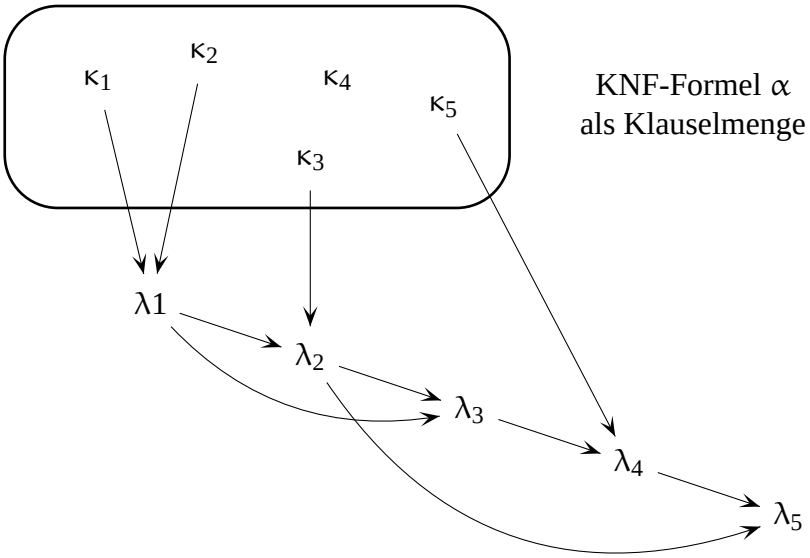


Abbildung 42: Schema einer linearen Resolution

hat. Eine lineare Widerlegung für α bezeichnet eine lineare Herleitung der leeren Klausel $\sqcup$ aus α. Die in Abbildung 41 auf Seite 171 angegebene P-Widerlegung ist in der Tat zugleich eine lineare Widerlegung. Lineare Widerlegungen sind zwar leichter lesbar, jedoch sind sie oftmals wesentlich länger (im schlimmsten Fall exponentiell länger) als kürzeste gewöhnliche Widerlegungen. Wir erwähnen ohne Beweis, dass jede Widerlegung im Wesentlichen durch Vertauschen von Resolutionsschritten linearisiert werden kann, woraus sich die Widerlegungsvollständigkeit der linearen Resolution ergibt. Die Korrektheit ist aufgrund des Resolutionssatzes klar.

Input-Resolution. Ein Spezialfall linearer Resolution ist die sogenannte Input-Resolution, in der in jedem Resolutionsschritt eine der Klauseln κ der zugrundegelegten KNF-Formel α als Elternklausel eingesetzt werden muss. Eine Input-Herleitung aus einer KNF-Formel α ist eine lineare Herleitung

$$\langle \kappa_0, \kappa_1, \lambda_1 \rangle \ \langle \lambda_1, \kappa_2, \lambda_2 \rangle \ \ldots \ \langle \lambda_{m-1}, \kappa_m, \lambda_m \rangle$$

aus α, so dass $\kappa_0, \kappa_1, \ldots, \kappa_m$ Klauseln von α sind. Input-Widerlegungen sind Input-Herleitungen der leeren Klausel. Am Beispiel der unerfüllbaren KNF-Formel α mit der induzierten Klauselmenge

$$\big\{\{x, y\}, \{\neg x, y\}, \{x, \neg y\}, \{\neg x, \neg y\}\big\}$$

kann man sich klarmachen, dass die Input-Resolution *nicht widerlegungsvollständig* ist, da in jedem Resolutionsschritt, in dem eine Klausel λ mit einer Klausel κ in α resolviert wird, eine Klausel mit mindestens einem Literal entsteht. Man beachte, dass alle Klauseln in α zweielementig sind. Resolvierung mit einer Klausel κ in α als Elternklausel liefert daher eine Klausel, die wenigstens eines der Literale von κ enthält. Die leere Klausel kann daher nicht durch Input-Herleitungen aus α generiert werden. Dennoch spielt die Input-Resolution eine wichtige Rolle für die Logikprogrammierung, da sie – wie wir im nächsten Abschnitt sehen werden – für die Klasse der Hornformeln widerlegungsvollständig ist.

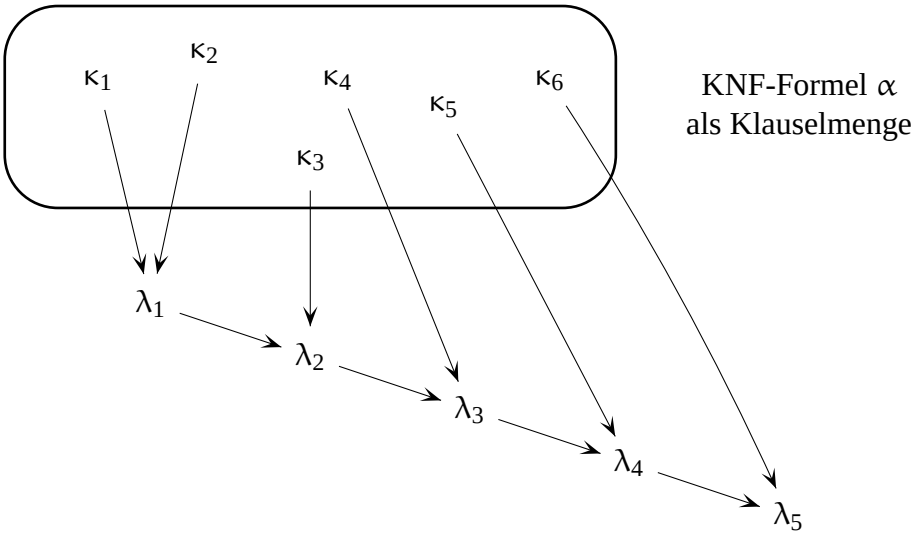


Abbildung 43: Schema einer Input-Resolution

Resolution für Hornformeln. Im Kontext der Logikprogrammierung spielt eine spezielle Resolutionsstrategie für Hornformeln eine wichtige Rolle, die auf einer Variante der linearen Resolution beruht. Hierauf gehen wir nicht ein, sondern beenden diesen Abschnitt mit ein paar einfachen Bemerkungen zur Resolution für Hornformeln.

Lemma 4.59. *Der Resolvent zweier Hornklauseln ist wieder eine Hornklausel.*

Beweis. Seien κ und τ zwei Hornklauseln und λ ein Resolvent von κ und τ; etwa

$$\lambda \;=\; \kappa \setminus \{x\} \;\cup\; \tau \setminus \{\neg x\}$$

wobei $x \in \kappa$ und $\neg x \in \tau$. Die Elternklausel τ enthält höchstens ein positives Literal, während κ außer x kein weiteres positives Literal enthält. Daher enthält auch λ höchstens ein positives Literal. □

Z.B. haben die Klauseln $x \wedge y \to z \;\equiv\; \neg x \vee \neg y \vee z$ und $z \wedge w \to \mathit{false} \;\equiv\; \neg z \vee \neg w$ den Resolventen

$$x \wedge y \wedge w \to \mathit{false} \;\equiv\; \neg x \vee \neg y \vee \neg w.$$

Die Resolventenbildung für Hornklauseln ist sehr eingeschränkt. Je zwei Zielklauseln (also Hornklauseln, die kein positives Literal enthalten) haben keine Resolventen. Ebenso wenig können Resolventen aus zwei Fakten (positiven Einheitsklauseln) gebildet werden. Zwei Regeln haben nur dann einen Resolventen, wenn der Kopf der einen Regel im Rumpf der anderen Regel vorkommt. Z.B. haben

$$x \wedge y \to v \;\equiv\; \neg x \vee \neg y \vee v \quad \text{und} \quad x \wedge z \to w \;\equiv\; \neg x \vee \neg z \vee w$$

keinen Resolventen.

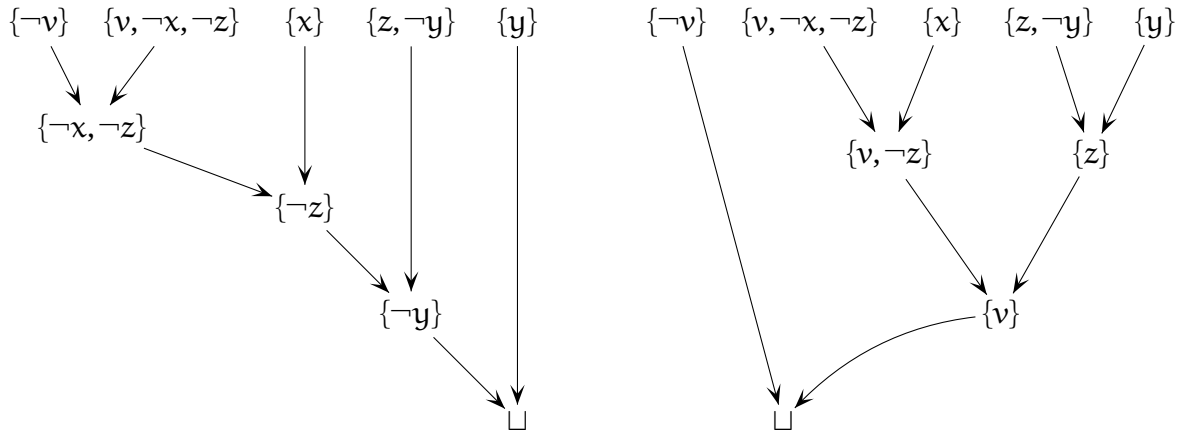
Resolutionsstrategien für Hornformeln. Da die N-Resolution für alle Klauselmengen widerlegungsvollständig ist, trifft dies natürlich auch auf Hornformeln zu. Für unerfüllbare Hornformeln kann daher stets eine Widerlegung angegeben werden, so dass in jedem Resolutionsschritt eine Zielklausel als Elternklausel eingesetzt wird. Resolventenbildungen zwischen zwei definiten Programmklauseln (Fakten oder Regeln) müssen also nicht betrachtet werden.

Aus der Widerlegungsvollständigleit der P-Resolution ergibt sich, dass unerfüllbare Hornformeln stets eine Widerlegung haben, in der in jedem Resolutionsschritt ein Faktum als Elternklausel eingesetzt wird.

Als Beispiel betrachten wir die unerfüllbare Hornformel

$$\begin{aligned} \alpha &= (x \wedge z \to v) \wedge (y \to z) \wedge y \wedge \neg v \wedge x \\ &\equiv (\neg x \vee \neg z \vee v) \wedge (\neg y \vee z) \wedge y \wedge \neg v \wedge x \end{aligned}$$

Das Bild unten zeigt links eine N-Widerlegung und rechs eine P-Widerlegung für α. In diesem Beispiel haben die N- und P-Widerlegung für α also völlig unterschiedliche Struktur.



Ist eine der Elternklauseln eine Zielklausel und die andere Elternklausel entweder eine Regel oder ein Faktum, so enthält der Resolvent kein positives Literal. Daher sind N-Resolventen, die aus den Klauseln einer Hornformel gebildet werden, stets Zielklauseln. Jede N-Herleitung induziert daher eine Input-Herleitung.[18] Aus der Widerlegungsvollständigkeit der N-Resolution folgt daher die Widerlegungsvollständigkeit der Input-Resolution für Hornformeln.

The End

[18] Zunächst müssen N-Herleitungen nicht linear sein. Liegt eine N-Herleitung aus einer Hornformel vor, so kann diese durch Streichen von Resolutionsschritten, die für die hergeleitete Hornklausel irrelevant sind, in eine lineare N-Herleitung mit derselben hergeleiteten Klausel überführt werden. Diese ist dann zugleich eine Input-Herleitung.

Index