

Vorlesung Formale Sprachen und Automaten

Markus Lohrey

Universität Siegen

Sommersemester 2024

Unter

https://www.eti.uni-siegen.de/ti/lehre/sommer_2024/fsa/ gibt es

- aktuelle Versionen der Folien,
- Übungsblätter,
- aktuelle Informationen, etc.

Literaturempfehlungen:

- Uwe Schöning, Theoretische Informatik – kurz gefasst, Spektrum Akademischer Verlag (5. Auflage): Die Vorlesung folgt inhaltlich sehr eng diesem Buch.
- Lutz Priese, Katrin Erk, Theoretische Informatik: Eine umfassende Einführung. Springer: Ist elektronisch über die Universitätsbibliothek verfügbar.
- Alexander Asteroth, Christel Baier, Theoretische Informatik, Pearson Studium: Dieses Buch ist vom Aufbau etwas anders strukturiert als die Vorlesung, stellt aber dennoch eine sehr gute Ergänzung dar.

Mengentheoretische Grundlagen (Wiederholung aus DMI)

Naive Definition (Mengen, Elemente, \in , \notin)

Eine **Menge** ist die Zusammenfassung von bestimmten unterschiedlichen Objekten (die **Elemente der Menge**) zu einem neuen Ganzen.

Wir schreiben $x \in M$, falls das Objekt x zur Menge M gehört.

Wir schreiben $x \notin M$, falls das Objekt x nicht zur Menge M gehört.

Eine Menge, welche nur aus endlich vielen Objekten besteht (eine endliche Menge), kann durch explizite Auflistung dieser Elemente spezifiziert werden.

Beispiel: $M = \{2, 3, 5, 7\}$.

Hierbei spielt die Reihenfolge der Auflistung keine Rolle:

$$\{2, 3, 5, 7\} = \{7, 5, 3, 2\}.$$

Auch Mehrfachauflistungen spielen keine Rolle:

$$\{2, 3, 5, 7\} = \{2, 2, 2, 3, 3, 5, 7\}.$$

Eine besonders wichtige Menge ist die **leere Menge** $\emptyset = \{\}$, die keinerlei Elemente enthält.

In der Mathematik hat man es häufig auch mit unendlichen Mengen zu tun (Mengen, die aus unendlich vielen Objekten bestehen).

Solche Mengen können durch Angabe einer Eigenschaft, welche die Elemente der Menge auszeichnet, spezifiziert werden.

Beispiele:

- $\mathbb{N} = \{0, 1, 2, 3, 4, 5, \dots\}$ (Menge der natürlichen Zahlen)
- $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ (Menge der ganzen Zahlen)
- $P = \{n \in \mathbb{N} \mid n \geq 2, n \text{ ist nur durch } 1 \text{ und } n \text{ teilbar}\}$
(Menge der Primzahlen)

Definition (\subseteq , Potenzmenge, \cap , \cup , \setminus , disjunkt)

Seien A und B zwei Mengen.

- $A \subseteq B$ bedeutet, dass jedes Element von A auch zu B gehört (A ist eine **Teilmenge** von B); formal:

$$\forall a : a \in A \rightarrow a \in B$$

- $2^A = \{B \mid B \subseteq A\}$ (**Potenzmenge von A**)
- $A \cap B = \{c \mid c \in A \text{ und } c \in B\}$ (**Schnitt von A und B**)
- $A \cup B = \{c \mid c \in A \text{ oder } c \in B\}$ (**Vereinigung von A und B**)
- $A \setminus B = \{c \in A \mid c \notin B\}$ (**Differenz von A und B**)
- Zwei Mengen A und B sind **disjunkt**, falls $A \cap B = \emptyset$ gilt.

Definition (beliebige Vereinigung und Schnitt)

Sei I eine Menge und für jedes $i \in I$ sei A_i wiederum eine Menge. Dann definieren wir:

$$\bigcup_{i \in I} A_i = \{a \mid \exists j \in I : a \in A_j\}$$

$$\bigcap_{i \in I} A_i = \{a \mid \forall j \in I : a \in A_j\}$$

Beispiele:

$$\bigcup_{a \in A} \{a\} = A \text{ für jede Menge } A$$

$$\bigcap_{n \in \mathbb{N}} \{m \in \mathbb{N} \mid m \geq n\} = \emptyset$$

Definition (Kartesisches Produkt)

Für zwei Mengen A und B ist

$$A \times B = \{(a, b) \mid a \in A \text{ und } b \in B\}$$

das **kartesische Produkt** von A und B (Menge aller Paare aus einem Element von A und einem Element von B).

Allgemeiner: Für Mengen A_1, \dots, A_n ($n \geq 2$) sei

$$\begin{aligned} \prod_{i=1}^n A_i &= A_1 \times A_2 \times \dots \times A_n \\ &= \{(a_1, \dots, a_n) \mid \text{für alle } 1 \leq i \leq n \text{ gilt } a_i \in A_i\} \end{aligned}$$

Falls $A_1 = A_2 = \dots = A_n = A$ schreiben wir auch A^n für diese Menge.

Beispiele und einige einfache Aussagen:

- $\{1, 2, 3\} \times \{4, 5\} = \{(1, 4), (1, 5), (2, 4), (2, 5), (3, 4), (3, 5)\}$
- Für alle Mengen A , B , und C gilt:

$$(A \cup B) \times C = (A \times C) \cup (B \times C)$$

$$A \times (B \cup C) = (A \times B) \cup (A \times C)$$

$$(A \cap B) \times C = (A \times C) \cap (B \times C)$$

$$A \times (B \cap C) = (A \times B) \cap (A \times C)$$

Vollständige Induktion (Wiederholung aus DMI)

Um eine Aussage $P(n)$ für jede natürliche Zahl $n \in \mathbb{N}$ zu beweisen, genügt es, folgendes zu zeigen:

- 1 $P(0)$ gilt (Induktionsanfang).
- 2 Für jede natürliche Zahl $n \in \mathbb{N}$ gilt: Wenn $P(n)$ gilt, dann gilt auch $P(n+1)$ (Induktionsschritt).

Dieses Beweisprinzip nennt man das Prinzip der **vollständigen Induktion**.

Beispiel: Wir beweisen mittels vollständiger Induktion, dass für alle natürlichen Zahlen n gilt:

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}.$$

Vollständige Induktion (Wiederholung aus DMI)

Induktionsanfang: Es gilt $\sum_{i=1}^0 i = 0 = \frac{0 \cdot 1}{2}$.

Induktionsschritt: Angenommen es gilt

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}.$$

Dann gilt auch

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

Vollständige Induktion (Wiederholung aus DMI)

Für Induktionsanfang (Induktionsschritt) schreiben wir häufig kurz **IA** (**IS**).

Mittels des Prinzips der Induktion kann man auch Objekte definieren.

Angenommen, wir wollen für jede natürliche Zahl $n \in \mathbb{N}$ ein Objekt A_n definieren.

Dies kann man wie folgt machen:

- 1 Definiere A_0 .
- 2 Gib eine allgemeine Vorschrift an, wie das Objekt A_{n+1} aus den (bereits konstruierten) Objekten A_0, A_1, \dots, A_n konstruiert werden kann.

Den Inhalt von Folie 12–44 finden Sie im Buch von Schöning auf Seite 3–18.

Eine zentrale Datenstruktur in der Informatik sind endliche Symbolfolgen, auch bekannt als **Wörter** oder **Strings**.

Beispiele:

- 1 Ein Byte ist eine Folge von 8 Bits, z.B. 00110101
- 2 Ein deutscher oder englischer Text ist eine Folge bestehend aus den Symbolen $a, b, c, \dots, z, A, B, C, \dots, Z, 1, 2, \dots, 9, _$ (blank) und den Interpunktionszeichen $. , ! , ?$ sowie $,$
- 3 Ein Gen ist eine Folge der Symbole A, G, T, C (4 DNA-Basen)

Definition (Alphabet, Wörter)

Ein **Alphabet** ist eine endliche nicht-leere Menge.

Ein **Wort** über dem Alphabet Σ ist eine endliche Zeichenkette der Form $a_1 a_2 \cdots a_n$ mit $a_i \in \Sigma$ für $1 \leq i \leq n$. Die **Länge** dieses Worts ist n .

Für ein Wort w schreiben wir auch $|w|$ für die Länges des Wortes w .

Für $n = 0$ erhalten wir das **leere Wort** (das Wort der Länge 0), welches mit ε bezeichnet wird.

Mit Σ^* bezeichnen wir die Menge aller Wörter über dem Alphabet Σ .

Die Menge aller nicht-leeren Wörter ist $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.

Beispiel 1: Sei $\Sigma = \{a, b, c\}$. Dann sind mögliche Wörter aus Σ^* :

$$\varepsilon, a, b, aa, ab, bc, bbbab, \dots$$

Für die Längen gilt $|\varepsilon| = 0$, $|a| = |b| = 1$, $|aa| = |ab| = |bc| = 2$, and $|bbbab| = 5$.

Beispiel 2: Ein Genom ist ein Wort über dem Alphabet $\{A, G, T, C\}$.

Bemerkung: Häufig wird gefragt, wozu man das leere Wort ε braucht.

Das leere Wort wird sich in vielen Betrachtungen als nützlich erweisen. Man kann das leere Wort ε mit der Zahl $0 \in \mathbb{N}$ vergleichen. In der Tat hat es ähnliche Eigenschaften wie die Zahl 0.

Konventionen: Wörter aus Σ^* werden mit Kleinbuchstaben (aus der hinteren Hälfte des Alphabets) bezeichnet: u, v, w, x, y, z, \dots

Definition (Konkatenation von Wörtern)

Für Wörter $u = a_1 \cdots a_m$ und $v = b_1 \cdots b_n$ mit $a_1, \dots, a_m, b_1, \dots, b_n \in \Sigma$ ist das Wort

$$u \circ v = a_1 \cdots a_m b_1 \cdots b_n.$$

die **Konkatenation** (oder Hintereinanderschreibung) der Wörter u und v .

Anstatt $u \circ v$ schreiben wir meistens nur uv .

Offensichtlich gilt für alle Wörter $u, v, w \in \Sigma^*$:

- $(u \circ v) \circ w = u \circ (v \circ w)$ oder kurz $(uv)w = u(vw)$ (Assoziativgesetz)
- $\varepsilon \circ u = u = u \circ \varepsilon$

Wir schreiben für $(uv)w = u(vw)$ auch einfach uvw .

Erinnerung aus DMI: (Σ^*, \circ) ist also ein Monoid, man nennt es auch das **von Σ erzeugte freie Monoid**. Das leere Wort ε ist das neutrale Element.

Beachte: Für Wörter u und v gilt im Allgemeinen $uv \neq vu$.

Es gilt z.B. $ab \neq ba$ für $a, b \in \Sigma$ mit $a \neq b$.

Konkatenation von Wörtern ist **nicht kommutativ**.

Angenommen Σ ist ein Alphabet mit n Symbolen: $|\Sigma| = n$.

Dann gibt es genau n^k viele Wörter der Länge k über dem Alphabet Σ :

$$|\{w \in \Sigma^* \mid |w| = k\}| = n^k.$$

Begründung: Für das erste Symbol in einem Wort gibt es genau n Möglichkeiten, für das zweite Symbol gibt es ebenfalls n Möglichkeiten, u.s.w. Insgesamt gibt es also

$$\underbrace{n \cdot n \cdot n \cdots n}_{k \text{ viele}} = n^k$$

Möglichkeiten.

Für die Menge $\{w \in \Sigma^* \mid |w| = k\}$ (Menge aller Wörter der Länge k) schreiben wir auch Σ^k .

Sprachen

Im Kontext von natürlichen Sprachen (z.B. Deutsch oder Englisch) kann man eine Sprache als die Menge aller Wörter über dem Alphabet aus Beispiel 2, Folie 12, definieren, die einen korrekten Satz ergeben.

Z.B. wäre die Zeichenkette *Der_Hund_jagt_die_Katze.* ein Element der Sprache Deutsch.

Definition (Sprache)

Sei Σ ein Alphabet.

Eine (formale) **Sprache** L über dem Alphabet Σ ist eine beliebige Teilmenge von Σ^* , d.h. $L \subseteq \Sigma^*$.

Beispiel: Sei $\Sigma = \{ (,), +, -, *, /, a \}$. Dann können wir die Sprache *EXPR* der korrekt geklammerten Ausdrücke definieren. Es gilt beispielsweise:

- $(a - a) * a + a / (a + a) - a \in EXPR$
- $((((a)))) \in EXPR$
- $((a+) - a(\notin EXPR$

Grammatiken in der Informatik sind – ähnlich wie Grammatiken für natürliche Sprachen – ein Mittel, um alle syntaktisch korrekten Sätze (hier: Wörter) einer Sprache zu erzeugen.

Beispiel: Grammatik zur Erzeugung von Elementen aus *EXPR*:

$$E \rightarrow a$$

$$E \rightarrow E + E$$

$$E \rightarrow E - E$$

$$E \rightarrow E * E$$

$$E \rightarrow E / E$$

$$E \rightarrow (E)$$

Mit Hilfe dieser (endlichen) Grammatik ist es möglich, Elemente aus *EXPR* abzuleiten.

Beispiel:

$$E \rightarrow E * E \rightarrow (E) * E \rightarrow (E + E) * E \rightarrow (a + a) * a$$

Offensichtlich kann man mit der Grammatik unendlich viele Wörter erzeugen.

Das heißt, die zu der Grammatik gehörende Sprache (man sagt auch: die von der Grammatik erzeugte Sprache) ist unendlich.

Grammatiken (Definition)

Grammatiken besitzen Produktionen der Form

$$\textit{linke Seite} \rightarrow \textit{rechte Seite}$$

Sowohl auf der linken, als auch auf der rechten Seite können zwei Typen von Symbolen vorkommen:

- **Nicht-Terminale** (die **Variablen**, aus denen noch weitere Wortbestandteile abgeleitet werden sollen)
- **Terminale** (die “eentlichen” Symbole)

Im vorherigen Beispiel: auf der linken Seite befindet sich immer genau ein Nicht-Terminal; man spricht von einer kontextfreien Grammatik.

Es gibt aber auch allgemeinere Grammatiken.

Es gibt sogar Grammatiken, die auf Bäumen und Graphen statt auf Wörtern arbeiten. Diese werden in der Vorlesung jedoch nicht behandelt.

Grammatiken (Definition)

Definition (Grammatik, Satzform)

Eine **Grammatik** G ist ein 4-Tupel $G = (V, \Sigma, P, S)$, das folgende Bedingungen erfüllt:

- V ist ein **Alphabet** (Menge der **Nicht-Terminalen** oder **Variablen**).
- Σ ist ein **Alphabet** (Menge der **Terminal(symbol)e**) mit $V \cap \Sigma = \emptyset$, d.h., kein Zeichen ist gleichzeitig Terminal und Nicht-Terminal.
- $P \subseteq ((V \cup \Sigma)^+ \setminus \Sigma^*) \times (V \cup \Sigma)^*$ ist eine endliche Menge von **Produktionen** (**Produktionen**).
- $S \in V$ ist die **Startvariable** (**Axiom**).

Ein Wort aus $(V \cup \Sigma)^*$ nennt man auch eine **Satzform**.

Eine Produktion aus P ist also ein Paar (ℓ, r) von Wörtern über $V \cup \Sigma$, das zumeist als $\ell \rightarrow r$ geschrieben wird. Dabei gilt:

- Sowohl ℓ als auch r bestehen aus Variablen und Terminalsymbolen.
- ℓ darf nicht nur aus Terminalen bestehen. Eine Regel muss also immer zumindest ein Nicht-Terminal ersetzen.

Konventionen:

- Variablen (Elemente aus V) werden mit Großbuchstaben bezeichnet: $A, B, C, \dots, S, T, \dots$
- Terminalsymbole (Elemente aus Σ) werden mit Kleinbuchstaben dargestellt: a, b, c, \dots

Beispiel-Grammatik

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

- $V = \{S, B, C\}$
- $\Sigma = \{a, b, c\}$
- $P = \{S \rightarrow aSBC, S \rightarrow aBC, CB \rightarrow BC, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc\}$

Grammatiken (Ableitungen)

Wie werden die Produktionen eingesetzt, um Wörter aus der Startvariablen S zu erzeugen?

Definition (Ableitungsschritt)

Sei $G = (V, \Sigma, P, S)$ eine Grammatik und seien $u, v \in (V \cup \Sigma)^*$. Es gilt:

$u \Rightarrow_G v$ (u geht unter G unmittelbar über in v),

falls eine Produktion $(\ell \rightarrow r) \in P$ und Wörter $x, y \in (V \cup \Sigma)^*$ existieren mit

$$u = x\ell y \quad v = xry.$$

Man kann \Rightarrow_G als binäre Relation auf $(V \cup \Sigma)^*$, d.h. als Teilmenge von $(V \cup \Sigma)^* \times (V \cup \Sigma)^*$ auffassen:

$$\Rightarrow_G = \{(u, v) \mid \exists (\ell \rightarrow r) \in P \exists x, y \in (V \cup \Sigma)^* : u = x\ell y, v = xry\}$$

Grammatiken (Ableitungen)

Statt $u \Rightarrow_G v$ schreibt man auch $u \Rightarrow v$, wenn klar ist, um welche Grammatik es sich handelt.

Definition (Ableitung)

Eine Folge von Wörtern $w_0, w_1, w_2, \dots, w_n$ mit $w_0 = S$ und
 $w_0 \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n$

heißt **Ableitung** von w_n (aus S). Dabei darf w_n sowohl Terminale als auch Variablen enthalten, ist also eine Satzform.

Hier ist eine Ableitung von $aabbcc$ aus S mittels der Grammatik G von Folie 24:

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

Definition (die von einer Grammatik erzeugte Sprache)

Die von einer Grammatik $G = (V, \Sigma, P, S)$ **erzeugte (dargestellte, definierte) Sprache** ist

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

Dabei ist \Rightarrow_G^* die **reflexive und transitive Hülle** von \Rightarrow_G , d.h. $u \Rightarrow_G^* v$ genau dann, wenn $n \geq 0$ und Satzformen $u_0, u_1, \dots, u_n \in (V \cup \Sigma)^*$ existieren mit: $u_0 = u$, $u_n = v$ und $u_i \Rightarrow_G u_{i+1}$ für alle $0 \leq i \leq n-1$.

In anderen Worten: Die von G erzeugte Sprache $L(G)$ besteht genau aus den Satzformen, die in beliebig vielen Schritten aus S abgeleitet werden können und nur aus Terminalen bestehen.

Die vorherige Beispielgrammatik G (Folie 24) erzeugt die Sprache

$$L(G) = \{a^n b^n c^n \mid n \geq 1\}.$$

Dabei ist $a^n = \underbrace{a \dots a}_{n\text{-mal}}$.

Die Behauptung, dass G wirklich diese Sprache erzeugt, ist nicht offensichtlich.

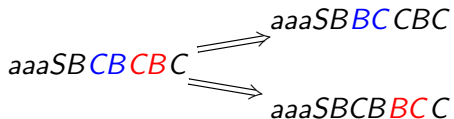
Bemerkung: Ableiten ist kein **deterministischer**, sondern ein **nichtdeterministischer** Prozess. Für ein $u \in (V \cup \Sigma)^*$ kann es entweder gar kein, ein oder mehrere v geben mit $u \Rightarrow_G v$.

In anderen Worten: \Rightarrow_G ist keine Funktion.

Dieser Nichtdeterminismus kann durch zwei verschiedene Effekte verursacht werden ...

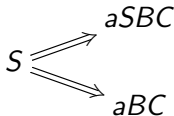
- Eine Regel ist an zwei verschiedenen Stellen anwendbar.

Beispiel-Grammatik:



- Zwei verschiedene Produktionen sind anwendbar (entweder an der gleichen Stelle – wie unten abgebildet – oder an verschiedenen Stellen):

Beispiel-Grammatik:



Weitere Bemerkungen:

- Es kann beliebig lange Ableitungen geben, die nie zu einem Wort aus Terminalsymbolen führen:

$$S \Rightarrow aSBC \Rightarrow aaSBCBC \Rightarrow aaaSBCBCBC \Rightarrow \dots$$

- Manchmal können Ableitungen in einer Sackgasse enden, d.h., obwohl noch Variablen in einer Satzform vorkommen, ist keine Regel mehr anwendbar.

$$S \Rightarrow aSBC \Rightarrow aaBCBC \Rightarrow aabCBC \Rightarrow aabcBC \not\Rightarrow$$

Chomsky-Hierarchie

Typ 0 – Chomsky-0

Jede Grammatik ist vom Typ 0 (keine Einschränkung der Produktionen).

Typ 1 – Chomsky-1

Eine Grammatik $G = (V, \Sigma, P, S)$ ist vom Typ 1 (oder **monoton**, **kontextsensitiv**), falls $|\ell| \leq |r|$ für alle Produktionen $(\ell \rightarrow r) \in P$ gilt.

Typ 2 – Chomsky-2

Eine Grammatik $G = (V, \Sigma, P, S)$ ist vom Typ 2 (oder **kontextfrei**), falls sie (i) vom Typ 1 ist und (ii) zusätzlich $\ell \in V$ für jede Produktion $(\ell \rightarrow r) \in P$ gilt.

Insbesondere muss $|r| \geq |\ell| = 1$ gelten.

Chomsky-Hierarchie

Typ 3 – Chomsky-3

Eine Grammatik $G = (V, \Sigma, P, S)$ ist vom Typ 3 (oder **regulär**), falls sie (i) vom Typ 2 ist und (ii) zusätzlich für alle Produktionen $(A \rightarrow r) \in P$ gilt: $r \in \Sigma$ oder $r = aB$ mit $a \in \Sigma, B \in V$.

D.h., die rechten Seiten von Produktionen sind entweder einzelne Terminale oder ein Terminal gefolgt von einer Variablen.

Typ- i -Sprache

Eine Sprache $L \subseteq \Sigma^*$ heißt vom Typ i ($i \in \{0, 1, 2, 3\}$), falls es eine Typ- i -Grammatik G gibt mit $L(G) = L$.

Solche Sprachen nennt man dann auch **semi-entscheidbar** bzw. **rekursiv aufzählbar** (Typ 0), **kontextsensitiv** (Typ 1), **kontextfrei** (Typ 2) oder **regulär** (Typ 3).

Bemerkungen:

- Woher kommt der Name “kontextsensitiv”?

Bei kontextfreien Grammatiken gibt es nur Produktionen der Form $A \rightarrow x$, wobei $A \in V$ und $x \in (\Sigma \cup V)^*$. Das bedeutet: A kann – unabhängig vom Kontext – durch x ersetzt werden.

Bei den mächtigeren kontextsensitiven Grammatiken sind dagegen Produktionen der Form $uAv \rightarrow uxv$ möglich, mit der Bedeutung: A kann nur in bestimmten Kontexten durch x ersetzt werden.

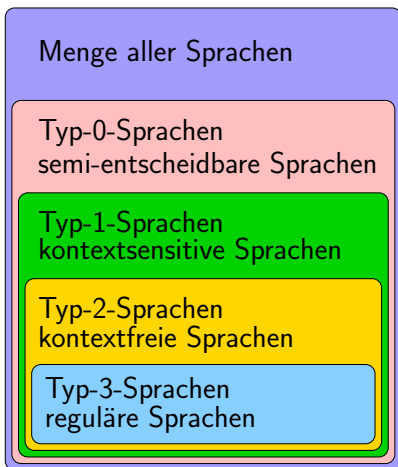
- **ε -Sonderregelung:** Bei Typ-1-Grammatiken (und damit auch bei regulären und kontextfreien Grammatiken) sind Produktionen der Form $\ell \rightarrow \varepsilon$ zunächst nicht zugelassen, wegen $|\ell| > 0$ und $|\ell| \leq |r|$ für alle $(\ell \rightarrow r) \in P$. Das bedeutet aber: das leere Wort ε kann nicht abgeleitet werden!

Wir modifizieren daher die Grammatik-Definition für Typ-1 (und Typ-2, Typ-3) Grammatiken leicht und erlauben $S \rightarrow \varepsilon$, falls S das Startsymbol ist und auf keiner rechten Seite vorkommt.

Chomsky-Hierarchie

Jede Typ- i -Grammatik ist eine Typ- $(i-1)$ -Grammatik (für $i \in \{1, 2, 3\}$) \rightsquigarrow die entsprechenden Mengen von Sprachen sind ineinander enthalten.

Außerdem: die Inklusionen sind echt, d.h., es gibt für jedes i eine Typ- $(i-1)$ -Sprache, die keine Typ- i -Sprache ist (z. B. eine kontextfreie Sprache, die nicht regulär ist). Das werden wir später zeigen.



Wortproblem

Definition (Wortproblem)

Sei $G = (V, \Sigma, P, S)$ eine Grammatik (von beliebigem Typ). Das **Wortproblem** für $L(G)$ ist das folgende Entscheidungsproblem:

EINGABE: Ein Wort $w \in \Sigma^*$.

FRAGE: Gilt $w \in L(G)$?

Satz (Entscheidbarkeit des Wortproblems für Typ 1)

Es gibt einen Algorithmus, der als Eingabe eine Typ-1-Grammatik $G = (V, \Sigma, P, S)$ und ein Wort $w \in \Sigma^*$ bekommt, und nach endlicher Zeit "Ja" (bzw. "Nein") ausgibt, falls $w \in L(G)$ (bzw. $w \notin L(G)$) gilt.

Man sagt auch: Das Wortproblem ist entscheidbar für Typ-1-Sprachen (eine genauere Definition kommt später in der Vorlesung).

Beweis:

Falls $w = \varepsilon$ gilt, müssen wir nur überprüfen, ob $S \rightarrow \varepsilon$ eine Produktion ist.

Wenn ja, gilt $w \in L(G)$, sonst gilt $w \notin L(G)$.

Sei nun $w \neq \varepsilon$ und sei $n = |w| \geq 1$.

Wir definieren einen gerichteten **endlichen** Graphen \mathcal{G} wie folgt:

- Die Menge der Knoten von \mathcal{G} ist die Menge

$$K := \{u \in (V \cup \Sigma)^+ \mid |u| \leq n\}$$

aller Satzformen der Länge höchstens n .

- Für $u, v \in K$ gibt es eine Kante $u \rightarrow v$, falls $u \Rightarrow_G v$ gilt.

Beachte: $|K| = \sum_{i=1}^n (|V| + |\Sigma|)^i$.

Da G eine Typ-1-Grammatik ist gilt: $w \in L(G)$ genau dann, wenn es in dem Graphen \mathcal{G} einen Pfad vom Knoten $S \in K$ zum Knoten $w \in K$ gibt.

Begründung: Leitet man mit einer Typ-1-Grammatik ein Wort der Länge $n \geq 1$ aus dem Startsymbol ab, so kommt in der Ableitung keine Satzform der Länge $> n$ vor (dies gilt bei einer Typ-0-Grammatik im Allgemeinen nicht).

Man konstruiert nun den Graphen \mathcal{G} indem man alle Knoten aus K in einer for-Schleife durchläuft und für jeden Knoten $u \in K$ die Menge $\{v \mid u \Rightarrow_G v\}$ aller direkten Nachfolgerknoten von u generiert.

Mittels Tiefensuche (\rightsquigarrow Vorlesung *Algorithmen & Datenstrukturen*) kann man nun testen, ob es im Graphen \mathcal{G} einen Pfad von S nach w gibt. \square

Bemerkung: Dieser Algorithmus ist nicht sehr effizient, da die Größe des konstruierten Graphens exponentiell mit der Länge des Eingabewortes w steigt (man spricht von einem Exponentialzeitalgorithmus).

Man vermutet, dass dies aber auch nicht vermeidbar ist:

Das Wortproblem für Typ-1-Grammatiken ist ein sogenanntes PSPACE-vollständiges Problem, siehe Vorlesung *Komplexitätstheorie I*.

Für PSPACE-vollständige Probleme kennt man keine Algorithmen mit einer polynomiellen Laufzeit.

Syntaxbäume und Eindeutigkeit

Wir betrachten folgende Beispiel-Grammatik (eine Typ-2-Grammatik) zur Erzeugung von korrekt geklammerten arithmetischen Ausdrücken:

$$G = (\{E, T, F\}, \{ (,), a, +, * \}, P, E)$$

mit folgender Produktionenmenge P (in abkürzender Backus-Naur-Form):

$$E \rightarrow T \mid E + T$$

$$T \rightarrow F \mid T * F$$

$$F \rightarrow a \mid (E)$$

In der **Backus-Naur-Form** für Typ-2-Grammatiken schreibt man mehrere Produktionen

$$A \rightarrow w_1, A \rightarrow w_2, \dots, A \rightarrow w_k \quad (1)$$

in der Form

$$A \rightarrow w_1 \mid w_2 \mid \dots \mid w_k.$$

Dies ist nur eine Abkürzung für (1).

Syntaxbäume und Eindeutigkeit

Für die meisten Wörter der von G erzeugten Sprache gibt es mehrere mögliche Ableitungen:

$$E \Rightarrow T \Rightarrow T * F \Rightarrow F * F \Rightarrow a * F \Rightarrow a * (E)$$

$$\Rightarrow a * (E + T) \Rightarrow a * (T + T) \Rightarrow a * (F + T)$$

$$\Rightarrow a * (a + T) \Rightarrow a * (a + F) \Rightarrow a * (a + a)$$

$$E \Rightarrow T \Rightarrow T * F \Rightarrow T * (E) \Rightarrow T * (E + T)$$

$$\Rightarrow T * (E + F) \Rightarrow T * (E + a) \Rightarrow T * (T + a)$$

$$\Rightarrow T * (F + a) \Rightarrow T * (a + a) \Rightarrow F * (a + a) \Rightarrow a * (a + a)$$

Die erste Ableitung ist eine sogenannte **Linksableitung** (in jedem Schritt wird das am weitesten links stehende Nicht-Terminal ersetzt), die zweite eine **Rechtsableitung** (in jedem Schritt wird das am weitesten rechts stehende Nicht-Terminal ersetzt).

Wir bilden nun aus beiden Ableitungen den **Syntaxbaum**, indem wir

- Die Wurzel des Baums mit der Startvariablen der Grammatik beschriften.
- Bei jeder Anwendung einer Produktion $A \rightarrow z$ zu A genau $|z|$ Kinder hinzufügen, die mit den Zeichen von z beschriftet sind.

Syntaxbäume lassen sich für alle Ableitungen von kontextfreien Grammatiken aufbauen.

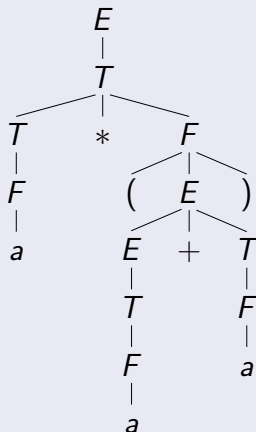
Syntaxbäume und Eindeutigkeit

Dabei erhalten wir in beiden Fällen den gleichen Syntaxbaum.

Man sagt, eine Grammatik ist **eindeutig**, wenn es für jedes Wort in der erzeugten Sprache genau einen Syntaxbaum gibt

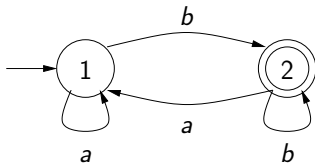
\iff es gibt für jedes Wort genau eine Linksableitung

\iff es gibt für jedes Wort genau eine Rechtsableitung.



Den Inhalt der Folien 44–88 finden Sie im Buch von Schöning auf Seite 19–27.

In diesem Abschnitt beschäftigen wir uns mit regulären Sprachen, aber zunächst unter einem anderen Blickwinkel. Statt Typ-3-Grammatiken betrachten wir **zustandsbasierte Automatenmodelle**, die man auch als “Spracherzeuger” bzw. “Sprachakzeptierer” betrachten kann.



Deterministische endliche Automaten

Definition (Deterministischer endlicher Automat)




Ein **(deterministischer) endlicher Automat** M ist ein 5-Tupel $M = (Z, \Sigma, \delta, z_0, E)$, wobei:

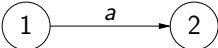
- Z eine **endliche** Menge von **Zuständen** ist,
- Σ das **endliche Eingabealphabet** (mit $Z \cap \Sigma = \emptyset$) ist,
- $z_0 \in Z$ der **Startzustand** ist,
- $E \subseteq Z$ die Menge der **Endzustände** ist und
- $\delta: Z \times \Sigma \rightarrow Z$ die **Überföhrungsfunktion** (oder **Übergangsfunktion**) ist.

Abkürzung: DFA (deterministic finite automaton)

Deterministische endliche Automaten

Graphische Notation:

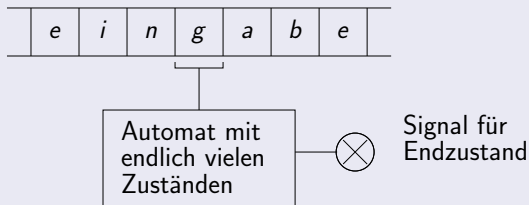
Zustand:  Startzustand:  Endzustand: 

Übergang $\delta(1, a) = 2$: 

Deterministische endliche Automaten

Woher kommt der Name “endlicher Automat”?

Vorstellung von einer Maschine, die sich in endlich vielen Zuständen befinden kann, die eine Eingabe liest und die signalisiert, sobald die Eingabe akzeptiert ist.



Analogie Fahrkartenautomat:

Ein Fahrkartenautomat kann sich in folgenden Zuständen befinden:

- Keine Eingabe
- Fahrtziel ausgewählt
- Geld eingegeben
- Fahrkarte wurde ausgegeben

Das ist natürlich nur die halbe Wahrheit, da ein Fahrkartenautomat mitzählen muss, wieviel Geld bereits eingeworfen wurde. Eine Modellierung mit nur endlich vielen Zuständen ist daher stark vereinfacht.

Deterministische endliche Automaten

Von einem zugegebenerweise sehr abstrakten Standpunkt aus, ist jeder reale Rechner auch ein DFA:

- Die Menge der Zustände ist die Menge aller möglichen Speicherbelegungen.

Wenn der gesamte Speicher des Rechners aus n Bits besteht, dann gibt es 2^n mögliche Speicherbelegungen (eine Speicherbelegung kann man sich auch als ein Wort aus $\{0, 1\}^n$ vorstellen).

Beispiel: Ein Rechner mit 8 GB Hauptspeicher und 512 GB Festplattenspeicher kann insgesamt $8 \cdot 520 \cdot 1000^3 = 4160000000000$ Bits speichern und entspricht damit einem DFA mit $2^{4160000000000}$ Zuständen!

- Der Anfangszustand ist die Speicherbelegung im Werkszustand.

Deterministische endliche Automaten

- Die Überföhrungsfunktion ergibt sich aus dem Verhalten des Rechners bei Eingaben.

Angenommen ihr Rechner bekommt Eingaben nur über die Tastatur.

Dann besteht das Eingabealphabet aus den Tasten des Rechners.

Befindet sich der Rechner in einem bestimmten Speicherzustand und wird eine bestimmte Taste gedrückt (Eingabe), dann geht der Rechner in einen neuen Zustand über.

- Endzustände machen bei einem realen Rechner weniger Sinn, da ein Computer eher selten zum akzeptieren von Wörtern eingesetzt wird.

Obige Sichtweise ist für die Praxis natürlich viel zu abstrakt und auch völlig inpraktikabel, wie man an den $2^{4160000000000}$ Zuständen sieht, wird aber dennoch bei kleineren Hardwarekomponenten im Bereich der sogenannten Hardwareverifikation (siehe Mastervorlesung *Model-Checking* von Prof. Lochau) eingesetzt.

Deterministische endliche Automaten

Die bisherige Übergangsfunktion δ eines DFA liest nur ein Zeichen auf einmal ein. Wir verallgemeinern sie daher zu einer Übergangsfunktion $\hat{\delta}$, die die Übergänge für ganze Wörter ermittelt.

Definition (Mehr-Schritt-Übergänge eines DFA)

Zu einem gegebenen DFA $M = (Z, \Sigma, \delta, z_0, E)$ definieren wir eine Funktion $\hat{\delta}: Z \times \Sigma^* \rightarrow Z$ induktiv wie folgt, wobei $z \in Z$, $x \in \Sigma^*$ und $a \in \Sigma$:

$$\begin{aligned}\hat{\delta}(z, \varepsilon) &= z \\ \hat{\delta}(z, ax) &= \hat{\delta}(\delta(z, a), x)\end{aligned}$$

Intuition: $\hat{\delta}(z, a_1 a_2 \cdots a_n)$ ist der Zustand, den man vom Zustand z aus erreicht, indem man erst der mit a_1 beschrifteten Kante folgt, dann der mit a_2 beschrifteten Kante folgt, u.s.w.:

$$z \xrightarrow{a_1} z_1 \xrightarrow{a_2} z_2 \xrightarrow{a_3} \cdots \xrightarrow{a_n} z_n = \hat{\delta}(z, a_1 a_2 \cdots a_n).$$

Deterministische endliche Automaten

Ohne es immer explizit zu erwähnen, verwenden wir häufig die folgende einfach zu zeigende Aussage:

Lemma 1

Für alle Wörter $u, v \in \Sigma^*$ und jeden Zustand $z \in Z$ gilt:

$$\hat{\delta}(z, uv) = \hat{\delta}(\hat{\delta}(z, u), v).$$

Definition (von einem DFA akzeptierte Sprache)

Die von einem DFA $M = (Z, \Sigma, \delta, z_0, E)$ **akzeptierte Sprache** ist

$$T(M) = \{x \in \Sigma^* \mid \hat{\delta}(z_0, x) \in E\}.$$

In anderen Worten:

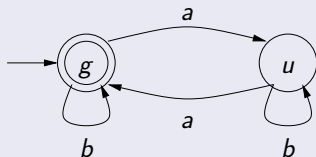
Die Sprache kann man dadurch erhalten, indem man allen Pfaden vom Anfangszustand zu einem Endzustand folgt und dabei alle Zeichen auf den Übergängen aufammelt.

Deterministische endliche Automaten

Beispiel 1: Wir suchen einen DFA, der folgende Sprache L akzeptiert:

$$L = \{w \in \{a, b\}^* \mid \#_a(w) \text{ gerade}\}.$$

Dabei ist $\#_a(w)$ die Anzahl der a 's in w .



Bedeutung der Zustände:

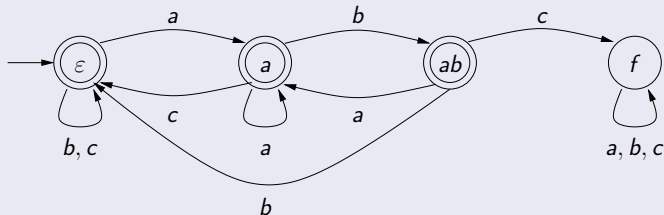
g – gerade Anzahl a 's

u – ungerade Anzahl a 's

Deterministische endliche Automaten

Beispiel 2: Wir suchen einen DFA M mit

$$T(M) = \{w \in \{a, b, c\}^* \mid \text{das Teilwort } abc \text{ kommt in } w \text{ nicht vor}\}.$$



Bedeutung der Zustände:

- ϵ : kein Präfix von abc gelesen
- a : letztes gelesenes Zeichen war ein a
- ab : zuletzt ab gelesen
- f abc kam im bereits gelesenen Wort vor (Fangzustand, Fehlerzustand)

Satz (DFAs \rightarrow reguläre Grammatik)

Jede von einem DFA akzeptierte Sprache ist regulär.

Bemerkung: Es gilt auch die umgekehrte Aussage: jede reguläre Sprache kann von einem DFA akzeptiert werden (dazu später mehr.)

Beweis:

Sei $M = (Z, \Sigma, \delta, z_0, E)$ ein DFA.

Zunächst modifizieren wir M so, dass in den Anfangszustand keine Kanten hineinführen, d.h.

$$\delta(z, a) \neq z_0$$

für alle $z \in Z$ und $a \in \Sigma$.

Idee: Wir führen eine Kopie z'_0 des Anfangszustands z_0 zum DFA hinzu, der die gleichen ausgehenden Kanten wie z_0 hat. Dann leiten wir alle Kanten, die zum Zustand z_0 führen, nach z'_0 um.

Formal: Sei $z'_0 \notin Z$ ein neuer Zustand und $Z' = Z \cup \{z'_0\}$.

Deterministische endliche Automaten

Sei $M' = (Z', \Sigma, \delta', z_0, E')$, wobei gilt:

$$\begin{aligned}\delta'(z, a) &= \begin{cases} \delta(z, a) & \text{falls } z \in Z \text{ und } \delta(z, a) \neq z_0 \\ z'_0 & \text{falls } z \in Z \text{ und } \delta(z, a) = z_0 \end{cases} \\ \delta'(z'_0, a) &= \begin{cases} \delta(z_0, a) & \text{falls } \delta(z_0, a) \neq z_0 \\ z'_0 & \text{falls } \delta(z_0, a) = z_0 \end{cases} \\ E' &= \begin{cases} E & \text{falls } z_0 \notin E \\ E \cup \{z'_0\} & \text{falls } z_0 \in E \end{cases}\end{aligned}$$

Dann gilt:

- $\delta'(z, a) \neq z_0$ für alle $z \in Z'$ und $a \in \Sigma$ und
- $T(M') = T(M)$.

Deterministische endliche Automaten

Wir schreiben nun wieder Z, δ, E für Z', δ', E' .

Wir definieren nun eine Typ-3 Grammatik $G = (V, \Sigma, P, S)$ mit $L(G) = T(M)$ wie folgt:

$$V = Z$$

$$S = z_0$$

$$P = \{z \rightarrow a \delta(z, a) \mid z \in Z, a \in \Sigma\} \cup \\ \{z \rightarrow a \mid z \in Z, a \in \Sigma, \delta(z, a) \in E\} \cup \\ \{z_0 \rightarrow \varepsilon\} \text{ falls } z_0 \in E$$

Beachte: ε -Sonderregelung ist erfüllt.

Behauptung 1: Für alle $z, z' \in Z$ und $w \in \Sigma^*$ gilt:

$$z \Rightarrow_G^* wz' \iff \hat{\delta}(z, w) = z'.$$

Deterministische endliche Automaten

Behauptung 1 zeigt man durch Induktion über $|w|$.

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

$$z \Rightarrow_G^* z' \Leftrightarrow z = z' \Leftrightarrow \hat{\delta}(z, \varepsilon) = z'$$

Induktionsschritt: Sei nun $|w| = n + 1$.

Wir können dann w schreiben als $w = av$ mit $|v| = n$ und $a \in \Sigma$.

Induktionshypothese: Behauptung 1 gilt für v .

Es gilt:

$$\begin{aligned} z \Rightarrow_G^* avz' &\iff \exists z'' \in Z : (z \rightarrow az'') \in P \text{ und } z'' \Rightarrow_G^* vz' \\ &\iff \delta(z, a) \Rightarrow_G^* vz' \\ \text{Ind.hyp.} &\iff \hat{\delta}(\delta(z, a), v) = z' \\ &\iff \hat{\delta}(z, av) = z' \end{aligned}$$

Deterministische endliche Automaten

Behauptung 2: Für alle $w \in \Sigma^*$ gilt: $w \in L(G) \iff w \in T(M)$.

1. Fall $w = \varepsilon$.

Es gilt:

$$\varepsilon \in L(G) \iff (z_0 \rightarrow \varepsilon) \in P \iff z_0 \in E \iff \varepsilon \in T(M)$$

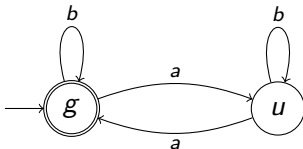
2. Fall: $w \neq \varepsilon$.

Sei $w = va$ mit $a \in \Sigma$ und $v \in \Sigma^*$. Es gilt:

$$\begin{aligned} va \in L(G) &\iff \exists z \in Z : z_0 \Rightarrow_G^* vz \Rightarrow_G va \\ &\stackrel{\text{Beh. 1}}{\iff} \exists z \in Z : \hat{\delta}(z_0, v) = z, \hat{\delta}(z, a) \in E \\ &\iff \hat{\delta}(z_0, va) \in E \\ &\iff va \in T(M) \end{aligned}$$

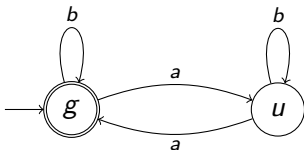
Deterministische endliche Automaten

Beispiel: Nimm den DFA von Folie 54:

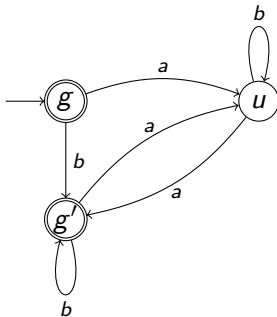


Deterministische endliche Automaten

Beispiel: Nimm den DFA von Folie 54:



Die Konstruktion von Folie 57–58 liefert folgenden DFA:



Beispiel (Fortsetzung): Die Konstruktion von Folie 59 liefert die Typ-3 Grammatik $G = (V, \{a, b\}, P, S)$ mit:

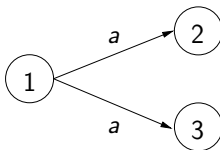
- $V = \{g, u, g'\}$,
- $S = g$,
- P besteht aus folgenden Produktionen:

$g \rightarrow \varepsilon$	$u \rightarrow a$	$g' \rightarrow b$
$g \rightarrow b$	$u \rightarrow bu$	$g' \rightarrow au$
$g \rightarrow au$	$u \rightarrow ag'$	$g' \rightarrow bg'$
$g \rightarrow bg'$		

Nichtdeterministische endliche Automaten

Im Gegensatz zu Grammatiken gibt es bei DFAs keine **nichtdeterministischen Effekte**. Das heißt, sobald das nächste Zeichen eingelesen wurde, ist klar, welcher Zustand der Folgezustand ist.

Aber: In vielen Fällen ist es natürlicher, wenn man auch nichtdeterministische Übergänge zuläßt. Das führt auch oft zu kleineren Automaten.



Definition (Nichtdeterministischer endlicher Automat)

Ein nichtdeterministischer endlicher Automat M ist ein 5-Tupel $M = (Z, \Sigma, \delta, S, E)$, wobei:

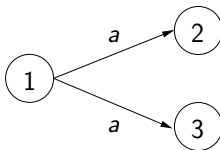
- Z ist eine **endliche** Menge von **Zuständen**,
- Σ ist das **endliche Eingabealphabet** (mit $Z \cap \Sigma = \emptyset$),
- $S \subseteq Z$ ist die Menge der **Startzustände**,
- $E \subseteq Z$ ist die Menge der **Endzustände** und
- $\delta: Z \times \Sigma \rightarrow 2^Z$ ist die **Überföhrungsfunktion** (oder **Übergangsfunktion**).

Abkürzung: NFA (nondeterministic finite automaton)

Nichtdeterministische endliche Automaten

Zur Erinnerung: $2^Z = \{A \mid A \subseteq Z\}$ ist die **Potenzmenge** von Z .

Beispiel: $\delta(1, a) = \{2, 3\}$



Nichtdeterministische endliche Automaten

Die Übergangsfunktion δ kann wieder zu einer Mehr-Schritt-Übergangsfunktion erweitert werden:

Definition (Mehr-Schritt-Übergänge eines NFA)

Zu einem gegebenen NFA $M = (Z, \Sigma, \delta, S, E)$ definieren wir eine Funktion

$$\hat{\delta}: 2^Z \times \Sigma^* \rightarrow 2^Z$$

induktiv wie folgt, wobei $Y \subseteq Z$, $x \in \Sigma^*$ und $a \in \Sigma$:

$$\begin{aligned}\hat{\delta}(Y, \varepsilon) &= Y \\ \hat{\delta}(Y, ax) &= \hat{\delta}\left(\bigcup_{z \in Y} \delta(z, a), x\right)\end{aligned}$$

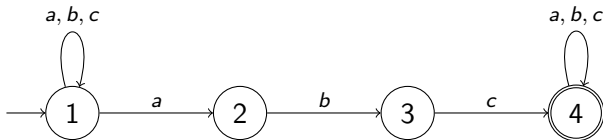
Nichtdeterministische endliche Automaten

Beachte: Die Menge

$$\bigcup_{z \in Y} \delta(z, a) = \{z' \in Z \mid \exists z \in Y : z' \in \delta(z, a)\}$$

enthält alle von einem Zustand aus Y mittels a erreichbaren Zustände.

Beispiel: Für den NFA



gilt $\widehat{\delta}(\{1\}, abca) = \{1, 2, 4\}$ und $\widehat{\delta}(\{2, 3\}, abca) = \emptyset$.

Definition (von einem NFA akzeptierte Sprache)

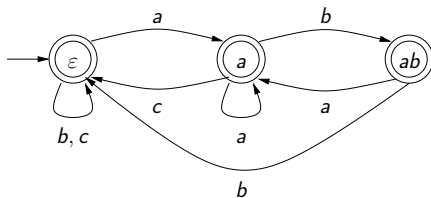
Die von einem NFA $M = (Z, \Sigma, \delta, S, E)$ akzeptierte Sprache ist

$$T(M) = \{x \in \Sigma^* \mid \hat{\delta}(S, x) \cap E \neq \emptyset\}.$$

In anderen Worten: ein Wort x wird akzeptiert, genau dann wenn es einen Pfad von einem Anfangszustand zu einem Endzustand gibt, dessen Übergänge mit den Zeichen von x markiert sind (es könnte auch mehrere solche Pfade geben).

Nichtdeterministische endliche Automaten

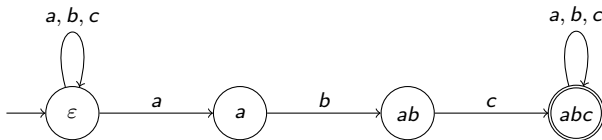
Beispiel 1: bei nicht-deterministischen Automaten darf auch $\delta(z, a) = \emptyset$ für ein $a \in \Sigma$ gelten, das heißt, es muss nicht für jedes Alphabetsymbol immer einen Übergang geben und der Fangzustand kann weggelassen werden.



Beispiel 2: gesucht ist ein NFA, der die Sprache

$$L = \{w \in \{a, b, c\}^* \mid \text{das Teilwort } abc \text{ kommt in } w \text{ vor}\}$$

akzeptiert.



Dieser Automat entscheidet zu einem bestimmten Zeitpunkt nicht-deterministisch, dass jetzt das Teilwort abc beginnt.

Bemerkung: Reale Rechner sind immer deterministisch: der nächste Zustand ist eindeutig durch den aktuellen Zustand und die Eingabe festgelegt.

Warum braucht man dann überhaupt Nichtdeterminismus?

- NFAs erlauben in vielen Fällen eine kleinere Repräsentation von regulären Sprachen als DFAs. Ein konkretes Beispiel kommt auf Folie 81–83.
- NFAs können Systeme, über die wir kein vollständiges Wissen haben, modellieren.
- Nichtdeterministische Systeme entstehen häufig durch Abstraktion realer (deterministischer) Systeme.
- Nichtdeterminismus spielt auch in der Komplexitätstheorie eine wichtige Rolle, siehe Vorlesung *Komplexitätstheorie I*.

Andere Interpretation von Nichtdeterminismus:

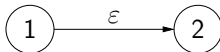
Jedes Mal, wenn eine nicht-deterministische Verzweigung möglich ist, werden mehrere “Paralleluniversen” erzeugt, in denen verschiedene Kopien der Maschine die verschiedenen möglichen Pfade erkunden.

Das Wort wird akzeptiert, wenn es in einem dieser Paralleluniversen akzeptiert wird.

Nichtdeterministische endliche Automaten

Es gibt auch nichtdeterministische Automaten mit sogenannten ε -Kanten (spontante Übergänge, bei denen kein Alphabetsymbol eingelesen wird). Diese werden jedoch in der Vorlesung im allgemeinen nicht benutzt.

Beispiel für eine ε -Kante:

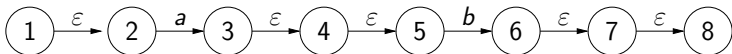


Neue Übergangsfunktion: $\delta: Z \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Z$

Im obigen Beispiel: $\delta(1, \varepsilon) = \{2\}$.

Nichtdeterministische endliche Automaten

Neue Mehr-Schritt-Übergangsfunktion: $\hat{\delta}: 2^Z \times \Sigma^* \rightarrow 2^Z$. Dabei dürfen zwischen dem Einlesen der Zeichen beliebig viele ε -Übergänge gemacht werden.



$$\hat{\delta}(\{1\}, ab) = \{6, 7, 8\}$$

Äquivalenz von NFAs mit und ohne ε -Übergänge

Jeder NFA mit ε -Übergängen kann in einen NFA ohne ε -Übergänge umgewandelt werden, ohne die akzeptierte Sprache zu ändern und ohne die Anzahl der Zustände zu erhöhen.

(Ohne Beweis.)

Satz (NFAs \rightarrow DFAs; Rabin, Scott)

Jede von einem NFA akzeptierbare Sprache kann auch von einem DFA akzeptiert werden.

Beweis:

Idee: Wir lassen die verschiedenen “Paralleluniversen” von einem Automaten simulieren. Dieser merkt sich, in welchen Zuständen er sich gerade befindet.

Das heißt, die Zustände dieses Automaten sind Mengen von Zuständen des ursprünglichen NFA. Man nennt diese Konstruktion daher auch **Potenzmengenkonstruktion**.

NFAs, DFAs und reguläre Grammatiken

Sei $M = (Z, \Sigma, \delta, S, E)$ ein NFA.

Definiere den DFA

$$M' = (2^Z, \Sigma, \gamma, S, F)$$

wobei

$$\begin{aligned}\gamma(Y, a) &= \bigcup_{z \in Y} \delta(z, a) \text{ für } Y \subseteq Z, a \in \Sigma \\ F &= \{Y \subseteq Z \mid Y \cap E \neq \emptyset\}\end{aligned}$$

Intuition: $\gamma(Y, a)$ ist die Menge aller Zustände $z' \in Z$ die von einem Zustand aus Y durch eine a -Kante erreicht werden können.

Durch Induktion über die Länge des Wortes $w \in \Sigma^*$ zeigen wir für alle $Y \subseteq Z$:

$$\hat{\gamma}(Y, w) = \hat{\delta}(Y, w)$$

NFAs, DFAs und reguläre Grammatiken

Induktionsanfang: $\hat{\gamma}(Y, \varepsilon) = Y = \hat{\delta}(Y, \varepsilon)$

Induktionsschritt: Sei $w = ax$ mit $a \in \Sigma$ und $x \in \Sigma^*$. Dann gilt:

$$\begin{aligned}\hat{\gamma}(Y, ax) &= \hat{\gamma}(\gamma(Y, a), x) \\ &\stackrel{\text{Ind.hyp.}}{=} \hat{\delta}(\gamma(Y, a), x) \\ &= \hat{\delta}\left(\bigcup_{z \in Y} \delta(z, a), x\right) \\ &= \hat{\delta}(Y, ax)\end{aligned}$$

Also gilt für jedes Wort $w \in \Sigma^*$:

$$\begin{aligned}w \in T(M') &\iff \hat{\gamma}(S, w) \in F \\ &\iff \hat{\delta}(S, w) \cap F \neq \emptyset \\ &\iff w \in T(M)\end{aligned}$$




Bemerkung:

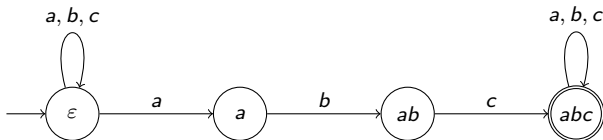
- Die Potenzmengenkonstruktion macht aus einem NFA mit n Zuständen einen äquivalenten DFA mit 2^n Zuständen.
- In vielen Fällen werden diese 2^n Zustände aber nicht alle benötigt.
- Es ist daher angeraten, bei der Potenzmengenkonstruktion nur die Teilmengen von Z , die auch wirklich benötigt werden, in dem DFA aufzunehmen.

Auf der nächsten Folie konstruieren wir auf diese Weise schrittweise für den NFA von Folie 71 einen äquivalenten DFA.

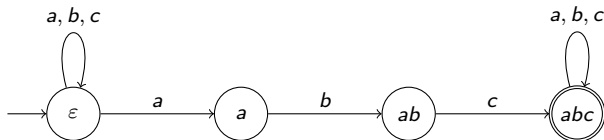
Nur 6 der $2^4 = 16$ möglichen Teilmengen werden benötigt.

Der Knoten  steht z.B. für die Teilmenge $\{\varepsilon, ab, abc\}$.

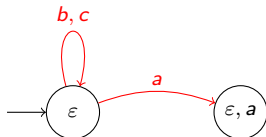
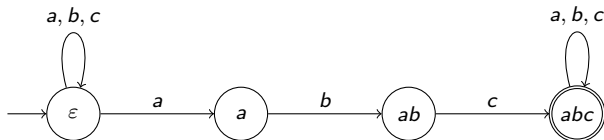
Beispiel 1:



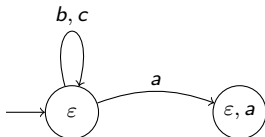
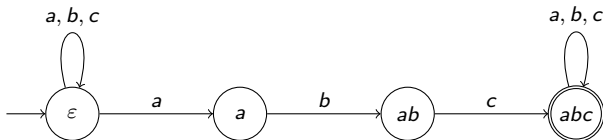
Beispiel 1:



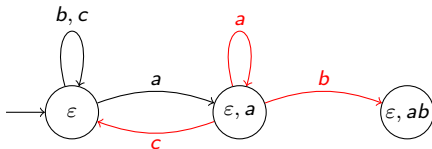
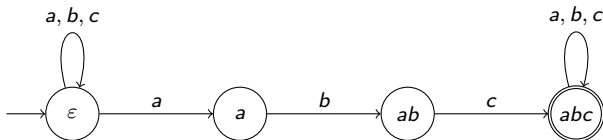
Beispiel 1:



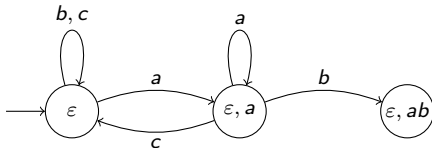
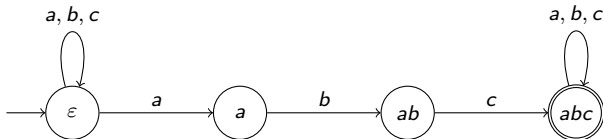
Beispiel 1:



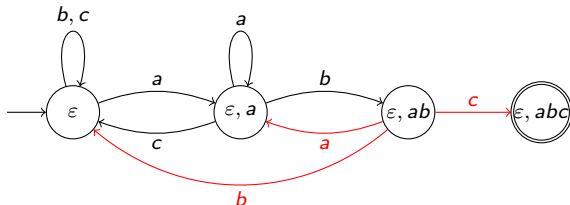
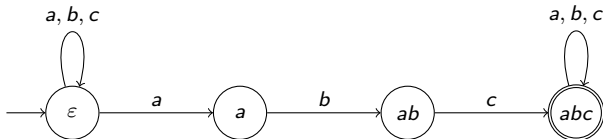
Beispiel 1:



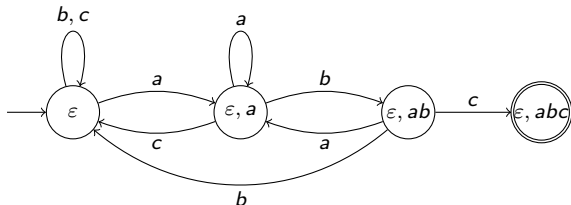
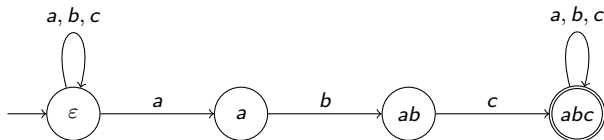
Beispiel 1:



Beispiel 1:

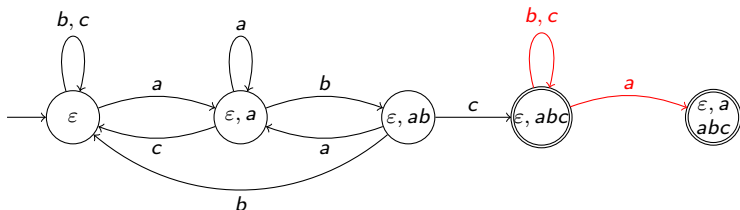
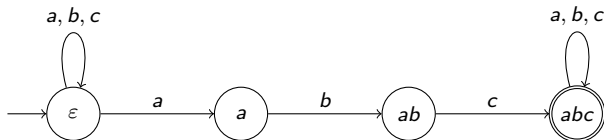


Beispiel 1:



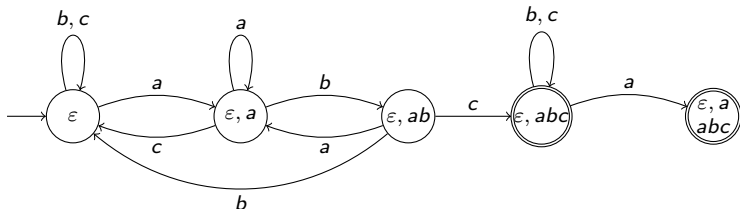
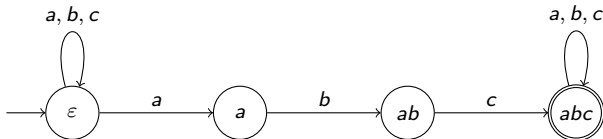
NFAs, DFAs und reguläre Grammatiken

Beispiel 1:

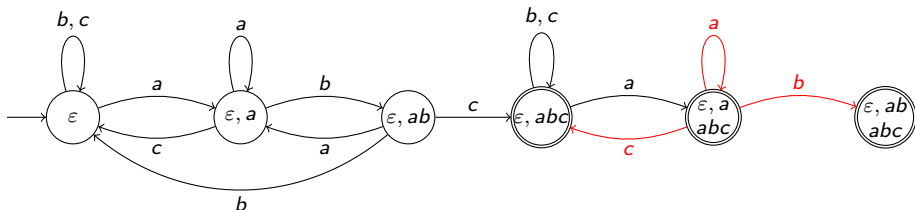
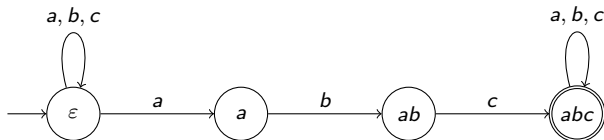


NFAs, DFAs und reguläre Grammatiken

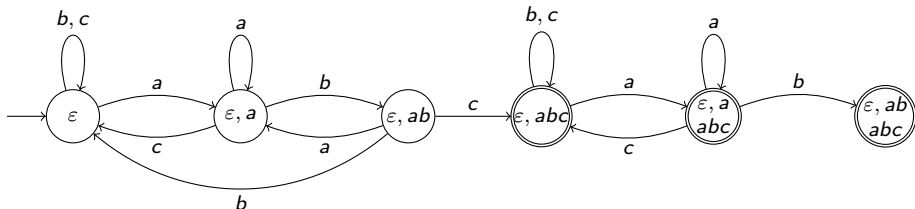
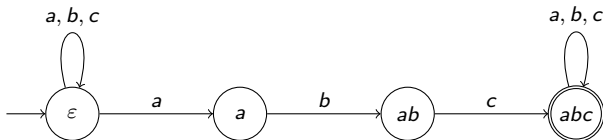
Beispiel 1:



Beispiel 1:

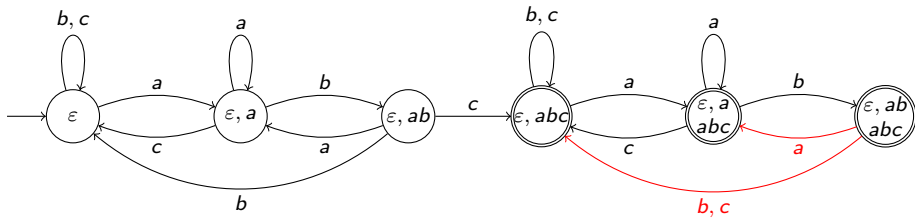
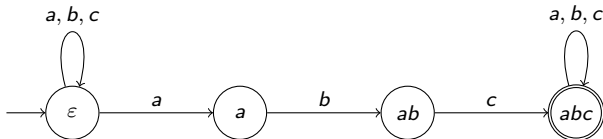


Beispiel 1:



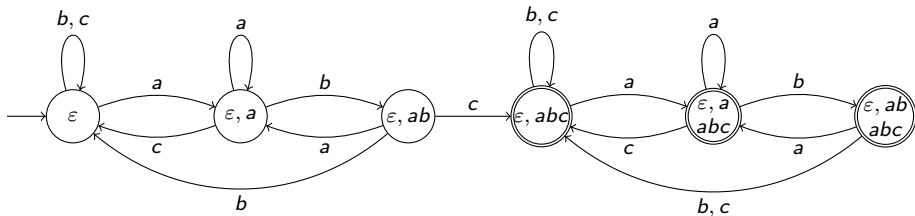
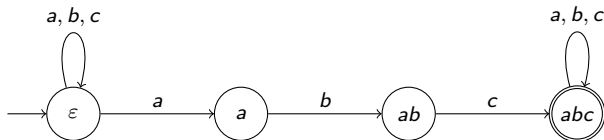
NFAs, DFAs und reguläre Grammatiken

Beispiel 1:



NFAs, DFAs und reguläre Grammatiken

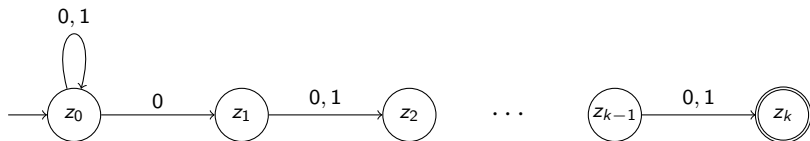
Beispiel 1:



Beispiel 2: Für $k \geq 1$ sei

$$L_k = \{w \in \{0,1\}^* \mid |w| \geq k, \text{ das } k\text{-letzte Zeichen von } w \text{ ist } 0\}.$$

(A) Es gibt einen NFA M mit $k + 1$ Zuständen und $T(M) = L_k$:



(B) Es gibt **keinen** DFA M mit weniger als 2^k Zuständen und $T(M) = L_k$.

Beweis von (B):

Angenommen, $M = (Z, \{0, 1\}, \delta, z_0, E)$ wäre ein DFA mit weniger als 2^k Zuständen und $T(M) = L_k$.

Dann gibt es Wörter $w_1, w_2 \in \{0, 1\}^k$ mit $w_1 \neq w_2$ und $\hat{\delta}(z_0, w_1) = \hat{\delta}(z_0, w_2)$ (denn es gibt 2^k viele Wörter in $\{0, 1\}^k$).

Sei $i \in \{1, \dots, k\}$ die erste Position, an der sich w_1 und w_2 unterscheiden.

Sei $w \in \{0, 1\}^{i-1}$ beliebig.

NFAs, DFAs und reguläre Grammatiken

Dann existieren Wörter $v, v' \in \{0, 1\}^{k-i}$ und $u \in \{0, 1\}^{i-1}$ mit (o.B.d.A.)

$$w_1 w = u0vw \quad \text{und} \quad w_2 w = u1v'w.$$

Wegen $|vw| = |v'w| = k - i + i - 1 = k - 1$ gilt

$$w_1 w \in L_k \quad \text{und} \quad w_2 w \notin L_k.$$

Aber:

$$\hat{\delta}(z_0, w_1 w) = \hat{\delta}(\hat{\delta}(z_0, w_1), w) = \hat{\delta}(\hat{\delta}(z_0, w_2), w) = \hat{\delta}(z_0, w_2 w),$$

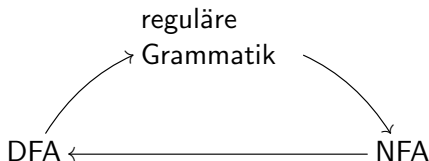
d.h. $w_1 w \in L_k \Leftrightarrow w_2 w \in L_k$. **Widerspruch!**

NFAs, DFAs und reguläre Grammatiken

Wir können nun

- NFAs in DFAs umwandeln und
- DFAs in reguläre Grammatiken umwandeln.

Es fehlt noch die Richtung “reguläre Grammatik \rightarrow NFA”, dann haben wir die Äquivalenz aller dieser Formalismen gezeigt.



Satz (Reguläre Grammatiken \rightarrow NFAs)

Zu jeder regulären Grammatik G gibt es einen NFA M mit $L(G) = T(M)$.

Beweis:

Sei $G = (V, \Sigma, P, S)$ eine reguläre Grammatik.

Wir definieren den NFA $M = (V \cup \{X\}, \Sigma, \delta, \{S\}, E)$, wobei $X \notin V$ und

$$\delta(A, a) = \{B \mid (A \rightarrow aB) \in P\} \cup \{X \mid (A \rightarrow a) \in P\} \text{ für } A \in V, a \in \Sigma$$

$$\delta(X, a) = \emptyset \text{ für } a \in \Sigma$$

$$E = \begin{cases} \{S, X\} & \text{falls } (S \rightarrow \varepsilon) \in P \\ \{X\} & \text{falls } (S \rightarrow \varepsilon) \notin P \end{cases}$$

NFAs, DFAs und reguläre Grammatiken

Wegen der Konstruktion gilt

$$\varepsilon \in L(G) \iff (S \rightarrow \varepsilon) \in P \iff \{S\} \cap E \neq \emptyset \iff \varepsilon \in T(M).$$

Wir müssen also noch für alle Wörter $w \in \Sigma^+$ zeigen:

$$w \in L(G) \iff w \in T(M).$$

Behauptung: Für alle $w \in \Sigma^*$ und alle $A, B \in V$ gilt:

$$A \Rightarrow_G^* wB \iff B \in \widehat{\delta}(\{A\}, w)$$

Wir zeigen diese Behauptung durch Induktion über $|w|$.

Induktionsanfang: $w = \varepsilon$. Es gilt:

$$A \Rightarrow_G^* B \iff A = B \iff B \in \{A\} = \widehat{\delta}(\{A\}, \varepsilon)$$

NFAs, DFAs und reguläre Grammatiken

Induktionsschritt: Sei $w = av$ ($a \in \Sigma$, $v \in \Sigma^*$) und gelte die Behauptung bereits für das Wort v .

$$\begin{aligned} A \Rightarrow_G^* avB &\iff \exists C \in V : (A \rightarrow aC) \in P \text{ und } C \Rightarrow_G^* vB \\ &\iff \exists C \in V : C \in \delta(A, a) \text{ und } B \in \widehat{\delta}(\{C\}, v) \\ &\iff \exists C \in V \cup \{X\} : C \in \delta(A, a) \text{ und } B \in \widehat{\delta}(\{C\}, v) \\ &\iff B \in \widehat{\delta}(\{A\}, av) \end{aligned}$$

Dies zeigt die Behauptung.

Sei nun $w \in \Sigma^+$, etwa $w = va$ mit $a \in \Sigma$. Dann gilt:

$$\begin{aligned} va \in L(G) &\iff \exists A \in V : S \Rightarrow_G^* vA \text{ und } (A \rightarrow a) \in P \\ &\stackrel{\text{Beh.}}{\iff} \exists A \in V : A \in \widehat{\delta}(\{S\}, v) \text{ und } X \in \delta(A, a) \\ &\iff \exists A \in V \cup \{X\} : A \in \widehat{\delta}(\{S\}, v) \text{ und } X \in \delta(A, a) \\ &\iff X \in \widehat{\delta}(\{S\}, va) \\ &\iff va \in T(M) \end{aligned}$$

Beachte für die letzte Äquivalenz: Entweder

- X ist der einzige Endzustand von M oder
- S ist der zweite Endzustand.

Dann gilt $(S \rightarrow \varepsilon) \in P$.

Wegen der ε -Sonderregelung kommt dann S nicht auf der rechten Seite einer Produktion aus P vor.

Also gilt $S \notin \delta(A, a)$ für alle $A \in V \cup \{X\}$, $a \in \Sigma$.

Hieraus folgt $S \notin \widehat{\delta}(\{S\}, va)$.

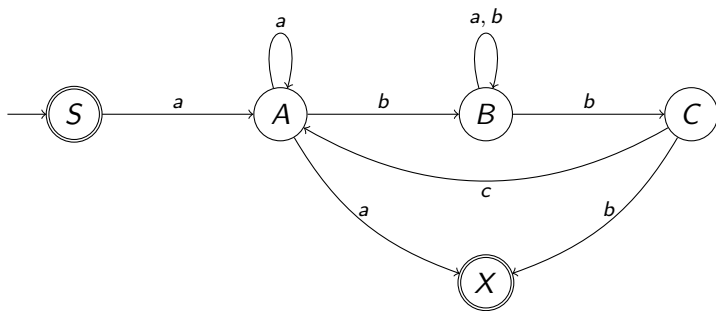


NFAs, DFAs und reguläre Grammatiken

Beispiel: Sei G die reguläre Grammatik mit folgenden Produktionen (wir verwenden die Backus-Naur Form, siehe Folie 41):

$$\begin{array}{ll} S \rightarrow \varepsilon \mid aA & A \rightarrow aA \mid bB \mid a \\ B \rightarrow aB \mid bB \mid bC & C \rightarrow cA \mid b \end{array}$$

Die Konstruktion von Folie 85 liefert den folgenden NFA:



Zwischenzusammenfassung

Wir haben verschiedene Modelle zur Beschreibung regulärer Sprachen kennengelernt:

- **Reguläre Grammatiken:** Schaffen die Verbindung zur Chomsky-Hierarchie. Werden zur Erzeugung von Sprachen eingesetzt. Sind weniger gut geeignet, um zu entscheiden, ob ein bestimmtes Wort zur Sprache gehört.
- **NFAs:** Erlauben oft kleine, kompakte Darstellungen von Sprachen. Sind, wegen ihres Nichtdeterminismus, genauso wie Grammatiken weniger gut für die Lösung des Wortproblems geeignet. Besitzen aber eine intuitive graphische Notation.
- **DFAs:** Können gegenüber äquivalenten NFAs exponentiell größer sein. Sobald jedoch ein DFA vorliegt, erlaubt dieser eine effiziente Lösung des Wortproblems (einfach den Übergängen des Automaten nachlaufen und überprüfen, ob ein Endzustand erreicht wird).

Reguläre Ausdrücke

Alle Modelle benötigen jedoch relativ viel Schreibaufwand und Platz für die Notation. Gesucht wird also eine kompaktere Repräsentation. Dies sind reguläre Ausdrücke.

Definition (reguläre Ausdrücke)

Die Menge $\text{Reg}(\Sigma)$ der **regulären Ausdrücke** über dem Alphabet Σ ist die kleinste Menge mit folgenden Eigenschaften:

- $\emptyset \in \text{Reg}(\Sigma)$, $\varepsilon \in \text{Reg}(\Sigma)$, $\Sigma \subseteq \text{Reg}(\Sigma)$.
- Wenn $\alpha, \beta \in \text{Reg}(\Sigma)$, dann auch $\alpha\beta, (\alpha|\beta), (\alpha)^* \in \text{Reg}(\Sigma)$.

Bemerkungen:

- Statt $(\alpha|\beta)$ wird oft auch $(\alpha + \beta)$ geschrieben.
- Überflüssige Klammern lassen wir häufig weg.
Z. B. $(a|b)^*$ anstatt $((a|b))^*$.

Zum Einsparen von Klammern verwenden wir sogenannte **Operatorpräzedenzregeln**:

- $*$ bindet stärker als die Konkatenation.
- Konkatenation bindet stärker als $|$.

Beispiel: $ab^*|c$ lesen wir als $(a(b)^*|c)$.

Dies sind die gleichen Operatorpräzedenzregeln die man vom Rechnen mit den arithmetischen Operatoren $+$, \cdot und Potenzieren kennt.

$xy^n + z$ lesen wir als $((x \cdot (y)^n) + z)$.

Nach der Festlegung der Syntax regulärer Ausdrücke, müssen wir auch deren Bedeutung (Semantik) festlegen.

Die Semantik eines reguläre Ausdruck ist eine Sprache:

Definition (Sprache eines regulären Ausdrucks)

- $L(\emptyset) = \emptyset$ (leere Sprache), $L(\varepsilon) = \{\varepsilon\}$, $L(a) = \{a\}$ für $a \in \Sigma$.
- $L(\alpha\beta) = L(\alpha)L(\beta)$, wobei $L_1L_2 = \{w_1w_2 \mid w_1 \in L_1, w_2 \in L_2\}$ für zwei Sprachen L_1, L_2 (Konkatenation von L_1 und L_2).
- $L(\alpha|\beta) = L(\alpha) \cup L(\beta)$
- $L((\alpha)^*) = (L(\alpha))^*$, wobei $L^* = \{w_1 \cdots w_n \mid n \geq 0, w_1, \dots, w_n \in L\}$ für eine Sprache L

Beispiel für Konkatenation von Sprachen:

$$\{a, b, ab\}\{c, ba\} = \{ac, bc, abc, aba, bba, abba\}.$$

Bemerkungen zum $*$ -Operator: $L^* = \{w_1 \cdots w_n \mid n \in \mathbb{N}, w_i \in L\}$

- Für $n = 0$ ist $w_1 \cdots w_n = \varepsilon$.
- L^* enthält immer das leere Wort ε .
Spezialfall: $\emptyset^* = \{\varepsilon\}$.
- Der Operator $*$ wird oft **Kleenesche Hülle genannt**. Nur durch ihn kann man unendliche Sprachen erzeugen.
Genauer: L^* ist unendlich genau dann wenn $L \cap \Sigma^+ \neq \emptyset$
- Beispiel für die Anwendung des $*$ -Operators:

Sei $L = \{a, bb, cc\}$. Dann gilt

$$L^* = \{\varepsilon, a, bb, cc, aa, abb, acc, bba, bbbb, bbcc, cca, ccbb, cccc, \dots\}$$

Alle Kombinationen beliebiger Länge sind möglich.

Weitere Bemerkungen:

- Beachten Sie: reguläre Ausdrücke sind rein syntaktische Ausdrücke. Erst durch die Definition auf Folie 93 wird einem regulären Ausdruck eine Sprache zugeordnet.
- Die Unterscheidung zwischen Syntax und Semantik findet man in vielen Bereichen der Informatik (Programmiersprachen, Logik, etc.)
- Bei Programmiersprachen definiert man auch erst, was syntaktische korrekte Programme sind. Danach definiert man die Semantik eines Programms (was macht das Programm).
Dies kann z.B. die von einem Programm berechnete Funktion sein. Dies werden wir später auch für sehr einfache Programmiersprachen (GOTO-Programme, While-Programme) machen.

- Formal sollte man auch zwischen dem regulären Ausdruck \emptyset und der regulären Sprache \emptyset (leere Sprache) unterscheiden, aber wir wollen es nicht übertreiben.
- Häufig werden die Sprachen \emptyset und $\{\varepsilon\}$ verwechselt.
 \emptyset ist die leere Sprache (hat null Elemente).
 $\{\varepsilon\}$ ist eine Sprache, die genau ein Wort (das leere Wort) enthält.

Beispiele für reguläre Ausdrücke über dem Alphabet $\Sigma = \{a, b\}$.

Beispiel 1: Sprache aller Wörter, die mit a beginnen und mit bb enden

$$\alpha = a(a|b)^*bb$$

Beispiel 2: Sprache aller Wörter, die das Teilwort aba enthalten.

$$\alpha = (a|b)^*aba(a|b)^*$$

Beispiel 3: Sprache aller Wörter, die gerade viele a 's enthalten.

$$\alpha = (b^*ab^*a)^*b^* \quad \text{oder} \quad \alpha = (b \mid ab^*a)^*$$

Satz (reguläre Ausdrücke \rightarrow NFAs)

Zu jedem regulären Ausdruck γ gibt es einen NFA M mit $L(\gamma) = T(M)$.

Beweis: Induktion über den Aufbau von γ .

Induktionsanfang: Für $\gamma = \emptyset$, $\gamma = \varepsilon$, $\gamma = a$ ($a \in \Sigma$) gibt es offensichtlich entsprechende NFAs.

Induktionsschritt: Sei nun $\gamma = \alpha\beta$. Dann gibt es NFAs

$$M_\alpha = (Z_\alpha, \Sigma, \delta_\alpha, S_\alpha, E_\alpha)$$

$$M_\beta = (Z_\beta, \Sigma, \delta_\beta, S_\beta, E_\beta)$$

mit $T(M_\alpha) = L(\alpha)$ und $T(M_\beta) = L(\beta)$.

Wir können annehmen, dass $Z_\alpha \cap Z_\beta = \emptyset$.

Reguläre Ausdrücke

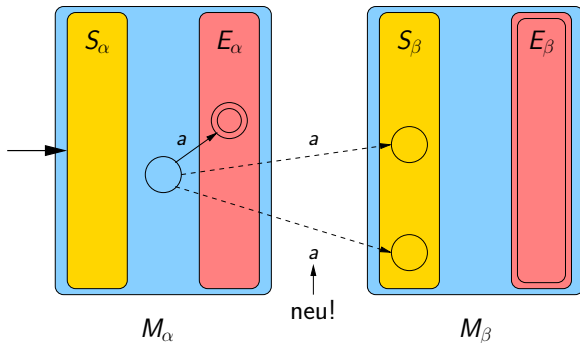
Wir verknüpfen nun M_α und M_β sequentiell zu einem NFA M :

- M hat als Zustände die Vereinigung beider Zustandsmengen, die gleichen Startzustände wie M_α und die gleichen Endzustände wie M_β . Falls $\varepsilon \in L(\alpha)$, so sind auch die Startzustände von M_β Startzustände von M .
- Alle Übergänge von M_α bzw. M_β bleiben erhalten. Alle Zustände, die einen Pfeil zu einem Endzustand von M_α haben, erhalten zusätzlich genauso beschriftete Pfeile zu allen Startzuständen von M_β .

Formal: $M = (Z_\alpha \cup Z_\beta, \Sigma, \delta, S, E_\beta)$, wobei

$$S = \begin{cases} S_\alpha & \text{falls } \varepsilon \notin L(\alpha) \\ S_\alpha \cup S_\beta & \text{falls } \varepsilon \in L(\alpha) \end{cases}$$

$$\delta(z, a) = \begin{cases} \delta_\beta(z, a) & \text{für } z \in Z_\beta \\ \delta_\alpha(z, a) & \text{für } z \in Z_\alpha \text{ mit } \delta_\alpha(z, a) \cap E_\alpha = \emptyset \\ \delta_\alpha(z, a) \cup S_\beta & \text{für } z \in Z_\alpha \text{ mit } \delta_\alpha(z, a) \cap E_\alpha \neq \emptyset \end{cases}$$



Es gilt $T(M) = T(M_\alpha)T(M_\beta) = L(\alpha)L(\beta) = L(\alpha\beta) = L(\gamma)$

Reguläre Ausdrücke

Sei nun $\gamma = (\alpha \mid \beta)$. Dann gibt es NFAs

$$M_\alpha = (Z_\alpha, \Sigma, \delta_\alpha, S_\alpha, E_\alpha)$$

$$M_\beta = (Z_\beta, \Sigma, \delta_\beta, S_\beta, E_\beta)$$

mit $T(M_\alpha) = L(\alpha)$ und $T(M_\beta) = L(\beta)$.

Wir können annehmen, dass $Z_\alpha \cap Z_\beta = \emptyset$.

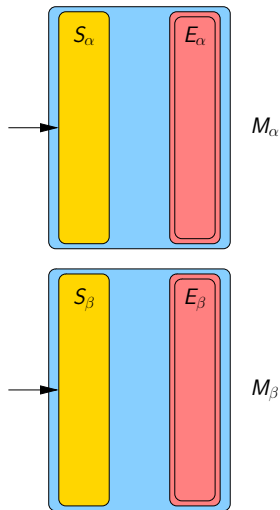
Wir bauen nun aus diesen zwei NFAs einen Vereinigungs-NFA M :

- M hat als Zustände die Vereinigung beider Zustandsmengen. Ebenso ergeben sich die Startzustände als Vereinigung der Startzustandsmengen und die Endzustände als Vereinigung der Endzustandsmengen.
- Alle Übergänge von M_α bzw. M_β bleiben erhalten.

Formal: $M = (Z_\alpha \cup Z_\beta, \Sigma, \delta, S_\alpha \cup S_\beta, E_\alpha \cup E_\beta)$, wobei

$$\delta(z, a) = \begin{cases} \delta_\alpha(z, a) & \text{für } z \in Z_\alpha \\ \delta_\beta(z, a) & \text{für } z \in Z_\beta \end{cases}$$

Reguläre Ausdrücke



$$\begin{aligned}\text{Es gilt } T(M) &= T(M_\alpha) \cup T(M_\beta) \\ &= L(\alpha) \cup L(\beta) \\ &= L(\alpha \mid \beta) \\ &= L(\gamma)\end{aligned}$$

Sei nun $\gamma = (\alpha)^*$. Dann gibt es einen NFA

$$M_\alpha = (Z_\alpha, \Sigma, \delta_\alpha, S_\alpha, E_\alpha)$$

mit $T(M_\alpha) = L(\alpha)$.

Wir bauen aus diesem NFA nun wie folgt einen NFA M :

- Falls $\varepsilon \notin T(M_\alpha)$, so gibt es einen zusätzlichen Zustand, der sowohl Start- als auch Endzustand ist (damit auch das leere Wort erkannt wird).
- Die anderen Zustände, Start- und Endzustände sowie Übergänge bleiben erhalten.
- Alle Zustände, die einen Pfeil zu einem Endzustand von M_α haben, erhalten zusätzlich genauso beschriftete Pfeile zu allen Startzuständen von M_α (Rückkopplung).

Formal: $M = (Z, \Sigma, \delta, S, E)$, wobei:

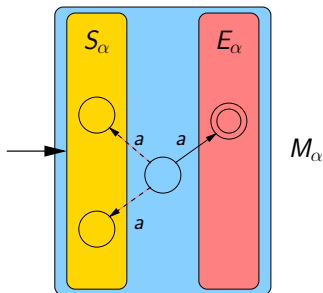
$$Z = \begin{cases} Z_\alpha & \text{falls } \varepsilon \in L(\alpha) \\ Z_\alpha \cup \{s_0\} & \text{falls } \varepsilon \notin L(\alpha) \end{cases}$$

$$S = \begin{cases} S_\alpha & \text{falls } \varepsilon \in L(\alpha) \\ S_\alpha \cup \{s_0\} & \text{falls } \varepsilon \notin L(\alpha) \end{cases}$$

$$E = \begin{cases} E_\alpha & \text{falls } \varepsilon \in L(\alpha) \\ E_\alpha \cup \{s_0\} & \text{falls } \varepsilon \notin L(\alpha) \end{cases}$$

$$\delta(z, a) = \begin{cases} \delta_\alpha(z, a) & \text{für } z \in Z_\alpha \text{ mit } \delta_\alpha(z, a) \cap E_\alpha = \emptyset \\ \delta_\alpha(z, a) \cup S_\alpha & \text{für } z \in Z_\alpha \text{ mit } \delta_\alpha(z, a) \cap E_\alpha \neq \emptyset \end{cases}$$

Hierbei gilt $s_0 \notin Z_\alpha$.



evtl. zusätzl. Zustand

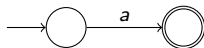
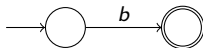
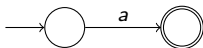
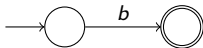
Es gilt $T(M) = (T(M_\alpha))^* = (L(\alpha))^* = L(\alpha^*) = L(\gamma)$.

Beispiel: Wie konstruieren schrittweise einen NFA für den regulären Ausdruck $(b \mid ab^*a)^*$.

Reguläre Ausdrücke

Beispiel: Wie konstruieren schrittweise einen NFA für den regulären Ausdruck $(b \mid ab^*a)^*$.

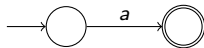
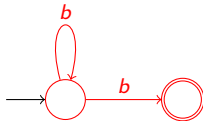
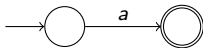
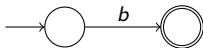
Wir beginnen mit den Übergängen für einzelne Symbole.



Reguläre Ausdrücke

Beispiel: Wie konstruieren schrittweise einen NFA für den regulären Ausdruck $(b \mid ab^*a)^*$.

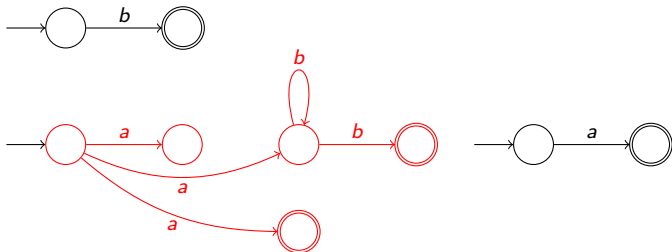
NFA für b^*



Reguläre Ausdrücke

Beispiel: Wie konstruieren schrittweise einen NFA für den regulären Ausdruck $(b \mid ab^*a)^*$.

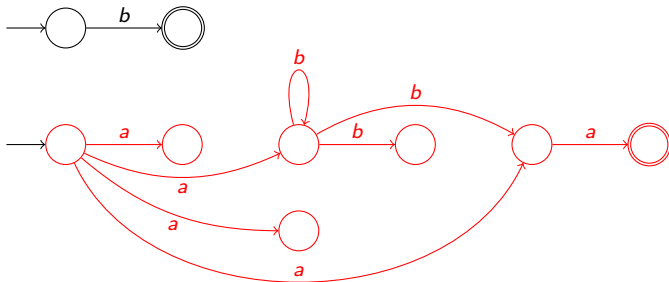
NFA für ab^*



Reguläre Ausdrücke

Beispiel: Wie konstruieren schrittweise einen NFA für den regulären Ausdruck $(b \mid ab^*a)^*$.

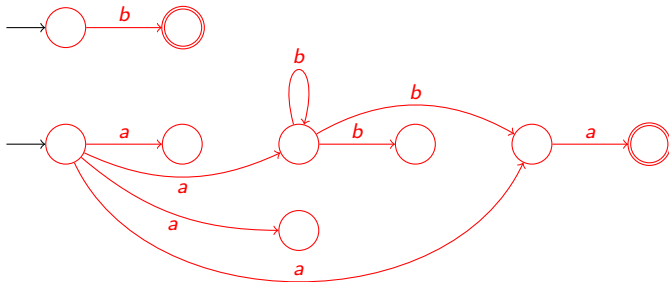
NFA für ab^*a



Reguläre Ausdrücke

Beispiel: Wie konstruieren schrittweise einen NFA für den regulären Ausdruck $(b \mid ab^*a)^*$.

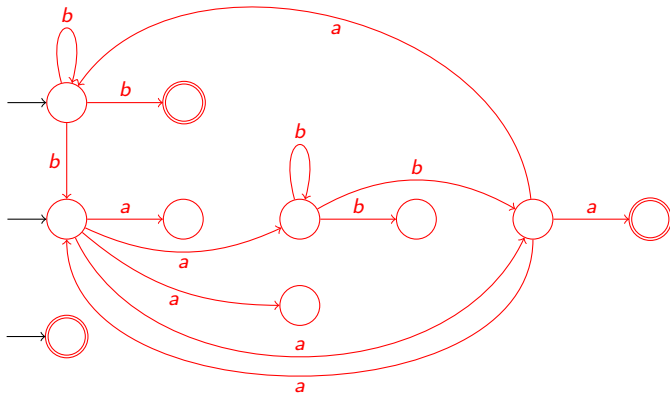
NFA für $(b \mid ab^*a)$



Reguläre Ausdrücke

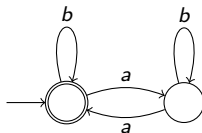
Beispiel: Wie konstruieren schrittweise einen NFA für den regulären Ausdruck $(b \mid ab^*a)^*$.

NFA für $(b \mid ab^*a)^*$



Beispiel (Fortsetzung): Dieser NFA enthält viele redundante Zustände und kann vereinfacht werden.

Ein viel einfacher NFA für $(b \mid ab^*a)^*$ ist:



Reguläre Ausdrücke

Satz (DFAs \rightarrow Reguläre Ausdrücke)

Zu jedem DFA M gibt es einen regulären Ausdruck γ mit $T(M) = L(\gamma)$.

Beweis: Sei $M = (\{z_1, \dots, z_n\}, \Sigma, \delta, z_1, E)$ ein DFA.

Wir konstruieren einen regulären Ausdruck γ mit $T(M) = L(\gamma)$.

Für ein Wort $w \in \Sigma^*$ sei

$$\text{Pref}(w) = \{u \in \Sigma^* \mid \exists v : w = uv, \varepsilon \neq u \neq w\}$$

die Menge aller nicht-leeren echten Präfixe von w .

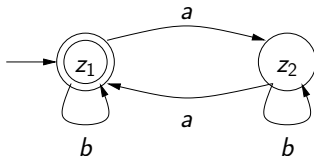
Beispiel: $\text{Pref}(abbca) = \{a, ab, abb, abbc\}$

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

$$L_{i,j}^k = \{w \in \Sigma^* \mid \widehat{\delta}(z_i, w) = z_j, \forall u \in \text{Pref}(w) : \widehat{\delta}(z_i, u) \in \{z_1, \dots, z_k\}\}.$$

Intuitiv: Ein Wort w gehört zu $L_{i,j}^k$ genau dann, wenn w den Zustand z_i in den Zustand z_j überführt, und dabei kein Zwischenzustand (ausser ganz am Anfang und ganz am Ende) aus $\{z_{k+1}, \dots, z_n\}$ vorkommt.

Beispiel: Betrachte den folgenden DFA M :



Dann gilt z.B.:

$$L_{1,1}^0 = \{\varepsilon, b\}, \quad L_{1,2}^0 = \{a\}, \quad L_{2,2}^1 = \{ab^n a \mid n \geq 0\} \cup \{\varepsilon, b\}$$

und $L_{1,1}^2 = T(M) = \{w \in \{a, b\}^* \mid w \text{ enthält gerade viele } a\text{'s}\}.$

Reguläre Ausdrücke

Wir konstruieren für alle $i, j \in \{1, \dots, n\}$ und $k \in \{0, \dots, n\}$ reguläre Ausdrücke $\gamma_{i,j}^k$ mit $L(\gamma_{i,j}^k) = L_{i,j}^k$.

Falls $E = \{z_{i_1}, z_{i_2}, \dots, z_{i_m}\}$, ergibt sich dann

$$L(\gamma_{1,i_1}^n \mid \gamma_{1,i_2}^n \mid \dots \mid \gamma_{1,i_m}^n) = T(M).$$

Konstruktion von $\gamma_{i,j}^k$ durch Induktion über $k \in \{0, \dots, n\}$.

Induktionsanfang: $k = 0$. Es gilt:

$$L_{i,j}^0 = \begin{cases} \{\varepsilon\} \cup \{a \in \Sigma \mid \delta(z_i, a) = z_j\} & \text{falls } i = j \\ \{a \in \Sigma \mid \delta(z_i, a) = z_j\} & \text{falls } i \neq j \end{cases}$$

Einen regulären Ausdruck $\gamma_{i,j}^0$ mit $L(\gamma_{i,j}^0) = L_{i,j}^0$ können wir leicht angeben.

Induktionsschritt: Sei $0 \leq k < n$ und seien die regulären Ausdrücke $\gamma_{p,q}^k$ für alle $p, q \in \{1, \dots, n\}$ bereits konstruiert.

Behauptung: Für alle $i, j \in \{1, \dots, n\}$ gilt

$$L_{i,j}^{k+1} = L_{i,j}^k \cup L_{i,k+1}^k (L_{k+1,k+1}^k)^* L_{k+1,j}^k. \quad (2)$$

Begründung:

\subseteq : Sei $w \in L_{i,j}^{k+1}$ und sei $\ell \geq 0$ so, dass der Zustand z_{k+1} auf dem eindeutigen mit w beschrifteten Pfad von z_i nach z_j genau ℓ mal als echter Zwischenzustand auftaucht.

1.Fall: $\ell = 0$, d.h. z_{k+1} kommt garnicht als echter Zwischenzustand vor.

Dann gilt $w \in L_{i,j}^k$ und somit $w \in L_{i,j}^k \cup L_{i,k+1}^k (L_{k+1,k+1}^k)^* L_{k+1,j}^k$.

2.Fall: $\ell > 0$.

Dann kann w als $w = w_0 w_1 \cdots w_{\ell-1} w_\ell$ geschrieben werden, wobei:

$$\begin{aligned} \widehat{\delta}(z_i, w_0) &= z_{k+1} \\ \widehat{\delta}(z_{k+1}, w_p) &= z_{k+1} \text{ für } 1 \leq p \leq \ell - 1 \\ \widehat{\delta}(z_{k+1}, w_\ell) &= z_j \end{aligned}$$

Reguläre Ausdrücke

Es folgt $w_0 \in L_{i,k+1}^k$, $w_1, \dots, w_{\ell-1} \in L_{k+1,k+1}^k$, $w_\ell \in L_{k+1,j}^k$ und somit

$$w = w_0(w_1 \cdots w_{\ell-1})w_\ell \in L_{i,k+1}^k(L_{k+1,k+1}^k)^*L_{k+1,j}^k.$$

\supseteq : $L_{i,j}^k \subseteq L_{i,j}^{k+1}$ ist offensichtlich.

Falls $w \in L_{i,k+1}^k(L_{k+1,k+1}^k)^*L_{k+1,j}^k$, existiert ein $\ell \geq 1$ und eine Faktorisierung $w = w_0w_1 \cdots w_{\ell-1}w_\ell$ mit

$$w_0 \in L_{i,k+1}^k, w_1, \dots, w_{\ell-1} \in L_{k+1,k+1}^k, w_\ell \in L_{k+1,j}^k.$$

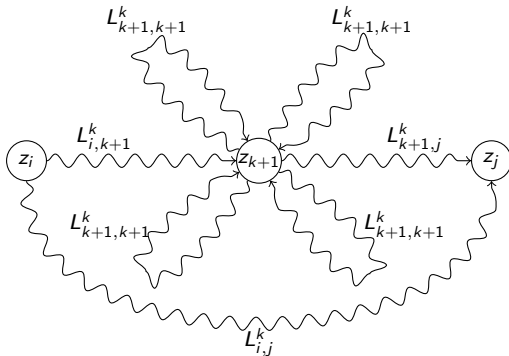
Hieraus ergibt sich leicht $w \in L_{i,j}^{k+1}$. Dies zeigt die Behauptung.

Da die regulären Ausdrücke $\gamma_{i,j}^k, \gamma_{i,k+1}^k, \gamma_{k+1,k+1}^k, \gamma_{k+1,j}^k$ schon konstruiert sind (Induktionsannahme), können wir wegen Gleichung (2) den regulären Ausdruck $\gamma_{i,j}^{k+1}$ wie folgt definieren:

$$\gamma_{i,j}^{k+1} = \gamma_{i,j}^k \mid \gamma_{i,k+1}^k(\gamma_{k+1,k+1}^k)^*\gamma_{k+1,j}^k$$

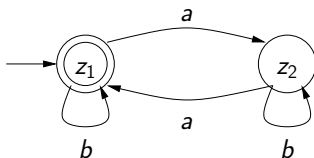


$$L_{i,j}^{k+1} = L_{i,j}^k \cup L_{i,k+1}^k (L_{k+1,k+1}^k)^* L_{k+1,j}^k$$



Reguläre Ausdrücke

Beispiel: Betrachte den folgenden DFA:



Damit ergibt sich (bei Durchführung offensichtlicher Vereinfachungen):

$$\gamma_{1,1}^0 = \varepsilon | b \quad \gamma_{1,2}^0 = a \quad \gamma_{2,1}^0 = a \quad \gamma_{2,2}^0 = \varepsilon | b$$

$$\gamma_{1,1}^1 = \gamma_{1,1}^0 | \gamma_{1,1}^0 (\gamma_{1,1}^0)^* \gamma_{1,1}^0 = \varepsilon | b | (\varepsilon | b)(\varepsilon | b)^*(\varepsilon | b) = b^*$$

$$\gamma_{1,2}^1 = \gamma_{1,2}^0 | \gamma_{1,1}^0 (\gamma_{1,1}^0)^* \gamma_{1,2}^0 = a | (\varepsilon | b)(\varepsilon | b)^* a = b^* a$$

$$\gamma_{2,1}^1 = \gamma_{2,1}^0 | \gamma_{2,1}^0 (\gamma_{1,1}^0)^* \gamma_{1,1}^0 = a | a(\varepsilon | b)^*(\varepsilon | b) = ab^*$$

$$\gamma_{2,2}^1 = \gamma_{2,2}^0 | \gamma_{2,1}^0 (\gamma_{1,1}^0)^* \gamma_{1,2}^0 = \varepsilon | b | a(\varepsilon | b)^* a = \varepsilon | b | ab^* a$$

$$\gamma_{1,1}^2 = \gamma_{1,1}^1 | \gamma_{1,2}^1 (\gamma_{2,2}^1)^* \gamma_{2,1}^1 = b^* | b^* a(\varepsilon | b | ab^* a)^* ab^*$$

Wozu sind reguläre Ausdrücke in der Praxis nützlich?

- **Suchen und Ersetzen** in Editoren
(Ausprobieren mit `vi`, `emacs`, ...)
- **Pattern-Matching** und Verarbeitung großer Texte und Datenmengen,
z.B., beim Data-Mining
(Tools: Stream-Editor `sed`, `awk`, ...)
- **Übersetzung** von Programmiersprachen:
Lexikalische Analyse – Umwandlung einer Folge von Zeichen (das Programm) in eine Folge von Tokens, in der bereits die Schlüsselwörter, Bezeichner, Daten, etc. identifiziert sind.
(Tools: `lex`, `flex`, ...), siehe Vorlesung *Compilerbau* (dort wird auch eine effizientere Version der Umwandlung von regulären Ausdrücken in NFAs behandelt).

Definition (Abgeschlossenheit)

Gegeben sei eine Menge M und ein binärer Operator $\otimes: M \times M \rightarrow M$. Man sagt, eine Menge $M' \subseteq M$ ist unter \otimes **abgeschlossen**, wenn für zwei beliebige Elemente $m_1, m_2 \in M'$ gilt: $m_1 \otimes m_2 \in M'$.

Wir betrachten hier Abschlusseigenschaften für die Menge aller regulären Sprachen (d.h. wir setzen $M =$ Menge aller Sprachen und $M' =$ Menge aller regulären Sprachen)

Die interessante Frage ist:

Falls L_1, L_2 **regulär** sind, sind dann auch $L_1 \cup L_2, L_1 \cap L_2, L_1 L_2, \overline{L_1} = \Sigma^* \setminus L_1$ (Komplement) und L_1^* **regulär**?

Kurze Antwort: Die regulären Sprachen sind unter allen diesen Operationen abgeschlossen.

Warum sind Abschlusseigenschaften interessant?

Sie sind vor allem dann interessant, wenn sie **konstruktiv** verwirklicht werden können, das heißt, wenn man – gegeben Automaten für L_1 und L_2 – auch einen Automaten beispielsweise für den Schnitt von L_1 und L_2 konstruieren kann.

Damit hat man dann mit Automaten eine **Datenstruktur für unendliche Sprachen**, die man maschinell weiterverarbeiten kann.

Satz (Abschluss unter Vereinigung)

Wenn L_1 und L_2 reguläre Sprachen sind, dann ist auch $L_1 \cup L_2$ regulär.

Beweis:

Den Automaten für $L_1 \cup L_2$ kann man mit der selben Methode bauen wie den Automaten für $L(\alpha|\beta)$ bei der Umwandlung von regulären Ausdrücken in NFAs (Folien 101). □

Satz (Abschluss unter Komplement)

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

Bemerkung: bei Bildung des Komplements muss immer festgelegt werden, bezüglich welcher Obermenge das Komplement gebildet werden soll. Hier ist das die Menge Σ^* aller Wörter über dem Alphabet Σ , das gerade betrachtet wird.

Beweis:

Aus einem DFA $M = (Z, \Sigma, \delta, z_0, E)$ für L gewinnt man leicht einen DFA M' für \bar{L} indem man die End- und Nicht-Endzustände vertauscht. D.h. $M' = (Z, \Sigma, \delta, z_0, Z \setminus E)$.

Dann gilt:

$$w \in \bar{L} \iff w \notin T(M) \iff \hat{\delta}(z_0, w) \notin E \iff \hat{\delta}(z_0, w) \in Z \setminus E \iff w \in T(M').$$

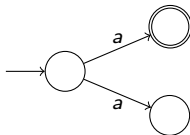


Abschlusseigenschaften

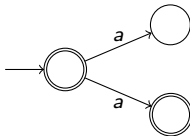
Vorsicht: In dem Beweis auf der vorherigen Folie ist es wichtig, dass M ein DFA ist.

Vertauscht man in einem NFA End- und Nicht-Endzustände, so erhält man im Allgemeinen **keinen** NFA für das Komplement.

Beispiel: Betrachte den folgenden NFA für die Sprache $\{a\} \subseteq \{a\}^*$.



Durch Vertauschen von End- und Nicht-Endzustände erhält man einen NFA für $\{\epsilon, a\} \neq \{a\}^* \setminus \{a\}$:



Will man einen NFA M komplementieren (also einen NFA für $\Sigma^* \setminus T(M)$ konstruieren), so ist die im Wesentlichen beste Methode folgende:

- 1 Konstruiere mittels der Potenzmengenkonstruktion einen DFA M' mit $T(M') = T(M)$.
- 2 Vertauschen von End- und Nicht-Endzuständen in M' liefert einen DFA (und damit auch einen NFA) M'' mit $T(M'') = \Sigma^* \setminus T(M') = \Sigma^* \setminus T(M)$.

Satz (Abschluss unter Produkt/Konkatenation)

Wenn L_1 und L_2 reguläre Sprachen sind, dann ist auch L_1L_2 regulär.

Beweis:

Den Automaten für L_1L_2 kann man mit der selben Methode bauen wie den Automaten für $L(\alpha\beta)$ bei der Umwandlung von regulären Ausdrücken in NFAs (Folien 100). □

Satz (Abschluss unter der Stern-Operation)

Wenn L eine reguläre Sprache ist, dann ist auch L^* regulär.

Beweis:

Den Automaten für L^* kann man mit der selben Methode bauen wie den Automaten für $L((\alpha)^*)$ bei der Umwandlung von regulären Ausdrücken in NFAs (Folien 105). □

Satz (Abschluss unter Schnitt)

Wenn L_1 und L_2 reguläre Sprachen sind, dann ist auch $L_1 \cap L_2$ regulär.

Beweis 1:

Es gilt $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ und wir wissen bereits, dass reguläre Sprachen unter Komplement und Vereinigung abgeschlossen sind. \square

Durch das Komplementieren entsteht im obigen Beweis 1 ein sehr großer Automat für $L_1 \cap L_2$.

Beweis 2:

Es gibt noch eine andere direktere Konstruktion. Dabei werden die zwei Automaten für L_1 und L_2 miteinander synchronisiert und quasi “parallelgeschaltet”. Dies erfolgt durch das Bilden des Kreuzprodukts.

Seien $M_1 = (Z_1, \Sigma, \delta_1, S_1, E_1)$, $M_2 = (Z_2, \Sigma, \delta_2, S_2, E_2)$ NFAs mit $T(M_1) = L_1$ und $T(M_2) = L_2$. Dann akzeptiert der folgende NFA M die Sprache $L_1 \cap L_2$:

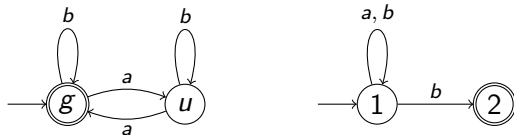
$$M = (Z_1 \times Z_2, \Sigma, \delta, S_1 \times S_2, E_1 \times E_2),$$

wobei $\delta((z_1, z_2), a) = \{(z'_1, z'_2) \mid z'_1 \in \delta_1(z_1, a), z'_2 \in \delta_2(z_2, a)\}$.

M akzeptiert ein Wort w genau dann, wenn sowohl M_1 als auch M_2 das Wort w akzeptieren. □

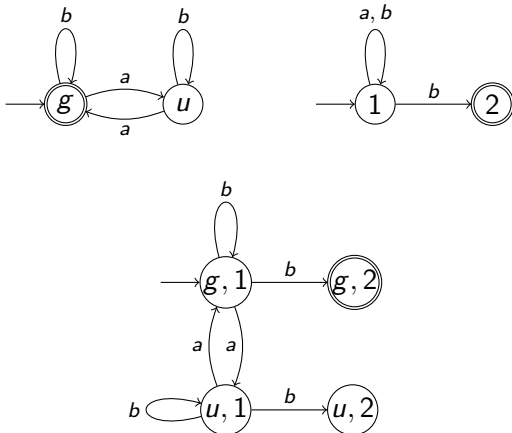
Beispiel für ein Kreuzprodukt:

Bilde das Kreuzprodukt der folgenden zwei Automaten:



Beispiel für ein Kreuzprodukt:

Bilde das Kreuzprodukt der folgenden zwei Automaten:



Weitere wichtige Fragen

- Wie kann man zeigen, dass eine Sprache **nicht** regulär ist?

Beispiel: Die Sprache $\{a^n b^n c^n \mid n \geq 1\}$, die als Beispiel auftauchte, scheint nicht regulär zu sein. Wie kann man das zeigen?

- Wenn eine Sprache regulär ist, wie groß ist dann der kleinste Automat, der die Sprache akzeptiert?

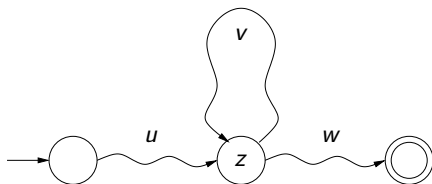
Gibt es überhaupt **den** kleinsten Automaten?

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

Idee: Man versucht auszunutzen, dass eine reguläre Sprache von einem Automat mit **endlich** vielen Zuständen akzeptiert werden muss.

Das bedeutet auch: wenn ein Wort $x \in L$ ausreichend lang ist, so besucht man damit beim Durchlauf durch den Automaten mindestens einen Zustand z zweimal.

Das Pumping-Lemma



Die dadurch entstehende Schleife kann nun mehrfach (oder gar nicht) durchlaufen werden, dadurch wird das Wort $x = uvw$ “aufgepumpt” und man stellt fest, dass uw , uv^2w , uv^3w , ... auch in L liegen müssen.

Bemerkung: Es gilt $v^i = \underbrace{v \dots v}_{i\text{-mal}}$.

Außerdem kann man für u , v , w folgende Eigenschaften verlangen, wobei n die Anzahl der Zustände des Automaten ist.

- 1 $|v| \geq 1$: Die Schleife ist auf jeden Fall nicht trivial, d.h. sie enthält mindestens einen Übergang.
- 2 $|uv| \leq n$ = Anzahl der Zustände des NFA: Spätestens nach n Alphabetsymbolen wird der Zustand z das zweite Mal erreicht.

Das Pumping-Lemma

Satz (Pumping-Lemma, uvw -Theorem)

Sei L eine reguläre Sprache. Dann gibt es eine Zahl n , so dass sich alle Wörter $x \in L$ mit $|x| \geq n$ zerlegen lassen in $x = uvw$, so dass folgende Eigenschaften erfüllt sind:

- 1 $|v| \geq 1$,
- 2 $|uv| \leq n$ und
- 3 für alle $i \geq 0$ gilt $uv^i w \in L$.

Dabei ist n die Anzahl der Zustände eines Automaten, der L erkennt.

Dieses Lemma spricht jedoch nicht über Automaten, sondern nur über die Eigenschaften der Sprache. Daher ist es dazu geeignet, Aussagen über Nicht-Regularität zu machen.

Das Pumping-Lemma

Beweis des Pumping-Lemmas:

Sei L eine reguläre Sprache.

Sei $M = (Z, \Sigma, \delta, S, E)$ ein NFA mit $L = T(M)$, sei $n = |Z|$.

Sei nun x ein beliebiges Wort mit $x \in L = T(M)$ und $|x| \geq n$, d.h. $x = a_1 a_2 \cdots a_m$ mit $m \geq n$ und $a_1, a_2, \dots, a_m \in \Sigma$.

Da $x \in T(M)$, existieren Zustände $z_0, z_1, \dots, z_m \in Z$ mit

$$z_0 \in S, \quad z_j \in \delta(z_{j-1}, a_j) \text{ für } 1 \leq j \leq m, \quad z_m \in E.$$

Wegen $|Z| = n$ existieren $0 \leq j < k \leq n$ mit $z_j = z_k$ (Schubfachprinzip).

Sei $u = a_1 \cdots a_j$, $v = a_{j+1} \cdots a_k$ und $w = a_{k+1} \cdots a_m$.

Dann gilt:

- $|v| = k - (j + 1) + 1 = k - j > 0$ und $|uv| = k \leq n$
- für alle $i \geq 0$: $z_m \in \widehat{\delta}(\{z_0\}, uv^i w)$ und damit $uv^i w \in T(M) = L$,



Das Pumping-Lemma

Wie kann man das Pumping-Lemma nutzen, um zu zeigen, dass eine Sprache nicht regulär ist?

Aussage des Pumping-Lemmas mit logischen Operatoren:

L regulär

→

$\exists n : \forall x \in L \text{ mit } |x| \geq n :$

$\exists u, v, w \text{ mit } |v| \geq 1, |uv| \leq n, x = uvw \text{ und } \forall i : uv^i w \in L$

Das ist logisch äquivalent zu

$\forall n : \exists x \in L \text{ mit } |x| \geq n :$

$\forall u, v, w \text{ mit } |v| \geq 1, |uv| \leq n \text{ und } x = uvw : \exists i : uv^i w \notin L$

→ L ist nicht regulär

Beachte hierfür: $A \rightarrow B \equiv \neg B \rightarrow \neg A$ und $\neg \forall x \exists y F \equiv \exists x \forall y \neg F$

“Kochrezept” für das Pumping-Lemma

Gegeben sei eine Sprache L .

Beispiel: $\{a^k b^k \mid k \geq 0\}$

Wir wollen zeigen, dass sie nicht regulär ist.

- 1 Nimm eine **beliebige** Zahl n . Diese Zahl darf nicht speziell gewählt werden (es muss eine beliebige Zahl sein).
- 2 Wähle ein geeignetes Wort $x \in L$ mit $|x| \geq n$. Damit das Wort auch wirklich mindestens die Länge n hat, empfiehlt es sich, dass n (beispielsweise als Exponent) im Wort auftaucht.

Beispiel: $x = a^n b^n$

“Kochrezept” für das Pumping-Lemma

- ③ Betrachte nun **alle** möglichen Zerlegungen $x = uvw$ mit den Einschränkungen $|v| \geq 1$ und $|uv| \leq n$.

Beispiel: Aus $uvw = a^n b^n$, $|v| \geq 1$ und $|uv| \leq n$ folgt, dass $j \geq 0$ und $\ell \geq 1$ existieren mit:

$$u = a^j, v = a^\ell \text{ und } w = a^m b^n \text{ mit } j + \ell + m = n$$

- ④ Wähle für **jede** dieser Zerlegungen ein i (das kann jedes Mal ein anderes i sein), so dass $uv^i w \notin L$.

In vielen Fällen sind $i = 0$ und $i = 2$ eine gute Wahl.

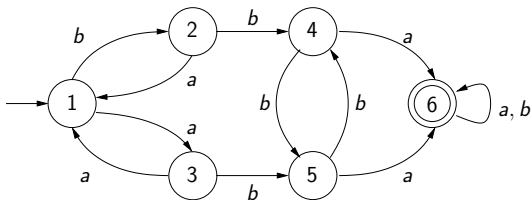
Beispiel: wähle $i = 2$, dann gilt $uv^2w = a^{j+2\ell+m}b^n \notin L$, da $j + 2\ell + m = n + \ell \neq n$ wegen $\ell \geq 1$.

Wir beschäftigen uns nun mit folgenden Fragen:

- Gibt es zu jeder Sprache immer **den** kleinsten deterministischen/nicht-deterministischen Automat?
- Kann man direkt aus der Sprache die Anzahl der Zustände des minimalen Automaten ablesen?
- Wie bestimmt man den minimalen Automat?

Äquivalenzrelationen und Minimalautomat

Betrachte den
folgenden DFA M :



Feststellung: für die Zustände 4, 5 gilt

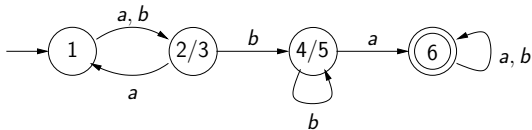
- mit einem Wort, das ein a enthält, landet man von dort aus immer im Zustand 6 (Endzustand)
- mit einem Wort, das kein a enthält, landet man von dort aus immer im Zustand 4 bzw. 5 (kein Endzustand)

Daraus folgt: 4 und 5 sind **erkenntnisäquivalent** und können zu einem Zustand verschmolzen werden.

Äquivalenzrelationen und Minimalautomat

Ebenso: die Zustände 2 und 3 sind **erkenntnisäquivalent**

Entstehender Automat M' :



Jetzt sind keine Zustände mehr erkenntnisäquivalent und sie können daher nicht weiter verschmolzen werden.

\rightsquigarrow Der Automat M' ist **minimal** für diese Sprache.

Definition (Erkennungsäquivalenz)

Gegeben sei ein DFA $M = (Z, \Sigma, \delta, q_0, E)$.

Zwei Zustände $z_1, z_2 \in Z$ heißen **erkennungsäquivalent** genau dann, wenn für jedes Wort $w \in \Sigma^*$ gilt:

$$\hat{\delta}(z_1, w) \in E \iff \hat{\delta}(z_2, w) \in E.$$

Die Relation $\{(z_1, z_2) \in Z \times Z \mid z_1 \text{ und } z_2 \text{ sind erkennungsäquivalent}\}$ ist eine **Äquivalenzrelation** auf der Zustandsmenge Z .

Äquivalenzrelation werden im Modul *Diskrete Mathematik für Informatiker* behandelt.

Eine binäre Relation $R \subseteq A \times A$ ist eine **Äquivalenzrelation**, falls gilt:

- R ist **reflexiv**: für alle $a \in A$ gilt $(a, a) \in R$.
- R ist **symmetrisch**: für alle $a, b \in A$ gilt: wenn $(a, b) \in R$, dann auch $(b, a) \in R$.
- R ist **transitiv**: für alle $a, b, c \in A$ gilt: wenn $(a, b) \in R$ und $(b, c) \in R$, dann auch $(a, c) \in R$.

Häufig schreibt man $a R b$ anstatt $(a, b) \in R$ (Infixschreibweise)

Einschub Äquivalenzrelation

Für $x \in A$ ist $[x] = \{y \in A \mid x R y\}$ die **Äquivalenzklasse** von x .

Manchmal schreibt man auch $[x]_R$ um klar zu machen, dass es sich um eine Äquivalenzklasse bezüglich der Äquivalenzrelation R handelt.

Wenn aber klar ist, um welche Äquivalenzrelation R es geht, schreiben wir nur $[x]$.

Beachte:

- Es gilt stets $x \in [x]$.
- $x R y$ genau dann, wenn $[x] = [y]$.

Die Äquivalenzklassen von R bilden eine **Partition** von A , d.h. jedes Element von A gehört zu genau einer Äquivalenzklasse.

Äquivalenzrelationen und Minimalautomat

Jedem Wort $x \in \Sigma^*$ kann man in einem DFA einen eindeutigen Zustand $z = \hat{\delta}(z_0, x)$ zuordnen. Daher kann die Definition der Erkennungsäquivalenz auf Wörter aus Σ^* und Sprachen (anstatt Automaten) ausgedehnt werden.

Definition (Myhill-Nerode-Äquivalenz)

Gegeben sei eine Sprache L und Wörter $x, y \in \Sigma^*$.

Wir definieren eine Äquivalenzrelation R_L mit $x R_L y$ genau dann wenn

$$\forall w \in \Sigma^* (xw \in L \iff yw \in L).$$

Für eine reguläre Sprache L besteht folgender Zusammenhang zwischen der Myhill-Nerode-Äquivalenz R_L und dem Begriff der Erkennungsäquivalenz:

Äquivalenzrelationen und Minimalautomat

Lemma 2

Sei $M = (Z, \Sigma, \delta, z_0, E)$ ein DFA und $L = T(M) \subseteq \Sigma^*$. Dann gilt für alle Wörter $x, y \in \Sigma^*$:

$$x R_L y \iff \hat{\delta}(z_0, x) \text{ und } \hat{\delta}(z_0, y) \text{ sind erkenntungsäquivalent}$$

Beweis: Es gilt

$$\begin{aligned} x R_L y &\iff \forall w \in \Sigma^* (xw \in L \iff yw \in L) \\ &\iff \forall w \in \Sigma^* (xw \in T(M) \iff yw \in T(M)) \\ &\iff \forall w \in \Sigma^* (\hat{\delta}(z_0, xw) \in E \iff \hat{\delta}(z_0, yw) \in E) \\ &\iff \forall w \in \Sigma^* (\hat{\delta}(\hat{\delta}(z_0, x), w) \in E \iff \hat{\delta}(\hat{\delta}(z_0, y), w) \in E) \\ &\iff \hat{\delta}(z_0, x) \text{ und } \hat{\delta}(z_0, y) \text{ sind erkenntungsäquivalent} \end{aligned}$$



Äquivalenzrelationen und Minimalautomat

Bemerkungen:

- Die Myhill-Nerode-Äquivalenz R_L ist für jede Sprache L definiert, nicht nur für reguläre Sprachen.
- Aus $x R_L y$ folgt: $x \in L \Leftrightarrow y \in L$.
Für jede Äquivalenzklasse $[x]$ gilt somit: $[x] \subseteq L$ oder $[x] \cap L = \emptyset$

Häufiger Fehler: Oft wird gedacht, dass $x R_L y$ genau dann gilt, wenn $\forall w \in \Sigma^* (xw \in L \text{ und } yw \in L)$.

Das ist aber **falsch**!

Die Definition von R_L kann man auch wie folgt schreiben:
 $x R_L y$ gilt genau dann, wenn für alle Wörter $w \in \Sigma^*$

- $(xw \in L \text{ und } yw \in L)$ **oder**
- $(xw \notin L \text{ und } yw \notin L)$ gilt.

Äquivalenzrelationen und Minimalautomat

Beispiel 1 für Myhill-Nerode-Äquivalenz: Gegeben sei die Sprache

$$L = \{w \in \{a, b\}^* \mid \#_a(w) \text{ gerade}\}.$$

Es gibt folgende Äquivalenzklassen für R_L :

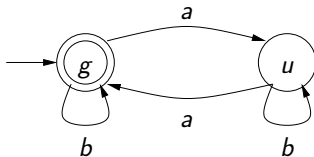
- $[\varepsilon] = \{w \in \{a, b\}^* \mid \#_a(w) \text{ gerade}\} = L$
(Äquivalenzklasse von ε)
- $[a] = \{w \in \{a, b\}^* \mid \#_a(w) \text{ ungerade}\} = \{a, b\}^* \setminus L$
(Äquivalenzklasse von a)

Die Wörter ε und aa sind äquivalent, denn:

- Wird an beide ein Wort mit gerade vielen a 's angehängt, so bleiben sie beide in der Sprache.
- Wird an beide ein Wort mit ungerade vielen a 's angehängt, so fallen sie beide aus der Sprache heraus.

Äquivalenzrelationen und Minimalautomat

DFA für $\{w \in \{a, b\}^* \mid \#_a(w) \text{ gerade}\}$:



Beispiel 2 für Myhill-Nerode-Äquivalenz: Gegeben sei die Sprache

$$L = \{w \in \{a, b, c\}^* \mid \text{das Teilwort } abc \text{ kommt in } w \text{ nicht vor}\}.$$

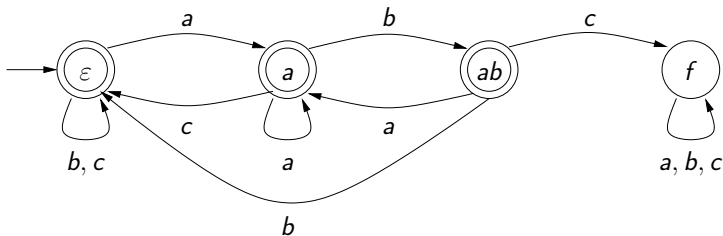
Es gibt folgende Äquivalenzklassen für R_L :

- $[\varepsilon] = \{w \in \{a, b, c\}^* \mid w \text{ endet nicht auf } a \text{ oder } ab \text{ und enthält } abc \text{ nicht}\}$
- $[a] = \{w \in \{a, b, c\}^* \mid w \text{ endet auf } a \text{ und enthält } abc \text{ nicht}\}$
- $[ab] = \{w \in \{a, b, c\}^* \mid w \text{ endet auf } ab \text{ und enthält } abc \text{ nicht}\}$
- $[abc] = \{w \in \{a, b, c\}^* \mid w \text{ enthält } abc\}$ (Fangzustand)

Die Wörter a und ab sind nicht äquivalent, denn wird an beide ein c angehängt, so ist ac noch in L , abc ist es aber nicht.

Äquivalenzrelationen und Minimalautomat

DFA für $\{w \in \{a, b, c\}^* \mid \text{das Teilwort } abc \text{ kommt in } w \text{ nicht vor}\}$:



Einschub zu Äquivalenzrelationen: Sei $R \subseteq A \times A$ eine Äquivalenzrelation auf der Menge A .

Der **Index** $\text{index}(R)$ von R ist die Anzahl der Äquivalenzklassen von R (kann unendlich sein): $\text{index}(R) = |\{[x] \mid x \in A\}| \in \mathbb{N} \cup \{\infty\}$.

Beispiel: Auf den ganzen Zahlen \mathbb{Z} definiert man für eine natürliche Zahl $k \geq 2$ die Äquivalenzrelation \equiv_k durch $a \equiv_k b$ (a kongruent b modulo k) genau dann, wenn ein $q \in \mathbb{Z}$ mit $a - b = q \cdot k$ existiert (siehe Vorlesung DMI). Dann gilt $\text{index}(\equiv_k) = k$.

Einschub Äquivalenzrelationen

Beobachtung: Seien R und S Äquivalenzrelationen auf der gleichen Menge A . Wenn $R \subseteq S$ (d.h. aus $a R b$ folgt $a S b$) dann gilt $\text{index}(S) \leq \text{index}(R)$ (es gelte dabei $x \leq \infty$ für $x \in \mathbb{N} \cup \{\infty\}$).

Begründung: Sei $[A]_R$ ($[A]_S$) die Menge der Äquivalenzklassen von R (S).

Wir definieren eine Abbildung $f : [A]_R \rightarrow [A]_S$ durch die Vorschrift $f([a]_R) = [a]_S$.

Vorsicht: Ist hierdurch $f([a]_R)$ eindeutig definiert?

Der Wert $f([a]_R)$ darf nicht davon abhängen, welchen Repräsentanten wir für die Äquivalenzklasse $[a]_R$ wählen.

Genauer: Wir müssen $[a]_R = [b]_R \implies [a]_S = [b]_S$ zeigen:

$$[a]_R = [b]_R \iff a R b \implies a S b \iff [a]_S = [b]_S$$

Äquivalenzrelationen und Minimalautomat

Natürlich ist f auch surjektiv: Jede Äquivalenzklasse $[a]_S$ wird getroffen:
 $f([a]_R) = [a]_S$.

Nun gilt für beliebige Mengen X und Y : $|X| \geq |Y|$ genau dann, wenn es eine surjektive Abbildung $f : X \rightarrow Y$ gibt (dies ist in der Tat die Definition von $|X| \geq |Y|$).

Also gilt in unserer Situation: $\text{index}(R) = |[A]_R| \geq |[A]_S| = \text{index}(S)$.

Einer der berühmtesten Sätze aus der Automatentheorie ist die folgende Charakterisierung der regulären Sprachen:

Satz von Myhill-Nerode

Sei L eine Sprache. L ist regulär $\iff \text{index}(R_L) < \infty$

Äquivalenzrelationen und Minimalautomat

Beweis:

\implies : Sei L regulär.

Sei $M = (Z, \Sigma, \delta, z_0, E)$ ein DFA mit $T(M) = L$.

Definiere eine Äquivalenzrelation R_M auf Σ^* wie folgt:

$$x R_M y \iff \hat{\delta}(z_0, x) = \hat{\delta}(z_0, y).$$

Beachte:

- R_M ist in der Tat eine Äquivalenzrelation.
- $\text{index}(R_M) \leq |Z|$

Genauer: $\text{index}(R_M) = \text{Anzahl der Zustände, die vom Anfangszustand erreicht werden können, d.h. } \text{index}(R_M) = |\{\hat{\delta}(z_0, x) \mid x \in \Sigma^*\}|$.

Behauptung: $\forall x, y \in \Sigma^* (x R_M y \implies x R_L y)$, d.h. $R_M \subseteq R_L$.

Beweis der Behauptung:

$$\begin{aligned} x R_M y &\iff \widehat{\delta}(z_0, x) = \widehat{\delta}(z_0, y) \\ &\iff \forall w \in \Sigma^* : \widehat{\delta}(z_0, xw) = \widehat{\delta}(z_0, yw) \\ &\implies \forall w \in \Sigma^* : xw \in T(M) = L \Leftrightarrow yw \in T(M) = L \\ &\iff x R_L y \end{aligned}$$

Die Bemerkung auf Folie 150 zeigt $\text{index}(R_L) \leq \text{index}(R_M) \leq |Z| < \infty$.

Äquivalenzrelationen und Minimalautomat

\Leftarrow : Sei $\text{index}(R_L) < \infty$.

Sei $[x_1], \dots, [x_n]$ eine Auflistung aller Äquivalenzklassen von R_L .

Beachte:

- $\Sigma^* = [x_1] \cup \dots \cup [x_n]$.
- Wenn $[x] = [y]$, dann $[xa] = [ya]$ für alle $a \in \Sigma$:

$$\begin{aligned}[x] = [y] &\iff x R_L y \\ &\iff \forall w \in \Sigma^* (xw \in L \iff yw \in L) \\ &\implies \forall w \in \Sigma^+ (xw \in L \iff yw \in L) \\ &\iff \forall a \in \Sigma \forall w \in \Sigma^* (xaw \in L \iff yaw \in L) \\ &\iff \forall a \in \Sigma (xa R_L ya) \\ &\iff \forall a \in \Sigma [xa] = [ya]\end{aligned}$$

Äquivalenzrelationen und Minimalautomat

Wir definieren nun den DFA (den sogenannten
Äquivalenzklassenautomaten für L)

$$M_L = (\{[x_1], \dots, [x_n]\}, \Sigma, \delta_L, [\varepsilon], \{[w] \mid w \in L\}),$$

wobei $\delta_L([x_i], a) = [x_i a]$ für alle $1 \leq i \leq n$ und $a \in \Sigma$.

Beachte:

- Die Menge der Endzustände $\{[w] \mid w \in L\}$ ist eine Menge von Äquivalenzklassen und daher eine Teilmenge der Zustandsmenge $\{[x_1], \dots, [x_n]\}$ (der Menge aller Äquivalenzklassen).
- Die Überföhrungsfunktion δ_L ist wohl-definiert auf Grund der Bemerkung auf der vorherigen Folie.
- Für alle $x \in \Sigma^*$ gilt: $\widehat{\delta}_L([\varepsilon], x) = [x]$.

Äquivalenzrelationen und Minimalautomat

Behauptung: $T(M_L) = L$ (dies zeigt dann, dass L regulär ist).

Beweis der Behauptung:

$$\begin{aligned}x \in T(M_L) &\iff \hat{\delta}_L([\varepsilon], x) \in \{[w] \mid w \in L\} \\&\iff [x] \in \{[w] \mid w \in L\} \\&\iff \exists w \in L : [x] = [w] \\&\iff \exists w \in L : x R_L w \\&\iff x \in L\end{aligned}$$



Äquivalenzrelationen und Minimalautomat

Mit dem Satz von Myhill-Nerode kann man auch zeigen, dass eine Sprache L **nicht regulär** ist.

Dazu muss man nur unendlich viele Wörter aus Σ^* finden, die in verschiedenen R_L -Äquivalenzklassen liegen.

Beispiel 3 für Myhill-Nerode-Äquivalenz:

Sei $L = \{a^k b^k \mid k \geq 0\}$

Betrachte die Wörter $a, aa, aaa, \dots, a^i, \dots$

Es gilt: $\neg(a^i R_L a^j)$ für $i \neq j$, denn $a^i b^i \in L$ und $a^j b^i \notin L$.

Also hat R_L unendlich viele Äquivalenzklassen und L ist nicht regulär.

Äquivalenzrelationen und Minimalautomat

Sei M ein DFA mit n Zuständen. Wir sagen, dass M ein **minimaler DFA** für die reguläre Sprache L ist, falls

- $T(M) = L$ und
- kein DFA M' mit $T(M') = L$ und weniger als n Zuständen existiert.

Betrachten wir nochmals den auf Folie 155 konstruierten DFA M_L .

Satz

Sei L regulär.

- 1 M_L ist ein minimaler DFA für L .
- 2 Sei M ein DFA mit $T(M) = L$ und in dem alle Zustände vom Anfangszustand aus erreicht werden können. Dann gilt:
 M ist minimaler DFA für $L \iff R_L = R_M$.
- 3 Falls M ein minimaler DFA für L ist, so kann man M aus M_L durch Umbenennung der Zustände bilden.

Äquivalenzrelationen und Minimalautomat

Beweis:

Sei $M = (Z, \Sigma, \delta_M, z_0, E)$ ein beliebiger DFA mit $T(M) = L$.

Sei $M_L = (\{[x_1], \dots, [x_n]\}, \Sigma, \delta_L, [\varepsilon], \{[w] \mid w \in L\})$ der Äquivalenzklassenautomat.

Für (1) müssen wir zeigen, dass M_L höchstens so viele Zustände wie M hat.

Auf Folie 153 haben wir gesehen: $\text{index}(R_L) \leq |Z|$.

Ausserdem: Anzahl der Zustände von $M_L = \text{index}(R_L)$.

Dies zeigt (1).

Äquivalenzrelationen und Minimalautomat

Angenommen in M sind alle Zustände vom Anfangszustand z_0 aus erreichbar, aber M ist dennoch nicht minimal für L .

Dann gilt $\text{index}(R_L) < |Z| = \text{index}(R_M)$
(siehe letzte Bemerkung auf Folie 152).

Also gilt $R_L \neq R_M$.

Ist andererseits M minimal für L so gilt
 $|Z| = \text{Anzahl der Zustände von } M_L = \text{index}(R_L)$.

Wegen $|Z| = \text{index}(R_L) \leq \text{index}(R_M) \leq |Z|$ (siehe Folie 153 unten) gilt
 $\text{index}(R_L) = \text{index}(R_M) < \infty$.

Mit $R_M \subseteq R_L$ (siehe Folie 153 oben) folgt $R_M = R_L$.

Dies zeigt (2).

Äquivalenzrelationen und Minimalautomat

Für (3) nehmen wir wieder an, dass M minimal für L ist.

Also gilt $R_M = R_L = R_{M_L}$ und $[x_1], \dots, [x_n]$ sind genau die Äquivalenzklassen von $R_M = R_L$.

Definiere $f : Z \rightarrow \{[x_1], \dots, [x_n]\}$ durch $f(z) = \{w \in \Sigma^* \mid \widehat{\delta}_M(z_0, w) = z\}$.

Dann ist f eine Bijektion.

Außerdem gilt:

- $f(z_0) = [\varepsilon]$ ist der Anfangszustand von M_L .
- Sei $z \in Z$ und sei $w \in \Sigma^*$ so, dass $\widehat{\delta}_M(z_0, w) = z$ und daher $f(z) = [w]$. Dann gilt:

$$f(\delta_M(z, a)) = f(\widehat{\delta}_M(z_0, wa)) = [wa] = \delta_L([w], a) = \delta_L(f(z), a)$$

$$z \in E \iff w \in L \iff f(z) = [w] \text{ ist Endzustand von } M_L$$

Äquivalenzrelationen und Minimalautomat

Dies bedeutet, dass wir M_L aus M bilden können, indem jeder Zustand $z \in Z$ in $f(z)$ umbenannt wird.

Oder umgekehrt: M entsteht aus dem Äquivalenzklassenautomat M_L indem jeder Zustand $[x_i]$ in $f^{-1}([x_i])$ umbenannt wird. □

Bemerkung: Es gibt also für eine reguläre Sprache bis auf Umbenennung von Zuständen genau einen minimalen DFA.

Der minimale DFA M_L für eine reguläre Sprache ist sozusagen ein eindeutiger Repräsentant für L .

Nächstes Ziel: Konstruiere den minimalen Automaten M_L aus einem nicht unbedingt minimalen DFA $M = (Z, \Sigma, \delta, z_0, E)$ mit $T(M) = L$.

Zunächst können wir voraussetzen, dass jeder Zustand $z \in Z$ vom Anfangszustand z_0 erreicht werden kann, d.h. $\exists x \in \Sigma^* : \widehat{\delta}(z_0, x) = z$.

Äquivalenzrelationen und Minimalautomat

Ist ein Zustand z vom Anfangszustand nicht erreichbar, so können wir z aus dem DFA entfernen ohne die akzeptierte Sprache zu verändern.

Beachte: Gibt es eine Kante von z' nach z , so ist auch z' nicht von z_0 aus erreichbar.

Es gilt:

M nicht minimal für L

$$\text{Folie 158} \quad \Longleftrightarrow R_M \subsetneq R_L \text{ (d.h. } R_M \subseteq R_L \text{ und } R_M \neq R_L)$$

$$\Longleftrightarrow \exists x, y \in \Sigma^* : (x, y) \in R_L \wedge (x, y) \notin R_M$$

$$\text{Folie 143} \quad \Longleftrightarrow \exists x, y \in \Sigma^* : \hat{\delta}(z_0, x), \hat{\delta}(z_0, y) \text{ sind erkenntungsäquivalent} \\ \wedge \hat{\delta}(z_0, x) \neq \hat{\delta}(z_0, y)$$

$$\Longleftrightarrow \exists z_1, z_2 \in Z : z_1, z_2 \text{ sind erkenntungsäquivalent und } z_1 \neq z_2$$

Für die letzte Äquivalenz verwenden wir, dass für jeden Zustand $z \in Z$ ein $x \in \Sigma^*$ mit $\hat{\delta}(z_0, x) = z$ existiert.

Äquivalenzrelationen und Minimalautomat

Lösung: wir verschmelzen in M alle erkenntungsäquivalenten Zustände.

Um herauszufinden, welche Zustände erkenntungsäquivalent sind, markieren wir alle Zustandspaare $\{z, z'\}$, die **nicht** erkenntungsäquivalent sind.

Wir schreiben hier Paare als 2-elementige Teilmengen $\{z, z'\}$, da die Reihenfolge egal ist: $\{z, z'\} = \{z', z\}$.

Zunächst sind sicherlich alle Paare $\{z, z'\}$ mit $z \in E$ und $z' \notin E$ nicht erkenntungsäquivalent, diese Paare markieren wir zu Beginn.

Angenommen für ein Paar $\{z, z'\}$ existiert ein $a \in \Sigma$, so dass $\{\delta(z, a), \delta(z', a)\}$ nicht erkenntungsäquivalent sind.

Dann sind auch $\{z, z'\}$ nicht erkenntungsäquivalent.

Diese Beobachtung erlaubt uns, weitere Paare als nicht erkenntungsäquivalent zu markieren.

Äquivalenzrelationen und Minimalautomat

Algorithmus Minimalautomat

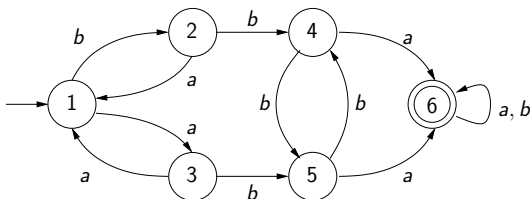
Eingabe: DFA M (Zustände, die vom Startzustand aus nicht erreichbar sind, sind bereits entfernt.)

Ausgabe: Mengen von erkenntungsäquivalenten Zuständen

- 1 Stelle eine Tabelle aller Zustandspaare $\{z, z'\}$ mit $z \neq z'$ auf.
- 2 Markiere alle Paare $\{z, z'\}$ mit $z \in E$ und $z' \notin E$.
- 3 Für jedes noch unmarkierte Paar $\{z, z'\}$ und jedes $a \in \Sigma$ teste, ob $\{\delta(z, a), \delta(z', a)\}$ bereits markiert ist. Wenn ja: markiere auch $\{z, z'\}$.
- 4 Wiederhole den vorherigen Schritt, bis sich keine Änderung in der Tabelle mehr ergibt.
- 5 Für alle jetzt noch unmarkierten Paare $\{z, z'\}$ gilt: z und z' sind erkenntungsäquivalent.

Äquivalenzrelationen und Minimalautomat

Beispiel für Durchführung des Minimierungsalgorithmus:

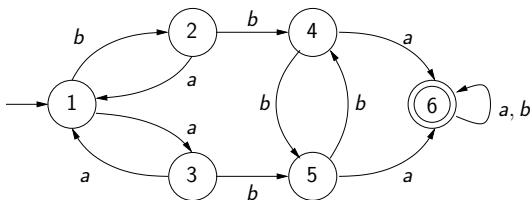


2					
3					
4					
5					
6					
	1	2	3	4	5

Erstelle eine Tabelle aller Zustandspaare.

Äquivalenzrelationen und Minimalautomat

Beispiel für Durchführung des Minimierungsalgorithmus:

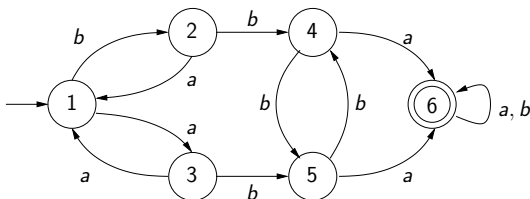


2					
3					
4					
5					
6	1	1	1	1	1
	1	2	3	4	5

(1) Markiere Paare von Endzuständen und Nicht-Endzuständen.

Äquivalenzrelationen und Minimalautomat

Beispiel für Durchführung des Minimierungsalgorithmus:

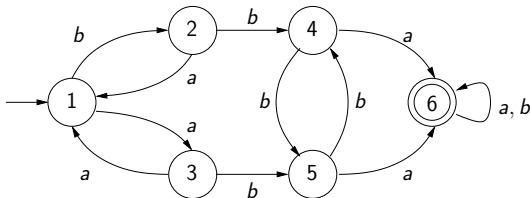


2					
3					
4		2			
5					
6	1	1	1	1	1
	1	2	3	4	5

(2) Markiere $\{2, 4\}$ wegen $\delta(2, a) = 1$, $\delta(4, a) = 6$ und $\{1, 6\}$ markiert.

Äquivalenzrelationen und Minimalautomat

Beispiel für Durchführung des Minimierungsalgorithmus:

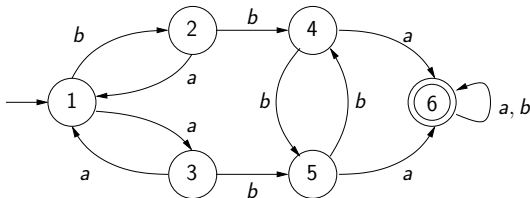


2					
3					
4		2			
5			3		
6	1	1	1	1	1
	1	2	3	4	5

(3) Markiere $\{3, 5\}$ wegen $\delta(3, a) = 1$, $\delta(5, a) = 6$ und $\{1, 6\}$ markiert.

Äquivalenzrelationen und Minimalautomat

Beispiel für Durchführung des Minimierungsalgorithmus:

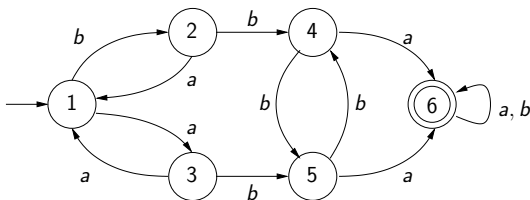


2					
3					
4		2			
5		4	3		
6	1	1	1	1	1
	1	2	3	4	5

(4) Markiere $\{2, 5\}$ wegen $\delta(2, a) = 1$, $\delta(5, a) = 6$ und $\{1, 6\}$ markiert.

Äquivalenzrelationen und Minimalautomat

Beispiel für Durchführung des Minimierungsalgorithmus:

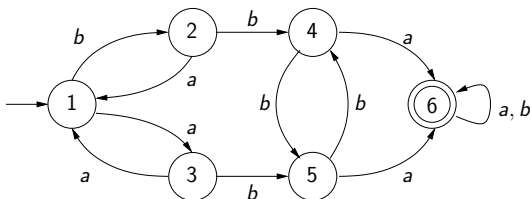


2					
3					
4		2	5		
5		4	3		
6	1	1	1	1	1
	1	2	3	4	5

(5) Markiere $\{3, 4\}$ wegen $\delta(3, a) = 1$, $\delta(4, a) = 6$ und $\{1, 6\}$ markiert.

Äquivalenzrelationen und Minimalautomat

Beispiel für Durchführung des Minimierungsalgorithmus:

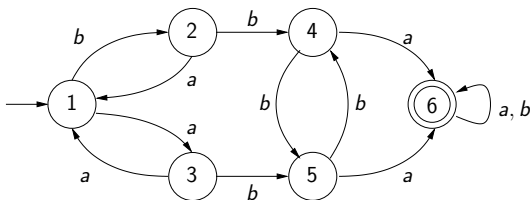


2					
3					
4		2	5		
5	6	4	3		
6	1	1	1	1	1
	1	2	3	4	5

(6) Markiere $\{1, 5\}$ wegen $\delta(1, a) = 3$, $\delta(5, a) = 6$ und $\{3, 6\}$ markiert.

Äquivalenzrelationen und Minimalautomat

Beispiel für Durchführung des Minimierungsalgorithmus:

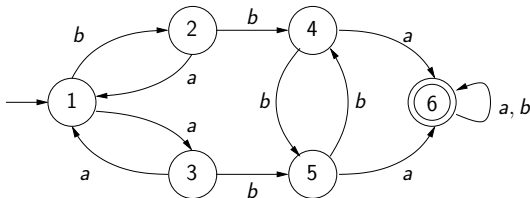


2					
3					
4	7	2	5		
5	6	4	3		
6	1	1	1	1	1
	1	2	3	4	5

(7) Markiere $\{1, 4\}$ wegen $\delta(1, a) = 3$, $\delta(4, a) = 6$ und $\{3, 6\}$ markiert.

Äquivalenzrelationen und Minimalautomat

Beispiel für Durchführung des Minimierungsalgorithmus:

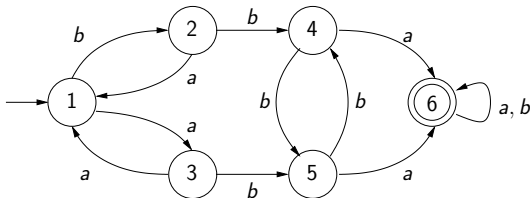


2					
3	8				
4	7	2	5		
5	6	4	3		
6	1	1	1	1	1
	1	2	3	4	5

(8) Markiere $\{1, 3\}$ wegen $\delta(1, b) = 2$, $\delta(3, b) = 5$ und $\{2, 5\}$ markiert.

Äquivalenzrelationen und Minimalautomat

Beispiel für Durchführung des Minimierungsalgorithmus:

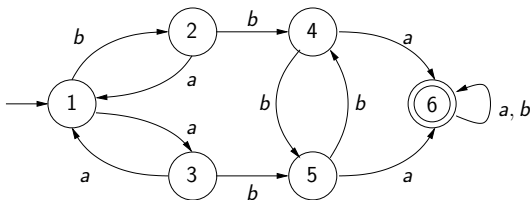


2	9				
3	8				
4	7	2	5		
5	6	4	3		
6	1	1	1	1	1
	1	2	3	4	5

(9) Markiere $\{1, 2\}$ wegen $\delta(1, b) = 2$, $\delta(2, b) = 4$ und $\{2, 4\}$ markiert.

Äquivalenzrelationen und Minimalautomat

Beispiel für Durchführung des Minimierungsalgorithmus:



2	9				
3	8				
4	7	2	5		
5	6	4	3		
6	1	1	1	1	1
	1	2	3	4	5

Die verbleibenden Zustandspaare $\{2, 3\}$ und $\{4, 5\}$ können nicht mehr markiert werden. \rightsquigarrow Sie sind erkenntungsäquivalent.

Äquivalenzrelationen und Minimalautomat

Satz (Korrektheit des Minimierungsalgorithmus)

Für einen gegebenen DFA $M = (Z, \Sigma, \delta, z_0, E)$ markiert der Minimierungsalgorithmus ein Paar $\{z, z'\}$ ($z, z' \in Z, z \neq z'$) genau dann, wenn z und z' **nicht** erkenntungsäquivalent sind.

Beweis:

(A) Falls $\{z, z'\}$ markiert wird, so sind z und z' nicht erkenntungsäquivalent.

Beweis durch Induktion über den Zeitpunkt, zu dem $\{z, z'\}$ markiert wird.

Induktionsanfang: $\{z, z'\}$ wird zu Beginn markiert, weil $z \in E$ und $z' \notin E$.

Dann sind z und z' nicht erkenntungsäquivalent.

Induktionsschluss: $\{z, z'\}$ wird irgendwann markiert, weil ein $a \in \Sigma$ existiert, so dass $\{\delta(z, a), \delta(z', a)\}$ zu einem **früheren** Zeitpunkt markiert wurde.

Äquivalenzrelationen und Minimalautomat

Nach Induktionshypothese sind $\delta(z, a)$ und $\delta(z', a)$ nicht erkenntungsäquivalent.

Dann sind auch z und z' nicht erkenntungsäquivalent.

(B) Wenn z und z' nicht erkenntungsäquivalent sind, dann wird $\{z, z'\}$ irgendwann markiert.

Seien z und z' nicht erkenntungsäquivalent.

Sei $\lambda(z, z')$ die Länge eines **kürzesten** Wortes w mit $\widehat{\delta}(z, w) \in E$ und $\widehat{\delta}(z', w) \notin E$ (oder umgekehrt).

Wir zeigen durch Induktion über $\lambda(z, z')$, dass $\{z, z'\}$ markiert wird.

Induktionsanfang: $\lambda(z, z') = 0$

Dann gilt $z \in E$ und $z' \notin E$

Also wird $\{z, z'\}$ zu Beginn markiert.

Äquivalenzrelationen und Minimalautomat

Induktionsschluss: Sei $\lambda(z, z') > 0$.

Dann gibt es ein Wort au ($a \in \Sigma$ und $u \in \Sigma^*$) mit $|au| = \lambda(z, z')$, so dass

$$\hat{\delta}(z, au) = \hat{\delta}(\delta(z, a), u) \in E, \quad \hat{\delta}(z', au) = \hat{\delta}(\delta(z', a), u) \notin E$$

(oder umgekehrt).

Dann sind auch $\delta(z, a)$ und $\delta(z', a)$ nicht erkenntungsäquivalent und $\lambda(\delta(z, a), \delta(z', a)) \leq |u| < \lambda(z, z')$.

Nach Induktionshypothese wird $\{\delta(z, a), \delta(z', a)\}$ irgendwann markiert.

Also wird auch $\{z, z'\}$ irgendwann markiert. □

Hinweise für die Durchführung des Minimierungsalgorithmus:

- Die Tabelle möglichst so aufstellen, dass jedes Paar nur genau einmal vorkommt! Also bei Zustandsmenge $\{1, \dots, n\}$: $2, \dots, n$ vertikal und $1, \dots, n - 1$ horizontal notieren.
- Bitte angeben, welche Zustände in welcher Reihenfolge und warum markiert wurden!

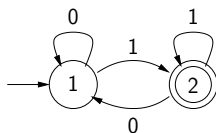
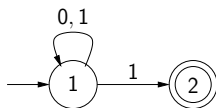
Im Buch von Schöning werden immer nur Sternchen (*) verwendet, aber daraus werden bei der Korrektur die Reihenfolge und die Gründe für die Markierung nicht ersichtlich.

Äquivalenzrelationen und Minimalautomat

Für **nicht-deterministische Automaten** kann man folgende Aussagen treffen:

- Es gibt **nicht den minimalen NFA**, sondern es kann mehrere geben.

Folgende zwei minimale NFAs erkennen $L = ((0|1)^*1)$ und haben zwei Zustände (mit nur einem Zustand kann L nicht erkannt werden).



- Gegeben ein DFA M . Dann hat ein minimaler NFA, der $T(M)$ erkennt, immer **höchstens so viel Zustände** wie M , denn M selbst ist schon ein NFA.

Außerdem: der minimale NFA kann **exponentiell kleiner** sein als der minimale DFA.

Siehe $L_k = \{x \in \{0,1\}^* \mid |x| \geq k, \text{ das } k\text{-letzte Zeichen von } x \text{ ist } 0\}$.

Wir diskutieren nun, ob es Verfahren gibt, um die folgenden Fragestellungen bzw. Probleme für reguläre Sprachen zu entscheiden. Dabei nehmen wir an, dass reguläre Sprachen als DFAs, NFAs, Grammatiken oder reguläre Ausdrücke gegeben sind.

Probleme

- **Wortproblem:** Gilt $w \in L$ für eine gegebene reguläre Sprache L und $w \in \Sigma^*$?
- **Leerheitsproblem:** Gilt $L = \emptyset$ für eine gegebene reguläre Sprache L ?
- **Endlichkeitsproblem:** Ist eine gegebene reguläre Sprache L endlich?
- **Schnittproblem:** Gilt $L_1 \cap L_2 = \emptyset$ für gegebene reguläre L_1, L_2 ?
- **Inklusionsproblem:** Gilt $L_1 \subseteq L_2$ für gegebene reguläre L_1, L_2 ?
- **Äquivalenzproblem:** Gilt $L_1 = L_2$ für gegebene reguläre L_1, L_2 ?

Wortproblem:

Sei $L \subseteq \Sigma^*$ eine reguläre Sprache, gegeben durch einen DFA $M = (Z, \Sigma, \delta, z_0, E)$ mit $T(M) = L$, und sei $w \in \Sigma^*$.

Frage: $w \in L$?

Lösung:

Sei $w = a_1 a_2 \cdots a_n$ mit $a_i \in \Sigma$.

Verfolge die Zustandsübergänge von M , die durch die Symbole a_1, \dots, a_n vorgegeben sind:

$z := z_0$

for $i := 1$ **to** n **do**

$z := \delta(z, a_i)$

endfor

if $z \in E$ **then return**(JA) **else return**(NEIN)

Leerheitsproblem:

Sei $M = (Z, \Sigma, \delta, S, E)$ ein NFA.

Frage: $T(M) \neq \emptyset$?

Lösung:

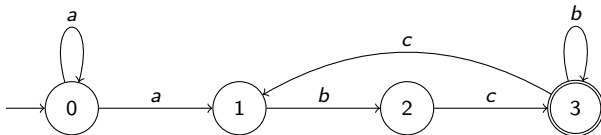
Sei $G = (Z, \rightarrow)$ der gerichtete Graph mit

$$z \rightarrow z' \iff \exists a \in \Sigma : z' \in \delta(z, a).$$

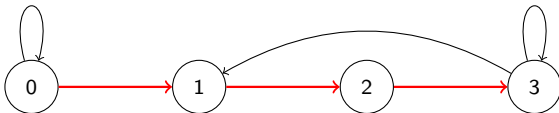
Dann gilt: $T(M) \neq \emptyset$, genau dann, wenn es in dem Graphen G einen (evtl. leeren) Pfad von einem Knoten aus S zu einem Knoten aus E gibt.

Dies kann z. B. mit Tiefen- oder Breitensuche (siehe Vorlesung *Algorithmen und Datenstrukturen*) entschieden werden.

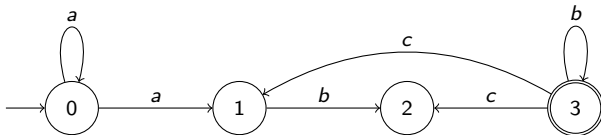
Beispiel 1: Betrachte folgenden Automaten M .



Der Automat erkennt eine nicht-leere Sprache, wie der folgende Pfad in dem Graphen G zeigt:



Beispiel 2: Der folgende Automat akzeptiert hingegen die leere Sprache, das es keinen Pfad in dem Graphen G von 0 nach 3 gibt:



Endlichkeitsproblem:

Sei $M = (Z, \Sigma, \delta, S, E)$ ein NFA.

Frage: Ist $T(M)$ endlich?

Lösung:

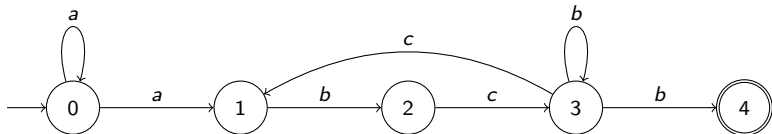
Sei G wie auf der vorherigen Folie definiert.

Dann gilt: $T(M)$ ist unendlich, genau dann, wenn es Zustände $z_0 \in S$, $z \in Z$ und $z_1 \in E$ gibt mit:

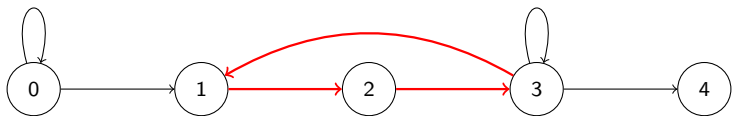
- $z_0 \rightarrow^* z$ (z ist vom Anfangszustand z_0 aus erreichbar),
- $z \rightarrow^+ z$ (von z gibt es einen Pfad zurück nach z mit mindestens einer Kante, d.h. z liegt auf einem Kreis),
- $z \rightarrow^* z_1$ (von z erreicht man den Endzustand z_1).

Dies kann wieder durch Tiefen- oder Breitensuche entschieden werden.

Beispiel: Betrachte wieder den folgenden Automaten M .



Der Automat erkennt eine unendliche Sprache: Der rote Kreis ist von 0 aus erreichbar und von dem roten Kreis kommt man zum Knoten 4 (dem Endzustand des NFA).



Schnittproblem:

Seien M_1 und M_2 NFAs.

Frage: Gilt $T(M_1) \cap T(M_2) = \emptyset$?

Lösung:

Konstruiere aus M_1 und M_2 den Produktautomaten M
($\rightsquigarrow T(M) = T(M_1) \cap T(M_2)$), siehe Folie 125.

Teste, ob $T(M) = \emptyset$ gilt.

Inklusionsproblem:

Seien M_1 und M_2 NFAs.

Frage: Gilt $T(M_1) \subseteq T(M_2)$?

Lösung: Aus M_1 und M_2 können wir einen NFA M mit $T(M) = \overline{T(M_2)} \cap T(M_1)$ konstruieren.

Es gilt: $T(M_1) \subseteq T(M_2)$ genau dann, wenn $T(M) = \emptyset$.

Äquivalenzproblem:

Seien M_1 und M_2 NFAs.

Frage: Gilt $T(M_1) = T(M_2)$?

Lösung 1:

Es gilt: $T(M_1) = T(M_2)$ genau dann, wenn $T(M_1) \subseteq T(M_2)$ und $T(M_2) \subseteq T(M_1)$.

Lösung 2:

Bestimme zu M_i ($i \in \{1, 2\}$) einen äquivalenten **minimalen DFA** N_i .

Dann gilt: $T(M_1) = T(M_2) \Leftrightarrow T(N_1) = T(N_2) \Leftrightarrow N_1$ und N_2 sind isomorph (d.h. können durch Umbenennung der Zustände ineinander überführt werden).

Effizienzbetrachtungen:

Je nachdem, in welcher Darstellung eine reguläre Sprache L gegeben ist, kann die Laufzeit der oben beschriebenen Verfahren sehr unterschiedlich ausfallen.

Beispiel: Äquivalenzproblem $L_1 = L_2$:

- L_1, L_2 gegeben als DFAs
 \rightsquigarrow Laufzeit $O(n^2)$
- L_1, L_2 gegeben als Grammatiken, reguläre Ausdrücke oder NFAs
 \rightsquigarrow Komplexität NP-hart

Das bedeutet unter anderem: Es ist nicht bekannt, ob dieses Problem in polynomieller Zeit lösbar ist.

Mehr zur Komplexitätsklasse NP und verwandten Fragestellungen \rightsquigarrow
Master Vorlesung Komplexitätstheorie

Wir behandeln nun die **kontextfreien oder Typ-2-Sprachen**.

Wiederholung: Produktionen kontextfreier Grammatiken

Bei **kontextfreien Grammatiken** haben alle Produktionen die Form $A \rightarrow w$, wobei $A \in V$ (d.h., A ist eine Variable) und $w \in (V \cup \Sigma)^+$.

Ausnahme (ε -Sonderregelung): $S \rightarrow \varepsilon$, dann darf das Startsymbol S nicht auf der rechten Seite einer Produktion vorkommen.

Betrachtete Beispielgrammatiken:

- Grammatik, die korrekt geklammerte arithmetische Ausdrücke erzeugt
- Grammatik, die Sätze der natürlichen Sprache erzeugt

Ein weiteres Beispiel: die Sprache $L = \{a^k b^k \mid k \geq 0\}$ ist kontextfrei.

Produktionen: $S \rightarrow \varepsilon \mid T, T \rightarrow ab \mid aTb$

Anwendungen kontextfreier Sprachen

Hauptanwendung: Beschreibung der **Syntax von Programmiersprachen**

Viele der hier besprochenen Techniken sind daher interessant für den Einsatz im **Compilerbau**.

Bemerkung: eine Grammatik, die eine natürliche Sprache beschreibt, kann trotz mancher kontextfreier Bestandteile nicht kontextfrei sein, da bei Sprache viele subtile Kontextabhängigkeiten berücksichtigt werden müssen.

Bisher ist es auch noch niemandem gelungen eine vollständige Grammatik aller korrekten natürlichsprachigen Sätze zu bilden.

Frage: Was ist überhaupt ein korrekter Satz?

Inhalt des Abschnitts “Kontextfreie Sprachen”

- **Normalformen** – wichtig für die Anwendung bestimmter Verfahren/Techniken ist es, eine Grammatik in eine bestimmte Normalform zu bringen.
- **Pumping-Lemma** für kontextfreie Sprachen
- **Abschlusseigenschaften** – die kontextfreien Sprachen verhalten sich hier nicht ganz so gutartig wie die regulären Sprachen.
- **Wortproblem** – und der Algorithmus, um das Wortproblem zu lösen (CYK-Algorithmus)
- **Kellerautomaten** – das Automatenmodell zu kontextfreien Sprachen

Wir beschäftigen uns zunächst noch einmal mit der “ ε -Sonderregelung”:

Die Definition für kontextfreie Grammatiken (mit ε -Sonderregelung) fordert, dass S auf keiner rechten Seite auftauchen darf, wenn $S \rightarrow \varepsilon$ als Produktion vorkommt. Außerdem dürfen keine weiteren Produktionen der Form $A \rightarrow \varepsilon$ auftauchen.

Was passiert, wenn man diese Bedingungen aufhebt und beliebige Regeln der Form $A \rightarrow \varepsilon$ erlaubt? Kann es dann passieren, dass man eine nicht-kontextfreie Sprache erzeugt?

Antwort: nein

Satz (ε -freie Grammatiken)

Gegeben sei eine Grammatik $G = (V, \Sigma, P, S)$, deren Produktionen alle von der Form $A \rightarrow w$ für $A \in V$, $w \in (V \cup \Sigma)^*$ sind.

Dann gibt es eine kontextfreie Grammatik $G' = (V, \Sigma, P', S)$, so dass:

- alle Produktionen in P' die Form $A \rightarrow w$ mit $A \in V$, $w \in (V \cup \Sigma)^+$ haben, und
- $L(G') = L(G) \setminus \{\varepsilon\}$ gilt.

Man darf also ε -Produktionen beliebig verwenden. Sie verändern nichts an der Ausdrucksmächtigkeit kontextfreier Grammatiken.

Beweis:

Sei $V_\varepsilon = \{A \in V \mid A \Rightarrow_G^* \varepsilon\}$ die Menge aller Variablen, aus denen das leere Wort abgeleitet werden kann.

Die Menge V_ε kann mittels des folgenden Algorithmus berechnet werden:

```
U := ∅  
V_ε := {A ∈ V | (A → ε) ∈ P}  
while U ≠ V_ε do  
    U := V_ε  
    V_ε := U ∪ {A ∈ V | ∃ w ∈ U+ : (A → w) ∈ P}  
endwhile
```

Dann gilt:

- Wenn eine Variable A irgendwann in die Menge V_ε aufgenommen wird, gilt $A \Rightarrow_G^* \varepsilon$.

Dies zeigt man leicht durch eine Induktion über dem Zeitpunkt t , zu dem A in die Menge V_ε aufgenommen wird.

- Wenn $A \Rightarrow_G^* \varepsilon$, dann wird irgendwann A in die Menge V_ε aufgenommen.

Dies zeigt man durch eine Induktion über die Länge ℓ der Ableitung $A \Rightarrow_G^* \varepsilon$:

Gilt $(A \rightarrow \varepsilon) \in P$, so wird A ganz am Anfang in V_ε aufgenommen.

Ansonsten gibt es eine Produktion $(A \rightarrow A_1 A_2 \cdots A_n) \in P$ mit $A_i \Rightarrow_G^* \varepsilon$ für alle $1 \leq i \leq n$, wobei die Ableitung $A_i \Rightarrow_G^* \varepsilon$ Länge $< \ell$ hat.

Nach Induktion wird jede Variable A_i ($1 \leq i \leq n$) irgendwann in V_ε aufgenommen.

Dann gilt das gleiche aber auch für A .

Normalformen

Für ein nicht-leeres Wort $w \in (V \cup \Sigma)^+$ definieren wir die Menge von Wörtern $F(w) \subseteq (V \cup \Sigma)^+$ wie folgt:

Sei $w = w_0 A_1 w_1 A_2 \cdots w_{n-1} A_n w_n$ wobei $n \geq 0$, $A_1, \dots, A_n \in V_\varepsilon$ und in dem Wort $w_0 w_1 \cdots w_n$ keine Variable aus V_ε vorkommt. Dann sei

$$F(w) = \{w_0 A_1^{e_1} w_1 A_2^{e_2} \cdots w_{n-1} A_n^{e_n} w_n \mid e_1, \dots, e_n \in \{0, 1\}\} \setminus \{\varepsilon\}.$$

wobei $A_i^0 = \varepsilon$ und $A_i^1 = A_i$.

Intuitiv: Alle Wörter, die man aus w bilden kann, indem manche (aber nicht unbedingt alle) Vorkommen von Variablen aus V_ε gelöscht werden, wobei das leere Wort nicht genommen wird.

Wir können nun die Produktionsmenge P' der ε -freien Grammatik G' wie folgt definieren:

$$P' = \{A \rightarrow w' \mid \exists w : (A \rightarrow w) \in P \text{ und } w' \in F(w)\}.$$

Behauptung: $L(G') = L(G) \setminus \{\varepsilon\}$

Beweis der Behauptung:

- $L(G') \subseteq L(G) \setminus \{\varepsilon\}$: Nach Konstruktion von G' gilt $\varepsilon \notin L(G')$.

Ausserdem gilt für jede Produktion $(A \rightarrow w') \in P'$ von G' :

$$A \Rightarrow_G^* w'.$$

Dies impliziert $L(G') \subseteq L(G) \setminus \{\varepsilon\}$.

- $L(G) \setminus \{\varepsilon\} \subseteq L(G')$: Durch Induktion über die Länge von Ableitungen zeigen wir für alle Nichtterminale $A \in V$ und Wörter $w \in \Sigma^+$:

$$A \Rightarrow_G^* w \quad \text{impliziert} \quad A \Rightarrow_{G'}^* w.$$

Gelte also $A \Rightarrow_G^* w$.

Wenn $(A \rightarrow w) \in P$, dann $(A \rightarrow w) \in P'$ und somit $A \Rightarrow_{G'}^* w$.

Angenommen die Ableitung $A \Rightarrow_G^* w$ hat Länge mindestens 2.

Es muss eine Produktion $(A \rightarrow w_0 A_1 w_1 A_2 w_2 \cdots A_n w_n) \in P$ und kürzere Ableitungen $A_i \Rightarrow_G^* u_i$ ($1 \leq i \leq n$) mit $w = w_0 u_1 w_1 u_2 w_2 \cdots u_n w_n$ geben.

Sei $J = \{i \mid 1 \leq i \leq n, u_i = \varepsilon\}$.

Sei w' das Wort, dass aus $w_0 A_1 w_1 A_2 w_2 \cdots A_n w_n$ entsteht, indem alle A_i mit $i \in J$ durch ε ersetzt werden (beachte: $A_i \in V_\varepsilon$ für alle $i \in J$).

Da $w \neq \varepsilon$ muss auch $w' \neq \varepsilon$ gelten.

Nach Definition von P' gilt $(A \rightarrow w') \in P'$.

Ausserdem gilt nach Induktion: $A_i \Rightarrow_{G'}^* u_i$ für alle $i \in \{1, \dots, n\} \setminus J$.

Insgesamt ergibt sich $A \Rightarrow_{G'}^* w$.



Der soeben bewiesene Satz zeigt insbesondere:

Satz

Sei $G = (V, \Sigma, P, S)$ eine Grammatik, deren Produktionen alle von der Form $A \rightarrow w$ für $A \in V$, $w \in (V \cup \Sigma)^*$ sind. Dann ist $L(G)$ kontextfrei.

Beweis: Konstruiere aus G eine kontextfreie Grammatik G' , so dass $L(G') = L(G) \setminus \{\varepsilon\}$ und G' keine Produktionen der Form $A \rightarrow \varepsilon$ enthält.

Falls $\varepsilon \notin L(G)$ gilt, erhalten wir $L(G') = L(G)$.

Sei nun $\varepsilon \in L(G)$.

Nimm ein neues Startsymbol S' und füge zu G' die Produktionen $S' \rightarrow \varepsilon \mid S$ hinzu.

Die resultierende Grammatik H ist kontextfrei (ε -Sonderregelung) und es gilt $L(G) = L(G') \cup \{\varepsilon\} = L(H)$. □

Beispiel: Betrachte die Grammatik G mit den folgenden Produktionen:

$$S \rightarrow aABC, \quad A \rightarrow \varepsilon \mid AA, \quad B \rightarrow \varepsilon \mid BbA, \quad C \rightarrow \varepsilon \mid CAc$$

Es gilt $V_\varepsilon = \{A, B, C\}$.

Für die (nicht-leeren) rechten Seiten der Grammatik ergibt sich:

- $F(aABC) = \{aABC, aBC, aAC, aAB, aA, aB, aC, a\}$
- $F(AA) = \{AA, A\}$
- $F(BbA) = \{BbA, bA, Bb, b\}$
- $F(CAc) = \{CAc, Ac, Cc, c\}$

Beachte: $\varepsilon \notin L(G)$. Daher erfüllt die Grammatik G' mit den folgenden Produktionen $L(G) = L(G')$:

$$S \rightarrow aABC \mid aBC \mid aAC \mid aAB \mid aA \mid aB \mid aC \mid a$$

$$A \rightarrow AA \mid A$$

$$B \rightarrow BbA \mid bA \mid Bb \mid b$$

$$C \rightarrow CAc \mid Ac \mid Cc \mid c$$

Bemerkung: Die auf Folie 190 definierte Menge kann bis zu 2^n Wörter enthalten. Dies kann dazu führen, dass die konstruierte Grammatik G' recht groß wird.

Wir betrachten nun eine weitere wichtige Normalform:

Definition (Chomsky-Normalform)

Eine kontextfreie Grammatik G mit $\varepsilon \notin L(G)$ heißt in **Chomsky-Normalform** (kurz CNF), falls alle Produktionen eine der folgenden zwei Formen haben:

$$A \rightarrow BC \quad A \rightarrow a$$

Dabei sind $A, B, C \in V$ Variablen und $a \in \Sigma$ ein Terminalsymbol.

Satz (Umwandlung in Chomsky-Normalform)

Zu jeder kontextfreien Grammatik G mit $\varepsilon \notin L(G)$ gibt es eine Grammatik G' in **Chomsky-Normalform** mit $L(G) = L(G')$.

Beweis:

Schritt 1:

Auf Grund des Satzes “ ε -freie Grammatiken” (Folie 187) können wir davon ausgehen, dass G keine Produktionen der Form $A \rightarrow \varepsilon$ hat.

Schritt 2:

Wir führen für jedes Terminalsymbol $a \in \Sigma$ eine neue Variable $A_a \notin V$ zusammen mit der Produktion $A_a \rightarrow a$ ein.

Dann können wir jedes Vorkommen von a in einer rechten Seite $\neq a$ durch A_a ersetzen.

Danach haben alle Produktionen die Form $A \rightarrow a$ oder $A \rightarrow A_1 \cdots A_n$ mit $a \in \Sigma$, $n \geq 1$ und Variablen A_1, \dots, A_n .

Schritt 3: Elimination von Kettenregeln.

Wir eliminieren nun alle Produktionen der Form $A \rightarrow B$ für Variablen A, B (Kettenregeln) wie folgt:

Für jede Variable A fügen wir die Produktion $A \rightarrow \alpha$ hinzu, falls α keine Variable ist und eine Variable B mit $A \Rightarrow^* B \rightarrow \alpha$ existiert.

Dannach können wir alle Kettenregeln weglassen.

Alle Produktionen haben nun die Form $A \rightarrow a$ oder $A \rightarrow A_1 \cdots A_n$ mit $a \in \Sigma$, $n \geq 2$ und Variablen A_1, \dots, A_n .

Schritt 4: Elimination von Produktionen der Form $A \rightarrow A_1 \cdots A_n$ mit $n \geq 3$.

Sei $A \rightarrow A_1 \cdots A_n$ eine Produktion mit $n \geq 3$.

Wir führen neue Variablen B_2, \dots, B_{n-1} ein und ersetzen die Produktion $A \rightarrow A_1 \cdots A_n$ durch die folgenden Produktionen:

$$A \rightarrow A_1 B_2, \quad B_i \rightarrow A_i B_{i+1} \quad (2 \leq i \leq n-2), \quad B_{n-1} \rightarrow A_{n-1} A_n$$



Beispiel: Sei

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

mit folgender Produktionsmenge P :

$$S \rightarrow aAb$$

$$A \rightarrow S \mid aaSc \mid \varepsilon$$

Wir wandeln G in CNF um.

Schritt 1: Wir machen G ε -frei.

Dies ergibt die Produktionen

$$S \rightarrow aAb \mid ab$$

$$A \rightarrow S \mid aaSc$$

Schritt 2: Dies ergibt die Produktionen

$$S \rightarrow A_a A A_b \mid A_a A_b$$

$$A \rightarrow S \mid A_a A_a S A_c$$

$$A_a \rightarrow a$$

$$A_b \rightarrow b$$

$$A_c \rightarrow c$$

Schritt 3: Elimination von Kettenregeln.

Die einzige Kettenregel unserer Grammatik ist $A \rightarrow S$. Deren Elimination liefert die folgenden Produktionen:

$$S \rightarrow A_a A A_b \mid A_a A_b$$

$$A \rightarrow A_a A A_b \mid A_a A_b \mid A_a A_a S A_c$$

$$A_a \rightarrow a$$

$$A_b \rightarrow b$$

$$A_c \rightarrow c$$

Schritt 4: Elimination von Regeln der Gestalt $A \rightarrow A_1 \cdots A_n$ mit $n \geq 3$.

$$S \rightarrow A_a B \mid A_a A_b$$

$$A \rightarrow A_a B \mid A_a A_b \mid A_a C$$

$$B \rightarrow A A_b$$

$$C \rightarrow A_a S A_c$$

$$A_a \rightarrow a$$

$$A_b \rightarrow b$$

$$A_c \rightarrow c$$

Schritt 4: Elimination von Regeln der Gestalt $A \rightarrow A_1 \cdots A_n$ mit $n \geq 3$.

$$S \rightarrow A_a B \mid A_a A_b$$

$$A \rightarrow A_a B \mid A_a A_b \mid A_a C$$

$$B \rightarrow A A_b$$

$$C \rightarrow A_a D$$

$$D \rightarrow S A_c$$

$$A_a \rightarrow a$$

$$A_b \rightarrow b$$

$$A_c \rightarrow c$$

Normalformen

Definition (Greibach-Normalform)

Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ mit $\varepsilon \notin L(G)$ ist in **Greibach-Normalform**, falls alle Produktionen aus P folgende Form haben:

$$A \rightarrow aB_1B_2 \dots B_k \quad \text{mit } k \geq 0$$

Dabei sind $A, B_1, \dots, B_k \in V$ Variablen und $a \in \Sigma$ ein Alphabetsymbol.

Die Greibach-Normalform garantiert, dass bei jedem Ableitungsschritt genau ein Alphabetsymbol entsteht.

Sie ist nützlich, um zu zeigen, dass Kellerautomaten (d.h., Automaten für kontextfreie Sprachen) keine ε -Übergänge brauchen.

Satz (Umwandlung in Greibach-Normalform)

Zu jeder kontextfreien Grammatik G mit $\varepsilon \notin L(G)$ gibt es eine Grammatik G' in **Greibach-Normalform** mit $L(G) = L(G')$.

Normalformen

Beweis: Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik mit $\varepsilon \notin L(G)$.

Vorüberlegung:

Angenommen in P gibt es für eine Variable A die folgenden Produktionen:

$$A \rightarrow A\alpha_1 \mid \cdots \mid A\alpha_k \mid \beta_1 \mid \cdots \mid \beta_\ell.$$

Hierbei sind $\alpha_1, \dots, \alpha_k, \beta_1, \dots, \beta_\ell \in (V \cup \Sigma)^*$ und $\beta_1, \dots, \beta_\ell$ beginnen nicht mit A .

Dann kann man mit diesen Produktionen genau die gleichen Satzformen erzeugen wie mit

$$\begin{aligned} A &\rightarrow \beta_1 \mid \cdots \mid \beta_\ell \mid \beta_1 B \mid \cdots \mid \beta_\ell B \\ B &\rightarrow \alpha_1 \mid \cdots \mid \alpha_k \mid \alpha_1 B \mid \cdots \mid \alpha_k B. \end{aligned}$$

Mit beiden Regelsätzen lassen sich alle Satzformen aus

$$(\beta_1 \mid \cdots \mid \beta_\ell)(\alpha_1 \mid \cdots \mid \alpha_k)^*$$

erzeugen.

Sei nun A_1, \dots, A_m eine beliebige Aufzählung aller Variablen von G .

Schritt 1: Mit dem Algorithmus auf der nächsten Folie formen wir G in eine äquivalente kontextfreie Grammatik um, in der für alle Produktionen der Form $A_i \rightarrow \alpha$ gilt:

$$\alpha = a\beta \text{ mit } a \in \Sigma, \beta \in V^* \text{ oder } \alpha = A_j\beta \text{ mit } j > i, \beta \in V^*$$

O.B.d.A. können wir davon ausgehen, dass G in Chomsky-Normalform ist.

```
for  $i := 1$  to  $m$  do
  for  $j := 1$  to  $i - 1$  do
    for all  $(A_i \rightarrow A_j \alpha) \in P$  do
      Seien  $A_j \rightarrow \beta_1 \mid \dots \mid \beta_n$  alle Regeln mit linker Seite =  $A_j$ .
       $P := (P \cup \{A_i \rightarrow \beta_1 \alpha \mid \dots \mid \beta_n \alpha\}) \setminus \{A_i \rightarrow A_j \alpha\}$ 
    endfor
  endfor
  if es gibt Produktionen der Form  $A_i \rightarrow A_j \alpha$  then
    Wende die Transformation aus der Vorüberlegung auf  $A_i$  an
    (dabei wird eine neue Variable  $B_i$  eingeführt).
  endif
endfor
```

Normalformen

Nach Schritt 1 sind insbesondere alle Produktionen mit linker Seite $= A_m$ von der Form $A_m \rightarrow a\alpha$ mit $a \in \Sigma, \alpha \in V^*$.

Schritt 2: Der folgende Algorithmus erzwingt, dass alle Produktionen mit linker Seite A_i rechts mit einem Terminalsymbol beginnen.

```
for  $i := m - 1$  downto 1 do  
  forall  $(A_i \rightarrow A_j\alpha) \in P$  mit  $j > i$  do  
    Seien  $A_j \rightarrow \beta_1 \mid \dots \mid \beta_n$  alle Regeln mit linker Seite  $= A_j$ .  
     $P := (P \cup \{A_i \rightarrow \beta_1\alpha \mid \dots \mid \beta_n\alpha\}) \setminus \{A_i \rightarrow A_j\alpha\}$   
  endfor  
endfor
```

Nach Schritt 2 sind alle Produktionen mit linker Seite $= A_i$ ($1 \leq i \leq m$) in Greibach-Normalform.

Aber: Die in Schritt 1 eingeführten Produktionen für die neuen Variablen B_i könnten nicht in Greibach-Normalform sein.

Sei $B_i \rightarrow A_j \alpha$ eine solche Regel, die die Greibach-Normalform verletzt.

Seien $A_j \rightarrow \beta_1 \mid \cdots \mid \beta_k$ alle Produktionen mit linker Seite $= A_j$.

Dann beginnen β_1, \dots, β_k mit Terminalsymbolen.

Ersetze $B_i \rightarrow A_j \alpha$ durch $B_i \rightarrow \beta_1 \alpha \mid \cdots \mid \beta_k \alpha$.

Nun ist die Grammatik in Greibach-Normalform. □

Normalformen

Beispiel: Sei G die Grammatik in CNF mit folgenden Produktionen:

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 \mid b$$

$$A_3 \rightarrow A_1 A_2 \mid a.$$

In Schritt 1 wird nur die Produktion $A_3 \rightarrow A_1 A_2$ im Durchlauf $i = 3$ wie folgt ersetzt:

- Bei $j = 1$: $A_3 \rightarrow A_2 A_3 A_2$
- Bei $j = 2$: $A_3 \rightarrow A_3 A_1 A_3 A_2 \mid b A_3 A_2$

Nun wird eine neue Variable B_3 eingeführt, und die Produktionen

$$A_3 \rightarrow A_3 A_1 A_3 A_2 \mid b A_3 A_2 \mid a$$

werden ersetzt durch

$$A_3 \rightarrow b A_3 A_2 B_3 \mid a B_3 \mid b A_3 A_2 \mid a$$

$$B_3 \rightarrow A_1 A_3 A_2 B_3 \mid A_1 A_3 A_2.$$

Normalformen

Wir haben also nach Schritt 1 folgende Grammatik vorliegen:

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow A_3 A_1 \mid b$$

$$A_3 \rightarrow b A_3 A_2 B_3 \mid a B_3 \mid b A_3 A_2 \mid a$$

$$B_3 \rightarrow A_1 A_3 A_2 B_3 \mid A_1 A_3 A_2.$$

Beachte: Alle Produktionen für A_3 beginnen in der Tat mit einem Terminalsymbol auf der rechten Seite.

Nach Schritt 2, Durchlauf $i = 2$:

$$A_1 \rightarrow A_2 A_3$$

$$A_2 \rightarrow b A_3 A_2 B_3 A_1 \mid a B_3 A_1 \mid b A_3 A_2 A_1 \mid a A_1 \mid b$$

$$A_3 \rightarrow b A_3 A_2 B_3 \mid a B_3 \mid b A_3 A_2 \mid a$$

$$B_3 \rightarrow A_1 A_3 A_2 B_3 \mid A_1 A_3 A_2$$

Nach Schritt 2, Durchlauf $i = 1$:

$$A_1 \rightarrow bA_3A_2B_3A_1A_3 \mid aB_3A_1A_3 \mid bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3$$

$$A_2 \rightarrow bA_3A_2B_3A_1 \mid aB_3A_1 \mid bA_3A_2A_1 \mid aA_1 \mid b$$

$$A_3 \rightarrow bA_3A_2B_3 \mid aB_3 \mid bA_3A_2 \mid a$$

$$B_3 \rightarrow A_1A_3A_2B_3 \mid A_1A_3A_2$$

Nun muss noch in den rechten Seiten der B_3 -Produktionen A_1 durch die rechten Seiten von A_1 ersetzt werden:

$A_1 \rightarrow bA_3A_2B_3A_1A_3 \mid aB_3A_1A_3 \mid bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3$

$A_2 \rightarrow bA_3A_2B_3A_1 \mid aB_3A_1 \mid bA_3A_2A_1 \mid aA_1 \mid b$

$A_3 \rightarrow bA_3A_2B_3 \mid aB_3 \mid bA_3A_2 \mid a$

$B_3 \rightarrow bA_3A_2B_3A_1A_3A_3A_2B_3 \mid aB_3A_1A_3A_3A_2B_3 \mid bA_3A_2A_1A_3A_3A_2B_3 \mid$
 $aA_1A_3A_3A_2B_3 \mid bA_3A_3A_2B_3 \mid bA_3A_2B_3A_1A_3A_3A_2 \mid$
 $aB_3A_1A_3A_3A_2 \mid bA_3A_2A_1A_3A_3A_2 \mid aA_1A_3A_3A_2 \mid bA_3A_3A_2$

Bemerkung zum leeren Wort ε : Mit Grammatiken in Chomsky-Normalform bzw. Greibach-Normalform lassen sich nur kontextfreie Sprachen L mit $\varepsilon \notin L$ erzeugen.

Hat man nun eine kontextfreie Grammatik G mit $\varepsilon \in L(G)$ vorliegen, so kann man wie folgt vorgehen:

- Konstruiere aus G eine kontextfreie Grammatik G' mit $L(G') = L(G) \setminus \{\varepsilon\}$ (siehe Satz auf Folie 187).
- Wandle G' in eine Grammatik G'' in Chomsky-Normalform bzw. Greibach-Normalform um.

Sei S das Startsymbol von G'' und seien $S \rightarrow \alpha_1 \mid \dots \mid \alpha_n$ alle Produktionen in G'' mit linker Seite $= S$.

- Nimm ein neues Startsymbol S' und füge zu G'' die Produktionen $S' \rightarrow \varepsilon \mid \alpha_1 \mid \dots \mid \alpha_n$ hinzu.

Für die resultierende Grammatik H gilt $L(G) = L(H)$, und bis auf die Produktion $S' \rightarrow \varepsilon$ sind alle Produktionen in H in Chomsky-Normalform bzw. Greibach-Normalform.

Weitgehend analog zu regulären Sprachen kann man nun ein **Pumping-Lemma** für kontextfreie Sprachen zeigen.

Die für reguläre Sprachen und endliche Automaten geltende Aussage

Jedes ausreichend lange Wort durchläuft einen Zustand des Automaten zweimal.

wird dabei ersetzt durch

Auf einem Pfad des Syntaxbaums, der die Ableitung eines ausreichend langen Wortes durch eine kontextfreie Grammatik darstellt, kommt eine Variable mindestens zweimal vor.

Pumping-Lemma

Was bedeutet hier “ausreichend langes Wort”?

Die Beantwortung dieser Frage hängt davon ab, in welcher Form sich die Grammatik befindet.

Wir nehmen an, sie befinde sich in **Chomsky-Normalform**.

Dann gilt: **Syntaxbäume** sind (bis auf die unterste Schicht der Blätter) immer **Binärbäume** (aufgrund der Produktionen der Form $A \rightarrow BC$).

Für Binärbäume gilt nun:

Lemma (Länge von Pfaden in Binärbäumen)

Sei B ein Binärbaum (d.h., jeder Knoten in B hat entweder null oder zwei Kinder) mit mindestens 2^k Blättern.

Dann hat B einen Pfad von der Wurzel zu einem Blatt, der aus mindestens k Kanten und $k + 1$ Knoten besteht.

Pumping-Lemma

Beweis: Induktion über k .

Induktionsanfang: $k = 0$.

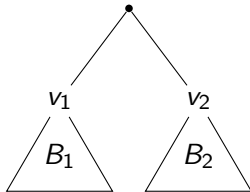
Sei B ein Binärbaum mit mindestens $2^0 = 1$ Blättern.

Dann hat B einen Pfad, der aus mindestens einem Knoten (nämlich der Wurzel) besteht.

Induktionsschluss: $k \geq 0$.

Sei B ein Binärbaum mit mindestens $2^{k+1} = 2^k + 2^k$ Blättern.

Seien v_1 und v_2 die beiden Kinder der Wurzel, und seien B_1 und B_2 die Binärbäume mit Wurzel v_1 bzw. v_2 :



Dann muss B_1 oder B_2 mindestens 2^k Blätter haben: Hätte sowohl B_1 als auch B_2 echt weniger als 2^k Blätter, dann hätte der Baum B echt weniger als $2^k + 2^k = 2^{k+1}$ Blätter.

Ohne Beschränkung der Allgemeinheit habe B_1 mindestens 2^k viele Blätter.

Nach Induktionshypothese gibt es in B_1 einen Pfad von der Wurzel v_1 zu einem Blatt mit mindestens k Kanten und $k + 1$ Knoten.

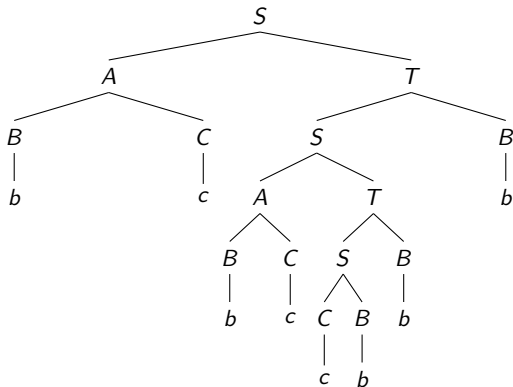
Indem wir zu diesem Pfad die Kante von der Wurzel nach v_1 hinzufügen, erhalten wir in B einen Pfad von der Wurzel zu einem Blatt mit mindestens $k + 1$ Kanten und $k + 2$ Knoten. □

Pumping-Lemma

Beispiel: Die kontextfreie Grammatik G (in CNF) bestehe aus folgenden Produktionen:

$$S \rightarrow AT \mid CB, \quad T \rightarrow SB, \quad A \rightarrow BC, \quad B \rightarrow b, \quad C \rightarrow c.$$

Betrachte folgenden Syntaxbaum:

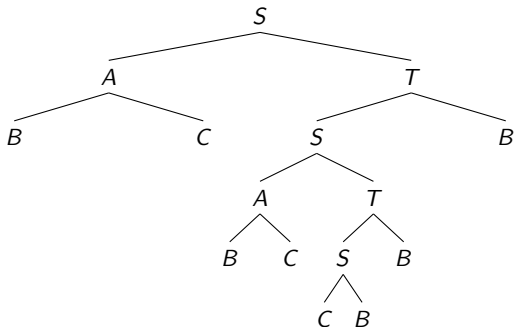


Pumping-Lemma

Beispiel: Die kontextfreie Grammatik G (in CNF) bestehe aus folgenden Produktionen:

$$S \rightarrow AT \mid CB, \quad T \rightarrow SB, \quad A \rightarrow BC, \quad B \rightarrow b, \quad C \rightarrow c.$$

Entfernen der Blätter liefert einen Binärbaum:

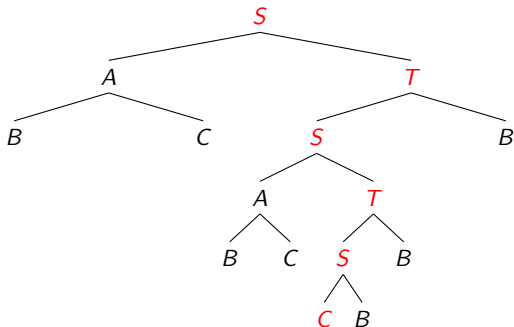


Pumping-Lemma

Beispiel: Die kontextfreie Grammatik G (in CNF) bestehe aus folgenden Produktionen:

$$S \rightarrow AT \mid CB, \quad T \rightarrow SB, \quad A \rightarrow BC, \quad B \rightarrow b, \quad C \rightarrow c.$$

Entfernen der Blätter liefert einen Binärbaum:



Pumping-Lemma

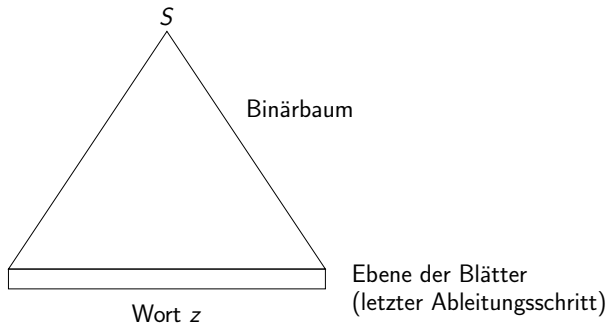
Sei nun $G = (V, \Sigma, P, S)$ eine Grammatik in Chomsky-Normalform mit $k = |V|$ vielen Variablen.

Sei weiter $z \in L(G)$.

- Wenn $|z| \geq 2^k$, dann hat jeder Syntaxbaum für z offensichtlich mindestens 2^k Blätter.
- Betrachte einen Syntaxbaum für z und entferne die mit den Terminalsymbolen beschrifteten Blätter. Dies ergibt einen Binärbaum T .
- Betrachte einen längsten Pfad in T von der Wurzel zu einem Blatt.
- Das Lemma von Folie 217 impliziert, dass dieser Pfad mindestens $k + 1 > |V|$ viele Knoten hat.
- Also kommt auf dem Pfad eine Variable A mindestens zweimal vor (wir sprechen von einem **Doppelvorkommen** im folgenden).

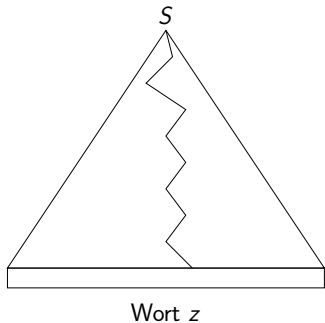
Pumping-Lemma

Syntaxbaum für ein Wort z mit $|z| \geq n = 2^k$
 n ist hier die **“Konstante des Pumping-Lemmas”**.



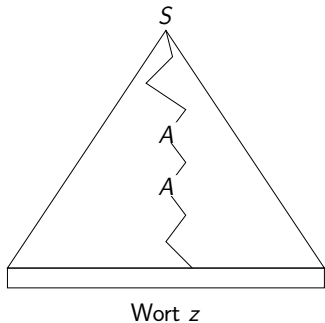
Pumping-Lemma

Ein längster Pfad hat mindestens $k + 1$ innere Knoten.



Pumping-Lemma

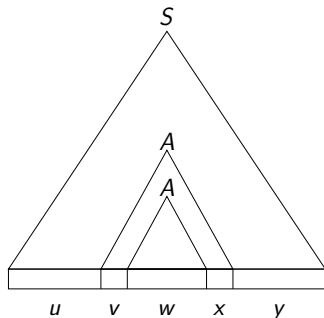
Auf diesem Pfad gibt es eine **Variable, die zweimal** auftaucht, etwa A .



Pumping-Lemma

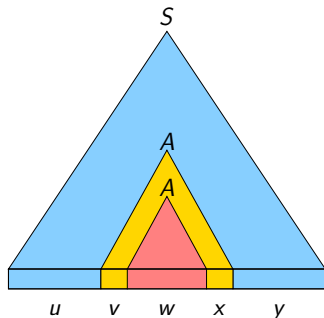
Das Wort z wird nun in **fünf Teilwörter** u, v, w, x, y aufgespalten:

- w wird aus dem unteren A abgeleitet: $A \Rightarrow^* w$
- vwx wird aus dem oberen A abgeleitet: $A \Rightarrow^* vAx \Rightarrow^* vwx$



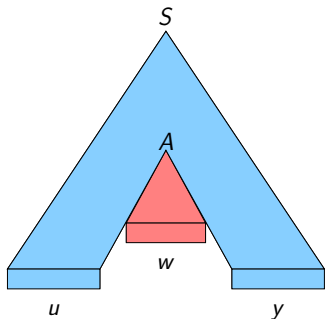
Pumping-Lemma

Damit erhält man **drei ineinander enthaltene Teil-Syntaxbäume**, die man neu zusammenstecken kann.



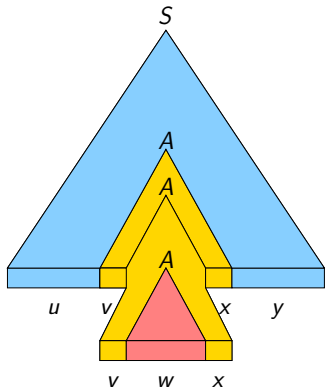
Pumping-Lemma

Durch **Weglassen des mittleren Teilbaums** erhält man einen Syntaxbaum für uw . Damit gilt: $uw \in L(G)$.



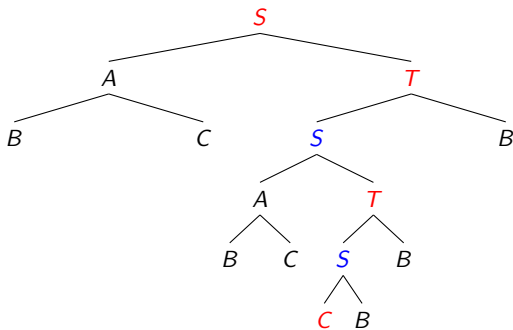
Pumping-Lemma

Durch **Verdoppeln des mittleren Teilbaums** erhält man einen Syntaxbaum für uv^2wx^2y . Damit gilt: $uv^2wx^2y \in L(G)$.



Pumping-Lemma

Am konkreten Beispiel von Folie 220.



Wir erhalten: $u = bc$, $v = bc$, $w = cb$, $x = b$, $y = b$

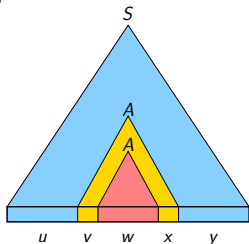
Pumping-Lemma

Außerdem kann man für v , w , x folgende Eigenschaften verlangen:

$$|vwx| \leq n = 2^k:$$

Wir können annehmen, dass wir das am **weitesten unten liegende Doppelvorkommen einer Variable** gewählt haben, d.h., das Doppelvorkommen mit der größten Tiefe.

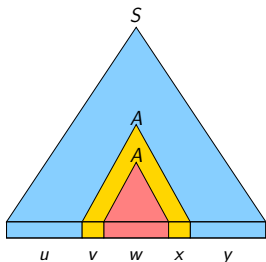
Das kann dadurch erreicht werden, dass ein Pfad maximaler Länge von unten nach oben verfolgt wird, bis ein Doppelvorkommen entdeckt wird. Demnach ist der **Abstand des oberen A zur Blattebene höchstens k** und der darunter hängende Binärbaum hat höchstens 2^k Blätter.



Pumping-Lemma

$|vx| \geq 1$:

Seien B, C die beiden **Kinder** des oberen A . Dann geht das untere A entweder aus B oder C hervor. Die jeweils andere Variable muss – da die Grammatik in **Chomsky-Normalform** ist – ein **nicht-leeres Wort** ableiten. Und dieses Wort ist ein **Teilwort von v oder von x** .



Wir haben somit den folgenden Satz bewiesen:

Satz (Pumping-Lemma, $uvwx$ -Theorem)

Sei L eine kontextfreie Sprache. Dann gibt es eine Zahl n , so dass sich alle Wörter $z \in L$ mit $|z| \geq n$ zerlegen lassen als $z = uvwxy$, so dass folgende Eigenschaften erfüllt sind:

- 1 $|vx| \geq 1$,
- 2 $|vwx| \leq n$,
- 3 für alle $i \geq 0$ gilt: $uv^iwx^iy \in L$.

Dabei geht $n = 2^k$ aus der Anzahl k der Variablen einer kontextfreien Grammatik in CNF für L hervor.

Anwendung des Pumping-Lemmas:

Wir zeigen, dass die Sprache $L = \{a^m b^m c^m \mid m \geq 1\}$ **nicht kontextfrei** ist.

- 1 Wir nehmen eine **beliebige** Zahl n an.
- 2 Wir wählen ein Wort $z \in L$ mit $|z| \geq n$. In diesem Fall eignet sich $z = a^n b^n c^n$.
- 3 Wir betrachten nun **alle** möglichen Zerlegungen $z = uvwxy$ mit den Einschränkungen $|vx| \geq 1$ und $|vwx| \leq n$.

Wegen $|vwx| \leq n$ gilt, dass vx nicht aus a 's, b 's und c 's bestehen kann, denn es kann sich nicht über den gesamten b -Block erstrecken.

- 4 Wir wählen für alle diese möglichen Zerlegungen $i = 2$ und betrachten uv^2wx^2y . Wegen der obigen Überlegungen sind nun ein oder zwei Alphabetsymbole gepumpt worden, mindestens eines jedoch nicht.

Damit ist klar, dass uv^2wx^2y nicht in L liegen kann, denn jedes Wort in L hat gleich viele a 's, b 's und c 's.

Pumping-Lemma

Man kann auch für folgende Sprachen zeigen, dass sie nicht kontextfrei sind:

$$L_1 = \{a^p \mid p \text{ ist Primzahl}\}$$

$$L_2 = \{a^n \mid n \text{ ist Quadratzahl}\}$$

$$L_3 = \{a^{2^n} \mid n \geq 0\}$$

Die Sprachen L_1 , L_2 , L_3 sind alle **unär**, d.h., sie sind Sprachen über einem einelementigen Alphabet: $L_1, L_2, L_3 \subseteq \Sigma^*$ mit $|\Sigma| = 1$

Für unäre Sprachen gilt sogar folgender Satz (ohne Beweis).

Satz (unäre kontextfreie Sprachen)

Jede kontextfreie Sprache über einem einelementigen Alphabet ist bereits regulär.

Abgeschlossenheit

Die kontextfreien Sprachen sind **abgeschlossen** unter:

- Vereinigung (L_1, L_2 kontextfrei $\Rightarrow L_1 \cup L_2$ kontextfrei)
- Produkt/Konkatenation (L_1, L_2 kontextfrei $\Rightarrow L_1 L_2$ kontextfrei)
- Stern-Operation (L kontextfrei $\Rightarrow L^*$ kontextfrei)

Die kontextfreien Sprachen sind **nicht abgeschlossen** unter:

- Schnitt
- Komplement

Abschluss unter Vereinigung

Wenn L_1 und L_2 kontextfreie Sprachen sind, dann ist auch $L_1 \cup L_2$ kontextfrei.

Begründung: Seien $G_1 = (V_1, \Sigma, P_1, S_1)$ und $G_2 = (V_2, \Sigma, P_2, S_2)$ kontextfreie Grammatiken.

Ohne Beschränkung der Allgemeinheit gelte $V_1 \cap V_2 = \emptyset$.

Sei $S \notin V_1 \cup V_2$.

Dann ist $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}, S)$ eine kontextfreie Grammatik mit $L(G) = L(G_1) \cup L(G_2)$.

Abschluss unter Produkt/Konkatenation

Wenn L_1 und L_2 kontextfreie Sprachen sind, dann ist auch L_1L_2 kontextfrei.

Begründung: Seien

$$G_1 = (V_1, \Sigma, P_1, S_1), \quad G_2 = (V_2, \Sigma, P_2, S_2)$$

kontextfreie Grammatiken. Ohne Beschränkung der Allgemeinheit gelte $V_1 \cap V_2 = \emptyset$.

Sei $S \notin V_1 \cup V_2$.

Dann ist

$$G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \rightarrow S_1S_2\}, S)$$

eine kontextfreie Grammatik mit $L(G) = L(G_1)L(G_2)$.

Abschluss unter der Stern-Operation

Wenn L eine kontextfreie Sprache ist, dann ist auch L^* kontextfrei.

Begründung: Sei

$$G_1 = (V_1, \Sigma, P_1, S_1)$$

eine kontextfreie Grammatik.

Sei $S \notin V_1$.

Dann ist

$$G = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \rightarrow \varepsilon, S \rightarrow S_1 S\}, S)$$

eine kontextfreie Grammatik mit $L(G) = L(G_1)^*$.

Kein Abschluss unter Schnitt

Es gibt kontextfreie Sprachen L_1 und L_2 , so dass $L_1 \cap L_2$ nicht kontextfrei ist.

Gegenbeispiel: Die Sprachen

$$L_1 = \{a^j b^k c^k \mid j \geq 0, k \geq 0\}$$

$$L_2 = \{a^k b^k c^j \mid j \geq 0, k \geq 0\}$$

sind beide kontextfrei (L_1 wird z.B. durch eine Grammatik mit den Produktionen $S \rightarrow aS \mid A$, $A \rightarrow \varepsilon \mid bAc$ erzeugt).

Für ihren Schnitt gilt jedoch

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

und diese Sprache ist – wie mit dem Pumping-Lemma gezeigt wurde – nicht kontextfrei.

Kein Abschluss unter Komplement

Es gibt eine kontextfreie Sprache L , so dass $\bar{L} = \Sigma^* \setminus L$ nicht kontextfrei ist.

Begründung:

Nehmen wir an, die kontextfreien Sprachen wären unter Komplement abgeschlossen und seien L_1 und L_2 kontextfrei. Wegen

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

(Regel von de Morgan) wäre dann auch $L_1 \cap L_2$ kontextfrei.

Dies ist ein Widerspruch zum Gegenbeispiel auf der vorherigen Folie.

Der CYK-Algorithmus

Wir kennen bereits ein Verfahren, mit dem man das Wortproblem für G lösen kann, wobei G eine Typ-1-, Typ-2- oder Typ-3-Grammatik sein kann (Folie 37).

Im wesentlichen: Aufzählen aller Wörter bis zu einer bestimmten Länge.

Da dieses Verfahren jedoch exponentielle Laufzeit (in der Länge des Wortes) haben kann, betrachten wir hier ein effizienteres Verfahren für kontextfreie Grammatiken: den **CYK-Algorithmus** (entwickelt von Cocke, Younger, Kasami).

Voraussetzung: die Grammatik ist in Chomsky-Normalform, alle Produktionen haben also die Form $A \rightarrow a$ oder $A \rightarrow BC$.

Idee: Gegeben sei ein Wort $x \in \Sigma^*$. Wir wollen feststellen, aus welchen Variablen es abgeleitet werden kann.

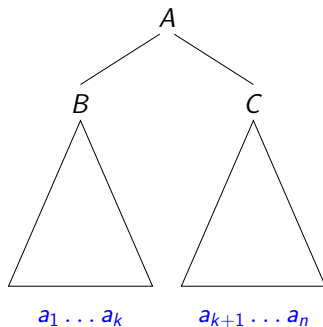
- **Möglichkeit 1:** $x = a \in \Sigma$, d.h., x besteht aus einem einzigen Alphabetsymbol.

Dann kann x nur aus Variablen A abgeleitet werden, für die es eine Produktion $A \rightarrow a$ gibt.

- **Möglichkeit 2:** $x = a_1 \cdots a_n$ mit $n \geq 2$.

In diesem Fall gilt: Zunächst muss eine Produktion $A \rightarrow BC$ angewandt werden, dann muss ein Teil $a_1 \cdots a_k$ des Wortes aus B und der andere Teil $a_{k+1} \cdots a_n$ aus C abgeleitet werden ($1 \leq k < n$).

Möglichkeit 2 läßt sich schematisch folgendermaßen darstellen:



Der CYK-Algorithmus

Es ist jedoch nicht klar, wo das Wort x geteilt werden muss, d.h., wie groß die Position k ist!

Daher: Probiere alle möglichen k 's durch. Das heißt:

Gegeben ein Wort $x = a_1 \cdots a_n$.

Für alle k mit $1 \leq k < n$ mache Folgendes:

- Bestimme die Menge V_1 aller Variablen, aus denen sich $a_1 \cdots a_k$ ableiten lässt.
- Bestimme die Menge V_2 aller Variablen, aus denen sich $a_{k+1} \cdots a_n$ ableiten lässt.
- Stelle fest, ob es Variablen A, B, C gibt mit $(A \rightarrow BC) \in P$, $B \in V_1$ und $C \in V_2$.
In diesem Fall lässt sich x aus A ableiten .

Um Mehraufwand zu vermeiden, verwenden wir die Methode der **dynamischen Programmierung**, das heißt:

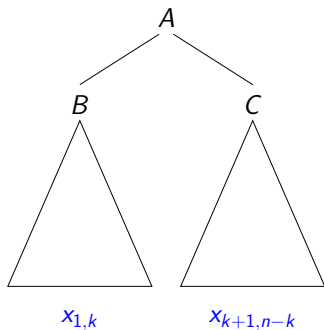
- berechne zuerst alle Variablen, aus denen sich Teilwörter der Länge 1 ableiten lassen,
- berechne dann alle Variablen, aus denen sich Teilwörter der Länge 2 ableiten lassen,
- \vdots
- zuletzt berechne alle Variablen, aus denen sich x ableiten läßt. Falls sich die Startvariable S unter diesen Variablen befindet, so liegt x in der von der Grammatik erzeugten Sprache.

Der CYK-Algorithmus

Notation: Wir bezeichnen mit $x_{i,j}$ das Teilwort von x , das an der Stelle i beginnt und die Länge j hat.

$$x = a_1 \cdots a_n \implies x_{i,j} = a_i \cdots a_{i+j-1}$$

Damit sieht das obige Bild folgendermaßen aus:



Der CYK-Algorithmus

Wir bezeichnen mit $T_{i,j}$ die Menge aller Variablen, aus denen sich $x_{i,j}$ ableiten lässt:

$$T_{i,j} = \{A \in V \mid A \Rightarrow_G^* x_{i,j}\}$$

Dann gilt :

- $T_{i,1} = \{A \in V \mid (A \rightarrow a_i) \in P\}.$
- Für $j \geq 2$ lässt sich $T_{i,j}$ aus den Mengen $T_{\ell,k}$ mit $k < j$ folgendermaßen bestimmen:

$$T_{i,j} = \{A \mid \exists (A \rightarrow BC) \in P \exists 1 \leq k < j : B \in T_{i,k} \text{ und } C \in T_{i+k,j-k}\}$$

Der CYK-Algorithmus

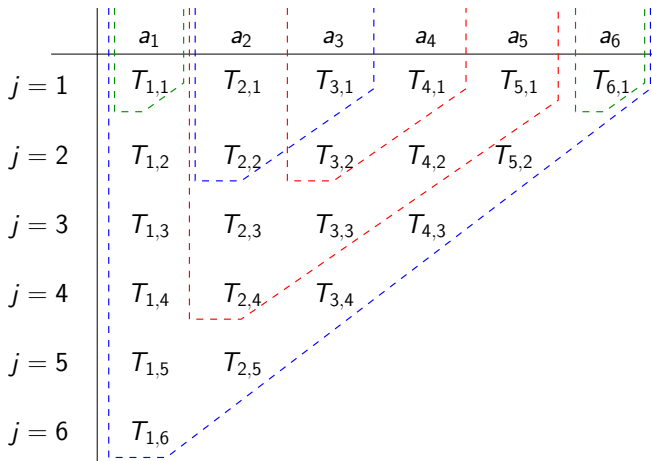
Praktische Ausführung des CYK-Algorithmus:

Wir tragen die Variablenmengen $T_{i,j}$ in folgende Tabelle ein:

	a_1	a_2	\dots	a_{n-1}	a_n
$j = 1$	$T_{1,1}$	$T_{2,1}$	\dots	$T_{n-1,1}$	$T_{n,1}$
$j = 2$	$T_{1,2}$	$T_{2,2}$	\dots	$T_{n-1,2}$	
\dots	\dots	\dots	\dots		
$j = n - 1$	$T_{1,n-1}$	$T_{2,n-1}$			
$j = n$	$T_{1,n}$				

Der CYK-Algorithmus

Folgendermaßen lässt sich veranschaulichen, welche Variablenmenge welches Teilwort ableitet:



Der CYK-Algorithmus

	a_1	a_2	a_3	a_4	a_5	a_6
$j = 1$						$T_{6,1}$
$j = 2$						
$j = 3$						
$j = 4$						
$j = 5$	$T_{1,5}$					
$j = 6$	$T_{1,6}$					

$x = a_1 a_2 a_3 a_4 a_5 \mid a_6$

$(A \rightarrow BC) \in P,$

$B \in T_{1,5}, C \in T_{6,1} \Rightarrow A \in T_{1,6}$

Der CYK-Algorithmus

	a_1	a_2	a_3	a_4	a_5	a_6
$j = 1$						
$j = 2$					$T_{5,2}$	
$j = 3$						
$j = 4$	$T_{1,4}$					
$j = 5$						
$j = 6$	$T_{1,6}$					

$x = a_1 a_2 a_3 a_4 \mid a_5 a_6$

$(A \rightarrow BC) \in P,$

$B \in T_{1,4}, C \in T_{5,2} \Rightarrow A \in T_{1,6}$

Der CYK-Algorithmus

	a_1	a_2	a_3	a_4	a_5	a_6
$j = 1$						
$j = 2$						
$j = 3$	$T_{1,3}$			$T_{4,3}$		
$j = 4$						
$j = 5$						
$j = 6$	$T_{1,6}$					

$x = a_1 a_2 a_3 \mid a_4 a_5 a_6$

$(A \rightarrow BC) \in P,$

$B \in T_{1,3}, C \in T_{4,3} \Rightarrow A \in T_{1,6}$

Der CYK-Algorithmus

	a_1	a_2	a_3	a_4	a_5	a_6
$j = 1$						
$j = 2$	$T_{1,2}$					
$j = 3$						
$j = 4$			$T_{3,4}$			
$j = 5$						
$j = 6$	$T_{1,6}$					

$x = a_1 a_2 \mid a_3 a_4 a_5 a_6$

$(A \rightarrow BC) \in P,$

$B \in T_{1,2}, C \in T_{3,4} \Rightarrow A \in T_{1,6}$

Der CYK-Algorithmus

	a_1	a_2	a_3	a_4	a_5	a_6
$j = 1$	$T_{1,1}$					
$j = 2$						
$j = 3$						
$j = 4$						
$j = 5$		$T_{2,5}$				
$j = 6$	$T_{1,6}$					

$x = a_1 \mid a_2 a_3 a_4 a_5 a_6$

$(A \rightarrow BC) \in P,$

$B \in T_{1,1}, C \in T_{2,5} \Rightarrow A \in T_{1,6}$

Beispiel 1: Betrachte eine Grammatik für die Sprache $L = \{a^k b^k c^j \mid k, j > 0\}$ mit folgenden Produktionen:

$$S \rightarrow AB$$

$$A \rightarrow ab \mid aAb$$

$$B \rightarrow c \mid cB$$

Wir zeigen mittels des CYK-Algorithmus, dass $aaabbbcc \in L$ gilt.

Zunächst müssen wir die Grammatik in Chomsky-Normalform überführen.

Dies ergibt die Grammatik auf der nächsten Folie.

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1								
<i>j</i> = 2								
<i>j</i> = 3								
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2								
<i>j</i> = 3								
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	<i>A_a</i>	<i>A_a</i>	<i>A_a</i>	<i>A_b</i>	<i>A_b</i>	<i>A_b</i>	<i>B, A_c</i>	<i>B, A_c</i>
<i>j</i> = 2	∅							
<i>j</i> = 3								
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset						
$j = 3$								
$j = 4$								
$j = 5$								
$j = 6$								
$j = 7$								
$j = 8$								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A					
<i>j</i> = 3								
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset				
<i>j</i> = 3								
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow A A_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset			
<i>j</i> = 3								
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset		
<i>j</i> = 3								
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3								
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset							
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	<i>A_a</i>	<i>A_a</i>	<i>A_a</i>	<i>A_b</i>	<i>A_b</i>	<i>A_b</i>	<i>B, A_c</i>	<i>B, A_c</i>
<i>j</i> = 2	∅	∅	<i>A</i>	∅	∅	∅	<i>B</i>	
<i>j</i> = 3	∅							
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset						
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset						
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C					
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C					
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset				
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset				
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset			
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset			
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4								
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset							
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset							
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	<i>A_a</i>	<i>A_a</i>	<i>A_a</i>	<i>A_b</i>	<i>A_b</i>	<i>A_b</i>	<i>B, A_c</i>	<i>B, A_c</i>
<i>j</i> = 2	∅	∅	<i>A</i>	∅	∅	∅	<i>B</i>	
<i>j</i> = 3	∅	<i>C</i>	<i>C</i>	∅	∅	∅		
<i>j</i> = 4	<i>C</i>							
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	\emptyset						
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	\emptyset						
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A						
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset					
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset					
$j = 5$								
$j = 6$								
$j = 7$								
$j = 8$								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset					
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset				
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset				
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset				
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5								
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	a	a	a	b	b	b	c	c
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset	\emptyset	\emptyset			
$j = 5$								
$j = 6$								
$j = 7$								
$j = 8$								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset							
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset							
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset	\emptyset	\emptyset			
$j = 5$	\emptyset							
$j = 6$								
$j = 7$								
$j = 8$								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	<i>A_a</i>	<i>A_a</i>	<i>A_a</i>	<i>A_b</i>	<i>A_b</i>	<i>A_b</i>	<i>B, A_c</i>	<i>B, A_c</i>
<i>j</i> = 2	∅	∅	<i>A</i>	∅	∅	∅	<i>B</i>	
<i>j</i> = 3	∅	∅	<i>C</i>	∅	∅	∅		
<i>j</i> = 4	∅	<i>A</i>	∅	∅	∅			
<i>j</i> = 5	∅							
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C						
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset	\emptyset	\emptyset			
$j = 5$	\emptyset	C						
$j = 6$								
$j = 7$								
$j = 8$								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C						
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C						
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset					
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset					
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset					
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset					
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	a	a	a	b	b	b	c	c
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset	\emptyset	\emptyset			
$j = 5$	\emptyset	C	\emptyset	\emptyset				
$j = 6$								
$j = 7$								
$j = 8$								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6								
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	\emptyset							
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	a	a	a	b	b	b	c	c
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset	\emptyset	\emptyset			
$j = 5$	\emptyset	C	\emptyset	\emptyset				
$j = 6$	\emptyset							
$j = 7$								
$j = 8$								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	\emptyset							
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	\emptyset							
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A							
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset						
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset	\emptyset	\emptyset			
$j = 5$	\emptyset	C	\emptyset	\emptyset				
$j = 6$	A	\emptyset						
$j = 7$								
$j = 8$								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset	\emptyset	\emptyset			
$j = 5$	\emptyset	C	\emptyset	\emptyset				
$j = 6$	A	\emptyset						
$j = 7$								
$j = 8$								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset						
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset						
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset	\emptyset	\emptyset			
$j = 5$	\emptyset	C	\emptyset	\emptyset				
$j = 6$	A	\emptyset	\emptyset					
$j = 7$								
$j = 8$								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset	\emptyset	\emptyset			
$j = 5$	\emptyset	C	\emptyset	\emptyset				
$j = 6$	A	\emptyset	\emptyset					
$j = 7$								
$j = 8$								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7								
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow A A_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7	S							
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7	S							
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7	S							
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	a	a	a	b	b	b	c	c
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset	\emptyset	\emptyset			
$j = 5$	\emptyset	C	\emptyset	\emptyset				
$j = 6$	A	\emptyset	\emptyset					
$j = 7$	S							
$j = 8$								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7	S							
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	<i>A_a</i>	<i>A_a</i>	<i>A_a</i>	<i>A_b</i>	<i>A_b</i>	<i>A_b</i>	<i>B, A_c</i>	<i>B, A_c</i>
<i>j</i> = 2	∅	∅	<i>A</i>	∅	∅	∅	<i>B</i>	
<i>j</i> = 3	∅	∅	<i>C</i>	∅	∅	∅		
<i>j</i> = 4	∅	<i>A</i>	∅	∅	∅			
<i>j</i> = 5	∅	<i>C</i>	∅	∅				
<i>j</i> = 6	<i>A</i>	∅	∅					
<i>j</i> = 7	<i>S</i>							
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7	S	\emptyset						
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7	S	\emptyset						
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7	S	\emptyset						
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7	S	\emptyset						
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset	\emptyset	\emptyset			
$j = 5$	\emptyset	C	\emptyset	\emptyset				
$j = 6$	A	\emptyset	\emptyset					
$j = 7$	S	\emptyset						
$j = 8$								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7	S	\emptyset						
<i>j</i> = 8								

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7	S	\emptyset						
<i>j</i> = 8	\emptyset							

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow A A_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	a	a	a	b	b	b	c	c
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset	\emptyset	\emptyset			
$j = 5$	\emptyset	C	\emptyset	\emptyset				
$j = 6$	A	\emptyset	\emptyset					
$j = 7$	S	\emptyset						
$j = 8$	S							

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7	S	\emptyset						
<i>j</i> = 8	S							

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	a	a	a	b	b	b	c	c
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset	\emptyset	\emptyset			
$j = 5$	\emptyset	C	\emptyset	\emptyset				
$j = 6$	A	\emptyset	\emptyset					
$j = 7$	S	\emptyset						
$j = 8$	S							

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	a	a	a	b	b	b	c	c
$j = 1$	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
$j = 2$	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
$j = 3$	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
$j = 4$	\emptyset	A	\emptyset	\emptyset	\emptyset			
$j = 5$	\emptyset	C	\emptyset	\emptyset				
$j = 6$	A	\emptyset	\emptyset					
$j = 7$	S	\emptyset						
$j = 8$	S							

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	A_a	A_a	A_a	A_b	A_b	A_b	B, A_c	B, A_c
<i>j</i> = 2	\emptyset	\emptyset	A	\emptyset	\emptyset	\emptyset	B	
<i>j</i> = 3	\emptyset	\emptyset	C	\emptyset	\emptyset	\emptyset		
<i>j</i> = 4	\emptyset	A	\emptyset	\emptyset	\emptyset			
<i>j</i> = 5	\emptyset	C	\emptyset	\emptyset				
<i>j</i> = 6	A	\emptyset	\emptyset					
<i>j</i> = 7	S	\emptyset						
<i>j</i> = 8	S							

Der CYK-Algorithmus

$$S \rightarrow AB$$

$$A_a \rightarrow a$$

$$A \rightarrow A_a A_b \mid A_a C$$

$$A_b \rightarrow b$$

$$C \rightarrow AA_b$$

$$A_c \rightarrow c$$

$$B \rightarrow c \mid A_c B$$

	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>c</i>
<i>j</i> = 1	<i>A_a</i>	<i>A_a</i>	<i>A_a</i>	<i>A_b</i>	<i>A_b</i>	<i>A_b</i>	<i>B, A_c</i>	<i>B, A_c</i>
<i>j</i> = 2	∅	∅	<i>A</i>	∅	∅	∅	<i>B</i>	
<i>j</i> = 3	∅	∅	<i>C</i>	∅	∅	∅		
<i>j</i> = 4	∅	<i>A</i>	∅	∅	∅			
<i>j</i> = 5	∅	<i>C</i>	∅	∅				
<i>j</i> = 6	<i>A</i>	∅	∅					
<i>j</i> = 7	<i>S</i>	∅						
<i>j</i> = 8	<i>S</i>							

Beispiel 2: Betrachte eine Grammatik mit folgenden Produktionen:

$$S \rightarrow AD \mid FG$$

$$D \rightarrow SE \mid BC$$

$$E \rightarrow BC$$

$$F \rightarrow AF \mid a$$

$$G \rightarrow BG \mid CG \mid b$$

$$A \rightarrow a$$

$$B \rightarrow b$$

$$C \rightarrow c$$

Frage: Sei $x = aabcbcb$. Gilt $x \in L$?

Der CYK-Algorithmus

Hier ist die Tabelle, die sich mit dem CYK-Algorithmus ergibt
(Sie sollten dies überprüfen):

	<i>a</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>b</i>	<i>c</i>
$j = 1$	<i>A, F</i>	<i>A, F</i>	<i>B, G</i>	<i>C</i>	<i>B, G</i>	<i>C</i>
$j = 2$	<i>F</i>	<i>S</i>	<i>D, E</i>	<i>G</i>	<i>D, E</i>	
$j = 3$	<i>S</i>	<i>S</i>	<i>G</i>			
$j = 4$		<i>S</i>				
$j = 5$	<i>S</i>	<i>D</i>				
$j = 6$	<i>S</i>					

Der CYK-Algorithmus

Komplexität des CYK-Algorithmus

Sei $n = |x|$ die Länge des Wortes, das untersucht wird. Die Größe der Grammatik wird als konstant angesehen. Dann gilt:

- $O(n^2)$ Tabellenfelder müssen ausgefüllt werden.
- Für das Ausfüllen jedes Tabellenfeldes müssen bis zu $O(n)$ andere Felder betrachtet werden.

(Für $T_{1,n}$ müssen beispielsweise die Felder $T_{1,n-1}$, $T_{n,1}$ und $T_{1,n-2}$, $T_{n-1,2}$ und ... und $T_{1,1}$, $T_{2,n-1}$ betrachtet werden. Insgesamt $n - 1$ Paare von Feldern.)

Daher ergibt sich insgesamt als Zeitkomplexität: $O(n^3)$.

Die Zeitkomplexität ist noch polynomiell, aber für das Parsen großer Programme eigentlich nicht mehr geeignet.

Kellerautomaten

Was ist ein geeignetes Automatenmodell für kontextfreie Sprachen?

Analog zu regulären Sprachen suchen wir hier ein Automatenmodell für kontextfreie Sprachen.

Antwort: **Kellerautomaten**, d.h., Automaten, die mit einem zusätzlichen Keller ausgestattet sind.

Nutzen eines solchen Automatenmodells

Manche Konstruktionen und Verfahren lassen sich besser mit Hilfe des Automatenmodells durchführen (anstatt auf Grammatiken).

- **Wortproblem:** Wir werden herausfinden, dass das Wortproblem unter bestimmten Umständen effizienter als in Zeit $O(n^3)$ gelöst werden kann.
- **Abschlusseigenschaften:** Abschluss der kontextfreien Sprachen unter Schnitt mit regulären Sprachen lässt sich gut mit Kellerautomaten zeigen.

Wir betrachten die Sprache

$$L = \{a_1 a_2 \cdots a_n \$ a_n \cdots a_2 a_1 \mid a_i \in \Delta\}$$

mit $\Sigma = \Delta \cup \{\$\}$, $\$ \notin \Delta$.

Ein **endlicher Automat** kann diese Sprache deshalb nicht erkennen, weil er sich keine beliebig langen Wörter der Form $a_1 a_2 \cdots a_n$ “merken” kann.

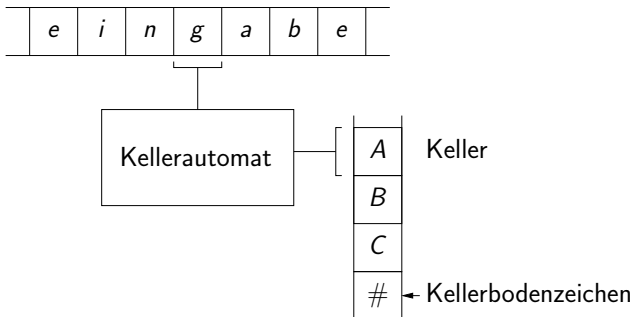
Er müsste sich aber solche Wörter merken, um die Übereinstimmung mit dem Wortteil nach dem $\$$ zu überprüfen.

Um ein Automatenmodell für kontextfreie Sprachen zu erhalten,

- führen wir daher einen **Keller** oder **Pushdown-Speicher** ein, auf dem sich eine beliebig lange Sequenz von Zeichen befinden darf.
- Beim Einlesen eines neuen Zeichens darf das oberste Zeichen des Kellers gelesen und folgendermaßen verändert werden:
 - Entweder bleibt der Keller unverändert oder
 - das oberste Zeichen des Kellers wird entfernt und durch eine (evtl. leere) Sequenz von Zeichen ersetzt.

An anderen Stellen darf der Keller nicht gelesen oder verändert werden.

Schematische Darstellung eines Kellerautomaten:



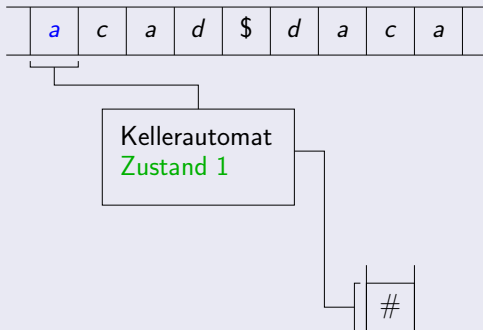
Kellerautomaten

Sei $\Delta = \{a, b, c, d\}$ und $L = \{a_1 a_2 \cdots a_n \$ a_n \cdots a_2 a_1 \mid a_i \in \Delta\}$.

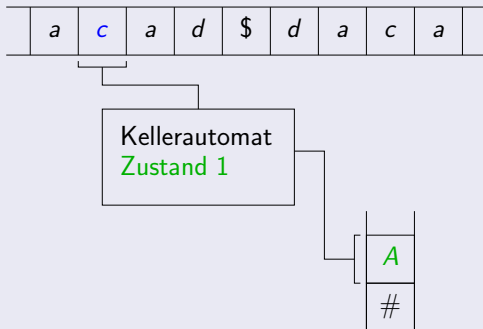
Ein Kellerautomat erkennt diese Sprache folgendermaßen:

- Ein Wort w wird von links nach rechts eingelesen.
- Solange $\$$ noch nicht erreicht ist, wird jedes eingelesene Symbol als Großbuchstabe auf den Keller gelegt ($a \rightsquigarrow A, b \rightsquigarrow B, \dots$).
- Wenn $\$$ eingelesen wird, bleibt der Keller unverändert.
- Anschließend wird für jedes neu eingelesene Zeichen überprüft, ob der passende Großbuchstabe auf dem Keller liegt. Dieser wird dann entfernt.
- Falls irgendwann keine Übereinstimmung festgestellt wird, blockiert der Kellerautomat.
- Falls immer Übereinstimmung herrscht, wird schließlich auch das Kellerbodenzeichen $\#$ entfernt, und der Automat akzeptiert mit leerem Keller.

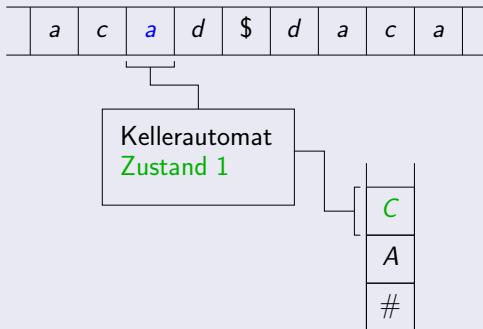
Simulation



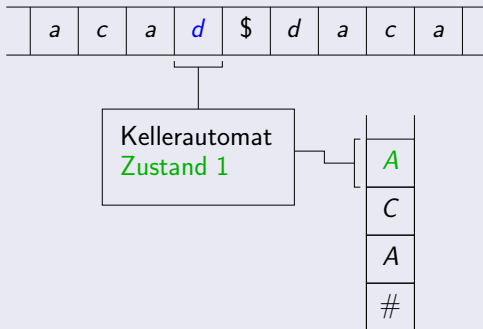
Simulation



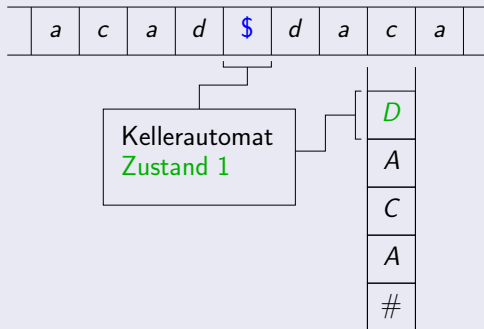
Simulation



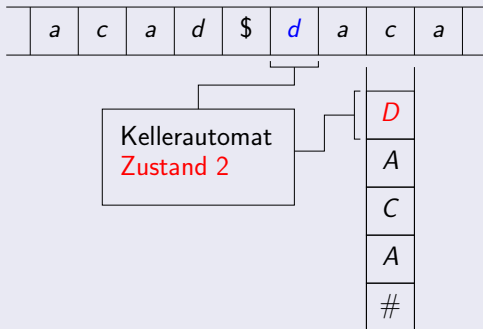
Simulation



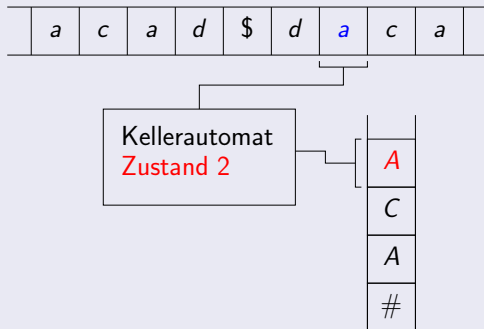
Simulation



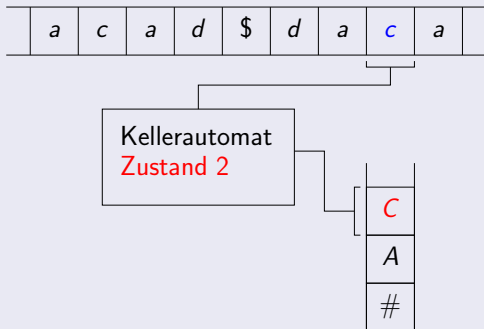
Simulation



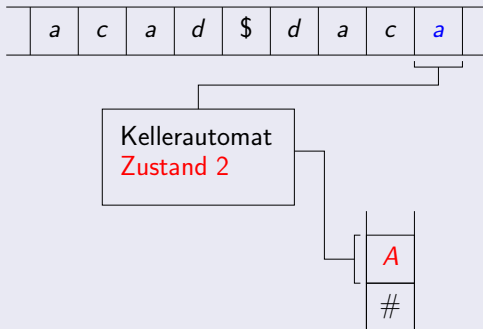
Simulation



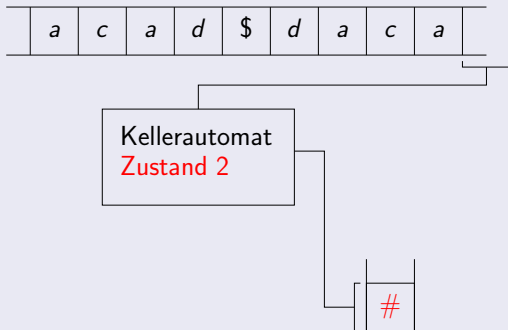
Simulation



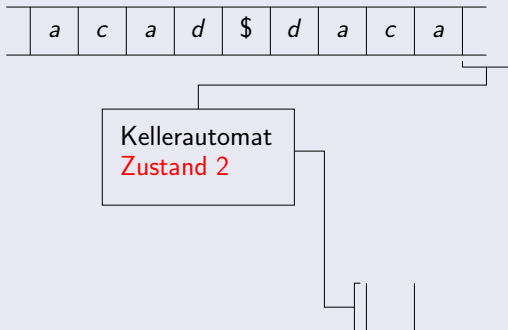
Simulation



Simulation



Simulation



Definition (Kellerautomat)

Ein **nichtdeterministischer Kellerautomat** M ist ein 6-Tupel

$M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$, wobei

- Z die endliche Menge der **Zustände**,
- Σ das endliche **Eingabealphabet** (mit $Z \cap \Sigma = \emptyset$),
- Γ das endliche **Kelleralphabet**,
- $z_0 \in Z$ der **Startzustand**,
- $\# \in \Gamma$ das **unterste Kellerzeichen** oder **Kellerbodenzeichen**, und
- $\delta: Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow 2^{Z \times \Gamma^*}$ die **Überföhrungsfunktion** ist, wobei $\delta(z, a, A)$ für alle $(z, a, A) \in Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ endlich sein muss.

Abkürzung: KA oder PDA (pushdown automaton).

- Wir betrachten die **Überföhrungsfunktion**

$$\delta: Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow 2^{Z \times \Gamma^*}.$$

Falls $(z', B_1 \cdots B_k) \in \delta(z, a, A)$, so bedeutet das:

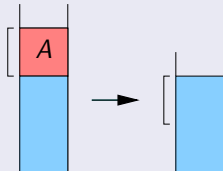
- wenn im Zustand z das Eingabesymbol a gelesen wird und das Zeichen A als oberstes auf dem Keller liegt, dann
- wird A vom Keller entfernt und durch $B_1 \cdots B_k$ ersetzt (B_1 liegt zuoberst) und der Automat geht in den Zustand z' über.

Es kann auch $a = \varepsilon$ gelten. In diesem Fall wird kein Eingabesymbol eingelesen. Wir sprechen dann von einer **ε -Transition**.

Wir betrachten verschiedene Fälle von Werten der Überföhrungsfunktion δ :

$(z', \varepsilon) \in \delta(z, a, A)$

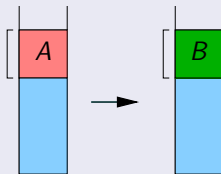
- Zeichen a wird gelesen.
- Zustand ändert sich von z nach z' .
- Symbol A wird vom Keller entfernt:



$(z', B) \in \delta(z, a, A)$

- Zeichen a wird gelesen.
- Zustand ändert sich von z nach z' .

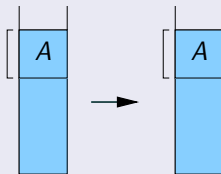
- Symbol A auf dem Keller wird durch B ersetzt:



$(z', A) \in \delta(z, a, A)$

- Zeichen a wird gelesen.
- Zustand ändert sich von z nach z' .

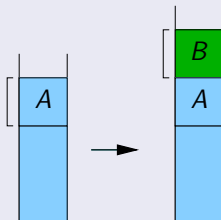
- Symbol A bleibt auf dem Keller:



$$(z', BA) \in \delta(z, a, A)$$

- Zeichen a wird gelesen.
- Zustand ändert sich von z nach z' .

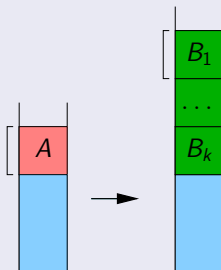
- Symbol B wird neu auf den Keller gelegt:



$$(z', B_1 \cdots B_k) \in \delta(z, a, A)$$

- Zeichen a wird gelesen.
- Zustand ändert sich von z nach z' .

- Symbol A wird durch mehrere neue Symbole ersetzt:



- Zu Beginn einer jeden Berechnung enthält der Keller genau das **Kellerbodenzeichen #**.
- Der Keller ist **nicht beschränkt** und kann beliebig wachsen. Es gibt **unendlich viele mögliche Kellerinhalte**, das unterscheidet Kellerautomaten von endlichen Automaten.
- Die von uns betrachteten Kellerautomaten akzeptieren immer mit **leerem Keller** (in diesem Fall gibt es auch keine Übergangsmöglichkeiten mehr). Es gibt aber auch andere Varianten von Kellerautomaten, die mit Endzustand akzeptieren.

Beispiel:

PDA für $L = \{a_1 a_2 \cdots a_n \$ a_n \cdots a_2 a_1 \mid n \geq 0, a_1, \dots, a_n \in \{a, b\}\}$:

$$M = (\{z_1, z_2\}, \{a, b, \$\}, \{\#, A, B\}, \delta, z_1, \#),$$

wobei δ folgendermaßen definiert ist (wir schreiben $(z, a, A) \rightarrow (z', x)$, falls $(z', x) \in \delta(z, a, A)$).

$$\begin{array}{lll} (z_1, a, \#) \rightarrow (z_1, A\#) & (z_1, a, A) \rightarrow (z_1, AA) & (z_1, a, B) \rightarrow (z_1, AB) \\ (z_1, b, \#) \rightarrow (z_1, B\#) & (z_1, b, A) \rightarrow (z_1, BA) & (z_1, b, B) \rightarrow (z_1, BB) \\ (z_1, \$, \#) \rightarrow (z_2, \#) & (z_1, \$, A) \rightarrow (z_2, A) & (z_1, \$, B) \rightarrow (z_2, B) \\ (z_2, a, A) \rightarrow (z_2, \varepsilon) & (z_2, b, B) \rightarrow (z_2, \varepsilon) & (z_2, \varepsilon, \#) \rightarrow (z_2, \varepsilon) \end{array}$$

Definition (Konfiguration eines PDA)

Eine **Konfiguration** eines PDA ist ein Tripel $k \in Z \times \Sigma^* \times \Gamma^*$.

Bedeutung der Komponenten von $k = (z, w, \gamma) \in Z \times \Sigma^* \times \Gamma^*$:

- $z \in Z$ ist der **aktuelle Zustand** des PDA.
- $w \in \Sigma^*$ ist der **noch zu lesende Teil der Eingabe**.
- $\gamma \in \Gamma^*$ ist der **aktuelle Kellerinhalt**. Dabei steht das oberste Kellerzeichen ganz links.

Übergänge zwischen Konfigurationen ergeben sich aus der Überföhrungsfunktion δ :

Definition (Konfigurationsübergänge eines PDA)

Es gilt

$$(z, aw, A\gamma) \vdash (z', w, B_1 \cdots B_k \gamma),$$

falls $(z', B_1 \cdots B_k) \in \delta(z, a, A)$, und es gilt

$$(z, w, A\gamma) \vdash (z', w, B_1 \cdots B_k \gamma),$$

falls $(z', B_1 \cdots B_k) \in \delta(z, \varepsilon, A)$.

Hierbei ist $\gamma \in \Gamma^*$ eine beliebige Folge von Kellersymbolen, $A, B_1, \dots, B_k \in \Gamma$, $w \in \Sigma^*$, $a \in \Sigma$, und $z, z' \in Z$.

Im ersten Fall wird ein Zeichen der Eingabe gelesen, im zweiten jedoch nicht.

Wir definieren \vdash^* als die reflexive and transitive Hülle von \vdash .

Damit kann jetzt die von einem PDA **akzeptierte Sprache** definiert werden:

Definition (Akzeptierte Sprache eines PDA)

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$ ein PDA. Dann ist die von M **akzeptierte Sprache**:

$$N(M) = \{x \in \Sigma^* \mid (z_0, x, \#) \vdash^* (z, \varepsilon, \varepsilon) \text{ für ein } z \in Z\}.$$

Das heißt die akzeptierte Sprache enthält alle Wörter, mit Hilfe derer es möglich ist, den Keller vollständig zu leeren.

Da Kellerautomaten jedoch nicht-deterministisch sind, kann es auch Berechnungen für dieses Wort geben, die den Keller nicht leeren.

Folgende Folge von Konfigurationsübergängen zeigt, dass der Kellerautomat von Folie 264 das Wort $ab\$ba$ akzeptiert.

$(z_1, ab\$ba, \#) \vdash (z_1, b\$ba, A\#)$	wegen $(z_1, a, \#) \rightarrow (z_1, A\#)$
$\vdash (z_1, \$ba, BA\#)$	wegen $(z_1, b, A) \rightarrow (z_1, BA)$
$\vdash (z_2, ba, BA\#)$	wegen $(z_1, \$, B) \rightarrow (z_2, B)$
$\vdash (z_2, a, A\#)$	wegen $(z_2, b, B) \rightarrow (z_2, \varepsilon)$
$\vdash (z_2, \varepsilon, \#)$	wegen $(z_2, a, A) \rightarrow (z_2, \varepsilon)$
$\vdash (z_2, \varepsilon, \varepsilon)$	wegen $(z_2, \varepsilon, \#) \rightarrow (z_2, \varepsilon)$

Ein weiteres **Beispiel**: ein PDA für die Sprache

$$L = \{a_1 a_2 \cdots a_n a_n \cdots a_2 a_1 \mid n \geq 0, a_1, \dots, a_n \in \{a, b\}\}.$$

Idee: anstatt auf das Zeichen \$ zu warten, kann sich der Automat nicht-deterministisch entscheiden, in den Zustand z_2 (= Keller abbauen) überzugehen, sobald das aktuelle Zeichen auf dem Band mit dem Zeichen auf dem Keller übereinstimmt (oder wenn der Keller leer ist).

Veränderte Überföhrungsfunktion δ (3. Zeile ist geändert):

$$\begin{array}{lll} (z_1, a, \#) \rightarrow (z_1, A\#) & (z_1, a, A) \rightarrow (z_1, AA) & (z_1, a, B) \rightarrow (z_1, AB) \\ (z_1, b, \#) \rightarrow (z_1, B\#) & (z_1, b, A) \rightarrow (z_1, BA) & (z_1, b, B) \rightarrow (z_1, BB) \\ (z_1, \varepsilon, \#) \rightarrow (z_2, \#) & (z_1, a, A) \rightarrow (z_2, \varepsilon) & (z_1, b, B) \rightarrow (z_2, \varepsilon) \\ (z_2, a, A) \rightarrow (z_2, \varepsilon) & (z_2, b, B) \rightarrow (z_2, \varepsilon) & (z_2, \varepsilon, \#) \rightarrow (z_2, \varepsilon) \end{array}$$

Anmerkung: dieser Kellerautomat ist (im Gegensatz zum vorherigen) nicht-deterministisch, d.h., eine Konfiguration kann mehrere mögliche Nachfolger haben. (Und möglicherweise enden einige Konfigurationsfolgen als Sackgassen und führen nicht dazu, dass der Keller geleert wird.)

Beispiel: Kellerautomat erhält die Eingabe *aabbaa*.

Folgende Folge von Konfigurationsübergängen zeigt, dass diese Eingabe akzeptiert wird:

$(z_1, aabbaa, \#) \vdash (z_1, abbaa, A\#)$	wegen $(z_1, a, \#) \rightarrow (z_1, A\#)$
$\vdash (z_1, bbaa, AA\#)$	wegen $(z_1, a, A) \rightarrow (z_1, AA)$
$\vdash (z_1, baa, BAA\#)$	wegen $(z_1, b, A) \rightarrow (z_1, BA)$
$\vdash (z_2, aa, AA\#)$	wegen $(z_1, b, B) \rightarrow (z_2, \varepsilon)$
$\vdash (z_2, a, A\#)$	wegen $(z_2, a, A) \rightarrow (z_2, \varepsilon)$
$\vdash (z_2, \varepsilon, \#)$	wegen $(z_2, a, A) \rightarrow (z_2, \varepsilon)$
$\vdash (z_2, \varepsilon, \varepsilon)$	wegen $(z_2, \varepsilon, \#) \rightarrow (z_2, \varepsilon)$

Beachte: Es gibt auch viele andere mögliche Berechnungen, bei denen der Keller am Ende nicht leer ist, wie z.B.

$(z_1, aabbaa, \#) \vdash (z_1, abbaa, A\#)$	wegen $(z_1, a, \#) \rightarrow (z_1, A\#)$
$\vdash (z_1, bbaa, AA\#)$	wegen $(z_1, a, A) \rightarrow (z_1, AA)$
$\vdash (z_1, baa, BAA\#)$	wegen $(z_1, b, A) \rightarrow (z_1, BA)$
$\vdash (z_1, aa, BBAA\#)$	wegen $(z_1, b, B) \rightarrow (z_1, BB)$
$\vdash (z_1, a, ABBA\#)$	wegen $(z_1, a, B) \rightarrow (z_1, AB)$
$\vdash (z_1, \varepsilon, AABBA\#)$	wegen $(z_1, a, A) \rightarrow (z_1, AA)$

Solche Berechnungen ändern jedoch nichts mehr an der Tatsache, dass das Wort *aabbaa* akzeptiert wird.

Hierfür genügt die Existenz der einen Berechnung auf der vorherigen Folie bei der nach Lesen der Eingabe der Keller leer ist.

Wir müssen nun noch zeigen, dass man mit Kellerautomaten wirklich genau die kontextfreien Sprachen akzeptieren kann.

Satz (kontextfreie Grammatiken \rightarrow Kellerautomaten)

Zu jeder kontextfreien Grammatik G gibt es einen PDA M mit $L(G) = N(M)$.

Beweisidee:

- 1 Wir können ohne Beschränkung der Allgemeinheit davon ausgehen, dass G in Greibach-Normalform ist.
- 2 Wir simulieren eine Ableitung von G , indem wir den Keller zur Abspeicherung derjenigen Variablen, für die noch etwas abgeleitet werden muss, verwenden.
- 3 Eine Produktion $A \rightarrow aA_1 \cdots A_n$ wird wie folgt simuliert:
Falls a das nächste Eingabesymbol ist und auf dem Keller oben A liegt, kann A durch $A_1 \cdots A_n$ ersetzt werden.
- 4 Wenn die komplette Eingabe gelesen wurde, und der Keller gleichzeitig leer ist, dann wurde eine komplette Ableitung für das Eingabewort simuliert.

Formaler: Zunächst nehmen wir an, dass $\varepsilon \notin L(G)$.

Dann können wir o.B.d.A. davon ausgehen, dass $G = (V, \Sigma, P, S)$ in Greibach-Normalform ist.

Wir definieren den PDA

$$M = (\{z\}, \Sigma, V, \delta, z, S)$$

mit folgender Überföhrungsfunktion: Für $A \in V$ und $a \in \Sigma$ sei

$$\delta(z, a, A) = \{(z, A_1 \cdots A_m) \mid (A \rightarrow aA_1 \cdots A_m) \in P\}$$

Beachte:

- M hat nur einen Zustand (z).
- M hat keine ε -Transitionen.
- Das Startsymbol S von G ist das Kellerbodenzeichen.
- Da G in Greibach-Normalform ist, sind alle Produktionen in P von der Form $A \rightarrow aA_1 \cdots A_m$ mit $m \geq 0$, $A, A_1, \dots, A_m \in V$ und $a \in \Sigma$.

Behauptung: Für alle $u \in \Sigma^*$ und $\gamma \in V^*$ gilt:

$$(z, u, \gamma) \vdash^* (z, \varepsilon, \varepsilon) \iff \gamma \Rightarrow_G^* u.$$

Beweis: Induktion über $|u|$.

Induktionsanfang: $u = \varepsilon$.

Dann gilt

$$(z, \varepsilon, \gamma) \vdash^* (z, \varepsilon, \varepsilon) \iff \gamma = \varepsilon \iff \gamma \Rightarrow_G^* \varepsilon.$$

Induktionsschritt: Sei $u = av$ mit $a \in \Sigma$, $v \in \Sigma^*$.

Falls $\gamma = \varepsilon$ gilt weder $\gamma \Rightarrow_G^* av$ noch $(z, av, \gamma) \vdash^* (z, \varepsilon, \varepsilon)$.

Sei nun $\gamma = A\gamma'$ mit $A \in V$ und $\gamma' \in V^*$.

Dann gilt:

$$A\gamma' \Rightarrow_G^* av$$

$$\iff \exists (A \rightarrow aA_1 \cdots A_m) \in P : A_1 \cdots A_m \gamma' \Rightarrow_G^* v$$

$$\iff \exists (A \rightarrow aA_1 \cdots A_m) \in P : (z, v, A_1 \cdots A_m \gamma') \vdash^* (z, \varepsilon, \varepsilon)$$

$$\iff \exists (z, A_1 \cdots A_m) \in \delta(z, a, A) : (z, v, A_1 \cdots A_m \gamma') \vdash^* (z, \varepsilon, \varepsilon)$$

$$\iff (z, av, A\gamma') \vdash^* (z, \varepsilon, \varepsilon)$$

Aus der obigen Behauptung folgt:

$$w \in L(G) \iff S \Rightarrow_G^* w \iff (z, w, S) \vdash^* (z, \varepsilon, \varepsilon) \iff w \in N(M).$$

Falls $\varepsilon \in L(G)$ gilt, können wir o.B.d.A. davon ausgehen, dass bis auf die Produktion $S \rightarrow \varepsilon$ alle Produktionen von G in Greibach-Normalform sind, und S auf keiner rechten Seite von G vorkommt.

Wir fügen dann zu dem auf Folie 275 definierten PDA noch $\delta(z, \varepsilon, S) = \{(z, \varepsilon)\}$ (die einzige ε -Transition) hinzu.

Diese kann nur ganz zu Beginn einer Berechnung $(z, w, S) \vdash^* (z, \varepsilon, \varepsilon)$ angewendet werden (dann muss $w = \varepsilon$ gelten).

Es gilt dann wieder wie gewünscht $L(G) = N(M)$. □

Alternative Konstruktion:

Wir können auch direkt aus einer beliebigen kontextfreien Grammatik $G = (V, \Sigma, P, S)$ einen PDA M mit $L(G) = N(M)$ konstruieren.

Definiere den PDA $M = (\{z\}, \Sigma, V \cup \Sigma, \delta, z, S)$ mit einem Zustand z und Kelleralphabet $V \cup \Sigma$.

Überföhrungsfunktion δ :

$$\delta(z, \varepsilon, A) = \{(z, \alpha) \mid (A \rightarrow \alpha) \in P\} \text{ für } A \in V$$

$$\delta(z, a, a) = \{(z, \varepsilon)\} \text{ für } a \in \Sigma$$

Mit Produktionen vom ersten Typ werden Ableitungsschritte auf dem Keller ohne Lesen der Eingabe simuliert.

Mit Produktionen vom zweiten Typ wird ein Symbol der Eingabe mit dem Keller verglichen.

Beachte: M hat ε -Produktionen.

Kellerautomaten

Wir betrachten folgende kontextfreie Grammatik mit dem zweielementigen Alphabet $\Sigma = \{[,]\}$, die korrekte Klammerstrukturen erzeugt:

$$S \rightarrow [S]S \mid \varepsilon$$

Aufgabe: wandle diese Grammatik in einen Kellerautomaten um und akzeptiere damit das Wort $[[]][[]]$.

Wir verwenden die Konstruktion von Folie 279:

- Zustandsmenge = $\{z\}$
- Kelleralphabet = $\{[,], S\}$
- Kellerbodenzeichen = S
- Überföhrungsfunktion:

$$\delta(z, \varepsilon, S) = \{(z, \varepsilon), (z, [S]S)\}$$

$$\delta(z, a, a) = \{(z, \varepsilon)\} \text{ f ur } a \in \{[,]\}$$

Links finden Sie eine Ableitung des Worts $[[]][[]]$ mit der Grammatik, rechts steht die entsprechende Berechnung des obigen Kellerautomaten:

$S \Rightarrow [S]S$	$(z, [[]][], S) \vdash (z, [[]][], [S]S)$
	$\vdash (z, [[]][], S]S)$
$\Rightarrow [[S]S]S$	$\vdash (z, [[]][], [S]S]S)$
	$\vdash (z, []][], S]S]S)$
$\Rightarrow [[]S]S$	$\vdash (z, []][],]S]S)$
	$\vdash (z, []][], S]S)$
$\Rightarrow [[]]S$	$\vdash (z, []][],]S)$
	$\vdash (z, []][], S)$
$\Rightarrow [[]][S]S$	$\vdash (z, []][], [S]S)$
	$\vdash (z, []][], S]S)$
$\Rightarrow [[]][]S$	$\vdash (z, []][],]S)$
	$\vdash (z, []][], \varepsilon, S)$
$\Rightarrow [[]][[]]$	$\vdash (z, []][], \varepsilon, \varepsilon)$

Nun geht es darum zu zeigen, dass es zu jedem Kellerautomaten eine entsprechende kontextfreie Grammatik gibt.

Das ist die schwierigere Richtung.

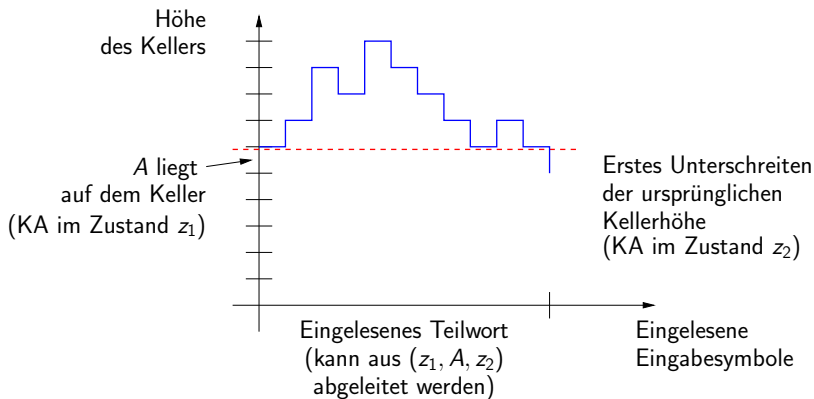
Satz (Kellerautomaten \rightarrow kontextfreie Grammatiken)

Zu jedem Kellerautomaten M gibt es eine kontextfreie Grammatik G mit $N(M) = L(G)$.

Beweisidee:

- 1 Wir wollen beschreiben, welche Wörter man durch Abbauen eines bestimmten Kellersymbols akzeptieren kann. Die vom Automaten akzeptierte Sprache besteht nämlich aus allen Wörtern, die man durch Abbauen von $\#$ erzeugen kann.
“Abbauen” bedeutet: zwischendurch dürfen weitere Symbole auf den Keller gelegt werden, aber zuletzt muss der Keller um dieses eine Symbol kürzer geworden sein.
- 2 Die zu erstellende kontextfreie Grammatik besitzt Variablen der Form (z_1, A, z_2) mit der Bedeutung:
Aus (z_1, A, z_2) kann man genau die Wörter ableiten, die der Kellerautomat einliest, wenn er im Zustand z_1 startet, A vom Keller abbaut und im Zustand z_2 aufhört.

Kellerautomaten



Zwischendurch kann A durch ein anderes Symbol ersetzt werden. Die ursprüngliche Kellerhöhe wird jedoch nicht unterschritten.

Formale Bedeutung der Symbole (z_1, A, z_2) :

$$(z_1, A, z_2) \Rightarrow^* x \iff (z_1, x, A) \vdash^* (z_2, \varepsilon, \varepsilon)$$

Gegeben sei ein Kellerautomat $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$.

Wir definieren eine Grammatik $G = (V, \Sigma, P, S)$ wie folgt
(siehe nächste Folie):

- Variablen: $V = \{S\} \cup Z \times \Gamma \times Z$
(Eigene Startvariable und Variablen der Form (z_1, A, z_2))
- Produktionen haben folgende Form:

$$S \rightarrow (z_0, \#, z) \quad \text{für alle } z \in Z$$

(Entfernen des Kellerbodenzeichens)

$$(z, A, z') \rightarrow a \quad \text{falls } (z', \varepsilon) \in \delta(z, a, A)$$

(Symbol A kann – bei Einlesen von a – sofort entfernt werden)

$$(z, A, z') \rightarrow a(z_1, B_1, z_2)(z_2, B_2, z_3) \cdots (z_k, B_k, z') \quad \text{für alle } (z_1, B_1 \cdots B_k) \in \delta(z, a, A), z_2, \dots, z_k \in Z, k \geq 1$$

(Symbol A wird durch $B_1 \dots B_k$ ersetzt, diese müssen über Zwischenzustände z_1, \dots, z_k entfernt werden.)

Beispiel: Wir betrachten den Kellerautomaten

$$M = (\{z_1, z_2\}, \{a, b\}, \{A, \#\}, \delta, z_1, \#)$$

mit folgender Überföhrungsfunktion δ :

$$(z_1, \varepsilon, \#) \rightarrow (z_2, \varepsilon)$$

$$(z_1, a, \#) \rightarrow (z_1, AA)$$

$$(z_1, a, A) \rightarrow (z_1, AAA)$$

$$(z_1, b, A) \rightarrow (z_2, \varepsilon)$$

$$(z_2, b, A) \rightarrow (z_2, \varepsilon)$$

Es gilt: $N(M) = \{a^n b^{2n} \mid n \geq 0\}$.

Aufgabe: Umwandlung von M in eine kontextfreie Grammatik.

$$S \rightarrow (z_1, \#, z_1)$$

$$S \rightarrow (z_1, \#, z_2)$$

$$(z_1, \#, z_2) \rightarrow \varepsilon$$

$$(z_1, A, z_2) \rightarrow b$$

$$(z_2, A, z_2) \rightarrow b$$

$$(z_1, \#, z_i) \rightarrow a(z_1, A, z_j)(z_j, A, z_i)$$

$$(z_1, A, z_i) \rightarrow a(z_1, A, z_j)(z_j, A, z_k)(z_k, A, z_i)$$

Die letzten beiden Produktionen sind dabei für alle $i, j, k \in \{1, 2\}$ vorhanden.

Insgesamt hat die Grammatik also 17 Produktionen.

Bemerkung zu den Umwandlungen

“Kontextfreie Grammatik \leftrightarrow Kellerautomat”:

Zu jedem Kellerautomaten M gibt es immer einen äquivalenten Kellerautomaten M' mit nur einem Zustand und ohne ε -Transitionen (falls $\varepsilon \notin N(M)$).

- 1 Wandle M zunächst in eine kontextfreie Grammatik G um.
- 2 Wandle dann G in eine kontextfreie Grammatik G' in Greibach-Normalform um.
- 3 Wandle schließlich G' in einen Kellerautomaten M' um.

Es wird ausgenutzt, dass bei der Umwandlung einer Grammatik (in Greibach-Normalform) in einen Kellerautomaten immer nur Automaten mit einem Zustand und ohne ε -Transitionen konstruiert werden.

Deterministisch kontextfreie Sprachen

Wir betrachten nun eine Unterklasse von Kellerautomaten, die dazu verwendet werden können, Sprachen deterministisch und damit effizient zu erkennen.

Definition (deterministischer Kellerautomat)

Ein **deterministischer Kellerautomat** M ist ein 7-Tupel

$M = (Z, \Sigma, \Gamma, \delta, z_0, \#, E)$, wobei

- $(Z, \Sigma, \Gamma, \delta, z_0, \#)$ ein **Kellerautomat** ist,
- $E \subseteq Z$ eine Menge von **Endzuständen** ist, und
- die Überföhrungsfunktion $\delta: Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow 2^{Z \times \Gamma^*}$ in folgendem Sinne **deterministisch** ist:

Für alle $z \in Z$, $a \in \Sigma$ und $A \in \Gamma$ gilt:

$$|\delta(z, a, A)| + |\delta(z, \varepsilon, A)| \leq 1.$$

Unterschiede zwischen Kellerautomaten und deterministischen Kellerautomaten:

- Deterministische Kellerautomaten haben eine Menge von Endzuständen und akzeptieren mit Endzustand – und nicht mit leerem Keller.

Bei deterministischen Kellerautomaten ist dies ein Unterschied, für nicht-deterministische Kellerautomaten sind beide Akzeptanzmöglichkeiten gleichwertig.

- Für jeden Zustand z und jedes Kellersymbol A gilt:
 - entweder gibt es höchstens einen ε -Übergang
 - oder es gibt für jedes Alphabetsymbol höchstens einen Übergang.

Konfigurationen und Übergänge zwischen Konfiguration bleiben jedoch gleich definiert.

Konfigurationsfolgen werden jedoch zu linearen Ketten, d.h., es gibt immer höchstens eine Folgekonfiguration.

Diese Tatsache wird dann für die effiziente Lösung des Wortproblems ausgenutzt.

Definition (akzeptierte Sprache bei det. Kellerautomaten)

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \#, E)$ ein deterministischer PDA. Dann ist die von M **akzeptierte Sprache**:

$$D(M) = \{x \in \Sigma^* \mid (z_0, x, \#) \vdash^* (z, \varepsilon, \gamma) \text{ für ein } z \in E, \gamma \in \Gamma^*\}.$$

Vergleiche mit der Definition für nicht-deterministische Kellerautomaten!

Bei deterministischen Kellerautomaten ist folgendes anders:

- Der erreichte Zustand z muss ein Endzustand sein.
- Es darf ein Kellerinhalt γ übrigbleiben.

Definition (deterministisch kontextfreie Sprachen)

Eine Sprache heißt **deterministisch kontextfrei** genau dann, wenn sie von einem deterministischen PDA akzeptiert wird.

Beispiele:

- Die Sprache $L = \{a_1 a_2 \dots a_n \$ a_n \dots a_2 a_1 \mid a_i \in \Delta\}$ ist **deterministisch kontextfrei** (siehe den entsprechenden PDA).
- Die Sprache $L = \{a_1 a_2 \dots a_n a_n \dots a_2 a_1 \mid a_i \in \Delta\}$ ist jedoch **nicht deterministisch kontextfrei** (ohne Beweis).

Deterministisch kontextfreie Sprachen

Beachte: A priori folgt aus der Definition von deterministisch kontextfreien Sprachen nicht sofort, dass deterministisch kontextfreie Sprachen auch kontextfrei sind (Akzeptanz mit Endzuständen versus leeren Keller).

Dies ist aber der Fall: Aus einem deterministischen PDA

$M = (Z, \Sigma, \Gamma, \delta, z_0, \#, E)$ konstruieren wir einen (nicht-deterministischen) PDA $M' = (Z \cup \{z'_0, z_f\}, \Sigma, \Gamma \cup \{\#\prime\}, \delta', z'_0, \#\prime)$, wobei gilt:

$$\delta'(z'_0, \varepsilon, \#\prime) = \{(z_0, \#\#\prime)\}$$

$$\delta'(z, a, A) = \begin{cases} \delta(z, a, A) & \text{falls } (z \in Z \setminus E \text{ oder } a \in \Sigma), A \in \Gamma \\ \delta(z, a, A) \cup \{(z_f, \varepsilon)\} & \text{falls } z \in E, a = \varepsilon, A \in \Gamma \end{cases}$$

$$\delta(z, \varepsilon, \#\prime) = \{(z_f, \varepsilon)\} \text{ falls } z \in E$$

$$\delta(z_f, \varepsilon, A) = \{(z_f, \varepsilon)\} \text{ falls } A \in \Gamma \cup \{\#\prime\}$$

Dann gilt: $N(M') = D(M)$.

Deterministisch kontextfreie Sprachen

Die Konstruktion auf der vorherigen Folie zeigt auch, wie man einen (nicht-deterministischen) PDA, der mit Endzuständen akzeptiert, in einen (nicht-deterministischen) PDA, der mit leeren Keller akzeptiert, umwandelt.

Umgekehrt kann man einen (nicht-deterministischen) PDA $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$, der mit leeren Keller akzeptiert, in einen (nicht-deterministischen) PDA, der mit Endzuständen akzeptiert, wie folgt umwandeln:

Sei $M' = (Z \cup \{z'_0, z_f\}, \Sigma, \Gamma \cup \{\#'\}, \delta', z'_0, \#', \{z_f\})$, wobei gilt:

$$\delta'(z'_0, \varepsilon, \#') = \{(z_0, \#\#')\}$$

$$\delta'(z, a, A) = \delta(z, a, A) \text{ falls } z \in Z, a \in \Sigma \cup \{\varepsilon\}, A \in \Gamma$$

$$\delta(z, \varepsilon, \#') = \{(z_f, \varepsilon)\} \text{ für alle } z \in Z$$

Dann gilt: $N(M') = N(M)$.

Weitere Bemerkungen:

- **Effizienz:** Mit Hilfe von deterministischen Kellerautomaten hat man jetzt ein Verfahren zur Lösung des Wortproblems, das die Komplexität $O(n)$ hat. Hierbei ist n die Länge des Eingabewortes.

Dazu lässt man einfach den Automaten auf dem Wort arbeiten und überprüft ob man in einen Endzustand gelangt.

- **Deterministisch kontextfreie Grammatiken:** Da die Syntax von Sprachen einfacher mit Hilfe von Grammatiken als mit Hilfe von Kellerautomaten definiert werden kann, ist es notwendig, die zu deterministischen Kellerautomaten passende Klasse von **deterministisch kontextfreien Grammatiken** zu definieren.

Da dies nicht ganz einfach ist, gibt es hierzu mehrere Ansätze. Der bekannteste davon sind die sogenannten **$LR(k)$ -Grammatiken** (siehe Compilerbau und Syntaxanalyse).

Deterministisch kontextfreie Sprachen

Die Abschlusseigenschaften bei deterministisch kontextfreien Sprachen sehen etwas anders aus als bei kontextfreien Sprachen.

Satz (Abschluss unter Komplement)

Wenn L eine deterministisch kontextfreie Sprache ist, dann ist auch $\bar{L} = \Sigma^* \setminus L$ deterministisch kontextfrei.

Wir verzichten hier auf den recht technischen Beweis.

Kein Abschluss unter Schnitt

Es gibt deterministisch kontextfreie Sprachen L_1 und L_2 , so dass $L_1 \cap L_2$ nicht deterministisch kontextfrei ist.

Begründung:

Die Beispiel-Sprachen aus dem Argument, dass die kontextfreien Sprachen unter Schnitt nicht abgeschlossen sind, sind sogar deterministisch kontextfrei, ihr Schnitt jedoch noch nicht einmal kontextfrei:

$$L_1 = \{a^j b^k c^k \mid j \geq 0, k \geq 0\}$$

$$L_2 = \{a^k b^k c^j \mid j \geq 0, k \geq 0\}$$

Kein Abschluss unter Vereinigung

Es gibt deterministisch kontextfreie Sprachen L_1 und L_2 , so dass $L_1 \cup L_2$ nicht deterministisch kontextfrei ist.

Begründung:

Aus dem Abschluss unter Vereinigung und Komplement würde auch der Abschluss unter Schnitt folgen (wegen $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$).

Deterministisch kontextfreie Sprachen

Man hat jedoch sehr wohl Abschluss unter Schnitt mit regulären Sprachen:

Satz (Abschluss unter Schnitt mit regulären Sprachen)

Sei L eine deterministisch kontextfreie Sprache und R eine reguläre Sprache. Dann ist $L \cap R$ eine deterministisch kontextfreie Sprache.

Beweisidee: (analog der Kreuzprodukt-Konstruktion für NFAs)

Sei $M = (Z_1, \Sigma, \Gamma, \delta_1, z_0^1, \#, E_1)$ ein deterministischer PDA für L .

Sei $A = (Z_2, \Sigma, \delta_2, z_0^2, E_2)$ ein DFA für R .

Konstruktion eines deterministischen PDA M' für $L \cap R$:

$$M' = (Z_1 \times Z_2, \Sigma, \Gamma, \delta', (z_0^1, z_0^2), \#, E_1 \times E_2).$$

Hierbei ist die Überföhrungsfunktion δ wie folgt definiert ist:

$$\delta'((z_1, z_2), a, A) = \{((z'_1, z'_2), B_1 \cdots B_k) \mid (z'_1, B_1 \cdots B_k) \in \delta_1(z_1, a, A), \\ \delta_2(z_2, a) = z'_2, a \in \Sigma\}$$

$$\delta'((z_1, z_2), \varepsilon, A) = \{((z'_1, z_2), B_1 \cdots B_k) \mid (z'_1, B_1 \cdots B_k) \in \delta_1(z_1, \varepsilon, A)\}$$

Beachte: Die so definierte Überföhrungsfunktion erföüllt die in der Definition von deterministischen PDAs gestellten Voraussetzungen. □

Nochmal Abschlusseigenschaften

Mit der gleichen Technik und unter Ausnutzung der Tatsache, dass für allgemeine (nicht-deterministische) Kellerautomaten die Akzeptanz mit leerem Keller gleichmächtig zur Akzeptanz mit Endzustand ist, lässt sich auch folgendes zeigen:

Satz (Abschluss unter Schnitt mit regulären Sprachen II)

Sei L eine kontextfreie Sprache und R eine reguläre Sprache. Dann ist $L \cap R$ eine kontextfreie Sprache.

Wir betrachten nun noch Probleme für kontextfreie Sprachen und stellen fest, ob sie **entscheidbar** sind, d.h., ob es entsprechende Algorithmen zu ihrer Lösung gibt.

Wortproblem für eine kontextfreie Sprache L

Gegeben $w \in \Sigma^*$.

Gilt $w \in L$?

Ist die kontextfreie Sprache L durch eine kontextfreie Grammatik in Chomsky-Normalform gegeben, so kann das Wortproblem mit dem CYK-Algorithmus in $O(|w|^3)$ Zeit gelöst werden.

Ist L deterministisch kontextfrei und durch einen deterministischen PDA gegeben, so kann das Wortproblem für L sogar in Zeit $O(n)$ gelöst werden.

Entscheidbarkeit

Leerheitsproblem für kontextfreie Sprachen

Gegeben eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$.

Gilt $L(G) = \emptyset$?

Bestimme die Menge

$$W = \{A \in V \mid \exists w \in \Sigma^* : A \Rightarrow_G^* w\}$$

aller **produktiven** Variablen (Variablen mit denen ein Terminalwort abgeleitet werden kann):

$$W := \{A \in V \mid \exists w \in \Sigma^* : (A \rightarrow w) \in P\}$$

$$W' := \emptyset$$

while $W' \neq W$ **do**

$$W' := W$$

$$W := W \cup \{A \in V \mid \exists w \in (\Sigma \cup W)^* : (A \rightarrow w) \in P\}$$

endwhile

Dann gilt: $L(G) \neq \emptyset \iff S \in W$.

Endlichkeitsproblem für kontextfreie Sprachen

Gegeben eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$.

Ist $L(G)$ endlich?

Ohne Beschränkung der Allgemeinheit können wir davon ausgehen, dass G in Chomsky-Normalform ist.

Wir definieren einen Graphen (W, E) auf der Menge W der produktiven Variablen (siehe vorherige Folie) mit folgender Kantenrelation:

$$E = \{(A, B) \in W \times W \mid \exists C \in W : (A \rightarrow BC) \in P \text{ oder } (A \rightarrow CB) \in P\}$$

Behauptung: $|L(G)| = \infty \iff \exists A \in W : (S, A) \in E^*$ und $(A, A) \in E^+$.

Beachte: $(B, C) \in E^*$ (bzw. $(B, C) \in E^+$) bedeutet, dass es einen Pfad (bzw. einen nicht leeren Pfad, d.h. einen Pfad mit mindestens einer Kante) von B nach C in der binären Relation E gibt. $(B, B) \in E^*$ gilt immer!

“ \Leftarrow ”: Sei $A \in W$ so, dass $(S, A) \in E^*$ und $(A, A) \in E^+$.

Dann existieren in G Ableitungen der Form:

$$S \Rightarrow_G^* uAy, \quad A \Rightarrow_G^+ vAx, \quad A \Rightarrow_G^* w$$

mit $u, v, w, x, y \in \Sigma^*$.

Also gilt $S \Rightarrow_G^* uv^iwx^iy \in \Sigma^*$ für alle $i \geq 0$.

Da in der Ableitung $A \Rightarrow_G^+ vAx$ mindestens ein Ableitungsschritt gemacht wird, und G in Chomsky-Normalform ist, muss $vx \neq \varepsilon$ gelten.

Also ist $\{uv^iwx^iy \mid i \geq 0\}$ unendlich, weshalb $L(G)$ unendlich ist.

“ \Rightarrow ”: Sei $L(G)$ unendlich.

Sei n die Konstante aus dem Pumping-Lemma ($= 2^{|V|}$) und sei $z \in L(G)$ mit $|z| \geq n$ (so ein Wort z gibt es, wenn $L(G)$ unendlich ist!)

Im Beweis des Pumping-Lemmas haben wir gesehen, dass eine Variable A existiert mit Ableitungen $S \Rightarrow_G^* uAy$, $A \Rightarrow_G^+ vAx$, und $A \Rightarrow_G^* w$, wobei $z = uvwxy$.

Also ist A produktiv: $A \in W$.

Die Ableitungen $S \Rightarrow^* uAy$ und $A \Rightarrow^+ vAx$ (genauer, der Pfad im Syntaxbaum von der Wurzel S bis zum zweiten Vorkommen von A) zeigen, dass $(S, A) \in E^*$ und $(A, A) \in E^+$ gilt. \square

Beispiel:

Sei G die Grammatik in Chomsky-Normalform mit den Produktionen

$$S \rightarrow AC$$

$$A \rightarrow BC$$

$$B \rightarrow CA \mid b$$

$$C \rightarrow a$$

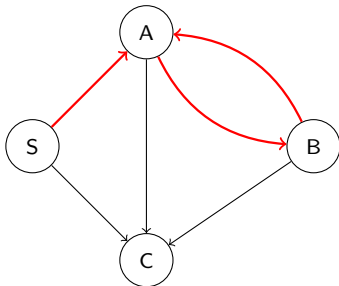
In diesem Fall ist $W = \{S, A, B, C\}$, d.h. alle Variablen sind produktiv:
Nach Durchlauf i durch die **while-Schleife** (Folie 305) erhalten wir

- ❶ für $i = 0$: $W = \{B, C\}$
- ❷ für $i = 1$: $W = \{A, B, C\}$
- ❸ für $i = 2$: $W = \{S, A, B, C\}$

Wegen $S \in W$ gilt $L(G) \neq \emptyset$.

Beispiel (Fortsetzung):

Der Graph (W, E) ist dann



Der rote Pfad zeigt, dass $L(G)$ unendlich ist.

Unentscheidbarkeit bei kontextfreien Sprachen

Folgende Probleme sind für kontextfreie Sprachen nicht entscheidbar, d.h. man kann zeigen, dass es keinen entsprechenden Algorithmus gibt:

- **Äquivalenzproblem:** Gegeben zwei kontextfreie Sprachen L_1, L_2 . Gilt $L_1 = L_2$?
- **Schnittproblem:** Gegeben zwei kontextfreie Sprachen L_1, L_2 . Gilt $L_1 \cap L_2 = \emptyset$?

Bemerkung: In der Vorlesung **Berechenbarkeit und Logik** werden wir sehen, wie man solche Unentscheidbarkeitsresultate zeigen kann.

Das **Schnittproblem** ist jedoch **entscheidbar**, wenn von einer der beiden Sprachen L_1 , L_2 bekannt ist, dass sie regulär ist, und sie als endlicher Automat gegeben ist.

Algorithmus:

- 1 In diesem Fall kann ein Kellerautomat M konstruiert werden (Konstruktion siehe weiter oben), der $L_1 \cap L_2$ akzeptiert.
- 2 Der Kellerautomat M kann dann in eine kontextfreie Grammatik G umgewandelt werden.
- 3 Durch Bestimmung der produktiven Variablen von G kann dann ermittelt werden, ob S nicht-produktiv ist und damit, ob $L_1 \cap L_2$ leer ist.

Entscheidbarkeit bei deterministisch kontextfreien Sprachen

Folgende Probleme sind für deterministisch kontextfreie Sprachen (repräsentiert durch einen deterministischen Kellerautomaten) entscheidbar:

- **Wortproblem für eine deterministisch kontextfreie Sprache L :** Gegeben $w \in \Sigma^*$. Gilt $w \in L$?

Mit einem deterministischen Kellerautomaten in $O(|w|)$ Zeit.

- **Leerheitsproblem:** Gegeben eine deterministisch kontextfreie Sprache L . Gilt $L = \emptyset$?

Siehe das entsprechende Entscheidungsverfahren für kontextfreie Sprachen.

Entscheidbarkeit bei deterministisch kontextfreien Sprachen

- **Endlichkeitsproblem:** Gegeben eine deterministisch kontextfreie Sprache L . Ist L endlich?

Siehe das entsprechende Entscheidungsverfahren für kontextfreie Sprachen.

- **Äquivalenzproblem:** Gegeben zwei deterministisch kontextfreie Sprachen L_1, L_2 . Gilt $L_1 = L_2$?

War lange offen und die Entscheidbarkeit wurde 1997 von Géraud Sénizergues gezeigt.

Unentscheidbarkeit bei deterministisch kontextfreien Sprachen

Folgende Probleme sind für deterministisch kontextfreie Sprachen nicht entscheidbar, d.h. man kann zeigen, dass es kein entsprechendes Verfahren gibt:

- **Schnittproblem:** Gegeben zwei deterministisch kontextfreie Sprachen L_1, L_2 . Gilt $L_1 \cap L_2 = \emptyset$?

Wie bei kontextfreien Sprachen ist dieses Problem jedoch entscheidbar, wenn eine der beiden Sprachen regulär ist.

- **Inklusionsproblem:** Gegeben zwei deterministisch kontextfreie Sprachen L_1, L_2 . Gilt $L_1 \subseteq L_2$?

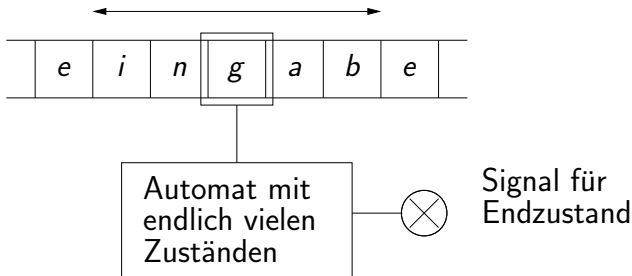
Im Rest der Vorlesung werden wir Maschinemodelle für the Chomsky-0 und Chomsky-1 Sprachen einführen.

- Chomsky-0 Sprachen: Turingmaschinen (benannt nach Alan Turing, 1912-1954)
- Chomsky-1 Sprachen: Linear beschränkte Automaten (eine Einschränkung von Turingmaschinen)

Turingmaschinen

Schematische Darstellung einer Turingmaschine:

Kopf kann sich nach links und rechts
bewegen und Zeichen überschreiben



Eigenschaften von Turingmaschinen:

- Wie endliche Automaten haben Turingmaschinen **endlich viele Zustände** und lesen eine **Eingabe von einem Band**, welches in Zellen (Felder) unterteilt ist.
- In jedem Feld des Bands steht ein Zeichen aus einem endlichen **Bandalphabet**. Ein Lesekopf fährt über das Band.
- Unterschied zu endlichen Automaten: der **Lesekopf kann sich nach links und rechts bewegen** und auch **Zeichen überschreiben**.
- Falls nur Zeichen des Eingabeworts überschrieben werden: Turingmaschine heißt **linear beschränkt** (Maschinenmodell für Chomsky-1-Sprachen).
- Falls der Lesekopf auch über den linken und rechten Rand des Eingabeworts hinauslaufen und dort schreiben kann: **allgemeine** Turingmaschine mit unbeschränktem Band (Maschinenmodell für Chomsky-0-Sprachen).

Turingmaschinen und Computer:

- Das **Konzept der Turingmaschine** wurde von **Alan Turing 1936** erfunden, noch bevor die ersten echten Computer gebaut wurden.
- Es ist nicht nur aus historischen Gründen interessant, sondern auch, weil es ein **sehr einfaches Berechnungsmodell** darstellt.

Wenn man zeigen will, dass etwas **nicht berechenbar** ist, dann ist es viel besser, dies mit einem **möglichst einfachen Berechnungsmodell** zu tun. (Natürlich sollte man vorher sicherstellen, dass dieses Berechnungsmodell äquivalent zu komplexeren Modellen ist.)

- **Analogie zu einem heutigen Computer:**
 - Kontrolle mit endlich vielen Zuständen \rightsquigarrow Programm
 - (Eingabe-)Band \rightsquigarrow Speicher

Beispiel 1: Turingmaschine, die eine Binärzahl auf dem Band um eins inkrementiert.

Idee:

- Kopf der Turingmaschine steht zunächst auf dem am weitesten links befindlichen (höchstwertigen) Bit der Binärzahl.
- Kopf nach rechts laufen lassen, bis ein Leerzeichen gefunden wird.
- Dann wieder nach links laufen und jede 1 durch 0 ersetzen, solange bis eine 0 oder ein Leerzeichen \square (ein spezielles Bandsymbol) auftaucht.
- Dieses Zeichen dann durch 1 ersetzen, bis zum Zahlenanfang laufen und in einen Endzustand übergehen.

Simulation (Inkrementiere Binärzahl 10111)



□ = Leerzeichen

Simulation (Inkrementiere Binärzahl 10111)



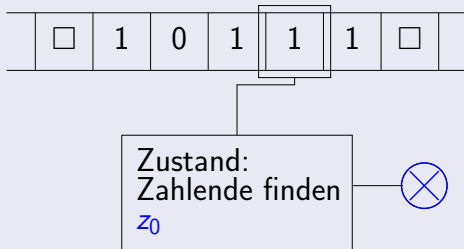
\square = Leerzeichen

Simulation (Inkrementiere Binärzahl 10111)



\square = Leerzeichen

Simulation (Inkrementiere Binärzahl 10111)



□ = Leerzeichen

Simulation (Inkrementiere Binärzahl 10111)



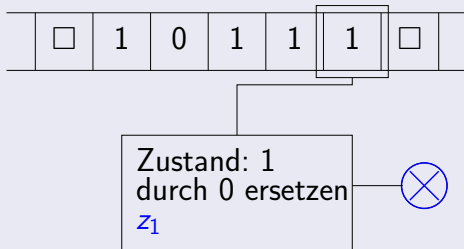
□ = Leerzeichen

Simulation (Inkrementiere Binärzahl 10111)



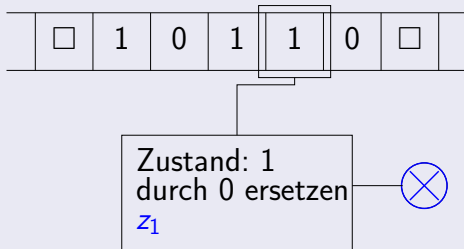
□ = Leerzeichen

Simulation (Inkrementiere Binärzahl 10111)



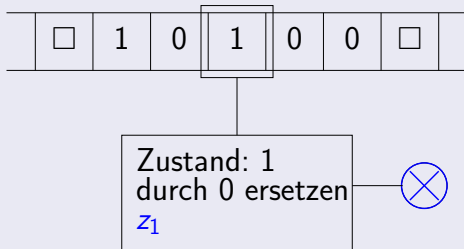
□ = Leerzeichen

Simulation (Inkrementiere Binärzahl 10111)



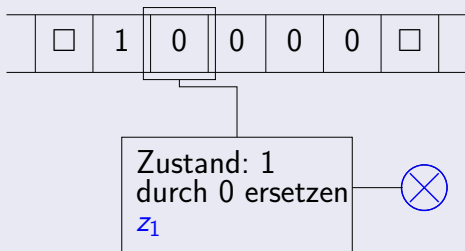
□ = Leerzeichen

Simulation (Inkrementiere Binärzahl 10111)



□ = Leerzeichen

Simulation (Inkrementiere Binärzahl 10111)



□ = Leerzeichen

Simulation (Inkrementiere Binärzahl 10111)



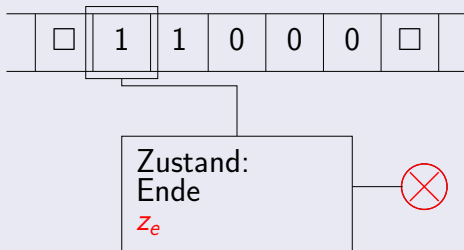
□ = Leerzeichen

Simulation (Inkrementiere Binärzahl 10111)



□ = Leerzeichen

Simulation (Inkrementiere Binärzahl 10111)



□ = Leerzeichen

Turingmaschine (Definition)

Eine **deterministische Turingmaschine** M ist ein 7-Tupel

$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$, wobei

- Z die endliche Menge der **Zustände**,
- Σ das endliche **Eingabealphabet**,
- Γ mit $\Gamma \supseteq \Sigma$ das endliche **Arbeitsalphabet** oder **Bandalphabet** (es soll $\Gamma \cap Z = \emptyset$ gelten),
- $z_0 \in Z$ der **Startzustand**,
- $E \subseteq Z$ die Menge der **Endzustände**,
- $\delta: (Z \setminus E) \times \Gamma \rightarrow Z \times \Gamma \times \{L, R, N\}$ die **Überföhrungsfunktion**, und
- $\square \in \Gamma \setminus \Sigma$ das **Leerzeichen** oder **Blank** ist.

Abkürzung: TM

Bedeutung der Überföhrungsfunktion:

sei $\delta(z, a) = (z', b, x)$ mit $z, z' \in Z$, $a, b \in \Gamma$, $x \in \{L, R, N\}$.

Falls die Turingmaschine sich im Zustand z befindet und in der Zelle, wo der (Schreib-Lese-)Kopf aktuell steht, das Bandsymbol a steht, so

- wechselt sie in den Zustand z' ,
- überschreibt das a in der aktuell gelesenen Zelle durch b und
- führt folgende Kopfbewegung aus.
 - Kopf ein Feld nach links, falls $x = L$.
 - Kopf bleibt stehen, falls $x = N$.
 - Kopf ein Feld nach rechts, falls $x = R$.

Beachte: da $\delta: (Z \setminus E) \times \Gamma \rightarrow Z \times \Gamma \times \{L, R, N\}$ (d.h. δ ist auf Paaren (z, a) mit $z \in E$ nicht definiert), terminiert die Turingmaschine genau dann, wenn der aktuelle Zustand ein Endzustand aus E ist.

Neben **deterministischen Turingmaschinen** gibt es auch **nichtdeterministische Turingmaschinen**.

Überföhrungsfunktion für nichtdeterministische Turingmaschinen:

$$\delta: (Z \setminus E) \times \Gamma \rightarrow 2^{Z \times \Gamma \times \{L,R,N\}}.$$

Jedem Zustand und Bandsymbol wird eine (eventuell leere) Menge von möglichen Aktionen zugeordnet.

Zunächst wollen wir uns aber auf deterministische Turingmaschinen konzentrieren.

Beispiel: Turingmaschine zur Inkrementierung einer Binärzahl

$$M = (\{z_0, z_1, z_2, z_e\}, \{0, 1\}, \{0, 1, \square\}, \delta, z_0, \square, \{z_e\}) \quad \text{mit}$$

Überföhrungsfunktion: rechtes Zahlende finden

$$\delta(z_0, 0) = (z_0, 0, R)$$

$$\delta(z_0, 1) = (z_0, 1, R)$$

$$\delta(z_0, \square) = (z_1, \square, L)$$

Überföhrungsfunktion: 1 durch 0 ersetzen

$$\delta(z_1, 0) = (z_2, 1, L)$$

$$\delta(z_1, 1) = (z_1, 0, L)$$

$$\delta(z_1, \square) = (z_e, 1, N)$$

Überföhrungsfunktion: zurück zum linken Zahlanfang (ist nicht so wichtig)

$$\delta(z_2, 0) = (z_2, 0, L)$$

$$\delta(z_2, 1) = (z_2, 1, L)$$

$$\delta(z_2, \square) = (z_e, \square, R)$$

Beispiel 2: Turingmaschinen zur Spracherkennung

Wir suchen eine Turingmaschine, die die Sprache $L = \{a^{2^n} \mid n \geq 0\}$ (nicht kontextfrei!) erkennt.

Idee:

- Kopf steht zunächst am linken Ende der Folge von a 's.
- Links neben der Folge von a 's die Binärzahl 0 aufs Band schreiben.
- a 's nacheinander durch ein anderes Zeichen ($\#$) ersetzen. Nach jeder Ersetzung nach links zum Zähler laufen und diesen um eins inkrementieren.
- Sobald alle a 's verschwunden sind (nach dem letzten $\#$ kommt ein \square), überprüfen, ob der Zähler die Form $10 \cdots 0$ hat.

Beachte: Eine Zahl n ist eine Zweierpotenz genau dann, wenn ihre Binärdarstellung die Form $10 \cdots 0$ hat.

Turingmaschinen

Wie bei anderen Maschinenmodellen (z.B. Kellerautomaten) gibt es auch bei Turingmaschinen den Begriff einer **Konfiguration**, d.h., einer Momentaufnahme einer Turingmaschinen-Berechnung.

Konfiguration (Definition)

Eine **Konfiguration** einer Turingmaschine ist ein Wort

$$k \in \Gamma^* Z \Gamma^+.$$

Bedeutung: $k = \alpha z \beta$ mit $z \in Z$, $\alpha \in \Gamma^*$, $\beta \in \Gamma^+$
(β ist also ein nicht-leeres Wort)

- links vom Kopf steht auf dem Band das Wort $\cdots \square \alpha$
- ab der Zelle, wo der Kopf gerade steht, und rechts davon steht auf dem Band das Wort $\beta \square \cdots$. Der Kopf steht auf dem ersten Zeichen von β (hier ist $\beta \neq \varepsilon$ wichtig).
- $z \in Z$ ist der aktuelle Zustand.

Turingmaschinen

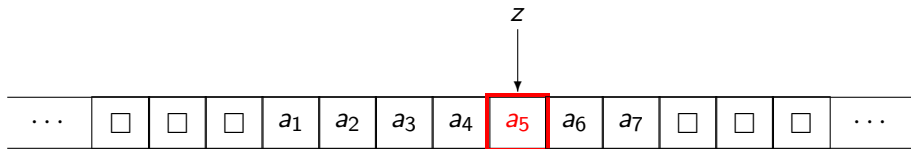
$\dots \square$ steht hier für eine unendliche nach links laufende Folge von \square 'en.

$\square \dots$ steht für eine unendliche nach rechts laufende Folge von \square 'en.

Das Band ist also nach links und rechts unbeschränkt, aber nur ein endlicher Abschnitt des Band enthält Bandsymbole aus $\Gamma \setminus \{\square\}$.

Beachte: die Wörter $\alpha z \beta$ und $\square \alpha z \beta \square$ beschreiben die gleiche Konfiguration (die Blanks am Anfang und Ende von $\square \alpha z \beta \square$ sind sozusagen überflüssig).

Beispiel: Grafische Darstellung der Konfiguration $a_1 a_2 a_3 a_4 z a_5 a_6 a_7$



Definition einer Übergangsrelation \vdash_M , die beschreibt, welche Konfigurationsübergänge möglich sind.

Keine Bewegung

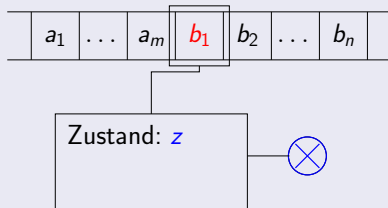
Es gilt: $a_1 \cdots a_m \mathbf{z} \mathbf{b}_1 b_2 \cdots b_n \vdash_M a_1 \cdots a_m \mathbf{z}' \mathbf{c} b_2 \cdots b_n$,
falls $\delta(\mathbf{z}, \mathbf{b}_1) = (\mathbf{z}', \mathbf{c}, \mathbf{N})$ ($m \geq 0, n \geq 1$).

Turingmaschinen

Definition einer Übergangsrelation \vdash_M , die beschreibt, welche Konfigurationsübergänge möglich sind.

Keine Bewegung

Es gilt: $a_1 \cdots a_m \mathbf{z} \mathbf{b}_1 b_2 \cdots b_n \vdash_M a_1 \cdots a_m \mathbf{z}' \mathbf{c} b_2 \cdots b_n$,
falls $\delta(\mathbf{z}, \mathbf{b}_1) = (\mathbf{z}', \mathbf{c}, \mathbf{N})$ ($m \geq 0, n \geq 1$).

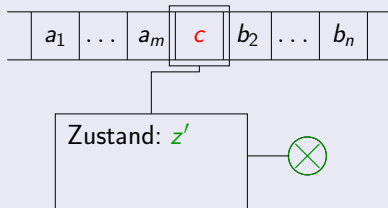


Turingmaschinen

Definition einer Übergangsrelation \vdash_M , die beschreibt, welche Konfigurationsübergänge möglich sind.

Keine Bewegung

Es gilt: $a_1 \cdots a_m \mathbf{z} \mathbf{b}_1 b_2 \cdots b_n \vdash_M a_1 \cdots a_m \mathbf{z}' \mathbf{c} b_2 \cdots b_n$,
falls $\delta(\mathbf{z}, \mathbf{b}_1) = (\mathbf{z}', \mathbf{c}, \mathbf{N})$ ($m \geq 0, n \geq 1$).



Definition einer Übergangsrelation \vdash_M , die beschreibt, welche Konfigurationsübergänge möglich sind.

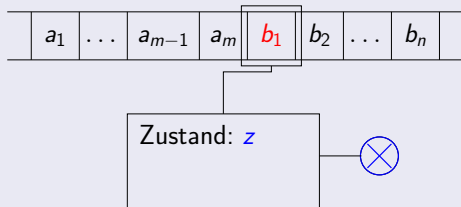
Schritt nach links

Es gilt: $a_1 \cdots a_{m-1} a_m \mathbf{z} \mathbf{b}_1 b_2 \cdots b_n \vdash_M a_1 \cdots a_{m-1} \mathbf{z}' a_m \mathbf{c} b_2 \cdots b_n$,
falls $\delta(\mathbf{z}, \mathbf{b}_1) = (\mathbf{z}', \mathbf{c}, \mathbf{L})$ ($m \geq 1, n \geq 1$).

Definition einer Übergangsrelation \vdash_M , die beschreibt, welche Konfigurationsübergänge möglich sind.

Schritt nach links

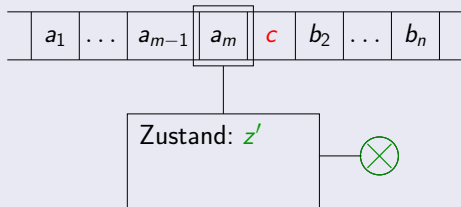
Es gilt: $a_1 \cdots a_{m-1} a_m z b_1 b_2 \cdots b_n \vdash_M a_1 \cdots a_{m-1} z' a_m c b_2 \cdots b_n$,
falls $\delta(z, b_1) = (z', c, L)$ ($m \geq 1, n \geq 1$).



Definition einer Übergangsrelation \vdash_M , die beschreibt, welche Konfigurationsübergänge möglich sind.

Schritt nach links

Es gilt: $a_1 \cdots a_{m-1} a_m z b_1 b_2 \cdots b_n \vdash_M a_1 \cdots a_{m-1} z' a_m c b_2 \cdots b_n$,
falls $\delta(z, b_1) = (z', c, L)$ ($m \geq 1, n \geq 1$).



Definition einer Übergangsrelation \vdash_M , die beschreibt, welche Konfigurationsübergänge möglich sind.

Schritt nach rechts

Es gilt: $a_1 \cdots a_m \textcolor{red}{z} \textcolor{red}{b}_1 b_2 \cdots b_n \vdash_M a_1 \cdots a_m \textcolor{red}{c} \textcolor{red}{z}' b_2 \cdots b_n$,

falls $\delta(\textcolor{red}{z}, \textcolor{red}{b}_1) = (\textcolor{red}{z}', \textcolor{red}{c}, \textcolor{teal}{R}) \quad (m \geq 0, n \geq 2)$.

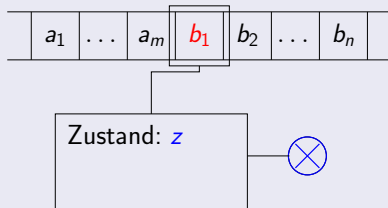
Turingmaschinen

Definition einer Übergangsrelation \vdash_M , die beschreibt, welche Konfigurationsübergänge möglich sind.

Schritt nach rechts

Es gilt: $a_1 \cdots a_m \mathbf{z} \mathbf{b_1} b_2 \cdots b_n \vdash_M a_1 \cdots a_m \mathbf{c} \mathbf{z'} b_2 \cdots b_n$,

falls $\delta(\mathbf{z}, \mathbf{b_1}) = (\mathbf{z'}, \mathbf{c}, \mathbf{R})$ ($m \geq 0, n \geq 2$).



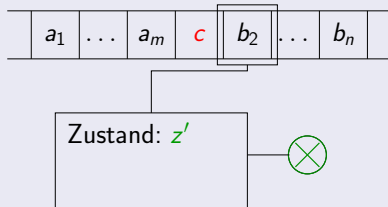
Turingmaschinen

Definition einer Übergangsrelation \vdash_M , die beschreibt, welche Konfigurationsübergänge möglich sind.

Schritt nach rechts

Es gilt: $a_1 \cdots a_m z b_1 b_2 \cdots b_n \vdash_M a_1 \cdots a_m c z' b_2 \cdots b_n$,

falls $\delta(z, b_1) = (z', c, R)$ ($m \geq 0, n \geq 2$).



Sonderfälle: Bandende erreicht \rightsquigarrow zusätzliches Leerzeichen muss hinzugefügt werden

Linkes Bandende

Es gilt: $z b_1 b_2 \cdots b_n \vdash_M z' \square c b_2 \cdots b_n$,

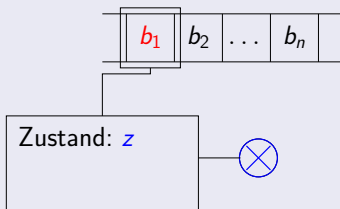
falls $\delta(z, b_1) = (z', c, L)$.

Sonderfälle: Bandende erreicht \rightsquigarrow zusätzliches Leerzeichen muss hinzugefügt werden

Linkes Bandende

Es gilt: $zb_1b_2 \cdots b_n \vdash_M z' \square cb_2 \cdots b_n$,

falls $\delta(z, b_1) = (z', c, L)$.

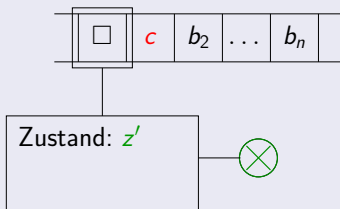


Sonderfälle: Bandende erreicht \rightsquigarrow zusätzliches Leerzeichen muss hinzugefügt werden

Linkes Bandende

Es gilt: $z b_1 b_2 \cdots b_n \vdash_M z' \square c b_2 \cdots b_n$,

falls $\delta(z, b_1) = (z', c, L)$.



Sonderfälle: Bandende erreicht \rightsquigarrow zusätzliches Leerzeichen muss hinzugefügt werden

Rechtes Bandende

Es gilt: $a_1 \cdots a_m z b_1 \vdash_M a_1 \cdots a_m c z' \square$,

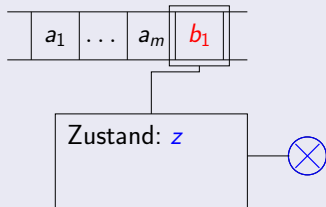
falls $\delta(z, b_1) = (z', c, R)$.

Sonderfälle: Bandende erreicht \rightsquigarrow zusätzliches Leerzeichen muss hinzugefügt werden

Rechtes Bandende

Es gilt: $a_1 \cdots a_m z b_1 \vdash_M a_1 \cdots a_m c z' \square$,

falls $\delta(z, b_1) = (z', c, R)$.

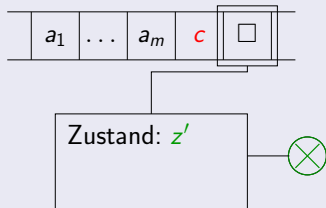


Sonderfälle: Bandende erreicht \rightsquigarrow zusätzliches Leerzeichen muss hinzugefügt werden

Rechtes Bandende

Es gilt: $a_1 \cdots a_m z b_1 \vdash_M a_1 \cdots a_m c z' \square$,

falls $\delta(z, b_1) = (z', c, R)$.



Akzeptierte Sprache (Definition)

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ eine Turingmaschine. Dann ist die von M **akzeptierte Sprache**:

$$T(M) = \{x \in \Sigma^* \mid \exists k \in \Gamma^* E \Gamma^+ : z_0 x \square \vdash_M^* k\}.$$

Akzeptierte Sprache: Alle Eingabe-Wörter, mit denen die Turingmaschine in einen Endzustand gelangen kann. Dabei startet die Turingmaschine im Anfangszustand z_0 , der Kopf befindet sich auf dem ersten Zeichen des Eingabe-Wortes. Falls dieses nicht existiert (Eingabe x ist das leere Wort) liest der Kopf ein Blank \square .

Beispiel: Die Berechnung auf der nächsten Folie entspricht der Simulation auf Folie 321 für die Turingmaschine von Folie 325–326.

$z_0 10111 \vdash_M 1z_0 0111$

$\vdash_M 10z_0 111$

$\vdash_M 101z_0 11$

$\vdash_M 1011z_0 1$

$\vdash_M 10111z_0 \square$

$\vdash_M 1011z_1 1\square$

$\vdash_M 101z_1 10\square$

$\vdash_M 10z_1 100\square$

$\vdash_M 1z_1 0000\square$

$\vdash_M z_2 11000\square$

$\vdash_M z_2 \square 11000\square$

$\vdash_M \square z_e 11000\square$

wegen $\delta(z_0, 1) = (z_0, 1, R)$

wegen $\delta(z_0, 0) = (z_0, 0, R)$

wegen $\delta(z_0, 1) = (z_0, 1, R)$

wegen $\delta(z_0, 1) = (z_0, 1, R)$

wegen $\delta(z_0, 1) = (z_0, 1, R)$

wegen $\delta(z_0, \square) = (z_1, \square, L)$

wegen $\delta(z_1, 1) = (z_1, 0, L)$

wegen $\delta(z_1, 1) = (z_1, 0, L)$

wegen $\delta(z_1, 1) = (z_1, 0, L)$

wegen $\delta(z_1, 0) = (z_2, 1, L)$

wegen $\delta(z_2, 1) = (z_2, 1, L)$

wegen $\delta(z_2, \square) = (z_e, \square, R)$

Für **nicht-deterministische Turingmaschinen** müssen die Definitionen folgendermaßen angepaßt werden:

- Falls sich die Turingmaschine im Zustand z befindet und das Zeichen b auf dem Band steht, sind alle Konfigurationsübergänge möglich, die durch die Menge $\delta(z, b)$ beschrieben werden.
- Ein Wort ist akzeptiert, wenn es eine mögliche Folge von Konfigurationen gibt, die zu einem Endzustand führt, auch wenn andere Folgen in Sackgassen geraten oder unendlich lang sind, ohne dabei je einen Endzustand zu erreichen.

Linear beschränkte Automaten

Wir definieren nun ein Maschinenmodell für Chomsky-1-Sprachen (erzeugt durch monotone Grammatiken): **linear beschränkte Automaten**, die **niemals außerhalb der Eingabe** arbeiten dürfen.

Linear beschränkte Automaten

Ein (nicht)deterministischer **linear beschränkter Automat (LBA)** ist ein Tupel $A = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$, welches den gleichen Eigenschaften wie eine (nicht)deterministische Turingmaschine genügt, nur dass (i) A nicht das Blanksymbol \square durch ein Nicht-Blanksymbol überschreiben darf und (ii) A nicht ein Nicht-Blanksymbol durch \square überschreiben darf.

Die Relation \vdash_A ist wie für eine TM definiert, nur dass wir die Sonderfälle für das linke und rechte Bandende (Folie 333 und 334) weglassen.

Die von dem LBA A **akzeptierte Sprache** ist

$$T(A) = \{w \in \Sigma^* \mid \exists k \in \Gamma^* E \Gamma^+ : z_0 w \square \vdash_A^* k\}$$

Bemerkung: Das abschließende Blanksymbol \square erlaubt A , das rechte Bandende zu erkennen. Es dient sozusagen als rechtes Begrenzungssymbol.

Satz 3 (Kuroda)

Eine Sprache L wird von einem nichtdeterministischen LBA erkannt, genau dann, wenn es eine Typ-1-Grammatik G mit $L = L(G)$ gibt.

Beweis:

Sei zunächst $G = (V, \Sigma, P, S)$ eine Typ-1-Grammatik, d.h. es gilt $|\ell| \leq |r|$ für alle $(\ell, r) \in P$ (einzige Ausnahme: $S \rightarrow \varepsilon$, siehe ε -Sonderregelung, Folie 35).

Sei $w \in \Sigma^*$ eine Eingabe.

Wir simulieren nun eine Ableitung $S \Rightarrow_G^* w$ **rückwärts** mittels eines nichtdeterministischen LBA A .

$B[i]$ ist im folgenden das i -te Zeichen auf dem Band des LBA A .

$\tilde{\square}$ ist ein neues Bandsymbol, welches als Kopie des Blank-Symbols \square fungiert.

Der LBA A arbeitet wie folgt:

- 1 A bewegt den Kopf zum linkensten Symbol auf dem Band, “rät” nichtdeterministisch eine Regel $(\ell, r) \in P$ und merkt sich diese im Zustand.
- 2 Nun läuft der Kopf von A nach rechts zu einer nichtdeterministisch geratenen Position i .
- 3 Falls $B[i] \cdots B[i + |r| - 1] = r$ gilt, schreibt A auf den Bandabschnitt $B[i] \cdots B[i + |\ell| - 1]$ das Wort ℓ . Ansonsten gehe wieder zu (1).

- 4 Falls $|\ell| < |r|$ gilt, muss M nun jedes Zeichen auf dem Band ab Position $i + |r|$ um genau $|r| - |\ell|$ viele Positionen nach links verschieben.

Falls dabei auf dem Band eine Satzform der Länge $< |w|$ entsteht, füllt der LBA am rechten Ende die Satzform mit Symbolen $\tilde{\square}$ auf (beachte: A darf nicht Nicht-Blanks mit dem eigentlichen Blank-Symbol \square überschreiben).

- 5 A akzeptiert, falls das aktuelle Band mit $S\square$ oder $S\tilde{\square}$ beginnt, ansonsten gehe wieder zu (1).

Falls $S \rightarrow \varepsilon$ eine Produktion in P ist (d.h. $\varepsilon \in L(G)$), kann A bei Lesen von \square direkt aus dem Anfangszustand in einen Endzustand übergehen.

Chomsky-1-Sprachen

Für diesen LBA A gilt $L(G) = T(A)$.

Wir zeigen nun die andere Richtung.

Das folgende Lemma wird hilfreich sein.

Lemma 4

Sei $G = (V, \Sigma \cup \{r\}, P, S)$ eine Typ-1-Grammatik mit $r \notin \Sigma$ und $L(G) \subseteq \Sigma^* r$. Dann existiert eine Typ-1-Grammatik G' mit

$$L(G') = \{w \in \Sigma^* \mid wr \in L(G)\}.$$

Beweis:

O.B.d.A. können wir annehmen, dass gilt:

- Für jede Produktion $(u, v) \in P$ gilt $0 \leq |v| - |u| \leq 1$
- S kommt nicht in einer rechten Seite vor.

Wir definieren eine neue Variablenmenge V' durch

$$V' = V \cup \{r\} \cup \{A_{ab} \mid a, b \in V \cup \Sigma \cup \{r\}\}.$$

Intuition: A_{ab} ist ein Nichtterminal, welches die letzten beiden Symbole ab in einer Satzform zu einem Symbol zusammenfasst.

Die neue Produktionsmenge P' der Grammatik G' besteht aus den Produktionen auf der nächsten Folie.

In allen Fällen ist dabei $a, b, c, d \in V \cup \Sigma \cup \{r\}$ und $x, y \in (V \cup \Sigma \cup \{r\})^*$.

- $S \rightarrow \varepsilon$ falls $r \in L(G)$,
- $S \rightarrow A_{ab}$ falls $S \Rightarrow_G^* ab$
- $xA_{ab} \rightarrow yA_{cd}$ falls $(xab \rightarrow ycd) \in P$
- $xA_{ab} \rightarrow yA_{cb}$ falls $(xa \rightarrow yc) \in P$
- $A_{ab} \rightarrow A_{ac}$ falls $(b \rightarrow c) \in P$
- $A_{ab} \rightarrow aA_{cd}$ falls $(b \rightarrow cd) \in P$
- alle Produktionen aus P
- $A_{ar} \rightarrow a$ falls $a \in \Sigma$

Dann ist $G' = (V', \Sigma, P', S)$ die gesuchte Grammatik.



Völlig analog zeigt man:

Lemma 5

Sei $G = (V, \Sigma \cup \{\ell\}, P, S)$ eine Typ-1-Grammatik mit $\ell \notin \Sigma$ und $L(G) \subseteq \ell \Sigma^*$. Dann existiert eine Typ-1-Grammatik G' mit

$$L(G') = \{w \in \Sigma^* \mid \ell w \in L(G)\}.$$

und durch Anwendung beider Lemmata:

Lemma 6

Sei $G = (V, \Sigma \cup \{\ell, r\}, P, S)$ eine Typ-1-Grammatik mit $\ell, r \notin \Sigma$ und $L(G) \subseteq \ell \Sigma^* r$. Dann existiert eine Typ-1-Grammatik G' mit

$$L(G') = \{w \in \Sigma^* \mid \ell w r \in L(G)\}.$$

Nun zurück zum Beweis von Satz 3. Sei $A = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ ein LBA.

Aufgrund von Lemma 6 genügt es, eine Typ-1-Grammatik für die Sprache $\{\$w\square \mid w \in T(A)\}$ anzugeben ($\$$ ist ein neues Terminalsymbol).

Hierzu simulieren wir A rückwärts mittels der Typ-1-Grammatik $G = (V, \Sigma \cup \{\$, \square\}, P, S)$ mit der Variablenmenge

$$V = \{S, B, C\} \cup (\Gamma \setminus (\Sigma \cup \{\square\})) \cup (Z \times \Gamma)$$

und der folgenden Produktionsmenge P (Folie 346–348):

$$S \rightarrow \$B$$

$$B \rightarrow aB \mid (z, a)C \mid (z, \square) \quad \text{für alle } a \in \Gamma \setminus \{\square\}, z \in E$$

$$C \rightarrow aC \mid \square \quad \text{für alle } a \in \Gamma \setminus \{\square\}$$

Mit den Regeln für S , B und C kann man beliebige Wörter der Form

$$\$a_1a_2\cdots a_n(z,a)b_1b_2\cdots b_m\Box \quad \text{oder} \quad \$a_1a_2\cdots a_n(z,\Box)$$

mit $a_1, \dots, a_n, a, b_1, \dots, b_m \in \Gamma \setminus \{\Box\}$ und $z \in E$ erzeugen.

Dies sind genau die Konfigurationen, in denen A akzeptiert, bis auf das Detail, dass wir den Zustand z und das aktuell gelesene Bandsymbol a zu einem Nichtterminal $(z, a) \in Z \times \Gamma$ zusammenfassen (macht den Rest der Grammatik etwas einfacher).

Mit den folgenden Produktionen wird der LBA A **rückwärts** simuliert:

$$\begin{aligned}(z', a') &\rightarrow (z, a) && \text{für alle } (z', a', N) \in \delta(z, a) \\ a'(z', b) &\rightarrow (z, a)b && \text{für alle } (z', a', R) \in \delta(z, a), b \in \Gamma \\ (z', b)a' &\rightarrow b(z, a) && \text{für alle } (z', a', L) \in \delta(z, a), b \in \Gamma\end{aligned}$$

Wird mittels der Produktionen auf der vorherigen Folie aus der zu Beginn erzeugten akzeptierenden Konfiguration schließlich eine initiale Konfiguration der Form

$$$(z_0, c_1)c_2 \cdots c_n\Box \quad \text{oder} \quad $(z_0, \Box)$$

abgeleitet (beachte: z_0 ist der Anfangszustand des LBA A), so wird mittels der folgenden Produktionen das Wort $$(c_1c_2 \cdots c_n\Box$ beziehungsweise $$(\Box$ abgeleitet:$$

$$$(z_0, a) \rightarrow \$a \quad \text{für alle } a \in \Sigma$$

$$$(z_0, \Box) \rightarrow \$\Box$$

Dann gilt $L(G) = \{ \$w\Box \mid w \in T(A) \}$.



Satz 7 (Turingmaschinen und Chomsky-0-Sprachen)

Eine Sprache L wird von einer nichtdeterministischen Turingmaschine erkannt, genau dann, wenn es eine Typ-0-Grammatik G mit $L = L(G)$ gibt.

Beweisidee: durch Modifikation des Beweises von Satz 3:

Grammatiken \rightarrow Turingmaschinen: hier muss bei der Simulation der Grammatik auf dem Turingmaschinen-Band bei verkürzenden Regeln (linke Seite ist länger als rechte Seite) der Bandinhalt auseinandergeschoben werden.

Turingmaschinen \rightarrow Grammatiken: hier muss dafür gesorgt werden, dass die Grammatik bei Simulation der Turingmaschine links und rechts Leerzeichen erzeugen kann und diese nach erfolgreicher Berechnung auch wieder löscht.

Chomsky-0-Sprachen

Formal: Wir simulieren eine TM $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ mittels der Typ-0 Grammatik $G = (\{S, B, C, \$_1, \$_2\} \cup (\Gamma \setminus \Sigma) \cup (Z \times \Gamma), \Sigma, P, S)$ mit der folgenden Produktionsmenge P :

$$S \rightarrow \$_1 B$$

$$B \rightarrow aB \mid (z, a)C \quad \text{für alle } a \in \Gamma, z \in E$$

$$C \rightarrow aC \mid \$_2 \quad \text{für alle } a \in \Gamma$$

$$(z', a') \rightarrow (z, a) \quad \text{für alle } (z', a', N) \in \delta(z, a)$$

$$a'(z', b) \rightarrow (z, a)b \quad \text{für alle } (z', a', R) \in \delta(z, a), b \in \Gamma$$

$$(z', b)a' \rightarrow b(z, a) \quad \text{für alle } (z', a', L) \in \delta(z, a), b \in \Gamma$$

$$\$_1 \square \rightarrow \$_1$$

$$\$_1(z_0, a) \rightarrow a \quad \text{für alle } a \in \Sigma$$

$$\square \$_2 \rightarrow \$_2$$

$$a \$_2 \rightarrow a \quad \text{für alle } a \in \Sigma$$

$$\$_1(z_0, \square) \$_2 \rightarrow \varepsilon$$

Wieder wird die Turingmaschine M rückwärts simuliert.

Die verkürzenden Regeln $\$1\Box \rightarrow \1 und $\Box\$2 \rightarrow \2 erlauben es Blanksymbole am Anfang und Ende der Konfiguration zu löschen.

Das ist wichtig um am Ende aus einer durch Rückwärtssimulation der TM abgeleiteten initialen Konfiguration

$$\$_1\Box \cdots \Box(z_0, c_1)c_2 \cdots c_n\Box \cdots \Box\$2 \quad \text{bzw.} \quad \$_1\Box \cdots \Box(z_0, \Box)\Box \cdots \Box\$2$$

zunächst

$$\$_1(z_0, c_1)c_2 \cdots c_n\$2 \quad \text{bzw.} \quad \$_1(z_0, \Box)\$2$$

abzuleiten.

Hieraus wird dann mittels der Produktionen $\$_1(z_0, c_1) \rightarrow c_1$ und $c_n\$2 \rightarrow c_n$ bzw. $\$_1(z_0, \Box)\$2 \rightarrow \varepsilon$ das Eingabewort $c_1c_2 \cdots c_n$ bzw. ε abgeleitet.

Es gilt dann $L(G) = T(M)$. □

Ergebnisse für Chomsky-1- und Chomsky-0-Sprachen

Satz 8 (Abschluss unter Komplement von Typ-1-Sprachen, Immerman, Szelepcsényi)

Wenn L eine Typ-1-Sprache ist, dann ist auch $\bar{L} = \Sigma^* \setminus L$ eine Typ-1-Sprache.

Ein Beweis wird in der Vorlesung **Strukturelle Komplexitätstheorie** vorgestellt.

Satz 9 (Nicht-Abschluss unter Komplement von Typ-0-Sprachen)

Es gibt eine Typ-0-Sprache $L \subseteq \Sigma^*$, so dass $\bar{L} = \Sigma^* \setminus L$ keine Typ-0-Sprache ist.

Begründung und Beispiele in der Vorlesung **Berechenbarkeit und Logik**.

Satz 10 (Determinismus und Nichtdeterminismus bei Turingmaschinen)

Zu jeder nichtdeterministischen Turingmaschine gibt es eine deterministische Turingmaschine, die dieselbe Sprache akzeptiert.

Beweis:

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ eine nichtdeterministische TM, d. h.

$$\delta: (Z \setminus E) \times \Gamma \rightarrow 2^{Z \times \Gamma \times \{L, R, N\}}.$$

Idee: Wir konstruieren eine deterministische Turingmaschine, die bei Eingabe $x \in \Sigma^*$ systematisch nach einer **erfolgreichen Berechnung** von M sucht.

Sei $\# \notin Z \cup \Gamma$ ein neues Symbol.

Ergebnisse für Chomsky-1- und Chomsky-0-Sprachen

Eine erfolgreiche Berechnung von M auf Eingabe x ist ein Wort der Form

$$k_0 \# k_1 \# \cdots k_{m-1} \# k_m$$

mit folgenden Eigenschaften:

- ① $k_0, k_1, \dots, k_m \in \Gamma^* Z \Gamma^+$
- ② $k_0 = z_0 x \square$.
- ③ $\forall i \in \{0, 1, \dots, m-1\} : k_i \vdash_M k_{i+1}$
- ④ $k_m \in \Gamma^* E \Gamma^+$

Offensichtlich gilt $x \in T(M)$ genau dann, wenn eine erfolgreiche Berechnung von M auf Eingabe x existiert.

Eine deterministische Turingmaschine M' kann bei Eingabe von x und $w \in (Z \cup \Gamma \cup \{\#\})^*$ feststellen, ob w eine erfolgreiche Berechnung von M auf Eingabe x ist (geht sogar mit einem deterministischen LBA).

Hierzu muss M' lediglich die vier Eigenschaften (1)–(4) überprüfen.

Ergebnisse für Chomsky-1- und Chomsky-0-Sprachen

Nun muss man nur noch eine deterministische Turingmaschine M'' konstruieren, die systematisch der Reihe nach alle Wörter $w \in (Z \cup \Gamma \cup \{\#\})^*$ durchgeht, und jedesmal (mittels M') überprüft, ob w eine erfolgreiche Berechnung von M auf Eingabe x ist.

“Systematisch der Reihe nach” kann hier z. B. mittels einer **längenlexikographischen Ordnung** formalisiert werden.

Sei zunächst \sqsubset eine beliebige lineare Ordnung auf dem Alphabet $\Omega = Z \cup \Gamma \cup \{\#\}$.

Die zu \sqsubset gehörende längenlexikographische Ordnung \sqsubset_{lex} auf Ω^* ist wie folgt definiert:

Für $u, v \in \Omega^*$ gilt $u \sqsubset_{\text{lex}} v$ genau dann, wenn

- $|u| < |v|$ (u ist kürzer als v) oder
- $|u| = |v|$ und es gibt $x, y, z \in \Omega^*$, $a, b \in \Omega$ mit $u = xay$, $v = xbz$, und $a \sqsubset b$ (an der ersten Position, wo sich u und v unterscheiden, steht in u das kleinere Symbol).

Ergebnisse für Chomsky-1- und Chomsky-0-Sprachen

Grobstruktur der deterministischen Turingmaschine M'' :

- 1 Initialisiere hinter der Eingabe x auf dem Band ein Wort $w \in (Z \cup \Gamma \cup \{\#\})^*$ mit ε
- 2 Überprüfe mittels M' ob w eine erfolgreiche Berechnung von M auf Eingabe x ist.
Falls ja, gehe in einen Endzustand über, sonst gehe zu (3)
- 3 Inkrementiere w , d. h. überschreibe w mit dem längenlexikographisch nächsten Wort w' (formal: w' ist das bezüglich \sqsubset_{lex} kleinste Wort mit $w \sqsubset_{\text{lex}} w'$).
- 4 Gehe zu (2). □

Determinismus und Nichtdeterminismus bei LBAs (erstes LBA-Problem)

Es ist nicht bekannt, ob für jeden LBA A ein deterministischer LBA A' mit $T(A) = T(A')$ existiert.