Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Christian Ivicevic

Sommersemester 2015
Studentisches Tutorium
Übungsblatt 5
18. Mai 2015

Theoretische Informatik

– Chomsky-Normalform, CYK-Algorithmus,
Pumping-Lemma für kontextfreie Sprachen, Kanonischer Minimalautomat –

Chomsky-Normalform bei kontextfreien Grammatiken

Bisher ist es uns gelungen das Wortproblem bei regulären Sprachen mithilfe eines Automaten zu entscheiden, jedoch wird dies bei kontextfreien Sprachen nicht so einfach. Hierzu muss man sich klar machen, dass bei der Arbeit mit kontextfreien Sprachen die Situation eintreten kann, dass zwei unterschiedliche Grammatiken die gleiche Sprache beschreiben, da diese unterschiedlich komplex aufgebaut sein können ähnlich wie bei aussagenlogischen Formeln, deren Semantik identisch ist, die Formeln aber sehr unterschiedlich sind. Auf Basis dessen hat man sich für eine Normalform entschieden um Algorithmen zu entwickeln, die dank dynamischer Programmierung und einer einheitlichen Normalform recht effizient arbeiten können.

Definition (Chomsky-Normalform)**.** *Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ ist in Chomsky-Normalform (CNF), wenn alle Produktionen die Form*

$$A \to \alpha \in \Sigma \qquad \textit{oder} \qquad A \to BC \in V \times V$$

haben.

Synthesealgorithmus für die Chomsky-Normalform

Es ist möglich mit wenigen Schritten eine kontextfreie Grammatik G in CNF zu überführen, auch wenn dies teils unübersichtlich werden kann.

1. Elimiere ε-Produktionen durch Auflösen
2. Elimiere Kettenproduktionen der Form $A \to B \to C$
3. Ersetze bestimmte Terminale
4. Packe sämtliche Ketten der Länge ≥ 3
5. Füge die Produktion $S \to \varepsilon$ zur Grammatik G hinzu, falls das leere Wort vorher bereits in der Sprache war, d.h. $\varepsilon \in L(G)$

Beispiel 1. Wir wollen in diesem Beispiel die folgende Grammatik in CNF überführen:

$$
\begin{aligned}
&S \to 0A0 \mid 1B1 \mid BB\\
&A \to D\\
&B \to S \mid A\\
&C \to S \mid \varepsilon\\
&D \to C
\end{aligned}
$$

Um dies übersichtlicher zu gestalten werden wir Schritt für Schritt extrem detailliert notieren und Zwischenergebnisse festhalten.

Schritt 1 Zu Beginn werden wir alle vorhandenen ε-Produktionen direkt auswerten und diese somit überflüssig machen. Da $C \to \varepsilon$ und $D \to C$ gelten, werden wir $D \to \varepsilon$ hinzufügen. Nun ist auch $A \to D$ vorhanden weshalb wir $A \to \varepsilon$ ebenfalls hinzufügen müssen. Durch $B \to A$ ist $B \to \varepsilon$ analog hinzuzufügen. Nun betrachten wir die Produktion $S \to 0A0$ und werden $S \to 00$ hinzufügen, ebenfalls $S \to 11$. Mit $S \to BB$ können wir jeweils eines oder sogar beide B ersetzen und erhalten $S \to B$ und $S \to \varepsilon$. Abschließend entfernen wir sämtliche ε-Produktionen.

$$
\begin{aligned}
&S \to 0A0 \mid 1B1 \mid BB \mid 00 \mid 11 \mid B\\
&A \to D\\
&B \to S \mid A\\
&C \to S\\
&D \to C
\end{aligned}
$$

Schritt 2 Im nächsten Schritt werden wir Kettenproduktionen der Form $A \to B \to C$ auflösen, indem wir alle Vorkommen von B direkt mit C ersetzen. Da in unserem Beispiel die Variable D ausschließlich auf C zeigt können wir D entfernen und aus der Produktion $A \to D$ die Produktion $A \to C$ machen. Weiterhin können wir C elimieren und alle Vorkommen mit S ersetzen. Zuletzt haben wir die Produktion $A \to S$ erzeugt, die wir ebenso elimieren können und alle Vorkommen von A mit S ersetzen. Zusätzlich erhalten wir die Produktion $B \to S \mid S$, die äquivalent ist zu $B \to S$ und können ebenso B durch S ersetzen und erhalten abschließend $S \to S$.

$$S \to 0S0 \mid 1S1 \mid SS \mid 00 \mid 11$$

Schritt 3 Im aktuellen Schritt werden wir bestimmte Terminale, also Symbole, die keine Variablen sind, durch Variablen ersetzen, die direkt darauf zeigen. D.h. für 0 und 1 werden wir $N \to 0$ und $E \to 1$ hinzufügen.

$$
\begin{aligned}
&S \to NSN \mid ESE \mid SS \mid NN \mid 11\\
&N \to 0\\
&E \to 1
\end{aligned}
$$

Schritt 4 Zuletzt werden wir alle Produktionen verkürzen, die mehr als 2 Variablen beinhalten und beispielweise Produktionen $A \to BCD$ zu $A \to BE$ mit $E \to CD$ verkürzen. Dies ist insbesondere nicht eindeutig und kann beliebig gemacht werden.

$$\begin{aligned} S &\to NT_1 \mid ET_2 \mid SS \mid NN \mid EE \\ T_1 &\to SN, \qquad T_2 \to SE \\ N &\to 0,\ E \to 1 \end{aligned}$$

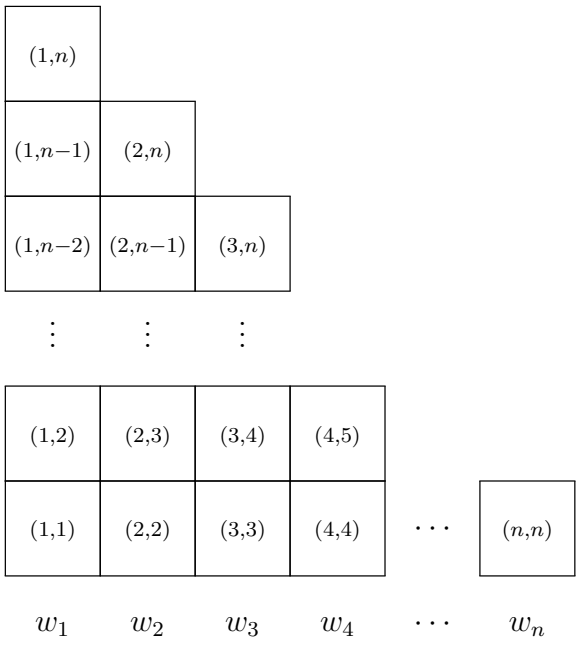
Wortproblem für kontextfreie Sprachen

In diesem Abschnitt werden wir uns mit dem CYK-Algorithmus auseinandersetzen um zu überprüfen, ob ein Wort von einer kontextfreien Grammatik ableitbar ist. Dieser Algorithmus setzt voraus, dass die betrachtete Grammatik sich in Chomsky-Normalform befindet, was uns das Arbeiten mit treppenförmigen Tabellen ermöglicht. Mithilfe dynamischer Programmierung bestimmt der Algorithmus, mit welchen Variablen kleinere Teilwörter hergeleitet werden können und diese setzt er dann zu größeren Wörtern zusammen um schlussendlich zu prüfen, ob das gesamte Wort hergeleitet werden kann.

CYK-Algorithmus

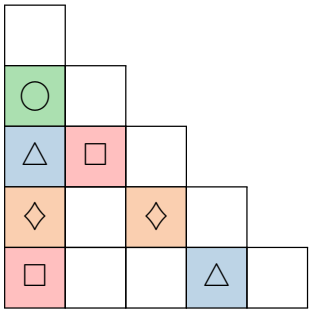
Sind eine CFG $G = (V, \Sigma, P, S)$ in CNF und ein Wort $w = w_1 \dots w_n \in \Sigma^*$ gegeben, so entscheidet der Algorithmus in $\mathcal{O}(n^3)$ Laufzeit, ob $w \in L(G)$ gilt.

Schritt 1 Konstruiere eine Tabelle mit $|w| = n$ Stufen:



Schritt 2 Zur Initialisierung wird die unterste Zeile der Tabelle betrachtet. Trage in die Felder (i, i) alle Variablen V ein, für die $V \to w_i$ gilt.

Schritt 3 Nun wird durch die Tabelle zeilenweise von unten nach oben iteriert um die Elemente der Zellen zu bestimmen. Dabei stehen in (i, j) Variablen V, von denen aus das Teilwort $w_i \dots w_j$ ableitbar ist, also $V \to_G^* w_i \dots w_j$ gilt. So setzt sich der Inhalt von ◯ aus den Kombinationen von Variablen aus □, ♢ und △ zusammen. Sollen beispielweise die Variablen A, B, C mit den Variablen D, E kombiniert werden, so suchen wir Variablen V für die $V \to AD$, $V \to AE$, $V \to BD$, $V \to BE$, $V \to CD$ oder $V \to CE$ gelten und tragen diese an entsprechender Stelle ein.

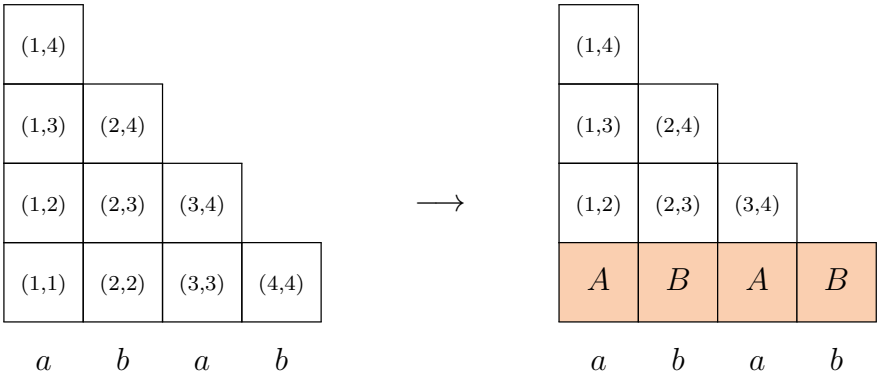


Schritt 4 Sofern die Tabelle komplett durchlaufen wurde ist der Inhalt der Zelle $(1, n)$ von Interesse. Falls das Startsymbol S darin enthalten ist, gilt $S \to_G^* w_1 \dots w_n = w$ und $w \in L(G)$.

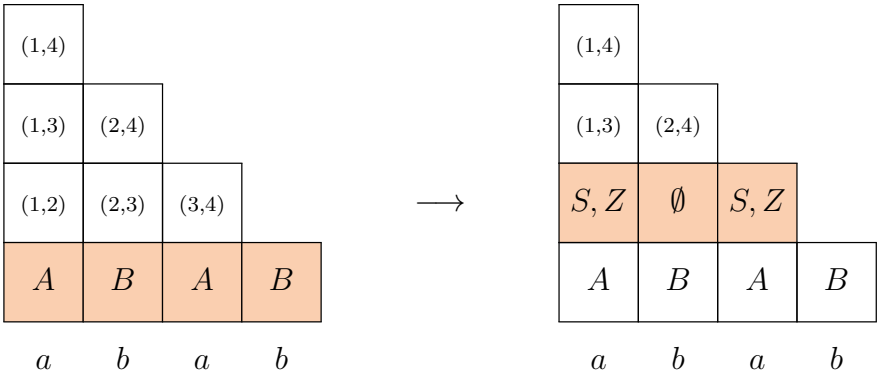
Beispiel 2. Wir betrachten die kontextfreie Grammatik $G = (V, \{a, b\}, P, S)$ mit Produktionen in Chomsky-Normalform

$$
\begin{aligned}
S &\to YC_3 \mid YX \mid AC_3 \mid AX \mid YY \mid YB \mid YA \mid AY \mid AB \mid AA,\\
X &\to ZC_2 \mid ZB \mid BZ \mid BB,\\
Y &\to AC_1 \mid AY \mid AA,\\
C_1 &\to YZ \mid AZ,\\
Z &\to YC_3 \mid YX \mid AC_3 \mid AX \mid YY \mid YB \mid YA \mid AY \mid AB \mid AA,\\
C_3 &\to XZ \mid YZ \mid BZ \mid AZ,\\
A &\to a,\\
B &\to b.
\end{aligned}
$$

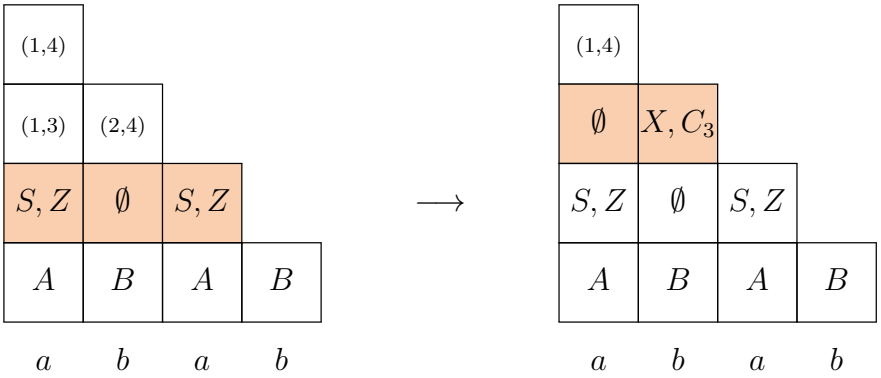
Wir entscheiden nun mit dem CYK-Algorithmus, ob das Wort *abab* in der Sprache $L(G)$ enthalten ist. Hierzu stellen wir eine Tabelle mit 4 Stufen auf und initialisieren diese sogleich indem wir den ersten Schritt der Algorithmus durchführen:



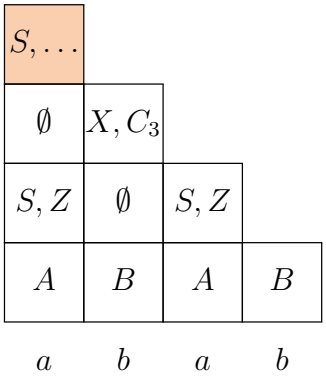
Um den Inhalt von $(1,2)$ zu bestimmen nutzen wir das Schema aus Schritt 3 und kombinieren die Variablen aus $(1,1)$ sowie $(2,2)$ und bestimmen die Variablen V für die $V \to AB$ gilt. Dies sind S und Z und wir tragen diese eine. Für $(2,3)$ kombinieren wir $(2,2)$ und $(3,3)$ und suchen Variablen V mit $V \to BA$, jedoch gibt es in G keine solche Produktion und wir tragen $\emptyset$ ein. Das Feld $(3,4)$ ist analog zu $(1,2)$ und wir erhalten folgendes Zwischenergebnis:



Nun folgt die dritte Zeile - das Feld $(1,3)$ setzt sich aus $(1,1)$ mit $(2,3)$ zusammen sowie $(1,2)$ und $(3,3)$. D.h. wir suchen Variablen V für die $V \to A\emptyset$, $V \to SB$ oder $V \to ZB$ gilt, jedoch gibt es keine solche Variable und wir tragen $\emptyset$ ein. Für $(2,4)$ betrachten wir Variablen V mit $V \to BS$, $V \to BZ$ oder $V \to \emptyset B$ und erhalten die Variablen X und C_3 mithin folgendes Zwischenergebnis:



Abschließend gilt für $(1,4)$ $S \to AX$ und wir an dieser Stelle terminieren können und somit $S \to_G^* abab$ gilt, was durch folgende finale Tabelle ablesbar ist:



Pumping-Lemma für kontextfreie Sprachen

Analog zum Fall für reguläre Sprachen ist es möglich das Pumping-Lemma zu nutzen um zu beweisen, dass eine Sprache nicht kontextfrei ist. Der wesentliche Unterschied ist jedoch, dass man ein Wort in fünf anstelle von drei Teilen zerlegt und zwei davon gleichzeitig aufpumpt.

Definition (Pumping-Lemma)**.** *Sei L eine kontextfreie Sprache, dann existiert eine dazugehörige* Pumping-Lemma-Zahl $n \in \mathbb{N}_0$ *für alle ausreichend langen Wörter $z \in L$ mit $|z| \geq n$, die sich in Teilwörter u, v, w, x, y zerlegen lassen mit $z = uvwxy, vx \neq \varepsilon$ und $|vwx| \leq n$, so dass für alle $i \in \mathbb{N}_0$ das Wort z aufpumpbar ist und $uv^iwx^iy \in L$ gilt.*

Beispiel 3. Wir betrachten wir $L = \{a^nb^mc^k \mid 0 < n < m < k\}$ und wollen beweisen, dass diese Sprache nicht kontextfrei ist. Hierzu nehmen wir an, dass L kontextfrei sei und führen dies zum Widerspruch. Weiterhin sei $n \in \mathbb{N}_0$ eine beliebige, aber feste, PLZ für L. Dann existiert ein nicht aufpumpbares $z = a^nb^{n+1}c^{n+2} \in L$ mit $|z| \geq n$. Nun ist zu zeigen, dass sich **keine** Zerlegung aufpumpen lässt. Schematisch sieht das betrachtete Wort wie folgt aus:

$$\underbrace{a\ldots\ldots}_{u}\underbrace{\ldots ab\ldots bc\ldots\ldots}_{1\leq|vwx|\leq n}\underbrace{\ldots c}_{y}$$

Die zugehörigen Zerlegungen finden wir, indem wir nun gedanklich die mittlere Klammer strecken und stauchen und notieren, welche Buchstabenkombinationen darin vorkommen. Man beachte hierzu, dass die Teilwörter u und y keine Längenbeschränkungen haben, mithin das Teilwort vwx ziemlich flexibel sein kann. An dieser Stelle unterscheiden wir insgesamt fünf Fälle:

- Zerlegung 1: Das Teilwort vx besteht lediglich aus a's. Mit $i \geq 2$ und dem Wort uv^iwx^iy erkennen wir, dass durch das Aufpumpen mindestens ein a hinzugefügt wird (da $vx \neq \varepsilon$ gilt) und wir mindestens so viele a's wie b's haben. Somit gilt aber nach Konstruktionsvorschrift $uv^iwx^iy \notin L$. Dies steht im Widerspruch zur Annahme, dass L kontextfrei sei!

- Zerlegung 2: Das Teilwort vx besteht lediglich aus a's und b's. Mit $i \geq 2$ und dem Wort uv^iwx^iy erkennen wir, dass durch das Aufpumpen mindestens ein b hinzugefügt wird (da $vx \neq \varepsilon$ gilt) und wir mindestens so viele b's wie c's haben. Somit gilt aber nach Konstruktionsvorschrift $uv^iwx^iy \notin L$. Dies steht im Widerspruch zur Annahme, dass L kontextfrei sei!

- Zerlegung 3: Das Teilwort vx besteht lediglich aus b's. Analog zu Zerlegung 2.

- Zerlegung 4: Das Teilwort vx besteht lediglich aus b's und c's. Mit $i = 0$ und dem Wort uv^iwx^iy erkennen wir, dass durch das Aufpumpen mindestens ein b entfernt wird (da $vx \neq \varepsilon$ gilt) und wir mindestens so viele a's wie b's haben. Somit gilt aber nach Konstruktionsvorschrift $uv^iwx^iy \notin L$. Dies steht im Widerspruch zur Annahme, dass L kontextfrei sei!

- Zerlegung 5: Das Teilwort vx besteht lediglich aus c's. Mit $i = 0$ und dem Wort uv^iwx^iy erkennen wir, dass durch das Aufpumpen mindestens ein c entfernt wird (da $vx \neq \varepsilon$ gilt) und wir mindestens so viele b's wie c's haben. Somit gilt aber nach Konstruktionsvorschrift $uv^iwx^iy \notin L$. Dies steht im Widerspruch zur Annahme, dass L kontextfrei sei!

Somit haben wir für jede mögliche Zerlegung gezeigt, dass diese nicht aufpumpbar ist und L im Widerspruch zur Annahme nicht kontextfrei sein kann.

Kanonische Minimalautomaten

Wir haben bisher gesehen, wie zu einem gegebenen Automaten ein minimaler Quotientenautomat mit der *Moore-Konstruktion* hergeleitet werden kann. Sofern man jedoch einen minimalen Automaten zu einer gegebenen Sprache direkt konstruieren soll, muss man geschickter an die Sache rangehen. Hierzu nutzen wir erneut das Konzept der Äquivalenz um sämtliche Wörter aus Σ^* in Klassen einzuteilen, die wir nutzen um einen Automaten zu konstruieren. Zunächst schauen wir uns an, wie genau wir Wörter klassifizieren können.

Definition (Äquivalenz von Wörtern)**.** *Zu einer gegebenen Sprache L bezeichnen wir zwei Wörter $u, v \in \Sigma^*$ als äquivalent, dann und nur dann wenn*

$$u \equiv_L v \Leftrightarrow (\forall w \in \Sigma^* : uw \in L \Leftrightarrow vw \in L)$$

gilt.

Beispiel 4. Wir betrachten die Sprache $R = L(a^*b^*)$ und werden zeigen, dass $ab \not\equiv_R aa$ gilt. Wählen wir $w = a$, so gilt $abw = aba \notin R$ und $aaw = aaa \in R$, mithin sind die beiden Wörter nicht äquivalent und mit dem Wort w unterscheidbar. Analog gilt $aba \not\equiv_R \varepsilon$, da die beiden Wörter mit $w = \varepsilon$ unterscheidbar sind.

Beispiel 5. Nun wird die Sprache $\{w \in \Sigma^* \mid w \text{ endet mit } 00\}$ über dem binären Alphabet betrachtet. Ob ein Wort vom zugehörigen DFA akzeptiert wird, hängt also ausschließlich von den letzten beiden Zeichen ab, die eingelesen wurden. Somit können wir Zustände modellieren, die folgende Informationen während dem Einlesen repräsentieren:

- das Wort endet aktuell nicht mit 0
- das Wort endet aktuell mit 0, jedoch nicht mit 00
- das Wort endet aktuell mit 00

Jedes beliebige Wort $w \in \Sigma^*$ kann genau einer der drei Klassen zugeordnet werden. Hierbei handelt es sich um eine Partitionierung von Σ^* in diese drei disjunkten Klassen, die formal wie folgt aussehen:

$$\begin{aligned}
[\varepsilon]_\equiv &= \{w \mid w \text{ endet aktuell nicht mit } 0\} \\
[0]_\equiv &= \{w \mid w \text{ endet aktuell mit } 0, \text{ jedoch nicht mit } 00\} \\
[00]_\equiv &= \{w \mid w \text{ endet aktuell mit } 00\}
\end{aligned}$$

Die *Repräsentanten* $\varepsilon, 0$ und 00 der jeweiligen Klassen stehen jeweils für Wörter, die keine, genau eine bzw. mindestens zwei 0en am Ende besitzen. Auch wenn ein Automat *gedächtnislos* ist, kann dieser Umstand durch eine solch geschickte Modellierung umgangen werden, in der jede Klasse einen Zustand repräsentiert. Der zugehörige *kanonische Minimalautomat* lässt sich daraufhin aus diesen Klassen ableiten.

Definition (Kanonischer Minimalautomat)**.** *Sei $L \subset \Sigma^*$ eine reguläre Sprache über dem Alphabet Σ, dann bezeichnet $M_L = (\Sigma^* / \equiv, \Sigma, \delta_L, [\varepsilon]_\equiv, F_L)$ den* kanonischen Minimalautomaten *mit der Zustandsmenge $\Sigma^* / \equiv$ aller Äquivalenzklassen, sowie der Zustandsübergangsfunktion $\delta_L([w]_\equiv, a) = [wa]_\equiv$. Weiterhin gilt $F_L = \{[w]_\equiv \mid w \in L\}$.*

Der Automat aus Beispiel 5 kann nun mit der Definition hergeleitet werden:

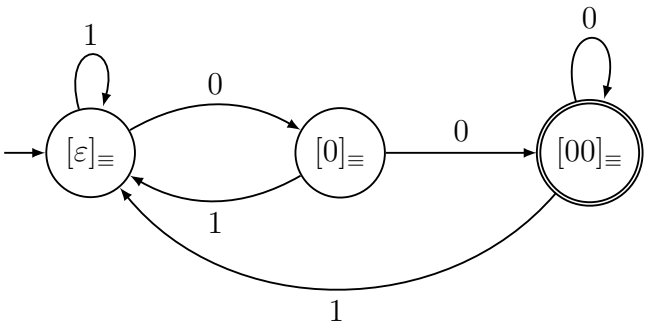


Abbildung 1: Kanonischer Minimalautomat zu Beispiel 5

Beispiel 6. Nun soll ein minimaler Automat M_G zur Sprache $G = \{a^m b^n \mid m, n \text{ gerade}\}$ hergeleitet werden. Erneut hängt es von den letzten zwei Zeichen ab, ob ein Wort vom Automaten akzeptiert werden soll. Die gesuchten Äquivalenzklassen müssen beschreiben, dass

- aktuell nichts gelesen wurde (das Gedächtnis des Automaten zurückgesetzt ist), jedoch auf ein (evtl. weiteres) a oder b gewartet wird,
- aktuell genau ein a als letztes gelesen wurde und auf mindestens ein weiteres a gewartet wird,
- aktuell genau ein b als letztes gelesen wurde und auf mindestens ein weiteres b gewartet wird,
- aktuell genau zwei b als letztes gelesen wurden (ohne das Gedächtnis zurücksetzen zu dürfen, da offensichtlich keine a mehr gelesen werden dürfen) und in diesem Zustand ein Wort akzeptiert werden könnte,
- aktuell genau ein a und genau ein b bzw. genau ein b und genau ein a als letztes gelesen wurden, was nicht akzeptiert werden darf und als Fangzustand dient.

Formal sind diese Klassen wie folgt gegeben

$$
\begin{aligned}
[\varepsilon]_{\equiv} &= \{a^m \mid m \text{ gerade}\} \\
[a]_{\equiv} &= \{a^m \mid m \text{ ungerade}\} \\
[b]_{\equiv} &= \{a^m b^n \mid m \text{ gerade}, n \text{ ungerade}\} \\
[bb]_{\equiv} &= \{a^m b^n \mid m, n \text{ gerade}, n \geq 2\} \\
[ab]_{\equiv} &\text{ besteht aus dem Rest}
\end{aligned}
$$

und der zugehörige Automat hier zu sehen.

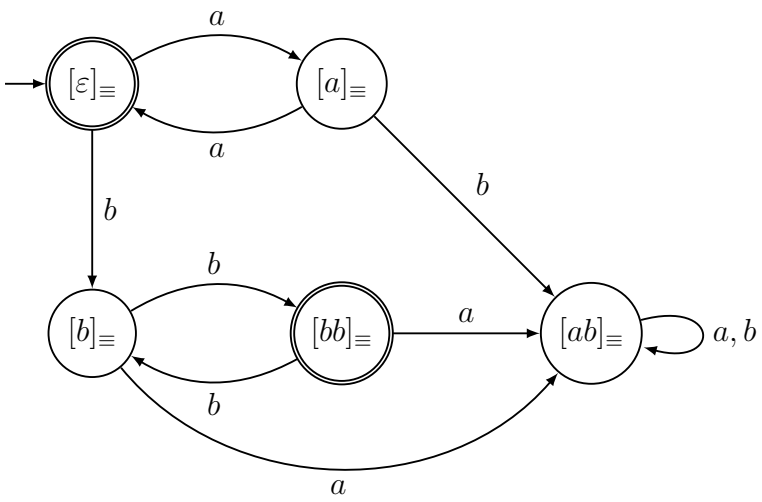


Abbildung 2: Kanonischer Minimalautomat zu Beispiel 6

Satz (Myhill-Nerode)**.** *Sei $L \subset \Sigma^*$ eine Sprache über dem Alphabet Σ, dann ist L regulär, dann und nur dann, wenn die Anzahl an Äquivalenzklassen von $\equiv_L$ endlich ist. Weiterhin gibt es einen eindeutigen minimalen DFA A mit $L(A) = L$.*

Beweis. Trivial. □