Script zur Lehrveranstaltung

Automatentheorie und formale Sprachen

Prof. Annika Wagner

Florian Schleich

May 1, 2013

Hochschule Fulda
University of Applied Sciences

Contents

1 Grammatiken

In diesem Kapitel lernen wir, wie man Grammatiken definiert und Wörter aus diesen Grammatiken erzeugt.

1.1 Symbole

Symbole sind alle verwendbaren Zeichen um durch Kombination Wörter zu erschaffen.

1.2 Alphabet

Ein Alphabet, abgekürzt Σ (Sigma), ist eine endliche Menge von einzelnen Symbolen. Ein Alphabet wird als endliche Menge von Symbolen definiert.

$\Sigma = \{☺, a, Anton\}$

Neben einzelnen Buchstaben können auch Wörter im Sinne der deutschen Sprache verwendet werden. Es können aber auch andere Symbole (☺) verwendet werden. Bei Verwendung von Wörtern als Symbole eines Alphabets ist aber zu beachten, dass bei der späteren Verwendung die einzelnen Buchstaben oder Teile des Wortes nicht als Symbole genommen werden dürfen, sondern das gesamte Wort als ein Symbol bezeichnet wird. Im hier verwendeten Fall darf also das große *A* nicht als Symbol verwendet werden darf. ebenso dürfen auch keine Wortteile wie *ton* nicht verwendet werden. Ein zulässiges Symbol ist allerdings das gesamte Wort *Anton*.

1.3 Wörter

Wörter werden aus Kombination von einzelnen Symbolen gebildet. Die bildbaren Wörter werden in einer Menge Σ^* (Sigma mit Kleene-Stern) gesammelt. Wird die Menge aller Wörter gebildet muss jetzt explizit das leere Wort ϵ (Epsilon) angegeben werden. Auf Grund der in der Grammatik verwendeten Regeln wird die Menge der möglichen Wörter eingeschränkt. In unserem Beispiel kann die Menge aller Wörter so aussehen:

$\Sigma^* = \{\epsilon, ☺, a, Anton, ☺a, aAnton, ☺Anton, a☺, Antona, Anton☺, ☺☺, ...\}$

Die Menge wird gebildet, in dem erst das leere Wort, dann alle einstelligen Wörter, alle zweistelligen Wörter, alle dreistelligen Wörter usw. in die Menge integriert werden. Die Menge der Wörter ist unendlich, da die Länge der Wörter nicht bestimmt ist.

Die Länge eines Wortes $|w|$ bezeichnet die Anzahl der für das Wort verwendeten Symbole.

$|\epsilon| = 0$
$|a| = 1$
$|Anton| = 1$
$|aa| = 2$

Figure 1.1: Beispiele für Wortlängen

1.4 Grammatiken

Eine Grammatik G wird definiert durch:

- eine Menge von Variablen $V = \{S, U\}$, die bei der Ableitung verschwinden,
- eine Menge von Terminalsymbolen Σ, gelichbedeutend mit dem definierten Alphabet
- ein Startsymbol $S \in V$

- und eine Menge von Regeln P

Formal wird eine Grammatik also beschrieben als:

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

1.4.1 Regeln

Eine Regel besteht immer aus einer linken und einer rechten Regelseite (Tupel). Bei der Ableitung von Regeln wird der Ausdruck auf der linken Seite durch den Ausdruck auf der rechten Seite ersetzt.

Bei der Regel $S \rightarrow Anton$ wird die Variable S durch das Symbol $Anton$ ersetzt.

1.4.2 Ableitung

Bei der Ableitung von Regeln beginnt man beim Startsymbol (hier S) und ersetzt das Startsymbol durch den entsprechenden Ausdruck einer Regel, die auf das Startsymbol passt. Zwischen zwei Ableitungsschritten kommt dann ein Doppelpfeil ($\Rightarrow$). Eine kürzere Schreibweise erlaubt der Doppelpfeil mit Kleene-Stern ($\Rightarrow^*$), bei dem nicht jede Regel explizit aufgeschrieben werden muss. Auch 0-Schritte sind bei dieser Ableitungsart zulässig (siehe auch Abschnitt 1.7.1, Seite 12).

$S \rightarrow Anton$ (Regel der Grammatik)
$S \Rightarrow Anton$ (Ableitung, direkte Anwendung der Regel ohne Zwischenschritte)
$S \Rightarrow^* Anton$ (Ableitung, mehrmaliges Anwenden von Regeln mit Zwischenschritten)

Figure 1.2: Einfaches Beispiel für eine Ableitung

1.4.3 Regelalternativen

Regeln können auch Alternativen enthalten, um mehr Möglichkeiten zu bieten Wörter zu erzeugen.

Alternativen werden auf der rechten Regelseite durch ein Pipe-Symbol (|) eingeleitet. Eine Regel mit Alternativen könnte folgendermaßen aussehen:

$S \rightarrow Anton|Anton☺|Antona$

Dadurch wären die Wörter *Anton*, *Anton*☺ und *Antona* für die Variable S möglich.

1.4.4 Rekursion in Regeln

$S \rightarrow AntonG$
$G \rightarrow ☺|☺G$

Figure 1.3: Grammatik mit Rekursion

Rekursionen in Grammatiken können Wörter erzeugen, deren Länge (Betrag) unendlich lang sind. Eine mögliche Ableitung für die obige Grammatik könnte folgendermaßen lauten.

$S \Rightarrow AntonG$
$\Rightarrow Anton☺G$
$\Rightarrow Anton☺☺$

Figure 1.4: Ableitung für eine rekursive Grammatik

Wir sehen also, dass durch die Rekursion auf G die Wörter unendlich lang werden können und durch das Symbol ☺ aufgefüllt werden. Des weitern stellen wir fest, dass Ableitungen erst enden, wenn alle Variablen entfernt, bzw. ersetzt wurden.

1.4.5 Mengen von Regeln

Als weitere Komponente für die Definition von Grammatiken wird die Menge aller Regeln der Grammatik benötigt. Im vorherigen Abschnitt haben wir gelernt, wie Regeln definiert werden. Die Menge der Regeln einer Grammatik ist eine Teilmenge des kartesische Produktes der Elemente aus V und Σ. Das kartesische Produkt wird durch die Kombination aller Elemente einer Menge (alles vor dem $\times$) mit sämtlichen Elementen einer anderen Menge (alles nach dem $\times$) gebildet.

$A = \{0, 1\}$
$B = \{x\}$
$A \times B = \{(0, x), (1, x)\}$

Figure 1.5: Kartesisches Produkt der Mengen A und B

Die Menge aller Regeln einer Grammatik wird nun durch $(V \cup \Sigma)^* \times (V \cup \Sigma)^*$ gebildet. Durch diese Definition lassen sich sämtliche Kombinationen aus Terminalsymbolen und Variablen auf beden Seiten der Regeln ausfstellen. Es ist aber meistens gewünscht nur ein paar bestimmte Regeln zuzulassen. Man bildet also eine bestimmte Teilmenge der eben gebildeten Gesamtmenge.

HINWEIS **Unterschied zwischen {} und ()**
Bei Verwendung von {} ist die Reihenfolge und Anordnung der Elemente egal, da es sich um eine Menge handelt. Bei Verwendung von () ist die Anordnung der Elemente **nicht** egal.

1.4.6 Überprüfung der Regeln

Um sicherzustellen, dass eine Grammatik eine bestimmte Sprache erzeugt, müssen Überprüfungen bezüglich der Vollständigkeit und Korrektheit dieser Grammatik erfolgen (siehe auch Abschnitt 1.6, Seite 9).

Vollständigkeit Eine Grammatik ist vollständig, wenn sie (alle) Wörter der Sprache erzeugt.

Korrektheit Eine Grammatik ist korrekt, wenn sie keine Wörter ausserhalb der Sprache erzeugt.

1.5 Kontextfrei oder regulär

Kontextfreie Grammatiken haben auf der linken Regelseite genau **eine** Variable und eine Beliebige Kombination aus Terminalsymbolen und Variablen auf der rechten

Regelseite. Grammatiken sind regulär, wenn die verwendeten Regeln auf der linken Regelseite **eine** Variable und auf der rechten Seite entweder das leere Wort (ϵ), genau ein Terminalsymbol oder genau ein Terminalsymbol und eine Variable besitzen.

Allgemein: $P \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$
Kontextfrei: $P \subseteq (V) \times (V \cup \Sigma)^*$
Regulär: $P \subseteq (V) \times (Y \cup \Sigma \cup \{\epsilon\})$ wobei $Y = aV$

HINWEIS **Links- und Rechtsregulär**

In dieser LVA wurde der Begriff "regulär" von Frau Wagner als "rechtsregulär" (immer zuerst rechteste Variable ersetzen) definiert!

1.6 Alternative Schreibweise für Grammatiken

Um Grammatiken und deren Regeln einfacher kommunizieren zu können kann man die erzeugten Wörter auch als "Formel" aufschreiben. Dazu folgende Aufgabe.

Aufgabe: Sei $\Sigma = \{0, 1\}$. Leiten Sie mit der folgenden Grammatik das Wort 0101001010 ab.

$S \rightarrow 0T0$
$T \rightarrow 1S1|\epsilon$

Ableitung:

$S \Rightarrow 0T0$
$\Rightarrow \underline{0}1S1\underline{0}$
$\Rightarrow \underline{01}0T0\underline{10}$
$\Rightarrow \underline{0101}S1\underline{1010}$
$\Rightarrow \underline{01010}T0\underline{1010}$
$\Rightarrow \underline{0101001010}$

Figure 1.6: Zugehörige Ableitung

Aus dieser Grammatik lassen sich folgende mögliche Wörter ableiten:

- 00

- 010010
- 0101001010
- ...

Da die Grammatik rekursiv ist und damit die Wortliste nicht endlich ist, kann man unmöglich alle Wörter auflisten. Um alle Wörter die erzeugt werden können trotzdem aufschrieben zu können, kann die Menge der Wörter auch formelähnlich aufgeschrieben werden.

$\{(01)^n 00 (10)^n | n \geq 0\}$

Der erste Teil (01) ist genau so oft enthalten wie der letzte Teil (10) kann aber auch weggelassen werden. der mittlere Teil (00) ist immer vorhanden und kann auch nicht wiederholt werden. Durch diese vereinfachte Schreibweise ist es sehr leicht möglich weitere Wörter zu erzeugen. Man muss nur einen Wert für n einsetzen und wiederholt die entsprechenden Teile n-mal.

Aufgabe: Sei $\Sigma = \{0, 1\}$. Entwicklen Sie eine Grammatik, die folgende Sprache erzeugt: $L = \{w \in \Sigma^* | \, |w| \ ist\ gerade\}$

Ansatz: Das das zu erzeugende Wort (w) ein Σ^* enthalten sein soll, kann es also aus allen möglichen Kombinationen aus Σ (0 und 1) bestehen. Als zusätzliche Beschränkung für die Menge aller Wörter soll der Betrag, also die Länge, des Wortes gerade (durch Zwei teilbar) sein.

Aus dem obigen Ansatz ergeben sich folgende Regeln für die Grammatik.

$S \to PS \mid P$	(beliebig viele Pärchen aneinanderreihen)
$P \to ZZ \mid \epsilon$	(Pärchen von Zahlen aus Ziffern bilden)
$Z \to 0 \mid 1$	(Ziffern)

$S \Rightarrow PS$
$\Rightarrow \underline{P}P$
$\Rightarrow ZZ\underline{P}$
$\Rightarrow \underline{ZZ}ZZ$
$\Rightarrow 0\underline{ZZ}$
$\Rightarrow \underline{0}1\underline{Z}Z$
$\Rightarrow \underline{010}\underline{Z}$
$\Rightarrow \underline{0100}$

Figure 1.7: Mögliche Ableitung für das Wort 0100

1.7 Ableitungsbaum

Aus dieser Ableitung lässt sich ein Ableitungsbaum erstellen. Dieser stellt strukturiert dar, wie ein Wort abgeleitet wird.

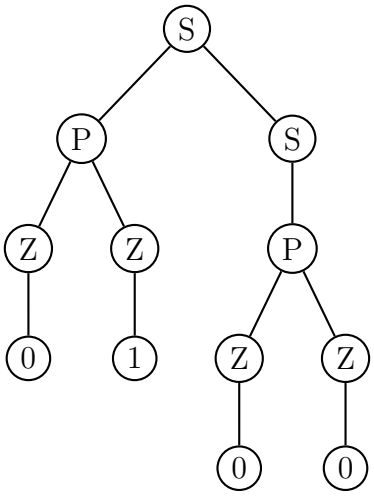


Figure 1.8: Ableitungsbaum für die obige Ableitung

Egal in welcher Reihenfolge abgeleitet wird, bleibt der Ableitungsbaum immer der selbe. Die Ableitung nach $S \Rightarrow PS \Rightarrow ZZS \Rightarrow ...$ oder nach $S \Rightarrow PS \Rightarrow PP \Rightarrow ...$ erzeugen den selben Ableitungsbaum. Anders sieht es aus, wenn eine andere Regelalternative zur Ableitung genutzt wird. Dann kann sich der Ableitungsbaum durchaus unterscheiden.

Die Ableitung $S \Rightarrow PS \Rightarrow PPS \Rightarrow PPP \Rightarrow ZZPP \Rightarrow ZZZZP \Rightarrow ZZZZ\epsilon \Rightarrow ...$ erzeugt den folgenden Ableitungsbaum.

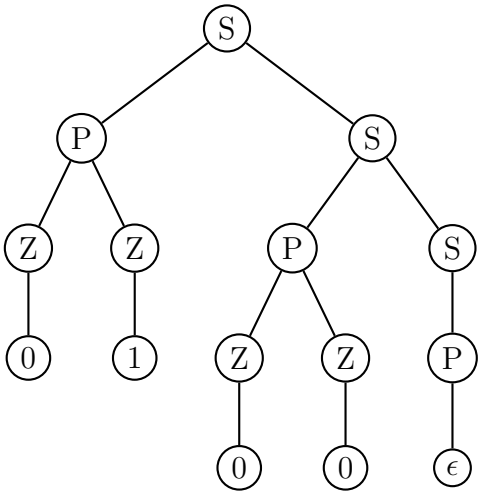


Figure 1.9: Ableitungsbaum für eine alternative Ableitung

HINWEIS **Mehrdeutigkeit**

Da es für die hier verwendete Grammatik strukturell unterschiedliche Ableitungsbäume gibt, ist die Grammatik mehrdeutig. Daher ist es nicht möglich einen Parser für die erzeugte Sprache zu generieren.

Durch Linksableitung kann Mehrdeutigkeit von Bäumen (Ableitungen) eindeutig bewiesen werden. Die Linksableitung ist eine Konvention, dass von Ableitungsschritt zu Ableitungsschritt die linkeste Variable zuerst abgeleitet wird.

1.7.1 Reflexiver & transitiver Abschluss von $\Rightarrow$

Der Ableitungsoperator $\Rightarrow$ erlaubt es, nur einen Ableitungsschritt darzustellen. Der Ableitungsoperator $\Rightarrow^*$ hingegen erlaubt es beliebg viele Ableitungsschritte zusammenzufassen. Wird eine Grammatik mit dem einfachen Operator so abgeleitet $S \Rightarrow bB \Rightarrow bb$, kann die Ableitung mit dem reflexiven und trasitiven Abschluss so aussehen: $S \Rightarrow^* bb$. Dies spart Zeit und schreibarbeit, ist aber nicht so übersichtlich, da zwischeninformationen über die Ableitungen verloren gehen.

1.8 Greibach Normalform (GNF)

Die Greibach Normalform erzeugt bei jedem ableitungsschritt genau ein Terminalsymbol. Bei der GNF besteht die rechte Regelseite aus genau einem Terminalsymbol und beliebig vielen Variablen ($P \subseteq V \times \Sigma V^*$).

Im folgenden werden Regeln besprochen, die aus einer kontextfreien Grammatik eine Grammatik in Greibach Normalform erzeugen.

1.8.1 Problem 1: Mehrere Terminalsymbole

Wenn eine Regel mehrere Terminalsymbole auf der rechten Regelseite besitzt, verstößt sie gegen die GNF. Um das Problem zu lösen, erzeugt man einfach eine neue Variable, die das Terminalsymbol ersetzt.

$S \to a\underline{a}$	wird zu	$S \to a\underline{A}$
		$\underline{S \to a}$

1.8.2 Problem 2: Mehrere Variablen, aber kein Terminalsymbol

Ein weiteres Problem entsteht, wenn zwar mehrere Variablen auf der rechten Regelseite stehen, aber kein Terminalsymbol vorhanden ist. Dazu wird die vorderste Variable durch die rechte Regelseite der Variable ersetzt.

$S \to \underline{B}S\|c$	wird zu	$S \to \underline{b}S\|c$
$B \to b$		$B \to b$

1.8.3 Problem 3: Epsilon

Bei der GNF ist die Verwendung von ϵ generell untersagt. Überall wo ϵ einsetzbar ist, muss eine neue rechte Seite ohne ϵ entstehen (Variable die zu ϵ führt muss also

gestrichen werden).

$S \to aA$	wird zu	$S \to \underline{a}\|aA$
$A \to bA\|\underline{\epsilon}$		$A \to bA\|\underline{b}$

Dabei ist zu beachten, dass das kürzeste Wort **pro Regel** erhalten bleiben muss (für A wurde jeweils ϵ eingesetzt).

$S \Rightarrow aA \Rightarrow \underline{a}$
$A \Rightarrow bA \Rightarrow \underline{b}$

2 Endliche Automaten

Endliche Automaten werden dazu genutzt um zu prüfen ob ein Wort auf eine bestimmte Grammatik passt. Diese Eigenschaft nutzt ein Parser um den Quellcode auf seine syntaktische Richtigkeit zu prüfen. Um aus einer Grammatik einen Automat herzuleiten, ist es wichtig zu beachten, dass die Grammatik regulär ist.

2.1 Reguläre Grammatiken und deren Verabreitung

Endliche Automaten werden als Zustandsdiagramm dargestellt. Ein Zustandsdiagramm für eine Lampe lässt sich in Abbildung 2.1 finden.

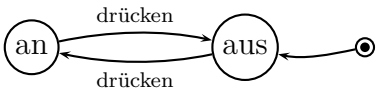


Figure 2.1: Zustandsdiagramm für eine Lampe

Mit jedem Druck auf eine Taste wird der Status der Lampe geändert. Der Startzustand ist dabei "aus".

Ein Automat für die schon vorher besprochene Grammatik: "'Länge des Wortes gerade"', könnte nach einer kurzen Überlegung wie in Abbildung 2.3 aussehen.

Die zugehörige formale Grammatik findet sich in Grafik 2.1

$\Sigma = \{0, 1\}$
$S \to 0T|1T|\epsilon$
$T \to 0S|1S$

Figure 2.2: Grammatik für Wörter mit gerader Länge

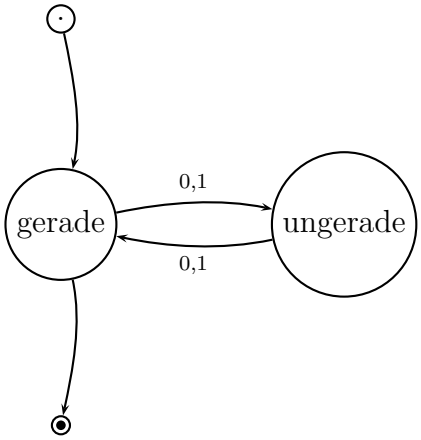


Figure 2.3: Intuitiver Automat für gerade Wörter

Dieser Automat ist leicht zu verstehen. Wir kommen durch den Startzustand in den Zustand "gerade". Mit einem Eingabesymbol (1 oder 0) kommen wir in den Zustand "ungerade", da das Wort jetzt die Länge 1 hat und damit ungerade ist.Fügen wir ein weiteres Eingabesymbol an, kommen wir wieder in den Zustand "gerade". Da dieser Zustand auch gleichzeitig Endzustand ist, wird das Wort mit der Länge 2 erkannt.

2.2 Erzeugen von endlichen Automaten

Nun stellt sich aber die Frage, wie man aus einer regulären Grammatik einen endlichen Automaten ableitet. Für diese Problemstellung, gibt es ein Kochrezept, welches nun kurz vorgestellt wird.

2.2.1 Schritt 1: Zustände definieren

Zustände in Automaten sind gleichbedeutend mit den verwendeten Variablen der Grammatik. Für jede Variable, die in der Grammatik definiert wurde, gibt es auch ein speziellen Zustand. In unserem Fall wurden die Variablen S und T verwendet.

Figure 2.4: Automat nach 1. Schritt

2.2.2 Schritt 2: Startzustand einzeichnen

Der Startzustand ergibt sich aus der formalen Definition der Grammatik. Da Zustände gleichbedeutend mit Variablen sind, ist auch der Startzustand gleichbedeutend mit der Startvariable.

Um einen Startzustand einzuzeichen, wird ein dickerer Punkt mittels Pfeil ohne Beschriftung auf den Startzustand gezeichnet.

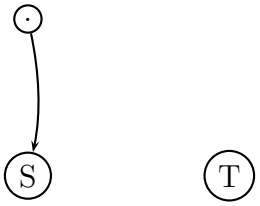


Figure 2.5: Automat nach Schritt 2

2.2.3 Schritt 3: Endzustände einzeichnen

Endzustände sind all diejenigen, bei denen die Verarbeitung eines Wortes aufhören kann. Klassischerweise enthält ein Endzustand das leere Wort (ϵ). Allerdings kann die Verarbeitung auch bei einem einzelnen Terminalsymbol beendet werden. Dazu aber an späterer Stelle mehr. Ein Endzustand wird durch einen Punkt mit umschlossenen Kreis dargestellt. Auf diesen Kreis zeigt dann ausgehend von einem Endzustand ein Pfeil. Es kann durchaus vorkommen, dass es mehrere Endzustänge gibt, wohingegen nur ein Startzustand existieren darf.

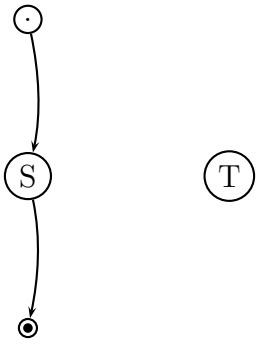


Figure 2.6: Automat nach dem 3. Schritt

2.2.4 Schritt 4: Kanten einzeichnen

Eine Kante oder Transition in einem Automat, wird durch eine Regel erzeugt. Da die Regeln in einer regulären Grammatik immer einem einfachen Shema folgen, ist die Erstellung von Kanten kein Problem. Kanten werden durch Pfeile auf andere Zustände dargestellt. Der linke Regelteil definiert den Anfang des Pfeils. Im rechten Regelteil stehen immer ein Termianlsymbol gefolgt von einer Variable. Die Variable im rechten Regelteil gibt an, auf welchen Zustand die Kante zeigen soll. Das Terminalsymbol der rechten Regelseite wird an die Kante geschrieben.

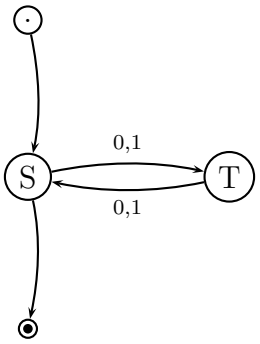


Figure 2.7: Automat nach Schritt 4

2.2.5 Sonderfall bei regulären Grammatiken

Regüläre Grammatiken lassen auch Regeln nach dem Schema $s \rightarrow a$ zu. Diese Regeln können nach dem bisher besprochenen Schema nicht in eine Kante für einen Automaten umgewandelt werden. Um diese Art von Regeln zu behandeln, ist ein kleiner Trick von nöten. Man erzeugt einfach einen neuen Zustand. Dieser Zustand wird durch die Variable auf der linken Regelseite und das Terminalsymbol benannt (S_a). Die Kante die zu diesem Zustand führt wird mit dem Terminalsymbol beschriftet. Der neue Zustand wird dann selbst zu einen Endzustand. Im folgender Grafik nehmen wir an, dass eine Regel $S \rightarrow a$ existiert.

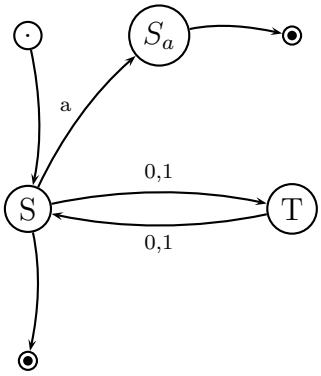


Figure 2.8: Automat mit zusätzlichem Zustand S_a

2.3 Determinismus

In der Automaten Therorie wird zwischen deterministischen und nicht deterministischen Automaten unterschieden. Ein nicht deterministischer Automat hat mindestens eine der folgenden Eigenschaften.

Uneindeutigkeit Ein Automat ist uneindeutig, wenn es für einen Zustand in einem Terminalsymbol mehrere Kanten und damit Möglichkeiten weiterzugehen gibt.

Unterspezifiziert Umgekehrt kann ein Automat unterspezifiziert sein, wenn es für einen Zustand und ein Terminalsymbol keine ausgehende Kante gibt.

Ist auch nur eine dieser Eigenschaften erfüllt, wird der Automat als nicht deterministisch eingestuft.

2.4 Suchbäume

Nicht deterministische Automaten sind nicht empfehlenswert. Um ein Wort zu parsen, was die Hauptaufgabe von Automaten ist, müssen bei nicht deterministischen Suchbäume eingesetzt werden. Um bei einem Suchbaum zu einem Ergebnis zu kommen, müssen möglicherweise alle Äste durchsucht werden. Dies begründet

sich dadurch, dass bei nicht deterministischen Automaten mehrere Möglichkeiten existieren, ein Wort zu erkennen.

$\Sigma = \{a, b\}$
$S \rightarrow aS|aT|a|b$
$T \rightarrow bT|\epsilon$

Figure 2.9: Nicht deterministische Grammatik

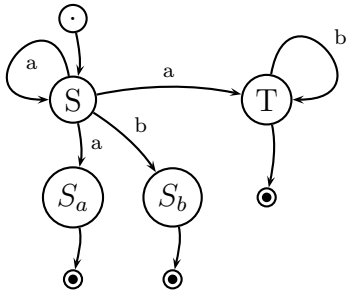


Figure 2.10: Nicht deterministischer Automat

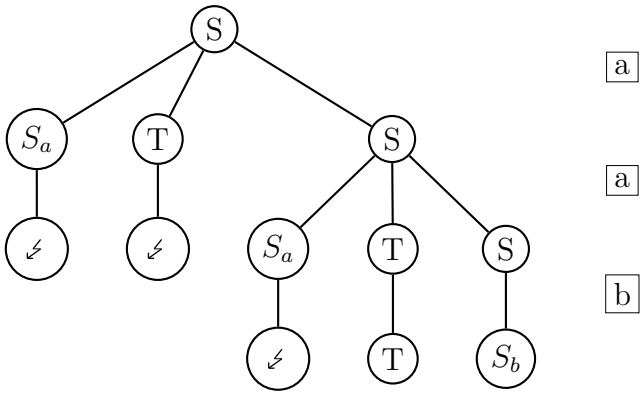


Figure 2.11: Suchbaum für das Wort "aab"

Nicht deterministische Automaten müssen wie eingangs schon gesagt über Suchbäume verarbeitet werden. Dadurch, dass es mehrere Möglichkeiten gibt zu einem Ergebnis zu gelangen, muss der Suchbaum durch das Backtracking Verfahren traversiert werden. Der Aufwand beim Backtracking kann quadratisch zum zu parsenden Wort sein. Je nachdem wie schnell ein Ast gefunden wird, der das Wort vollständig erkennt. Bei deterministischen Automaten ist der Aufwand hingegen immer linear. Es werden genau so viele Schritte benötigt um das Wort zu erkennen, wie das Wort lang ist. Deterministische Automaten parsen wesentlich schneller, als nicht deterministische.

2.5 Lazy Evaluation Verfahren

Wir haben im letzten Abschnitt festgestellt, dass nicht deterministische Automaten nicht optimal sind. Ziel ist es einen dterministischen Automaten herzuleiten. Im folgenden wird ein Verfahren aufgezeigt, dass es ermöglicht aus einem nicht deterministischen einen deterministischen Automaten herzustellen.

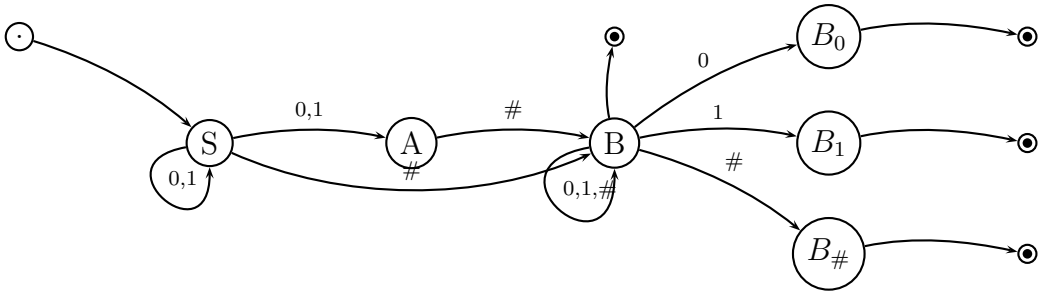


Figure 2.12: Beispiel für einen nicht deterministischen Automaten

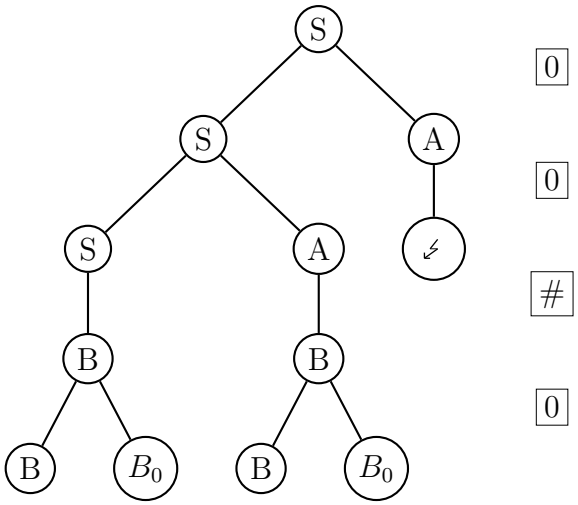


Figure 2.13: Zugehöriger Suchbaum für das Wort "00#0"

Wir werden den nachweislich nicht deterministischen Automaten mit Hilfe des Lazy Evaluation Verfahren einen equivalenten deterministischen Automaten bilden.

2.5.1 Schritt 1: Startzustand

Der Startzustand des NEA (nicht deterministischer endlicher Automat) wird zum neuen Startzustand des DEA (deterministischer endlicher Automat).

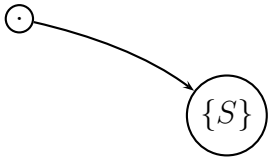


Figure 2.14: Automat nach dem ersten Schritt

2.5.2 Schritt 2: Folgezustand für Startzustand

Wir nehmen nun den Startzustand und überprüfen für jedes Eingabesymbol (Terminalsymbol), welche Folgezustände es im NEA gibt. Diese Folgezustände werden als Menge zusammengefasst und als einen eigenen Zustand im DEA erzeugt.

Für den Startzustand ergeben sich folgende Folgezustände:

$S \xrightarrow{0} \{S, A\}$
$S \xrightarrow{1} \{S, A\}$
$S \xrightarrow{\#} \{B\}$

Da für die Eingabesymbole 0 und 1 die Folgezustände gleich sind, werden hierfür keine neuen Zustände eingezeichnet. Es wrd lediglich eine neue Kante mit dem Eingabesymbol eingezeichnet.

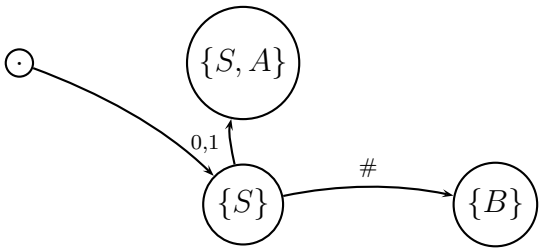


Figure 2.15: Automat nach dem zweiten Schritt

2.5.3 Schritt 3: Folgezustand für weitere Zustände

Nun nehmen wir die eben eingezeichneten Folgezustände des Startzustandes und errechnen wieder die Folgezustände für alle Eingabesymbole. Errechnet wird ein Folgezustand, indem wir die Folgezustände für das entsprechende Eingabesymbol für alle Elemente der Menge im Zustand in eine neue Menge vereinigen.

Die Folgezustände für den Zustand $\{S, A\}$ werden berechnet:

$S \xrightarrow{0} \{S, A\}$
$A \xrightarrow{0} \{\}$
(vereinigen:) $\{S, A\} \cup \{\} = \{S, A\}$

Der Folgezustand von $\{S, A\}$ mit dem Eingabesymbol 0 ist also $\{S, A\}$. Wir zeichenen also einen Verweis auf sich selbst. Alnalog zu diesem Schema werden auch die Folgezustände für die anderen Eingabesymbole (1 und #) berechnet und eingezeichnet.

Dieser Schritt wird solange wiederholt, bis der Automat deterministisch ist. Deterministisch ist ein Automat, wenn er für alle Eingabesymbole genau eine ausgehende Kante hat.

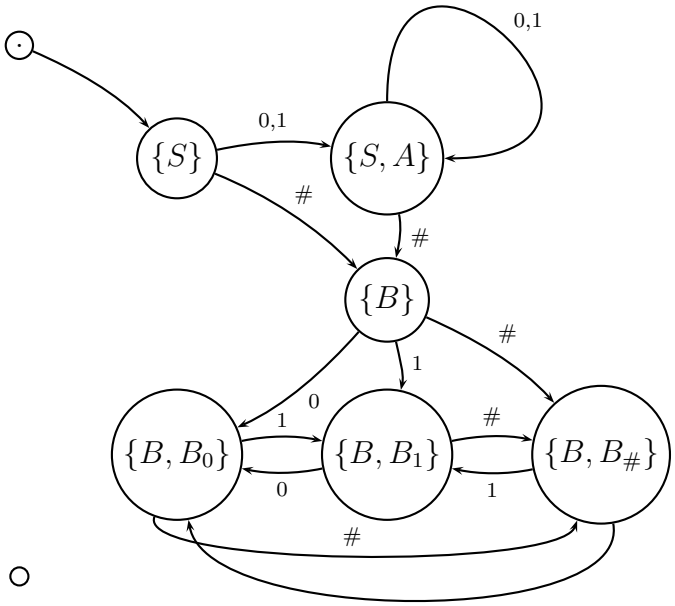


Figure 2.16: Automat nach dem dritten Schritt

2.5.4 Schritt 4: Endzustände

Jeder Zustand des neu angelegten DEA, der mindestens ein Element enthält, dass im NEA auch schon ein Endzustand war, wird zu einem neuen Endzustand.

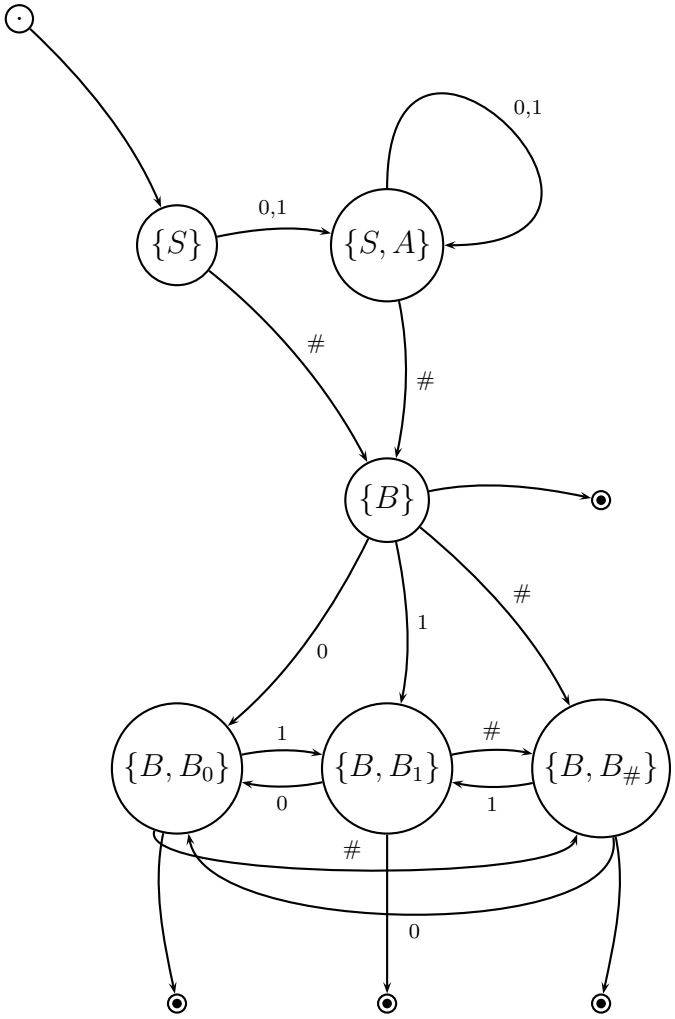
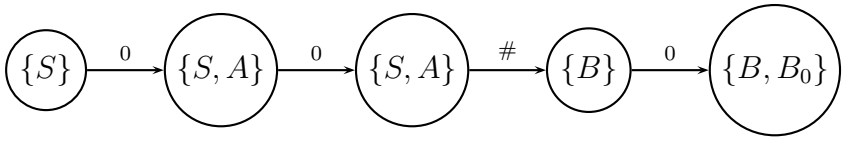


Figure 2.17: Automat nach dem dritten Schritt

Nach der Optimierung nach dem Lazy Evaluation Verfahren wird für das Wort 00#0 folgender Suchbaum erzeugt.



Wir erkennen, dass der zugehörige Suchbaum um ein vielfaches geschrumpft ist und zu dem auch linearisiert wurde. Das bringt enorme Geschwindigkeitsvorteile.