CYK Algorithmus

Projektarbeit von

Yener Özsoy

an der Hochschule Karlsruhe
Fakultät für Informatik und Wirtschaftsinformatik

Referent: Prof. Dr. Heiko Körner

Inhaltsverzeichnis

Zusammenfassung

Die theoretische Informatik beschäftigt sich mit grundlegenden Fragestellungen, deren Struktur und Verarbeitung, sowie dem Zusammenhang der zugrundeliegenden Informationen. Eines dieser Fragestellungen ist die Entscheidbarkeit. Hierbei versucht man auf einer Menge die Eigenschaft der Entscheidbarkeit zuzuordnen. Dies ist der Fall, wenn ein Entscheidungsverfahren, für jedes Element in der Menge, die Eigenschaft zuordnen kann. Wenn es kein Entscheidungsverfahren gibt, dann ordnet man dieser Menge die Eigenschaft der Unentscheidbarkeit zu.
In der heutigen Zeit werden sehr viele Anwendungen entwickelt, die den Menschen im Alltag, sowie im Berufsleben führen und unterstützen sollen. Diese reichen von Sprachassistenten und Smart Home Geräten, bis hin zu Steuergeräten von Sicherheitselementen, wie Airbags in den Autos. Für all diese Anwendungsbereiche werden Softwarelösungen angeboten, die sich in der Programmiersprache, der Architektur und dem Algorithmus unterscheiden. Während eine lange Berechnung des Sprachassistenten nicht von Wichtigkeit ist, ist dies für ein Airbag System von enormer Bedeutung. Man möchte garantieren, dass der zugrundeliegende Algorithmus mit jeder Eingabemenge eine Entscheidung, in der vorgegeben Zeit, treffen kann. Hier und in vielen anderen Problemen kommt der Begriff der Entscheidbarkeit ins Spiel.
Ein Problem aus dem Gebiet der theoretischen Informatik ist das Wortproblem. Das Wortproblem befasst sich mit den Typ 3 Sprachen nach der Chomsky Hierarchie. Diese werden auch als kontextfreie Sprachen bezeichnet. Hierbei soll angegeben werden, ob ein gegebenes Wort aus der kontextfreien Sprache konstruiert werden kann.
Der Cocke-Younger-Kasami-Algorithmus (CYK-Algorithmus) ist ein Entscheidungsverfahren, dass sich mit dem Wortproblem für kontextfreie Sprachen beschäftigt. Mit diesem Algorithmus lässt sich feststellen, ob ein Wort zu einer kontextfreien Sprache gehört.

1. Einführung

Das Wortproblem ist eine Problemstellung aus der theoretischen Informatik, dass sich mit der Fragestellung beschäftigt, ob ein gegebenes Wort aus der Grammatik konstruierbar ist. Man möchte für dieses, sowie für weitere Probleme und Algorithmen bestimmen können, ob diese für jede Eingabemenge terminieren und eine Entscheidung treffen können. Dabei differenziert man zwischen den Eigenschaften der Entscheidbarkeit bzw. der Semi-Entscheidbarkeit für Entscheidungsverfahren.
Neben der Kategorisierung dieser Verfahren, lässt sich auch unter anderem das Verfahren für weitere Beweisskizzen in der theoretischen Informatik einsetzen. Dazu nutzt man Reduktionsverfahren, bei denen das Problem auf ein anderes zurückgeführt wird. Teure und aufwendige Berechnungen können damit vermieden und Kosten eingespart werden. Auch semi-entscheidbare Probleme, die nur die halbe Charakteristische Funktion erfüllen, können für gewisse, eingeschränkte Mengen nachgewiesen werden. Im Vergleich zu anderen Methoden, wie zum Beispiel dem „Try and Error“ Verfahren, lassen sich somit wissenschaftliche Nachweise liefern. Zudem werden hohe Berechnungskosten auf Supercomputern eingespart.
Durch die Zusicherung der Entscheidbarkeitseigenschaft und der Komplexitätstheorie, kann für Algorithmen in lebensrelevanten Systemen die Terminierung der Berechnung und dessen maximaler Zeitaufwand garantiert werden.

Der CYK-Algorithmus ist ein Entscheidungsverfahren, dass die charakteristische Funktion für ein gegebenes Wort und einer Grammatik in Chomsky Normalform berechnet. Das bedeutet, dass der Algorithmus für jedes Wort terminiert und ein Ergebnis liefert, wenn das Wort aus der Grammatik ableitbar ist.
In den kommenden Kapiteln werden die Grundlagen behandelt, die zum Anwenden des Entscheidungsverfahrens benötigt werden. Außerdem werden die Technologien vorgestellt, die in der Projektarbeit eingesetzten wurden. Anschließend wird die zentrale Thematik des CYK Algorithmus erklärt und anhand eines Beispiels demonstriert. Im darauffolgenden Kapitel wird die technische Umsetzung des zuvor erklärten Algorithmus mit Java behandelt. Dazu werden Umsetzung und Designentscheidung der eingesetzten Technologien, aus verschiedenen Aspekten, begründet und dargestellt. Eine Anleitung zur Installation und Nutzung der Anwendung liegt bei.

2. Grundlage

Dieses Kapitel befasst sich mit den Grundlagen, auf denen die Projektarbeit basiert. Um den CYK-Algorithmus anwenden zu können, muss die zugehörige Grammatik eine kontextfreie Grammatik sein und in der Chomsky Normalform vorliegen. In dem Abschnitt 2.1 wird der Begriff kontextfreie Grammatik näher erläutert. In dem Abschnitt 2.2 wird die Chomsky Normalform und ihre Eigenschaften definiert. Die in Kapitel 7.1 definierte Grammatik dient als Beispiel und wird in diesem Abschnitt, sowie während des CYK-Algorithmus genutzt, um die theoretischen Aspekte anhand eines Beispiels zu demonstrieren.

2.1 Kontextfreie Grammatik

Grammatiken sind Modelle, die anhand von Produktionsregeln, formale Sprachen erzeugen. Sie besitzen ein Startsymbol und eine Regelmenge, die für bestimmte Zeichen neue Zeichenfolgen definiert, durch die sie ersetzt werden können. Das Vokabular einer Grammatik besteht aus den Mengen der Terminalsymbole und der Nichtterminalsymbole. Während Nichtterminalsymbole Ersetzungsregeln definieren, stellen die Terminalsymbole das Ende der Ableitungsregeln dar. Somit besteht eine Grammatik aus folgendem Tupel:
$G = (V, T, P, S)$, wobei

- V, eine endliche Menge von Symbolen,
- $T \subset V$, die Teilmenge der Terminalsymbole,
- $P \subset (V^* \backslash T^*) \times V^*$, die Menge der Produktionsregeln, sowie
- S, das Startsymbol darstellen.

Grammatiken können anhand von gewissen Eigenschaften in Kategorien bzw. Chomsky Hierarchien eingeteilt werden. Diese Hierarchien sind in 4 Typen kategorisiert. Jedes dieser Typen spezialisiert den vorherigen um weitere Eigenschaften. Die Typen sind folgende:

- Typ-0-Grammatiken – diese entsprechen der allgemeinen Chomsky Grammatik
- Typ-1-Grammatiken – diese entsprechen der kontextsensitiven Grammatik
- Typ-2-Grammatiken – diese entsprechen der kontextfreien Grammatiken
- Typ-3-Grammatiken – diese entsprechen der regulären Grammatiken

In diesem Abschnitt beschäftigen wir uns mit den Typ-2-Grammatiken. Diese haben die Eigenschaft, dass immer nur genau ein Nichtterminalsymbol auf eine Folge von Nichtterminalsymbolen und Terminalsymbolen abgeleitet wird. Einfacher gesagt, haben die Produktionsregeln die Länge eins auf der linken Seite.

2.2 Chomsky Normalform

Die Chomsky Normalform ist eine Normalform für kontextfreie Grammatiken. Sie hat eine einfache Struktur der Produktionsregeln die wie folgt definiert ist:
Eine formale Grammatik $G = (V, \Sigma, P, S)$ ist in Chomsky Normalform, wenn jede Produktion eine der folgenden Formen hat:

- $A \rightarrow BC$
- $A \rightarrow a$

Hierbei sind A, B und C Nichtterminalsymbole und a ein Terminalsymbol aus Σ.
Formell ausgedrückt darf die rechte Seite für Nichtterminalsymbole die Länge 2 nicht überschreiten. Falls auf ein Terminalsymbol abgebildet wird darf auf das Terminalsymbol kein weiteres Symbol mehr folgen.
„Zu jeder kontextfreien Sprache L existiert eine Grammatik G in Chomsky Normalform mit $L(G) = L$“ (Satz 3.3.3, Vorlesungsskript zu theoretischer Informatik von Heiko Körner).
Um diese Aussage zu beweisen benötigen wir das Lemma 1.
Der Beweis besteht aus vier Regeln, mit denen auch jede kontextfreie Grammatik in Chomsky Normalform überführt werden kann.

1. **Regel: Alle Regeln enthalten auf der rechten Seite eine Folge aus Nichtterminalsymbolen oder bilden nur auf ein Terminalsymbol ab**

   Für jedes Terminalsymbol $a \in \Sigma$ fügen wir ein neues Nichtterminalsymbol X_a ein mit der Ableitungsregel $X_a \rightarrow a$. Somit erweitern sich die Mengen V und P um das neue Nichtterminalsymbol und dessen Ableitungsregel. Dann wird jedes Vorkommen von a auf der rechten Seite der Ableitungsregel durch das neue Nichtterminalsymbol X_a ersetzt.
   Somit entstehen Regeln die entweder $A \rightarrow a$ mit $a \epsilon \Sigma$ oder $A \rightarrow B_1 B_2 \dots B_k$ mit $k \geq 2$ und $B_i \in V \backslash T$ mit $i \in \mathbb{N}$ erfüllen. Das bedeutet das jede Regel mit $A \rightarrow \cdots a B_1 \cdots$ jetzt durch $A \rightarrow \cdots X_a B_1 \cdots$ und $X_a \rightarrow a$ dargestellt werden kann. Es ist leicht zu sehen, dass sich die grundlegende Grammatik und die darin enthaltenen Produktionsregeln nicht ändern.

   Beispiel:

   Die Grammatik erfüllt diese Regel in den Ableitungen S, A, B und D nicht, da diese Produktionsregeln auf eine Folge abbilden, die Terminalsymbole enthält. Um die 1.Regel zu erfüllen, wird für die Terminalsymbole a und b das neue Nichtterminalsymbol A_1 bzw. B_1 eingeführt. Nach dem Anwenden der Regel sieht die Grammatik wie folgt aus:

$$
\begin{aligned}
&S \rightarrow A \mid A_1\ A\ A_1 \mid B_1 B\ B_1 \mid \varepsilon \\
&A \rightarrow C \mid a \\
&B \rightarrow b \\
&C \rightarrow C\ D\ E \mid \varepsilon \\
&D \rightarrow A \mid B \mid A_1 B_1 \\
&E \rightarrow B \mid A \\
&A_1 \rightarrow a \\
&B_1 \rightarrow b
\end{aligned}
$$

2. Regel: Alle rechten Seiten haben eine maximale Länge von zwei

Für jede Ableitungsregel mit einer Länge $k \geq 3$ führen wir $k-1$ neue Nichtterminalsymbole und Produktionen ein. Diese werden nach und an jedes Symbol angehängt und die darauffolgenden Symbole als Ableitungsregeln zu dem neuen, angefügten Nichtterminalsymbol hinzugefügt. Somit terminieren die Ableitungsregeln nach zwei Symbolen und die gesamte Produktionsregeln wird anhand der neu eingeführten Nichtterminalsymbole fortgeführt, sodass die Ableitungen erhalten bleiben. Man kann sich dieses Vorgehen wie ein Daten und Zeiger Paar vorstellen. Nach dem das erste Symbol ersetzend eingetragen wird, zeigt das zweite wie ein Zeiger zu den nächsten Ableitungsregeln.
Dieser Schritt wird für alle Symbole angewendet, mit Ausnahme der letzten zwei Symbole der Abbildung.
Die Abbildung $A \rightarrow B_1 B_2 \dots B_k$ wird umgeformt zu:

$$A \rightarrow B_1 A_1,\ A_1 \rightarrow B_2 A_2, \dots,\ A_{k-2} \rightarrow B_{k-2} A_{k-1},\ A_{k-1} \rightarrow B_{k-1} B_k$$

Beispiel:

Die Ableitungen in S und C überschreiten eine Länge von zwei. Diese werden nach der obigen Regel gekürzt. Es werden die neuen Nichtterminalsymbole S_1, S_2 und C_1 eingeführt.

$$\begin{aligned}
&S \rightarrow A \mid A_1 S_1 \mid B_1 S_2 \mid \varepsilon \\
&A \rightarrow C \mid a \\
&B \rightarrow b \\
&C \rightarrow C\ C_1 \mid \varepsilon \\
&D \rightarrow A \mid B \mid A_1 B_1 \\
&E \rightarrow B \mid A \\
&A_1 \rightarrow a \\
&B_1 \rightarrow b \\
&S_1 \rightarrow A\ A_1 \\
&S_2 \rightarrow B\ B_1 \\
&C_1 \rightarrow D\ E
\end{aligned}$$

3. Regel: Epsilon-Übergänge werden aufgelöst

Jede Ableitung der Form $A \to \varepsilon$ wird entfernt und Ableitungen der Form $A \to^* \varepsilon$ simuliert. Das heißt, dass alle Ableitungen $A \to BC$ mit $B \in V_\varepsilon$ oder $C \in V_\varepsilon$ zu der neuen Ableitung $A \to C$ bzw. $A \to B$ überführt werden. Dies betrifft alle Ableitungen, die nach endlich vielen Schritten Epsilon erreichen und eine Länge von $k \geq 2$ aufweisen. Um Epsilon für diese Produktionsregeln zu simulieren, wird Epsilon an die entsprechende Stelle auf der rechten Seite der Produktionsregeln eingesetzt.

Beispiel:

Die Ableitungen in S und C bilden auf ε ab. Folgende Ableitungen auf ε sind auch möglich, die es aufzulösen gilt:

$$S \to A \to C \to \varepsilon$$

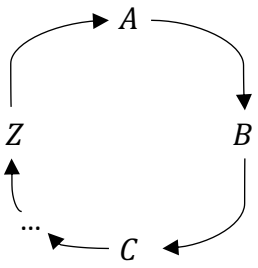
Diese Ableitung gilt es zu simulieren. Eine Teilfolge ist in C, sowie in S_1 enthalten. Diese werden aufgelöst, indem die betroffenen Stellen mit ε ersetzt werden. Da ε das leere Wort repräsentiert, kann es dann endgültig entfernt werden.
Für die Produktionsregel von S_1 wird das Nichtterminalsymbol A, durch das zu simulierende Epsilon ersetzt. Daraus entsteht für die erste Produktion:

$$AA_1 \Longrightarrow \varepsilon A_1 \Longrightarrow A_1$$

$S \to A \mid A_1 S_1 \mid B_1 S_2$
$A \to C \mid a$
$B \to b$
$C \to C\ C_1 \mid C_1$
$D \to A \mid B \mid A_1 B_1$
$E \to B \mid A$
$A_1 \to a$
$B_1 \to b$
$S_1 \to A\ A_1 \mid A_1$
$S_2 \to B\ B_1$
$C_1 \to D\ E \mid D \mid E$

4. Regel: Alle Kettenregeln auflösen

Als Kettenregel werden Ableitungen bezeichnet, die einen Zyklus beschreiben. Zyklen sind sich endlose wiederholende Muster bzw. Produktionsregeln. Man kann sich einen Zyklus auch als geschlossenen Kreis vorstellen, der kein Anfang und Ende hat.



Grafik 1 zu einem geschlossenen Zyklus

Das sind alle Ableitungen der Form $A \rightarrow^* A$. Um solch einen Zyklus aufzulösen werden die Produktionsregeln, die ineinander überführbar sind, zu dem Wurzelelement hinzugefügt. Alle Symbole, die auf der linken Seite der Produktionsregel standen werden durch das Wurzelelement ersetzt. Dadurch, dass das Wurzelelement alle Produktionsregeln enthält und die Vorkommnisse der am Zyklus beteiligten Nichtterminalsymbole ersetzt wurden, können diese aus dem Tupel der Grammatik entfernt werden.

Beispiel:

Es liegen zuletzt noch die Zyklen $A \rightarrow C \rightarrow C_1 \rightarrow D \rightarrow A$ und $A \rightarrow C \rightarrow C_1 \rightarrow E \rightarrow A$ vor, die es noch aufzulösen gilt. Dazu werden alle Produktionen der obigen Wurzelelemente zu A hinzugefügt. Alle Vorkommen von C, C_1, D und E werden durch A ersetzt. Die resultierende, zyklenfreie Grammatik muss noch aufgeräumt werden. Dazu werden mehrfach vorkommende, identische Produktionen eines Wurzelelementes entfernt, sodass nur noch eines vorhanden ist. Ableitungen auf nur ein Nichtterminalsymbol werden durch dessen Ableitungen ersetzt.

$$
\begin{aligned}
&S \rightarrow a \mid b \mid A\ A \mid A_1 B_1 \mid A_1 S_1 \mid B_1 S_2 \\
&A \rightarrow a \mid b \mid A\ A \mid A_1 B_1 \\
&B \rightarrow b \\
&A_1 \rightarrow a \\
&B_1 \rightarrow b \\
&S_1 \rightarrow A\ A_1 \mid a \\
&S_2 \rightarrow B\ B_1
\end{aligned}
$$

2.3 Entscheidbarkeit

Die theoretische Informatik beschäftigt sich mit grundlegenden Fragestellungen, deren Struktur und Verarbeitung, sowie dem Zusammenhang der zugrundeliegenden Informationen. Man möchte für diese Fragestellungen bzw. Problemstellungen ein Verfahren angeben, dass dieses löst. Doch nicht alle Problemstellungen lassen sich lösen. Ein Beispiel für ein nicht lösbares Problem, wäre das bekannte Halteproblem.
Zu diesen Problemstellungen lässt sich die Eigenschaft der Entscheidbarkeit zuordnen, wenn es ein Verfahren gibt, dass für diese Menge eine Berechnung oder Aktion durchführt, terminiert und eine Lösung angibt. Formal ausgedrückt bedeutet der Begriff entscheidbar folgendes:

$$\chi_T(t) = \begin{cases} 1, falls\ t \in T \\ 0, sonst \end{cases}$$

Neben der Entscheidbarkeit gibt es noch den Begriff der Semi-Entscheidbarkeit. Dieser ist wie folgt definiert:

2.3.1 Semi-Entscheidbarkeit

Als Semi-Entscheidbar werden Berechnungsverfahren bezeichnet, die nur die halbe charakteristische Funktion der Entscheidbarkeit erfüllen. Das bedeutet, dass das Verfahren nur dann terminiert, wenn die zugrunde liegende Menge auch in T enthalten ist. Während die Entscheidbarkeit, auf jeder Menge terminiert und die Eingabe akzeptiert, wenn $t \in T$ gilt, ist die Semi-Entscheidbarkeit für den nicht akzeptierenden Teil undefiniert.

$$\chi_T(t) = \begin{cases} 1, & falls\ t \in T \\ undefiniert, & sonst \end{cases}$$

Das Wortproblem ist ein Entscheidbarkeitsproblem der theoretischen Informatik. Sie behandelt die Fragestellung, ob ein gegebenes Wort zu einer formalen Sprache gehört. Der nachfolgende CYK-Algorithmus ist ein Verfahren für Typ-2-Grammatiken, um dieses Problem entscheiden zu können. Mit diesem Verfahren kann genau bestimmt werden, ob das gegebene Wort in der Grammatik liegt oder nicht.

2.4 Halteproblem

Das Halteproblem beschreibt die Problemstellung, für einen gegebenen Algorithmus und eine endliche Eingabe die Terminierung zu bestimmen. Dementsprechend wird nach einem Algorithmus bzw. einer Turingmaschine gesucht, die als Eingabe einen Algorithmus und eine endliche Eingabe erhält und diese ausführt. Hält der Algorithmus aus der Eingabe an, so soll der Algorithmus, der dieses ausgeführt hat, terminieren und die Eingabe bestätigen. Dieses Problem zählt zu den nicht entscheidbaren Problemen. Denn während das für einfache Algorithmen entscheidbar zu sein scheint, ist die Funktionsweise bei nicht haltenden Algorithmen undefiniert. [Quelle: Vorlesungsskript Prof. Dr. Körner – Seite 147, Satz 4.5.2]

2.5 Zeitkomplexität

Unter der Zeitkomplexität eines Problems versteht man die Anzahl der Rechenschritte, die ein Algorithmus braucht um das Problem zu lösen. Dabei wird die Anzahl der Rechenschritte in Relation mit der Länge der Eingabe gestellt, da diese Größen in Abhängigkeit zueinanderstehen, um das Problem zu lösen. Die Zeitkomplexität beschreibt hierbei nicht, wie viel Zeit eine spezifische Maschine zum Lösen des Problems benötigt, sondern viel mehr, wie der Zeitbedarf mit steigender Eingabemenge wächst.
Für die Zeitkomplexität gibt es folgende drei Laufzeitvarianten:

1. Die Worst-Case Laufzeit:
   Dies ist die maximale Zeit, die für Eingaben einer bestimmten Größe erforderlich sind.

2. Die Average-Case Laufzeit:
   Diese gibt die durchschnittliche Laufzeit des Algorithmus bei einer gegebenen Verteilung der Eingaben an. Da jedoch nicht immer eine Verteilung der Eingaben bekannt ist, ist es nicht immer möglich die erwartete Laufzeit zu ermitteln.

3. Die Best-Case Laufzeit:
   Diese gibt die minimale Laufzeit an, die der Algorithmus bei einer Eingabe braucht, auf der sie ideal arbeitet.

Die am häufigsten vorkommende Variante ist die „worst-case“ Laufzeit, die in groß O Notation angegeben wird. Beispielsweise gilt ein Algorithmus als linear, wenn die Zeitkomplexität mit der Eingabe gleichmäßig wächst. Für eine Eingabe mit der Länge n würde man demnach die Zeitkomplexität als $O(n)$ angeben.

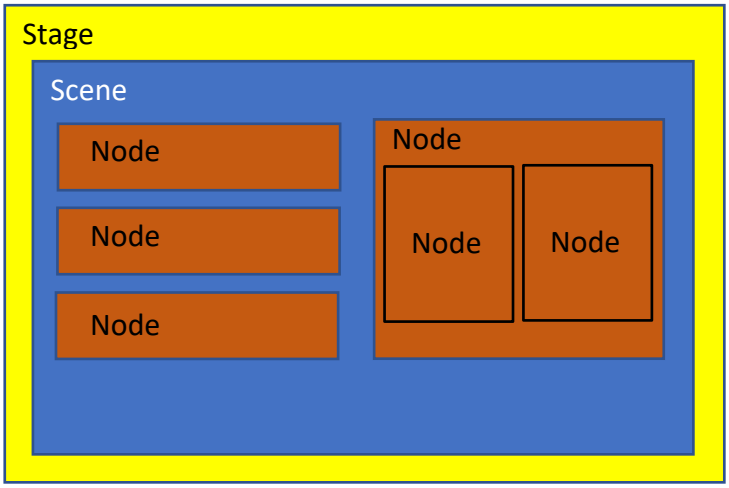
2.6 Java

Java ist eine objektorientierte Programmiersprache, die erstmals 1995 von James Gosling bei Sun Microsystems entwickelt wurde. Im Jahr 2010 wurde Sun Microsystems von Oracle aufgekauft. Die Programmiersprache Java ist eine der populärsten Programmiersprachen und ist Bestandteil der Java-Technologie. Diese besteht aus dem Java-Entwicklungswerkzeug (engl. Java Development Kit **JDK**) und der Java-Laufzeitumgebung (engl. Java Runtime Environment **JRE**). Die Java-Laufzeitumgebung umfasst unter anderem auch die virtuelle Maschine (engl. Java Virtual Machine **JVM**), auf der Java-Bytecode ausgeführt wird. Demnach wird Java Code beim Kompilieren in Java-Bytecode umgewandelt.

Der Quellcode eines Programms, dass in Java geschrieben wurde, kann nicht direkt ausgeführt werden. Dieser Quellcode ist ein, nach der Java Sprachdefinition (engl. Java Language Specification) definierter Text, der für den Menschen verständlich ist und beim Kompilieren in maschinenverständlichen Java-Bytecode übersetzt wird.
Java-Bytecode wird nicht direkt von der Hardware ausgeführt, sondern von der JVM. Dadurch erreicht die Java-Technologie eine Plattformunabhängigkeit. Der Java-Bytecode muss nicht für jedes System separat in die entsprechende Maschinensprache kompiliert werden, sondern kann einmal kompiliert auf jedem System, dass die JVM ausführen kann gestartet und ausgeführt werden.

2.7 JavaFX

JavaFX ist eine Java-Spezifikation von Oracle, die über sämtliche Java-Plattformen das Erstellen von graphischen Oberflächen (engl. Graphical User Interface **GUI**) erleichtern soll. Das Framework bietet mit FXML eine deklarative Beschreibung der Oberfläche, basierend auf der Auszeichnugssprache XML.
Die Oberfläche wird bei JavaFX durch eine Stage, dem sogenannten Rahmen dargestellt. Dieser Rahmen beinhaltet dann eine Szene (engl. Scene), die den Szenengraphen verwaltet. Der Szenengraph ist eine objektorientierte Datenstruktur, die häufig bei der Entwicklung von graphischen Oberflächen in Anwendungen genutzt wird. Dabei besteht der Szenengraph aus einem zusammenhängenden gerichteten Graphen ohne gerichtete Kreise.
Die Szene stellt in JavaFX das Wurzelelement des gerichteten Graphen dar. Unter dieser befinden sich weitere Knoten (engl. Nodes). Die Knoten sind entweder Wurzelknoten, die weitere Knoten enthalten oder es sind Blätterknoten, die Elemente auf dem Bildschirm repräsentieren.



Grafik 2 zum Aufbau einer Stage

2.8 XML

Die Erweiterbare Auszeichnungssprache (engl. Extensible Markup Language **XML**) ist eine Auszeichnungssprache, die am 10. Februar 1998 vom World Wide Web Consortium (W3C) veröffentlicht wurde. XML dient zur Darstellung hierarchisch strukturierter Daten in einer Textdatei. Die Sprache ist Plattform- und Implementationsunabhängig, welches den Gebrauch weit verbreitet. Mit der XML Technologie können Daten strukturiert werden und je nach Anwendungsfall gespeichert oder übertragen werden. So werden beispielweise häufig Daten in Binär oder Textform von Vektorgrafikprogrammen gespeichert oder zum Austausch der Daten über das Netzwerk genutzt.
Die Sprache nutzt, so ähnlich wie HTML, Tags und Attribute. Der Unterschied zu HTML ist jedoch, dass die Tags vom Nutzer selbst festgelegt werden können, während sie bei HTML eine Bedeutung tragen. Tags werden in spitzen Klammern geschrieben. Jeder Tag beginnt die Definition des Inhalts und beendet diesen durch erneute Notation des Tags mit voranschreitendem Schrägstrich („/“) vor dem Tagnamen.
Ein Beispiel würde so aussehen: $< diesIstEinTag > Wert </diesIstEinTag >$. Tags werden hierarchisch angelegt und besitzen ein Wurzelelement. Dadurch kann man die Struktur eines XML Dokumentes auch auf eine Baumstruktur übertragen. Durch die Nutzung von Tags, werden XML Dateien im Vergleich zu binären Formaten größer. Auf Kosten der Speicherverwaltung gewinnt man jedoch eine Datenstruktur, die auch von einem Menschen gelesen und interpretiert werden kann. Da heutzutage Festplattenerweiterungen an Rechnersystemen keine teure Anschaffung mehr sind, ist der Platzbedarf kein negativer Aspekt von großer Bedeutung.
Doch die Nutzung von Tags bietet auch Vorteile. Darunter besteht der direkte Zugriff auf Tags mit Hilfe von XML Parsern. Diese ermöglichen Daten auszulesen ohne das ganze Dokument zu interpretieren.
Die Syntax von XML ist recht simpel gehalten. Die Zeichen, die zwischen dem Start- und End-Tag stehen werden Inhalt genannt und können beliebig groß sein. Es können Elemente ohne Inhalt existieren. Die Benennung der Tags können eine beliebige Kombination aus Klein- und Großbuchstaben sein. Jedoch müssen Start- und End-Tag miteinander übereinstimmen. Den XML-Elementen können mit Hilfe von Attributen zusätzliche Eigenschaften verliehen werden. Dazu werden dem Start-Tag Name und Wert zugeordnet.

Beispiel:

$$
\begin{aligned}
&< Projektarbeit\ semester = \text{"WS 2018/2019"} > \\
&\quad < titel > CYK\ Algorithmus </title > \\
&\quad < vorname > Yener </vorname > \\
&\quad < nachname > Oezsoy </nachname > \\
&\quad < student > \\
&\qquad < studiengang > Informatik </studiengang > \\
&\qquad < hochschule > \\
&\qquad\qquad Hochschule\ Karlsruhe\ Technik\ und\ Wirtschaft \\
&\qquad </hochschule > \\
&\quad </student > \\
&</Projektarbeit >
\end{aligned}
$$

2.8.1 Beispiel Grammatik in XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<grammatik>
  <name>Grammatik</name>
  <n>S;A;B;C;D;E</n>
  <t>a;b</t>
  <s>S</s>
  <p>
    <zeile>
      <zelle>S</zelle>
      <zelle>a A a</zelle>
      <zelle>b B b</zelle>
      <zelle>A</zelle>
      <zelle>/Eps</zelle>
    </zeile>
    <zeile>
      <zelle>A</zelle>
      <zelle>C</zelle>
      <zelle>a</zelle>
    </zeile>
    <zeile>
      <zelle>B</zelle>
      <zelle>b</zelle>
    </zeile>
    <zeile>
      <zelle>C</zelle>
      <zelle>C D E</zelle>
      <zelle>/Eps</zelle>
    </zeile>
    <zeile>
      <zelle>D</zelle>
      <zelle>A</zelle>
      <zelle>B</zelle>
      <zelle>a b</zelle>
    </zeile>
    <zeile>
      <zelle>E</zelle>
      <zelle>B</zelle>
      <zelle>A</zelle>
    </zeile>
  </p>
  <change>mark(S); highlight(a,0); add(A1); addTo(A1,a); replace(S,a,A1); mark(S); highlight(a,2); replace(S,a,A1); mark(S); highlight(b,0); add(B1); addTo(B1,b); replace(S,b,B1); mark(S); highlight(b,2); replace(S,b,B1); mark(D); highlight(a,0); replace(D,a,A1); mark(D); highlight(b,1); replace(D,b,B1)</change>
</grammatik>
```

2.9 MVC

Das MVC Paradigma ist ein Entwurfsmuster aus der Softwaretechnik für wiederkehrende Entwurfsprobleme. MVC steht für Model View Controller (Modell Präsentation Steuerung) und stellt ein Muster zur Umsetzung eines Programms mit graphischer Oberfläche bereit. Hierbei werden drei Komponenten mit verschiedenen Aufgaben definiert.

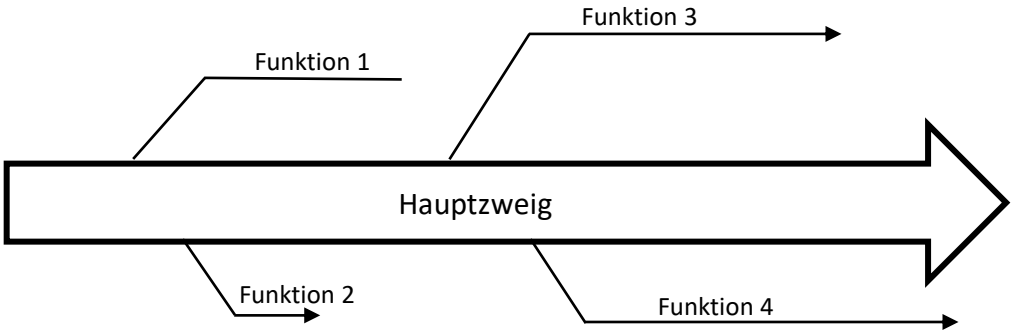
Modell:
Das Modell enthält die Daten und verarbeitet diese. Es ist von den anderen Komponenten unabhängig, bietet jedoch Zugriff auf die Zustandsdaten.

View:
Diese Komponente ist für die Darstellung der Daten und der Realisierung der Benutzeraktionen zuständig. Dafür kennt die Präsentationsschicht das Modell, dass die Daten beinhaltet und verwaltet. Selbst führt die Präsentationsschicht keine Verarbeitung der Daten durch, sondern zeigt diese lediglich an. Benutzerinteraktionen werden an die Steuerung weitergeleitet, die je nach Zustand die entsprechenden Komponenten für eine Änderung informiert.

Controller:
Die Steuerung verwaltet die Modell- und die Präsentationskomponente und ist für die Kommunikation der beiden zuständig. Dabei kann sie eine bis mehrere Präsentationskomponenten beinhalten, Benutzeraktionen von diesen entgegennehmen und auswerten. Je nach Aktion agiert dann die Steuerungskomponente und stößt die benötigten Methoden der Modellkomponente an.

2.10 GitHub

GitHub ist ein Online Versionsverwaltungssystem, dass im Februar 2008 von den Entwicklern Chris Wanstrath, PJ Hyett, Scott Chacon und Tom Preston-Werner veröffentlicht wurde. GitHub basiert auf dem Versionsverwaltungssystem Git und wurde mit den Programmiersprachen Ruby on Rails und Erlang geschrieben.
Versionsverwaltungssysteme bieten den Entwicklern die Möglichkeit, bei einem Projekt verschiedene Versionen zu pflegen. Dabei werden die Änderungen an Dokumenten erfasst und mit Informationen abgelegt. Dazu gehören die Zeitstempel, zu denen sie in das System eingepflegt wurden, die Benutzerkennungen und Kommentare zu diesen. Dadurch können alle Entwicklungsschritte eingesehen und sogar zu einem früheren Stand zurückgesetzt werden. Neben den herkömmlichen Verwaltungssystemen bietet Git weitere Funktionen, die den Entwicklern einen großen Vorteil bieten.
Mit Hilfe von Git können verschiedene Entwicklungszweige geöffnet und unabhängig voneinander entwickelt werden. Dies ermöglicht gleichzeitige Implementierung verschiedener Funktionalitäten, die wiederum am Ende zusammengeführt (Merge) werden können.



Grafik 3 zur Versionierung mit Git

Der Hauptzweig zeigt die Entwicklung der Anwendung in einem Zeitstrahl. Die verschiedenen Funktionen sind unabhängige Teilprojekte der Anwendung, die zu verschiedenen Stadien gestartet werden. Die Länge des Pfeiles soll dabei die Implementierungszeit repräsentieren. Während Funktion 2 recht früh abgeschlossen ist, so wird Funktion 4 noch in der Gegenwart entwickelt. Ein Entwickler ist nicht auf den einzelnen Zweig gebunden, sondern kann auch zwischen den Zweigen wechseln, ohne das Daten dabei verloren gehen.
Ein weiterer Vorteil von Git ist die Verteilung des Quellcodes (auch Repository genannt). Die Daten liegen lokal auf dem Rechner vor und können nach belieben bearbeitet werden. Änderungen an diesen geschehen lokal und sind erst für alle sichtbar, wenn der Entwickler diese auf den Server überträgt und persistiert (Commit). Um die neuste Version zu erhalten oder an dem aktuellsten Zustand zu entwickeln, müssen die anderen Entwickler ihr Projekt von dem Server aktualisieren (Update).
GitHub bietet diese Funktionen als Onlinedienst an. Sie stellen die Server zur Verfügung, so dass Unternehmen keine eigenen Server hosten und Benutzerkonten verwalten müssen. Die Repositories können standardmäßig von allen eingesehen werden. Dies umfasst auch Besucher der Seite, die kein Benutzerkonto haben. Außerdem stellt GitHub die Möglichkeit bereit, die globale Sichtbarkeit der Repositories abzuschalten.

3. CYK Algorithmus

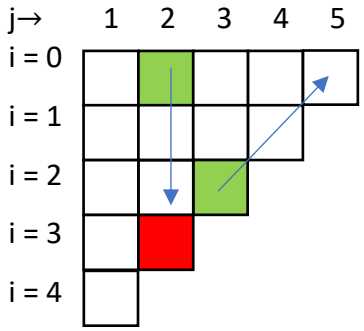
Der CYK-Algorithmus ist nach den Autoren Cocke, Younger und Kasami benannt. Dieses Verfahren löst das Wortproblem für reguläre Grammatiken. Die einzige Voraussetzung hierfür ist, dass die Grammatik in Chomsky-Normalform vorliegen muss.
Das gegebene Wort wird mit Hilfe eines Parse-Trees der Grammatik konstruiert. Das bedeutet, dass man das Wort w in Teilworte zerlegt und diese versucht mit der Grammatik zu rekonstruieren. Dabei werden die Teilworte nach und nach zusammengefügt und weiter durch die Grammatik abgebildet. Nach endlich vielen Schritten bildet das Teilwort genau das gegebene Wort w, falls dieses in der Grammatik liegt. Ansonsten lässt sich das Wort nicht rekonstruieren. Dieser Ansatz nutzt das Prinzip der dynamischen Programmierung.

3.1 Algorithmus

Der Algorithmus erhält als Eingabe eine kontextfreie Grammatik $G = (N, T, P, S)$ in Chomsky-Normalform und ein Wort $w = w_1 w_2 \dots w_k$ für den es gilt zu prüfen, ob es durch die Grammatik erzeugt werden kann ($w \in T^*$). Der CYK-Algorithmus besteht aus einer sogenannten Treppenform. Die Treppenform hat k Stufen für ein Wort w, wobei $|w| = k$ die Länge des Wortes darstellt. Im aller ersten Schritt wird die Treppenform initialisiert. Dabei wird das Wort w in die kleinsten Teilwörter zerlegt. Für diese lässt sich recht einfach bestimmen, ob diese durch die Grammatik beschreibbar sind, da sie die Terminalsymbole darstellen. Durch die Chomsky-Normalform ist nämlich gewährleistet, dass Ableitungen auf Terminalsymbole immer die Länge eins haben. Außerdem bilden Terminalsymbole auf keine weiteren Symbole ab. Auf den höheren Stufen wird mit steigendem Zeilenindex die Länge des zu betrachtenden Wortes länger. Pro Stufe, die man aufsteigt, betrachtet man ein Zeichen mehr. Aufgrund der steigenden Länge des Teilwortes, verkürzen sich die Zeilenlängen. Der Grund hierfür ist, dass höher gelegene Stufen die darunter liegenden Teilbäume beinhalten bzw. auf diese abbilden. Dadurch entsteht auch die charakteristische Treppenform des Algorithmus.
Auch wenn der Teilbaum auf höheren Stufen größer ist, müssen immer nur zwei Zellen als Feldpaar betrachtet werden. Denn Ableitungen haben durch die Chomsky-Normalform eine Länge von zwei. Die Feldpaare werden so gewählt, dass alle zugehörigen Teilworte durch die unteren Teilbäume abgedeckt werden. Hierfür sind mehrere Kombinationen möglich, die auch alle durch den Algorithmus abgedeckt werden. Hier bietet es sich an, dass man die ermittelten Ergebnisse für die unteren Teilbäume abspeichert, um sie nicht erneut zu berechnen. Durch den direkten Zugriff auf diese Ergebnisse spricht man bei dieser Art von Vorgehen, auch von einer dynamischen Programmierung.
Bei der Betrachtung der unteren Teilbäume ergibt sich eine Struktur für die Abarbeitung aller Ableitungskombinationen. Sei $Z_{i,j}$ die zu betrachtende Zelle in Zeile i und Spalte j mit $i \in \mathbb{N}$ und $j \in \mathbb{N}^*$, dann bildet diese Zelle auf das Teilwort $w_{j,i+j}$ ab, wobei der erste Index den Startindex und der zweite Index den Endindex angibt.
Für die erste Iteration gibt es keine Feldpaarkombinationen. Jede Zelle mit $Z_{i,j}$ bildet auf die darunterliegenden Symbole, den Terminalsymbolen ab. Daher gilt für die Zeile mit Index 0 $w_{j,j}$ für die Zelle $Z_{i,j}$, da $i + j = j$ gilt. Hier ist zu beachten, dass die Indizes für das Wort w nicht die Zeilen und Spalten, sondern den Start- und Endindex beschreiben.
Führen wir das Ganze für die nächste Iteration weiter, so bildet die erste Zelle mit $Z_{1,j}$ auf das Teilwort $w_{j,j+1}$ ab. Es folgt das Teilwort $w_{j+2,i+j+2}$, welches durch die Zelle

$Z_{i-2,j+2}$ repräsentiert wird. Wir können erkennen, dass für das erste Teilwort folgendes gilt: $Z_{x,j}$ mit $x \in \mathbb{N}$ bildet ab auf das Wort $w_{j,j+x}$. Für das zweite Teilwort gilt: $Z_{i-1-x,j+1+x}$ mit $x \in \mathbb{N}$ bildet ab auf das Wort $w_{j+1+x,j+i}$. Der Wert von x überschreitet jedoch niemals i, denn für das erste Teilwort befindet sich die Auswahl der Zelle immer in derselben Spalte. Es ändert sich lediglich der Zeilenwert. Demnach muss $x < i$ gelten, da es sonst direkt die auszufüllende Zelle betrifft bzw. eine Zelle die diese als Teilbaum beinhaltet.
Die letzte Feldpaarkombination, für das erste Teilwort $w_{j,j+i-1}$ kann durch die Zelle $Z_{i-1,j}$ mit dargestellt werden. Für das zweite Teilwort wird die Zelle $Z_{i-1-i+1,j+1+i-1}$, also vereinfacht $Z_{0,j+i}$ betrachtet. Diese Zelle bildet ab auf das Teilwort $w_{j+1+i-1,j+i} = w_{j+i,j+i}$.

Graphische Darstellung:



Grafik 4 zur Darstellung der Feldpaarkombinationen

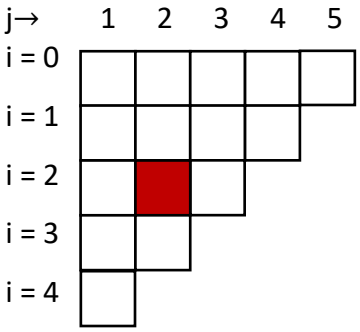
Die Zelle $Z_{3,2}$ stellt das Wurzelelement für das Teilwort $w_{2,5}$ dar. Die grün markierten Zellen veranschaulichen die erste Iteration des darunterliegenden Teilbaumes für $x = 0$. Mit steigendem Wert x, bewegen sich diese Markierungen entlang der Pfeile.

Das ganze zusammengefasst in Pseudo-Code:

```
for x = 0 to (i – 1) do {
   A_1 := Z_{x,j}
   A_2 := Z_{i-1-x, j+1+x}
   Z_{i,j} := Z_{i,j} ∪ {A | A ∈ V: A → A_1 A_2}
}
```

Beispiel: Kombination der Feldpaare

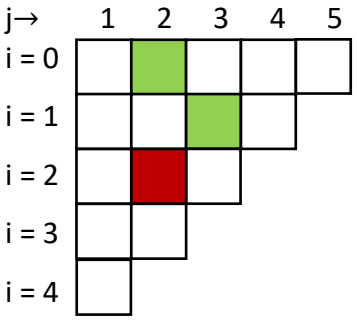
Wir betrachten nun folgende Treppenform:

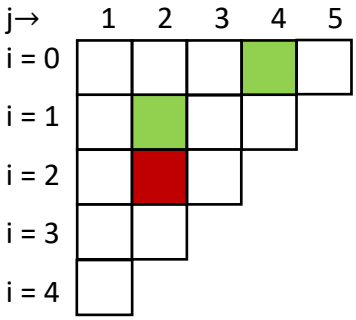


Grafik 5 als Beispiel für die Feldpaarkombinationen

Bei der Treppenform ist die Ausrichtung nicht von Bedeutung. Diese kann wie im obigen Beispiel umgekehrt sein. Die Ausführung des Algorithmus bleibt gleich.
Die oberste Reihe stellt die Nichtterminalsymbole dar, die auf die Terminalsymbole des Wortes an den entsprechenden Stellen zeigen.

Für die rot eingefärbte Zelle müssen die folgenden Zellen nach obiger Funktion betrachtet werden:





Grafiken 5.1 und 5.2 als Beispiel für die Feldpaarkombinationen

Das entscheidende ist die unterste Zelle, die die Wurzel der Treppenform darstellt. Ist am Ende darin das Startsymbol vorhanden, so liegt das Wort in der Grammatik. Ist das Feld leer so ist das Wort aus der Grammatik nicht ableitbar.

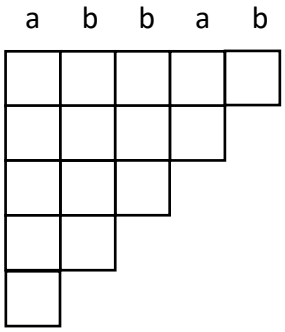
3.2 Zeitkomplexität

Sei n die Wortlänge für ein gegebenes Wort w und A ein zweidimensionales Array als Treppenform des CYK Algorithmus. So benötigt man zwei ineinander verschachtelte Schleifen, die über alle Felder iterieren. Diese werden, bedingt durch die Treppenform, pro Zeile kürzer,

aber dennoch benötigt das Iterieren über alle Felder im „worst-case“ Szenario $O(n^2)$. Als innerste Schleife wird die oben definierte Funktion genutzt. Hier ist die längste Berechnungszeit im Wurzelelement. Also die alleinstehende Stufe, die das Startsymbol symbolisiert. Für das Wort mit der Länge n ist diese Stufe an der Stelle $i = n - 1$. Grund dafür ist der Startindex von i, welches bei dem Wert 0 beginnt. Betrachten wir nun den Algorithmus so ist die Laufzeit in Landau-Notation $O(i - 1)$. Ein zusätzlicher Zugriff wird benötigt um diese Zelle zu lesen und zu entscheiden ob das Wort in der Grammatik liegt oder nicht. Dafür muss das Startsymbol in dieser stehen. Das ergibt eine Laufzeit von $O(i - 1 + 1) = O(i) = O(n)$. Dies entspricht einer Gesamtkomplexität von $O(n) * O(n) * O(n) = O(n^3)$.

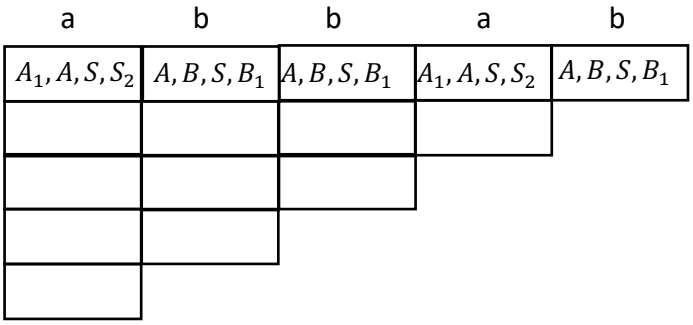
3.3 Beispiel

Wir nehmen für das Beispiel die Grammatik aus dem Kapitel des CNF. Wir betrachten das Wort $w = abbab$. Als erstes initialisieren wir die Treppenform. Das Wort w hat eine Länge von $|w| = 5$. Dementsprechend hat die Treppenform genau fünf Stufen.



Grafik 6 zur beispielhaften Durchführung des CYK Algorithmus

Wir schreiben über die Treppenform die einzelnen Symbole des Wortes, die die Terminalsymbole an dieser Stelle repräsentieren. Als nächstes gilt es die erste Iteration durchzuführen, indem wir in die erste Zeile die Nichtterminalsymbole eintragen, die auf die Terminalsymbole an dieser Stelle abbilden.



Grafik 6.1 zur Darstellung der 1.Iteration

Nun betrachten wir die zweite Iteration. Um eine Zelle aus dieser Reihe zu befüllen, müssen wir die direkt unter ihr befindliche und die davon rechte benachbarte Zelle betrachten. Kommagetrennte Einträge zeigen mehrere Ableitungsmöglichkeiten. Demnach müssen immer alle Kombinationen aus diesen Möglichkeiten in Betracht gezogen werden.

a	b	b	a	b
A_1, A, S, S_2	A, B, S, B_1	A, B, S, B_1	A_1, A, S, S_2	A, B, S, B_1
A, S	A, S, S_3	A, S, S_2	A, S	

Grafik 6.2 zur Darstellung der 2.Iteration

Für die dritte Iteration ist zu beachten, dass nun mehrere Feldpaare dieselben Teilworte abbilden können. Wenden wir die oben definierte Funktion an, so ergibt sich folgendes Ergebnis:

a	b	b	a	b
A_1, A, S, S_2	A, B, S, B_1	A, B, S, B_1	A_1, A, S, S_2	A, B, S, B_1
A, S	A, S, S_3	A, S, S_2	A, S	
A, S	A, S	A, S		

Grafik 6.3 zur Darstellung der 3.Iteration

Für die nächsten Iterationen gilt analog dasselbe wie für die dritte.

a	b	b	a	b
A_1, A, S, S_2	A, B, S, B_1	A, B, S, B_1	A_1, A, S, S_2	A, B, S, B_1
A, S	A, S, S_3	A, S, S_2	A, S	
A, S	A, S	A, S		
A, S, S_2	A, S			

Grafik 6.4 zur Darstellung der 4.Iteration

Für die letzte Iteration gilt, dass die Zelle grün markiert wird, falls das Wort w in der Grammatik liegt. Eine rote Markierung würde für die charakteristische Funktion, den Fall $\chi_T(w) = 0$ zeigen.

a	b	b	a	b
A_1, A, S, S_2	A, B, S, B_1	A, B, S, B_1	A_1, A, S, S_2	A, B, S, B_1
A, S	A, S, S_3	A, S, S_2	A, S	
A, S	A, S	A, S		
A, S, S_2	A, S			
A, S				

Grafik 6.5 zur Darstellung der 5.Iteration

Wir sehen, dass sich das Wort $abbab$ in der Grammatik befindet und beispielsweise mit $S \to AA \to AA_1B_1 \to AAA_1B_1 \to AAAA_1B_1 \to A_1B_1AA_1B_1 \to abbab$.

4. Umsetzung

Das Projekt wurde in Java umgesetzt. Dadurch konnte gewährleistet werden, dass das Programm Plattformunabhängig ist und auf allen Systemen läuft, die die JVM unterstützen.
Der Quellcode wurde nach dem MVC Paradigma implementiert. Dies ist ein Entwurfsmuster der Softwaretechnik, dass für Anwendungen mit grafischer Nutzeroberfläche genutzt werden kann. Hierbei wird die Logik der Anwendung in drei Komponenten verteilt. Die Daten, sowie die Geschäftslogik werden durch die Modellkomponente implementiert. Die Nutzeroberfläche (Graphical User Interface **GUI**) wird durch die Präsentationskomponente umgesetzt. Dazu gehören nicht nur das Anzeigen der verschiedenen Ansichten, sondern auch die Interaktionsmöglichkeiten mit dem Benutzer durch Eingaben an der Tastatur und der Maus. Die Kommunikation und Verwaltung dieser unabhängigen Komponenten werden von dem Controller gehandhabt.

4.1 Funktionen

Im kommenden Abschnitt werden die umgesetzten Funktionalitäten und ihre Realisierung thematisiert. Diese umfassen Benutzerinteraktionen an der Oberfläche, von manuellen bis automatisierten Mechanismen, den verschiedenen Darstellungsmöglichkeiten und der Repräsentation der Daten innerhalb der Anwendung.

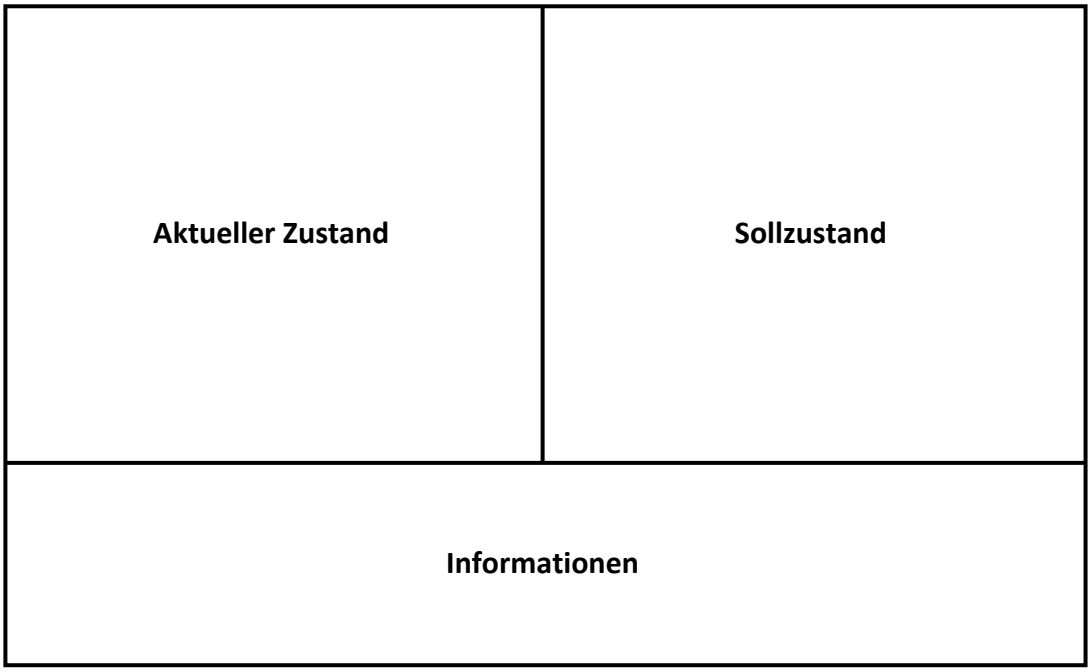
4.1.1 Benutzeroberfläche

Die Benutzeroberfläche besteht aus einer Stage und verschiedenen Szenen. Jeder dieser Szenen stellt einen Zustand der Anwendung dar. So gibt es beispielsweise ein Startfenster, indem der Nutzer eine neue Grammatik erstellen kann oder eine vorhandene in das Programm laden kann. Je nach Auswahl, geht die Anwendung entweder zu den Eingabefeldern für die neue Grammatik über oder öffnet das Auswahlfenster für die Algorithmen. Die Szenen werden nicht durch neue Stages repräsentiert, sondern werden von der einzigen Stage angezeigt, die beim Starten erzeugt wird. Der Vorteil hierbei ist, dass Ladezeiten vermieden werden, da die Szenen ausgeblendet und die aktuelle Zustandsszene eingeblendet wird. Zudem wird die Erwartungshaltung des Nutzers zufriedengestellt, da sich schließende und neu startende Fenster, als neue Anwendung wahrgenommen werden.

4.1.1.1 Benutzerinteraktionen

Der Benutzer wird durch Dialoge in der Anwendung geführt. Es steht jederzeit ein Schnellmenü zur Verfügung, mit dem das Auswahlfenster für Grammatiken oder das Auswahlfenster der Algorithmen erreicht werden kann. Zudem bietet das Schnellmenü einen Reiter „Hilfe“, welches Informationen zu den Algorithmen und der Handhabung der entsprechenden Fenstern bietet. Diese Dokumentation kann über den Hilfereiter ebenfalls geöffnet werden.
In den verschiedenen Szenen stehen dem Nutzer die Interaktionsmöglichkeiten per Maus und Schaltflächen zur Verfügung. Hier lag der Fokus auf einer einfachen und unkomplizierten Handhabung. Die Schaltflächen sind beschriftet mit den Aktionen, die sie ausführen. Wird ein Zustand erreicht, indem die Schaltflächen keine Funktion mehr bieten, so werden diese

deaktiviert und für den Nutzer ausgegraut. Dadurch wird die Anwendung für den Nutzer erwartungskonform.
Die Anwendung bietet dem Nutzer die Kontrolle, jederzeit die Geschwindigkeit selbst bestimmen zu können. So werden neue Szenen erst geladen, wenn der Nutzer die entsprechenden Schaltflächen betätigt. In den Szenen, die den Nutzer durch die einzelnen Schritte des Algorithmus leiten sollen, befinden sich zudem Steuerelemente für die Iterationen. Hier wurde bewusst auf eine manuelle und automatische Steuerungsmöglichkeit wert gelegt. Über die bekannten „Play/Pause“, „Next“ und „Previous“ Schaltflächen kann der Nutzer diese initiieren. Die Schaltflächen „Next“ und „Previous“ leiten dabei die nächste Iteration und die vorherige ein. Dadurch kann der Nutzer über manuelles betätigen der Schaltflächen die Geschwindigkeit bestimmen. Die „Play/Pause“ Schaltfläche aktiviert und deaktiviert den automatischen Modus für die nächste Iteration. Dabei wird in gewissen Zeitabständen zum nächsten Zustand iteriert. Die Zeitabstände können durch einen Schieberegler eingestellt werden. Der Nutzer hat durch alleiniges Ansehen der Schaltflächen eine genaue Vorstellung was diese tun. Die Erwartungskonformität, Steuerbarkeit und Selbstbeschreibungsfähigkeit der Anwendung werden damit erfüllt.
Die Szene der Chomsky Normalform besteht aus drei Knoten.

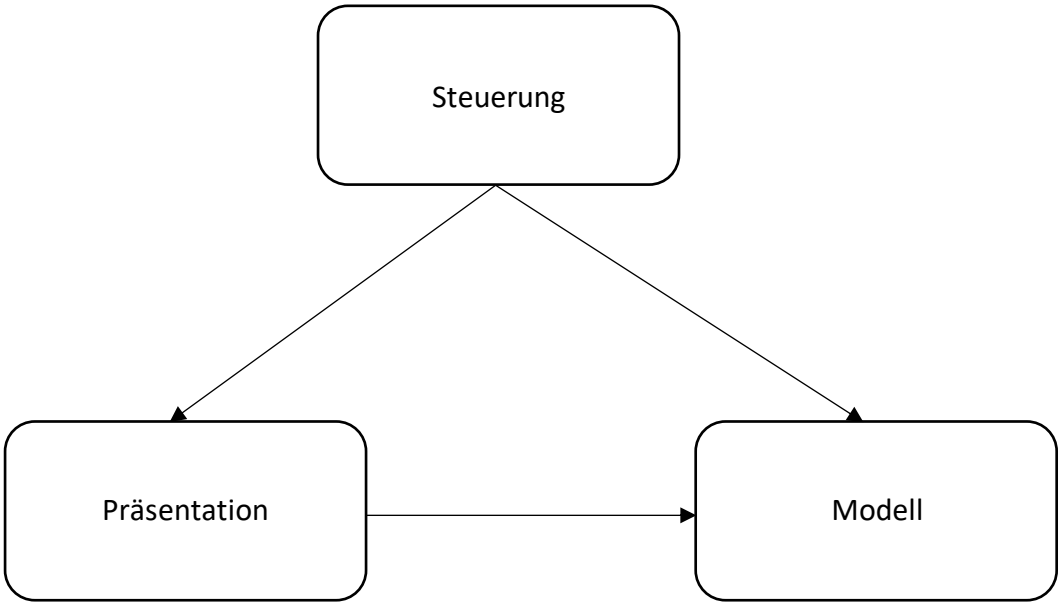


Grafik 7 zum Design für die CNF

Die beiden Zustandsknoten zeigen dem Nutzer in textbasiertem Format oder in grafischem Format die Grammatik an. Zwischen den beiden Modi kann per Klick, in das entsprechende Feld, gewechselt werden. Dabei zeigt der Knoten „aktueller Zustand“ die Änderungen in Echtzeit an der Grammatik. Der Sollzustandsknoten zeigt den Zustand an, der nach Anwenden der Regel erreicht werden soll. Die Informationsbox zeigt die aktuelle anzuwendende Regel, ihre Beschreibung und die Änderungen, die daraus resultieren, an.

4.1.1.2 Kommunikation

Entwurfsmuster der Softwaretechnik geben Pläne vor, mit denen man wiederkehrende Probleme lösen bzw. umsetzen kann. Anwendungen wie diese setzen häufig auf das MVC Paradigma. Hierbei wird die Anwendung in drei Komponenten implementiert. Das Modell, die Präsentation und die Steuerung.



Grafik 8 zum Aufbau des MVC Paradigma

Je nach Anwendungsfall können jedoch Änderungen an der Umsetzung vorgenommen werden. In dieser Anwendung wurde bewusst auf das MVC Paradigma mit dem Observer Pattern verzichtet. Nach diesem Pattern werden Interaktionen von der Präsentationskomponente an das Modell gereicht. Dieses nimmt dementsprechend Änderungen an den Daten hervor und informiert die Komponenten, die sich am Observer dafür angemeldet haben. Da in diesem Anwendungsfall mehrere Szenen vorhanden sind, in denen Daten gepflegt werden, hätte man mit diesem Pattern mehrere Zustände pflegen müssen. Da die einzelnen Szenen auch noch in mehrere Knoten unterteilt sind, hätte man für die Repräsentation auf dem Bildschirm eine Logik für die Präsentationskomponente implementieren müssen. Der Grund hierfür ist die unterschiedliche Darstellung der Daten in der Modellkomponente und der Präsentationskomponente.
Die Präsentationskomponente, ist mit der aktuellen Umsetzung nur für die Präsentation und Interaktion zuständig. Werden vom Nutzer Eingaben getroffen, so werden die entsprechenden Funktionen der Steuerungskomponente darüber benachrichtigt und die richtigen Knoten mit übergeben. Anschließend steuert die Steuerungskomponente die richtigen Funktionen der Modellkomponente an und übergibt diesem das Knotenelement. Daraufhin folgt die Berechnung des neuen Zustands. Die neuen Daten werden für die Repräsentation überführt und in den Knoten geschrieben. Diese Umwandlung geschieht, um die aktuellen Daten den Nutzer lesbarer zu machen. Letztlich wird die Kontrolle wieder zurückgegeben und auf neue Eingaben durch den Nutzer gewartet.

4.1.2 Modell

Das Modell pflegt und verwaltet die Daten. In dieser Komponente ist die Geschäftslogik implementiert. Für diese Anwendung kann die Logik in folgende Kategorien eingeteilt werden:

4.1.2.1 Daten

Bei der Fragestellung, wie die Daten gepflegt werden sollen, wurden die Aspekte der Einfachheit für den Nutzer, sowie die Wiederverwendbarkeit in Betracht gezogen. Mit Einfachheit ist hier die Bedienung der Anwendung gemeint. Dazu zählt die Vermeidung der Tastatureingaben, da die Eingabe einer Grammatik bestimmten Voraussetzungen folgt. Mit der Wiederverwendbarkeit wurde das Ziel gesetzt, die Grammatik auszulagern und bei einer Wiederverwendung erneut einzulesen. Um diese Ziele zu erreichen wurde auf die XML Technologie gesetzt. Diese ermöglicht es Daten in einem textbasierten Format abzuspeichern und mit Hilfe von Tags zu definieren. Der große Vorteil ist, dass XML Dateien erweiterbar sind und im Vergleich zu binären Verfahren immer dieselbe Bedeutung tragen. Durch ständige Erweiterung und Wartung der Anwendungen können sich Funktionen in ihrer Arbeitsweise ändern. Bei binären Datenformaten tritt häufig das Problem auf, dass Anwendungen in späteren Versionen die Daten anders interpretieren.
Ein weiteres Ziel war es, dem Nutzer die Möglichkeit zu geben, in beiden Algorithmen schrittweise vor oder zurück zu iterieren. Würde man dafür für jede Iteration den neuen Zustand berechnen, so müsste man um einen Schritt zurück zu iterieren, entweder alles von neuem Berechnen oder die einzelnen Zustände zwischenspeichern. In beiden Fällen erreicht man Performance- oder Speicheroverhead. Dies möchte man vermeiden. Daher wurde die Erweiterbarkeit der XML Technologie von Nutzen gemacht. Bei einer neuen Eingabe werden die Daten, in der definierten Struktur, in eine vom Nutzer definierte Datei geschrieben. Anschließend wird die Grammatik in Chomsky Normalform überführt. Alle Änderungen, die dabei gemacht werden, werden kategorisiert in dieser Datei abgelegt. Dadurch werden die Berechnungen nur einmal beim Anlegen der Datei ausgeführt und bei jedem weiteren Zugriff aus der Datei gelesen. Dies ermöglicht auch die direkte Ausführung des CYK Algorithmus, nachdem die Datei eingelesen wurde. Bei jeglichen Iterationen können jeweils die Daten vom vorherigen Zustand eingelesen und angewendet werden.
Startet man den CYK Algorithmus, so wird direkt die Grammatik in der Chomsky Normalform ausgelesen und angewendet.

4.1.2.2 Definition der Struktur des XML Dokumentes

$< grammatik >$
$< name > Name\ der\ Regel < /name >$
$< n > Nichtterminalsymbole < /n >$
$< t > Terminalsymbole < /t >$
$< s > Startsymbol < /s >$
$< p >$
$< zeile >$
$< zelle > Linke\ Seite\ der\ Ableitung < /zelle >$
$< zelle > Ableitung\ 1 < /zelle >$
$< zelle > Ableitung\ 2 < /zelle >$
$\ldots weitere\ Ableitungen$
$< zelle > Ableitung\ n < /zelle >$
$< /zeile >$
$\ldots weitere\ Zeilen$
$< /p >$
$< change > Änderungen < /change >$
$\ldots restliche\ Regeln$
$< /grammatik >$

Nach diesem Schema werden die Daten gespeichert. Das Wurzelelement der XML Datei ist der Tag „Grammatik“. Darunter befindet sich die Eingabegrammatik, sowie alle Iterationen aus den einzelnen Regeln. Diese werden mit dem Tag „Name“ definiert. Darunter befinden sich die Nichtterminalsymbole, die Terminalsymbole und das Startsymbol. Da es sich bei den Nichtterminal- und Terminalsymbolen um eine Aufzählung handelt, werden diese per Semikolon voneinander getrennt. Das Startsymbol muss in den Nichtterminalsymbolen vorhanden sein. Unter dem Tag „p“ werden alle Produktionen gespeichert. Dies geschieht zeilenweise, genauso wie die Eingabe des Nutzers und die textbasierte Repräsentation auf dem Bildschirm. Dabei stellt die erste Zelle immer die Linke Seite der Ableitung dar. Alle darauffolgenden Zellen entsprechen einer Ableitungsregel. Hierbei werden die Symbole durch ein Leerzeichen getrennt abgespeichert. Nach den Produktionsregeln kommen die Änderungen. Diese liegen alle in einem Tag, semikolongetrennt vor. Zu Beginn jeden Schrittes werden die Stellen in der Grammatik markiert, die die Regel verletzen. Dadurch kann man die einzelnen Schritte voneinander trennen und dem Nutzer die betroffenen Stellen farblich markieren. Um eine Iteration vor zu gehen, muss man demnach nur zur nächsten Markierung iterieren. Analog gilt dasselbe für eine vorhergehende Iteration. Gibt es keine Markierung mehr in dem Tag, so ist das Ende der Regel erreicht und man kann zur nächsten übergehen. Durch den schrittweise agierenden Algorithmus, können die Änderungen während sie geschehen abgespeichert und in der XML Datei persistiert werden. Es müssen keine neuen Berechnungen für die einzelnen Schritte gemacht werden.

4.1.2.3 Repräsentation der Daten in der Anwendung

Die Grammatik wird innerhalb der Anwendung in der Klasse „Tree“ gespeichert und verwaltet. Die Klasse liest die ihr übergebene Eingabe ein und baut aus dieser die Grammatik nach. Dabei speichert sie alle linken Seiten der Ableitungsregeln in einer HashMap<String,Elem> ab, um einen schnellen Zugriff auf diese zu gewährleisten. Somit kann mit Hilfe des Namens direkt auf das zugehörige Element geschlossen und zugegriffen werden. Die Klasse „Elem“ selbst stellt die Elemente dar, die eindeutig sind und nicht mehrmals vorkommen. Ein Element hat einen Namen, Typen und eine Liste, die mit Nodes befüllt wird. Es wird zwischen den Typen Nichtterminal und Terminal unterschieden. Die Klasse Nodes wiederum stellt die Knoten einer einzigen Ableitungsregel dar. Da Nichtterminalsymbole mehrere Ableitungsregeln haben können, wurden die Nodes eingeführt um zwischen diesen eine Abgrenzung zu schaffen. Terminalsymbole haben natürlicherweise keine Liste an Nodes, da diese nicht weiter ableitbar sind.
Auf diese Art und Weise der Umsetzung lässt sich die gesamte Grammatik, ausgehend von dem Startsymbol erreichen und in einer Baumdarstellung anzeigen. Es kann jedoch in früheren Iterationsstufen der Grammatik ein Zyklus auftreten. Daher werden die Elemente der Nichtterminalsymbole beim Erstellen der Baumstruktur in einer temporären Liste gespeichert. Falls ein erneuter Zugriff auf diese kommt, so werden lediglich die Referenzen übergeben und das Element landet nicht wieder im Prozess zur Verarbeitung.
So werden auch in der grafischen Darstellung der Grammatik, die Knoten eingezeichnet. Das heißt, dass ein zuvor eingezeichnetes Element bei erneutem auftreten an den Baum angehängt, aber ihre Blätter nicht weiter eingezeichnet werden.

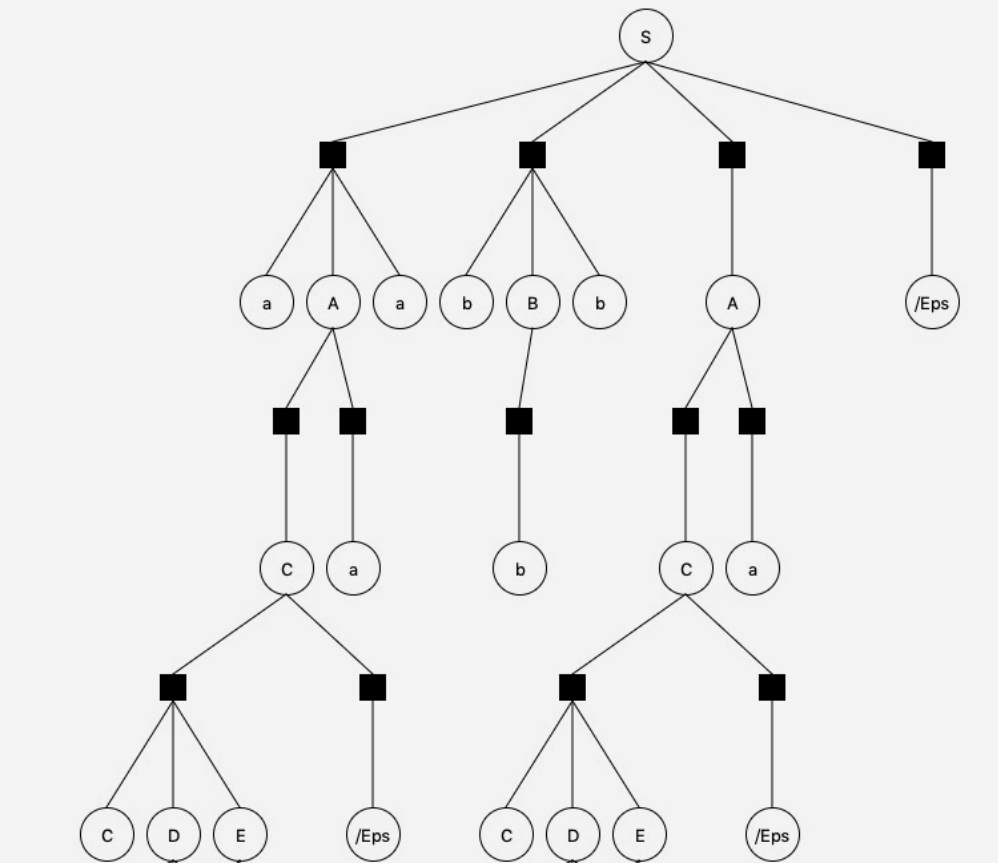
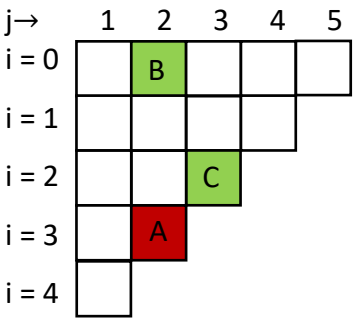


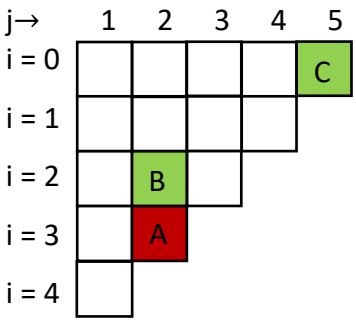
Bild 1 Screenshot zur graphischen Darstellung

Der CYK Algorithmus wird ähnlich zur Chomsky Normalform umgesetzt. Sie besitzt eine HashMap auf die man mit Ganzzahlen zugreifen kann. Die Ganzzahlen stellen die Ebenen bzw. Zeilen der Treppenform dar. Es wäre auch möglich gewesen hierzu eine Listenstruktur zu wählen, doch diese bieten keinen wahlfreien Zugriff an. Eine Iteration über alle Felder ist nötig, um zum letzten Element in der Liste zu gelangen. Dies entspricht einer worst-case Laufzeit von $O(n)$, während eine HashMap einen Zugriff in $O(1)$ gewährt. Über diese erhält man dann Zugriff auf eine Liste, die mit den Zellen befüllt ist. In den Zellen stehen dann die einzelnen Symbole. Im Vergleich zur Chomsky Normalform stehen hier die Ergebnisse nicht vorher fest, sondern werden zur Laufzeit berechnet. Dies geschieht nach der Eingabe des zu prüfenden Wortes. Der CYK Algorithmus hält intern Zähler für die Zeile, die Spalte und einen weiteren Zähler für die Einträge in dieser Zelle. Besitzt die Zelle drei Feldpaarkombinationen, so ist die innerste Liste von der Länge genauso lang. In jedem Listenabschnitt stehen die Ergebnisse der entsprechenden Feldpaare. Dadurch können Nichtterminalsymbole mehrfach auftreten, weil sie auf mehrere Feldpaarkombinationen abbilden können. Aus diesem Grund wird bei jedem Schreibzugriff in die entsprechende Zelle nochmal geprüft, ob die Rückgabewerte der Listenstruktur noch nicht vorhanden sind. Sind diese jedoch vorhanden, so werden sie zu den aktuellen Eingaben nicht hinzugefügt. Wird ein Schritt zum vorherigen Zustand initiiert, so wird der Counter und die restlichen Zähler angepasst. Dann wird ein Ergebnis zusammengesetzt, dass alle Nichtterminalsymbole, beginnend bei dem Listenindex null bis zum verminderten Zähler enthält. Mehrfache Vorkommen werden dabei nicht hinzugefügt. Dies ist durch die Nutzung von einem HashSet, ohne Mehraufwand zu bewerkstelligen, da HashSets maximal nur ein Vorkommen der Objekte erlauben. Dieses Ergebnis wird dann in die Zelle geschrieben. Das Zwischenspeichern der Ergebnisse ermöglicht einen schnelleren Zugriff auf die Werte, ohne die Ergebnisse von neuem zu Berechnen.
Es kann auftreten, dass ein Nichtterminalsymbol innerhalb einer Zelle durch mehrere Feldpaarkombinationen als Ergebnis in Frage kommt. In diesem Fall muss nicht gesondert ein Zähler für dieses eingesetzt werden, da die innerste Liste diese Vorkommen auch mehrmals in seinen Zellen ablegt. Beim Auslesen in beide Richtungen wird dieses Symbol rekonstruiert und angezeigt.

Beispiel:

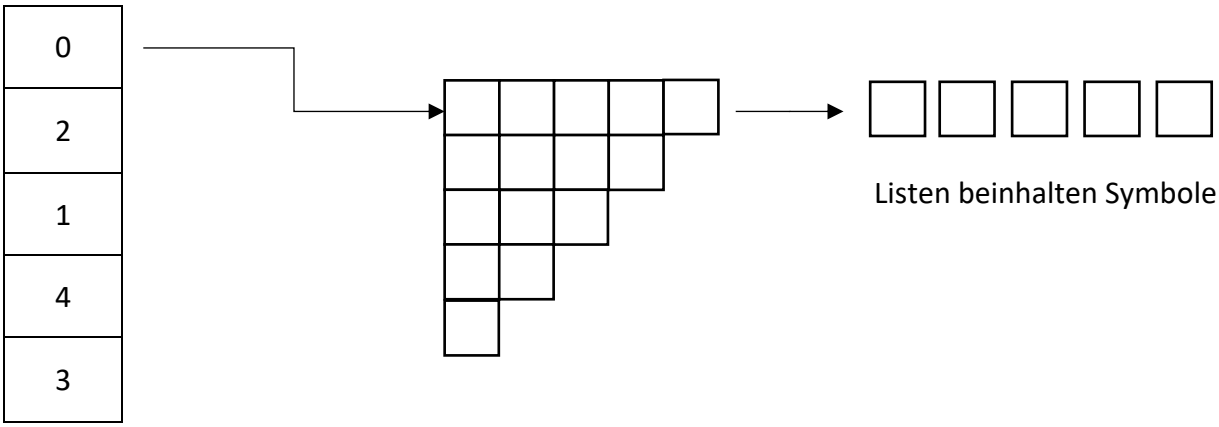
Sei $A \rightarrow BC$ eine Ableitungsregel und das Feld $Z_{3,2}$ hätte den Eintrag A aus folgenden zwei Kombinationen erhalten:





Grafiken 9.1 und 9.2 zur visuellen Darstellung des Problems

Darstellung der Daten für den CYK Algorithmus:



Grafik 10 zur internen Repräsentation der CYK Daten

4.1.2.4 Berechnung der Daten

Der Algorithmus, der die Chomsky Normalform berechnet, arbeitet ausgehend von dem Startsymbol alle erreichbaren Symbole ab. Dabei werden alle vier Regeln separat und nacheinander abgearbeitet. Beginnend mit der Regel werden Ableitungsregeln gelöst, die Nichtterminalsymbole und Terminalsymbole gleichzeitig beinhielten. Anschließend werden alle erreichbaren Symbole erneut abgetastet und die Ableitungsregeln mit einer Länge größer zwei werden gekürzt. Im Anschluss werden Epsilon Übergänge aufgelöst. Hierzu werden dann alle Symbole herausgesucht, die von diesem Epsilon Übergang betroffen sind. Letzten Endes werden die Kettenregeln aufgelöst. Hierzu wird erneut über alle Symbole iteriert und gleichzeitig die besuchten festgehalten. Dazu wird eine Tiefensuche angewendet. Tritt ein Nichtterminalsymbol mehrmals auf, so muss ein Zyklus vorhanden sein. Anschließend wird von diesem Nichtterminalsymbol ausgehend erneut dessen Kindelemente abgesucht, bis man an demselben Symbol ist. Dadurch erfasst man alle Symbole, die an dem Zyklus beteiligt sind. Ein Zyklus kann nicht nach einem beliebigen Element aufgelöst werden. Es muss das Wurzelelement aus dieser Menge extrahiert werden. Dazu beginnt man eine Breitensuche, beginnend bei dem Startsymbol. Das erste gefundene Element aus der Liste der Zyklenelemente, ist das Wurzelelement.

Der CYK Algorithmus erhält ein Wort und eine Grammatik in der Chomsky Normalform. Es bietet die Funktionen zur Iteration an. Dabei werden intern Zähler verwendet um auf die einzelnen Zellen zuzugreifen. Je nach Zeile müssen verschiedene Ableitungen gesucht werden. Für die erste Zeile gilt, dass nur Nichtterminalsymbole benötigt werden, die auf ein Terminalsymbol abbilden. In der zweiten Zeile müssen nur auf die darunterliegenden Zellen und ihre benachbarte Zelle ableitbar sein. Für alle anderen Zeilen gilt es, alle Kombinationen zu prüfen. Auch dazu werden intern Zähler verwendet, die je nach Iterationsaufruf inkrementell oder dekrementell gezählt werden.

5. Installationsanleitung

5.1 Java

Die neueste Java Runtime Environment (JRE) kann von der offiziellen Webseite https://java.com/de/download/ heruntergeladen werden. Diese beinhaltet unter anderem die Java Virtual Machine (JVM), die benötigt wird, um die Anwendung zu starten. Dazu muss man lediglich auf die Schaltfläche „Kostenloser Java-Download“ klicken.

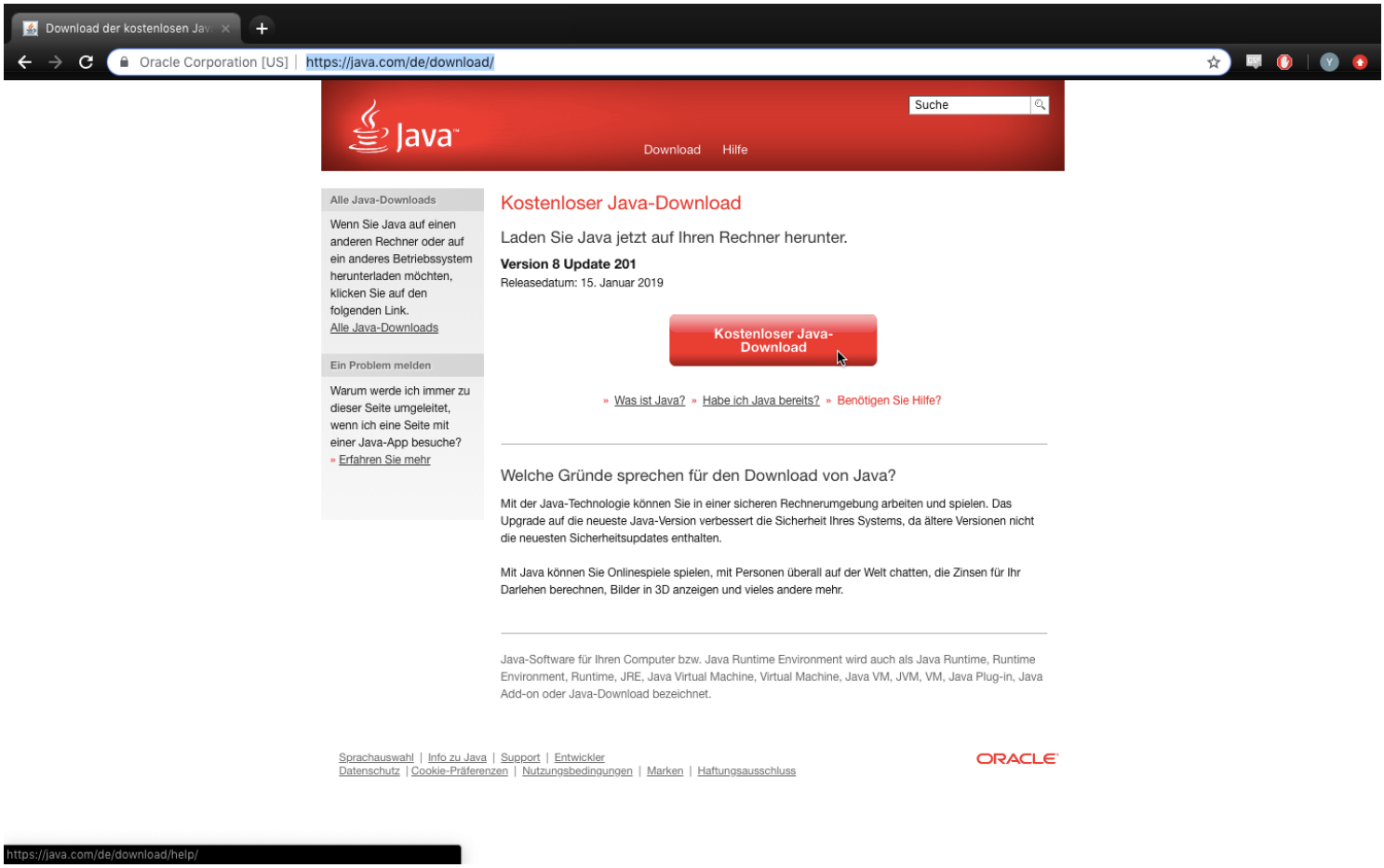


Bild 2 Screenshot Java Download

Im darauffolgenden Fenster wird die richtige Java Version automatisch ausgewählt und zum Runterladen angeboten. Die Endbenutzerlizenzvereinbarung müssen akzeptiert werden. Anschließend fängt der Download automatisch an.

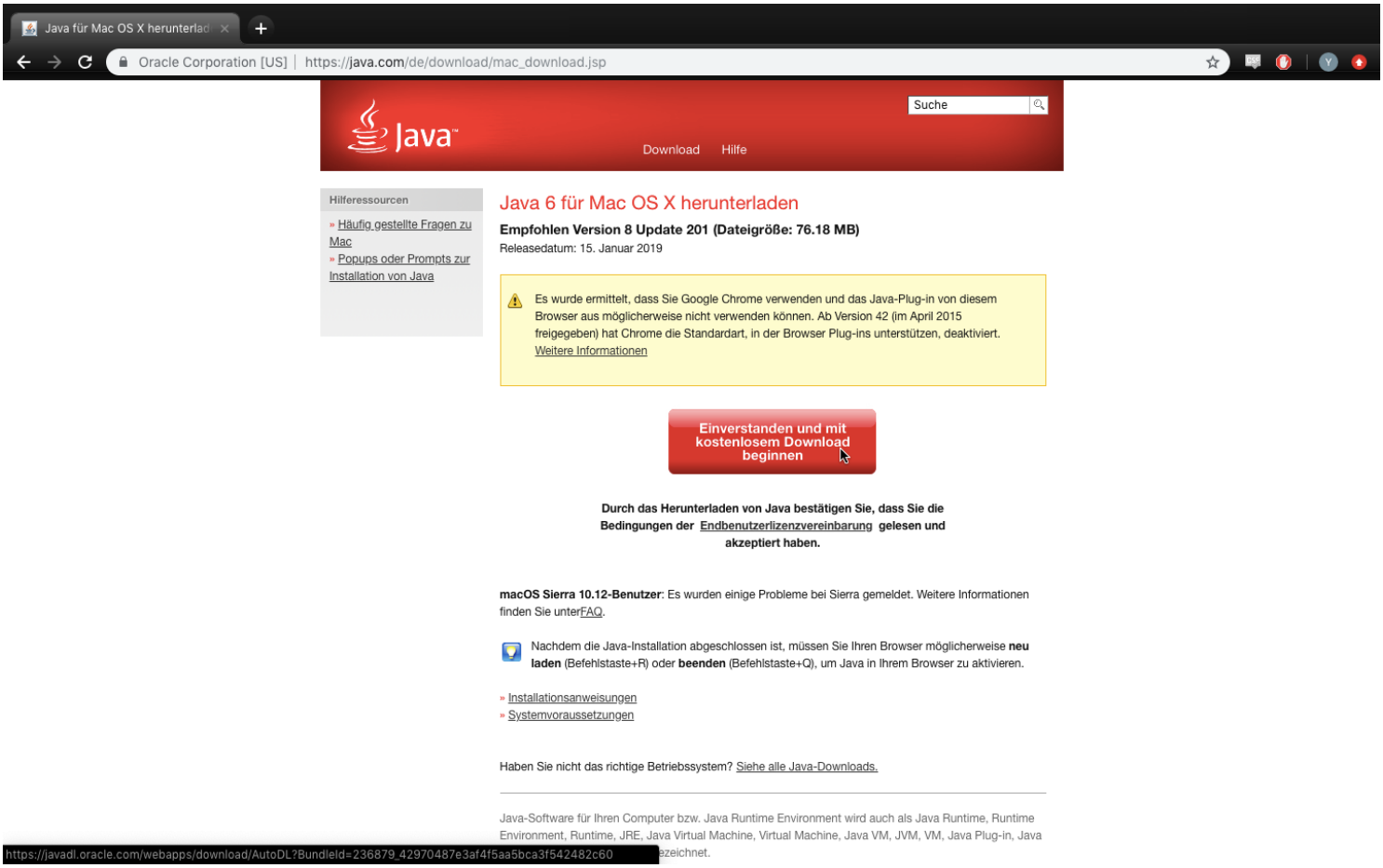


Bild 3 Screenshot Endnutzerlizenzvereinbarungen bei Java Download

Ist die aktuelle Java Version heruntergeladen, so muss diese installiert werden. Je nach Betriebssystem kann das Installationsfenster anders aussehen. Dazu wird man von den entsprechenden Dialogfenster geleitet. Es müssen keine besonderen Einstellungen beim Installieren vorgenommen werden.

Wichtig: Bei manchen Windows Rechnern, können nach der Installation von Java, keine Java Programme gestartet werden. Der Grund hierfür ist, dass die JRE nicht in den Umgebungsvariablen steht und Windows diese bei der Ausführung nicht finden kann. Dieser Fehler tritt häufig auf, wenn die Java-Installation außerhalb des standardmäßigen Ordners erfolgt.
Abhilfe:

1. Systemsteuerung öffnen
2. Erweiterte Systemeinstellungen wählen
3. Umgebungsvariablen wählen und Abschnitt Systemvariablen öffnen
4. Umgebungsvariable „PATH“ auswählen und bearbeiten anklicken
5. Im Variablenwertefeld den Pfad zum Installationsverzeichnis von Java anfügen
   Beispielsweise: C:\ProgramData\Oracle\Java\javapath;

5.1.1 Installation auf MacOS

Bild 4 Screenshot Installation von Java auf MacOS

Bild 5.1 Screenshot Installation auf MacOS

Bild 5.2 Screenshot Installation auf MacOS

5.1.2 Installation auf Windows

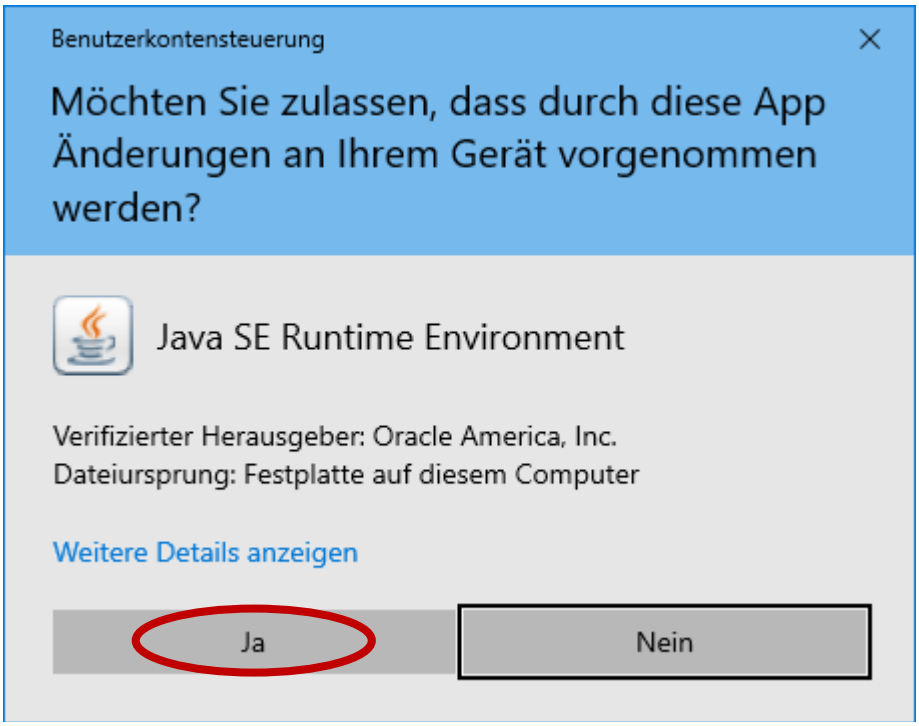


Bild 6.1 Screenshot Installation auf Windows

Bild 6.2 Screenshot Installation auf Windows

Bild 6.3 Screenshot Installation auf Windows

5.2 Anwendung (Projektarbeit)

Die Projektarbeit wurde in Java geschrieben und mit Hilfe von GitHub versioniert. Es wurde in der Entwicklung in kleinere Aufgabenbereiche unterteilt, die dann Funktion für Funktion abgearbeitet wurden. Die neuste Version des Projektes, sowie die Beispielgrammatik können auf GitHub gefunden und heruntergeladen werden. Dazu kann folgender Link angewählt werden, der direkt auf die Release-Seite des Projektes verweist. https://github.com/YenerOezsoy/CYK/releases

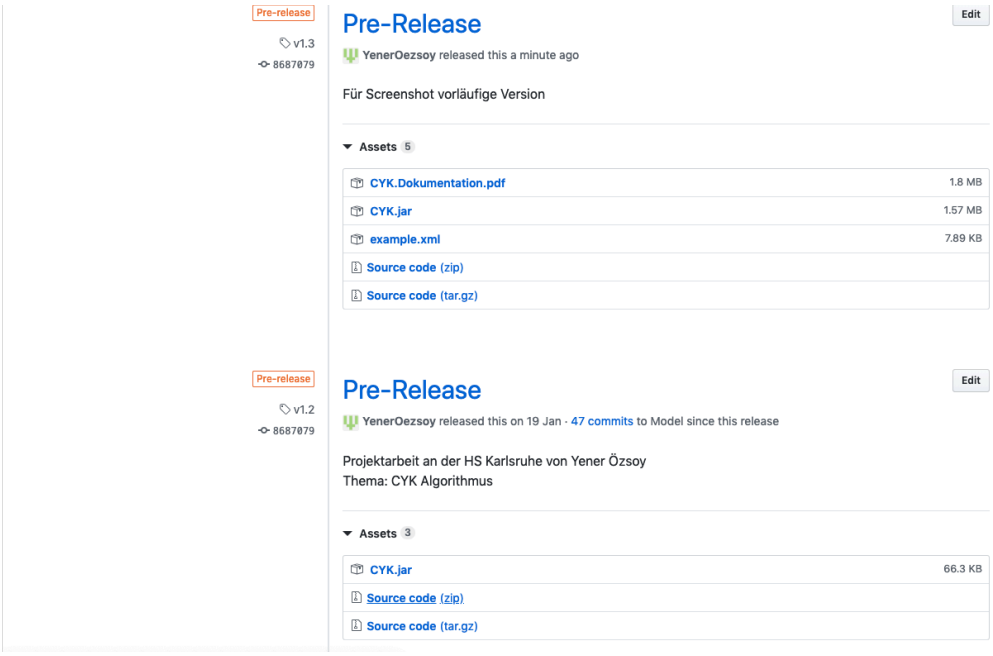


Bild 7 Screenshot Release Seite des Projekts

Klickt man hier auf die Jar-Datei, so wird die Anwendung direkt heruntergeladen. Ist Java schon installiert, so kann diese sogar direkt per Doppelklick gestartet werden. Einmal gestartet, stehen zwei Schaltflächen zur Verfügung. Mit diesen kann man eine Grammatik laden oder eine neue Grammatik definieren.

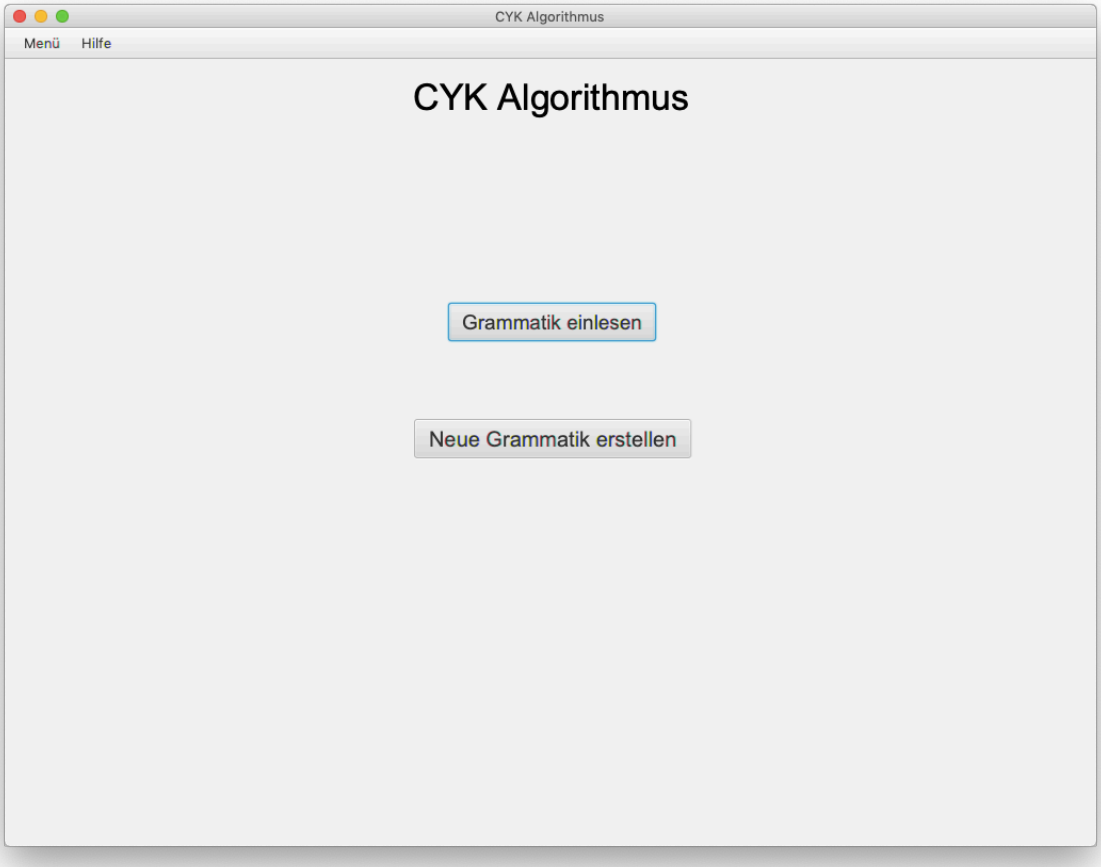


Bild 8 Screenshot Anwendung: Auswahlfenster der Grammatik

Auf GitHub steht die Beispielgrammatik, die wir in den vorherigen Kapiteln genutzt haben, zum Download bereit. Diese lesen wir in diesem Beispiel ein, indem die Schaltfläche „Grammatik einlesen“ angewählt wird. In der darauffolgenden Szene kann zwischen den beiden Algorithmen gewählt werden. In der Algorithmenansicht kann zwischen den beiden Algorithmen gewechselt werden. Zusätzlich bietet das Menü die Navigiermöglichkeit zur Startszene und zur Algorithmenauswahl.

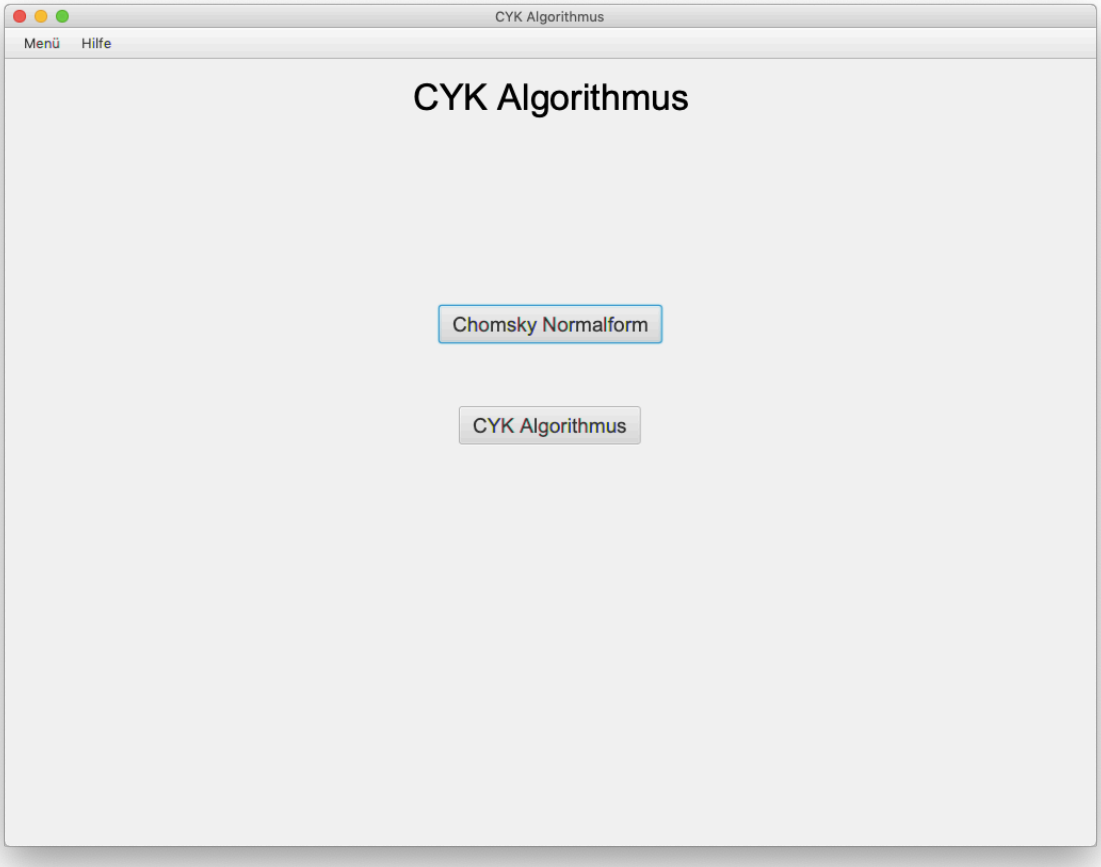


Bild 9 Screenshot Anwendung: Auswahlfenster der Algorithmen

Die Chomsky Normalform wird in der textuellen Ansicht folgendermaßen dargestellt.

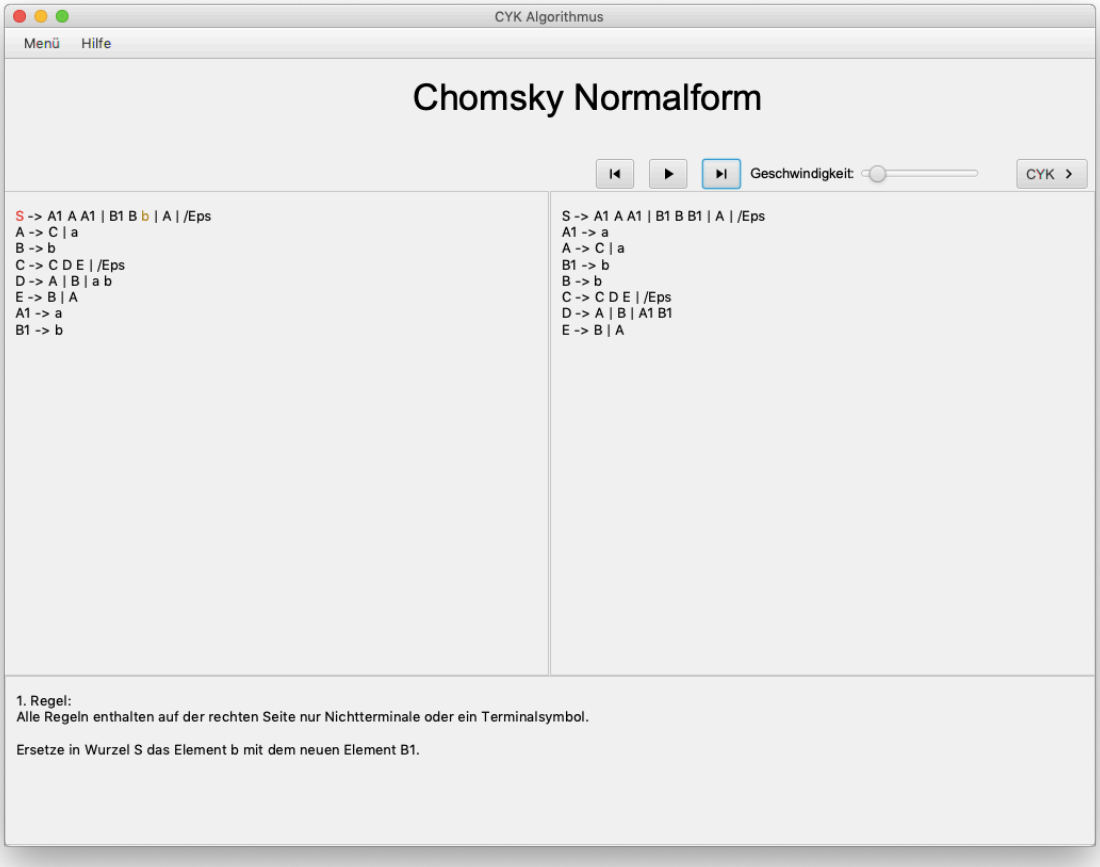


Bild 10.1 Screenshot Anwendung: Textuelle Darstellung der CNF

Mit einem Klick in eine beliebige Position kann die Ansicht zur grafischen Darstellung geändert werden. Beide Darstellungen lassen sich separat ändern.

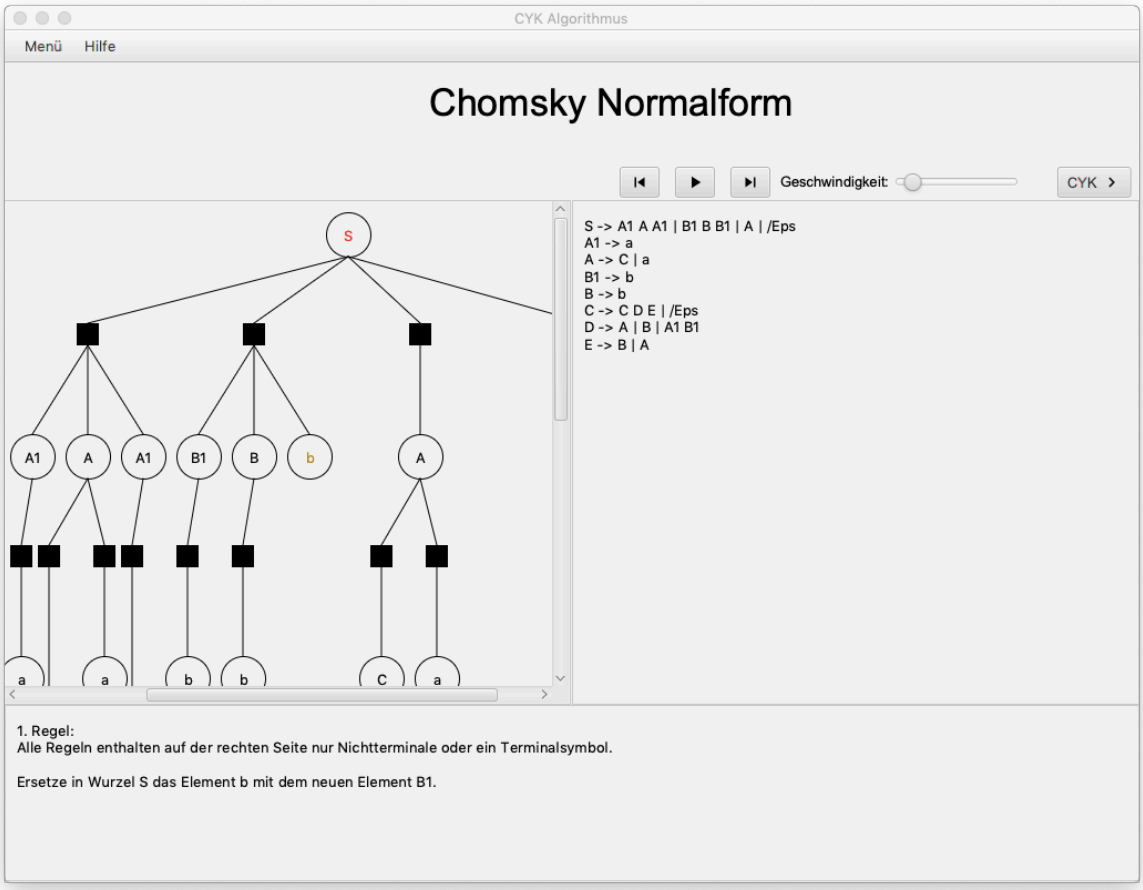


Bild 10.2 Screenshot Anwendung: Graphische Darstellung der CNF

Die Anordnung der Knoten kann per Drag and Drop verschoben werden. Die von der Regel betroffenen Elemente werden in beiden Ansichten farblich hervorgehoben.

Die Szene des CYK Algorithmus bietet nur eine grafische Darstellung an. Es wird die aktuell betrachtete Zelle mit einem Gelbton hervorgehoben. Die Zellen, die die Teilworte darstellen, werden mit einem Rotton markiert.

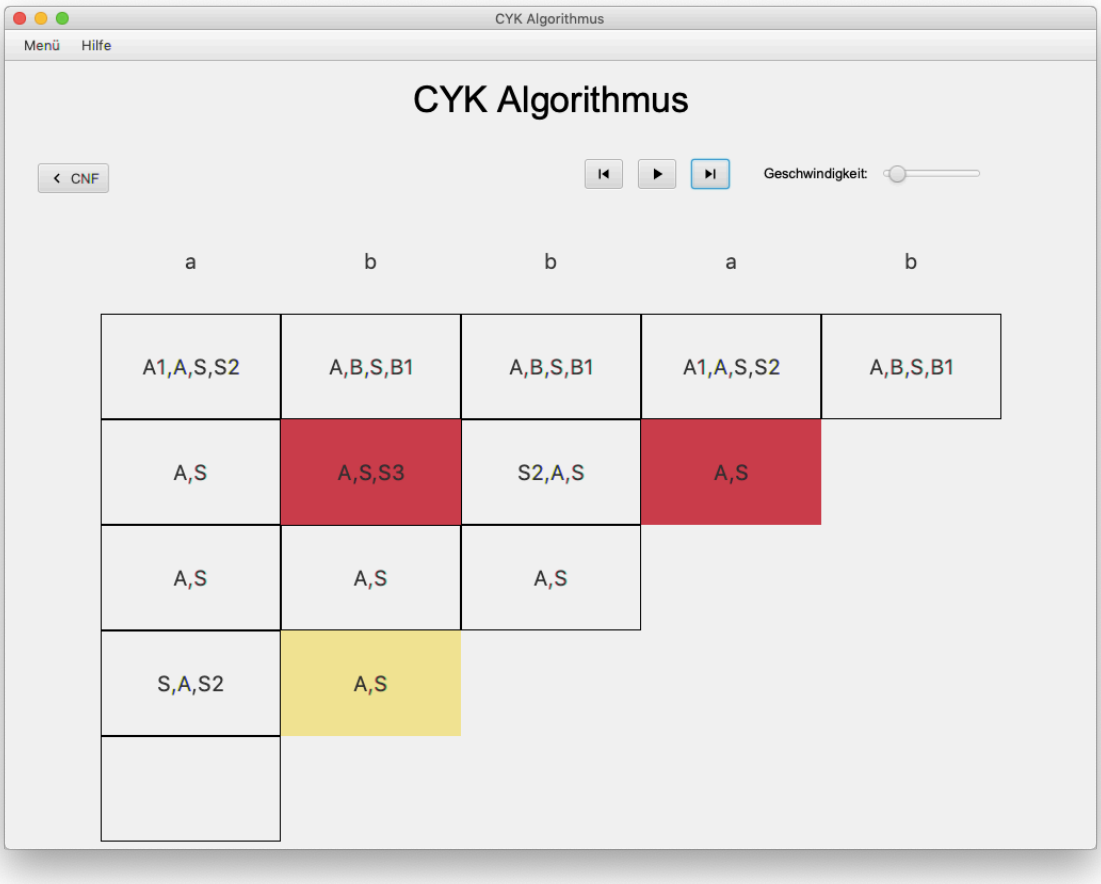


Bild 11 Screenshot Anwendung: Darstellung des CYK Algorithmus anhand eines Beispiels

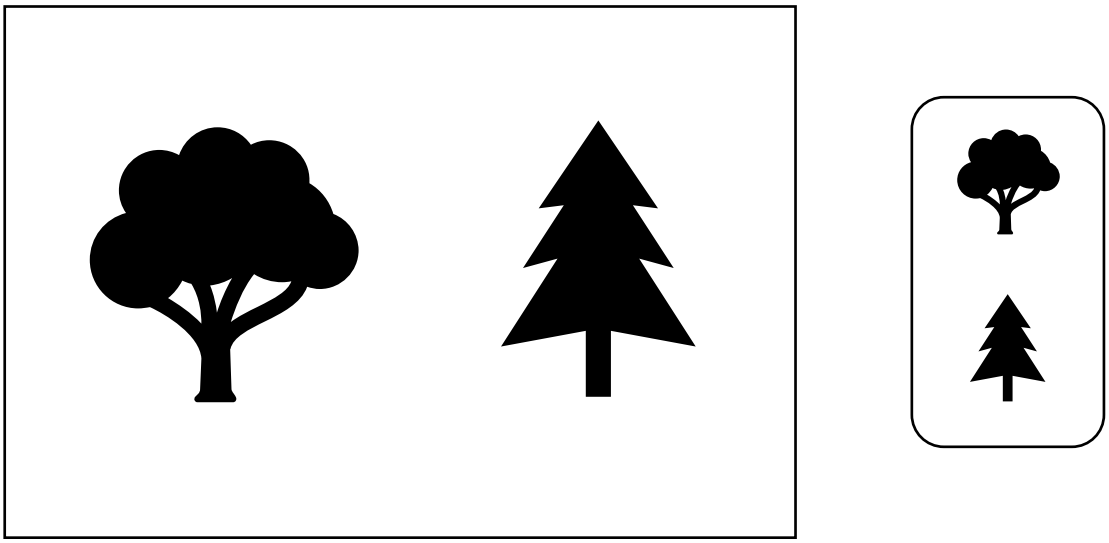
6. Zukunftsaussichten

6.1 Plattform

Ein großer Vorteil der Java Technologie ist die Plattformunabhängigkeit. Auf jedem Gerät, dass die Java Runtime Environment unterstützt, kann ein durch kompilieren erzeugter Bytecode ausgeführt werden. Dieser muss nicht für die darunterliegende Hardware spezifisch in Maschinencode umgewandelt werden. Um die Anwendung auf möglichst vielen Geräten zu verteilen, muss diese immer erneut heruntergeladen werden und Java muss auf diesem Gerät installiert sein. Das kann für manche Nutzer lästig wirken, wenn sie lediglich Kleinigkeiten testen wollen. Eine portable Online Lösung wäre hier von Vorteil. Durch Zugriff auf eine Webseite könnten Nutzer immer direkt auf die neuste Version zugreifen, ohne diese installieren zu müssen. Das ein- und auslesen der Dateien wäre in diesem Szenario kein Problem. Java wird von einigen Browsern unterstützt, was die Portierung der Software ermöglicht. Doch dafür muss Java im Browser installiert werden. Um dies zu umgehen, muss man eine vom Browser nativ unterstütze Programmiersprache wählen. Hier bietet sich ein Zusammenspiel von JavaScript und HTML an. Mit HTML können die einzelnen Szenen beschrieben und durch JavaScript gefüllt bzw. zur Laufzeit der Interaktionen manipuliert und angepasst werden. JavaScript würde die Kommunikation zwischen Server und Host übernehmen, die erhaltenen Daten auswerten und in das HTML Dokument einpflegen. Da JavaScript nicht Serverseitig, sondern Clientseitig läuft, entsteht für den Server keine zusätzliche Berechnung der Darstellung. Diese läuft für jeden Benutzer am eigenen Gerät. Auf dem Server könnte die Logik implementiert bzw. die Berechnung der Algorithmen übernommen werden. Die Eingaben, die von Clients kommen, würden damit auf dem Server ausgeführt werden. Die fertig berechneten XML Dateien könnten den Nutzern ohne Probleme zurückgesendet werden. Die Auswertung dieser würde auf dem Clientgerät in JavaScript geschehen. Es wäre eine portable Online Lösung, die auf jedem modernen Gerät laufen würde.

6.2 Graphical User Interface

Die GUI (Graphical User Interface) hat eine statische Größe von 600 x 400 Pixeln. Dementsprechend sind die einzelnen Artefakte auch mit einer festen Position und Größe in dem Fenster angeordnet. Die Entscheidung für eine statische, feste Größe wurde aufgrund von Design technischen Gründen getroffen. Würde das Fenster den gesamten Bildschirm füllen bzw. größere Fenstermaße annehmen, so wäre in der textuellen Darstellung der Grammatik ein Großteil der Anwendung leer.
Zudem müssten unter anderem die Objekte im Fenster neu positioniert und ausgerichtet werden. Da anfangs die Funktionalität der Anwendung im Vordergrund stand, wurde der Kompromiss der statischen Größe eingegangen. In weiteren Iterationsstufen der Entwicklung kann die Anwendung zu einem sogenannten Responsive Design übergehen. Dabei passen sich Fenster, sowie die darin befindlichen Objekte der Größe des Fensters an und positionieren sich nach festgelegten Regeln. Diese Art von Design wird häufig bei plattformunabhängigen Anwendungen eingesetzt wie zum Beispiel auf Internetseiten. Die Anwendung bzw. die Internetseite kann mit Hilfe von Systemmethoden auf die Größe des Fensters oder des Bildschirms zugreifen und sich dementsprechend anpassen. Bei größeren Fenstermaßen können mehr Objekte nebeneinander angezeigt werden, während sie bei kleineren Bildschirmen häufig untereinander angeordnet werden.
Im Zuge der Übertragung des Systems auf eine Internetdienstleistung, kann das Responsive Design mit Hilfe von HTML beschrieben und umgesetzt werden. Falls dieser Entwicklungsschritt nicht angegangen werden sollte, so kann das Responsive Design auch mit Hilfe von Java implementiert werden. Dafür können sogenannte Listener eingesetzt werden, die auf Veränderungen des Fensters in Bezug auf Höhe und Weite reagieren können. Je nachdem welche Maße das Fenster nach der Größe hätte, könnte ein neues Interface die Größe auswerten und das jeweilig betroffene Design an den Controller weiterleiten, welches das Fenster befüllt. Jedes Design müsste hierzu feste Regeln zur Anordnung der Objekte haben. Für die bessere Wartbarkeit wäre eine Kapselung dieser Regeln von Vorteil, da es im Laufe der Iterationen zu Änderungen der Objekte kommen kann. Dementsprechend könnten sich überschneidende Designregeln mit einer Änderung auf allen Designs angepasst werden.



Grafik 11 zur Darstellung der Elemente anhand des Responsive Designs

7. Zusatzmaterial

7.1 Grammatik

$S \rightarrow A \mid a\ A\ a \mid b\ B\ b \mid \varepsilon$
$A \rightarrow C \mid a$
$B \rightarrow b$
$C \rightarrow C\ D\ E \mid \varepsilon$
$D \rightarrow A \mid B \mid a\ b$
$E \rightarrow B \mid A$

7.2 Lemma 1

Sei G eine kontextfreie Grammatik mit $G = (V, \Sigma, P, S)$ ohne Kettenregeln, für die Sprache L und alle Produktionen seien wie folgt:

$$A \rightarrow a, a \in \Sigma$$
$$A \rightarrow X, X \in (V \cup \Sigma)^+, |X| \geq 2$$

Für jedes $a \in \Sigma$ fügt man nun ein neues Nichtterminalsymbol Y_a hinzu mit folgender Produktionsregel: $Y_a \rightarrow a$. Jedes Vorkommen auf der rechten Seite der Produktionsregel von a wird durch das neue Nichtterminalsymbol ersetzt.
Daraus folgt, dass alle Produktionsregeln folgende Form haben:

$$A \rightarrow a, a \in \Sigma$$
$$A \rightarrow B_1 B_2 \dots B_k, k \geq 2$$

Damit diese Grammatik in Chomsky Normalform ist, müssen die Produktionsregeln eine Länge von zwei nicht überschreiten, falls diese auf Nichtterminalsymbole abbilden. Hierzu führen wir weitere Variablen $C_1, C_2, \dots, C_{k-1}$ hinzu. Die Produktionsregeln wird dann nach diesem Schema umgeformt:

$$\begin{aligned} A &\rightarrow B_1 C_1 \\ C_1 &\rightarrow B_2 C_2 \\ &\vdots \\ C_{k-2} &\rightarrow B_{k-2} C_{k-1} \\ C_{k-1} &\rightarrow B_{k-1} B_k \end{aligned}$$

Es ist leicht zu sehen, dass sich die Sprache durch die Umformung nicht ändert.

7.3 Landau Notation

Die Landau Notation, oder auch O Notation genannt, wird in der Mathematik und Informatik für die Beschreibung von asymptotischen Folgen und Funktionen verwendet. Dabei dient die Notation als Maß für die Rechenschritte in Relation zu der Eingabegröße. Mit der Landau Notation lassen sich in der Komplexitätstheorie, Algorithmen und ihr Aufwand bemessen. Dadurch ergibt sich die Möglichkeit, unter anderem Algorithmen zu vergleichen, die sich mit derselben Problemstellung beschäftigen.

8. Bild- und Grafikverweis

8.1 Grafikverweis

Grafik 1: selbst erstellt
Grafik 2: selbst erstellt
Grafik 3: selbst erstellt
Grafik 4: selbst erstellt
Grafik 5: selbst erstellt
Grafik 5.1: selbst erstellt
Grafik 5.2: selbst erstellt
Grafik 6: selbst erstellt
Grafik 6.1: selbst erstellt
Grafik 6.2: selbst erstellt
Grafik 6.3: selbst erstellt
Grafik 6.4: selbst erstellt
Grafik 6.5: selbst erstellt
Grafik 7: selbst erstellt
Grafik 8: selbst erstellt
Grafik 9.1: selbst erstellt
Grafik 9.2: selbst erstellt
Grafik 10: selbst erstellt
Grafik 11: selbst erstellt

8.2 Bildverweis

Bild 1: Screenshot aus der Anwendung
Bild 2: Screenshot der Webseite https://java.com/de/download/
Bild 3: Screenshot der Webseite https://java.com/de/download/mac_download.jsp
Bild 4: Screenshot vom Java Installer auf MacOS
Bild 5.1: Screenshot vom Java Installer auf MacOS
Bild 5.2: Screenshot vom Java Installer auf MacOS
Bild 6.1: Screenshot vom Java Installer auf Windows 10
Bild 6.2: Screenshot vom Java Installer auf Windows 10
Bild 6.3: Screenshot vom Java Installer auf Windows 10
Bild 7: Screenshot der Webseite https://github.com/YenerOezsoy/CYK/releases
Bild 8: Screenshot der Anwendung vom Auswahlfenster der Grammatik
Bild 9: Screenshot der Anwendung vom Auswahlfenster des Algorithmus
Bild 10.1: Screenshot der Anwendung von der textuellen Repräsentation des CNF
Bild 10.2: Screenshot der Anwendung von der graphischen Repräsentation des CNF
Bild 11: Screenshot der Anwendung vom CYK Algorithmus

9. Quellenverweis

- Vorlesungsskript zur theoretischen Informatik 1 & 2 von Prof. Dr. Heiko Körner
  „Einführung in die theoretische Informatik“
  Hochschule Karlsruhe – Technik und Wirtschaft
- Vorlesungsskript zu verteilten Systemen von Prof. Dr. Christian Zirpins
  „Verteilte Systeme 1“
  Hochschule Karlsruhe – Technik und Wirtschaft
- Vorlesungsskript zu Mensch, Maschine und Kommunikaion von Prof. Dr. Ulrich Bröckl
  „Mensch Maschine Kommunikation“
  Hochschule Karlsruhe – Technik und Wirtschaft
- Vorlesungsskript zur deklarativen Programmierung von Prof. Dr. Christian Pape
  „XML“
  Hochschule Karlsruhe – Technik und Wirtschaft
- Vorlesungsskript zu Software-Engineering von Prof. Dr. Thomas Fuchß
  „Software-Engineering“
  Hochschule Karlsruhe – Technik und Wirtschaft
- Vorlesungsskript zu Softwarearchitektur von Prof. Dr. Walter F. Tichy
  „Softwaretechnik“
  Karlsruher Institut für Technologie
- Vorlesungsskript zur theoretischen Informatik 2 von Prof. Dr. Dorothea Wagner
  „Technische Grundlagen der Infomartik – TGI“
  Karlsruher Institut für Technologie
- Vorlesungsskript zu Algorithmen von Prof. Dr. Peter Sanders
  „Algorithmen 1“
  Karlsruher Institut für Technologie
- Seminar zu XML von Christian Weber
  „XML Grundlagen“
  http://wwwlgis.informatik.uni-kl.de/archiv/wwwdvs.informatik.uni-kl.de/courses/seminar/WS0203/ausarbeitung1.pdf
  Uni Kaiserslautern
- https://de.wikipedia.org/wiki/GitHub
- https://de.wikipedia.org/wiki/Java_(Programmiersprache)
- https://www.java.com/de/
- http://www.oracle.com/us/technologies/java/050862.pdf