TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Theoretische Informatik

Bachelorarbeit

HIERARCHISCHES A*-PARSING FÜR PROBABILISTISCHE KONTEXTFREIE GRAMMATIKEN

Richard John
Mat.-Nr.: 3680812

Betreut durch:
Dipl.-Inf. Toni Dietze

Eingereicht am 05. September 2013

DANKSAGUNG

Ich möchte mich an dieser Stelle bei meinem Betreuer Dipl.-Inf. Toni Dietze sowohl dafür bedanken, dass er mir dieses interessante Thema zur Verfügung gestellt hat, als auch für seine Unterstützung im Rahmen der Bachelorarbeit.
Des Weiteren gilt mein Dank dem Lehrstuhl Grundlagen der Programmierung und hierbei insbesondere Herrn Prof. Dr.-Ing. habil. Heiko Vogler, der für diese Arbeit der zuständige Hochschullehrer ist.
Ich danke weiterhin meiner Familie, meiner Freundin und Freunden, ohne die das Studieren unter diesen guten Bedingungen nicht möglich gewesen wäre.

b

ABSTRACT

Diese Bachelorarbeit befasst sich mit dem hierarchischen A*-Parsing im Bezug auf probabilistische kontextfreie Grammatiken (PCFG). Im Kontext von PCFGs hat Parsing das Ziel zu einem gegebenen Wort eine Ableitung aus Regeln der PCFG zu finden, die dieses Wort erzeugt.
Hierbei genügt es in vielen Anwendungen nicht eine beliebige Ableitung bezüglich Wort und PCFG zu finden, sondern man möchte diejenige Ableitung mit dem geringsten Gewicht berechnen. Um dies zu erreichen wurde von Falzenszwalb und McAllester in [FM07] ein Verfahren präsentiert, das die Suche nach der Ableitung mit dem geringsten Gewicht, auf der A*-Suche aufbauend, mit Hilfe einer Hierarchie von PCFGs beschleunigen kann. Allerdings muss das dort allgemein formulierte Verfahren dazu zuerst für PCFGs instanziiert werden.
Dabei werden ausgehend von einer PCFG weitere PCFGs erzeugt, wobei über die Nichtterminalsymbole abstrahiert wird. In der so entstehenden PCFG-Hierarchie dient die jeweils eine Ebene höher liegende Grammatik der vorherigen als Heuristik, um die Berechnung der Ableitung so zu beeinflussen, dass sie möglichst schnell zu einer Lösung findet.

d

Aufgabenstellung für die Bachelor-Arbeit

„Hierarchisches A*-Parsing für probabilistische kontextfreie Grammatiken“

Lehrstuhl Grundlagen der Programmierung
Institut für Theoretische Informatik
Technische Universität Dresden

Eine wichtige Aufgabe beim Umgang mit probabilistischen kontextfreien Grammatiken (PCFG) ist das Finden der wahrscheinlichsten Ableitung einer PCFG für eine Folge von Terminalsymbolen, was im Folgenden als Parsing bezeichnet wird. Für die maschinelle Verarbeitung natürlicher Sprachen (engl.: natural language processing, kurz: NLP) verwendete PCFG beinhalten oft sehr viele Regeln, sodass es effizienter Algorithmen für das Parsing bedarf.

Ein effizienter Parsing-Algorithmus ist das hierarchische A*-Parsing (kurz: HA*-Parsing). Er benötigt eine Hierarchie von PCFG, die wie folgt konstruiert werden kann: Eine PCFG lässt sich durch Zusammenfassen mehrerer Nichtterminalsymbole zu einem neuen Nichtterminalsymbol vereinfachen. Die ursprüngliche Grammatik nennt man dann *fein*, die neue *grob*. Dieser Vorgang lässt sich jeweils für eine so konstruierte Grammatik wiederholen, sodass eine Hierarchie von Grammatiken entsteht.

Beim HA*-Parsing trägt eine solche Grammatik-Hierarchie zur Beschleunigung des Parsings bei, indem jeweils gröbere Grammatiken zur Berechnung von Heuristiken für das gezielte Parsing mittels A*-Suche [HNR68] für feinere Grammatiken herangezogen werden. Die dem zugrunde liegende HA*-Suche wurde von Felzenszwalb und McAllester [FM07] formalisiert und von Pauls und Klein [PK09] für das Parsing von PCFG instanziiert.

Im Rahmen seiner Bachelor-Arbeit soll der Student folgende Aufgaben bearbeiten.

- Der Student soll die Instanz der HA*-Suche für das Parsing von PCFG formal beschreiben und die dafür benötigten Algorithmen darlegen.
- Der Student soll das HA*-Parsing in einer Programmiersprache seiner Wahl implementieren. Das Programm soll möglichst plattformunabhängig gestaltet werden und insbesondere unter GNU/Linux lauffähig sein. Außerdem soll das Programm in das am Lehrstuhl entwickelte System VANDA eingebunden werden.

e

- Beim Training von Grammatiken mit dem Berkeley-Parser [Pet+06] entsteht als Nebenprodukt eine für das HA*-Parsing verwendbare Hierarchie von Grammatiken. Die Implementierung des Studenten soll das Grammatik-Format des Berkeley-Parsers[1] unterstützen, sodass die beim Training entstandene Hierarchie direkt verwendet werden kann.

- Der Student soll anhand geeigneter Beispiele das Laufzeitverhalten seiner Implementierung untersuchen und dokumentieren, z. B. die Abhängigkeit der Laufzeit von der Länge der zu parsenden Terminalsymbolfolge.

Optional können folgende Aufgaben bearbeitet werden.

- Es ist wünschenswert, dass der Student seine Implementierung in Haskell verfasst.

- Der Student kann zusätzlich den von Pauls und Klein [PK10] beschriebenen Bridge-HA*-Algorithmus (BHA*) vorstellen und implementieren. BHA* hat eine gegenüber HA* komplexere Heuristik, was in vielen Fällen zu einer besseren Laufzeit führt.

Die Arbeit muss den üblichen Standards wie folgt genügen. Die Arbeit muss in sich abgeschlossen sein und alle nötigen Definitionen und Referenzen enthalten. Die Struktur der Arbeit muss klar erkenntlich sein, und der Leser soll gut durch die Arbeit geführt werden. Die Darstellung aller Begriffe und Verfahren soll mathematisch formal fundiert sein. Für jeden wichtigen Begriff sollen Beispiele angegeben werden, ebenso für die Abläufe der beschriebenen Verfahren. Wo es angemessen ist, sollten Illustrationen die Darstellung vervollständigen. Schließlich sollen alle Lemmata und Sätze möglichst lückenlos bewiesen werden. Die Beweise sollen leicht nachvollziehbar dokumentiert sein. Für die Implementierung soll eine ausführliche Dokumentation erfolgen, die sich angemessen auf den Quelltext und die schriftliche Ausarbeitung verteilt. Dabei muss die Funktionsfähigkeit des Programms glaubhaft gemacht werden.

Der Student verpflichtet sich ihm im Rahmen der Arbeit zugänglich gemachte Software und Quellcodes lediglich zur Erledigung seiner Aufgaben zu verwenden, vertraulich zu behandeln und nicht an Dritte weiterzugeben. Einer späteren Veröffentlichung seiner Arbeiten unter einer Open-Source-Lizenz stimmt der Student zu.

[1] Ausgabe einer Grammatik im Text-Format mittels:
`java -cp BerkeleyParser.jar edu.berkeley.nlp.PCFGLA.WriteGrammarToTextFile`

f

Personalien

Student:	Richard John
Matrikelnummer:	3680812
Studiengang:	Bachelor Informatik (Prüfungsordnung 2009)
Studienleistung:	Bachelor-Arbeit (12 Wochen, 12 LP) und Kolloquium (60 Minuten, 1 LP)
1. Gutachter:	Prof. Dr.-Ing. habil. Heiko Vogler
2. Gutachter, Betreuer:	Dipl.-Inf. Toni Dietze
Beginn:	13. Juni, 2013
Abgabedatum:	5. September, 2013

Dresden, 6. Juni, 2013

Unterschrift von Heiko Vogler

Unterschrift von Richard John

Literatur

[FM07] Pedro F. Felzenszwalb und David McAllester. „The generalized A* architecture“. In: *J. Artif. Int. Res.* 29.1 (Juni 2007), S. 153–190. ISSN: 1076-9757. URL: http://dl.acm.org/citation.cfm?id=1622606.1622612.

[HNR68] Peter E. Hart, Nils J. Nilsson und Bertram Raphael. „A Formal Basis for the Heuristic Determination of Minimum Cost Paths“. In: *Systems Science and Cybernetics, IEEE Transactions on* 4.2 (1968), S. 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136.

[Pet+06] Slav Petrov u. a. „Learning accurate, compact, and interpretable tree annotation“. In: *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. ACL-44. Sydney, Australia: Association for Computational Linguistics, 2006, S. 433–440. DOI: 10.3115/1220175.1220230.

[PK09] Adam Pauls und Dan Klein. „Hierarchical search for parsing“. In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. NAACL '09. Boulder, Colorado: Association for Computational Linguistics, 2009, S. 557–565. ISBN: 978-1-932432-41-1. URL: http://dl.acm.org/citation.cfm?id=1620754.1620835.

[PK10] Adam Pauls und Dan Klein. „Hierarchical A* parsing with bridge outside scores“. In: *Proceedings of the ACL 2010 Conference Short Papers*. ACLShort '10. Uppsala, Sweden: Association for Computational Linguistics, 2010, S. 348–352. URL: http://dl.acm.org/citation.cfm?id=1858842.1858906.

g

h

ERKLÄRUNG

Ich erkläre, dass ich die vorliegende Arbeit selbständig, unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Dresden, 05. September 2013

INHALTSVERZEICHNIS

1 EINLEITUNG

1.1 MOTIVATION

In der Informatik werden probabilistische kontextfreie Grammatiken in vielfältigen Bereichen genutzt. Am ehesten ist dem Leser davon wahrscheinlich die Sprachverarbeitung bekannt, in der PCFGs eine entscheidende Rolle spielen. In diesem Kontext ist es auch fachfremden Lesern wohl am intuitivsten, zu welchem Zweck die PCFGs verwendet werden. Sie dienen, wie auch im allgemeinen Fall üblich, der Modellierung eines Sachverhaltes, zum Beispiel der Satzstruktur natürlicher Sprache.
Aber auch in anderen Gebieten spielen diese Grammatiken eine wichtige Rolle. So werden sie unter anderem in der Biotechnologie zur Darstellung von RNA-Molekülen gebraucht und spiegeln dort die Basenpaarungen wider.
Nun kommt der Bereich des Parsings ins Spiel, also der Suche nach Ableitungen für ein gegebenes Wort. Um auf das Eingangsbeispiel zurück zu kommen, könnte dieses ein deutscher Satz sein, für den der wahrscheinlichste Aufbau bestimmt werden soll.
Da reale Grammatiken extrem groß werden können, ist es nicht praktikabel sich alle Möglichkeiten von passenden Parsebäumen aufgeben zu lassen und diese untereinander zu vergleichen. Es bedarf effizienter Verfahren, um das Problem des Findens der wahrscheinlichsten Ableitung zufriedenstellend zu lösen.
In dieser Arbeit wird nun ein solches Verfahren vorgestellt: das HA*-Parsing für PCFGs.

1.2 ZIELSTELLUNG

Das hierarchische A*-Parsing Verfahren wurde in [FM07] vorgestellt. Darin ist es sehr allgemein gehalten, um es für möglichst viele verschiedene Aufgabenbereiche anwendbar zu machen.
Der Kern dieser Arbeit ist es deshalb die Instanziierung des HA*-Parsings für den konkreten Fall der PCFGs. Die allgemeinen Formulierungen sollen also auf diesen speziellen Fall heruntergebrochen werden, um diesem gerecht zu werden.
Neben diesem sehr theoretischen Teil sollte eine Implementierung des Verfahrens realisiert und schließlich auf ihr Laufzeitverhalten getestet werden.

1.3 AUFBAU DER ARBEIT

In Kapitel 2 werden wichtige Begriffe, die sich überall in der Arbeit wiederfinden lassen, erklärt. Unterstützend zu Definitionen finden sich dort, wenn es passend ist, Abbildungen, die ein schnelles Verständnis der Sachverhalte fördern sollen. Einige der dort beschriebenen Begriffe kommen allgemein aus dem Bereich der formalen Sprachen und andere sind eher als spezifisch für das HA*-Parsing.
Nachdem also die Grundlagen für ein möglichst gutes Verständnis der Arbeit geschaffen wurden, wird direkt auf das HA*-Parsing eingegangen. Auch in diesem dritten Kapitel erfolgt erstmal eine weitere Vertiefung, bevor tatsächlich auf das Verfahren an sich eingegangen wird.
Kapitel 4 stellt dann den Übergang von der Theorie zur Praxis dar. Dort wird die, im Umfang dieser Bachelorarbeit verwirklichte, Implementierung des HA*-Parsings beschrieben.
Im vorletzten Kapitel soll dann, aufbauend auf dem vorherigen Kapitel, eine Analyse des Programms erfolgen.
Schließlich folgt eine Zusammenfassung der Arbeit.

2 GRUNDLAGEN

2.1 PCFG

Im Bereich der formalen Sprachen ist eine probabilistische kontextfreie Grammatik (PCFG) ein Spezialfall der kontextfreien Grammatik.
Eine Kontextfreie Grammatik entspricht dem Typ-2 der Chomsky-Hierarchie und kann als 4-Tupel $(\mathcal{V}, \Sigma, \mathcal{R}, \mathcal{S})$ formalisiert werden:

kontextfreie Grammatik $\mathcal{G}(\mathcal{V}, \Sigma, \mathcal{R}, \mathcal{S})$

- $\mathcal{V}$ endliche Menge der Nichtterminalsymbole
- Σ endliche Menge der Terminalsymbole
- $\mathcal{R}$ endliche Menge Regeln der Form A $\rightarrow \alpha$ mit A $\in \mathcal{V}$ und $\alpha \in (\mathcal{V} \cup \Sigma)$*
- $\mathcal{S}$ Startsymbol mit $\mathcal{S} \in \mathcal{V}$

Eine probabilistische kontextfreie Grammatik ist eine kontextfreie Grammatik $\mathcal{G}(\mathcal{V}, \Sigma, \mathcal{R}, \mathcal{S})$, deren Regeln jeweils eine Wahrscheinlichkeit zugewiesen ist, sodass die Regeln die Form

$$A \xrightarrow{p} \alpha$$

haben mit p $\in$ [0, 1] und folgende Bedingung für jedes A $\in \mathcal{V}$ erfüllt ist:

$$\sum_{A \xrightarrow{p} \alpha} p = 1$$

Beispiel für eine PCFG:

$$\mathcal{G}(\{\text{S, A}\}, \{\text{a, b}\}, \mathcal{R}, \text{S})$$

$$\begin{aligned}
\mathcal{R} = \{ \quad & S \xrightarrow{0,9} AS \\
& S \xrightarrow{0,1} a \\
& A \xrightarrow{0,5} AA \\
& A \xrightarrow{0,2} a \\
& A \xrightarrow{0,3} b \quad \}
\end{aligned}$$

2.2 ABLEITUNG

Eine Ableitung eines Wortes, also einer endlichen Zeichenfolge, bezüglich einer PCFG $\mathcal{G}(\mathcal{V}, \Sigma, \mathcal{R}, \mathcal{S})$ beginnt initial mit dem Startsymbol dieser Grammatik. Auf dieses wir dann eine Regel aus $\mathcal{R}$ angewandt. Durch diese Regelanwendung wird das Startsymbol durch die rechts vom Pfeil der Regel stehenden Symbole ersetzt. Allgemein formuliert wird bei einer Regelanwendung ein Vorkommen der Zeichenfolge, die auf der linken Seite einer Regel, also links vom Pfeil, steht, durch die Zeichenfolge, die sich rechts von jenem Pfeil befindet, ersetzt. Zur Erzeugung einer Ableitung werden nun fortlaufend Regelanwendungen vollzogen bis ein Wort entstanden ist, das keine Nichtterminalsymbole mehr enthält.
Eine solche Ableitung soll im Folgenden auch vollständige Ableitung genannt werden, während eine unvollständige Ableitung noch mindestens ein Nichtterminalsymbol enthält. Wenn ohne einen der beiden Zusätze nur schlicht von einer Ableitung gesprochen wird, soll es sich um eine vollständige Ableitung handeln.
In Abhängigkeit von der PCFG und ihrer Regeln kann es mehrere Ableitungen für ein Wort geben. In einigen praktischen Anwendungsfällen, wie aus dem Gebiet der Sprachverarbeitung, wird dann in der Regel nicht nach einer beliebigen Ableitung gefragt, sondern man möchte die wahrscheinlichste Ableitung eines Wortes berechnen.
Die Wahrscheinlichkeit einer Ableitung ergibt sich aus dem Produkt der Wahrscheinlichkeiten der in ihr angewandten Regeln. Falls eine Regel mehrfach benutzt wurde, muss ihre Wahrscheinlichkeit auch dementsprechend oft in das Produkt einfließen.
Zur Darstellung von Ableitungen werden üblicherweise Bäume verwendet. Das Startsymbol der Grammatik ist hierbei die Wurzel des Baumes, die Regelanwendungen werden jeweils durch Verzweigungen dargestellt und in den Blättern stehen Terminalsymbole, die von links nach rechts gelesen das erzeugte Wort ergeben.

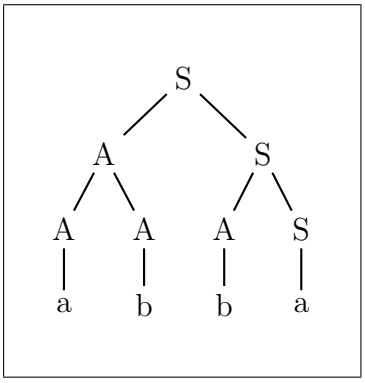

Abbildung 2.1: Beispiel für einen Ableitungsbaum

In der Abbildung ist ein Ableitungsbaum für das Wort abba zu sehen. Dabei wurden Regeln aus der in 2.1 gezeigten Beispielgrammatik benutzt.
Die Wahrscheinlichkeit, der durch den Baum illustrierten Ableitung, ergibt sich folgendermaßen, wenn er in BFS-Manier durchlaufen wird:

$$S \xrightarrow{0,9} AS \qquad A \xrightarrow{0,5} AA \qquad S \xrightarrow{0,9} AS \qquad A \xrightarrow{0,2} a \qquad A \xrightarrow{0,3} b \qquad A \xrightarrow{0,3} b \qquad S \xrightarrow{0,1} a$$

$$0,9 * 0,5 * 0,9 * 0,2 * 0,3 * 0,3 * 0,1 = 0,000729$$

2.3 AUSSAGE

Im Umfang dieser Arbeit soll die Notation $\mathcal{A}$(A, i, j) für eine Aussage stehen. Dabei ist A ein Nichtterminalsymbol einer PCFG $\mathcal{G}(\mathcal{V}, \Sigma, \mathcal{R}, \mathcal{S})$ und i, also auch j, sind Indizes aus dem Intervall [1, n+1], wobei n die Anzahl an Terminalen im zu parsenden Eingabewort $a_1 ... a_n$ ist und i $\leq$ j gilt.
Die Bedeutung der Aussage $\mathcal{A}$(A, i, j) ist, dass es unter Verwendung von $\mathcal{G}$ einen Ableitungsbaum für das Teilwort $a_i ... a_{j-1}$ mit Wurzel A gibt.
Eine Zielaussage ist in jedem Fall die Aussage $\mathcal{A}(\mathcal{S}, 1, n+1)$. Es gibt dann also einen Ableitungsbaum mit dem Startsymbol der PCFG als Wurzel, der das komplette Eingabewort produziert und somit eine Ableitung für das Eingabewort bezüglich der gegebenen PCFG $\mathcal{G}$.

2.4 KONTEXT

Aufbauend auf dem Begriff einer Aussage soll nun die eng damit verbundene Bezeichnung Kontext definiert werden.
Ein Kontext einer Aussage $\mathcal{A}$(A, i, j), im Zeichen $\mathcal{K}(\mathcal{A}$(A, i, j)), ist eine Ableitung der Zielaussage, die genau ein Vorkommen des Nichtterminals A hat, das nicht weiter abgeleitet wurde. Wenn nun dieses Vorkommen von A durch einen Ableitungsbaum mit Wurzel A ersetzt wird, dessen Blätter ausnahmslos mit Terminalsymbolen beschriftet sind, der in diesem Sinne also eine vollständige Ableitung repräsentiert, erhält man eine vollständige Ableitung der Zielaussage.

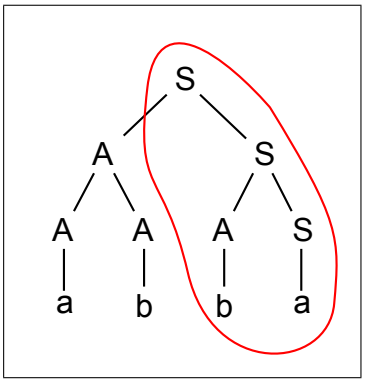

Abbildung 2.2: Beispiel für einen Kontext

Die Abbildung zeigt einen, durch eine rote Linie hervorgehobenen, Kontext $\mathcal{K}(\mathcal{A}$(A, 1, 3)). Diesem fehlt am obersten Vorkommen des Nichtterminals S nur der ebenfalls dargestellte Ableitungsbaum mit Wurzel A, um eine Ableitung der Zielaussage zu erhalten, deren Ableitungsbaum dem kompletten, in der Abbildung zu sehenden Baum entspricht.

2.5 INFERENZREGEL

Eine Inferenzregel soll im Folgenden diese Gestalt haben,

$$\{(\mathcal{X}_1 \cong w_1), \ldots, (\mathcal{X}_n \cong w_n)\} \models_p (\mathcal{X} \cong w)$$

wobei $(\mathcal{X}_1 \cong w_1), \ldots, (\mathcal{X}_n \cong w_n)$, $(\mathcal{X} \cong w)$ Gewichtszuweisungen und $\mathcal{X}_1, \ldots, \mathcal{X}_n, \mathcal{X}$ entweder Aussagen oder Kontexte von Aussagen sind.
Diese Regel gibt eine Möglichkeit an, wie die Gewichtszuweisung $\mathcal{X} \cong$ w hergeleitet werden kann. Dazu muss es Herleitungen der links stehenden Gewichtszuweisungen $(\mathcal{X}_1 \cong w_1), \ldots, (\mathcal{X}_n \cong w_n)$ geben. Wenn dies erfüllt ist, dann kann $\mathcal{X} \cong w$ gefolgert werden, wobei sich das Gewicht w aus g($w_1, \ldots, w_n$) ergibt.
In anderen Worten stehen auf der linken Seite einer Inferenzregel die Vorbedingungen und rechts die Schlussfolgerung.
Das kleine p steht für die Priorität der Regel. Es gilt p $\in \mathbb{R}_{\geq 0}$.

2.6 GEWICHTSZUWEISUNG

Eine Gewichtszuweisung $\mathcal{G}$ ist ein Ausdruck der Form $\mathcal{A}$(A, i, j) $\cong$ w, wobei $\mathcal{A}$(A, i, j) eine Aussage ist und w $\in \mathbb{R}_{\geq 0}$ das Gewicht ist, welches dieser Aussage zugeordnet wird.
Die Bedeutung einer Gewichtszuweisung $\mathcal{A}$(A, i, j) $\cong$ w ist, dass es eine Herleitung der Aussage $\mathcal{A}$(A, i, j) gibt, die das Gewicht w hat. Etwas anschaulicher steht es für die gefolgerte Existenz eines Ableitungsbaumes, der das Gewicht w hat, deren Wurzel mit dem Nichtterminal A beschriftet ist und dessen Blätter von links nach rechts gelesen die Terminalsymbole $a_i, \ldots, a_{j-1}$ des zu parsenden Eingabewortes $a_1 \ldots a_n$ enthalten.
Analog dazu ist der Begriff der Gewichtszuweisung für Kontexte von Aussagen definiert, welche dann die Form $\mathcal{K}(\mathcal{A}$(A, i, j)) $\cong$ w hat.

3 HA*-PARSING

In den folgenden Abschnitten der Arbeit wird nun das HA*-Parsing für PCFGs erklärt. Hierzu soll zunächst die Idee hinter dem Verfahren, dann der grobe Ablauf und danach die verwendeten Inferenzregeln vorgestellt und erklärt werden. Danach wird dann ein Parsingvorgang an einem kleinen Beispiel verdeutlicht. Dies soll dem Leser ein solides Verständnis vom HA*-Parsing geben.

3.1 IDEE DES HA*-ANSATZES

Im vorigen Abschnitt wurde eine Möglichkeit aufgezeigt, wie die geringsten Gewichte zu allen folgerbaren Aussagen und auch bezüglich der Kontexte von Aussagen erzeugt werden können. Ein großer Schwachpunkt des grundlegenden Verfahrens ist die mangelnde Effizienz. So läuft der Algotihmus solange, bis die Queue leer ist. Das ist natürlich nicht falsch, aber es kann in der Regel eigentlich schon vorher abgebrochen werden.
Aus diesem Grund wurde dann versucht die Berechnung gesteuerter ablaufen zu lassen, damit möglichst wenige gar nicht für die endgültige, gesuchte Lösung benötigte Gewichtszuweisungen erzeugt werden und die eigentliche Lösung auf einem möglichst direktem Weg berechnet wird.
Eine Verbesserung stellt der A*-Ansatz dar. Dabei wird eine heuristische Funktion verwendet, die die noch notwendigen Kosten abschätzt, die für eine Vervollständigung eines Ableitungsbaumes zu einem Parsebaum des Eingabewortes entstehen. Die Summe aus den tatsächlich schon durch Inferenz festgestellten Kosten und der Abschätzung der noch zusätzlichen Kosten bis zum Ziel unter Verwendung des bereits vorhandenen Teils liefert ein Gütekriterium für Aussagen. Dieser Wert entscheidet über die Reihenfolge der Bearbeitung der Gewichtszuweisungen.
Wichtig ist hierbei, dass die heuristische Funktion monoton ist. Nur so ist sichergestellt, dass bei der Kombination von optimalen Zwischenlösungen auch auf jeden Fall wieder optimal sind.

Nun stellt sich sofort die Frage woher die heuristische Funktion entsprechende Werte nehmen soll. In diesem Zusammenhang kann wurde in [FM07] gezeigt, dass sich Kontexte hierfür eignen. Um den heuristischen Wert für eine Aussage zu erhalten, wird das geringste Gewicht für einen Kontext bezüglich dieser Aussage berechnet. Allerdings wird dafür nicht auf den Kontext der Aussage zurückgegriffen, sondern der abstrakte Kontext dieser Aussage benutzt.

In Folge dieser Überlegung kommt man schließlich zum hierarchischen A*-Ansatz.

Wie gerade festgestellt wurde kann eine von einer PCFG durch Abstraktion entstandene größere PCFG dabei helfen die Berechnung durch eine Heuristik zu beschleunigen. Allerdings ist gleichzeitig klar, dass es wünschenswert ist, auch die für diese heuristische Funktion benötigten Werte auf der abstrakteren Ebene möglichst effizient zu berechnen. Da liegt es nahe, auch auf dieser

Ebene das selbe Prinzip anzuwenden und die betreffende PCFG auch zu abstrahieren, um genau den selben Effekt zu erzielen und dann auch auf der zweiten Ebene dank heuristischer Werte der dritten Ebene schneller Berechnungen durchführen zu können.
Nach diesem Schema entsteht eine Hierarchie von PCFGs, in der die heuristische Funktion auf einer bestimmten Ebene jeweils auf die nächsthöhere Ebene zurückgreift, um passende Kostenabschätzungen zu erhalten. Aus eher technischen Gründen wird immer eine abstrakteste Ebene in der Hierarchie vorausgesetzt, in der nur $\perp$ und $\mathcal{K}(\perp)$ abgeleitet werden können. Dies ist von Nöten, um einen Anfang in dem gesamten Prozess zu finden.

Die Abstraktion geschieht beim HA*-Parsing auf Grundlage der Nichtterminalsymbole. Für sie ist die Abstraktionsfunktion ebenenweise definiert und in Folge dessen werden automatisch auch die Grammatikregeln abstrahiert, da in ihnen Nichtterminale vorkommen. Um die Korrektheit der Heuristik zu bewahren, ist es bei der Erstellung einer neuen PCFG aus einer anderen durch Abstraktion der Nichtterminale notwendig, die umgeformten Regeln noch nachzubearbeiten. Denn durch die Abstraktion kann es zu Regelnduplikaten kommen, die bis auf ihre Gewichte vollständig identisch sind. In diesem Fall darf nur die Regelvariante tatsächlich in die neue PCFG übernommen werden, die das geringste Gewicht unter allen Duplikaten hat.

3.2 ABLAUF

Der Ablauf des HA*-Parsings wird durch einen Grundalgorithmus bestimmt, der aus dem Jahr 1977 stammt.
Dieser geht von einer vorgegebenen Menge an priorisierten Inferenzregeln aus und arbeitet in der hier vorgestellten Umgebung auf Gewichtszuweisungen.
Gesteuert wird der Ablauf durch eine Queue, die alle noch zu bearbeitenden Gewichtszuweisungen hält. Die Besonderheit hier ist die Tatsache, dass die Gewichtszuweisungen innerhalb der Queue bezüglich ihrer Priorität geordnet sind. Der Prioritätswert, unter dem ein neues Element in die Queue eingefügt wird, bestimmt die Inferenzregel, welche für die Erzeugung des betreffenden Elementes verantwortlich war.
Initialisiert wird die Queue mittels Inferenzregeln, die keine Vorbedingungen haben und somit schon anfangs, ohne das Vorhandensein von erzeugten Gewichtszuweisungen, angewandt werden können.
Danach wird in jeder Iteration einer while-Schleife, die solange läuft, wie Elemente in der Queue vorhanden sind, jeweils eine Gewichtszuweisung von der Queue geholt und erstmal geprüft, ob schon vorher eine, bis auf das Gewicht identische, Gewichtszuweisung produziert wurde. Nur wenn dies nicht der Fall ist und damit das erste Mal ein Gewicht für die entsprechende Aussage oder den entsprechenden Kontext einer Aussage gefunden wurde, wird versucht, damit weitere gewichtszuweisungen zu folgern, die dann ebenfalls in die Queue einsortiert werden.

Procedure $Run(P)$
1. $\mathcal{S} \leftarrow \emptyset$
2. Initialize $\mathcal{Q}$ with assignments defined by rules with no antecedents at their priorities
3. **while** $\mathcal{Q}$ is not empty
4. Remove the lowest priority element $(B = w)$ from $\mathcal{Q}$
5. **if** B has no assigned weight in $\mathcal{S}$
6. $\mathcal{S} \leftarrow \mathcal{S} \cup \{(B = w)\}$
7. Insert assignments derivable from $(B = w)$ and other assignments in $\mathcal{S}$ using some rule in P into $\mathcal{Q}$ at the priority specified by the rule
8. **return** $\mathcal{S}$

Abbildung 3.1: Pseudocode für die Ausführung von priorisierten Inferenzregeln aus [FM07],
(B = w) entspricht hier einer Gewichtszuweisung,
$\mathcal{S}$ ist die Menge aller schon erzeugten Gewichtszuweisungen,
$\mathcal{Q}$ steht für die Queue

Dieses Vorgehen führt unter gewissen Voraussetzungen dazu, dass jedes Element, das von der Queue geholt wird und für dessen Kontext einer Aussage oder dessen Aussage bisher noch keine Gewichtszuweisung gefunden wurde, jeweils das geringste Gewicht unter allen Gewichtszuweisungen für den entsprechenden Kontext einer Aussage oder für die entsprechende Aussage hat. Dies ist genau die Eigenschaft, die für die Suche nach dem wahrscheinlichsten Parsebaum mittels HA*-Parsing benötigt wird. Zu jedem Zeitpunkt müssen alle verwendeten Teillösungen optimal sein, damit größere, aus ihnen entstehende Teillösungen auch wieder optimal sind und schließlich das Ergebnis tatsächlich das gewünschte ist.

3.3 INFERENZREGELN

Wie in [FM07] beschrieben werden für das Parsing fünf verschiedene Inferenzregelarten verwendet, deren Bedeutung nun jeweils kurz beschrieben werden soll.

$$\{\ \} \models_0 (\bot \cong 0)$$
$$\{\ \} \models_0 (\mathcal{K}(\bot) \cong 0)$$

Diese Regeltypen werden mit "START1" beziehungsweise "START2" bezeichnet. Das liegt am Zeitpunkt der Verwendung, denn sie werden nur zur Initialisierung der Queue verwendet, in der alle noch zu bearbeitenden Gewichtszuweisungen, sortiert nach ihrer Priorität, enthalten sind. Begründen lässt sich die Erfordernis solcher Regeln in der Tatsache, dass anfangs noch keine Gewichtszuweisungen vorhanden sind, um Vorbedingungen von Inferenzregeln erfüllen zu können. Aus diesem Grund braucht man Inferenzregeln ohne Vorbedingung, die damit trivialerweise erfüllt sind, sodass in diesem Fall auf $(\bot \cong 0)$ beziehungsweise $(\mathcal{K}(\bot) \cong 0)$ geschlossen werden kann.

Die weiteren drei Regeltypen sind in dem Sinne bedeutender, dass sie nicht nur einmal anfangs verwendet werden können, sondern dies ist erst ausschließlich nach der Initialisierung der Queue mit den START-Regeln möglich, da sie gewisse Vorbedingungen haben.

$$\{(\mathcal{A}(\mathcal{S}_l, 1, n+1) \cong w)\}$$
$$\models_w (\mathcal{K}(\mathcal{A}(\mathcal{S}_l, 1, n+1)) \cong 0)$$

Bei dieser Inferenzregelart, "BASE" genannt, ist die Komplexität der Struktur schon etwas größer. Als einzige Vorbedingung wird eine Gewichtszuweisung für die Zielaussage von Hierarchielevel l gefordert. Von dieser aus lässt sich dann auf einen Kontext für die Zielaussage der Vorbedingung schließen, dessen Gewicht auf Null festgelegt wird.

$$\begin{aligned} \{\quad & (\mathcal{K}(\mathcal{A}(abs(A), i, k)) \cong w_K), \\ & (\mathcal{A}(B_1, i, j) \cong w_1), \\ & (\mathcal{A}(B_2, j, k) \cong w_2) \quad \} \\ \models_{v+w_1+w_2+w_K} & (\mathcal{A}(A, i, k) \cong v + w_1 + w_2) \end{aligned}$$

Die Inferenzregeln des Typs "UP" verwenden nun erstmals die Heuristik in Form eines abstrakten Kontextes, der als Vorbedingung in die Regel eingeht. Dieser stellt sicher, dass der Wert w_K zum Zeitpunkt der Anwendung der Regel vorhanden ist, da er bei der Berechnung der Priorität der zu erzeugenden Gewichtszuweisung benötigt wird.
Die Summe $v+w_1+w_2+w_K$, die die Priorität bestimmt, liefert eine Abschätzung wie groß das Gewicht einer Ableitung der Zielaussage sein wird, wenn für diese der Teilbaum, der durch $\mathcal{A}(A, i, k)$ repräsentiert wird, benutzt wird. In Folge der Definition der Abstraktion ist diese Abschätzung eine untere Schranke, da die heuristische Funktion die tatsächlichen Kosten nie überschätzt.
So wie die UP-Regeln hier dargestellt sind, beziehen sie sich auf eine Verbundregel $A \xrightarrow{v} B_1 B_2$. Gut zu erkennen ist die Beschaffenheit der Grenzen der Aussagen der Gewichtszuweisungen $(\mathcal{A}(B_1, i, j) \cong w_1)$ *und* $(\mathcal{A}(B_2, j, k) \cong w_2)$. Deren Aussagen decken zusammengenommen den selben Bereich i bis k-1 des Eingabewortes ab, wie auch die anderen beiden Aussagen, die in der Regel vorkommen.

Wenn einer UP-Regel nun keine Verbundregel, sondern eine Terminalregel einer PCFG zugrunde liegt, dann wird die Inferenzregel strukturell einfacher:

$$\begin{array}{c} \{ \quad (\mathcal{K}(\mathcal{A}(abs(A), i, i+1)) \cong w_K) \quad \} \\ \models_{v+w_k} (\mathcal{A}(A, i, i+1) \cong v) \end{array}$$

Nun bleibt lediglich eine Vorbedingung übrig und zudem kann man genauere Angaben über die Grenzen der Aussagen machen, denn Ableitungsbäume, in denen auf das in der Wurzel stehende Nichtterminal zuerst eine Terminalregel angewandt wird, können grundsätzlich immer nur ein Terminal des Eingabewortes betreffen. Daher ist vorgegeben, dass die obere Grenze, der in der Inferenzregel vorkommenden Aussagen, genau um eins größer sein muss, als die untere Grenze.

$$\begin{array}{c} \{(\mathcal{K}(\mathcal{A}(A, i, k)) \cong w_K), \\ (\mathcal{A}(B_1, i, j) \cong w_1), \\ (\mathcal{A}(B_2, j, k) \cong w_2)\} \\ \models_{v+w_1+w_2+w_k} (\mathcal{K}(\mathcal{A}(B_x, i, k)) \cong v + w_1 + w_2 + w_K - w_x) \end{array}$$

Der letzte, jetzt noch fehlende Inferenzregeltyp trägt den Namen "DOWN". Die Struktur der Regeln sieht auf den ersten Blick recht ähnlich aus, wie bei den UP-Regeln, allerdings gibt es entscheidende Unterschiede.
Zum einen ist der hier als Vorbedingung verwendete Kontext nicht aus einer höheren Hierarchieebene, als die Aussagen der anderen Gewichtszuweisungen. Zum anderen wird dieses Mal auf Kontexte gefolgert. Wenn die Vorbedingungen erfüllt sind, kann auf gleich zwei Kontexte geschlossen werden, was an der Variablen $x \in \{1, 2\}$ deutlich wird.

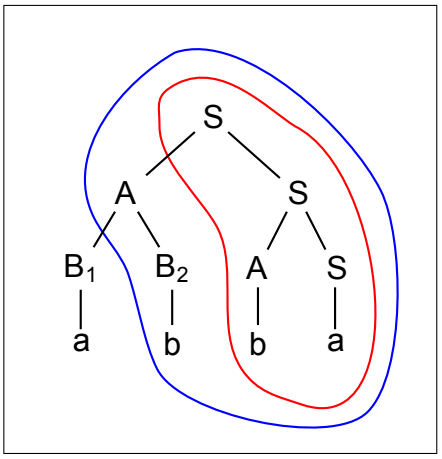

Abbildung 3.2: Entstehung eines Kontextes
$\mathcal{K}(\mathcal{A}(B_1, 1, 2))$ (blau) aus
$\mathcal{K}(\mathcal{A}(A, 1, 3))$ (rot)

In der Abbildung ist der Fall x=1 dargestellt. Falls im Bezug auf diese Abbildung alle Vorbedingungen der Inferenzregel erfüllt wären, würde diese so aussehen:

$$\begin{aligned} &\{(\mathcal{K}(\mathcal{A}(A, 1, 3)) \cong w_K), \\ &\quad (\mathcal{A}(B_1, 1, 2) \cong w_1), \\ &\quad (\mathcal{A}(B_2, 2, 3) \cong w_2)\} \\ \models_{v+w_1+w_2+w_K} &\ (\mathcal{K}(\mathcal{A}(B_1, 1, 2)) \cong v + w_2 + w_K) \end{aligned}$$

Im Bezug auf Terminalregeln einer PCFG können die DOWN-Regeln nicht angewandt werden, da Terminalregeln auf ihrer rechten Seite lediglich ein Terminalsymbol haben und somit kein Nichtterminal, für das gegebenenfalls auf einen neuen Kontext einer entsprechenden Aussage geschlossen werden könnte.

3.4 BEISPIEL

3.4.1 Vorbereitung

In diesem Abschnitt wird das HA*-Parsing für ein ganz kleines Beispiel Schritt für Schritt nachvollzogen, um ein besseres Verständnis für den Ablauf Verfahrens zu ermöglichen.
Als erstes werden alle Vorbereitungen getroffen. Dazu gehört die Angabe einer PCFG. Der Einfachheit halber wird hier davon ausgegangen, dass sich diese bereits in Chomsky-Normalform befindet, also nur Verbund- und Terminalregeln haben darf. Dies ist die im Folgenden verwendete PCFG:

$$\mathcal{G}(\{\mathrm{S}, \mathrm{A}\}, \{\mathrm{a}, \mathrm{b}\}, \mathcal{R}, \mathrm{S})$$

$$\begin{aligned} \mathcal{R} = \{ \quad & S \xrightarrow{0,5} AS \\ & S \xrightarrow{0,3} AA \\ & S \xrightarrow{0,2} b \\ & A \xrightarrow{0,1} a \\ & A \xrightarrow{0,9} b \quad \} \end{aligned}$$

Um mit additiven Regeln arbeiten zu können und mit der Suche nach den Gewichtszuweisungen mit geringsten Gewichten letztendlich doch den wahrscheinlichsten Parsebaum zu berechnen, müssen die Wahrscheinlichkeitswerte umgewandelt werden. Hier werden, so wie es in die Regel auch der Fall ist, die gegebenen Wahrscheinlichkeiten logarithmiert und anschließend negiert. Als Ergebnis der Transformation ergeben sich folgende, gerundete Werte für die Regeln:

$$\begin{aligned} \mathcal{R} = \{ \quad & S \xrightarrow{0,3} AS \\ & S \xrightarrow{0,52} AA \\ & S \xrightarrow{0,7} b \\ & A \xrightarrow{1} a \\ & A \xrightarrow{0,05} b \quad \} \end{aligned}$$

Ausgehend von diesen Regeln wird nun auf Basis der Nichtterminale abstrahiert, wobei beide Nichtterminale S und A auf das selbe Nichtterminal S' der nächsthöheren Ebene abgebildet werden. Die PCFG auf dieser Ebene enthält also nur das eine Nichtterminalsymbol S', das dementsprechend auch für diese Grammatik die Rolle des Startsymbols übernimmt. Da in diesem kleinen Beispiel nur zwei Hierarchiestufen vorkommen, wird die feinere, anfangs gegebene PCFG im weiteren Verlauf auch als untere Hierarchieebene und die von ihr abstrahierte PCFG als obere Hierarchieebene bezeichnet werden. Darüber hinaus gibt es trotzdem noch die abstrakteste Ebene, die eher eine technische Rolle hat und immer ganz oben in der Hierarchie steht. Allerdings wird sie nur anfangs zu Initialisierungszwecken verwendet und ist danach unbedeutend.
Anhand der Abstraktion werden nun alle Regeln der unteren Hierarchieebene abgebildet, sodass diese Regelmenge als Zwischenergebnis für die obere Hierarchieebene entsteht:

$$\begin{aligned}\mathcal{R}' = \{ \quad & S' \xrightarrow{0,3} S'S' \\ & S' \xrightarrow{0,52} S'S' \\ & S' \xrightarrow{0,7} b \\ & S' \xrightarrow{1} a \\ & S' \xrightarrow{0,05} b \quad \}\end{aligned}$$

Durch die Abstraktion der Nichtterminalvorkommen in den Grammatikregeln wurden im Beispiel nun in zwei Fällen jeweils zwei verschiedene Regeln der unteren PCFG auf die selbe Regel abgebildet. Für die Korrektheit des gesamten späteren Parsings ist es jetzt notwendig, dass genau die Regel mit dem kleinsten Gewicht, das zuvor aus der ursprünglichen Wahrscheinlichkeit der vorgegebenen Regel errechnet wurde, ausgewählt wird, wenn es mehrere Regeln gibt, die sich nach der Abstraktion nur noch im Gewichtswert unterscheiden.
Durch dieses Vorgehen, angewandt auf die eben erzeugte Regelmenge, verkleinert sich diese folgendermaßen:

$$\begin{aligned}\mathcal{R}' = \{ \quad & S' \xrightarrow{0,3} S'S' \\ & S' \xrightarrow{1} a \\ & S' \xrightarrow{0,05} b \quad \}\end{aligned}$$

Da die Terminalsymbole der unteren PCFG von der Abstraktion unangetastet bleiben, werden sie einfach in die nächsthöhere PCFG übernommen. Damit ist auch diese PCFG nun vollständig definiert:

$$\mathcal{G}'(\{S'\}, \{a, b\}, \mathcal{R}', S')$$

$$\mathcal{R}' = \{ \quad S' \xrightarrow{0,3} S'S' \quad S' \xrightarrow{1} a \quad S' \xrightarrow{0,05} b \quad \}$$

Die für das HA*-Parsing erforderliche PCFG-Hierarchie ist damit fertig. Es fehlt nur noch ein zu parsendes Eingabewort. Bei dessen Wahl ist man, ähnlich wie vorher bei der Festlegung auf eine Ausgangsgrammatik und deren Umfang, sehr stark eingeschränkt, wenn man einen Parsevorgang tatsächlich Schritt für Schritt aufzeigen möchte, denn mit der Länge des Eingabewortes steigt die Anzahl an Möglichkeiten für die Anwendung der Inferenzregeln und die somit gefolgerten Gewichtszuweisungen schnell in Bereiche, die es nicht mehr zulassen, den kompletten Ablauf in einem überschaubaren Rahmen unterzubringen.
Aus diesem Grund wird für dieses Beispiel das Eingabewort "ab" verwendet, sodass beim Parsing nach einer Gewichtszuweisung für die Aussage $\mathcal{A}(S, 1, 3)$ gefragt wird.
Im Folgenden wird das HA*-Parsing für die eben festgelegten Parameter ausgeführt. Dabei soll jeder Schritt in diesem Format festgehalten werden:

[verwendeter Inferenzregeltyp] || [Gewichtszuweisung] || [Inhalt der Queue]

[berechnete Gewichtszuweisungen]

Dabei ist der verwendete Inferenzregeltyp ein Element der Menge {BASE, UP, DOWN}.
Die Gewichtszuweisung ist diejenige, die zu Beginn des jeweiligen Schrittes aus der Queue genommen wird und damit die mit dem geringsten Prioritätswert unter allen, zu diesem Zeitpunkt in der Queue vorhandenen, Gewichtszuweisungen.
Ganz rechts von der ersten Zeile der Kodierung eines Schrittes wird ein Abbild der Queue nach der Ausführung der entsprechenden Iteration dargestellt.
In der zweiten Zeile werden alle Gewichtszuweisungen aufgelistet, die schon vor dem Start der jeweiligen Iteration berechnet wurden, um erkennbar zu machen, welche Gewichtszuweisungen für Inferenzregelanwendungen in diesem Schritt zur Verfügung stehen.

3.4.2 Parsing

Initialisiert wird die Queue gemäß der START1- und START2-Inferenzregel mit der einzigen Aussage, die es auf der abstraktesten Ebene gibt, welche laut Definition $\perp$ ist. Zudem wird auch ihr Kontext in die Queue angefügt.

$$- \quad \| \quad - \quad \| \quad [(\perp \cong 0)_0, (\mathcal{K}(\perp) \cong 0)_0]$$

$$\{\,\}$$

Im ersten Schritt wird nun das Elemente der Queue genommen, welches die geringste Priorität hat. Im Beispiel ist dies $(\perp \cong 0)_0$ mit der Priorität 0 in Indexschreibweise.

$$- \quad \| \quad (\perp \cong 0) \quad \| \quad [(\mathcal{K}(\perp) \cong 0)_0]$$

$$\{\,\}$$

Mit dieser Gewichtszuweisung lassen sich zu diesem Zeitpunkt keine weiteren Gewichtszuweisungen erzeugen, weshalb sie nur in die Menge aller
schon erhaltenen Gewichtszuweisungen eingetragen wird und danach mit dem nächsten Schritt weitergemacht werden kann.

$$UP \quad \| \quad (\mathcal{K}(\perp) \cong 0) \quad \| \quad [(\mathcal{A}(S', 2, 3) \cong 0,05)_{0,05}, (\mathcal{A}(S', 1, 2) \cong 1)_1]$$

$$\{(\perp \cong 0)\}$$

Jetzt konnten Varianten der UP-Inferenz angewendet werden, da es durch $(\mathcal{K}(\perp) \cong 0)$ für alle Aussagen bezüglich der oberen PCFG damit einen abstrakten Kontext gibt. Zwar kann auf Grund noch nicht erzeugter Aussagen für die obere PCFG keine Inferenz für eine Verbundregel stattfinden, aber für UP-Inferenzen der Terminalregeln werden diese nicht benötigt, sodass der abstrakte Kontext zur Erfüllung aller Vorbedingungen der entsprechenden Inferenzregeln ausreichend ist.
Zur Bestimmung der Grenzen der erzeugbaren Aussagen muss man prüfen, welches Terminal des Eingabewortes mit einer Terminalregel der oberen Hierarchieebene produziert werden kann und bezüglich dessen Index im Eingabewort die Grenzen festlegen.

Da es in der PCFG $\mathcal{G}'$ beispielsweise die Terminalregel $S' \xrightarrow{1} a$ gibt und das Terminalsymbol a an erster Stelle des Eingabewortes vorkommt, wird die Aussage $\mathcal{A}(S', 1, 2) \cong 1$ mit berechneter Priorität 1 in die Queue aufgenommen.

$$- \quad \| \quad (\mathcal{A}(S', 2, 3) \cong 0,05) \quad \| \quad [(\mathcal{A}(S', 1, 2) \cong 1)_1]$$

$$\{(\perp \cong 0), (\mathcal{K}(\perp) \cong 0)\}$$

Unter Verwendung der gerade aus der Queue geholten Gewichtszuweisung $(\mathcal{A}(S', 2, 3) \cong 0,05)$ lassen sich zu diesem Zeitpunkt keine Inferenzregeln anwenden.
Es handelt sich dabei nicht um die Zielaussage der oberen Hierarchieebene, weshalb BASE-Inferenz nicht möglich ist.
Für UP- oder DOWN-Inferenz müssten die Verbundregeln betrachtet werden und damit wäre eine weitere Gewichtszuweisung einer Aussage, deren untere Grenze 1 und obere Grenze 2 sein müsste, notwendig. Doch eine solche wurde bisher noch nicht erzeugt.
In diesem Schritt kann also nicht auf weitere Gewichtszuweisungen geschlossen werden, sondern die gerade betrachtete wird nur zu der Menge aller erzeugten Gewichtszuweisungen hinzugefügt.

$$UP \quad \| \quad (\mathcal{A}(S', 1, 2) \cong 1) \quad \| \quad [(\mathcal{A}(S', 1, 3) \cong 1,35)_{1,35}]$$

$$\{(\bot \cong 0), (\mathcal{K}(\bot) \cong 0), (\mathcal{A}(S', 2, 3) \cong 0,05)\}$$

Die Gewichtszuweisung $(\mathcal{A}(S', 1, 2) \cong 1)$ ermöglicht jetzt in Verbindung mit dem schon vorhandenen $(\mathcal{A}(S', 2, 3) \cong 0,05)$ und der UP-Inferenz den Schluss auf $(\mathcal{A}(S', 1, 3) \cong 1,35)$.

Das Gewicht 1,35 berechnet sich aus der Summe des Gewichtes der Verbundregel $S' \xrightarrow{0,3} S'S'$, die der benutzten Inferenzregel zu Grunde liegt, den Gewichten der kombinierten Gewichtszuweisungen, $(\mathcal{A}(S', 1, 2) \cong 1)$ und $(\mathcal{A}(S', 2, 3) \cong 0,05)$, und dem Gewicht des abstrakten Kontextes $\mathcal{K}(\bot)$: 0,3 + 1 + 0,05 + 0 = 1,35.

Es gibt keine Möglichkeit in diesem Fall DOWN-Inferenz zu verwenden, da dafür noch kein Kontext des selben Hierarchielevels vorhanden ist.

$$BASE \quad \| \quad (\mathcal{A}(S', 1, 3) \cong 1,35) \quad \| \quad [(\mathcal{K}(\mathcal{A}(S', 1, 3)) \cong 0)_{1,35}]$$

$$\{(\bot \cong 0), (\mathcal{K}(\bot) \cong 0), (\mathcal{A}(S', 2, 3) \cong 0,05), (\mathcal{A}(S', 1, 2) \cong 1)\}$$

Die Aussage $\mathcal{A}(S', 1, 3)$ ist die Zielaussage der oberen PCFG. BASE-Inferenz folgert daraus eine Gewichtszuweisung des Kontextes eben dieser Aussage.
Weder UP- noch DOWN-Inferenzen sind in dieser Iteration realisierbar, da $(\mathcal{A}(S', 1, 3) \cong 1,35)$ dabei verwendet werden müsste. Wegen der Maximalität beider Grenzen im Bezug auf das Eingabewort, ist keine Kombination mit einer weiteren Gewichtszuweisung machbar.

$$DOWN \quad \| \quad (\mathcal{K}(\mathcal{A}(S', 1, 3)) \cong 0) \quad \| \quad [(\mathcal{K}(\mathcal{A}(S', 1, 2)) \cong 0,35)_{1,35}, (\mathcal{K}(\mathcal{A}(S', 2, 3)) \cong 1,3)_{1,35}]$$

$$\{(\bot \cong 0), (\mathcal{K}(\bot) \cong 0), (\mathcal{A}(S', 2, 3) \cong 0,05), (\mathcal{A}(S', 1, 2) \cong 1), (\mathcal{A}(S', 1, 3) \cong 1,35)\}$$

DOWN-Inferenz erlaubt es nun weitere Kontexte für die obere Hierarchieebene zu generieren, wobei $(\mathcal{A}(S', 1, 2) \cong 1)$ und $(\mathcal{A}(S', 2, 3) \cong 0,05)$ als Gewichtszuweisungen der selben Hierarchiestufe, zu der auch $(\mathcal{K}(\mathcal{A}(S', 1, 3)) \cong 0)$ gehört, in die Regel einfließen.
Sowohl aus $(\mathcal{A}(S', 1, 2) \cong 1)$ als auch aus $(\mathcal{A}(S', 2, 3) \cong 0,05)$ entsteht in Folge der Inferenz eine neue Gewichtszuweisung für den jeweils entsprechenden Kontext, die dann beide in die Queue eingefügt werden.

$$UP \quad \| \quad (\mathcal{K}(\mathcal{A}(S', 1, 2)) \cong 0,35) \quad \| \quad [(\mathcal{A}(A, 1, 2) \cong 1)_{1,35}, (\mathcal{K}(\mathcal{A}(S', 2, 3)) \cong 1,3)_{1,35}]$$

$$\{(\bot \cong 0), (\mathcal{K}(\bot) \cong 0), (\mathcal{A}(S', 2, 3) \cong 0,05), (\mathcal{A}(S', 1, 2) \cong 1), (\mathcal{A}(S', 1, 3) \cong 1,35), (\mathcal{K}(\mathcal{A}(S', 1, 3)) \cong 0)\}$$

Auf Grundlage der Terminalregel $A \xrightarrow{1} a$ der unteren PCFG wird eine UP-Inferenzregel auf $(\mathcal{K}(\mathcal{A}(S', 1, 2)) \cong 0,35)$ eingesetzt.
Eine DOWN-Inferenz ist hier nicht durchführbar, da die obere Grenze nur um genau eins größer ist, als die untere Grenze und deshalb keine auf Verbundregeln basierenden DOWN-Inferenzregeln denkbar sind.

$$- \quad \| \quad (\mathcal{A}(A,1,2) \cong 1) \quad \| \quad [(\mathcal{K}(\mathcal{A}(S',2,3)) \cong 1,3)_{1,35}]$$

$$\{(\bot \cong 0), (\mathcal{K}(\bot) \cong 0), (\mathcal{A}(S',2,3) \cong 0,05), (\mathcal{A}(S',1,2) \cong 1), (\mathcal{A}(S',1,3) \cong 1,35),$$
$$(\mathcal{K}(\mathcal{A}(S',1,3)) \cong 0), (\mathcal{K}(\mathcal{A}(S',1,2)) \cong 0,35)\}$$

Dies ist nun wieder ein Schritt, in dem keine Inferenz stattfindet.
BASE-Inferenz geht nicht, weil die Gewichtszuweisung nicht die Zielaussage einer Ebene enthält. Die Anwendung von UP-Inferenz scheitert am Nichtvorhandensein einer zweiten Gewichtszuweisung für die untere Hierarchieebene. Die selbe Begründung gilt für DOWN-Inferenz.

$$UP \quad \| \quad (\mathcal{K}(\mathcal{A}(S',2,3)) \cong 1,3) \quad \| \quad [(\mathcal{A}(A,2,3) \cong 0,05)_{1,35}, (\mathcal{A}(S,2,3) \cong 0,7)_2]$$

$$\{(\bot \cong 0), (\mathcal{K}(\bot) \cong 0), (\mathcal{A}(S',2,3) \cong 0,05), (\mathcal{A}(S',1,2) \cong 1), (\mathcal{A}(S',1,3) \cong 1,35),$$
$$(\mathcal{K}(\mathcal{A}(S',1,3)) \cong 0), (\mathcal{K}(\mathcal{A}(S',1,2)) \cong 0,35), (\mathcal{A}(A,1,2) \cong 1)\}$$

Mittels UP-Inferenz auf Basis der Terminalregeln $A \xrightarrow{0,05} b$ und $S \xrightarrow{0,7} b$ und der für beide benutzbaren Gewichtszuweisung $(\mathcal{K}(\mathcal{A}(S',2,3)) \cong 1,3)$ wird auf $(\mathcal{A}(A,2,3) \cong 0,05)$ und $(\mathcal{A}(S,2,3) \cong 0,7)$ geschlossen.
Die Gewichte ergeben sich hierbei direkt aus denen der jeweiligen Grammatikregel und die Prioritäten, die über die Position in der Queue entscheiden, durch zusätzliche Addition mit dem Gewicht 1,3 des abstrakten Kontextes.

$$UP \quad \| \quad (\mathcal{A}(A,2,3) \cong 0,05) \quad \| \quad [(\mathcal{A}(S,1,3) \cong 1,57)_{1,57}, (\mathcal{A}(S,2,3) \cong 0,7)_2]$$

$$\{(\bot \cong 0), (\mathcal{K}(\bot) \cong 0), (\mathcal{A}(S',2,3) \cong 0,05), (\mathcal{A}(S',1,2) \cong 1), (\mathcal{A}(S',1,3) \cong 1,35),$$
$$(\mathcal{K}(\mathcal{A}(S',1,3)) \cong 0), (\mathcal{K}(\mathcal{A}(S',1,2)) \cong 0,35), (\mathcal{A}(A,1,2) \cong 1), (\mathcal{K}(\mathcal{A}(S',2,3)) \cong 1,3)\}$$

Unter Verwendung der Regel $S \xrightarrow{0,52} AA$, der Gewichtszuweisungen
$(\mathcal{K}(\mathcal{A}(S',1,3)) \cong 0)$, $(\mathcal{A}(A,1,2) \cong 1)$ und $(\mathcal{A}(A,2,3) \cong 0,05)$ erhält man bei UP-Inferenz die neue Gewichtszuweisung $(\mathcal{A}(S,1,3) \cong 1,57)$, deren Gewichtswert sich aus der Summe der benutzten Objekte ohne Berücksichtigung des Gewichtes des abstrakten Kontextes: 0,52 + 1 + 0,05 = 1,57.

Im nächsten Schritt wird $(\mathcal{A}(S,1,3) \cong 1,57)$ aus der Queue genommen.

Da dies eine Gewichtszuweisung für die Zielaussage der unteren Grammatikebene ist, bricht der Algorithmus an dieser Stelle das Verfahren erfolgreich ab.

3.4.3 Richtigkeit des Ergebnisses

Nun soll kurz erläutert werden, dass das berechnete Ergebnis auch wirklich dem Gewicht des wahrscheinlichsten Parsebaumes für das Wort "ab" entspricht.
Hierzu werden zuerst die beiden einzig möglichen Parsebäume für das Eingabewort und die vorgegebene Grammatik gezeigt:

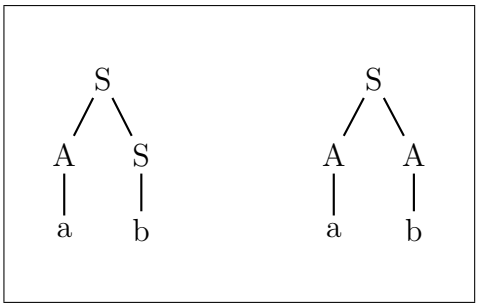

Abbildung 3.3: Parsebäume für Wort ab

Das Gewicht des linken Baumes ergibt sich aus dem Produkt der Einzelgewichte der Regeln, die in ihm enthalten sind.

$$S \xrightarrow{0,3} AS \qquad A \xrightarrow{1} a \qquad S \xrightarrow{0,7} b$$

$$0,3 + 1 + 0,7 = 2$$

Genauso wird mit dem rechten Parsebaum vorgegangen:

$$S \xrightarrow{0,52} AA \qquad A \xrightarrow{1} a \qquad A \xrightarrow{0,05} b$$

$$0,52 + 1 + 0,05 = 1,57$$

Der rechte Parsebaum ist dementsprechend der wahrscheinlicherere, da er das kleinere Gewicht hat und die über den negierten Logarithmus aus den Wahrscheinlichkeitenswerten berechneten Gewichte für umso größere Wahrscheinlichkeiten stehen, desto kleiner sie selber sind.

3.4.4 Gewinnung des Parsebaumes

Das HA*-Parsing-Verfahren hat als Ergebnis lediglich das Gewicht des wahrscheinlichsten Parsingbaumes für das Eingabewort bezüglich der vorgegebenen PCFG ausgerechnet. Wenn nun der Parsebaum, der zu diesem Gewichtswert gehört, gefunden werden soll, muss die Berechnung zurückverfolgt werden.
Das grundsätzliche Vorgehen für eine gerade betrachtete Gewichtszuweisung liegt im Auffinden der Grammatikregel, auf deren Grundlage die Inferenzregel basierte, deren Anwendung zur Produktion dieser Gewichtszuweisung führte.
Dieses Verfahren startet mit der Gewichtszuweisung der Zielaussage der niedrigsten Hierarchieebene und und geht dann jeweils rekursiv für diejenigen Gewichtszuweisungen weiter, die zur Erzeugung der anderen eingesetzt wurden.
Beim vorherigen Beispiel startet man zur Gewinnung des Parsebaumes mit der Gewichtszuweisung $(\mathcal{A}(S, 1, 3) \cong 1, 57)$.

Diese wurde durch UP-Inferenz gewonnen, der die Regel $S \xrightarrow{0,52} AA$ zugrunde lag. Damit steht fest, dass in dem wahrscheinlichten Parsebaum das Startsymbol zu zwei Vorkommen des Nichtterminals A wird. Des Weiteren kann aus der Inferenzregel abgelesen werden mit welchen Gewichtszuweisungen weiterverfahren soll. Das sind in diesem Fall $(\mathcal{A}(A, 1, 2) \cong 1)$ und $(\mathcal{A}(A, 2, 3) \cong 0, 05)$. Nun muss also genauso mit diesen die Suche weitergehen.

Die Gewichtszuweisung $(\mathcal{A}(A, 1, 2) \cong 1)$ entstand als Folge von UP-Inferenz bezüglich $A \xrightarrow{1} a$. Im gesuchten Parsebaum ist es deshalb diese Regel, die das erste Symbol des Eingabewortes erzeugt.

Die zweite noch zu verfolgende Gewichtszuweisung war $(\mathcal{A}(A, 2, 3) \cong 0, 05)$. Die Regel $S \xrightarrow{0,7} b$ diente als Basis für die auch hier verwendete UP-Inferenz und ist somit für die Erzeugung des zweiten Zeichen des Eingabeworten verantwortlich.

Der gesuchte Parsebaum ist damit bereits vollständig bestimmt.

4 IMPLEMENTIERUNG

4.1 EINLEITUNG

In diesem Kapitel soll nun konkret auf die Implementierung des zuvor formal vorgestellten Verfahrens eingegangen werden.
Zu diesem Zweck werden nach einer kurzen Einleitung die wichtigsten Klassen des Java-Programms und deren Aufgaben beschrieben. Danach erfolgt eine Erläuterung des Programmablaufes und schließlich eine Zusammenfassung der Schwierigkeiten, die sich im Laufe der Implementierungsphase ergeben haben.

Eine der gestellten Aufgaben dieser Bachelorarbeit war es, das HA*-Parsing Verfahren für PCFG zu implementieren. Dabei sollte das Programm auf Basis einer Grammatik arbeiten, die vom Berkeley-Parser zuvor erstellt wurde. Eine solche PCFG wird in Form von drei .txt Dateien vorgegeben:

- *.grammar.txt
- *.lexicon.txt
- *.splits.txt

Dabei steht * für einen beliebigen Namen der Grammatik.
Die erstgenannte Datei enthält zeilenweise Regeln der $A \xrightarrow{p} BC$, die folgendermaßen kodiert sind:

[Nichtterminal] -> [Nichtterminal] [Nichtterminal] [Wahrscheinlichkeit]

Des Weiteren sind in dieser Datei Kettenregeln, also Regeln der Form $A \xrightarrow{p} B$, gespeichert, deren Kodierung diesem Muster entspricht:

[Nichtterminal] -> [Nichtterminal] [Wahrscheinlichkeit]

Die zweite Datei "*.lexicon.txt" beinhaltet Regeln des Schemas $A \xrightarrow{p} b$. Es handelt sich hierbei um Terminalregeln mit deren Anwendung man genau ein Nichtterminalsymbolvorkommen durch ein Terminalsymbol ersetzen kann. Allerdings sind die einzelnen Regeln hier nicht zeilenweise abgelegt, sondern immer bezüglich einer Nichtterminalsymbol-Terminalsymbol-Kombination gruppiert. Da es mehrere Varianten eines Nichtterminals geben kann, die vom Berkeley-Parser durchnummeriert werden, gibt es für jede dieser Varianten eine Terminalregel. Die Wahrscheinlichkeiten der Regeln sind dann nach der Nennung des Nichtterminal- und Terminalsymbols in einer Klammer aufgelistet.

[Nichtterminal] [Terminal] [[Wahrscheinlichkeit 1], ... , [Wahrscheinlichkeit n]]

Die dritte Datei "*.splits.txt" liefert die letzten noch fehlenden Informationen, die für das Erstellen einer PCFG-Hierarchie notwendig sind.
Dort ist die Abstraktion eines jeden Nichtterminalsymbols mittels einer speziellen Kodierung hinterlegt.
Diese entsteht beim Berkeley-Parser während eine Grammatik gelernt wird. Dem coarse-to-fine-Prinzip folgend wird dabei die Menge der Nichtterminale so weit vergrößert, bis ausreichend viele verschiedene Nichtterminale entstanden sind. Jedes Mal, wenn mehr Nichtterminale benötigt werden, wird ein bereits vorhandenes Nichtterminalsymbol gesplittet. Das bedeutet, dass aus einem Nichtterminalsymbol mehrere Varianten gemacht werden und die Nummerierung aller dann existierenden Varianten dieses Nichtterminals entsprechend angepasst wird. Wenn es beispielsweise schon die drei Varianten A0, A1 und A2 des Nichtterminalsymbols A gibt und A0 dann auf zwei Varianten aufgesplittet werden soll, bekommt die erste dieser Varianten den Namen A0 und die zweite A1. Weil es aber schon ein A1 gab, muss dessen Name zu A2 aktualisiert werden und aus dem bisherigen A2 wird A3. Nach dem Split gibt es vier Varianten des Nichtterminals A: A0, A1, A2, A3.

Die Kodierung der Abstraktionen in der Datei soll nun an einem Beispiel verdeutlicht werden.

NNPS (0 (0 (0 0 1) (1 2)) (1 (2 3)))

Diese Zeile aus der "*.splits.txt" Datei kodiert die Varianten des Nichtterminalsymbols "NNPS" in den jeweiligen Hierarchieebenen. Um dies besser lesbar zu machen, kann man einen entsprechenden Baum zeichnen, der als Wurzel, und damit in der gröbsten Hierarchiestufe, nur eine Variante des Nichtterminals (NNPS0) hat. Jede Ebene dieses Baumes zeigt die Varianten einer Hierarchieebene auf und die Splits werden durch Verzweigungen dargestellt. Die in der Abbildung gezeigte Abstraktionshierarchie für das Nichtterminalsymbol NNPS hat vier Ebenen. Es lässt sich gut erkennen, wie die Nummerierung realisiert wird. Wenn eine Variante aufgesplittet wird, werden die daraus erzeugten Varianten fortlaufend von der Nummer der Variante durchnummeriert, aus der sie entstanden sind. Da es für alle Nummern größer als die der gesplitteten Variante gegebenenfalls schon Varianten gab, müssen dann deren Nummern entsprechend der Anzahl der erzeugten Varianten erhöht werden, sodass schließlich jede Variante wieder eine andere Nummer hat und die Nummerierung innerhalb einer Hierarchieebene lückenlos ist.

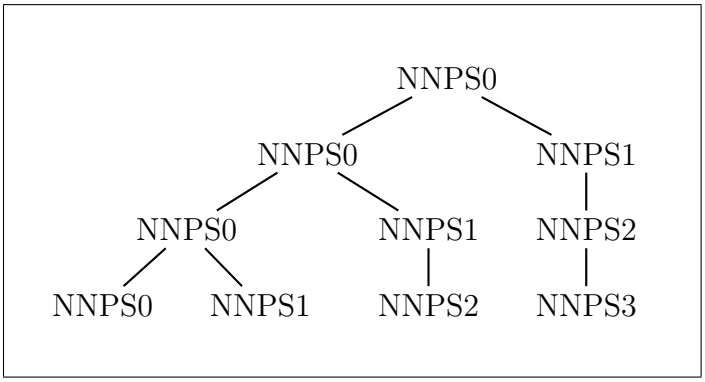

Abbildung 4.1: Abstraktionshierarchie des Nichtterminalsymbols NNPS

4.2 WICHTIGSTE KLASSEN

Um im hierauf folgenden Abschnitt ein Verständnis für das bei der Implementierung entstandene Programm zu ermöglichen, sollen nun zuerst die wichtigsten Java-Klassen genannt und ihre jeweilige Funktion im Programm vorgestellt werden.

PCFG

Ein Objekt der Klasse PCFG repräsentiert eine probabilistische kontextfreie Grammatik. Hierzu beinhaltet es unter anderem alle notwendigen Informationen von den Nichtterminalsymbolen über die Terminalsymbole und die Regeln bis hin zum Startsymbol.
Die Nichtterminalsymbole können entweder explizit zu der Grammatik hinzugefügt werden (Methode addNonTerminal(String)) oder sie werden implizit in die Grammatik integriert, wenn eine neue Regel aufgenommen wird, die ein noch nicht in der Grammatik vorhandenes Nichtterminalsymbol enthält. In beiden Fällen wird das neue Nichtterminalsymbol zum einen als String gespeichert, aber erhält gleichzeitig auch eine Nummer, welche einer fortlaufenden Nummerierung aller Nichtterminalsymbole innerhalb einer PCFG entspricht. Nach dem Hinzufügen eines neuen Nichtterminals zu einer Grammatik wird dieses nur noch in Form von seiner Nummer verwendet. Dies bietet sich an, da für alle folgenden Berechnungen der eigentliche Name eines Nichtterminals nicht mehr von Bedeutung ist, sondern nur eine Unterscheidung der Nichtterminals untereinander notwendig ist. Insbesondere bei Vergleichen wird somit Rechenzeit gespart, da der Vergleich zweier Zahlen schneller ist, als von zwei Strings. Um trotzdem wieder von der Nummer eines Nichtterminals zurück auf seinen tatsächlichen Namen schließen zu können, gibt es eine Abbildung, die dies effizient ermöglicht.
Analog zu diesem Vorgehen wird mit Terminalsymbolen verfahren. Auch bei ihnen wird intern, nach der Aufnahme in die PCFG, für Berechnungen nur noch die entsprechende laufende Nummer als Repräsentation genutzt.
Die Klasse PCFG unterstützt drei verschiedene Regeltypen. Neben den beiden Möglichkeiten, die die Chomsky-Normalform bietet, stellen Kettenregeln den dritten Typ dar. Für die Regeln gibt es nun jeweils mehrere Varianten sie zu einer Grammatik hinzuzufügen. Je nachdem inwiefern eine Regel schon der intern in der PCFG-Klasse verwendeten Form entspricht, kann eine passende Methode zum Anfügen an die Grammatik gewählt werden. Letztendlich wird die als Parameter übergebene Regel aber immer zuerst in das interne Format gebracht und danach in dieser Form (CNFRuleBaseInternal, CNFRuleCompoundInternal bzw. RuleChainInternal) in die jeweiligen Listen einsortiert.
Zu diesen Listen gehört je Regeltyp eine einfache Liste, die alle vorhandenen Regeln des entsprechenden Typs enthält. Des Weiteren gibt es noch zusätzliche Datenstrukturen, die die Regeln bezüglich ihrer Symbole sortiert speichern. Diese stellen im Kontext der eigentlichen PCFG Redundanz dar, werden aber später für das Parsing eine entscheidende Rolle spielen,
da es dort von Vorteil ist, immer direkt auf die Regeln zugreifen zu können, die gerade relevant sind und diese nicht erst aus allen Regeln des gleichen Typs zur Laufzeit heraussuchen
zu müssen.
Das Startsymbol einer PCFG wird explizit festgelegt (Methode setStartSymbol(String)). Falls das Symbol noch nicht in als Nichtterminal in der Grammatik enthalten sein sollte, wird es zuerst als solches angefügt und danach als Startsymbol bestimmt.
Die Methode aggregateRules() wird später wichtig, wenn von einer PCFG aus eine Abstraktion erstellt werden soll. Sie ermöglicht es die in einem PCFG-Objekt gekapselten Regeln auf Duplikate bezüglich der Symbole zu überprüfen und diese zu einer Regel zu identifizieren. Dazu werden alle Regeln eines Typs mit einer Hashfunktion auf eine Zahl abgebildet. Bei dieser Abbildung sind nur die in der Regel vorkommenden Symbole relevant, nicht aber ihre Wahrscheinlichkeiten. Wie zuvor beschrieben wird aus einer Menge von Regeln mit dem selben Hashwert nur diejenige übernommen, die den kleinsten Wert in ihrem probability-Attribut gespeichert hat.

PCFGHierarchy

Die Klasse PCFGHierarchy ist dafür da, mehrere PCFGs gesammelt zu halten, die zusammen eine Hierarchie bezüglich einer Abstraktionsfunktion darstellen. Neben einer Liste von PCFGs enthält ein Objekt der Klasse PCFGHierarchy noch die verwendete Abstraktionsfunktion und ihre Umkehrabbildung. Beide werden mit Hilfe von Listen abgespeichert.
Das Attribut absAtLevel verweist auf eine Liste von Abstraktionsfunktionen. Jede dieser Funktionen ist für die Abbildung der Nichtterminale einer Ebene der Hierarchie verantwortlich. Dementsprechend gibt es genau eine solche Funktion weniger, als es Ebenen in der Hierarchie gibt, da von der gröbsten Ebene aus nicht weiter abstrahiert werden muss. Zu verstehen ist eine Abstraktionsfunktion, die von einer Liste repräsentiert wird, so, dass jedem Nichtterminal einer Ebene eine Nummer eines Nichtterminals der darüberliegenden Ebene zugeordnet wird. Wenn an Position 2 der Liste also eine 5 gespeichert ist, wird das Nichtterminal mit der Nummer 2 auf das Nichtterminal mit der Nummer 5 der nächsthöheren Ebene abgebildet wird. Die Liste der Abstraktionsfunktion einer Ebene hat folgerichtig so viele Elemente, wie es Nichtterminalsymbole auf der selben Ebene gibt.
Ganz ähnlich dazu funktioniert es mit dem Attribut inversAbsAtLevel, nur dass hier nicht immer nur auf einen einzelnen Wert abgebildet wird, sondern auf eine Liste von Werten. Das ist zwingend notwendig, da die Liste, auf die abgebildet wird, alle Nichtterminale der nächsttieferen Ebene enthalten soll, die auf das selbe Nichtterminal der Ebene, in der die Liste steht, abstrahiert werden.
Die bedeutendste Methode der Klasse PCFGHierarchy ist computeFromBaseGrammarAndAbstraction(). Sie setzt eine vollständige PCFG und alle Abstraktionsfunktionen als gegeben voraus und berechnet auf Basis dessen für alle noch unvollständigen PCFGs, denen nur die Nicht- und Terminalsymbole bekannt sind, ihre Regeln. Dazu werden, Ebene für Ebene, alle Regeln abstrahiert, indem auf ihre Nichtterminale die der Ebene zugehörige Abstraktionsfunktion angewandt wird. Im Anschluss dessen ist der Aufruf der aggregateRule() Methode in der mit Regeln aufgefüllten PCFG von großer Bedeutung für die Korrektheit der PCFG-Hierarchie.

GrammarReader

Die Klasse GrammarReader ist als Schnittstelle zum Berkeley-Parser anzusehen. Sie macht die vom Berkeley-Parser erzeugten PCFGs erst für das hier vorgestellte Programm verwendbar.
Die drei zu einer PCFG des Berkely-Parsers gehörigen Dateien werden, beginnend mit den Splits, nacheinander eingelesen. Aus dieser Datei extrahiert die Methode readAbstraction() sowohl die Nichtterminalsymbole einer jeden Hierarchieebene, als auch die Abstraktionsfunktionen. Dafür muss die Datei zeilenweise gelesen und geparst werden. Die enthaltenen Nichtterminale werden in die entsprechende PCFG hinzugefügt und die Abstraktion eines jeden Nichtterminals wird zuerst als Baum dargestellt (Klassen AbstractionTree und AbstractionNode), aus dem dann die entscheidenden Informationen in die passenden Listen der Abstraktionsfunktionen eingehen, welche in dem über das Attribut grammarHierarchy zu findenden Objekt der Klasse PCFGHierarchy gespeichert sind.
Danach kann mit dem Einlesen der Regeln aus den anderen beiden Dateien begonnen werden. Auch hier geht das Programm zeilenweise durch die Datei, parst die gerade gelesene Zeile zu einem Regelobjekt und fügt dieses zu der PCFG der niedrigsten, feinsten Hierarchieebene hinzu.

HAstarParser

In der Klasse HAstarParser findet die eigentliche Implementierung des HA*-Parsings von PCFGs statt. Die meisten anderen Klassen des Programms leisten nur Vorarbeit, um schließlich hier das Parsing zu ermöglichen, beziehungsweise bilden eine Umgebung, in der das Parsing abläuft.

Gestartet wird eine Instanz dieser Klasse mit einer fertigen PCFG-Hierarchie und einem Eingabewort, für das sofort geprüft wird, ob jedes seiner Terminale in der feinsten PCFG der Hierarchie enthalten ist. Denn falls dies nicht gegeben wäre, könnte auch kein Parsebaum gefunden werden. Ein sehr wichtiger Bestandteil dieser Klasse sind die Datenstrukturen, die diejenigen Regeln halten, für deren Nichtterminale schon ein Kontext einer Aussage gefunden wurde, dessen Nichtterminal mit dem der Regel übereinstimmt. Diese Strukturen werden bei der Realisierung der Inferenzregeln in der Methode run() intensiv verwendet, um für jeden möglichen Fall sofort die richtigen Regeln zur Verfügung zu haben, ohne erst zur Laufzeit nach diesen Suchen zu müssen. Jedes Mal, wenn eine Gewichtszuweisung (Klasse WeightAssignment) beim Parsing aus der Queue genommen wird, werden die entsprechenden Listen der Datenstrukturen aktualisiert. Wird also beispielsweise eine Gewichtszuweisung, deren Aussage das Nichtterminal A hat, neu gefunden, muss unter anderem für alle Kettenregeln, die auf der linken Seite ebenfalls das Nichtterminal A haben, gespeichert werden, dass es ab jetzt einen Kontext mit einer Aussage mit eben diesem Nichtterminal gibt, damit im Folgenden beim Versuch entsprechende Inferenzregeln anzuwenden, auch diese Kettenregeln betrachtet werden. Diese Information wird in Form des Einsortierens der Regeln in die Datenstrukturen festgehalten.
Das Attribut priorityQueue verweist auf eine Queue, in die abgeleitete Gewichtszuweisungen eingefügt werden. Innerhalb der Queue sind diese durch den Prioritätswert, der ihnen über die Inferenzregel, mit der sie gewonnen wurden, zugeordnet wurde, geordnet.
Der Kern der HAstarParser Klasse ist die Methode run(). In ihr sind die Inferenzregeln, und mit ihnen der Ablauf des Parsings, verwirklicht. Hierfür wird für jede Gewichtszuweisung, die von der Queue genommen wird, eine mehrstufige Fallunterscheidung vorgenommen. Denn wie und ob welche Inferenzregeln angewandt werden, hängt davon ab, ob es sich um eine Gewichtszuweisung einer Aussage oder eines Kontextes einer Aussage handelt, aus welcher Hierarchieebene die Aussage stammt und wie die Grenzen der Aussage sind. Unter bestimmten Bedingungen kann so die Anwendung von Inferenzregeln schon nur anhand der gerade zu bearbeitenden Aussage ausgeschlossen werden. Zum Beispiel kann mit den Inferenzregeln der Art UP bezüglich der Terminalregeln einer PCFG auf keine neue Gewichtszuweisung geschlossen werden, falls die zuvor von der Queue genommene Gewichtszuweisung nicht für einen Kontext, sondern für eine Aussage, ist. Das liegt dann daran, dass die Inferenzregeln in diesem konkreten Fall nur einen Kontext einer Aussage als Vorbedingung haben, sodass eine neue Gewichtszuweisung einer Aussage keine neuen Resultate ermöglicht, da sie nicht die Vorbedingung der Inferenzregel erfüllen kann. Somit braucht der Algorithmus unter diesen Gegebenheiten gar nicht erst prüfen, ob Inferenzregeln des Typs UP für Terminalregeln zu neuen Gewichtszuweisungen führen würden.
Ein zweiter, beispielhafter Fall, in dem ganz bestimmte Inferenzregeln übersprungen werden können, ist folgender: Falls die gerade betrachtete Gewichtszuweisung zu einem Kontext einer Aussage gehört und versucht werden soll mittels Inferenzregeln, die sich auf Verbundregeln, also Regeln, die links ein Nichtterminal und rechtsseitig zwei Nichtterminale haben, beziehen, auf neue Gewichtszuweisungen zu schließen, kann trivialerweise dieser Fall schon ausgeschlossen werden, wenn die Grenzen der Aussage des Kontextes nur um eins voneinander abweichen, die linke Grenze also nur um eins kleiner ist, als die rechte Grenze. Das bedeutet nämlich, dass nur ein einziges Zeichen des Eingabewortes betroffen ist, was zur Folge hat, dass gar nicht erst nach passenden, schon vorhandenen Gewichtszuweisungen als linke beziehungsweise rechte Seite der Verbundregel gesucht werden muss, da es in genau diesem Fall keine geben kann. Das ist in der Tatsache begründet, dass passende Gewichtszuweisungen zu Aussagen gehören müssten, die zusammen nur dieses eine Zeichen des Eingabewortes abdecken. Dies kann allerdings nie sein, da die rechte Grenze einer Aussage immer mindestens um eins größer ist, als die linke Grenze und damit zwei, mittels einer Inferenzregel und ihr zugrunde liegenden Verbundregel verknüpften Gewichtszuweisungen mindestens zwei Zeichen des Eingabewortes betreffen.
Auf diese Art und Weise wird in der Methode run() gesteuert, mit welchen Inferenzregeln im konkreten Fall versucht wird auf neue Gewichtszuweisungen zu schließen.
Abgebrochen wird das ganze Verfahren entweder, wenn die Queue leer ist oder falls eine Gewichtszuweisung für die Zielaussage der untersten Hierarchieebene von der Queue geholt wird. In zweitgenannter Situation ist das Parsing erfolgreich gewesen. Es gibt also eine Ableitung des Eingabewortes bezüglich der PCFG auf unterster Hierarchieebene. Dann wird rückwärts mittels der während des Parsings gespeicherten Objekte der Klassen WeightAssignmentOriginCNF/-Chain ein Baum als Ausgabe berechnet, von dem sich die in der berechneten Ableitung angewandten Regeln ablesen lassen. Dies ist auf Grund des Verfahrens der Parsebaum des Eingabewortes bei der gegebenen PCFG mit der größten Wahrscheinlichkeit. Die Erzeugung dieses Baumes erfolgt in der Methode getParseTree().

4.3 PROGRAMMABLAUF

In diesem Abschnitt soll der gesamte Programmablauf vom Einlesen einer Grammatik bis hin zur Ausgabe des wahrscheinlichsten Parsebaumes beschrieben werden.
Das Programm startet mit der Erzeugung eines Objektes der Klasse GrammarReader. In diesem wird daraufhin die Methode GrammarReader.run() aufgerufen, die zuerst bestimmt wie viele Ebenen die Hierarchie haben wird und dann alle notwendigen Nichtterminalsymbole aus der Datei "[Grammatikname].splits.txt" extrahiert und diese in Objekte der Klasse PCFG speichert, wobei für jede Hierarchieebene genau ein solches Objekt angelegt wird. In dem selben Schritt werden die Abstraktionsfunktionen bestimmt und zusammen mit den noch unvollständigen PCFGs in einer Instanz der Klasse PCFGHierarchy gruppiert. Zusätzlich wird pro PCFG das Nichtterminal "ROOT_0" ausgenommen, da dies bei den Grammatiken, die vom Berkeley-Parser erzeugt wurden, standardmäßig das Startsymbol ist.
Nach der Ausführung der GrammarReader.readAbstraction() Methode wird nun die GrammarReader.readRules() Methode gestartet. Sie greift auf die Dateien "[Grammatikname].grammar.txt" und "[Grammatikname]. lexicon.txt" zu und liest alle Regeln daraus aus. Diese Verbund-, Terminal- und Kettenregeln werden der PCFG auf dem untersten Hierarchielevel zugeordnet. Allerdings werden Kettenregeln, die von einem Nichtterminal zum selben führen, nicht in die Grammatik übernommen, da sie später beim Parsing nicht verwendet werden würden. Außerdem wird in dieser Methode noch in jeder Grammatik die selbe Anzahl an Terminalsymbolen festgelegt, da diese von der Abstraktionsfunktion unbeeinflusst bleiben und somit für alle Ebenen gleich sind.
Nun ist der erste Abschnitt auf dem Weg zum erfolgreichen Parsing abgeschlossen. Alle durch die Grammatik vorgegebenen Informationen wurden eingelesen und in entsprechende Formate zur späteren Weiterverwendung gebracht. Die feinste PCFG ganz unten in der Hierarchie ist vollständig und die darüberliegenden haben schon sowohl ihre Nichtterminale, als auch die Terminale erhalten und ihre Startsymbole sind bestimmt.
Allerdings fehlen ihnen noch sämtliche Regeln. Dieser Mangel wird durch die Ausführung der Methode PCFGHierarchy.computeFromBaseGrammarAndAbstraction() auf dem zuvor erzeugten PCFG-Hierarchy Objekt beseitigt. Die Methode berechnet auf Grundlage der soweit erzeugten PCFGs und der Abstraktionsfunktionen die noch fehlenden Regeln der PCFGs. Dabei werden die Nichtterminale einer jeden Regel durch die Abstraktionsfunktion der jeweiligen Ebene auf Nichtterminale der nächsthöheren Ebene abgebildet. Falls es danach Regeln geben sollte, die sich nur im Wahrscheinlichkeitswert unterscheiden, wird nur eine dieser Regeln übernommen. Um die Richtigkeit der Heuristik sicherzustellen muss hierbei immer die Regel mit dem kleinsten Wert im Attribut probability gewählt werden.
Jetzt ist die Grammatikhierarchie auf allen Ebenen vollständig und bereit für ihre Verwendung beim Parsing eines Wortes. Dieses Eingabewort soll nun vom Nutzer über die Konsole angegeben werden. Diese beiden Paramater, PCFG-Hierarchie und Eingabewort, dienen dann zur Erstellung eines Objektes der Klasse HAstarParser, deren Methode HAstarParser.run() nach einigen Initialisierungen automatisch gestartet wird, womit der eigentliche Parsingprozess beginnt.
Angefangen wird hierbei mit der Initialisierung der priorisierten Queue. Für alle Terminalregeln auf der höchsten Hierarchieebene können entsprechende Inferenzregeln des Typs UP angewandt werden, vorausgesetzt diese Terminalregeln haben ein Terminal auf ihrer rechten Seite, das ein Vorkommen im Eingabewort hat.
Danach geht das Programm in eine while-Schleife, die erst dann abbricht, wenn kein Element mehr in der Queue vorhanden ist oder die Zielaussage der untersten Ebene gefolgert wurde. Innerhalb dieser Schleife wird nun in jeder Iteration zunächst das vorderste Element von der Queue geholt und für diese Gewichtszuweisung überprüft, ob bisher schon eine Gewichtszuweisung mit den, bis auf das Gewicht, gleichen Parametern gefunden worden ist. Wenn dies der Fall ist, wird mit der nächsten Iteration begonnen und wieder eine Gewichtszuweisung von der Queue geholt. Andernfalls wird die Gewichtszuweisung in gespeichert und Datenstrukturen, die Auskunft über das Vorhandensein von Kontexten und abstrakten Kontexten für ein bestimmtes Nichtterminal geben, aktualisiert. Alle Regeln, die auf ihrer linken Seite dieses Nichtterminal haben, müssen dann in die passenden Listen einsortiert werden. So ist später ein schneller Zugriff auf sie möglich.
Im Anschluss wird versucht unter Verwendung der aktuellen Gewichtszuweisung und schon vorher erzeugten Gewichtszuweisungen mit den Inferenzregeln weitere Gewichtszuweisungen

zu folgern. Um dies effizient zu gestalten wurden die verschiedenen Inferenzregeln (BASE, UP, DOWN) noch weiter in speziellere Fälle aufgefächert. So wird in Abhängigkeit der gerade zu bearbeitenden Gewichtszuweisung entschieden welche der Fälle zutreffend sind und dann dementsprechend alle relevanten Möglichkeiten betrachtet, die zu neuen Ergebnissen führen könnten.
Falls das Parsing erfolgreich endet und damit eine Gewichtszuweisung für die Zielaussage der niedrigsten Hierarchieebene gefunden wurde, wird dann mittels der Methode HAstarParser.getParseTree() der Parsebaum berechnet und ausgegeben. Dieser ergibt sich über die Zurückverfolgung der Entstehung der Gewichtszuweisungen und der Grammatikregeln, die der im jeweiligen Schritt verwendeten Inferenzregel zugrunde liegen.

4.4 SCHWIERIGKEITEN

Abschließend zum Implementierungskapitel werden nun im Folgenden noch einige Schwierigkeiten erläutert, die während der Implementierungsphase auftraten.
Zu Beginn der Implementierung musste zu allererst eine Umgebung für das eigentliche Parsing geschaffen werden. Hierzu waren unter andere mehrere Klassen für Grammatiken, deren Regeln oder die Gewichtszuweisungen nötig. Dies alles konnte recht problemlos und zügig geschehen, aber danach sollten die Inferenzregeln implementiert werden. Hierbei musste unbedingt darauf Wert gelegt werden wirklich alle Möglichkeiten zu berücksichtigen und keinen Fall zu übersehen. Wenn es beim Parsing zu Fehlern kam, war es aufwendig diese aufzuspüren, da schon bei kleinen Grammatiken ziemlich viele Inferenzen erfolgen, die alle nachvollzogen werden müssen.
Zudem stellte sich dabei heraus, dass die bisher geschaffenen Klassen noch nicht mächtig genug waren und beispielsweise eine einfache Speicherung der Regeln einer PCFG in Listen nicht befriedigend ist, um das Parsing zu realisieren. Es mussten andere Wege kreiert werden, um auf die Regeln zugreifen zu können und schnell alle Regeln mit einer Gemeinsamkeit, wie dem selben Nichtterminalsymbol auf der linken Seite, ohne Umwege zu erhalten. Im Zuge der Einführung solch neuer Datenstrukturen kamen entsprechende Methoden hinzu, die diese initialisieren und aktuell halten.
Nachdem das HA*-Parsing für PCFGs implementiert und funktionsfähig war, musste das Programm nur noch an die vom Berkeley-Parser erzeugten Grammatiken angepasst werden. Dies war allerdings aufwendiger, als vorher angenommen. Zwar konnten die Regeln relativ direkt eingelesen und umgewandelt werden, doch es gab nun auch Kettenregeln, die bis dahin noch nicht unterstützt wurden. An mehreren Stellen des Programms mussten dafür Änderungen vorgenommen werden. Unter anderem in der Methode HAstarParser.run() mussten nun spezielle Fälle für Kettenregeln eingefügt werden, wobei zum großen Teil auf den Programmcode für Verbundregeln zurückgegriffen werden konnte, der von der Struktur aber schon passend war, aber noch verändert werden musste.
Die größte Schwierigkeit beim Einlesen und anschließenden Einbinden der vorgegebenen Grammatiken in das Programm stellte sich bei der Umwandlung der Abstraktion der Nichtterminale heraus. Wie diese zu lesen und interpretieren ist wurde schnell deutlich, aber sie musste noch auf die schon vorhandene Form der Abstraktionsfunktion im Programm abgebildet werden. Diese Abbildung geschieht jetzt auf ziemlich komplizierten, aber intuitiven Weg über die Erstellung eines Baumes für die Abstraktion jedes Nichtterminals und darauf folgender Extraktion der benötigten Werte, die dann erst in die Listen eingetragen werden, welche intern die Abstraktionsfunktionen repräsentieren. Dies ist sicherlich nicht die effizienteste Möglichkeit das Problem zu lösen, aber eine weitere Optimierung hätte keine großen Auswirkungen auf die gesamte Performance des Programms, da dieser Schritt nur ca. 100 ms und damit vergleichsweise wenig Zeit in Anspruch nimmt.
Erst nach der vollständigen Adaptierung des Programms an die zu benutzenden, externen PCFGs wurde deutlich, auf welchem Niveau sich die Performance bewegt. Da anfangs nur selbst gewählte, sehr kleine PCFGs mit sehr wenig Regel, Nichtterminalsymbolen und Terminalsymbolen zum Testen der Korrektheit des Parsings verwendet wurden, um das ganze Verfahren in jedem Schritt noch überblicken und nachvollziehen zu können, konnte noch keine Aussage über die Parsinggeschwindigkeit bei Verwendung von Grammatiken mit Zehntausenden Regeln gemacht werden. Nach den ersten Tests mit einer solch großen PCFG wurde klar, dass es noch viel Raum für Optimierungen gibt. Schon vorher wurde versucht das Parsing effizient zu gestalten indem Datenstrukturen geschaffen wurden, die beispielsweise die Regeln sortiert vorhalten, sodass bei Bedarf direkt und ohne Suchverfahren auf diese zugegriffen werden kann. Dieses Prinzip ist in der Implementierung insbesondere in der Klasse HAstarParser wiederzufinden und wird dort intensiv angewandt.
In Folge dessen steigt allerdings analog der Speicheraufwand. Um diesen etwas einzudämmen wurden bei der Optimierung des Programms unter anderem neue Klassen für Grammatikregeln eingeführt, deren Namen auf "Internal" enden. Die Objekte dieser Klassen verzichten auf die Klartextnamen der Symbole, die in den Regeln vorkommen, und speichern nur noch deren Nummer. Außer beim Hinzufügen von Regeln zu einer PCFG werden danach grundsätzlich einzig Regeln in Form von Objekten dieser Klassen für die Berechnungen verwendet und erst bei der Berechnung der Ausgabe zurücktransformiert, falls dies erforderlich ist.

4.5 LAUFZEIT

In diesem Abschnitt soll abschließend zum praktischen Teil der Arbeit noch auf die Laufzeit des Programms eingegangen werden.

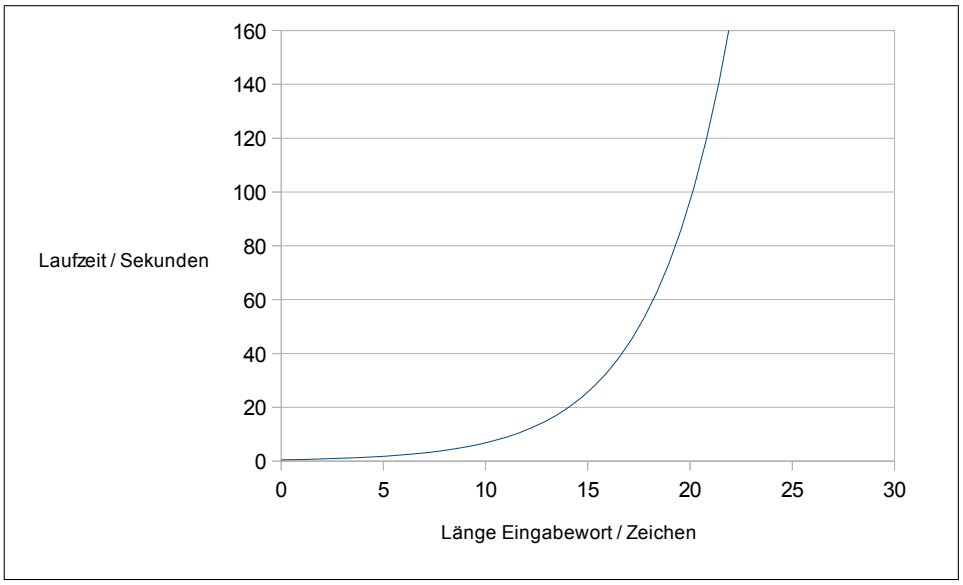

Abbildung 4.2: Laufzeit (in Sekunden) in Abhängigkeit der Länge des Eingabewortes

Zu allererst sollte bezüglich des Diagramms gesagt werden, dass es nur aus Testmessungen entstanden ist und deshalb nur eine ungefähre Vorstellung von der Laufzeit des Parsings, wie es implementiert wurde, geben soll.
Schon auf den ersten Blick ist klar zu erkennen, wie stark der Einfluss der Eingabewortlänge auf die Gesamtlaufzeit ist. Der Grund hierfür wird schnell deutlich, wenn man sich das Beispiel eines Parsingvorganges (3.4.2) angeschaut hat. Schon am Anfang nach der Initialisierung der Queue, wenn der abstrakteste Kontext betrachtet wird, kann durch UP-Inferenz bezüglich jeder Terminalregel und jeder Position, an der das Terminalsymbol der Regel im Eingabewort ein Vorkommen hat, eine Gewichtszuweisung produziert werden. Schon in dieser frühen Phase hat die Länge des Eingabewortes also großen Einfluss auf die Anzahl an folgerbaren Gewichtszuweisungen.
Mit dem Wachstum der Regelanzahl von oben nach unten in der Hierarchie steigen auch die Kombinationsmöglichkeiten nach unten hin immer weiter und schneller an. Daher ergibt sich dieser exponentielle Anstieg der Kurve in Abbildung 4.2. Wenn also nur einige wenige Symbole mehr zu parsen sind, wird das über die gesamte Hierarchie gesehen ein Wachstum der produzierten Gewichtszuweisungen nach sich ziehen.

Beispiele

Für Parsen des kurzen Satzes "The September index was 47.1 % ." benötigt das Programm ca. 2 Sekunden. In dieser Zeit werden ungefähr 31.000 Gewichtszuweisungen erzeugt.

Bei einer Verdoppelung der Eingabewortlänge auf 14 benötigt das Programm schon 26 Sekunden zum Parsen von "The commission is expected to rule on the Braidwood case by year end ." und rund zwei Millionen Gewichtszuweisungen werden mit den Inferenzregeln gefolgert.

Wenn wieder eine Erhöhung der Terminalanzahl in der Eingabe um sieben Zeichen auf 21 erfolgen soll und beispielsweise der Satz "Typically , money-fund yields beat comparable short-term investments because portfolio managers can vary maturities and go after the highest rates ." verwendet wird, dann vergehen ca. 144 Sekunden bis das Ergebnis erfolgreich berechnet wurde. Während des Parsings entstanden dabei ca. 9,84 Millionen Gewichtszuweisungen.

4.6 AUSGABE

Zu jeden Parsingvorgang gibt das Programm, zusätzlich zum eigentlichen Ergebnis, entsprechende Informationen über die wichtigsten Werte aus.
Dies soll nun präsentiert werden.

```
abstraction read 78 ms
rules read 4976 ms
aggregation: 4836
hierarchy built 5554 ms
number of rules on level:
0 - 1117137
1 - 530829
2 - 261259
3 - 127344
4 - 62704
5 - 30611
6 - 15645
```

Abbildung 4.3: Kenndaten der PCFG-Hierarchie und des Einlesens der vorgegebenen PCFG

In Abbildung 4.3 ist der erste Teil der Ausgabe zu sehen, der während und direkt nach dem Einlesen der gegebenen PCFG generiert wird. Es lässt sich ablesen, wie viel Zeit die einzelnen Teilschritte, die zum Einleseprozess gehören, benötigen.
Die Zeitangabe bei "rules read" bezieht sich auf das zeilenweise Lesen der Textdateien, Parsen der Zeilen zu Regelobjekten und das Hinzufügen zu der Ausgangsgrammatik, wobei sowohl Verbund- als auch Terminal- und Kettenregeln beinhaltet sind.
Der Wert unter "abstraction read" beschreibt die Dauer des Einlesens der Abstraktionsfunktion der Nichtterminale. Darin enthalten ist aber beispielsweise auch der Aufwand zum Hinzufügen der Nichtterminale zu den PCFGs der verschiedenen Hierarchieebenen.
Für die Dauer der Erstellung der PCGH-Hierarchie ist ebenfalls ein Wert angegeben. In dieser Zeit wurden die Regeln über ihre Nichtterminale abstrahiert und in den PCFGs eingefügt.
Außerdem gibt die Konsolenausgabe Aufschluss über die Anzahl an Regeln je Hierarchielevel.

```
ROOT
  S
   NP
    NNP Nissan
   VP
    VBD scheduled
    NP
     NP
      DT a
      JJ seven-yen
      JJ interim
      NN dividend
      NN payment
     ADJP
      JJ unchanged
```

Abbildung 4.4: wahrscheinlichster Parsebaum

Wenn das Parsing erfolgreich abgeschlossen werden konnte, wurde ein wahrscheinlichster Parsebaum bezüglich der Eingabeparameter, Eingabewort und Ausgangsgrammatik, gefunden. Dieser

wird in einer Form ausgegeben, die die verschiedenen Ebenen klar voneinander unterscheidbar macht.
Da die intern verwendeten Regeln nur maximal zwei Kinder an einem Knoten erlauben, musste das zuvor erstellte Zwischenergebnis erst in diese, hier zu sehende, Form transformiert werden.

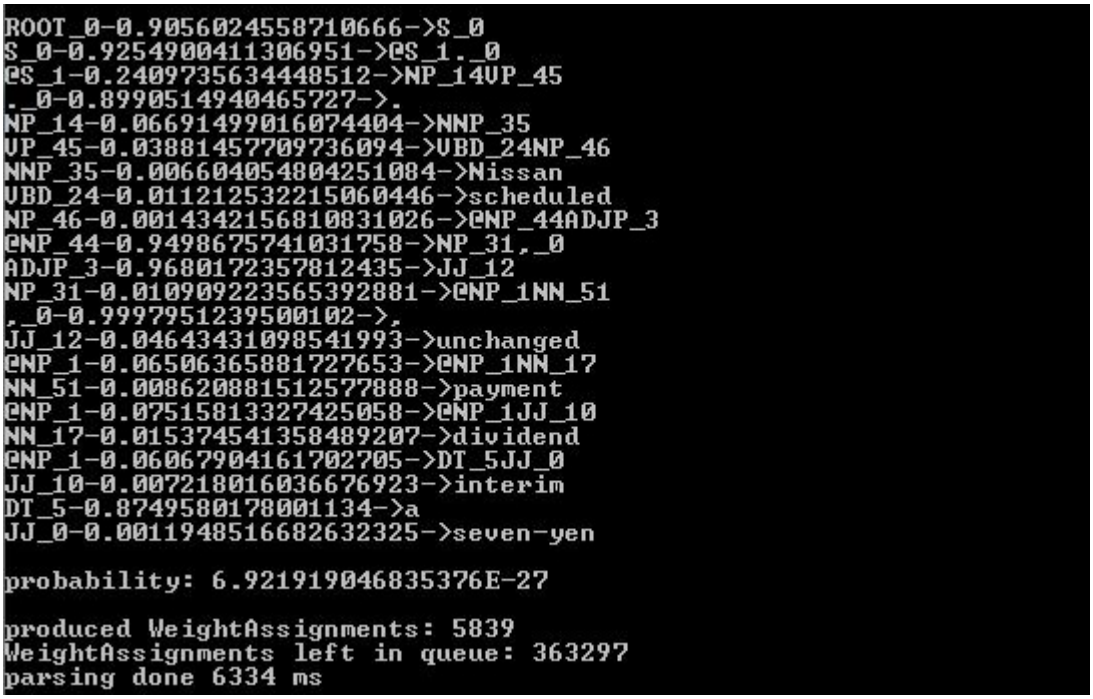

Abbildung 4.5: berechnete Wahrscheinlichkeit, Regelanwendungen und Parsingleistung

In diesem Teil der Ausgabe sind einige weitere Informationen enthalten.
Im oberen Teil von Abbildung 4.5 sieht man die Regelanwendungen, welche in BFS-Manier ausgegeben werden.
Viel interessanter ist aber wohl der nächste Wert: die berechnete Wahrscheinlichkeit des wahrscheinlichsten Parsebaumes. Sie wurde zuvor aus dem Gewicht der Zielaussage der niedrigsten PCFG-Ebene berechnet.
Die ersten zwei der unteren drei Werte sollen eine Gefühl für die Masse an Daten, die während des Parsens generiert werden, geben und ganz am Ende ist die Dauer des Parsevorgangs angegeben.

5 ZUSAMMENFASSUNG

Abschließend soll die vorgestellte Arbeit in einigen Worten zusammengefasst werden.

Mit der Einführung im ersten Kapitel sollte ein grober Einstieg in das Thema gegeben und das Interesse des Lesers an den verschiedenen Teilen der Arbeit geweckt werden. Nachdem so klar gewordenen Umfang und einer Eingrenzung des Themengebiets wurde danach innerhalb dieser Umgebung spezielles Wissen vermittelt, auf dem spätere Teile dann weiter aufbauen können. Schon innerhalb der Grundlagen erhielt der Leser Schritt für Schritt eine genauere Vorstellung des Themas, was eine anschließende, tiefgründigere Betrachtung des HA*-Parsings erlaubte. Dabei wurde auf alle wesentlichen, zum Verständnis notwendigen Bereiche eingegangen und stark auf die Unterstützung mittels Beispielen gesetzt.
Im Laufe der ersten drei Kapiteln wurde ausreichend viel theoretisches Wissen geboten, um danach in die Praxis zu schwenken und dort zu im vierten Kapitel zu erfahren, wie die Anwendung aussieht und auch welche Schwierigkeiten bei der Implementierung unter anderem auftreten können. Es wurde deutlich an welchen Stellen des vorgestellten Programms welcher Teil der Theorie einfließt.
Insgesamt wurde in dieser Arbeit ein Einblick in das HA*-Parsing dargeboten und dieses Parsingverfahren für PCFGs instanziiert.

6 ANHANG

VERWENDETE PROGRAMME

Zur Erstellung dieser Arbeit und der Durchführung der Implementierung wurden folgende Programme verwendet:

- Adobe Reader XI
- Apache Open Office 3
- Eclipse Juno
- GIMP
- Google Chrome 29
- Inkscape
- Microsoft Editor
- MiKTeX 2.9
- Notepad++

LITERATURVERZEICHNIS

[FM07] Pedro F. Felzenszwalb und David McAllester: *The generalized A* architecture*,
In: J. Artif. Int. Res. 29.1 (Juni 2007), S. 153-190.
ISSN: 1076-9757.
URL: http://dl.acm.org/citation.cfm?id=1622606.1622612.

[PK09] Adam Pauls und Dan Klein: *Hierarchical search for parsing*,
In: Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics. NAACL '09. Boulder, Colorado: Association for Computational Linguistics, 2009, S. 557-565.
ISSN: 978-1-932432-41-1.
URL: http://dl.acm.org/citation.cfm?id=1620754.1620835.