Inkrementelle Analyse mit Constraint-Dependency-Grammatiken

Diplomarbeit

Niels Beuck

Mai 2009

Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Department Informatik
Zentrum für Intelligente Systeme und Robotik
Arbeitsbereich Natürlichsprachliche Systeme

Studienfach:	Informatik
Erstbetreuer:	Prof. Dr.-Ing. Wolfgang Menzel
Zweitbetreuer:	Dr.-Ing. habil. Reinhard Moratz, Associate Professor

Kurzfassung

Natürliche Sprache wird vom Menschen sowohl beim Sprechen als auch beim Schreiben inkrementell Wort für Wort erzeugt. Dadurch entsteht eine Verzögerung zwischen Beginn und Abschluss einer Äußerung. Für gewöhnlich gehen heutige Syntaxparser jedoch von vollständig vorliegenden Sätzen als Eingabe aus, so dass diese Zeitspanne nicht für die Verarbeitung genutzt werden kann.

Ziel dieser Diplomarbeit ist es, einen bestehenden Parser um die Möglichkeit der inkrementellen Verarbeitung zu erweitern. Inkrementalität bezeichnet dabei die Fähigkeit, bereits für unvollständige Eingaben eine Teilanalyse zu erzeugen und bei Vorliegen einer erweiterten Eingabe diese Teilanalyse zu erweitern, statt eine neue Analyse zu erzeugen. Die durch dieses Vorgehen angestrebten Ziele sind zum einen die frühe Bereitstellung von Zwischenergebnissen und zum anderen die Nutzung der Zeit, während der nur ein Teil der Eingabe vorliegt, um einen Verarbeitungsvorsprung aufzubauen.

Der nichtinkrementelle Parser, auf dem aufgebaut wird, basiert auf Constraint-Dependency-Grammatiken (CDG). Dependenz-Grammatiken sind ein Formalismus zur Syntaxbeschreibung natürlichsprachlicher Sätze, sie können durch gewichtete Constraints beschrieben werden. Der CDG-Parser arbeitet transformationsbasiert, d.h. er erzeugt aus einer initialen Analyse, die evtl. gegen Constraints der Grammatik verstößt, durch eine Folge von lokalen Änderungen eine abschließende Analyse, die gegen möglichst wenige Constraints verstößt. Um ein inkrementelles Vorgehen auf dieser Basis zu implementieren, wird der Parser nacheinander auf einzelnen Satzpräfixen aufgerufen, wobei die abschließende Analyse für ein Präfix als inititale Analyse für die Verarbeitung des nächsten Präfixes verwendet wird. Aufbauend auf diesem Ansatz werden verschiedene Möglichkeiten untersucht, die Kopplung zwischen den einzelnen inkrementellen Parsingschritten weiter zu erhöhen.

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Ziel dieser Arbeit ist es, einen bestehenden Constraint-Dependency-Grammar-Parser [cdg], kurz CDG-Parser, um die Möglichkeit der inkrementellen Analyse zu erweitern. Dieses inkrementelle Vorgehen unterscheidet sich dadurch vom bisherigen Verhalten des CDG-Parsers und dem vieler anderer Parser, dass es den Satz nicht von vornherein als Ganzes bearbeitet, sondern in Teilprobleme zerlegt. Diese Teilprobleme befassen sich mit einem Präfix des Satzes, wobei jeder dieser Schritte ein Wort mehr betrachtet als sein Vorgänger.

Es gibt verschiedene Gründe, warum ein solches Vorgehen interessant ist, der wichtigste ist wohl der, dass der Mensch Sätze eben auf diese Weise verarbeitet [van Gompel and Pickering, 2007]. Er verarbeitet den Satz von vorne nach hinten, bzw. von links nach rechts und das Bemerkenswerte ist, dass er dies im Wesentlichen in linearer Zeit tut. Die Analyse braucht nur so viel Zeit wie der Satz, um ausgesprochen zu werden. Damit ist die Zeit, die der Mensch nach Abschluss eines typischen Satzes noch zur Analyse benötigt, konstant! Ein Parser, der, auch wenn er in linearer Zeit arbeiten würde, erst dann mit der Verarbeitung anfängt, wenn der gesamte Satz vorliegt, schneidet also im Vergleich mit dem Menschen immer schlechter ab.

Nun kann der Rechner geschriebene Texte natürlich viel schneller erfassen, als ein Mensch sie lesen kann, so dass in diesem Fall praktisch keine zusätzliche Zeit durch inkrementelles Vorgehen erschlossen wird. Bei der direkten Mensch-Computer-Interaktion per natürlicher Sprache, ob gesprochen oder getippt, ist der Rechner jedoch an den menschlichen Rhythmus gebunden. Der menschliche Interaktionspartner wird einerseits seine Sprachproduktion nicht deutlich beschleunigen können (noch wollen), wird sich aber andererseits über Gesprächspausen ärgern.

Diese Zeitaspekte sind insbesondere relevant in dynamischen Kontexten, in denen das System schnell reagieren muss, z.B. ein Roboter, der Anweisungen umsetzt. Die Sprechzeit des Gegenübers kann nicht nur zum Verstehen des Geäußerten benutzt werden, sondern auch um die eigene Reaktion zu planen. Es sind auch Reaktionen denkbar, bevor der Gesprächspartner seine Ausführung von sich aus beendet, um ihn etwa mit einer Frage oder einen Einwurf zu unterbrechen oder auch nur um zu bestätigen das Teile des gesagten verstanden wurden. Eine solche Reaktion zur Äußerungszeit setzt eine Bearbeitung zur Äußerungszeit voraus.

Die Tatsache, dass die menschliche Sprache in ihrer Entstehung und in allen Modalitäten inkrementell von vorne nach hinten produziert und verarbeitet wurde, legt den Verdacht oder die Hoffnung nahe, dass sie sich besonders dazu eignet, auf diese Weise verarbeitet zu werden. Zu dieser Annahme gehört zum einen, dass sich, auch wenn erst

ein Teil des Satzes vorliegt, bereits sinnvolle Entscheidungen treffen lassen. Dies sind solche Entscheidungen, die, später getroffen, zu keinem besseren Ergebnis führen und auch nicht weniger Zeit in Anspruch nehmen würden. Zum anderen könnte sich mit Hilfe eines Teilergebnisses der Problemraum zukünftiger Entscheidungen beschneiden lassen, so dass diese mit weniger zeitlichem Aufwand getroffen werden könnten.

Auch aus psycholinguistischer Sicht ist inkrementelles Parsing interessant. Die oben erwähnte Beobachtung des linearen Zeitbedarfs bei der menschlichen Syntaxanalyse weist unter bestimmten Bedingungen Ausnahmen auf. Dies geschieht bei Sätzen, bei denen anfangs zwei Lesarten möglich sind, die naheliegende jedoch später durch semantische oder syntaktische Hinweise ausgeschlossen wird. Bei diesen sogenannten Gardenpathsätzen wird eine Revision vorhergehender Entscheidungen nötig oder der Satz als ganzes wird nicht verstanden und als ungrammatisch empfunden.

Eine solche Reanalyse äußert sich in erhöhter Verarbeitungszeit, wie sie sich in Experimenten mit Augenbewegungen zur Ermittlung der Lesezeit feststellen lassen [Staub and Rayner, 2007]. Interessant dabei ist, dass der erhöhte Verarbeitungsaufwand nicht dort im Satz anfällt, wo die Mehrdeutigkeit auftritt, sondern dort wo die präferierte Lesart ungültig wird. Ein computerlinguistischer Parser, der frühzeitig Entscheidungen trifft, die sich später als falsch herausstellen können, ist mit ähnlichen Problemen konfrontiert.

1.2. Aufbau der Arbeit

Zur Einführung der Grundlagen wird in Kapitel 2 wird der Formalismus der Dependenzgrammatik allgemein erläutert. In Kapitel 3 wird dann der Constraint-Dependecy-Grammar-Parser im speziellen behandelt. Dieser ist die Grundlage für die in späteren Kapitel beschriebenen Erweiterungen. In Kapitel 4 wird der Begriff der Inkrementalität allgemein untersucht, sowie verschiedene Definitionen und ihre Implikationen dargestellt. In Kapitel 5 wird vorgestellt, wie die Konzepte von Dependenzparsing und Inkrementalität zusammengeführt werden können. Dabei wird ein inkrementelles Vorgehen für den CDG-Parser entworfen. In Kapitel 6 werden verschiedene Möglichkeiten untersucht, die Kopplung zwischen zwei inkrementellen Schritten weiter zu erhöhen. Es werden insbesondere zwei Heuristiken vorgestellt mit denen anhand von vorangegangenen Zwischenergebnissen unmögliche Hypothesen ermittelt und aus dem Problemraum entfernt werden können, um Zeit einzusparen. Kapitel 7 bietet einen Ausblick auf verschiedene Fragestellungen, die an die hier vorgestellten Themen anknüpfen, jedoch über den Skopus dieser Arbeit hinausgehen.

2. Dependenzgrammatiken

Ein wichtiger Schritt in der Verarbeitung natürlicher Sprache ist das Parsing, also die Zuweisung einer syntaktischen Struktur. In der Computerlinguistik sind zwei verschiedene Formalismen zur Beschreibung solcher syntaktischen Strukturen verbreitet.

Dies ist zum einen die Phrasenstrukturgrammatik (Phrase Structure Grammar, PSG), bei der die Gruppierung der Wörter im Vordergrund steht und zum anderen die Dependenzgrammatik (Dependency Grammar, DG), die die Abhängigkeiten der Wörter untereinander in den Mittelpunkt stellt. Einen guten Überblick über Phrasenstrukturgrammatik und eine gute Einführung in die Sprachverarbeitung allgemein finden sich in Jurafsky [2008]. Einen guten Überblick über die Dependenzgrammatik gibt Nivre [2005].

Das zentrale Element in der PSG ist die Konstituente, ein Teil des Satzes der nur als Ganzes verschoben oder ersetzt werden kann. Der Satz wird durch eine verschachtelte Konstituentenstruktur beschrieben, wie in Abbildung 2.1 zu sehen. Bei der DG ist die Dependenz das zentrale Element, die eine Unterordnung eines Wortes (des Dependenten) unter ein anderes (seinen Regenten) darstellt. Die Wörter werden in einer hierarchischen Struktur angeordnet, wie in Abbildung 2.2 zu sehen.

Die Dependenzgrammatik hat verschiedene Vorzüge und Nachteile gegenüber der Phrasenstrukturgrammatik. Die Dependenzgrammatik eignet sich besonders für Sprachen mit freier Wortstellung, wie das Deutsche oder auch die slawischen Sprachen, bei denen die Funktion eines Wortes im Satz durch sein Flexion und weniger durch seine Position bestimmt ist. Ein Beispiel ist das Phänomen der Topikalisierung, also das Vorziehen eines Satzgliedes an den Satzanfang. Dies dient der Betonung des Satzgliedes, die Bedeutung wird nicht verändert. Ein durch Flexion als "Akkusativ" markiertes Nomen kann kein Subjekt sein, auch wenn es am Anfang des Satzes steht. Die Rolle des Satzgliedes ist also nicht durch die Position bestimmt, sondern über Flexion, die den Kasus bestimmt. Die PSG hingegen hat eine lange Tradition im angelsächsischen Sprachraum, da im Englischen die Wortstellung starrer ist (auch wenn

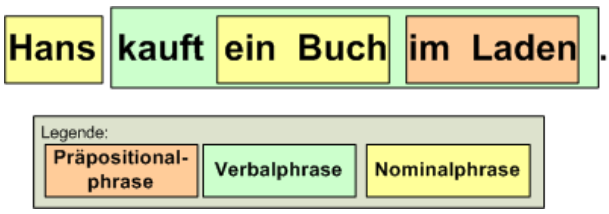

Abbildung 2.1.: Konstituentenstruktur eines Satzes

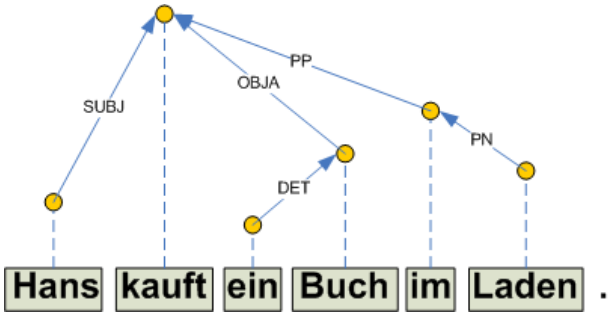

Abbildung 2.2.: Dependenzstruktur eines Satzes

die oben genannte Topikalisierung in Ausnahmefällen auch dort möglich ist).

Es können über Dependenzen nichtprojektive Strukturen erfasst werden, also solche, die aus einem nicht zusammenhängenden Teil des Satzes bestehen und von Wörtern unterbrochen werden, die nicht zur Struktur gehören. Eine Konstituente hingegen bezieht sich auf einen zusammenhängenden Teil des Satzes. Die PSG kann also zumindest in ihrer einfachen Form als kontextfreie Grammatik keine nichtprojektiven Strukturen abbilden. Ein Beispiel für nichtprojektive Strukturen sind Relativsätze, die nicht nur direkt hinter dem Wort, das sie modifizieren, sondern auch am Ende des Satzes stehen können.

Ein weiterer Vorteil der DG ist eine leichtere Syntax-Semantik-Integration. Syntaktische Anbindungen lassen sich häufig direkt in semantische Rollenzuweisungen abbilden. Das Subjekt eines Satzes entspricht z.B. meist dem Agenten des durch das Verb beschriebenen Ereignisses. Als Nachteil der DG sei hier die Skopusrepräsentation genannt. So ist es nicht klar, ob ein Supplement das Wort, dem es untergeordnet ist oder die Phrase, deren Kopf dieses ist, modifiziert. Das in dieser Arbeit vorgestellte und erweiterte Parsingsystem basiert auf der Dependenzgrammatik.

2.1. Linguistische Bedeutung der Dependenz

Nivre [2005] zählt verschiedene Kriterien dafür auf, wann man zwischen zwei Wörtern von einer Abhängigkeitsrelation, also Dependenz sprechen kann. Gegeben sei ein Wort D (Dependent), ein Wort H (Regent) und eine Konstruktion C die diese beiden Wörter enthält.

1. H bestimmt die syntaktische Kategorie von C und kann es evtl. sogar ersetzen.
2. H bestimmt die semantische Kategorie von C, D steuert semantische Spezifika bei
3. H ist obligatorisch, D kann optional sein
4. H legt fest, ob D erlaubt ist und ob es optional ist

5. Die Form von D hängt von H ab (Kongruenz)

6. die Position von D im Satz hängt von H ab

Durch eine Dependenz können also verschiedene linguistische Phänomene beschrieben werden. Das ist zum einen das Phänomen der Kongruenz, also dass ein Wort die Form eines anderen vorgeben kann. Kasus, Numerus und Genus eines Nomens bestimmen z.B. eben diese Attribute bei seinem Artikel. Schreibt ein Wort die Anwesenheit anderer Wörter vor, spricht man von Valenz. Das untergeordnete Wort ist ein Komplement, das die Valenz sättigt. Ein Beispiel hierfür sind Verben, die bestimmte Objekte verlangen. Das Verb "kaufen" verlangt z.B. ein Akkusativobjekt. Andere Wörter sind optional und stellen auf der semantischen Ebene eine Anreicherung durch eine zusätzliche Information dar. Man spricht von einem Supplement. Typische Beispiele hierfür sind Adjektive ("das rote Auto"), Adverbien ("fährt schnell") oder Präpositionen ("auf der Straße").

Es gibt sowohl syntaktische als auch semantische Gründe, ein Wort unter ein anderes unterzuordnen, die sich zum Teil widersprechen können. Im Beispiel "Hans kauft ein Buch im Laden." ist "Laden" der Ort des Kaufes, semantisch also diesem untergeordnet, syntaktisch ist jedoch noch die Präposition "im" dazwischen. Die Alternativen sind also die Abhängigkeitsstrukturen $kaufen \leftarrow im \leftarrow laden$ und $kaufen \leftarrow laden \leftarrow im$.

Eine andere Unklarheit ist die Behandlung von Konjunktionen ("Haus und Hof"). Hier ist es sowohl denkbar, beide Konjunkte der Konjunktion "und" unterzuordnen, als auch das eine Konjunkt indirekt über die Konjunktion dem anderen unterzuordnen. In diesen Fällen ist keine der Möglichkeiten pauschal der anderen vorzuziehen.

2.2. Der Dependenzgraph

Die aus den einzelnen Unterordnungen bestehende **Dependenzstruktur** ist ein **gerichteter** Graph, wobei die Wörter des Satzes die **Knoten** und die Dependenzen die **Kanten** bilden. Die Art der Anbindung ist durch die beteiligten Wörter nicht immer eindeutig gegeben, einem Verb können z.B. mehrere Nomen untergeordnet sein. Welches von diesen das Subjekt und welches ein Objekt ist, wird durch den Typ der Dependenz näher spezifiziert. Die Kanten im Graph sind also beschriftete Kanten, der Graph ein **beschrifteter** gerichteter Graph.

Bei den weiteren Einschränkungen gibt es verschiedene Varianten, die meisten Definitionen verlangen jedoch Zyklenfreiheit und eindeutige Unterordnung, also dass kein Knoten sich transitiv oder reflexiv selbst untergeordnet ist und dass jeder Knoten nicht mehr als einen Regenten hat. Der dadurch beschriebene Graph hat eine hierarchische Baumstruktur mit mindestens einem Wurzelknoten, der keinem anderen untergeordnet ist. Diese Rolle fällt typischerweise dem finiten Verb zu. Eine weitere mögliche Einschränkung ist die Projektivität, d.h. die Forderung, dass sich Kanten nicht überschneiden dürfen. Ein Vorteil der Dependenzgrammatik ist jedoch gerade, dass auch nichtprojektive Strukturen mit ihr beschrieben werden können. Diese Strukturen können in vielen Sprachen auftreten, wenn sie auch nicht häufig sind. Projek-

tivität wird hier daher nicht pauschal eingefordert, sondern durch Constraints in der Grammatik beschränkt.

Eine **Dependenzstruktur** zu einem Satz $S = (w_1, .., w_n)$ ist eine Menge von Kanten $K \subset \{w_1, .., w_n\} \times L \times \{w_1, .., w_n\}$, wobei L die Menge der möglichen Kantenbeschriftungen ist. Die Dependenzrelation d ist definiert als $(a, b) \in d \leftrightarrow (a, l, b) \in K$ für ein beliebiges $l \in L$ und d^* ist die reflexive, transitive Hülle von d.

Eine Dependenzstruktur gilt als wohlgeformt, wenn sie folgende Eigenschaften besitzt.

eindeutige Unterordnung $(a, b_1, c_1) \in K \wedge (a, b_2, c_2) \in K \Rightarrow b_1 = b_2 \wedge c_1 = c_2$

zyklenfrei $\neg d^*(w, w)$ für alle $w \in S$

Eine solche Dependenzstruktur weist also jedem Wort genau eine Kante und damit genau einen Regenten zu. Eine Ausnahme ist das Wort an der Wurzel des Baumes, dem bei der obigen Definition keine Kante zugewiesen wird. Um aus der partiellen eine vollständige Abbildung zu machen, erweitern wir die Menge der als Regenten zur Verfügung stehenden Knoten um 'NIL'. Dem Wurzelwort kann damit explizit der Regent 'kein Regent' zugeordnet werden.

2.2.1. Ebenen

Es gibt verschiedene Motivationen, ein Wort einem anderen unterzuordnen. Die syntaktisch motivierte Variante kann eine andere sein als die semantisch motivierte. Ein Weg, diese Mehrdeutigkeit abzubilden, ist die Einführung verschiedener Ebenen, wobei ein Wort auf jeder Ebene einen anderen Regenten haben kann. Für jede Ebene existiert dann eine Dependenzstruktur. Die resultierenden Graphen teilen sich die Knoten, also die Wörter des Satzes, haben jedoch jeweils eigene Kanten. Auf den verschiedenen Ebenen können verschiedene Anforderungen an den Graphen gestellt werden, z.B. hinsichtlich Projektivität oder Zusammenhang und für jede Ebene steht eine andere Menge von Kantenbeschriftungen zur Verfügung.

2.3. Lesarten der Wörter

Bisher haben wir die Knoten des Dependenzgraphen durch die Wörter des Satzes als gegeben betrachtet, ein Wort kann jedoch verschiedene Lesarten mit unterschiedlichen syntaktischen Eigenschaften besitzen. Diese Lesarten können sich in der Wortart unterscheiden, aber auch innerhalb einer Wortart gibt es Ambiguitäten bei grammatikalischen Eigenschaften wie Kasus, Genus, Numerus etc. Ob eine Dependenz grammatisch ist, hängt aber auch von der Lesart von Dependent und Regent ab. Mehrere eingehende und ausgehende Dependenzen eines Wortes könnten verschiedene Lesarten voraussetzen. Um dies zu verhindern, muss neben der Dependenzstruktur für jedes Wort eines Satzes eine Lesart bzw. ein Lexem ausgewählt werden.

Lexem Ein Lexem ist ein Lexikoneintrag für ein Wort und ist charakterisiert über Attribute, insbesondere Wortart und Wortstamm, M_{Lx} sei die Menge aller möglichen Lexeme

Lexikon $L_{exikon} : W \rightarrow \mathcal{P}(M_{Lx})$ wobei W die Menge aller Wörter ist. Ein Lexikon bildet also ein Wort auf eine Menge von Lexemen ab.

Lesartzuweisung Gegeben ein Lexikon L_{exikon} ist für einen Satz $s = (w_1, .., w_n)$ mit $w_i \in W$ die die Sequenz $z = (l_{x1}, .., l_{xn})$ mit $l_{xi} \in M_{Lx}$ eine Lesartzuweisung, wenn gilt: $l_{xi} \in L_{exikon}(w_i)$. Z_{La} sei die Menge aller möglichen Lesartenzuweisungen und $Z_{La,n}$ die Menge aller Lesartenzuweisungen der Länge n

Als Zusammenfassung von Dependenzstrukturen und Lesartzuweisung zu einem Satz können wir den Begriff der Analyse einführen. Eine **Analyse** a eines Satzes s ist das Tupel $a = (d_1, .., d_n, z)$ wobei n die Anzahl der Ebenen ist. Die Dependenzstrukturen teilen sich die Elemente von z als Knoten.

2.4. Grammatikbeschreibung

Bisher wurde betrachtet, wie Dependenzstrukturen und Analysen allgemein aussehen, aber nicht, wie festgelegt wird, welche Strukturen erlaubt sind. Eine Grammatik sei eine solche Instanz, in der Beschrieben ist, welche Kombinationen von Analyse und Satz erlaubt sind. Eine Grammatik G beschreibt also eine Sprache $L_G \subset S \times A$, wobei S die Menge aller möglichen Sätze und A die Menge aller möglichen Analysen ist. In der Grammatik sind festgelegt

1. die Anzahl n der Ebenen
2. ein Lexikon, dass die möglichen Lexeme für jedes Wort enthält.
3. für jede Ebene eine Teilgrammatik g_i, die die möglichen Dependenzstrukturen auf der Ebene i beschreibt, $g_i \subset (S \times D)$, wobei auch Abbildungen zwischen den Ebenen möglich sind.

Eine Grammatik ist also durch ein Tupel $G = (n, L_{exikon}, g_1, .., g_n)$ gegeben. Eine Dependenzstruktur $d \in D$ ist eine zu g_i konforme Struktur von $s \in S$, wenn $(s, d) \in L_G$ ist. Ein Satz $s \in S$ ist grammatikalisch korrekt bezüglich G, wenn es ein Analyse $a \in A$ gibt mit $a = (d_1, .., d_n, z)$, so dass $(s, d_i) \in g_i \forall i$ und z eine vom L_{exikon} erlaubte Lesartzuweisung zu s ist. Dann ist $(s, a) \in L_G$.

2.4.1. Constraintgrammatik

Eine Möglichkeit, eine solche Grammatik zu beschreiben, und diejenige, die wir hier betrachten wollen, ist eine Beschreibung durch Constraints. Dieser Ansatz der Constraint-Dependency-Grammar (CDG) geht auf Maruyama [1990] zurück. Die Grammatik besteht dabei aus einer Menge von Constraints, mit denen Anforderungen wie die, dass nur Verben Subjekte haben und dass jedes Verb genau ein Subjekt haben muss, modelliert werden können.

Ein Constraint C ist beschrieben durch eine prädikatenlogischen Formel, die ein n-Stelliges zusammengesetztes Prädikat $p_c(k_1, .., k_n)$ auf n Kanten definiert, wobei die in der Formel vorkommenden elementaren Prädikate sich beziehen können auf

- Die Beschriftung einer Kante
- Die Position von Dependent und Regent im Satz
- Die Wortart von Regent oder Dependent
- Weitere Attribute von Regent und Dependent, wie Kasus oder Numerus.
- Tokens des Satzes und die eine Kante umgebende Dependenzstruktur

Constraints haben eine Stelligkeit $n \geq 1$, die der Stelligkeit ihres zusammengesetzten Prädikats entspricht. Ein Constraint C gilt als erfüllt für n Kanten k_1 bis k_n, wenn $p_c(k_1, .., k_n)$ zu "wahr" ausgewertet wird. Ist ein Constraint nicht erfüllt, gilt es als verletzt. Eine Dependenzstruktur d ist konform zu einer Grammatik G, wenn für alle Kanten von d und jede in G vorkommende Constraintstelligkeit n, jede mögliche Kombination von n Kanten jedes n-stellige Prädikat erfüllt.

Die auf einer Ebene zulässigen Kantenbeschriftungen ließen sich über ein Constraint definieren, das für jede Kante eine Beschriftung aus einer festgelegten Menge vorschreibt. Aus praktischen Gesichtspunkten wird diese Menge jedoch außerhalb der Constraints definiert, damit die Menge der möglichen Kanten vor Auswertung der Constraints begrenzt ist.

2.4.2. Gewichtete Constraints

Grammatik-Definitionen mit Hilfe der oben beschriebenen Constraints haben den Nachteil, dass nicht garantiert werden kann, dass genau eine Dependenzstruktur zu einen gegebenen Satz existiert. Abweichungen davon sind in beide Richtungen problematisch. Eine Grammatik ohne Constraints erlaubt alle möglichen Dependenzstrukturen für einen Satz, jedes weitere Constraint schränkt diese Menge weiter ein. Werden zu wenige Constraints formuliert, ist eine Vielzahl an Dependenzstrukturen zu einem Satz erlaubt, bei zu vielen Constraints keine.

Während es möglich ist, für einen gegebenen einzelnen Satz oder einen sehr kleinen Ausschnitt einer Sprache eine Menge von Constraints zu formulieren, die nur genau die gesuchte Struktur erlauben, hat sich dieses Unterfangen für alle Sprachausschnitte relevanter Größe oder gar für die Erfassung einer Sprache insgesamt als nicht durchführbar erwiesen. Ein Grund dafür ist, dass in natürlichen Sprachen verschiedene Strukturierungsprinzipien gelten, die miteinander konkurrieren, nicht immer erfüllt sein müssen und in komplexeren Satzkonstruktionen auch gar nicht alle erfüllt sein können.

Solche Kriterien lassen sich jedoch nicht durch ein Constraint wie oben beschrieben erfassen. Werden sie als Constraint aufgenommen, werden alle Sätze, die dieses Kriterium verletzen, ausgeschlossen, auch wenn eine Sprecher der zu modellierenden Sprache ihn als grammatikalisch korrekt einstufen würde. Nimmt man sie nicht als Constraint auf, hat die Grammatik keine Möglichkeit eine Dependenzstruktur, die das Kriterium nicht verletzt, gegenüber einer, die es verletzt, zu bevorzugen.

Die Constraints im obigen Sinne sind also harte Constraints, sie sind erfüllt oder nicht, und ist eines nicht erfüllt, erlaubt die Grammatik die Struktur nicht. Der Ausweg ist eine Erweiterung der Constraint-Definition zu gewichteten Constraints [Menzel and

Schröder, 1998]. Auf diese Weise können auch weiche Constraints erfasst werden, die eine sie verletzende Struktur nicht ausschließen, jedoch schlechter bewerten. Neben dem Prädikat p besteht ein gewichtetes Constraint c also aus einer Bewertung b, $c = (p, b)$. Die Bewertung ist eine Zahl aus dem Intervall $[0, 1]$, sie kann eine feste Zahl sein oder sich aus den Kanten, auf die das Prädikat angewendet wird, berechnen. Um nun eine Bewertung für eine Dependenzstruktur d zu erhalten, werden die Straffaktoren für alle Constraintverletzungen aufmultipliziert, die von Kanten in d verursacht werden.

Ein Constraint mit der Bewertung 0 entspricht dann dem alten, harten Constraint. Wird es in einer Analyse verletzt, ist die Bewertung der Analyse auf 0 festgelegt, die Dependenzstruktur ist dann für den Satz nicht erlaubt. Ein Constraint mit der Bewertung 1 hingegen hat keinerlei Auswirkung auf die Bewertung einer Struktur.

Über die Bewertung lassen sich Dependenzstrukturen nun auch vergleichen, so dass unter verschiedenen möglichen Strukturen, die eine oder die wenigen am besten bewerteten ausgewählt werden können. Die Grammatik kann dadurch zwischen mehreren möglichen Strukturen disambiguieren. Um wirklich eine einzelne Struktur als die beste auszuwählen, muss die Grammatik jedoch sicherstellen, dass keine zwei Strukturen gleich bewertet werden. Dies lässt sich durch eine Erweiterung der Grammatik um nur schwach bestrafte (Bewertung nahe 1) Constraints annähern, um bei zwei möglichen Strukturen eine zu präferieren.

Mit gewichteten Constraints kann nun auch leicht ungrammatischen Sätzen eine Struktur zugewiesen werden, die dann zwar mehr oder weniger schlecht, jedoch > 0 bewertet wird. Solche ungrammatischen Sätze oder auch unvollständige Ellipsen treten im alltäglichen Sprachgebrauch auf, die Fähigkeit, sie zu analysieren, ist also durchaus für praktische Zwecke relevant.

2.5. Dependenz-Parsing

Im vorangegangenen Abschnitt haben wir betrachtet, wie eine Dependenzstruktur definiert ist und wie über eine gewichteten Constraint-Dependency-Grammatik die beste Analyse für einen Satz definiert werden kann. Die Grammatik legt jedoch nicht den Weg fest, wie diese Analyse berechnet werden kann. Ein Algorithmus, der aus allen möglichen Analysen eine gültige Analyse bzw. die beste Analyse ermittelt, ist ein Parsingalgorithmus.

Der einfachste Algorithmus ist die komplette Suche durch den Suchraum, wobei jede Analyse im Suchraum durch die Grammatik bewertet wird und zum Schluss die beste als Ergebnis ausgegeben wird. Der Suchraum hat dabei folgende Größe. Sei b die Anzahl der in der Grammatik möglichen Beschriftungen und n die Anzahl der Wörter im Satz, dann ist die maximale Anzahl der möglichen Kanten für jedes Wort als Dependent $k = (b \cdot n)$, die der möglichen Dependenzstrukturen über den ganzen Satz $d = (b \cdot n)^n$. Der Suchraum wächst also, schon ohne Betrachtung der verschiedenen Lesarten, exponentiell mit der Satzlänge. Den Suchraum erschöpfend zu durchsuchen kommt also nicht in Frage.

Es gibt verschiedene Ansätze für Parsing-Algorithmen, die bereits erfolgreich in Dependenzparsern verwendet werden. Im CDG-System ist u.a. ein transformationsba-

siertes Dependenz-Parsing [Foth, 2007] implementiert, mit dem bei bisherigen Evaluationen die besten Ergebnisse erzielt wurden.

2.5.1. Transformationsbasiertes Dependenz-Parsing

Diesem Ansatz liegt die Idee zugrunde, dass ausgehend von einer beliebigen Analyse durch kleine Veränderungen, also das Ersetzen einer oder weniger Kanten, sukzessiv bessere Analysen erzeugt werden, bis keine Verbesserung mehr möglich ist. Formal gesehen besteht ein transformations-basierter Parser also aus zwei Komponenten, einer Initialisierungsfunktion, die zu einem Satz s eine initiale Analyse a_0 generiert, und einer Transformationsfunktion, die eine Analyse a_i in einer andere a_{i+1} transformiert.

Über die Bewertung der initialen Analyse wird keine Aussage gemacht, es ist auch nicht garantiert, dass a_0 überhaupt eine unter der gegebenen Grammatik g gültige Analyse für s ist, also dass sie keine harten Constraints in g verletzt. Die Transformationsfunktion T ist eine Abbildung $T : A \rightarrow A \cup' halt'$, wobei A die Menge aller möglichen Analysen ist. Für $a_1, a_2 \in A$ mit $T(a_1) = a_2$ gilt, dass $B(a_2) > B(a_1)$ für eine Bewertungsfunktion $B : A \rightarrow [0, 1]$. Das Symbol $'halt'$ wird ausgegeben, wenn keine Verbesserung mehr möglich ist, in diesem Fall terminiert der Algorithmus. Dieser Algorithmus arbeitet nicht inkrementell. Bereits im ersten Schritt, der Erzeugung der initialen Analyse, werden alle Wörter des Satzes verwendet.

Der Algorithmus kann nach jeder Iteration die aktuelle Analyse als Zwischenergebnis vorweisen. Wird er vorzeitig abgebrochen, kann dieses als nicht optimales Endergebnis ausgegeben werden. Diese Eigenschaft ist die sog. Anytime-Fähigkeit. Vom Zeitverhalten her ist das transformations-basierte Parsing praktisch deutlich schneller als die komplette Suche, im Worst-Case Verhalten jedoch weiterhin exponentiell.

2.5.2. Andere Dependenz-Parser

Shift Reduce Pasing

Als eine Alternative zum transformationsbasierten Constraint-Dependency-Parsing sei hier das im Malt Parser verwendete Shift-Reduce-Parsing [Nivre, 2008] erwähnt. Das Vorgehen dieses Parsers ist in dem Sinne inkrementell, dass die Kanten nacheinander aufgebaut werden. Eine vollständige Analyse liegt erst am Ende des Prozesses vor. Die Kanten der Analyse werden nacheinander erzeugt, indem für jede Kombination von Wörtern [1] des Satzes eine sog. Orakelfunktion entscheidet, ob das eine dem anderen untergeordnet werden soll oder nicht.

Der Zustand eines Shift-Reduce-Algorithmus für nichtprojektives Dependenz-Parsing nach Nivre [2008] sei beschrieben durch ein Tupel $c = (\lambda_1, \lambda_2, \beta, A)$, wobei λ_1 und λ_2 Listen von Tokens sind, β ein Puffer für die Tokens des Eingabesatzes und A eine Menge von Dependenzen zwischen Tokens ist. Die Startkonfiguration ist $c_0 = ((0), (), w_1, .., w_n, \{\})$, die erste Liste enthält den NIL-Regenten 0, die andere Liste

[1] Dies gilt für die nicht projektive Version des Algorithmus, beschränkt man sich auf projektive Strukturen müssen nicht alle Kombinationen geprüft werden

und die Kantenmenge sind leer und der Puffer enthält alle Wörter des Satzes. Eine Endkonfiguration ist erreicht, wenn der Puffer β leer ist.

Der Algorithmus kennt vier verschiedene Transitionen zwischen Konfigurationen. Im Folgenden wird der Kopf der Liste λ_1 rechts, der von λ_2 und β links notiert. Ein Dependenz wird als Tupel (a, l, b) notiert, wobei der Regent a ein Token des Satzes oder 0 und der Dependent b ein Token des Satzes ist. l ist eine Kantenbeschriftung. a ist der Dependent und b der Regent. Die Konkatenation zweier Listen wird durch den Operator . dargestellt.

left Arc $(\lambda_1|i, \lambda_2, j|\beta, A) \Rightarrow (\lambda_1, i|\lambda_2, j|\beta, A \cup (j, l, i))$

right Arc $(\lambda_1|i, \lambda_2, j|\beta, A) \Rightarrow (\lambda_1, i|\lambda_2, j|\beta, A \cup (i, l, j))$

no Arc $(\lambda_1|i, \lambda_2, \beta, A) \Rightarrow (\lambda_1, i|\lambda_2, \beta, A)$

shift $(\lambda_1, \lambda_2, i|\beta, A) \Rightarrow (\lambda_1.\lambda_2|i, [], \beta, A)$

Eine Kante kann aufgrund der Anforderung der eindeutigen Unterordnung nicht hinzugefügt werden, wenn A bereits eine Kante mit dem selben Dependenten enthält. Außerdem kann 0 nie Dependent einer Kante sein. Der Kopf des Puffers ist das aktuelle Wort, die Listen enthalten alle vorangegangenen Wörter. Eine Anbindung wird für das erste Token des Puffers β und den Kopf der ersten Liste geprüft. Über die *Arc*-Transitionen wird zum nächsten Wort der ersten Liste übergegangen, wobei eine linksgerichtete, rechtsgerichtete oder keine Kante erstellt wird. Mit "*shift*" wird zum nächsten Wort des Puffers übergegangen.

Eine auf einem statistischen Modell basierte Orakelfunktion bestimmt für jede Konfiguration die durchzuführende Transition. Der Zeitbedarf wächst quadratisch mit der Länge des Satzes, für eine einfachere Variante, die sich auf projektive Strukturen beschränkt, wächst er sogar nur linear, vorausgesetzt eine in konstanter Zeit arbeitende Orakelfunktion. Der Shift-Reduce-Algorithmus selbst arbeitet inkrementell, die Orakelfunktion kann, abhängig von dem ihr zugrunde liegenden Featuremodell jedoch auf weiter rechts liegende Wörter, sprich auf Elemente des Puffers β nach dem ersten zurückgreifen. Um state-of-the-art Genauigkeit zu erreichen ist laut Nivre [2006] ein Vorgriff um drei Wörter von Nöten. Über eine Anbindung kann also erst dann entschieden werden, wenn die drei Wörter nach ihrem Regenten respektive Dependenten verfügbar sind.

Genauigkeits-Vergleich

In Tabelle 2.1 ist die Genauigkeit des CDG-Parsers mit der des Maltparsers und der eines weiteren State-Of-The-Art Dependenz-Parsers, dem MST-Parser [McDonald et al., 2005], verglichen. Alle Angaben gelten für unrestringierte deutschsprachige Sätze. Die angegebene Prozentzahl bezieht sich auf die sog. Labeled Accuracy, also den Anteil der Wörter, für die sowohl Anbindungspunkt als auch Typ verglichen mit einem Goldstandard korrekt bestimmt wurden.

Im Gegensatz zum CDG-Parser arbeiten der Malt-Parser und der MST-Parser nicht mit einer manuell erstellten Grammatik sondern mit einer Support-Vector-Maschine

Parser	Accuracy
CDG [Foth, 2007] (mit POS-Tagger und PP-Attacher)	88.9
Malt [Nivre et al., 2006]	85.82
McDonald [McDonald et al., 2006]	87.34

Tabelle 2.1.: Labeled Accuracy für Deutsch

(SVM), die auf einem Trainingskorpus der entsprechenden Sprache trainiert ist. Es sei erwähnt, dass die Featuremodelle der SVM sowohl des Malt-Parsers als auch des MST-Parsers nicht für das Deutsche optimiert wurden, sondern sprachunabhängig sind. Die Werte wurden auf unterschiedlichen Datensätzen bestimmt.

3. Der CDG-Parser

In diesem Kapitel findet sich eine Übersicht über den CDG-Parser [Foth, 2007], auf dem in dieser Arbeit aufgebaut wird. Während die in diesem Parser verwendete Weighted-Constraint-Dependency-Grammatik und der transformationsbasierte Parsingalgorithmus bereits allgemein erklärt wurden, will ich hier auf einige Details des Systems eingehen, wobei ich mich in der Darstellung auf die für die Belange dieser Arbeit relevanten Aspekte beschränken werde.

3.1. Repräsentation des Suchraumes

Wie weiter oben definiert, besteht eine Analyse aus einer Lesartzuweisung und einer Dependenzstruktur für jede Ebene, wobei einem Wort ein Lexem und eine Kante auf jeder Ebene zugeordnet wird. Der Suchraum ist also beschrieben durch die Menge der Lexeme für die Wörter des Satzes und die Menge der Kanten auf jeder Ebene für jedes Wort des Satzes. Die Lexeme sind in einem sog. Lexemgraphen zusammengefasst, die Kanten in sog. Domänen. Eine Domäne umfasst dabei alle Kanten einer Ebene, die von einem Wort ausgehen können, die also dieses Wort als Dependent haben können. In einer Analyse ist genau eine Kante jeder Domäne ausgewählt.

3.1.1. Der Lexemgraph

Die möglichen Lexeme sind in einem sog. Lexemgraphen zusammengefasst. Für den Satz wird eine Menge von Zeitpunkten definiert, das Intervall zwischen zwei unmittelbar aufeinander folgenden Zeitpunkten wird Zeitslot genannt. Ein Lexemknoten hat einen Start- und einen Endzeitpunkt, ist also einem oder mehreren aufeinander folgenden Zeitslots zugeordnet. Ein gültiger Pfad durch den Lexemgraphen ist eine Menge von Lexemknoten, in der für jeden Zeitslot genau ein ihm zugewiesener Lexemknoten vorkommt. Terminologisch ist zu beachten, dass Lexemknoten die Knoten im Graphen der Dependenzstruktur sind. Im Lexemgraphen entsprechen sie den Kanten!

Es muss unterschieden werden zwischen synchronen und asynchronen Lexemgraphen. Im ersten Fall überschneiden sich jeweils zwei Lexemknoten in allen oder in keinem der von ihnen abgedeckten Zeitslots. Das sind also vor allem Lexemgraphen, bei denen alle Lexemknoten jeweils nur einem Zeitslot zugeordnet sind. Wenn zu einem Satz mit eindeutig bekannten Wörtern der Lexemgraph aus den im Lexikon nachgeschlagenen Lexemen zu den Wörtern aufgebaut wird, entsteht ein solcher synchroner Lexemgraph. Die Wörter sind total geordnet und überlappen sich nicht.

Bei asynchronen Lexemgraphen ist die Anzahl der Lexemknoten in einem gültigen Pfad nicht festgelegt, da einige Lexemknoten mehrere Zeitslots überspannen können.

Dadurch ist auch die Anzahl der Kanten in einer Dependenzstruktur nicht festgelegt, da für diese ja verlangt wird, dass von jedem Lexemknoten genau eine Kante ausgeht. Der allgemeinere Fall der asynchronen Lexemgraphen ist insbesondere relevant, um mit Ungenauigkeiten bei der Erfassung gesprochener Sprache umzugehen, bei der die einzelnen Wörter nur mit einer gewissen Unsicherheit hypothetisiert werden können. Alternative Hypothesen in der Spracherkennung könnte dann in der syntaktischen Analyse durch CDG disambiguiert werden. Diese Fähigkeit ist in der Architektur von CDG angelegt, wird bei der Verarbeitung geschriebener Sprache jedoch aus oben genannten Gründen nicht gebraucht.

Bei der hier betrachtetet inkrementellen Verarbeitung sei ein Inkrement die Erweiterung um ein ganzes Wort (oder Satzzeichen). Die Erweiterung eines Lexemgraphen sei immer die Erweiterung um einen Zeitslot. Veränderungen an vorangegangen Zeitslots werden nicht vorgenommen, wir beschränken uns auf synchrone Lexemgraphen.

Größe des Lexemgraphen

Beim Aufbau des Lexemgraphen werden die verschiedenen Lesarten für jedes Wort aus dem Lexikon ermittelt. Die maximale Anzahl an Lexemen im Lexemgraphen resultiert also aus der Länge des Satzes (Anzahl der Wörter) sowie der Ambiguität des Lexikons (maximale Anzahl an Lesarten für ein Wort). $O(n) = n$ bei einem Satz der Länge n und bei gegebener Grammatik.

3.1.2. Kanten und Domänen

Kanten sind repräsentiert durch folgende Attribute:

1. Ebene der Grammatik
2. Menge von Lexemknoten als Dependent
3. Menge von Lexemknoten als Regenten
4. Kantenbeschriftungen

Kanten beziehen sich nicht auf Wörter sondern auf Lexemknoten, da eine Kante abhängig von dem für ein Wort ausgewählten Lexemknoten unterschiedlich bewertet werden kann. Gegeben sei eine Kante auf der Syntaxebene, ausgehend vom Wort "der" zum Wort "Hund" mit der Kantenbeschriftung "Determiner". Diese Unterordnung ist grammatikalisch korrekt für die Artikel-Lesart von "der", für die Pronomen-Lesart muss die Kante schlechter bewertet werden. Um die Anzahl der Kanten im Suchraum klein zu halten, werden Kanten mit Lexemknoten, die sich nicht in für die Kante relevanten Attributen unterscheiden, zusammengefasst. Welche Attribute für eine Kante relevant sind, geht aus den Constraints der verwendeten Grammatik hervor. Näher ausgeführt ist dieses Vorgehen in Foth [1999]. Aufgrund dieser Zusammenfassung von Lexemknoten sind Dependent und Regent einer Kante Mengen von Lexemknoten.

Die Menge der möglichen Kantenbeschriftungen ist für jede Ebene einzeln in der Grammatik definiert.

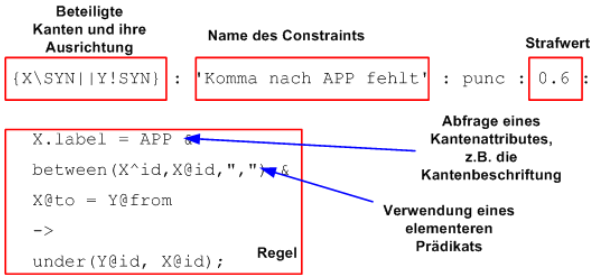

Abbildung 3.1.: Beispiel für ein Constraint in der Grammatik

3.1.3. Analyse

Eine Analyse besteht aus einem Pfad im Lexemgraphen und aus einer ausgewählten Kanten in jeder Domäne. Es ist zu beachten, dass an die Dependenzstruktur einer Analyse keine Anforderungen hinsichtlich Zusammenhang, Projektivität oder Zyklenfreiheit gestellt werden. Entsprechende Einschränkungen müssen bei Bedarf in der Grammatik als Constraints formuliert werden.

3.2. Die Grammatik

Die Grammatik beinhaltet eine Menge von Constraints. Abbildung 3.1 zeigt Beispielhaft ein solches Constraint. Ein Constraint besteht aus folgenden Komponenten:

- Die Ebene, auf der das Constraint gilt
- Anzahl und Ausrichtung der beteiligten Kanten
- Name und Kategorie
- Strafwert (eine fixe Zahl zwischen 0 und 1 oder ein Term, um eine solche zu berechnen)
- Logische Formel über die Eigenschaften der Kanten, meist in Form einer Implikation, Regelkopf → Regelkörper

In der logischen Formel können verschiedene Prädikate verwendet werden. Diese Prädikate können sich auf Attribute und Position von Regent und Dependent einer Kante beziehen. Andere beziehen sich auf die Ausrichtung der Kanten, wie links- oder rechtsgerichtet oder ob Kanten in der Dependenzstruktur sequenziell oder verzweigend angeordnet sind. Es sind auch Prädikate möglich, die die Anwesenheit bestimmter Wörter oder Satzzeichen in einem Abschnitt des Satzes oder auch einer bestimmten Kante an anderer Stelle in der Dependenzstruktur abfragen. Diese letztgenannten Prädikate nennen sich kontextsensitive Prädikate.

Anhand der Eigenschaften ihrer Bestandteile lassen sich die Constraints wie folgt kategorisieren.

Unär oder binär Es werden Constraints, die sich auf eine Kante beziehen unterschieden von solchen, die sich auf zwei beziehen. Höhere Stelligkeiten sind nicht vorgesehen

Hart oder weich Constraints mit einer Bewertung von 0 gelten als hart, alle anderen als weich.

Kontextsensitiv oder nicht Zwar kann sich ein Constraint nur auf höchstens zwei Kanten direkt beziehen, über Prädikate lässt sich aber die Existenz von bestimmten Phänomenen an anderer Stelle in der Dependenzstruktur abfragen.

Fixer oder berechneter Strafwert Manche Constraints haben eine flexible Bewertung, z.B. geht die Distanz von Regent und Dependent im Satz bei Abstandsbestrafungen ein.

Aufruf externer Prädiktoren? wie dem POS-Tagger oder dem PP-Attacher

Ein wichtiges Merkmal ist dabei, dass nur unäre, nicht kontextsensitive Constraints an einzelnen Kanten ohne eine umliegende Dependenzstruktur ausgewertet werden können.

3.2.1. Ebenen der Grammatik

In dieser Arbeit wird eine universale Grammatik für die deutsche Sprache verwendet [Foth, 2007]. Die Grammatik enthält zwei Ebenen, die Syntaxebene und die Referenzebene. In letzterer wird die pronominale Referenz eines Relativpronomens auf seinen Antezedenten angezeigt. Wörter, die keine Relativpronomen sind, haben hier NIL als Regenten.

Aus der Dokumentation der Grammatik:

```
Die Ebene SYN enthält den gesamten Syntaxbaum. Nicht-baumartige
Beziehungen, insbesondere Sekundärrelationen wie ‘logisches
Subjekt’ oder ‘zweites Objekt in der Konjunktion’ werden nicht
modelliert.
Labels der Syntaxebene:
 ADV, APP, ATTR, AUX, AVZ, CJ, DET, ETH, EXPL, GMOD, GRAD, KOM, KON, KONJ,
 NEB, NP2, OBJA, OBJA2, OBJD, OBJG, OBJC, OBJI, OBJP, PAR,
 PART, PN, PP, PRED, REL, S, SUBJ, SUBJC, VOK, ZEIT, ’’’’;

Die Ebene REF verbindet Relativpronomen mit ihrem Antezedenten.
Andere Referenzbeziehungen, etwa PPER-->NN, werden nicht modelliert.
Labels der Referenzebene ’’’’
```

3.2.2. Externe Prädiktoren

Über die Constraints in der Grammatik lassen sich verschiedene externe Programme in den Parsingprozess einbringen. In einer Vorverarbeitung werden diese auf den Satz

angewendet, das Ergebnis fließt dann als zusätzliche Kantenbewertung über ein entsprechendes Constraint in die Verarbeitung des Parsers ein. Für den Parser selbst ist die Bewertung durch die Prädiktoren also transparent. In der für diese Arbeit verwendeten Version von CDG werden zwei externe Prädiktoren verwendet. Zum einen ist das ein Part-Of-Speech-Tagger (POS-Tagger, Wortarterkenner), zum anderen ein PP-Attacher.

Der verwendete POS-Tagger ist der TnT-Tagger [Brants, 2000], der basierend auf einem Hidden-Markov-Modell Lexemknoten aufgrund der auftretenden Wortarten bewertet. Der TnT-Tagger wurde erweitert um einen handgeschriebenen Regelsatz, der globale Constraints enthält. Ein Beispiel ist die Bedingung, dass es nur ein finites Verb in einem Satz geben kann. Diese Bedingungen lassen sich über ein lokal arbeitendes n-Gramm-Modell nicht erfassen [Foth, 2007]. Der Tagger wird über ein Prädikat in einem entsprechenden Constraint aufgerufen, das Kanten über die vom Tagger dem Dependenten zugewiesene Wahrscheinlichkeit bewertet. CDG selbst kennt keine Lexembewertungen.

Das zweite externe Modul ist der sog. PP-Attacher. Bei der Anbindung von Präpositionalphrasen gibt es eine Ambiguität, die syntaktisch meist nicht aufgelöst werden kann. Der PP-Attacher bewertet die verschiedenen möglichen Anbindungspunkte aufgrund eines lexikalisierten Modells. Das Modell enthält Informationen, wie häufig eine Kombination von Präposition, Präpositionalnomen und Ziel der Anbindung in einem Trainingskorpus vorkommt.

Neben diesen beiden können beliebig andere Module als Prädiktoren eingebunden werden, z.B. ein anderer Parser. In der vorliegenden Arbeit beschränken wir uns jedoch auf die beiden oben erwähnten Prädiktoren.

3.2.3. Analysebewertung und -vergleich

Die Grammatik erlaubt es, einer Analyse eine Bewertung zuzuweisen, indem die Strafwerte aller durch Kanten der Analyse verursachten Constraintverletzungen aufmultipliziert werden. Zwei Analysen lassen sich anhand ihrer Bewertung vergleichen. Vergleicht man zwei Analysen zum selben Satz über die Bewertung durch die Grammatik, sind drei Fälle zu unterscheiden.

- Die Bewertung ist unterschiedlich, die eine Analyse ist also besser als die andere.
- Beide Analysen sind gleich und damit auch gleich bewertet, da das Bewertungsverfahren deterministisch ist
- Der dritte Fall ist der, dass die Analysen zwar gleich bewertet, jedoch nicht gleich sind. Dieser Fall ist durch Disambiguierungsconstraints in der Grammatik weitgehend ausgeschlossen. Diese Constraints, bestrafen jeweils eine Variante so leicht, dass sie nur eine Auswirkung haben, wenn keine anderen Constraints verletzt sind.

Gegeben eine korrekte Analyse, z.B. aus einem handannotierten Korpus, gilt die Bewertung durch die Grammatik als korrekt für einen Satz, wenn die korrekte Analyse

die am höchsten bewertete der möglichen Analysen für diesen Satz ist. Ist dies nicht für alle Sätze der Fall, sprechen wir von Modellierungsfehlern in der Grammatik.

3.3. Der CDG-Parsingalgorithmus

In diesem Abschnitt wird der nichtinkrementelle Algorithmus beschrieben, der als Ausgangspunkt für die inkrementelle Erweiterung dient und auf den in späteren Kapiteln zurückgegriffen wird. Die Eingabe besteht aus dem Satz selbst, also einer Sequenz von Wörtern und Satzzeichen, allgemein Tokens, und der zu verwendenden Grammatik. Ausgabe ist eine Analyse zum Eingabesatz. Es wird dazu zunächst der Suchraum mit allen darin vorkommenden Elemente initialisiert, eine initiale Analyse erstellt und dann in einem transformationsbasierten Prozess die optimale Analyse gesucht.

3.3.1. Initialisierung des Suchraumes

Zunächst wird der Lexemgraph initialisiert, indem zu jedem Wort des Satzes alle Lesarten aus dem Lexikon der Grammatik ermittelt werden. Für jede Lesart wird dann an entsprechender Stelle ein Lexemknoten in den Lexemgraphen eingefügt. Die Anzahl der erzeugten Lexemknoten ist bei gegebener Grammatik linear abhängig von der Länge des Satzes.

Anschließend werden systematisch alle Kanten aufgebaut. Dazu wird über alle Ebenen, Kantenbeschriftungen, Lexemknoten als Dependent und Lexemknoten als Regent iteriert. Ebenen, Kantenbeschriftungen und Lexemgruppen sind dabei bei gegebener Grammatik konstant, die Anzahl der auf diese Weise erzeugten Kanten hängt also quadratisch von der Länge des Satzes ab.

Die Kanten erfahren eine initiale Bewertung, indem alle Constraints der Grammatik, die ohne Kontext auf eine einzelnen Kante angewendet werden können, ausgewertet werden. Dies betrifft alle unären, nicht kontextsensitive Constraints. Kanten die hier schon mit Null bewertet werden, also gegen ein hartes Constraint verstoßen, können sofort wieder gelöscht und von der weiteren Verarbeitung ausgeschlossen werden. Dieses Schicksal trifft einen erheblichen Anteil an Kanten, denn durch das erschöpfende kombinatorische Vorgehen bei der Kantenerzeugung werden vielfach grammatikalisch sinnlose Kanten erzeugt. Abbildung 3.2 zeigt die Anzahl der Kanten nach Länge der Eingabesequenz. Dargestellt sind die Anzahl aller erstellter Kanten und die Anzahl der Kanten, die tatsächlich im Suchraum verbleiben. Die Differenz ist die Anzahl der durch die initiale Bewertung aussortierten Kanten. Die Größen unterscheiden sich in einem Faktor von ca. 100, was die Relevanz der initialen Bewertung hervorhebt. Außerdem stellt diese initiale Bewertung eine Obergrenze dar, im Kontext kann die Bewertung nur schlechter werden, indem noch weitere Constraints verletzt werden. Über diese Obergrenze kann die spätere Suche gesteuert werden.

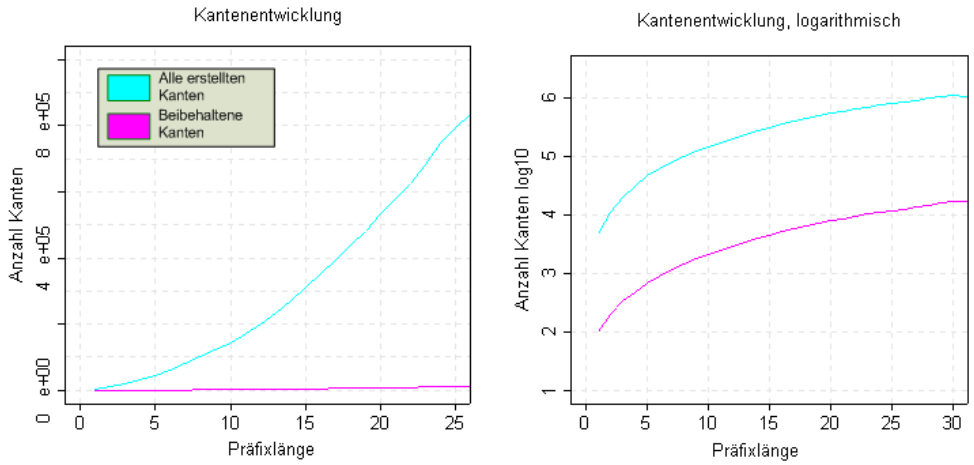

Abbildung 3.2.: Durchschnittliche Menge der Kanten im Suchraum, erstellt total und tatsächlich im Suchraum verbleibend

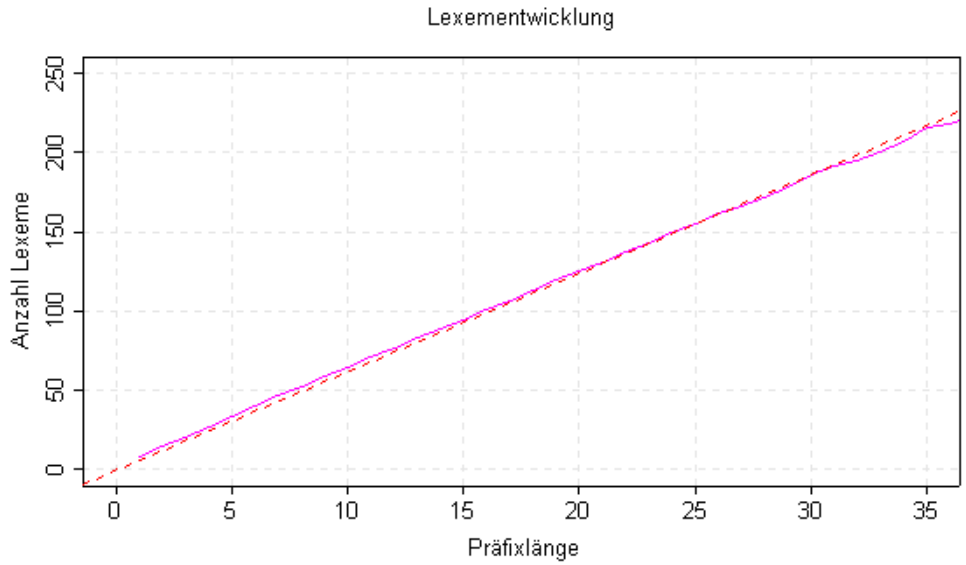

Abbildung 3.3.: Durchschnittliche Menge der Lexeme im Suchraum, nach Satzlänge

3.3.2. Initiale Analyse

Die initiale Analyse bildet den Startpunkt der anschließenden Suche. Die Auswahl der Kanten und Lexeme für diese Analyse kann prinzipiell beliebig erfolgen. Praktisch werden dabei Kanten mit hoher initialer Bewertung ausgewählt. Dieser Wert hängt insbesondere von der Bewertung der an den Enden der Kante vorkommenden Wortarten durch den POS-Tagger ab. Die initiale Analyse wird also im Wesentlichen durch den Tagger bestimmt.

Pseudoinkrementelles Parsing

Eine andere Art, die erste Analyse zu erstellen, ist es, eine andere Analyse auf einem Präfix des aktuellen Satzes heranzuziehen. Lexeme und Kanten zu den bereits im kürzeren Satz vorkommenden Wörtern werden direkt übernommen. Für die zusätzlichen Wörter werden die fehlenden Elemente wieder über die Kantenbewertung ausgewählt. Da über diese Initialisierung nur eine schwache informationelle Kopplung zwischen den inkrementellen Schritten gegeben ist, wird dieser Analyse-Modus als pseudoinkrementelles Parsing bezeichnet. Andere Informationen als die zu den der ausgewählten Kanten und Lexemen werden nicht übernommen. Ein einfaches inkrementelles Vorgehen ist also bereits im CDG-System am Ausgangspunkt dieser Arbeit möglich. Dieser pseudoinkrementelle Modus ist Ausgangspunkt und Baseline für die Implementation der vorliegenden Arbeit.

3.3.3. Transformationsbasierte Suche

Nachdem Suchraum und erste Analyse initialisiert sind, startet der eigentliche Suchalgorithmus. Dabei werden ausgehend von einer initialen Analyse Veränderungen vorgenommen, um die Bewertung zu verbessern. Bei der Veränderung einer Analyse wird dabei eine Kante durch eine andere ersetzt und die Lexeme evtl. entsprechend neu ausgewählt. Dieses Vorgehen ist also eben gerade nicht inkrementell. Von Anfang an ist bereits in jeder Domäne ein Element ausgewählt.

Für den Weg von der initialen zur abschließenden Analyse sind verschiedene Algorithmen denkbar und auch implementiert. Der im CDG-System erfolgreichste und auch bei dieser Arbeit zugrunde gelegte Ansatz bestimmt anhand der Constraintverletzungen die Domäne oder Domänen, in der eine Veränderung vorgenommen werden soll und unterstützt diese Suche gleichzeitig mit einer Beschneidung des Suchraumes. Diese kombinierte Suche trägt den Namen **Combined**.

Da mit einer einzelnen Veränderung, nicht unbedingt ein neues Maximum in der Bewertung erreicht werden kann, muss eine Kette von Veränderungen erlaubt sein, um aus lokalen Maxima zu entkommen. Dafür werden zu jeder Zeit zwei Analysen mitgeführt, die derzeit beste Analyse und ihre Arbeitskopie. Veränderungen werden nur an der Arbeitskopie vorgenommen.

Erst wenn die Bewertung der Arbeitskopie die des letzten Maximums übertrifft, wird die Arbeitskopie zur neuen besten Analyse und es wird dann mit ihrer Kopie weitergearbeitet. Andersherum, wenn keine Möglichkeit mehr besteht, die Arbeitskopie

durch weitere Veränderungen aus einem Bewertungstal zu befreien, bzw. falls eine maximale Schrittzahl erreicht wurde, wird die Arbeitskopie verworfen und durch eine neue Kopie der besten Analyse ersetzt.

Aufbauend auf der Combined-Strategie sind verschiedene Metastrategien möglich, von denen hier neben dem oben eingeführten pseudoinkrementellen Vorgehen das sog. "Threefold" erwähnt sei. Bei diesem wird der Satz zunächst anhand von Satzzeichen in Teilsätze zerlegt und für diese jeweils lokal eine Analyse ermittelt. Diese Analysen werden kombiniert und dienen als Ausgangspunkt für einen abschließenden Suchlauf über den gesamten Satz. Threefold hat eine höhere Genauigkeit als das einfache Combined auf Kosten eines höheren Zeitbedarfs.

Konfliktreparatur

Die auf der Analyse ausgeführte Transformation, ist konfliktgetrieben, d.h. es werden solche Kanten ausgetauscht, die an einem durch die aktuelle Analyse verletzten Constraint beteiligt sind. Jedes Constraint der Grammatik ist mit einem Straffaktor zwischen 0.0 und 1.0 versehen, der multiplikativ in die Bewertung einer Analyse eingeht, wenn das Constraint verletzt wurde. Die verletzten Constraints lassen sich ihrer Härte nach sortieren, es gibt also immer einen härtesten Konflikt in der Liste. Eine Ausnahme ist der Fall in dem es keine ungelösten Konflikte mehr gibt. In diesem Fall terminiert der Algorithmus.

Ein Constraint bezieht sich auf ein oder zwei Kanten. Somit sind durch den derzeit schärfsten Konflikt eine begrenzte Anzahl an Domänen gegeben. Durch Ersetzen der aktuell in diesen Domänen ausgewählten Kanten entstehen alternative Analysen, von denen die beste ausgewählt wird. Die neue Analyse muss dabei nicht besser bewertet sein als die alte. Diese Möglichkeit einer lokalen Verschlechterung der Analysebewertung ist notwendig, um aus lokalen Maxima zu entkommen. Eine solche Transformation stellt einen **Reparaturversuch** dar.

Die resultierende Analyse wird nun wieder genauso behandelt. Es wird wieder der schärfste der in ihr auftretenden Konflikte, die zum Teil durch die letzte Änderung erst eingeführt worden sein können, ausgewählt. Auf diese Weise wird eine Sequenz von Reparaturversuchen erzeugt, eine Reparaturkette. Diese wird solange erweitert, bis ein neues Maximum gefunden wurde, die maximale Schrittzahl erreicht wurde, oder keine Verbesserung mehr möglich ist. Der letzte Fall kann eintreten, wenn jede Veränderung einen vorher entfernten Konflikt wieder einführen würde. Es darf kein Konflikt, also die Kombination aus verletztes Constraint und beteiligten Kanten, der in einem Reparaturversuch behoben wurde, durch einen späteren Schritt in der Kette wieder eingeführt werden. Diese Einschränkung verhindert Zyklen in der Suche. Die Liste dieser Konflikte wird zurückgesetzt sobald ein neues Maximum gefunden wurde.

Wenn eine Reparaturkette abgebrochen werden muss, gilt der erste Konflikt am Anfang dieser Kette als unbehebbar. Ein solcher Konflikt wird in eine Liste der unbehebbaren Konflikte aufgenommen, die sog. "unremovable-conflicts"-Liste. Solange ein Konflikt in dieser Liste enthalten ist, kann er keinen Reparaturversuch initialisieren. Da er vor der erfolglosen Reparaturkette der schärfste Konflikt war, ist er es auch danach, da zu der alten Analyse zurückgesetzt wird. Er wird in Zukunft bei der Ermittlung

des schärfsten Konfliktes übergangen, es wird also nicht wieder aktiv versucht, ihn zu beheben. Sollte er dennoch als Seiteneffekt bei einem anderen Reparaturversuch behoben werden, wird er von der Liste gelöscht. Dies ist die einzige Möglichkeit einen Konflikt aus der Liste zu entfernen.

Konflikte auf diese Weise als unbehebbar zu identifizieren ist eine Heuristik. Die Suche nach Verbesserungsmöglichkeiten war nicht erschöpfend, es kann eine Analyse geben, die besser bewertet ist als das derzeitige Maximum, in der der besagte Konflikt nicht auftritt. Diese konnte durch die Suche jedoch nicht gefunden werden.

Sobald ein Analyse mit einer neuen besten Bewertung gefunden wurde, stellt ihre Bewertung ein globales Minimum für die beste mögliche Bewertung dar, die global bestmögliche Analyse ist mindestens eben so gut bewertet, der Beweis dafür ist die gerade vorliegende Analyse. Diese neue Information kann nun zum Beschneiden des Suchraumes genutzt werden.

Limits - Beschneidung des Suchraumes

Für jede Kante und jedes Lexem wird Buch geführt, wie gut eine Analyse, an der dieses Element beteiligt ist, maximal bewertet sein kann. Diese Abschätzung ist optimistisch, eine obere Schranke und wird darum auch 'Limit' genannt. Die tatsächliche Bewertung kann deutlich darunter liegen, aber niemals darüber. Aus diesem Grund kann jedes Element, dessen Limit unterhalb des derzeitigen globalen Minimums liegt, aus dem Suchraum entfernt werden. Die Limits werden wie folgt berechnet

- die Limits der Lexeme werden mit 1.0 initialisiert.
- die Limits der Kanten werden mit der initialen Bewertung, der Auswertung der unären, nicht kontextsensitiven Constraints initialisiert
- ein Lexem kann kein besseres Limit haben als die beste Kante, die es bindet
- eine Kante kann kein besseres Limit haben als das des jeweils besten Lexems jeweils für Regent und Dependent
- binäre Constraints fließen ein, wenn zwei Domänen auf der Suche nach einer Reparatur abgeglichen werden. Eine erschöpfende Auswertung der binären Constraints findet nicht statt.

Terminierung

Der Algorithmus terminiert, wenn

- keine unbehobenen, aber behebbaren und relevanten Konflikte verblieben sind.
- das globale Minimum dem globalen Maximum entspricht
- oder die Zeitbegrenzung abgelaufen ist

Ein Konflikt ist unbehebbar, wenn er in der oben erwähnten 'unremovable'-Liste enthalten ist, da vergeblich versucht wurde, ihn zu beheben. Für die Relevanz von Konflikten lässt sich eine Schwelle einstellen. Alle Konflikte deren Bewertung oberhalb dieser 'ignore threshold' liegt, erzeugen keine Reparaturversuche, sie dienen nur der Priorisierung zwischen Kanten. Das globale Maximum wird über das jeweils höchsten Limit in jeder Domäne ermittelt.

Anytime-Eigenschaft

Wie bereits erwähnt, kann die transformationsbasierte Suche, wann immer sie unterbrochen wird, eine Analyse als Ergebnis ausgeben. Dieses Ergebnis ist die bisher am besten bewertete Analyse, die jederzeit vorhanden ist, auch wenn ihre Arbeitskopie innerhalb einer Reparaturkette gerade schlechter bewertet ist. Die Bewertung der aktuell am besten bewerteten Analyse wächst also monoton. Ein Abbruchkriterium ist das Ablaufen der Zeitbegrenzung, mit dem sich also leicht ein Tradeoff zwischen zeitlicher Performanz und Ergebnisqualität einstellen lässt.

Befindet sich der Parser noch in einem der ersten beiden Schritte, Aufbau des Suchraumes oder der initialen Analyse, ist eine solche Analyse noch nicht vorhanden. Zu einem solch frühen Zeitpunkt kann der Parser nicht unterbrochen werden. Der Parser als Ganzes ist also nicht uneingeschränkt anytime-fähig.

4. Inkrementalität

Bevor wir uns der Frage widmen, wie sich das CDG-System um eine inkrementelle Vorgehensweise erweitern lässt, soll hier zunächst die Frage geklärt werden, was eine inkrementelle Vorgehensweise ausmacht.

Ein **inkrementeller Dependenz-Parsing-Algorithmus** ist ein Parsingalgorithmus, der, gegeben ein Eingabesatz, eine Syntaxanalyse derart berechnet, dass er in mehreren Schritten eine Sequenz von Zwischenergebnissen in Form von Analysen für einen jeweils größer werdendes Präfix des Satzes berechnet. Zur Berechnung der Analyse zu einem Präfix wird auf kein Wort des Satzes außerhalb des entsprechenden Präfixes zugegriffen.

Mit Hilfe dieser Zwischenergebnisse soll erreicht werden, dass die jeweils nächste Analyse unter Rückgriff auf die vorherige schneller berechnet werden kann, als wenn diese Analyse nicht vorliegen würde. Dadurch, dass der Vorgriff auf andere Wörter des Satzes ausgeschlossen ist, kann das entsprechende Zwischenergebnis bereits berechnet werden, bevor der Rest des Satzes vorliegt.

4.1. Formalen Definition

Um die oben aufgeführten Anforderungen an einen inkrementelles Vorgehen formal zu erfassen, abstrahieren wir zunächst von der Domäne des Syntaxparsings und definieren:

Datenanalyse

Eine **Analysefunktion** sei eine Funktion, die gegeben eine Menge von Eingabewerten als **Analyseergebnis** eine Abbildung ausgibt, die jedem Element der Eingabemenge einen Wert aus einer Ergebnismenge zuweist.

Gegeben die Menge der möglichen Eingabewerte M_{in} und die Menge der möglichen Ausgabewerte M_{out} ist eine Analysefunktion F also beschrieben durch

$$F : \mathcal{P}(M_{in}) \rightarrow (m \rightarrow M_{out}), m \in \mathcal{P}(M_{in})$$

wobei $\mathcal{P}(x)$ die Potenzmenge von x ist.

Sequenzanalyse

Zusätzlich zur obigen Definition sei bei der Sequenzanalyse die Reihenfolge der Eingabe relevant, es handelt sich also um eine Eingabesequenz. Die Eingabe ist beim Dependenz-Parsings der zu parsende Satz. Die Eingabewerte sind Wörter $\in \Sigma^*$ für ein Alphabet Σ und die Eingabemenge eine Sequenz von Wörtern, $M_{in} = \Sigma^*$. Ein Ausgabewert zu einem Wort besteht aus

- dem ausgewählten Lexem
- einer Kante für jede Ebene der Analyse, die wiederum aus Regent und Beschriftung besteht

Erweiterte Eingabe, direkter Nachfolger

I_{ex} ist eine **Erweiterung der Eingabe** I, wenn $I_{ex} \supset I$. I_n ist ein **direkter Nachfolger** von I_{n-1}, wenn I_n Erweiterung von I_{n-1} ist und $|I_n| - |I_{n-1}| = 1$, also um genau ein Element erweitert ist.

Inkrementelle Erweiterung

Gegeben eine bestehendes Analyseergebnis $F : I \rightarrow O$ und eine erweiterte Eingabemenge I_{ex} mit $I_{ex} \supset I$, dann ist eine **inkrementelle Erweiterung** von F eine Abbildung $F_{ex} : I_{ex} \rightarrow O_{ex}$.

Konservative Erweiterung

Sei $F_{ex} : I_{ex} \rightarrow O_{ex}$ eine Erweiterung des Analyseergebnisses $F : I \rightarrow O$, dann ist F_{ex} eine **konservative Erweiterung** von F, wenn $\forall i \in I$ gilt $F_{ex}(i) = F(i)$

Analysealgorithmus

Ein **Analysealgorithmus** ist ein Algorithmus, der die Analysefunktion berechnet. Die **Genauigkeit** eines Analysealgorithmus A auf einer Testmenge X und einer Vergleichs-Analysefunktion V ist der Prozentsatz an Elementen von X auf denen A und V das selbe Ergebnis berechnen. Der **Zeitbedarf** eines Analysealgorithmus bei gegebener Eingabe ist die Zeit, die zur Berechnung des Analyseergebnisses benötigt wird.

Inkrementeller Analysealgorithmus

Ein **inkrementeller Analysealgorithmus** ist ein Analysealgorithmus, dessen Vorgehen sich wie folgt in mehrere Iterationen oder inkrementelle Schritte aufteilen lässt:

Für eine initiale Eingabesequenz wird ein initiales Analyseergebnis berechnet. Für eine Reihe von inkrementellen Erweiterungen der jeweils vorangehenden Eingabesequenz und dem vorangehenden Analyseergebnis wird, ausgehend von der initialen Eingabe, jeweils ein Analyseergebnis berechnet. Die Eingabemenge der finalen Iteration entspricht dabei der Eingabemenge des Algorithmus.

Formaler: ein inkrementeller Algorithmus mit der Eingabe I_{ges} besteht aus

- einer Zerlegung von I_{ges} in eine Sequenz $[I_0, ..., I_n]$ mit $I_n = I_{ges}$.
- einer Initialisierungsfunktion $F_{init} : I_0 \rightarrow (O_0)$
- einer Iterationsfunktion $F_{iter} : (I_{n+1}, O_n) \rightarrow (O_{n+1})$. Sie bestimmt also aus der Erweiterung der Eingabe die Erweiterung des Ergebnisses.

Eine inkrementeller Analysealgorithmus ist:

Ergebnisäquivalent zu einem anderen Analysealgorithmus, wenn das Analyseergebnis der finalen Iteration dem Analyseergebnis des anderen Algorithmus entspricht. Der andere Algorithmus kann ebenfalls ein inkrementeller Algorithmus sein, in diesem Fall ist dessen Analyseergebnis das Ergebnis von dessen finaler Iteration.

Inkrementell zeitüberlegen gegenüber einem anderen Analysealgorithmus, wenn der Zeitbedarf der finalen Iteration kleiner ist als der gesamte Zeitbedarf des anderen Algorithmus.

Atomar , wenn gilt $|I_{init}| = 1$ und jede Eingabeerweiterung I_{n+1} im Iterationsschritt einen direkten Nachfolger von I_n darstellt.

Links-rechts-sequenziell , wenn seine Eingabemenge eine Sequenz $I_{ges} = \{i_0, i_1, ..i_n\}$ ist und für jede Iteration $k \leq n$ gilt $I_k = i_0, .., i_k$, also die Eingabemenge der Iteration ein Präfix der Länge k der Gesamteingabe ist.

Gegeben ein Eingabesatz lassen sich bestimmte graduelle Eigenschaften für einen Analysealgorithmus definieren.

Ergebnisähnlichkeit Ein Parsingalgorithmus hat für einen gegebenen Satz eine höhere Ergebnisähnlichkeit zu einem anderen, je mehr Kanten und Lexeme ihre jeweiligen Analysen gemeinsam haben.

Genauigkeit Ein inkrementeller Parsingalgorithmus ist um so genauer für einen gegebenen Satz mit einer gegebener Goldstandard-Analyse, je mehr Kanten und Lexeme sein Ergebnis mit dem Goldstandard gemein hat.

Konservativität Ein Algorithmus ist um so **konservativer**, je weniger Kanten und Lexeme, aus einem seiner Zwischenergebnisse in einem späteren Ergebnis ersetzt worden sind.

Diese Eigenschaften gelten immer pro Satz oder gemittelt für eine Menge von Sätzen.

4.2. Andere Definitionen von Inkrementalität

Nivre [2008] definiert **deterministisches inkrementelles Parsing** über die Monotonität der Kantenmenge in der Analyse. Und zwar fordert er, dass eine Kante, die in einem Zwischenergebnis vorhanden ist, auch in jedem weiteren enthalten ist, also absolute Konservativität.

Zu beachten ist, dass wir hier im Gegensatz dazu keinen solchen Determinismus voraussetzen, d.h. eine in einem Schritt zum aktuellen Zwischenergebnis hinzugefügte Kante kann in einem späteren Zwischenergebnis fehlen. Reanalysen sind explizit erlaubt. Darüber hinaus ist bei unserer Definition gegenüber Nivre [2008] ein Lookahead explizit ausgeschlossen. Jedes Zwischenergebnis ist einem Satzpräfix zugeordnet und

zur Berechnung der jeweiligen Analyse wird auf kein Wort außerhalb dieses Präfixes zurückgegriffen.

Nivre [2008] definiert Inkrementalität also über die Inkrementalität des Aufbaus der Analyse, während die hier vorgestellte Definition sich auf die Inkrementalität der Verarbeitung der Eingabe bezieht.

4.3. Zeitverhalten

Beim Ausführen eines Analysealgorithmus lassen sich einige markante Zeitpunkte definieren

E_i Element i der Eingabesequenz liegt vor

E_{compl} Eingabesequenz liegt komplett vor

S_i Start des i-ten inkrementellen Schrittes

$S_{non-incr}$ Start des nichtinkrementellen Analysealgorithmus

F_i Termination des i-ten inkrementellen Schrittes

$F_{non-incr}$ Termination des nichtinkrementellen Analysealgorithmus

Für jeden inkrementellen Schritt in der inkrementellen Verarbeitung gilt, dass die entsprechende Eingabe vor Beginn des Schrittes vorliegen muss. Des Weiteren muss in jedem Schritt als Eingabe die Ausgabe des vorherigen vorliegen. Wir gehen von einer seriellen Bearbeitung der inkrementellen Schritte aus.

$$S_i \geq E_i \wedge S_i \geq F_{i-1}$$

Der Startzeitpunkt hängt vom späteren der beiden genannten Zeitpunkte ab, wobei der Zeitpunkt der Eingabe von außen vorgegeben ist. Der Endzeitpunkt eines Schrittes kann vom System selbst bestimmt werden, wobei hier zwei Ausführungsmodelle denkbar sind.

pull Der Endzeitpunkt wird durch die Termination des inkrementellen Schrittes bestimmt, die evtl. schon vorliegende neue Eingabe wird so lange ignoriert, bis der Algorithmus den aktuellen Schritt von sich aus terminiert.

push Die nächste Iteration beginnt, sobald das nächste Element der Eingabe eintrifft, die aktuelle Iteration wird schnellstmöglich abgeschlossen. Für dieses Vorgehen wird also eine Anytime-Fähigkeit des Algorithmus vorausgesetzt, er muss sich jederzeit von außen unterbrechen lassen und trotzdem ein Analyseergebnis ausgeben.

Im pull-Vorgehen wird Zeit darauf verwendet, Analyseergebnisse für eine unvollständige Eingabe zu berechnen, obwohl bereits eine erweiterte Eingabe vorliegt. Dieses Vorgehen erzeugt vollständig berechnete Analyseergebnisse aller Zwischenschritte.

Sind wir jedoch nur am Analyseergebnis der finalen Iteration interessiert, wird evtl. Zeit verschwendet.

Im push-Vorgehen wird diese Zeit nicht verschwendet. Ein vorzeitig abgebrochener anytime-fähiger Algorithmus kann jedoch kein optimales Analyseergebnis garantieren. Dieses Vorgehen ist also nicht geeignet, wenn wir an den Zwischenergebnissen interessiert sind. Vor allem muss die Iterationsfunkion des inkrementellen Algorithmus mit suboptimalen Analyseergebnissen als Eingabe zurecht kommen. Dies kann sich einerseits direkt auf die Qualität der Ausgabe des nächsten Schrittes auswirken. Andererseits kann es den Zeitbedarf der nächsten Iteration erhöhen, und damit durch ein vorzeitiges Abbrechen bei eintreffen des nächsten Eingabeelementes ebenfalls die Qualität des nächsten Analyseergebnisses indirekt vermindern.

Im klassischen, nichtinkrementellen Ausführungsmodell muss die Eingabe vor Beginn der Ausführung vollständig vorliegen, also $S_{non-incr} \geq E_{compl}$. Ob ein inkrementeller Algorithmus das oben definierte Ziel der inkrementellen Zeitüberlegenheit gegen über dem nichtinkrementellen Algorithmus erreicht, hängt nicht zuletzt ab von der Zeit, die zwischen dem Eintreffen des ersten und des letzten Elementes der Eingabe vergeht.

Um zwei Extreme zu nennen:

Steht beliebig viel Zeit in der vorletzten Iteration zur Verfügung, könnten bei einer endlichen Menge von möglichen Fortsetzungen der Eingabe alle diese Möglichkeiten vorweg durchgerechnet werden. In der letzten Iteration müsste nur noch das Ergebnis für die tatsächlich eingetroffene Erweiterung ausgewählt werden.

Im anderen Extremfall kann ein inkrementeller Algorithmus bei gleichzeitigem oder im Verhältnis zur Gesamtlaufzeit praktisch gleichzeitigem Eintreffen der Eingabeelemente keinen Verarbeitungsvorsprung aufbauen.

Die Performance bzw. die Eignung eines inkrementellen Algorithmus muss also im Kontext gesehen werden.

5. Inkrementelles Constraint-Dependency Parsing

In den vorangegangenen Kapiteln wurde sowohl das nichtinkrementelle, transformationsbasierte Constraint-Dependency-Parsing vorgestellt, als auch der Begriff des inkrementellen Analysealgorithmus eingeführt. In diesem Kapitel wird nun beschrieben, wie diese beiden Konzepte zu einem inkrementellen Constraint-Dependency-Parsing zusammengeführt werden können.

Das Ziel ist ein Algorithmus zur inkrementellen Analyse natürlichsprachlicher Sätze, der gegenüber dem vorhandenen, nichtinkrementellen Algorithmus

- ergebnisäquivalent
- inkrementell zeitüberlegen

ist.

5.1. Allgemeines inkrementelles Vorgehen

Das allgemeine inkrementelle Vorgehen, zu sehen in Abbildung 5.1, besteht aus einer Wiederholung der Prozeduren "Eingabe erzeugen/erweitern" und "Analyse erzeugen/erweitern". Von beiden Prozeduren wird eine initiale und eine inkrementelle Variante benötigt, je nachdem ob ein vorheriges Element vorliegt oder nicht. Im ersten Schritt werden die initialen Varianten verwendet. Für alle weiteren Schritte werden die inkrementellen Varianten verwendet.

Abbildung 5.1.: Inkrementeller Algorithmus 1, allgemeine inkrementelle Analyse

- bestimme die initiale Teileingabe E_0
- erzeuge Analyse A_0 für E_0
- WHILE (aktuelle Eingabe $\neq$ vollständige Eingabe)
  - erweitere die aktuelle Eingabe, $E_{n-1} \rightarrow E_n$
  - erzeuge Analyse aus alter Analyse und erweiterter Eingabe, $(A_{n-1}, E_n) \rightarrow A_n$
- ENDWHILE

5.2. Inkrementelles Parsing natürlicher Sprache

In der Sprachverarbeitung ist die Eingabe eine Sequenz von Worten und Satzzeichen.

5.2.1. Inkrementelle Eingabe

Die erste Aufgabe der inkrementellen Verarbeitung ist es, die Eingabesequenz in Teileingaben zu zerlegen. Dabei stellen sich die Frage nach Reihenfolge und nach der Granularität der Inkremente. Sprache wird für gewöhnlich sowohl in schriftlicher als auch mündlicher Modalität von vorne nach hinten produziert. In diesen Fällen kann man also die Erweiterung der Eingabe auf eine Erweiterung am Ende der bisherigen Eingabesequenz beschränken.

Denkbar sind auch Erweiterungen in der Mitte des Satzes, etwa bei der Korrektur eines geschriebenen Satzes. Solche Einschübe sind aber eben nicht als inkrementelle Kommunikation einer von vornherein gegebenen Struktur zu sehen. Alle Annahmen über die bisher nur teilweise offenbarte Struktur können ungültig werden. Um eine Korrektur zu unterstützen, müssten neben Einschüben auch Löschungen und Ersetzungen berücksichtigt werden. Wir beschränken uns daher hier auf Erweiterungen am Ende des Satzes. Jede Eingabe ist also ein Präfix ihrer Erweiterung, die Verarbeitung erfolgt links-rechts-sequenziell.

Die zweite Frage ist die nach der Granularität der einzelnen Erweiterungen. Als Größe für ein Inkrement kommen Wörter, Phrasen und Teilsätze in Frage. Je kleiner die Inkremente, desto früher können sie jeweils in die Bearbeitung eingehen. Ein mehrere Worte umfassendes Inkrement wie eine Phrase kann erst eingebunden werden, sobald ihr letztes Wort verfügbar ist, alle ihre anderen Wörter gehen später als möglich ein.

Anderseits steigt bei kleinere Inkrementen die Wahrscheinlichkeit, dass für ein Wort der gesuchte Regent noch nicht verfügbar ist. Wird etwa eine Phrase Wort für Wort aufgebaut, treten zunächst viele Ambiguitäten auf, die, sobald die Phrase vollständig vorliegt, leicht aufgelöst werden können. Bei einer Nominalphrase etwa könnte der zunächst allein stehende Artikel ein Pronomen als eigenständige Nominalphrase oder der Artikel zu einem kommenden Nomen in einer noch nicht vollständigen Nominalphrase sein. Auch muss, je kleiner die Inkremente sind, öfter überprüft werden, ob ein Konflikt in der bisherigen Analyse mit der erweiterten Eingabe gelöst werden kann, was dem Zeitgewinn durch eine frühere Betrachtung der Wörter bei kleinen Inkrementen entgegenwirkt. Im parser-gesteuerten "pull"-Betrieb erhöht sich dadurch die Gesamtzeit, im input-gesteuerten "push"-Betrieb kann die Zeit zwischen dem Eintreffen der einzelnen Wörter unzureichend sein, um alle Konflikte nach jedem Wort zu überprüfen.

Ein weiterer Aspekt ist, mit welchem Aufwand sich die Inkrementgrenzen ermitteln lassen. Phrasen sind hier problematisch, da Strukturen dieser Art ja erst vom Parser erkannt werden. Die Phrasen könnten durch flaches Parsing in einer Vorverarbeitung erkannt werden.

Wir werden uns hier zunächst auf einzelne Wörter als Inkremente beschränken. Der inkrementelle Algorithmus arbeitet daher atomar und links-rechts-sequentiell. Wir

Der CDG-Parsingalgorithmus

- Erzeuge Suchraum, bestehend aus Lexemknoten und Kanten
- Erzeuge initiale Analyse A
- transformationsbasierte Suche ausgehend von A

Die transformationsbasierte Suche:

- WHILE (signifikante, behebbare Konflikte verbleiben AND Zeit nicht abgelaufen AND aktuelle Bewertung < potentiell beste Bewertung)
  - Verändere Domänen, die am aktuell schwersten Konflikt beteiligt sind
  - IF (neues Maximum gefunden)
    - speichere aktuelle Analyse als neues Maximum
    - entferne Kanten und Lexeme, deren Bewertungsobergrenze kleiner ist als das neue Maximum (Beschneidung des Suchraumes) und berechne diese Obergrezen neu
  - ENDIF
  - IF (Konflikt nicht reparierbar)
    - setze zum letzten Maximum zurück
    - und übergehe den aktuellen Konflikt in Zukunft (unremovable Liste)
  - ENDIF
- ENDWHILE

Abbildung 5.2.: Der CDG-Parsingalgorithmus

können also statt Eingabe konkret von Satzpräfix sprechen, wobei die initiale Eingabe nur aus dem ersten Wort besteht.

5.2.2. Inkrementelle Analyse

Die zweite Aufgabe ist die Ermittlung einer Analyse zu einer gegebenen Eingabe. Diese Funktionalität ist bereits durch den CDG-Parser abgedeckt, dessen Vorgehen in Abbildung 5.2 zusammengefasst ist.

Der CDG-Algorithmus und die Wort-für-Wort-Erweiterung der Eingabe können in den allgemeinen inkrementellen Algorithmus aus Abbildung 5.1 eingesetzt werden. Der daraus resultierende Algorithmus (vergl. Abbildung 5.3) realisiert ein Präfixparsing. Zu jedem Präfix des Eingabesatzes wird eine Analyse bestimmt. Die einzelnen Schritte sind dabei unabhängig. Auf die Analyse des vorangegangenen Präfixes wird nicht zurückgegriffen.

Dieser Algorithmus ist garantiert ergebnisäquivalent und garantiert nicht inkremen-

Abbildung 5.3.: Inkrementeller Algorithmus 2, Präfixparsing

- Nimm das erste Wort als Eingabe E
- Erzeuge Analyse für E
  - Erzeuge Suchraum P für E
  - Erzeuge initiale Analyse A für P
  - Suche Analyse ausgehend von A
- WHILE (Präfix kleiner als Satz)
  - erweitere das Präfix um ein Wort zu neuer Eingabe E
  - Erzeuge Suchraum P für E
  - Erzeuge initiale Analyse A für P
  - Suche in P ausgehend von A
- ENDWHILE

tell zeitüberlegen gegenüber dem nichtinkrementellen CDG-Parsing. Jeder Schritt entspricht dem nichtinkrementellen Vorgehen für das entsprechende Präfix. Es wird nicht auf vorangegangene Zwischenergebnisse zurückgegriffen. Die letzte Iteration, bei der die Eingabe dem gesamten Satz entspricht, erzeugt also die gleiche Analyse und dauert eben so lange wie das nichtinkrementelle Vorgehen.

Das Präfixparsing ist für praktische Belange uninteressant, alle Iterationen vor der letzten haben keinen Einfluss auf das Endergebnis. Die Zwischenergebnisse dieser Iterationen und die Zeitdauer sie zu berechnen bieten jedoch einen Vergleichsmaßstab für andere inkrementelle Verfahren.

5.2.3. Unspezifizierte Anbindungen

Um unvollständige Sätze beschreiben zu können, ist ein dedizierter unspezifizierter Anbindungspunkt hilfreich. Für ein Wort, das im vollständigen Satz einem weiter rechts stehenden Wort untergeordnet ist, steht im entsprechenden Satzpräfix dieser Anbindungspunkt noch nicht zur Verfügung. Um eine grammatikalisch nicht sinnvolle Anbindung an ein ungeeignetes, aber bereits zur Verfügung stehendes Wort zu verhindern, kann das Wort temporär, d.h. für die Analyse dieses Präfixes dem unspezifizierten Knoten untergeordnet werden.

Gegeben seien ein vollständiger Satz, eine Dependenzstruktur zu diesem und ein unvollständiges Präfix dieses Satzes. Erstellt man eine Dependenzstruktur zu dem Präfix, indem man aus einer vollständigen Dependenzstruktur alle Kanten entfernt, deren Regent oder Dependent im Präfix fehlt, lassen sich diese Kanten in drei Klassen einteilen (vergl. Abbildung 5.4).

1. Regent und Dependent der Kante liegen außerhalb des Präfixes

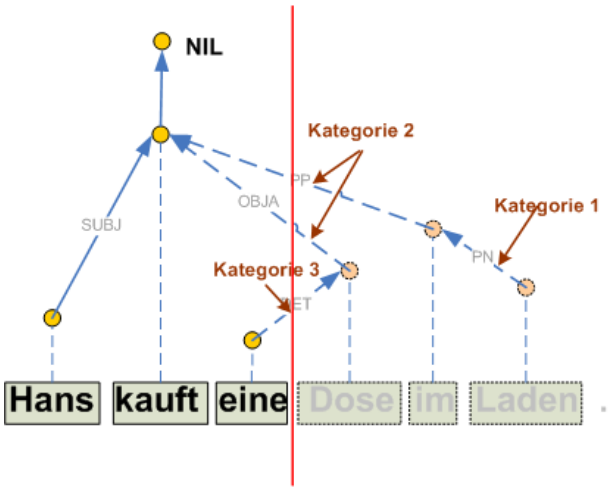

Abbildung 5.4.: Dependenzstruktur bei unvollständigem Satz

2. der Regent liegt im Präfix, der Dependent außerhalb
3. der Dependent liegt im Präfix, der Regent außerhalb

Kanten der ersten Kategorie betreffen kein Wort des Präfixes, ihre Abwesenheit verletzt nicht die Integrität der Dependenzstruktur. Die Abwesenheit einer Kategoriezwei-Kante könnte zu einer Constraintverletzung führen, wenn ein Constraint die Anwesenheit des Dependenten verlangt. Eine Valenz des Regenten würde also im Präfix unerfüllt bleiben oder müsste durch ein anderes geeignetes Wort erfüllt werden. Die Dependenzstruktur bleibt in beiden Fällen wohlgeformt Die Abwesenheit einer Kategoriedrei-Kante verletzt jedoch die grundlegende Forderung, dass jedes Wort einem Regenten untergeordnet sein muss. Unabhängig von Constraints der Grammatik muss also in der Dependenzstruktur zum Präfix eine alternative Kante für dieses Wort ausgewählt sein, auch wenn eine Anbindung an eines der verfügbaren Wörter oder die NIL-Anbindung durch die Constraints der Grammatik nicht erlaubt oder stark bestraft sind. Die Abwesenheit eines solchen geeigneten Anbindungspunktes kann als eine Erwartung interpretiert werden, dass der noch fehlende Teil des Satzes ein geeignetes Wort enthält.

Um diese Erwartung geeignet ausdrücken zu können, wird ein zusätzlicher möglicher Regent eingeführt, der alle in der Zukunft liegenden Wörter repräsentiert. Da diese Wörter noch unspezifiziert sind, wird dieser Anbindungspunkt als **NONSPEC** bezeichnet. NONSPEC besitzt keinerlei grammatikalische Attribute wie Wortart oder Kasus. Lediglich die Position ist als rechts von allen vorhandenen Wörtern liegend spezifiziert. Dem liegt die Annahme zugrunde, dass der noch unspezifizierte Teil rechts liegt, der unvollständige Satz also nur rechts erweitert wird. Ein Beispiel für eine Dependenzstruktur mit NONSPEC für ein Satzpräfix zeigt Abbildung 5.5.

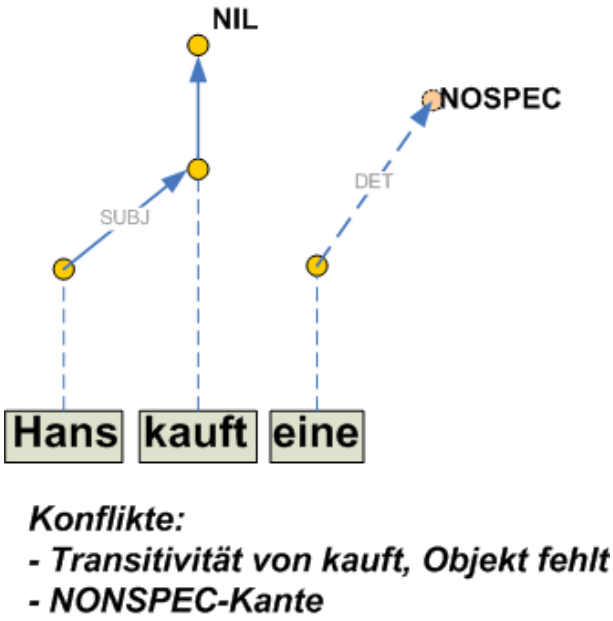

Abbildung 5.5.: Dedendenzstruktur mit NONSPEC-Kante

NONSPEC steht als Regent zur Verfügung, um eine artifizielle Unterordnung an eine anderes, ungeeignetes Wort zu verhindern. Es steht nicht als Dependent zur Verfügung, da die Abwesenheit einer Kante und eine dadurch auftretende Constraintverletzung eine adäquate Repräsentation der Erwartung an die weiteren Wörter des Satzes ist. Der hier vorgestellte NONSPEC-Mechanismus ist im CDG-System am Ausgangspunkt dieser Arbeit bereits vorhanden.

NONSPEC vs NIL

Mit NIL ist bereits ein Knoten vorhanden, der keinem Wort im Satz entspricht, und damit als Kandidat für temporäre Anbindungen in Frage kommt. NIL steht für die Wurzel des Baumes. Sind mehrere Wörter hier angebunden, haben wir es mit mehreren Sätzen zu tun. Ist etwas anderes als ein finites Verb dort untergeordnet, sprechen wir von Satzfragmenten. In den Constraints der Grammatik werden oft explizit NIL Kanten von nicht-NIL-Kanten unterschieden. Kanten nach NIL haben keine Ausrichtung (linksgerichtet VS rechtsgerichtet), im Gegensatz zu Kanten nach NONSPEC, die immer rechtsgerichtet sind.

Würde NIL als temporärer Anbindungspunkt verwendet und eine temporäre NIL-Kante gegen die später mögliche geeignete Kante ausgetauscht, würde dies die Struktur des Baumes und damit den Kontext für andere Kanten evtl. grundlegend ändern. Es sollten jedoch durch das Ersetzen einer temporären Kante nur vorher unspezifizierte

Eigenschaften spezifiziert werden, an NIL als Regent werden jedoch durch die Grammatik andere Forderungen gestellt als an die Wörter des Satzes.

Außerdem kann in der Grammatik nicht zwischen temporären und nicht temporären NIL-Anbindungen unterschieden werden. Der Straffaktor für temporäre Kanten wäre also nicht unabhängig einstellbar. Damit ist NIL als temporärer Regent ungeeignet.

NONSPEC-Spezifische Änderungen der Grammatik

Damit eine Grammatik robust mit NONSPEC-Anbindungen umgehen kann, müssen bestimmte Aspekte berücksichtigt werden. Es muss verhindert werden, dass NONSPEC als Anbindungspunkt gewählt wird, wenn ein geeigneter anderer Anbindungspunkt zur Verfügung steht. Dazu wird ein Constraint eingeführt, das eine Kante nach NONSPEC pauschal leicht bestraft.

Abfragen bestimmter Eigenschaften sind am NONSPEC-Knoten nicht möglich, wie z.B. die Frage nach Wortart, Numerus oder Kasus. Da NONSPEC als Platzhalter für einen geeigneten Regenten steht, muss er alle durch entsprechende Constraints geforderten Eigenschaften aufweisen. Der Aufruf eines entsprechenden Prädikats auf NONSPEC kann jedoch nicht pauschal mit *true* ausgewertet werden. Das Problem ist, dass das Prädikat auch verneint aufgerufen werden könnte, so könnte ein Constraint einen Regenten verlangen, der KEIN Nomen ist. NONSPEC leistet auch dies, es kann für ein Wort stehen, das, in diesem Beispiel, kein Nomen ist. Die einfachste Lösung ist es, die intendierte Behandlung von NONSPEC in den entsprechenden Constraints selbst festzulegen. In der Grammatik werden Terme, die für NONSPEC einen bestimmten Wert annehmen sollen, durch Guards wie dem disjunktiven Zusatz "nicht-spezifiziert(X) ODER (...)" erweitert.

5.3. Integration der jeweils vorangegangenen Analyse

Um eine inkrementelle Zeitüberlegenheit zu erreichen, müssen Entscheidungen aus der letzten Iteration in die früheren inkrementellen Schritte vorgezogen werden, bzw. diese Entscheidungen (Auswahl von Kanten und Lexemen) aus dem jeweils vorangegangenen Schritt müssen im jeweils nächsten Schritt berücksichtigt werden.

Der nächste logische Schritt ist es also, die initiale Analyse $A_{n,init}$ jedes Schrittes anhand der finale Analyse $A_{n-1,fin}$ des vorangegangenen Schrittes zu erzeugen. Konkret bedeutet dies, dass jede in $A_{n-1,fin}$ vorkommende Kante und jeder Lexemknoten auch in $A_{n,init}$ ausgewählt wird. Nur die Kanten und der Lexemknoten für das neu hinzugekommene Wort können nicht auf diese Weise ausgewählt werden. Diese werden auf andere Weise, über die initiale Kantenbewertung ausgewählt. Das daraus resultierende pseudoinkrementelle Parsing ist in Abbildung 5.6 dargestellt.

Unter der Annahme, dass die Analyse des vorangegangenen Schrittes in der Mehrzahl der Fälle näher an der gesuchten Analyse ist als eine herkömmlich initialisierte Analyse, kann davon ausgegangen werden, dass ein pseudoinkrementeller Schritt auch weniger Reparaturschritte und damit weniger Zeit beansprucht, als der entsprechende Schritt im Präfixparsing. Das pseudoinkrementelle Vorgehen ist inkrementell

- Nimm das erste Wort als Eingabe E
- Erzeuge Analyse für E
  - Erzeuge Suchraum P für E
  - Erzeuge initiale Analyse A für P
  -
- WHILE (Präfix kleiner als Satz)
  - erweitere das Präfix um ein Wort zu neuer Eingabe E
  - Erzeuge Suchraum P für E
  - Erzeuge initiale Analyse A für P **aus der Analyse des letzten Schrittes**
  - Suche in P ausgehend von A
- ENDWHILE

Abbildung 5.6.: Inkrementeller Algorithmus 3, Pseudoinkrementelles Parsing

zeitüberlegen für jeden Satz, bei dem diese Annahme zutrifft.

Da sich nur der Ausgangspunkt der Suche ändert, der Suchraum jedoch der gleiche bleibt wie im Präfixparsing, ist auf den ersten Blick eine Ergebnisäquivalenz gesichert. Es ist jedoch zu beachten, dass der Suchalgorithmus aufgrund der verwendeten Heuristiken nicht garantieren kann, die Analyse mit der optimalen Bewertung zu finden. Bei zwei Suchläufen mit leicht unterschiedlichen Ausgangsbedingungen kann also prinzipiell das Ergebnis der Suchprozesse voneinander abweichen. Diese Fluktuation ist aber *a*) relativ selten und *b*) kann die Bewertung in beide Richtungen abweichen: das inkrementelle Parsing könnte eine bessere Analyse finden als das nichtinkrementelle und umgekehrt. Die Auswirkungen sind in Tabelle 5.1 zu sehen.

gleiches Endergebnis	$59,6\%$
Pseudo besser	$22,3\%$
Präfix besser	$18,1\%$
Verhältnis des Zeitbedarfs (Pseudo zu Präfix, bei gleichem Ergebnis)	0.828

Tabelle 5.1.: Vergleich Präfixparsing vs pseudoinkrementelles Parsing

5.3.1. Probleme des pseudoinkrementellen Parsings

Während das pseudoinkrementelle Parsing einen Fortschritt gegenüber dem Präfixparsing darstellt, lässt es doch noch viele Zeitersparnispotentiale ungenutzt:

1. Vollständiger Neuaufbau des Suchraumes in jedem inkrementellen Schritt
2. interne Informationen zur Steuerung der Suche bleiben ungenutzt

3. keine weitere Beschneidung des Suchraumes

Die Suche selbst macht nur einen Teil des Zeitbedarfs aus. Ein bedeutender Teil der Bearbeitungszeit wird für den Aufbau des Suchraums aufgewendet, insbesondere für die initiale Kantenbewertung. Die Menge der Kanten in Schritt $n+1$ ist eine Obermenge der Kanten in Schritt n, im pseudoinkrementellen Vorgehen werden diese Kanten jedoch verworfen und neu erstellt. Werden diese Kanten stattdessen übernommen, wird die zur initialen Kantenbewertung benötigte Zeit eingespart. Dieses integrierte inkrementelle Parsing ist in Abbildung 5.7 zu sehen.

Außerdem benutzt der CDG-Parser, wie in vorangegangenen Kapiteln beschrieben, verschiedene Informationen zur Steuerung der Suche. Dazu gehören die Liste der unbehebbaren Konflikte und die Bewertungsobergrenzen der Kanten und Lexemknoten. Diese Erkenntnisse über den Suchraum werden im pseudoinkrementellen Parsing nach jedem Schritt komplett verworfen. Wenn die in diesen Informationen enthaltenen Erkenntnisse (zumindest teilweise) gültig bleiben, könnte durch sie weitere Zeit eingespart werden.

Der dritte Punkt bezieht sich darauf, dass im pseudoinkrementellen Parsing zu jeder in einem früheren Schritt getroffene Entscheidung alle Alternativen weiterhin verfügbar sind, d.h. alle Kanten weiterhin im Suchraum vorhanden sind. Im Idealfall lassen sich im inkrementelle Vorgehen Entscheidungen als stabil identifizieren, so dass sich der Suchraum um die Alternativen zu dieser Entscheidung, also insbesondere Kanten der gleichen Domäne, beschneiden lässt. Ein kleinerer Suchraum kann an verschiedenen Stellen zu Zeitersparnissen führen, zum einen bei einem Reparaturversuch, wenn bei der Auswahl der besten Alternative zu einer Kante weniger Kanten in einer Domäne vorhanden sind. Zum anderen führen weniger Lexeme dazu, dass weniger Kanten im nächsten Schritt zusätzlich erzeugt werden müssen. Dies betrifft alle Kanten mit dem aus dem Suchraum entfernten Lexemknoten an einem, und einem Lexem des neuen Wortes am anderen Ende. Dies spart direkt Zeit beim Kantenaufbau und indirekt durch die Auswirkung der reduzierten Kantenanzahl.

Die Annahme, die einem solchen Pruning zugrunde liegt, ist die, dass bestimmte lokal optimale Strukturen auch global, d.h. im ganzen Satz optimal sind. Ob eine Struktur grundsätzlich stabil ist, hängt von den Constraints der Grammatik ab. Ob eine grundsätzlich mögliche Instabilität praktisch relevant ist, hängt vom Sprachgebrauch ab. Stabile Strukturen lassen sich also einerseits über eine Analyse der Grammatik, andererseits über empirische Untersuchungen auf einem Korpus ermitteln. Die Gefahr des Prunings ist, dass ein Element aus dem Suchraum entfernt werden könnte, das in einem späteren Schritt an der optimalen Analyse beteiligt ist. Es würde verhindert, dass der Parser diese Analyse erzeugen kann. Die Beschneidung des Suchraumes ist also evtl. mit einem Kompromiss zwischen Gewinnen bei der zeitlichen Performanz und Einbußen bei der Korrektheit verbunden.

5.3.2. Übernahme aller Elemente des Suchraums

Anstatt die Elemente des Suchraums neu aufzubauen, wird die alte Datenstruktur beim integrierten inkrementellen Parsing aus Abbildung 5.7 nur erweitert. Diese Er-

Nimm das erste Wort als Eingabe E_0
Erzeuge Analyse für die erste Eingabe
    Erzeuge Suchraum P_0 für E_0
    Erzeuge initiale Analyse $A_{0,init}$ für P_0
    Suche in P_0 ausgehend von $A_{0,init}$ mit dem Ergebnis A_0
WHILE (Präfix kleiner als Satz)
    Erweitere das Präfix E_{n-1} um ein Wort zu neuer Eingabe E_n
    Erweitere P_{n-1} **zu** P_n
    Erzeuge $A_{n,init}$ für P_n **aus der Analyse** A_{n-1} **des letzten Schrittes**
    Suche in P_n ausgehend von $A_{n,init}$
ENDWHILE

Abbildung 5.7.: Inkrementeller Algorithmus 4, integriertes inkrementelles Parsing

weiterung umfasst alle Lexemknoten für das neue Wort sowie alle Kanten mit einem dieser Lexemknoten als Regent oder Dependent.

Es müssen also folgende Elemente hinzugefügt werden

- die Lexemknoten für das neue Wort
- die neuen Kanten ausgehend vom neuen Wort
- die neuen Kanten zum neuen Wort hinführend

Dies geschieht im Wesentlichen nach den selben Mechanismen wie beim nichtinkrementellen Aufbau des Suchraumes. Auch dort werden die Kanten nacheinander erzeugt.

5.3.3. Revision der Bewertung alter Elemente

Neben der Erweiterung um neue Elemente sind auch Revisionen an den aus dem vorangegangenen Schritt übernommenen Elementen notwendig.

Lexemknoten beinhalten das Wort im Satz, auf das sie sich beziehen sowie alle grammatikalischen Eigenschaften der Lesart, die sie darstellen. Diese hängen nur vom Wort im Satz und dem verwendeten Lexikon ab. Unter der Annahme, dass Wörter immer nur hinzugefügt aber nie verändert oder entfernt werden und dass das Lexikon während der Bearbeitung eines Satzes nicht ausgetauscht wird, bleibt ein einmal eingeführter Lexemknoten in allen nachfolgenden, durch inkrementelle Erweiterungen entstandenen Suchräumen gültig.

Bei den Kanten gilt ähnliches. Eine Kante geht aus von einer Menge von Lexemen hin zu einer anderen Menge von Lexemen, bezieht sich auf einen Ebene und hat eine Beschriftung. Verfügbare Ebenen und Beschriftungen sind abhängig von der Grammatik, die nicht zur Laufzeit ausgetauscht wird. Alle Lexemknoten in den Lexemgruppen für Dependenten und Regenten gehören jeweils alle zum selben Wort, sind also unabhängig davon, ob an anderer Stelle ein weiteres Wort hinzukommt.

Ein weiteres Attribut einer Kante ist ihre initiale Bewertung. Wie oben beschrieben, wird jede Kante bei ihrer Konstruktion durch die Grammatik bewertet, indem alle auf einzelnen Kanten auswertbaren Constraints auf ihr ausgewertet und die Straffaktoren der verletzten Constraints aufmultipliziert werden. Diese Auswertung ist für einen Großteil des Zeitbedarfs beim Aufbau des Suchraumes verantwortlich, eine Neuauswertung dieser Constraints sollte daher vermieden werden. In die initiale Bewertung fließen alle Constraints ein, die ohne eine umgebende Analyse ausgewertet werden können. Dies sind die unären, nicht kontextsensitiven Constraints, also diejenigen, die genau eine Kante betrachten. Die aus der Auswertung eines solchen Constraints auf einer gegeben Kante resultierende Bewertung verändert sich also prinzipiell nicht, wenn der Suchraum an anderer Stelle durch eine andere Kante erweitert wird.

Als nicht kontextsensitiv gelten jedoch auch alle Constraints, die sich nur auf den Satz selbst, nicht jedoch auf eine Analyse beziehen. Der Satz als Ganzes ist aber beim inkrementellen Vorgehen eben nicht an jedem Punkt vollständig bekannt. Diese satzbezogenen Constraints können nur auf dem bereits bekannten Satzpräfix ausgewertet werden. Da sie als nicht kontextsensitiv in die initiale Kantenbewertung eingehen, kann sich diese Bewertung zwischen den inkrementellen Schritten verändern und darf daher nicht uneingeschränkt übernommen werden.

Es gibt zwei Möglichkeiten zum Umgang mit diesen Constraints. Man kann sie entweder vor jedem inkrementellen Schritt neu berechnen oder man kann sie gänzlich aus der initialen Kantenbewertung entfernen, indem man sie ebenfalls als kontextsensitiv betrachtet. Die erste Methode verlangt, dass

1. der Einfluss der satzbezogenen Constraints auf die Kantenbewertung getrennt von der Bewertung durch andere Constraints gespeichert wird, um sie getrennt neu berechnen zu können.

2. auf allen bestehenden Kanten die betroffenen Constraints vor jedem inkrementellen Schritt neu ausgewertet werden müssen.

3. alle Kanten, die durch ein satzbezogenes Constraint mit 0 bewertet wurden, nicht endgültig gelöscht werden können, da sich diese Bewertung prinzipiell ändern kann. Sie können jedoch aus dem Suchraum des aktuellen inkrementellen Schrittes entfernt werden.

Der Overhead durch die getrennte Speicherung ist vergleichsweise gering. Auch die Zeit, die für eine Neuauswertung auf den aktiven Kanten benötigt wird, ist vertretbar. Zum Vergleich: in jedem Reparaturschritt werden auf allen Kanten einer Domäne alle Constraints, insbesondere auch zusätzlich die binären und kontextsensitiven ausgewertet. Problematisch ist der dritte Punkt. Die Neubewertung der verworfenen Kanten ist aufwändiger, da die Menge der Verworfenen Kanten deutlich größer ist als die Menge der aktiven Kanten (vergleiche Abbildung 3.2)

Die zweite Methode, also satzbezogene Constraints als kontextsensitiv zu markieren und damit aus der initialen Bewertung zu entfernen, hat zur Folge, dass über diese Constraints keine Kanten mehr von vornherein aus dem Suchraum ausgeschlossen werden können. Das Problem der Reevaluation gelöschter Kanten bei Methode Eins

wird bei der zweiten Methode also durch das größere Problem ersetzt, dass alle diese Kanten im Suchraum verbleiben.

Unabhängig davon, wie viele Kanten betroffen sind, ist also die Methode der Reevaluation dem Markieren von Kanten als kontextsensitiv vorzuziehen. Um den Aufwand der Reevaluation abschätzen zu können, müssen wir uns zunächst anschauen, welche Constraints konkret betroffen sind.

Satzbezogene Constraints

Constraints können im Wesentlichen auf drei Arten auf den Satz selber zugreifen, über:

- Externe Prädiktoren
- Abstandsmaße
- Vorkommen von bestimmten Wörtern oder Satzzeichen

Externe Prädiktoren arbeiten auf dem Eingabesatz, nicht auf dem Suchraum des CDG-Parsers. Über den Aufruf in einem kantenbezogenen Constraint wird das Ergebnis des Prädiktors in eine Kantenbewertung übersetzt. Ein Beispiel ist der POS-Tagger, dessen Wahrscheinlichkeitsbewertung für die Wortarten der durch eine Kante gebundenen Lexemknoten in eine Bewertung eben dieser Kante übersetzt wird.

Greift ein Prädiktor bei der Bearbeitung eines Wortes auf die weiter rechts liegenden Wörter zu, kann sich sein Ergebnis durch die inkrementelle Erweiterung des Satzpräfixes ändern. Ein einfacher n-Gramm-Tagger, der nur links vom betrachteten Wort liegende Wörter mit einbezieht, ändert seine Bewertung nicht. Für andere Tagger gilt dies nicht. Der hier verwendete Tagger verwendet z.B. ein Regelsystem, das über eine globale Betrachtung die lokalen Entscheidungen des statistischen Modells revidieren kann.

Ein weiterer Fall, in dem Constraints auf den Satz selbst zugreifen, ist über ein Distanz-Prädikat. Dieses Prädikat kann entweder in der Regel selbst oder auch im Term zur Berechnung des Bestrafungswertes vorkommen. Im letzteren Fall würde es dann in die numerische Bewertung eingehen. Ein Beispiel hierfür ist ein Constraint, das die Anbindung weit entfernter Komplemente bestraft, und zwar um so härter, je weiter diese entfernt sind. Da neue Wörter nur am Ende und nicht in der Mitte des Satzes angefügt werden, ändert sich eine solcherart abgefragte Distanz für bestehende Kanten nicht.

Ein dritter Fall ist die Überprüfung der Existenz eines Wortes oder insbesondere eines Satzzeichens zwischen zwei Positionen im Satz. Diese Abfrage ist unproblematisch, solange diese beiden Positionen innerhalb des Satzes liegen. Die Begrenzungen eines solchen Intervalls können sich jedoch auch auf die Grenzen des Satzes beziehen, etwa wenn ein Constraint die Existenz eines bestimmten Satzzeichens am Ende des Satzes abfragt. Ein solches Constraint wird dann auf dem vollständigen Satz evtl. mit einem anderen Ergebnis ausgewertet als auf dem Satzpräfix. In der vorliegenden Grammatik kommt dieser Fall jedoch in keinem unären, nicht kontextsensitiven Constraint vor.

Spezialfall NONSPEC

Der NONSPEC-Knoten steht für alle noch unspezifizierten, in der Zukunft liegenden Wörter des Satzes. Alle diese Wörter haben gemeinsam, dass sie rechts von den spezifizierten stehen. Bei einem bekannten Satzpräfix der Länge n steht NONSPEC also für alle Wörter der Positionen $n + 1$ oder höher. Wird die Position eines Regenten und damit bei bestimmten Kanten die Position von NONSPEC durch ein Constraint abgefragt, wird ein konkreter Wert erwartet. Dieser Wert kann für zwei Prädikate verwendet werden, für Distanzabfragen und für Existenzabfragen über einem Intervall. Im zweiten Fall ist der konkrete Wert egal, weil alle Werte höher als n gleichbedeutend mit dem Ende des Satzes als Intervallgrenze sind.

Distanzabfragen werden für gewöhnlich verwendet, um Kanten zwischen zwei entfernten Wörtern zu bestrafen. Als Position für NONSPEC kann also die optimistische Abschätzung $n+1$ angegeben werden. Alle Constraints, für die dieser Wert eine pessimistische Abschätzung darstellt, etwa weil ein Mindestabstand zwischen Wörtern erwartet wird, müssen den NONSPEC-Fall explizit behandeln. Wird das Satzpräfix um ein neues Wort erweitert, verschiebt sich also die NONSPEC Position nach hinten, die Länge der nach NONSPEC gehenden Kanten wächst. Alle unären, nicht kontextsensitiven Constraints, die eine Distanzabfrage beinhalten, müssen also den NONSPEC-Fall explizit behandeln oder für NONSPEC-Kanten, und nur für diese, nach jedem Schritt neu ausgewertet werden.

Bei Kanten mit NONSPEC als Regenten ergibt sich außerdem ein Problem mit intervallbezogenen Existenzabfragen, ähnlich wie das oben beschriebene mit dem Satzende. Ob zwischen einem Wort im Satz und einem noch in der Zukunft liegenden Wort oder dem Satzende ein bestimmtes Wort oder Satzzeichen liegt, ist noch nicht bekannt. Während das Satzende als Intervallgrenze in der Grammatik nur in bestimmten Constraints vorkommt, kann in jedem Constraint mit einem Regenten als Intervallgrenze NONSPEC dieser Regent sein.

Neben der Reevaluation dieser Constraints bietet sich die Möglichkeit an, dieses Problem in den Constraints selbst abzufangen. Da unbekannt ist, ob das gesuchte Token im zukünftigen Teil des Satzes enthalten ist, kann der NONSPEC-Fall durch eine optimistische Annahme abgefangen werden. Jedes positiv verwendete Existenzabfrage-Prädikat in einem Constraint wird mit einem NONSPEC-Guard versehen: ist der Knoten dessen Position die Intervallgrenze darstellt nicht spezifiziert, wird nicht ausgeschlossen, dass das Gesuchte in der Zukunft vorkommt.

Negiert verwendete Existenzabfragen, etwa durch die Verwendung im Regelkopf vor der Implikation, können nicht auf diese Weise abgefangen werden. Für den unspezifizierten Teil des Intervalls gilt die optimistische Annahme, dass das Gesuchte nicht auftritt. Tritt es nicht im spezifizierten Teil des Satzes auf, kann es dennoch in einer späteren Erweiterung vorhanden sein. Das Constraint würde die NONSPEC-Kante dann schlechter bewerten und muss in jedem Schritt neu ausgewertet werden.

Relevanz des Tagger-Constraints

Der POS-Tagger ist für einen Großteil der durch die initiale Bewertung verworfenen Kanten verantwortlich. Das Tagger-Constraint als kontextsensitiv zu behandeln, also erst in den konkreten Analysen auszuwerten statt bereits bei der Kantenerstellung, würde den Aufwand einer Reevaluation einsparen, jedoch zu deutlich mehr aktiven Kanten führen. Die Anzahl der Kanten im Suchraum würden um einen Faktor 100 ansteigen (vergleiche Abbildung 3.2). Die Erweiterung des Prozesses um eine Reevaluation bestimmter Constraints ist also für den POS-Tagger notwendig und kann auch auf die deutlich weniger relevanten anderen betroffenen Constraints ausgeweitet werden.

Implementation der Reevaluation

Alle bei der initialen Kantenbewertung auszuwertenden Constraints werden in zwei Klassen eingeteilt, fluktuierenden und nicht fluktuierenden Constraints. Fluktuation bezieht sich hierbei darauf, ob die Auswertung des betroffenen Constraints in verschiedenen inkrementellen Schritten zu unterschiedlichen Ergebnissen kommen kann, Als fluktuierend gelten also alle satzbezogenen Constraints, die nicht aus einem der oben beschriebenen Gründe ausgeschlossen werden konnten. Diese Aufteilung kann von der Art der Kante abhängen. So sind die oben angesprochenen Distanz-Constraints nur für NONSPEC-Kanten fluktuierend.

Für jede Kante werden die Bewertungen durch die fluktuierenden (b_f) und die durch die nicht fluktuierenden Constraints (b_{nf}) getrennt gespeichert. Nach jedem inkrementellen Schritt wird b_f neu berechnet, während die nicht fluktuierenden Constraints nicht neu ausgewertet werden müssen.

Wir müssen drei Arten von Kanten unterscheiden.

Dauerhaft entfernte Kanten Kanten mit $b_{nf} = 0$ können permanent aus dem Suchraum entfernt und müssen nicht reevaluiert werden.

Temporär entfernte Kanten Kanten mit $b_{nf} > 0$ und $b_f = 0$ können temporär aus dem Suchraum entfernt werden, müssen jedoch vor jedem inkrementellen Schritt reevaluiert werden.

Aktive Kanten Kanten mit $b_{nf} > 0$ und $b_f > 0$ sind die aktiven Kanten im Suchraum. Sie müssen ebenfalls reevaluiert werden.

Sollte die Bewertung einer temporär entfernten Kante sich bei der Reevaluation erhöhen, wird sie zu einer aktiven Kante und in die entsprechenden Domäne des Suchraums hinzugefügt. Die aktiven Kanten können in jedem Schritt eine andere Bewertung haben, sollte diese auf 0 fallen, werden sie temporär aus dem Suchraum entfernt. Da dies nur einen sehr kleinen Teil der Kanten betrifft und eine zusätzliche aber richtig bewertete Kante im Gegensatz zu einer fehlenden Kante keine Auswirkung auf das Parsing-Ergebnis hat, kann auf dieses Entfernen und den damit verbundenen Overhead verzichtet werden.

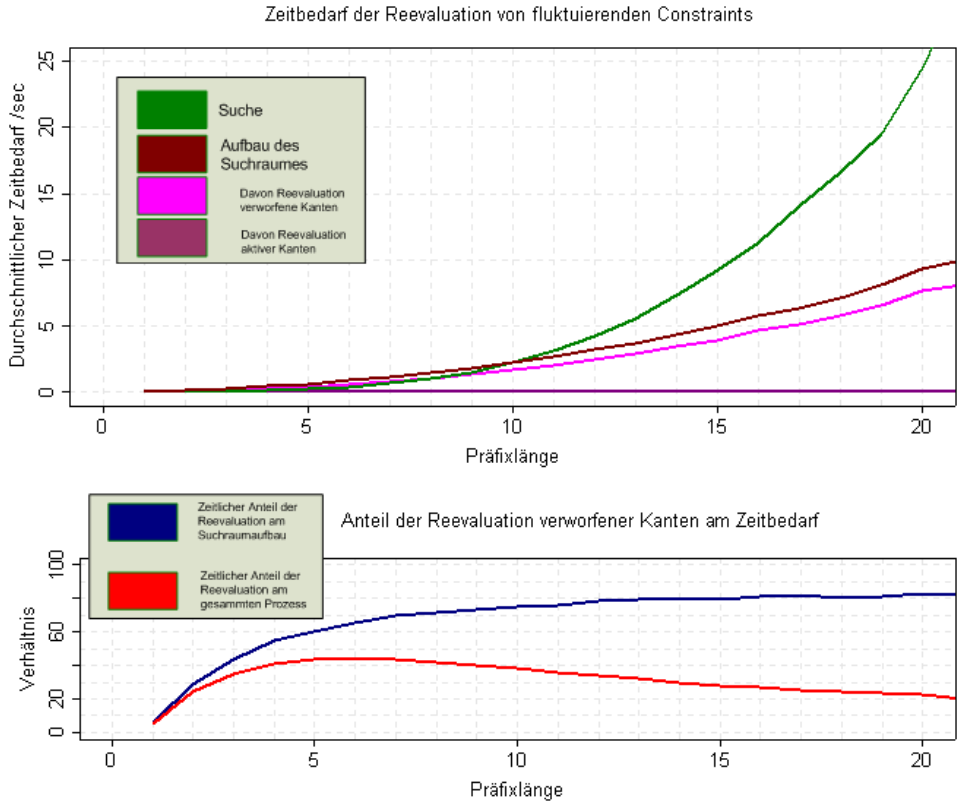

Abbildung 5.8.: Zeitbedarf der Reevaluation nach Satzlänge

5.3.4. Auswertung

Abbildung 5.8 zeigt den Zeitbedarf der durch die Reevaluation entsteht. Wir sehen, dass die Neubewertung aller aktiven Kanten wenig Zeit in Anspruch nimmt. Es ist ein kleiner Preis dafür, dass Kanten nun eine aktuelle Bewertung haben. Dies ist, wie oben ausgeführt, vor allem notwendig, um Differenzen zwischen einer nicht mehr aktuellen initialen Bewertung und der Bewertung in einer konkreten Analyse zu vermeiden, was zu fälschlicher weise gelöschten Kanten führen kann. Ein weiterer Einfluss der Bewertung ist die Reihenfolge bei der Auswahl von alternativen bei der Konfliktreparatur.

Die Reaktivierung vorher ausgeschlossener Kanten ist hingegen weniger positiv zu sehen. Die Anzahl der verworfenen Kanten aus allen zurückliegenden Schritten wächst quadratisch, während zwischen der Anzahl der Kanten eingehend und ausgehend vom jeweils aktuell neuen Knoten und der Satzlänge ein linearer Zusammenhang besteht. Die Reevaluation verworfener Kanten ist für den Großteil des Zeitbedarfs bei der Aktualisierung des Suchraumes verantwortlich und nimmt für kurze Satzpräfixe mit we-

niger als 9 Wörtern (vgl. Abb. 5.8) sogar mehr Zeit als die Suche selbst in Anspruch.

Diese Kosten im Zeitbedarf müssen mit dem Nutzen für die Ergebnisqualität verglichen werden. Ausgewertet auf 1000 Sätzen des Negra-Korpus ergibt sich folgendes Bild. In über 90% der Sätze wurde durch die Neuauswertung der verworfenen Kanten der Suchraum durch eine bereits verworfene Kante erweitert. Jedoch nur in 0.3% der Sätze wird eine auf diese Weise aktivierte Kante in einem Zwischenergebnis verwendet. Zwar werden auf diese Weise Kanten im größeren Umfang wieder verfügbar gemacht, für eine Lösung gebraucht werden sie jedoch selten.

Tabelle 5.2 zeigt den direkten Vergleich zwischen den Ergebnissen jeweils mit und ohne Neuauswertung verworfener Kanten. Auch hier zeigt sich der große Einfluss auf den Zeitbedarf gegenüber dem kleinen Einfluss auf die Ergebnisqualität.

gleiches Endergebnis	90.7%
mit Reaktivierung besser bewertet	4.9%
ohne Reaktivierung besser bewertet	4.4%
Zeitbedarf bei gleichem Ergebnis, ohne gegenüber mit Reaktivierung	58.8%

Tabelle 5.2.: Vergleich mit und ohne Reaktivierung vormals gelöschter Kanten

In Anbetracht der Diskrepanz zwischen Kosten und Nutzen, kann für praktischen Belange in der untersuchten Tagger/Grammatik-Konstellation von einer Reaktivierung abgesehen werden. Für andere Tagger kann dies anders aussehen, bei einem auf die Bedürfnisse der inkrementellen Verarbeitung ausgerichteten Tagger würde das Problem idealer Weise gar nicht mehr auftreten. In allen weiteren Testläufen ist die Reevaluation verworfener Kanten deaktiviert, nicht jedoch die Reevaluation bereits aktiver Kanten.

5.3.5. Informationen zur Steuerung der Suche

Neben den Elementen des Suchraums und ihrer initialen Bewertung wird die Suche auch durch eine Reihe weiterer Informationen gesteuert. Dies sind

die Liste unbehebbarer Konflikte (unremovables) Diese Liste enthält alle Konflikte, bei denen ein Reparaturversuch erfolglos abgebrochen wurde. Diese Konflikte gelten als unbehebbar und können keine weiteren Reparaturversuche initiieren.

die oberen Schranken (Limits) der Lexeme und Kanten Sie geben an, wie gut eine Analyse, an der das jeweilige Element beteiligt ist, maximal bewertet sein kann. Sie werden über die Auswertung der unären Constraints auf den Kanten bestimmt und werden weiter propagiert, indem für ein Lexem bzw. eine Kante ermittelt wird, welche Elemente des jeweils anderen Typs sie binden. Diese Schranken sind korrekt, jedoch meist viel zu hoch. Sollte ein Lexem oder eine Kante aber unter der Bewertung der aktuellen Analyse liegen, wird es als gelöscht markiert

In diesem Abschnitt wird diskutiert, inwieweit diese Informationen nach einer Erweiterung der Eingabe weiterhin gültig sind.

Unbehebbare Konflikte

Wenn ein Konflikt als unbehebbar markiert wurde, konnte er für den gegebenen Satz oder das Satzpräfix nicht behoben werden. Im um ein Wort erweiterten Präfix könnte er behebbar sein. Ein Beispiel für ein solches Verhalten ist die Verletzung eines Constraints, das die Anwesenheit eines Präpositionalnomens für eine Präposition verlangt. Sie tritt im inkrementellen Umfeld häufig auf, da die Präposition vor dem ihr untergeordneten Nomen auftaucht, eine einzelne Präposition jedoch keine gültige Präpositionalphrase ist. Dieser Konflikt lässt sich meist schon im nächsten Schritt beheben, war jedoch im vorhergehenden Schritt nicht behebbar und wurde daher auch als unbehebbar markiert.

Dieses Beispiel zeigt, dass die Liste nicht uneingeschränkt übernommen werden kann, da es Konflikte gibt, die nur kurzfristig nicht behebbar sind. Konzeptuell muss also zwischen flüchtigen und persistenten Konflikten unterschieden werden. Solche, die in einem Zwischenergebnis vorkommen, jedoch in der Analyse des vollständigen Satzes behoben sind, gelten als **flüchtig**. Als **persistent** gelten dann solche Konflikte, die, wenn sie in einem Zwischenergebnis enthalten sind, auch in allen weiteren Analysen vorkommen. Ließen sich die Konflikte klar in diese zwei Klassen einteilen bzw. könnte vorausgesagt werden, ob ein Konflikt nach einer Erweiterung der Eingabe wieder behebbar ist, dann könnte die Liste der unbehebbaren Konflikte zu einem Teil verworfen, zum anderen Teil übernommen werden.

Die falsche Klassifikation eines Konfliktes hätte unterschiedliche Auswirkungen. Ein persistenter, jedoch fälschlich als flüchtig markierte Konflikt würde in jedem inkrementellen Schritt einen erfolglosen Reparaturversuch initiieren. Diese können potentiell viel Zeit in Anspruch nehmen, haben jedoch keine Auswirkung auf die Ergebnisqualität, da alle Änderungen eines erfolglosen Reparaturversuches verworfen werden. Andersherum kann ein flüchtiger Konflikt, der keinen Reparaturversuch initiiert, nachdem er durch inkrementelle Erweiterungen behebbar geworden ist, viel Schaden verursachen. Zwar kann er immer noch durch einen Seiteneffekt einer anderen Reparatur behoben werden. Es fehlt der Suche dann jedoch die wichtige, mit jedem Konflikt verbundene Information, welche Kanten an ihm beteiligt sind und ersetzt werden sollten.

Es ist also notwendig, die Klasse der persistenten Konflikte so genau wie möglich zu erfassen. Ein erster Ansatzpunkt zur Klassifikation ist das beim Konflikt verletzte Constraint. Ein Kandidat für ein Constraint, das persistente Konflikte verursacht, ist zum Beispiel ein Constraint, das fehlende Determiner bestraft ("Determiner fehlt"). Im Gegensatz zum Präpositionalnomen steht der Artikel immer links von seinem Regenten, tritt also in der Links-Rechts-Verarbeitung vor diesem auf.

Constraint	Anteil des persistenten Auftretens
Vergleichswort fehlt	0.00%
PN fehlt	0.00%
irreführendes "auch noch"	0.00%
OBJA-Kasus	0.00%
Antezedens fehlt	0.00%
doppeltes Element in einer Sub-Clause	0.00%

Beiordnung fehlt	0.00%
OBJA2-Kasus	0.00%
substantiviertes Adjektiv als PN	1.26%
NONSPEC-Kante (SYN)	1.54%
substantiviertes Adjektiv	3.39%
Modalverb-Subjekt fehlt	3.57%
Transitivität	4.70%
Sprachebene überschritten	5.63%
PN-Kasus	5.88%
tagger	6.12%
Subjekt fehlt	7.23%
sein-Subjekt fehlt	7.69%
Substantivierung vor Nomen	9.38%
Subjekt-Numerus	10.00%
Konjunktion für Verb fehlt	12.70%
substantivierte Zahl	13.64%
unpersönliches Passiv	14.29%
Nominalgruppe	15.38%
Zahl vor Nomen	18.18%
Nebenordnung-Kategorie	19.05%
ADV-Unterordnung	20.00%
Bitransitivität	21.05%
metagrammatischer Gebrauch	21.05%
isolierter Nebensatz	21.95%
Einschub mit Komma 2	25.00%
entfernte Koordination 5	25.00%
Subjekt-Position	28.00%
entfernte Koordination 3	30.23%
PP-attachment	32.00%
Adverb nach Nomen	34.62%
Matrix-Objekt-Vermischung	36.84%
mehrere Sätze	40.00%
Adverb-Unterordnung	41.18%
Determiner fehlt	43.38%
literarischer Nebensatz	45.45%
mod-Distanz	45.77%
Doppelpunktüberschreitung	46.15%
Fragment	46.48%
entfernte Koordination 2	47.83%
topikalisiertes Objekt	60.00%
APP ohne Komma	72.73%
REF-Distanz	76.92%
PPP als Adverb	80.00%

Tabelle 5.3.: Constraint-Persistenz

Tabelle 5.3 zeigt eine Auswertung auf 1000 Sätzen des Negra-Korpus. Dargestellt ist der Anteil der Sätze bei denen das Constraint in der finalen Analyse verletzt wurde unter allen Sätzen, in denen das Constraint in mindestens einem Zwischenergebnis verletzt wurde. Es zeigt sich, dass sich keine klar als persistent klassifizierbaren Constraints finden lassen. Kein in repräsentativer Anzahl auftretendes Constraint kommt in die Nähe einer 100%-Persistenz. Auch der eben angesprochene gute Kandidat "Determiner fehlt" erreicht nur eine Quote von unter 50%.

Ohne eine zuverlässige Detektion persistenter Konflikte muss die Tabuliste der unbehebbaren Constraints nach jedem inkrementellen Schritt vollständig geleert werden. Die Informationen können nicht übernommen werden.

Bewertungsobergrenzen

Die Bewertungsobergrenzen oder Limits sind ein Mechanismus zum Beschneiden des Suchraumes. Über sie werden Kanten und Lexeme von der Suche ausgeschlossen, die garantiert an keiner Analyse beteiligt sein können, die gleich gut oder besser bewertet wäre als die derzeit beste bekannte Analyse. Sie stellen eine obere Grenze, eine optimistische Prognose für die Bewertung einer Analyse dar. Sie können prinzipiell höher sein als die beste Analyse, die tatsächlich mit diesem Element möglich ist, sind jedoch niemals niedriger.

Im nicht inkrementellen Umfeld sind sie damit monoton fallend. Der Wert, gegen die sie verglichen werden, die Bewertung der besten bekannten Analyse, ist hingegen monoton steigend. Nur durch diese Eigenschaft können zu schlecht bewertete Elemente ohne negative Auswirkung auf die Qualität des Ergebnisses gelöscht werden.

Im inkrementellen Umfeld kann es jedoch passieren, dass nach dem Hinzufügen eines Wortes die maximal erreichbare Bewertung steigt. Da ein Element mit zu niedrigem Limit rigoros gelöscht wird, kann es passieren, dass bei direkter Übernahme der Limits ein Lexem oder eine Kante ausgeschlossen wird, obwohl sie für die neue beste Lösung benötigt wird.

Durch das Propagieren der Limits kann ein Element, das auf diese Weise gelöscht wurde, dazu führen, dass eine Kaskade von anderen Elementen ebenfalls von der Suche ausgeschlossen wird. Andersherum führt eine nach der inkrementellen Erweiterung bessere Bewertung des auslösenden Elementes dazu, dass auch alle anderen Elemente wieder in Betracht gezogen werden müssen.

Da im derzeitigen CDG-System eine Erhöhung der Limits nicht vorgesehen ist, kann es sogar passieren, dass ein Lexem oder eine Kante gelöscht wird, wenn die neue Lösung, in der das Element selbst vertreten ist, einen höhere Bewertung als das alte Limit hat. Die neue Analyse enthielte dann ein gelöschtes Element und wäre damit ungültig. Dieser direkte Fall kann verhindert werden, indem Limits nach oben korrigiert würden, wenn eine Analyse an der sie beteiligt sind besser bewertet ist als das Limit angibt. Dadurch kann zwar das Auftreten ungültiger Analysen verhindert werden, ein für eine bessere Analyse benötigtes Element kann aber immer noch fälschlicher Weise gelöscht werden. Ein Element, das erst im nächsten Reparaturschritt in eine

verbesserte Analyse aufgenommen würde, kann bei nicht mehr aktuellem, zu niedrigem Limit vorher gelöscht werden.

Aus diesen Überlegungen geht hervor, dass die Limits, wie sie bisher berechnet werden, ihren Status als obere Schranke im inkrementellen Umfeld nicht beibehalten können. Sie müssen nach einer inkrementellen Erweiterung wieder erhöht werden. Um diese Erhöhung nur partiell durchzuführen, also einen möglichst großen Teil der Information in den Limits des letzten Schrittes zu verwenden, müsste bekannt sein

- auf welche Konflikte ein Limitwert zurückzuführen ist und
- ob ein solcher Konflikt auf der erweiterten Eingabe noch auftritt oder anders bewertet ist

Ersteres lässt sich, wenn überhaupt, nur mit einem großen Verwaltungsaufwand erreichen. Für jeden Limit-Wert, also insbesondere für jede Kante im Suchraum müsste eine Abhängigkeitsstruktur aufgebaut und gepflegt werden. Letzteres würde es notwendig machen, alle in die Limits eingegangenen potentiellen Konflikte neu auszuwerten. Jede dieser beiden Maßnahmen für sich würde, falls überhaupt durchführbar, das Einsparungspotential eines kleineren Suchraums durch den großen Zeitaufwand zunichte machen. Der Versuch einer partiellen Korrektur wird darum hier nicht weiter verfolgt. Die Limits werden pauschal wieder zurückgesetzt, für Kanten auf ihre initiale Bewertung, für Lexeme auf 1.0.

Im Zusammenhang mit dem Limitmechanismus seien noch folgende Werte erwähnt:

Global Minimum Bewertung der bisher besten Lösung oder von außen vorgegebener Druck (je nachdem was höher ist)

Global Maximum Bewertung der derzeit besten denkbaren Analyse

Gelöschte Lexeme und Kanten Ein Element wird dann als gelöscht markiert, wenn sein Limit unterhalb des aktuellen globalen Minimums liegt.

Das globale Minimum ist der Wert der besten bekannten Analyse. Im nichtinkrementellen Fall wächst dieser Wert monoton steigend. Da nach einer inkrementellen Erweiterung die Bewertung der Analyse des letzten Schrittes sich evtl. nicht wieder erreichen lässt, ist auch hier eine direkte Übernahme vom vorhergehenden Schritt nicht möglich.

Umgang mit Global Maximum

Das Globale Maximum ist die derzeit höchste mögliche Bewertung, den eine potentielle Analyse haben kann. Dieser Wert spielt bei der Termination der Suche eine Rolle und ergibt sich aus den besten verbliebenen Limits in jeder Domäne. Dadurch wird er mit dem Verwerfen der Limits selbst hinfällig

Gelöschte Lexeme und Kanten

Ein Element wird dann gelöscht, wenn sein Limit unterhalb des aktuellen globalen Minimums liegt. Auch diese Informationen sind also nach einer Erweiterung prinzipiell nicht mehr aktuell, das gelöschte Element könnte im neuen Problem an der besten Lösung beteiligt sein.

5.3.6. Finaler Durchlauf

Es gibt unterschiedliche Gründe, den letzten Suchlauf für die finale Analyse des gesamten Satzes anders zu behandeln als die Schritte davor. Zum einen ist das die Behandlung des NONSPEC-Knotens. Eine Anbindung an NONSPEC steht für die Erwartung, dass der geeignete Anbindungspunkt noch in der Zukunft liegt. Ist der Satz jedoch bereits abgeschlossen, macht eine solche Warteposition keinen Sinn mehr. In der Analyse des vollständigen Satzes darf NONSPEC daher nicht mehr als Anbindungspunkt verwendet werden. Dazu müssen alle Kanten mit NONSPEC als Regenten eine Abwertung erfahren, bzw. das Constraint zur Bestrafung von NONSPEC-Kanten muss härter bestraft werden. Es ist auch denkbar, bestimmte Systemparameter im abschließenden Durchlauf anders zu setzen, um die Güte der finalen Analyse zu verbessern.

Liegt ein getakteter Eingang der Eingabewörter und eine damit verbundene Zeitbegrenzung pro Schritt vor, kann am Ende des Satzes mehr Zeit zur Verfügung stehen. Eine Zeitbegrenzung pro Schritt könnte dann im letzten Schritt verändert oder aufgehoben werden. Es gibt verschiedene Möglichkeiten, solch einen finalen Durchlauf zu realisieren

Nachgelagert Nachdem die Eingabe um das letzte Element erweitert wurde, also dem Eingabesatz entspricht, und in dessen Suchraum die Suche abgeschlossen ist, wird die Suche noch einmal mit der gleichen Eingabe, jedoch mit veränderten Parametern gestartet.

Integriert Die veränderten Parameter werden bereits im letzten inkrementellen Schritt angewendet. Hierzu muss bereits vor Verarbeitung des letzten Inkrements bekannt sein, dass es sich um das letzte handelt.

kein auch ohne finalen Durchlauf lassen sich veränderte Bedingungen am Satzende realisieren, siehe unten.

Das Ende des Satzes kann ebenfalls auf verschiedene Weisen erkannt werden. Es kann

- von außen vorgegeben werden,
- an Satzzeichen erkannt werden
- oder am Ausbleiben weiterer Wörter erkannt werden

Realisierung des finalen Durchlaufs, von Veränderung betroffene Parameter und Satzende-Erkennung spielen auf unterschiedliche Weise zusammen. Es ist zum Beispiel nicht möglich, einen inkrementellen Schritt als final zu behandeln, wenn erst am Ausbleiben weiterer Wörter nach einiger Zeit erkannt wird, dass der Satz jetzt zu Ende ist. Andererseits kann das Satzende am Satzzeichen auch von Constraints der Grammatik erfasst werden, um so z.B. NONSPEC-Kanten weiter zu bestrafen, sobald ein Punkt, Ausrufezeichen oder Fragezeichen erscheint. Eine gesonderte Behandlung außerhalb der Grammatik wäre dann nicht nötig. Wenn die Zeitbegrenzung eines Schrittes sich dynamisch aus dem Auftreten des nächsten Wortes ergibt und nach Satzende eine längere Pause auftritt, verlängert sich die Zeitbegrenzung für das letzte Wort automatisch. Ist der Eingabetakt direkt durch einen menschlichen Benutzer gegeben, ist das Zeitverhalten kein geeignetes Kriterium. Die Person kann mitten im Satz längere Pausen zum Überlegen machen oder am Ende des Satzes nur sehr kurze. Bei Spracheingabe könnte die Tonhöhe bzw. Sprachmelodie herangezogen werden, um Satzgrenzen zu erkennen.

Die geeignete Variante lässt sich also nur mit Kenntnis des konkreten Anwendungsfalles ermitteln. Die Architektur muss so flexibel sein, alle sinnvollen Varianten zuzulassen. Konkret muss es also möglich sein, eine abgeschlossene Analyse nachgelagert mit veränderten Parametern noch einmal zu verarbeiten. Für die integrierte Variante müssen Parameter zwischen den Schritten verstellbar sein bzw. ein Token als final markiert werden können und Parameter für den letzten Schritt getrennt definierbar sein.

6. Heuristiken

In diesem Kapitel werden einige Ansätze vorstellen, mit denen inhärente Eigenschaften des inkrementellen Ansatzes ausgenutzt werden können, um die Verarbeitung heuristisch zu verbessern. Die Abwägung ist dabei immer die zwischen Zeitbilanz und Ergebnisgüte. Eine Heuristik beschränkt den Suchraum bzw. bricht die Suche bei zu geringer Erfolgsaussicht ab, mit dem Ziel, dadurch Zeit zu sparen und geringe Einbußen bei der Ergebnisgüte, wenn das beste Ergebnis durch die Heuristik nicht erreichbar ist, hinzunehmen. Das Ziel ist also eine positive Zeitbilanz und eine möglichst geringe negative Auswirkung auf die Ergebnisgüte. Es sind jedoch folgende Verflechtungen zwischen Zeitbedarf und Qualität zu beachten.

- Durch Zeitbegrenzungen wird Zeitersparnis in Ergebnisgüte umgewandelt. Dauern die ersten Reparaturschritte weniger lange, lässt sich in der selben Zeit evtl. noch ein weiterer Konflikt zusätzlich beheben.
- Da die Suche sich heuristisch verhält, kann ein veränderter Suchraum zu unterschiedlichen Analysen führen, auch wenn in beiden Suchräumen die gleiche Analyse das Optimum darstellt. Im kleineren Suchraum wird die Suche eine gute Analyse eher finden als in einem großen (solange sie in beiden möglich ist). Ein kleiner Suchraum bringt also nicht nur Zeit- sondern auch Qualitätsverbesserungen.
- Erfolglose Reparaturversuche dauern in der Regel länger als erfolgreiche. Eine verminderte Ergebnisgüte macht sich auch negativ im Zeitverhalten bemerkbar, da selbst solche Konflikte, die nicht mehr behoben werden können, trotzdem erfolglose Reparaturversuche erzeugen.

6.1. Konservatives inkrementelles Parsing

In Nivres [Nivre, 2008] Definition von deterministischem inkrementellen Dependenz-Parsing kann eine Kante, einmal in die Analyse eingefügt, in nachfolgenden inkrementellen Erweiterungen nicht wieder entfernt werden. Wir hatten diese Eigenschaft als konservative Erweiterung definiert. Die Menge der Kanten wächst monoton, eine einmal getroffene Entscheidung für die Anbindung eines Wortes kann nicht zurückgenommen werden. Der im vorangegangenen Kapitel vorgestellte inkrementelle Algorithmus ist in diesem Sinne nicht konservativ inkrementell, da eine in einem inkrementellen Schritt eingeführte Kante in einem späteren wieder ersetzt werden kann.

Bei einem konservativen inkrementellen Vorgehen sind die Möglichkeiten, eine Dependenzstruktur zu verändern, gegenüber dem nicht konservativen Fall stark eingeschränkt. Dies hat den Vorteil, dass die Suche nach der besten Erweiterung ein besseres Zeitverhalten aufweist, jedoch den Nachteil, dass eine in einem früheren Schritt

eingeführte Kante in einem späteren Schritt dadurch, dass sie nicht ersetzt werden kann, eine evtl. mögliche bessere Analyse verhindern kann.

Die Ergebnisse in Nivre [2006] zeigen, dass ein deterministisches inkrementelles Dependenz-Parsing mit state-of-the-Art Genauigkeit möglich ist, allerdings nur mit einem Vorgriff um mehrere Wörter. Das in Nivre [2006] benutzte Featuremodell bezieht Part-Of-Speech-Informationen der nächsten drei Wörter mit ein. Dies entspricht einer Verzögerung der Anbindung um drei inkrementelle Erweiterungen der Eingabe, nachdem der potentielle Regent und Dependent verfügbar sind. Um ein solches konservativ inkremnetelles Vorgehen auf das CDG-System zu übertragen, muss verhindert werden, dass eine in der finalen Analyse eines inkrementellen Schrittes vorkommende Kante in einem späteren Schritt wieder entfernt werden kann. Dies gilt nicht für Kanten, die während der transformationsbasierten Suche in eine Analyse eingefügt werden, im Ergebnis der Suche jedoch nicht mehr vorkommen.

Die Abfolge "initiale Analyse aus der Analyse A_{i-1} des letzten Schrittes erstellen" gefolgt von "transformationsbasierte Suche nach der am besten bewerteten Analyse A_i" kann also als das Hinzufügen einer Kante zu der Analyse des letzten Schrittes interpretiert werden, wenn A_i alle Kanten von A_{i-1} enthält, also eine konservative Erweiterung ist.

Der spätere Austausch einer Kante kann in CDG verhindert werden, indem alle ihre Konkurrenten, also die anderen Kanten der selben Domäne aus dem Suchraum entfernt werden. Wir sprechen davon, eine Domäne und ihre Kanten *einzufrieren*. Alle nicht durch die aktuellen Kanten erlaubten Lexemknoten können dann ebenfalls aus dem Suchraum entfernt werden. Das Einfrieren betrifft nicht nur bereits im Suchraum vorhandene Kanten, sondern auch Kanten, die erst in späteren inkrementellen Erweiterungen in die eingefrorenen Domänen eingefügt würden. Diese Kanten müssen dann gar nicht erst aufgebaut werden. Das Einfrieren einer Domäne spart also nicht nur durch einen kleineren Suchraum Zeit bei der Suche, sondern auch Zeit beim Aufbau neuer Kanten.

Der Anbindungspunkt für das aktuell hinzugekommene Wort kann beim inkrementellen Parsing noch in der Zukunft liegen, eine Kante ausgehend vom aktuellen Wort kann noch nicht erstellt werden. Beim Shift-Reduce-Parsing kann die Anbindung aufgeschoben werden, indem das betroffene Wort per *Shift* auf den Stack verschoben wird, ohne eine Kante zu erzeugen. Später können dann andere Wörter per *Shift* auf den Stack gelangen, bevor eine Kante für das Wort erzeugt wird. Im Terminationszustand des Parsers muss eine solche Kante jedoch vorhanden sein. Ein solches "Parken" auf dem Stack, die verschobene Entscheidung zu Anbindung entspricht in CDG einer Anbindung an NONSPEC, die ebenfalls nur temporär ist und in der Analyse des gesamten Satzes nicht mehr vorkommen darf. Eine Domäne, in der die temporäre NONSPEC-Anbindung ausgewählt ist, darf also noch nicht eingefroren werden.

Das Einfrieren aller spezifizierten Kanten stellt eine radikale Beschneidung des Suchraumes dar, die aus der Zeitbedarfs-Sicht extrem attraktiv ist. Eine zu früh getroffene Entscheidung, also eine zu früh spezifizierte Kante verhindert jedoch möglicherweise später eine bessere Analyse. Werden jedoch zu viele Wörter zunächst NONSPEC untergeordnet und beginnt der Parser erst mit der Auswahl spezifizierter Kanten, nachdem alle Wörter des Satzes bekannt sind, ist der Zeitvorteil eines inkrementellen Vorgehens

verloren. Eine genaue Bestimmung des Zeitpunktes, an dem eine Kante ohne Gefahr ausgewählt werden kann, ist sicherlich nicht möglich.

Ein Blick auf die Syntaxanalyse beim Menschen zeigt ebenfalls eine frühe Festlegung auf eine Lesart des Satzes. Stellt sich eine solche Erwartung später als falsch heraus, findet eine Reanalyse statt. Diese ist messbar in Lesezeitexperimenten [Staub and Rayner, 2007]. Im Extremfall, wie bei Gardenpathsätzen, wird der Satz zunächst gar nicht verstanden und als ungrammatisch wahrgenommen. Eine kleine Menge von Fehlentscheidungen kann also toleriert werden, wenn es zusätzlich einen Reanalyseprozess gibt, der es unter bestimmten Umständen erlaubt, ausgewählte Kanten doch zu ersetzen. In CDG müssten also vorher aus dem Suchraum entfernte Kanten wieder in diesen eingefügt werden. Ein kleiner Zeitgewinn bei den meisten Sätzen wird dann durch eine stark erhöhte Analysezeit bei wenigen anderen erkauft.

Eine naive Implementierung sieht also so aus, dass nach jedem inkrementellen Schritt Domänen identifiziert werden, in denen eine spezifizierte Anbindung ausgewählt wurde und alle nicht ausgewählten Kanten und Lexeme gelöscht werden.

6.1.1. Randphänomene

Ob eine Anbindung verzögert wird, also eine Kante nach NONSPEC ausgewählt wird oder nicht, hängt davon ab, wie diese im Vergleich zu den alternativen Kanten des selben Wortes bewertet ist. Diese Bewertung lässt sich über den Strafwert des NONSPEC-Constraints einstellen, das NONSPEC-Kanten pauschal bestraft. Wenn dieser Strafwert niedrig genug eingestellt ist, wird die NONSPEC-Anbindung solange beibehalten, bis ein geeigneter Regent auftaucht, eine Kante zu diesem also nur Konflikte verursacht, die weniger hart bestraft sind, als die NONSPEC-Kante. Unter der Annahme, das es eine korrekte Anbindung für ein Wort gibt, muss die NONSPEC-Bestrafung so eingestellt sein, dass a) die richtige Anbindung möglichst oft die erste ist, die weniger bestraft wird als die NONSPEC-Kante und b) die richtige Anbindung möglichst selten schwerer bestraft ist als NONSPEC.

Für jeden Satz, bei dem a) verletzt ist, wird die beste Analyse nicht oder erst nach einer zeitaufwändigen Reanalyse gefunden. Für jeden Satz, bei dem b) verletzt ist, wird die Kante erst ausgewählt, sobald die Bestrafung der NONSPEC-Kante am Ende des Satzes angepasst wird, es geht also ebenfalls Zeit verloren.

Das folgende Beispiel zeigt, dass für viele Sätze a) oder b) auftreten muss, da die falsche Anbindung lokal (bei Kenntnis nur des Präfixes) besser bewertet ist als die NONSPEC-Kante. Das Beispiel ist der Aufbau einer Nominalphrase wie in dem im Satz "Das Mädchen streichelt den kleinen Hund.", "den" und "kleinen" sind dabei in der Struktur des vollständigen Satzes unter "Hund" angebunden. Bevor "Hund" bekannt wird, müssten sie also nach den obigen Überlegungen unter NONSPEC angeordnet werden. Statt dessen wird jedoch jedes der beiden Präfixe der Nominalphrase als vollständige Nominalphrase unter dem Verb angeordnet, "den" als Pronomen und "den kleinen" als substantiviertes Adjektiv. Zwar ist der zweite Fall durch ein Constraint der Grammatik mit einer Strafe für diese Substantivierung versehen, die ggf. höher ist als diejenige für die NONSPEC-Anbindung, jedoch verlangt die Valenz des Verbs ein Akkusativobjekt. Somit kommt für die NONSPEC-Anbindung noch eine

Strafe für die verletzte Transitivität hinzu, die substantivierte Variante wird bevorzugt. Der erste Fall, Artikel als Pronomen, wird gar nicht bestraft, erst die später auftauchenden Wörter der Nominalphrase erzwingen die Artikel-Lesart.

Solche nur lokal für ein Präfix besseren Anbindungen treten vor allem am rechten Rand eines Präfixes, bei sich im Aufbau befindlichen Phrasen, auf. Randprobleme dieser Art sind in Anbetracht der Ergebnisse in Nivre [2006] und der daraus resultierenden Notwendigkeit eines Vorgriffes auf die nächsten Wörter zu erwarten. Es gibt verschiedene Möglichkeiten, mit diesem Problem umzugehen.

1. Veränderung der Grammatik, um bestimmte Konstruktionen wie Substantivierungen am rechten Rand eines Satzes (bzw. Satzpräfix) zu verhindern.

2. Erweiterung des NONSPEC-Mechanismus auf Dependenten, um Valenzsättigung durch NONSPEC zu erlauben.

3. Verhindern des Einfrierens bestimmter Kanten am rechten Rand der aktuellen Eingabe

Möglichkeit 1 hat den Nachteil, dass Strukturen wie Substantivierungen ja durchaus am rechten Rand des Satzes vorkommen können. In einem abgeschlossenen Satz müssten sie also erlaubt sein. Die Constraints müssten also entweder selbst über das Vorhandensein von Satzzeichen auf das Ende des Satzes schließen, oder der inkrementelle Prozess muss die entsprechenden Constraints am Ende des Satzes gezielt deaktivieren.

Möglichkeit 2 stellt zum einen einen großen Eingriff in das CDG-System dar, da von NONSPEC als Dependent ausgehende Kanten keiner Domäne zugehörig sind. Sie stehen nicht in Konkurrenz zu anderen Kanten. Es können beliebig viele solcher Kanten einer Analyse zugeordnet werden, da ein Wort Regent für prinzipiell beliebig viele andere Wörter sein kann. Zum anderen ist eine solche Veränderung auch nicht ausreichend. Die Pronomenlesart für den Artikel ist für das Satzpräfix, bei dem dieser Artikel ganz rechts steht, nicht bestraft. Auch wenn NONSPEC die Valenz sättigen könnte, würde das Pronomen in einer unmodifizierten Grammatik bevorzugt werden.

Möglichkeit 3 kommt ohne größere Eingriffe in das CDG-System aus und ist in ihren Auswirkungen auf den hier eingeführten, konservativ inkrementellen Modus beschränkt und soll daher hier weiter verfolgt werden. Neben nicht eingefrorenen, nichtspezifizierten Kanten und eingefrorenen spezifizierten Kanten, müssen also auch spezifizierte Kanten uneingefroren bleiben können.

Kanten am rechten Rand uneingefroren zu lassen entspricht dem bei Nivre [2006] benutzten Vorgriff um mehrere Wörter. Um den nicht einzufrierenden rechten Rand zu spezifizieren kann entweder ein durch eine feste Anzahl an Wörtern definiertes Fenster am rechten Rand oder ein strukturelles Maß gewählt werden. Da in Nominalphrasen wie die im obigen Beispiel prinzipiell beliebig viele Supplemente wie Adjektive vor dem Nomen vorkommen können, ist eine strukturelle Lösung vorzuziehen.

Eine mögliche strukturelle Definition für den rechten Rand kann wie folgt aussehen: Eine Kante mit Dependent d ist am **rechten Rand**, wenn es keinen transitiven Regenten r von d gibt, so dass r einen transitiven Dependenten d_2 hat, der rechts von d

liegt. Wörter sind am rechten Rand, wenn sie Regent oder Dependent einer Kante am rechten Rand sind. Wie sich diese Definition auswirkt, ist in Abbildung 6.1 zu sehen.

Darüber hinaus gehören Kanten, die einem Wort am rechten Rand untergeordnet sind, zum **erweiterten Rand**, wenn ihr Dependent selbst nicht Regent einer anderen Kante ist. Kanten im erweiterten Rand sind also isolierte einzelne Blätter. Diese sollen hier ebenfalls vom Einfrieren ausgenommen werden. Der Grund dafür liegt in Phrasen wie "ein kleiner grüner Kaktus". Eine Analyse ihres Präfixes "ein kleiner grüner" kann so aussehen, dass "ein" und "kleiner" jeweils dem substantivierten Adjektiv "grüner" untergeordnet sind. Die von "ein" ausgehende Kante hat dann einen rechten Nachbarn, würde nicht mehr zum rechten Rand gehören. Sie würde eingefroren und könnte nach der inkrementellen Erweiterung um "Kaktus" nicht mehr durch die jetzt bessere Anbindung an das neue Wort ersetzt werden. Daher wird ein solches einzelnes Wort zum erweiterten rechten Rand gezählt und nicht eingefroren.

6.1.2. Lexemambiguität

Kanten wie in der Nominalphrase *"das Haus"* können, auch wenn sie von Ziel und und Typ der Anbindung her stabil sind, noch bestimmte Ambiguitäten aufweisen. Und zwar sind Attribute wie Kasus und Numerus der beteiligten Lexeme evtl. noch ambig. Die Beispielphrase *"das Haus"* kann sowohl im Nominativ als auch im Akkusativ stehen, je nachdem ob sie als Subjekt oder Objekt angebunden ist. Diese Anbindung muss jedoch noch nicht festgelegt sein, wird sie verändert, müssen evtl. auch andere Lexeme für "das" und "Haus" mit dem entsprechenden Kasus ausgewählt werden. Da Kanten in CDG auch durch die Lexemknoten definiert sind, muss dann auch eine andere Kante ausgewählt werden. Lexemknoten, die nur in bestimmten Attributen vom ausgewählten abweichen und Kanten, die sich nur in diesen Lexemknoten von der ausgewählten unterscheiden, dürfen nicht aus dem Suchraum entfernt werden. Eine Auflistung der entsprechenden Attribute findet sich in Abbildung 6.2.

6.1.3. Reanalyse

Aus dem Suchraum entfernte Elemente können in einem späteren inkrementellen Schritt für eine geeignete Analyse benötigt werden. Ist gar keine oder nur eine sehr schlecht bewertete Analysen im verkleinerten Suchraum zu finden, ist es eine mögliche Strategie, den Suchraum um die entfernten Elemente wieder zu erweitern. Das Vorgehen, Interpretationen so früh wie möglich zu verwerfen und im Fehlerfall wieder in Betracht zu ziehen, lässt sich auch beim Menschen beobachten. Beim menschlichen Sprachverstehen ist im wesentlichen zu jedem Zeitpunkt nur eine Interpretation des Satzes aktiv [van Gompel and Pickering, 2007].

Eine solche Reanalyse kann in CDG eben durch ein Wiedereinführen gelöschter Kanten in den Suchraum realisiert werden. Um eine Kante wieder einführen zu können, muss die Kante im System vorgehalten werden, auch wenn sie aus dem Suchraum entfernt wurde. Hierbei gibt es nun zwei wesentliche Freiheitsgrade. Dies ist zum einen der Zeitpunkt, an dem eine Reanalyse stattfindet, und zum anderen die Auswahl der Kanten, die in einer Reanalyse reaktiviert werden sollen.

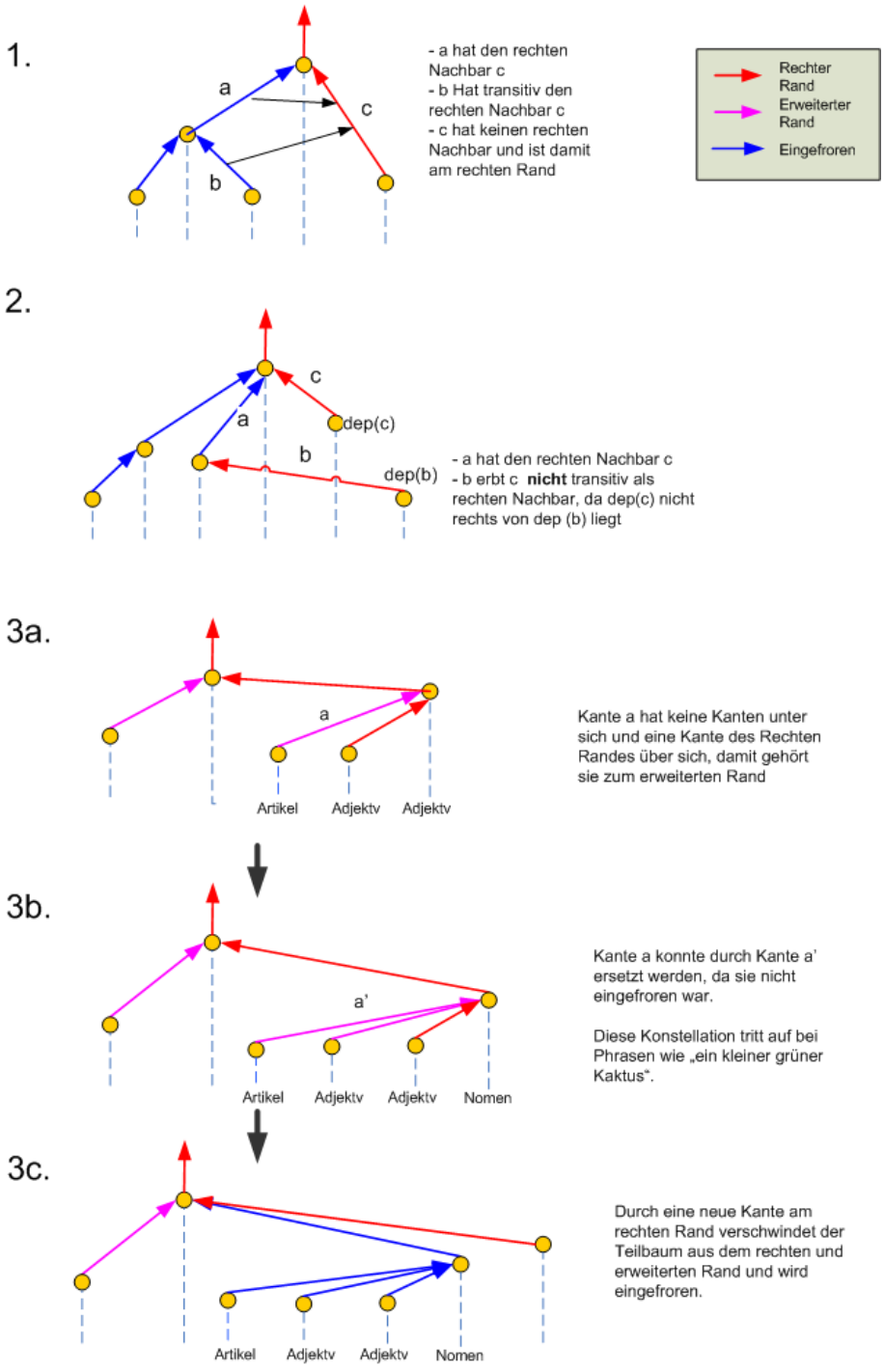

Abbildung 6.1.: Veranschaulichung eines strukturell definierten rechten Randes

- Kasus
- Numerus
- Loctype
- Flexion

Abbildung 6.2.: Lexemattribute, bei denen Lexeme eingefrorener Kanten flexibel bleiben müssen

Eine Reanalyse kommt immer dann in Betracht, wenn keine gültige Analyse ermittelt werden konnte. Ein sehr schlechte bewertetes Ergebnis kann ebenfalls auf einen zu stark eingeschränkten Suchraum hindeuten. Eine schlechte Bewertung muss immer im Kontext der bisherigen Zwischenergebnisse im inkrementellen Prozess gesehen werden. Dies lässt sich modellieren durch einen parametrisierbaren Faktor. Fällt die Bewertung der einer Analyse gegenüber ihrem Vorgänger um mehr als diesen Faktor ab, werden Kanten reaktiviert und die Suche von neuem gestartet. Es ist zu beachten, dass ein Abfall in der Bewertung auch den Grund haben kann, dass für das aktuelle Satzpräfix keine bessere Analyse existiert, unabhängig von den aus dem Suchraum entfernten Elementen.

Die zweite Fragestellung ist die, welche Kanten reaktiviert werden sollen. Alle verworfenen Kanten zu reaktivieren, ist dabei die einfachste Methode. Eine differenziertere Vorgehensweise teilt die Elemente nach der Wahrscheinlichkeit, noch zu einer Lösung betragen zu können, in verschiedene Klassen ein und reaktiviert diese Klassen nacheinander. Auf die Reaktivierung der unwahrscheinlichsten Klasse könnte ganz verzichtet werden.

Weiter oben wurden bereits einige Gründe besprochen, warum bestimmte Kanten und Lexeme in einer späteren Analyse die aktuell ausgewählte ersetzen könnten. Da von diesen Gründen jedoch nur wenige Kanten betroffen sind, und diese Kanten sehr oft benötigt werden, besteht die beste Alternative darin, sie im Suchraum zu belassen.

6.2. Einfrieren stabiler Bereiche

Über die Überlegungen im letzten Abschnitt hinaus soll in diesem Abschnitt eine weitere Heuristik zur Beschneidung des Suchraumes über inkrementelle Schritte hinweg entworfen werden.

Im inkrementellen Parsing stehen, wie im letzten Kapitel beschrieben, in jedem Schritt alle Entscheidungen der vorangegangenen Schritte wieder zur Disposition. Alle alten Kanten und Lexeme stehen in jedem Schritt zu Verfügung, auch wenn einige davon für eine Analyse des Präfixes ausgewählt, andere jedoch verworfen wurden. Damit wird der Tatsache Rechnung getragen, dass neue Informationen über die Fortsetzung des Satzes Revisionen notwendig machen können. Ein Großteil der Entscheidungen wird zwar tatsächlich nicht verändert, die Frage ist jedoch, welche Entscheidungen als

sicher gelten können und bei welchen eine Revision nicht ausgeschlossen werden kann. Wie weiter oben besprochen wurde, eignen sich die innerhalb eines inkrementellen Schrittes zur Bewertung von Kanten und Konflikten verwendeten Mechanismen, die Limits und die unremovable-Liste, nicht zur Prognose über inkrementelle Erweiterungen hinweg.

Dennoch ist es durchaus attraktiv, Kanten und Lexeme endgültig auszuschließen. Kanten werden bei jedem Reparaturversuch in ihrer Domäne in Erwägung gezogen, gelöschte Kanten sparen also bei jedem Reparaturversuch. Gelöschte Lexeme sparen vor allem Zeit beim Kantenaufbau in der Suchraumerweiterung. Lexemgruppen sind ein Faktor im Kreuzprodukt beim Aufbau der Kanten (Levels x Dependenten x Labels x Regenten). Für jede Gruppe von Lexemen, die komplett ausgeschlossen wurde, muss eine Vielzahl von Kanten nicht aufgebaut werden. Dies spart zum einen die Zeit zur Bewertung dieser Kanten, zum anderen muss nur ein kleinerer Suchraum abgearbeitet werden.

Statt direkt Elemente zu suchen, die ausgeschlossen werden sollen, können auch Elemente in der aktuellen Analyse ermittelt werden, die als stabil gelten können um dadurch indirekt alle Elemente, die mit ihnen konkurrieren, auszuschließen. Zu einer Kante konkurrieren alle Kanten der selben Domäne, die also den selben Dependenten haben. Zu einem Lexem konkurrieren alle anderen Lesarten des selben Wortes. Gesucht ist daher ein Mechanismus um Kanten und Lexeme bzw. größere, aus diesen aufgebaute Teilstrukturen ausfindig zu machen, die als stabil gelten können. Hier eine Strukturierung der Möglichkeiten, diese Suche und das darauf aufbauende Einfrieren zu gestalten:

Binäres VS graduelles Einfrieren

Beim binären bzw. harten Vorgehen werden die nicht mehr zu berücksichtigenden Elemente gelöscht und damit vollständig von der Verarbeitung ausgeschlossen. Alternativ können Strukturen auch graduell bestraft werden, z.B. indem die Bewertung von zurückliegenden, unbenutzten Kanten sukzessive vermindert wird. Beide Ansätze haben verschiedene Vor- und Nachteile.

Binär

pro Löschen spart Speicherplatz und vor allem späteren Rechenaufwand, denn gelöschte Lexeme vermindern später unnötig erzeugte Kanten

contra ein späteres Änderung der Interpretation ist nach dem Löschen nicht mehr, oder nur durch ein aufwändigeres Auftauen und Wiedereinbringen der Elemente möglich, ein rein hartes Vorgehen würde außerdem einen willkürlichen Punkt erfordern, an dem ein Element vom Status der gültigen Alternative zur vollständigen Entfernung aus dem Suchraum übergeht.

Graduell

pro eine späteres Umschlagen der Interpretation ist noch möglich, insbesondere wenn die aktuelle Struktur durch Erweiterung der Eingabe stark in ihrer Bewertung abfällt, so dass auch stark bestrafte Alternativen wieder konkurrenzfähig werden.

Die Wirkung von weicher Bestrafung kann außerdem schon einsetzen, bevor es sich ein harter Ansatz leisten kann, ein Element ganz auszuschließen

contra Keine Zeitersparnis durch Kanten, die gar nicht erst eingeführt werden müssen, der Suchraum wird nicht verkleinert sondern nur anders bewertet.

Prinzipiell lassen sich beide Ansätze kombinieren. Der binäre Ansatz hat jedoch sowohl Einsparungspotential und ist auch weniger invasiv in der Umsetzung. Es müssen nur Kanten und Lexeme als gelöscht markiert werden, statt neue Arten von Konflikten für eine zusätzliche Bewertung einzubauen.

Lexem- VS kantenbasiert

Da eine Kante ohne Lexeme bzw. ein Lexem ohne Kanten im weiteren Prozess automatisch gelöscht werden, und Bewertungen in Form von Limits zwischen Kanten und Lexemen weiter propagiert werden, reicht es aus, sich auf eine der beiden Strukturen zu beschränken. Da Constraint-Auswertung und Bewertung an den Kanten geschieht, sind sie in dieser Hinsicht naheliegender.

Orientierung am Abstand VS an der Struktur

Ein Kriterium dafür, wie stabil eine Struktur ist, kann ihr Alter sein, also wie weit die Erweiterung der Eingabe um die in der Struktur vorkommenden Elemente zurückliegt. Die Bestimmung des Alters kann über die Position im Satz oder die Position in der aktuellen Dependenzstruktur erfolgen.

Der Nachteil der Position im Satz ist, dass sich der Abstand in strukturell ähnlichen Sätzen deutlich unterscheiden kann, und unter anderem davon abhängig ist, wie viele Supplemente vorkommen (z.B. "Der Hund ..." VS "Der große braune Hund von nebenan, der mich schon letztens so blöd angekläfft hat, ...").

Berücksichtigung zwischenzeitlicher Änderungen

Neben der Position kann sich das Alter einer Teilstruktur auch nach dem Zeitpunkt der letzten Änderung bestimmen, also in welchem inkrementellen Schritt das letzte mal ein Lexem oder eine Kante der Struktur verändert wurde.

Pauschal VS differenziert (nach Attributen)

Evtl. lassen sich Kriterien ausfindig machen, nach denen Kanten bzw. Lexeme abhängig von ihren Attributen unterschiedlich behandelt werden.

Echtzeit-Fähigkeit

Alle diese Heuristiken müssen auch in Hinblick auf die wünschenswerte Erweiterung zur Echtzeitfähigkeit betrachtet werden. Bei einer Zeitbeschränkung für die inkrementellen Schritte können frühere Zwischenergebnisse noch erhebliche Fehler enthalten, also von der optimalen Analyse ihres Satzpräfix abweichen. Wenn die Zeit, um bereits

behebbare Konflikte zu beheben, erst mehrere Schritte später zur Verfügung steht, könnten Heuristiken, die bei lokal optimalen Zwischenergebnissen gut funktionieren, versagen.

6.2.1. Detektion stabiler Substrukturen

Im Nachhinein betrachtet kann eine Kante, nach nach dem Abschluss eines inkrementellen Parsings, als stabil ab einem inkrementellen Schritt gelten, wenn sie ab diesem Schritt in jedem weiteren Zwischenergebnis enthalten ist. Eine Kante als stabil zu markieren ist eine Prognose auf diese Eigenschaft. Es gibt verschiedene Kriterien , wann eine solche Prognose begründet ist.

- Auch, aber nicht nur in Hinsicht auf die eventuelle Zeitbeschränkungen kann man sich nicht darauf verlassen, dass ein Konflikt, der in einer Analyse auftritt, nicht in einem späteren inkrementellen Schritt behoben werden könnte. Von daher können Kanten, die an einem Konflikt beteiligt sind, nicht als stabil gelten.
- Eine Kante gilt als nicht stabil, wenn eine Kante die ihr untergeordnet ist, also ihren Dependenten als Regenten hat, nicht stabil ist. Ein Konflikt in einer untergeordneten Kante, der sich lokal nicht beheben lässt, kann eine auch die ihr übergeordneten Kanten betreffende Reinterpretation notwendig machen. Andersherum können lokale Strukturen auch bei einer globalen Reinterpretation stabil bleiben.
- Die jüngste, am weitesten rechts stehende Teilstruktur muss, wie im letzten Abschnitt dargelegt, als in Bearbeitung gelten, nur weiter zurückliegende Strukturen können als stabil gelten.
- Eine einzelne Kante, die keine untergeordnete Kanten hat und deren übergeordnete Kante bereits nicht als stabil gilt, gilt ebenfalls nicht als stabil. Strukturen der Größe eins haben sich als nicht stabil erwiesen.

6.2.2. Einfrieren der beteiligten Kanten und Lexeme

Ist ein stabiler Teilbaum gefunden, stellt sich die Frage, welche Kanten und Lexeme daraus resultierend verworfen werden können. Die ersten Kandidaten sind die Kanten aus den Domänen der aktuell ausgewählten Kanten sowie die Lexeme in den selben Zeitslots wie die ausgewählten Lexeme.

Dabei sind jedoch einige Einschränkungen zu machen. Für das Wort an der Spitze des Teilbaumes dürfen keine nach oben gehenden Kanten eingefroren werden. Denn ist z.B. eine Nominalphrase als stabil befunden worden, die Verbalphrase in der er teilnimmt jedoch noch nicht, so kann sich die Anbindung der Phrase ändern, z.B. als Objekt statt als Subjekt am selben Verb. Das oberste eingefrorene Element in einem Teilbaum ist also immer ein Lexemknoten, die von diesem ausgehende Kante ist nicht eingefroren. Ein eingefrorener Lexemknoten beschränkt jedoch die möglichen Alternativen zu dieser Kante auf solche Kanten, die das ausgewählte Lexem als Dependenten

erlauben. Prinzipiell wird für jede Kombination von Regenten, Dependenten (jeweils Position und Lexem) und Anbindungsart eine Kante im Suchraum erzeugt. Die Festlegung des Dependenten-Lexems schränkt dadurch die Auswahl der Kanten in einer Domäne, also Kanten ausgehend von der Dependenten-Position nur in sofern ein, dass solche Kanten aufgrund einer schlechten Bewertung bereits gelöscht sein könnten. Ist ein Wort also z.B. durch Einfrieren auf die Nomenlesart festgelegt, kann es nur noch auf für Nomen erlaubte Weise angebunden werden. In diesem Zusammenhang sei noch einmal auf die oben erwähnte, von stabilen Strukturen erwartete Eigenschaft hingewiesen, aus mehr als einem Wort zu bestehen. Bei einzelnen Blättern würden keine Kanten, sondern nur Lexeme eingefroren werden, da das Wort als Kopf der Struktur in der Anbindung flexibel bleiben muss. Das Einsparpotential ist also einerseits überproportional geringer als bei größeren Strukturen, andererseits ist für ein einzelnes Wort die Gefahr größer, trotz Erfüllens der anderen Kriterien bei Änderungen an den benachbarten Wörtern nicht stabil zu bleiben. Letzteres gilt zum Beispiel für Pronomen, die sich später als Artikel herausstellen können.

Direkt aus dem Beispiel der Änderung der Satzgliedart resultierend, müssen innerhalb einer stabilen Phrase aber weiterhin kleinere Änderungen erlaubt sein, damit sie flexibel in ihrer Anbindung nach außen bleibt. Das sind insbesondere Änderungen bezüglich des Kasus oder auch des Numerus, die sich nicht immer aus der Phrase selbst, sondern nur aus ihrem Regent ergeben. Alle Lexeme, die sich nur in solchen Attributen unterscheiden, und damit alle Kanten, die sich nur in diesen Lexemen unterscheiden, müssen weiterhin verfügbar sein. Auch diese Einschränkung ist analog zu der im letzten Abschnitt vorkommenden. Eine Auflistung der betroffenen Lexem-Attribute findet sich in Abbildung 6.2.

Neben diesen Einschränkungen bei den direkt konkurrierenden Elementen kommen jedoch auch andere Elemente für eine Entfernung aus dem Suchraum in Betracht. In Frage kommen hier alle in die stabile Struktur eingehenden Kanten, die also ein Wort der Struktur als Regent haben. Gilt eine Phrase in sich als stabil, kann weiter angenommen werden, dass sie auch nicht durch neue Anbindungen erweitert wird. Hier liegt auch eine Projektivitätsannahme zugrunde, denn stabile Unterbäume müssen nach obiger Bedingung eine Kante rechts von sich haben. In bestimmten Fällen kann diese Projektivitätsforderung verletzt werden. Kanten die von rechts kommend die neuere Kante rechts des Teilbaumes kreuzen, sind nach der Grammatik vor allem Nebensätze wie z.B. ein Relativsatz, der nachgelagert angebunden ist. Solche Kanten müssen weiterhin erlaubt sein. Solange die Grammatik jedoch harte Projektivitätsconstraints enthält, wie die vorliegende, lässt sich einschränken, welche Kanten unter diese Ausnahme fallen. Es ist auch der Fall zu berücksichtigen, dass eine ganze Nominalphrase in einem späteren Schritt unter eine benachbarte untergeordnet werden muss. Und zwar kann ein vermeintliches Dativobjekt zu einer Genitivanbindung werden, wenn eine später auftauchende Nominalphrase den Platz des Dativobjektes für sich beansprucht. Eine Veranschaulichung findet sich in Abbildung 6.3.

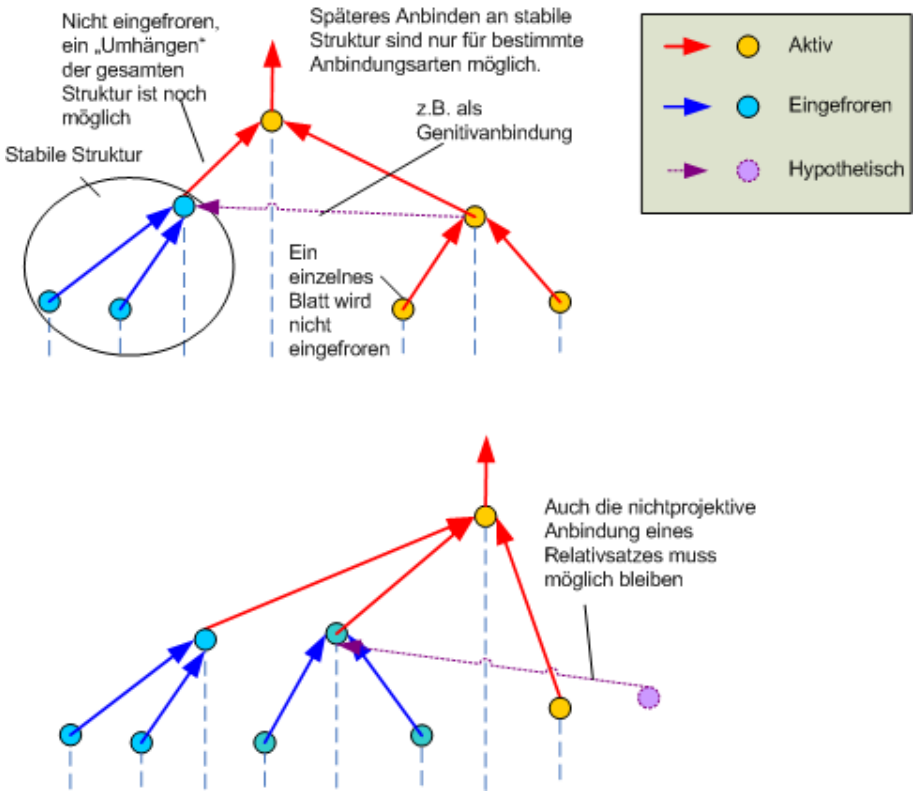

Abbildung 6.3.: Veranschaulichung stabiler Strukturen für die Einfrierheuristik

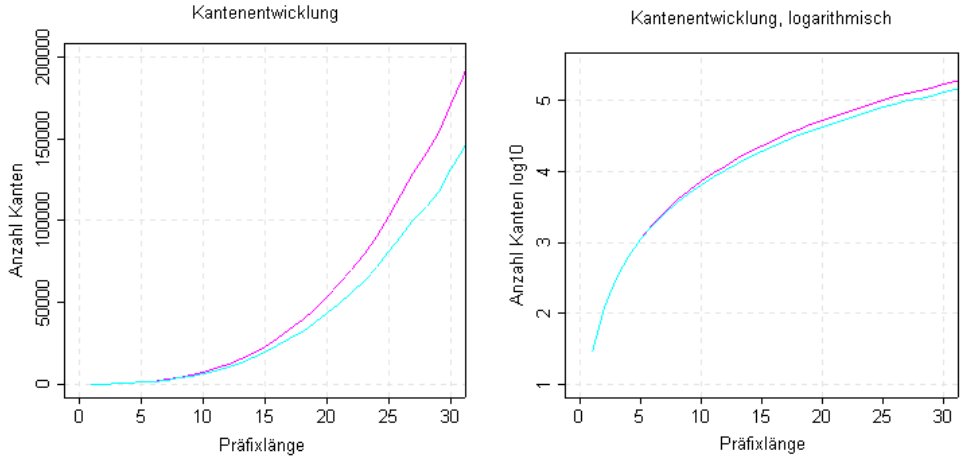

Abbildung 6.4.: Durchschnittliche Kantenanzahl im Suchraum, mit und ohne Einfrieren

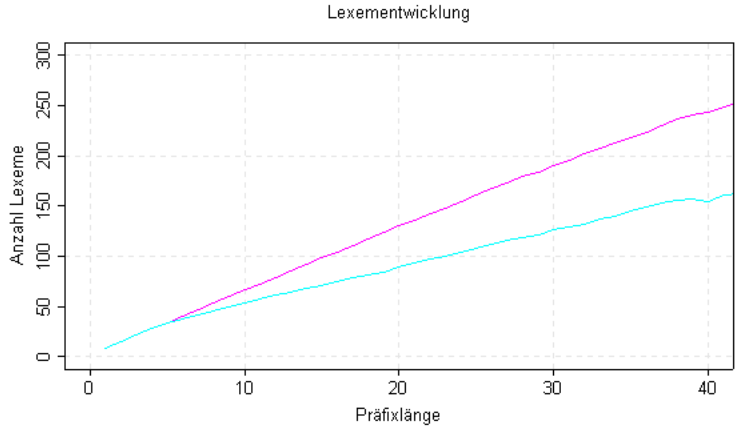

Abbildung 6.5.: Durchnittliche Lexemanzahl im Suchraum, mit und ohne Einfrieren

6.2.3. Auswertung

Abbildung 6.4 zeigt die Entwicklung der Kantenanzahl im Suchraum mit und ohne Einfrieren. Tabelle 6.1 zeigt eine Zeitersparnis von durchschnittlich 21.7% bei einer Qualitätseinbuße, bei 3% der Sätze.

gleiches Endergebnis	70.3%
mit Heuristik besser bewertet	13.3%
ohne Heuristik besser bewertet	16.3%
Zeitbedarf bei gleichem Ergebnis	78.3%

Tabelle 6.1.: Vergleich aktivierte VS deaktivierte Einfrierheuristik

6.3. Zielvorgaben und unvollständige Transformation

Die transformationsbasierte Suche terminiert, abgesehen von einer Zeitbegrenzung, erst, wenn keine Verbesserungen mehr möglich sind. Die schwerer bestraften Konflikte werden zuerst zu reparieren versucht. Daraus resultiert, dass auch wenn schwer bestrafte Konflikte nicht behoben werden konnten, die Bewertung der resultierenden Analyse einen bestimmten Wert nicht mehr übersteigen kann, evtl. bestehende weniger schwer bestrafte Konflikte weitere Reparaturversuche initiieren.

Ein früher Abbruch der Suche spart also Zeit auf Kosten des Resultats dieser Suche. Dies geschieht unter der Annahme, dass in einem späteren Suchlauf mehr Informationen zur Verfügung stehen, die die Struktur des Suchraumes derart verändern, dass eine bessere Analyse als Resultat dieses Suchlaufes wenig bis keinen Einfluss auf Qualität und Zeitbedarf des späteren Suchlaufes hat. In einem nichtinkrementellen Parsingprozess mit nur einem Suchlauf ist ein solches verfrühtes Abbrechen also nicht zielführend. Das hier vorgestellte inkrementelle Vorgehen kennt zwei Mechanismen, durch die der Suchraum erweitert und ein weiterer Suchlauf initiiert werden kann. Zum einen durch die Erweiterung der Eingabe um ein Wort, zum anderen durch einen Reanalyseprozess, bei dem derzeit eingefrorene Kanten dem Suchraum hinzugefügt werden.

Im inkrementellen Umfeld könnte ein schwerer Konflikt an der Unvollständigkeit des Satzes liegen und ein oder wenige Schritte später bereits behebbar sein. Es lohnt sich also nicht, eine Analyse mit einem schweren, unbehebbaren Konflikt im Detail zu optimieren, wenn zu erwarten ist, dass im nächsten Schritt diese Optimierungen im Kleinen durch eine Veränderung im Großen ungültig werden könnten.

Die Zeitersparnis bei einem inkrementellen Schritt lohnt sich nur in einem *pull*-Modell, bei dem die aus dem nächsten Wort bestehende Information tatsächlich früher eingebracht werden kann. Die Zeitersparnis vor einer Reanalyse hingegen lohnt sich auch und vor allem in einem *push*-Modell, da die für ein Inkrement zur Verfügung stehende Zeit von außen vorgegeben ist. Eine Reanalyse sollte dann, wenn nötig, so früh wie möglich innerhalb dieser Zeitspanne gestartet werden.

6.3.1. Definition eines neuen Abbruchkriteriums

Aufbauend auf den obigen Überlegungen kann ein weiteres Abbruchkriterium für die Suche formuliert werden. Gegeben eine äußere Erwartung an die Qualität der resultierenden Analyse, kann die Suche abgebrochen werden sobald die nicht behebbaren Konflikte diese Erwartung selbst unerfüllbar machen, d.h. sobald das Produkt der Strafwerte aller unbehebbaren Konflikte die erwartete Mindestbewertung unterschreitet. Ein numerischer Zielwert lässt sich über die Bewertung der vorangegangenen Analysen formulieren und ist abhängig von Zweck des Abbruchs, also ob der nächste Suchlauf nach einer Erweiterung der Eingabe oder einer Reaktivierung eingefrorener Kanten erfolgt.

Ein plötzlicher starker Abfall in der Bewertung zwischen zwei inkrementellen Erweiterungen kann auf einen kurzfristigen Konflikt hindeuten. In je mehr aufeinander folgenden Schritten dieser Konflikt auftritt, desto wahrscheinlicher ist es, dass er persistent ist. Bleiben die Zwischenergebnisse über mehrere Schritte hinweg schlecht, steigt die Wahrscheinlichkeit, dass auch die weiteren Analysen schlecht bewertet sind. Einbrüche in der Bewertung von Zwischenergebnissen sollten sich also nur verzögert auf die Erwartung auswirken, bzw. die Erwartung sollte sich graduell an die Ergebnisqualität anpassen.

Dieses Verhalten lässt sich mit dem folgenden Algorithmus modellieren. Sei E_{i-1} die Erwartung im letzten Schritt und B_{i-1} die Bewertung der resultierenden Analyse im letzten Schritt und f ein einstellbarer Verfallsfaktor mit $0 \leq f \leq 1$. Dann gilt für die Erwartung E_n des Suchlaufes im aktuellen Schritt

- $E_n = f \cdot B_{i-1}$, falls $B_{i-1} > E_{i-1}$
- sonst $E_n = f \cdot E_{i-1}$

Die Erwartung für einen Suchlauf ist abhängig davon, ob die Erwartung im letzten inkrementellen Schritt erfüllt wurde oder ob der Suchlauf abgebrochen wurde. Wurde die Erwartung erfüllt, ergibt die Bewertung der resultierenden Analyse multipliziert mit einem einstellbaren Faktor die neue Erwartung. Nach einem Abbruch ist die resultierende Analyse keine gute Grundlage für die weitere Erwartung, da sie beliebig niedrig bewertet sein könnte. Die Erwartung würde sehr schnell einbrechen. Stattdessen wird die letzte Erwartung als Berechnungsgrundlage genommen. Bei dauerhaft niedrigen Analysebewertungen gleicht sich die Erwartung dann exponentiell diesem niedrigem Stand an.

Zu beachten ist, dass am Ende des Satzes weiteres Aufschieben keinen Sinn mehr macht. Auch wenn dann noch ein schwerer Konflikt in der Analyse vorhanden ist, sollte diese als Ausgabe des gesamten Parsingprozesses weiter optimiert werden.

Für den Zweck der Reanalyse lässt sich ein ähnliches Vorgehen definieren. Sei B_{i-1} die Bewertung der resultierenden Analyse im letzten Schritt und f ein einstellbarer Verfallsfaktor mit $0 \leq f \leq 1$. Dann gilt für die Erwartung im aktuellen Schritt $E_n = f \cdot B_{i-1}$. Kann dieser Wert nicht erreicht werden und bricht die Suche also mit einer Analyse mit Bewertung $B_i < E_n$ ab, wird die Suche unter Einbeziehung der zu reaktivierenden Elemente wiederholt.

Die Buchführung der unbehebbaren Konflikte geschieht über den bereits erwähnten Mechanismus der *unremovable Conflicts*-Liste. Es ist zu beachten dass die auf diese Weise als unbehebbar markierten Konflikte als Nebeneffekt der Reparaturen anderer Konflikte trotzdem noch behoben werden können.

Auf diese Weise kann durch die oben beschrieben erwartungsbasierte Heuristik die Behebung eines Konfliktes verzögert werden, da er zwar nicht direkt, aber bei der Reparatur eines anderen Konfliktes behoben wird. Ist dieser andere Konflikt jedoch weniger hart bestraft und bricht die Suche ab, nachdem der erste Konflikt nicht direkt behoben werden konnte, bleiben beide Konflikte unbehoben, bis die Erwartung weit genug abgesunken ist.

So gibt es zum Beispiel ein binäres Constraint das Substantivierungen vor Nomen bestraft und ein unäre Constraint das substantivierte Adjektive pauschal bestraft. Das erste Constraint ist schwerer bestraft, der erste Reparaturversuch wird also von diesem ausgehen. Jetzt kann der seltene Fall eintreten, dass die Reparatur des ersten Konfliktes misslingt, beim zweiten, das unär ist und damit die zu verändernde Domäne genauer vorgibt, jedoch erfolgreich ist. Ist der Konflikt mit dem substantivierten Adjektiv behoben, ist damit auch die Substantivierung vor einem Nomen entfallen.

7. Evaluation

In diesem Kapitel werden die verschiedenen in dieser Arbeit vorgestellten Algorithmen und Heuristiken miteinander verglichen. Evaluiert werden soll dabei sowohl die Korrektheit des Ergebnisses als auch die zeitliche Performance. Dazu will ich zuerst ein wenig über die Messgrößen, die Testumgebung und die zu vergleichenden Algorithmen sagen, bevor wir uns den Ergebnissen selbst zuwenden.

7.1. Messgrößen

Die interessanten Größen bei der Evaluation sind die Korrektheit und die Zeitbedarf. Um zwei Vorgehensweisen zu vergleichen, müssen diese Größen in geeigneter Weise quantifiziert werden.

Um die Korrektheit zu messen, stehen zwei Methoden zur Verfügung, der Vergleich über die Bewertung durch die Grammatik und der Vergleich über eine dritte, als korrekt gesetzte Analyse für jeden Satz, z.B. aus einem handannotierten Korpus. Im ersten Verfahren wird die Bewertung verglichen und der Unterschied in der Ergebnisqualität lässt sich durch den Unterschied in der Bewertung quantifizieren. Im zweiten Verfahren lassen sich nur drei Fälle unterscheiden, die Analysen sind beide korrekt, beide nicht korrekt oder es ist eine von beiden korrekt. Sind sie beide nicht korrekt, lässt sich also nicht sagen, ob eine besser ist als die andere. Gilt eine von beiden als korrekt, lässt sich nicht unterscheiden, ob die andere nur eine leichte Abweichung ist oder gar keine geeignete Lösung.

Der Nachteil der Bewertung durch die Grammatik ist der, dass Modellierungsfehler in der Grammatik zu einer fälschlicher Weise besser bewerteten Analyse führen können.

Jeder hier zu vergleichenden Variante des CDG-Parsers liegt die gleiche Grammatik zugrunde. Die Grammatik ist bei dieser Arbeit als gegeben zu betrachtet, also Teil der Rahmenbedingungen und nicht der zu testenden Komponenten. Getestet wird die Fähigkeit der verschiedenen Konfigurationen, bei gegebener Grammatik die bessere Analyse zu ermitteln. Aus diesem Grund wird hier als Vergleichsmethode für Analysen die Bewertung durch die Grammatik benutzt.

Der Vergleich in der Ergebnisgüte lässt sich durch das Verhältnis $r = B(a_1)/B(a_2)$ der beiden Bewertungen angeben. Eine Verhältnis von $r = 1$ zeigt gleichwertige Analysen an, bei $r > 1$ ist die erste Analyse besser, bei $r < 1$ die zweite. Das Verhältnis lässt sich nicht angeben, wenn eine der beiden Analysen ein hartes Constraint verletzt, also eine Bewertung von 0 hat.

Der Zeitbedarf lässt sich direkt über die benötigte Zeit oder über die Zahl der Iterationen im Programmablauf, also z.B. die Zahl der Reparaturversuche, messen. Das erste Vorgehen hat ein Problem bei der Reproduzierbarkeit, da die Zeit von der Umge-

bung bei der Ausführung, also insbesondere von der verwendeten Hardware abhängt. Der Programmablauf selbst ist, abgesehen von einem zeitbegrenzungsbedingten Abbruch, unabhängig von der Ausführungsumgebung und damit reproduzierbar. Es ist jedoch kein Schritt im Programmablauf in der Anzahl seiner Iterationen repräsentativ für den Zeitbedarf des Programms. Der Programmablauf unterscheidet zwei wesentliche Phasen, den Aufbau des Suchraums und die Suche. Die Reparaturversuche treten nur in der zweiten Phase auf. Einsparungen wie etwa eine Reduzierung der zu erzeugenden Kanten würde über die Zählung der Reparaturversuche nicht erfasst werden. Auch Einsparungen innerhalb eines Reparaturversuchs würden ebenfalls nicht erfasst. Eine Reduzierung der Kanten im Suchraum hat aber eben eine solche Beschleunigung der Reparaturversuche zur Folge.

Die hier verwendete Methode ist die der direkten Zeitmessung in Millisekunden. Der Unterschied im Zeitbedarf lässt sich über das Verhältnis der benötigten Zeiten ausdrücken. Alle Testläufe werden in gleicher Hard- und Softwareumgebung durchgeführt. Der Vergleich des Zeitbedarfs ist nur für solche Sätze interessant, bei denen die beiden zu vergleichenden Verfahren das gleiche Ergebnis erzeugt haben. Eingesparte Zeit bei verschiedenen Ergebnissen ist nicht vergleichbar. Das Verhältnis des Zeitbedarfs wird also nur für solche Sätze bestimmt, bei denen beide Verfahren gleichwertige Analysen bestimmt haben. In allen Testläufen wird eine Zeitbegrenzung eingestellt. Dies ist notwendig, da die transformationsbasierte Suche ein im Worst-Case exponentielles Zeitverhalten aufweist. Der Anteil der Sätze, bei denen ein Verfahren durch die Zeitbegrenzung abgebrochen wurde, ist für jeden Testlauf angegeben.

7.2. Testumgebung

Die Testdaten bestehen aus 1000 Sätzen des Negra-Korpus [negra]
CDG-Einstellungen:

Zeitlimit (pro inkrementellem Schritt), eingestellt auf 5 Minuten

Maximale Schrittzahl Maximale Anzahl an Reparaturschritten zwischen zwei Maxima in der transformationsbasierten Suche, eingestellt auf 22

NONSPEC-Bestrafung Strafwert des Constraints, das Kanten mit NONSPEC als Regenten bestraft, eingestellt auf $0,95$

ignorethreshold alle Konflikte mit einem Strafwert über dieser Schwelle initiieren keine Reparaturversuche und dienen nur dem Vergleich von Kanten, eingestellt auf $0,951$

7.3. Ergebnisse

7.3.1. Inkrementelle Zeitüberlegenheit

Ein Ziel der inkrementellen Verarbeitung war es, gegenüber dem nichtinkrementellen Vorgehen einen Verarbeitungsvorsprung aufzubauen. Ab Vorliegen des gesamten Satzes

sollte der inkrementelle Algorithmus weniger Zeit benötigen als der nichtinkrementelle insgesamt.

Zum Vergleich werden hier zwei nichtinkrementelle Algorithmen herangezogen, die in Kapitel 3 vorgestellten "Combined" und "Threefold". "Combined" kommt mit einem Suchlauf für den ganzen Satz aus, während "Threefold" Teilsätze anhand von Satzzeichen erkennt und für jeden Teilsatz eine lokale Analyse erstellt, um diese in einem globalen Suchlauf zu einer Analyse des ganzen Satzes zu kombinieren und in einem dritten Suchlauf noch einmal zu optimieren. Die initiale Analyse des zweiten Suchlaufs ist also aus den Analysen der Teilsätze zusammengesetzt. Der Vergleich mit den nichtinkrementellen Algorithmen findet sich in Tabelle 7.1. Abbildung 7.1 zeigt, wie viel Prozent der nichtinkremnetellen Verarbeitungszeit durch inkrementelle Zeitüberlegenheit eingespart wurde.

	VS combined	VS threefold
Bewertung des Endergebnisses:		
gleich	$54,0\%$	$56,82\%$
inkrementell besser	$27,4\%$	$17,4\%$
inkrementell schlechter	$18,5\%$	$25,7\%$
inkrementell Zeitüberlegenheit:		
Ab letztem Token (Satzendzeichen)	$95,8\%$	$97,6\%$
Ab letztem Wort	$68,0\%$	$71,5\%$
Zeitbedarf insgesamt ($inkrementell/nichtinkrementell$)	$2,20$	$2,08$
von Zeitbegrenzung betroffen	$< 1\%$	$< 1\%$

Tabelle 7.1.: Vergleich Präfixparsing vs Pseudoinkrementelles Parsing

Die Ergebnisqualität des inkrementellen Parsings ist besser als beim rein nichtinkrementellen "Combined"-Vorgehen, jedoch schlechter als beim "Threefold". Die Ergebnisse stimmen in beiden Fällen für ca. 55% der Sätze exakt überein. Bei den Sätzen mit abweichenden Ergebnissen ist festzustellen, dass Combined in der Differenz mehr schlechter bewertete und Threefold mehr besser bewertete Ergebnisse liefert als das inkrementelle Parsing. Die Differenz beträgt 10% der Sätze mit besser bewerteten Analysen gegenüber Combined und 7% schlechtere gegenüber Threefold.

Für die inkrementelle Zeitüberlegenheit ist eine Messung ab dem letzten Token und ab dem vorletzten Token angegeben. Die Sätze im Testkorpus sind zum Großteil mit einem Punkt beendet. Setzt man den Referenzpunkt für die inkrementelle Zeitüberlegenheit, also ab wann der Satz als vollständig vorliegend gilt, auf das Eintreffen dieses Satzzeichens, hat der inkrementelle Prozess bereits eine alle Wörter des Satzes beinhaltende Eingabe verarbeitet. Ohne die Information, dass der Satz beendet ist, sind jedoch noch eventuelle NONSPEC-Anbindungen vorhanden, die nach Vorliegen des ganzen Satzes inklusive Satzendzeichen ersetzt werden müssen. Es zeigt sich, dass das Ziel der inkrementellen Zeitüberlegenheit für den ersten Fall mit über 95% als erfüllt gelten kann. Setzt man den Referenzpunkt auf das Eintreffen des letzten Wortes, ist das Kriterium der inkrementellen Zeitüberlegenheit mit einer Erfolgsquote

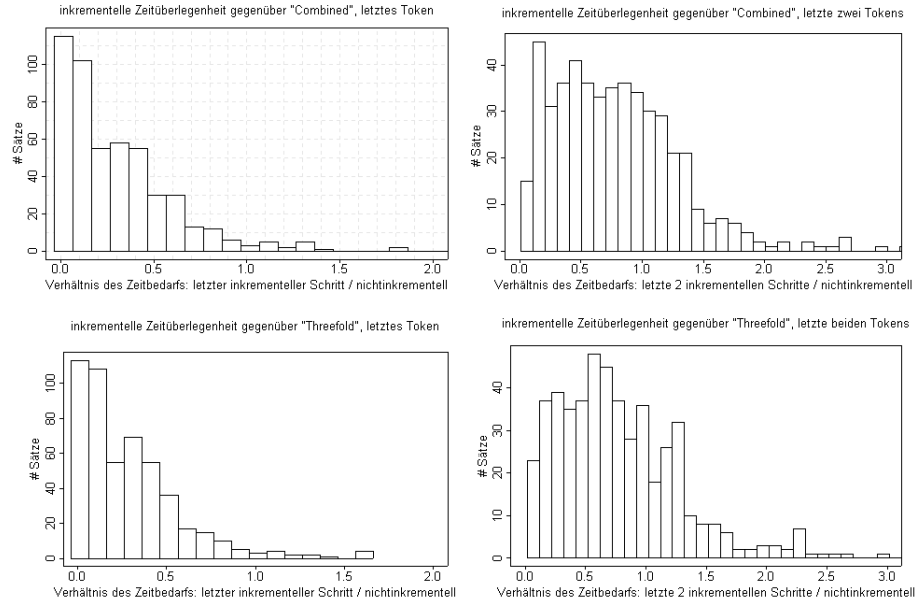

Abbildung 7.1.: Vergleich, wie viel Zeit der letzte bzw. die letzten beiden inkrementellen Schritte benötigen gegenüber den nichtinkrementellen Verfahren. Ein Wert von 1 entspricht gleichem Zeitbedarf, bei kleineren Werten ist das inkrementelle Verfahren inkrementell Zeitüberlegen

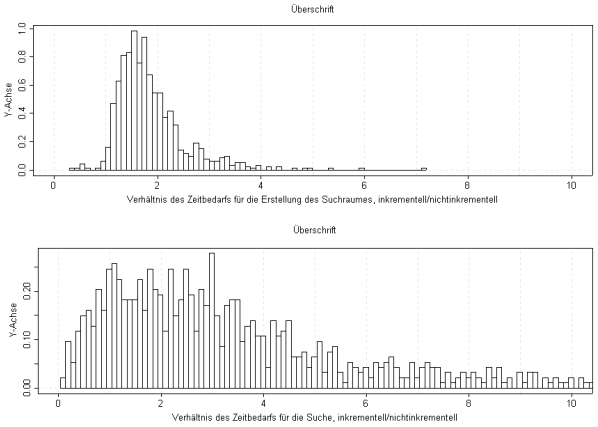

Abbildung 7.2.: Vergleich des Zeitbedarfs zwischen inkrementellem Parsing und nichtinkrementellem "Combined" Parsing

von ca. 70% für die meisten, jedoch nicht für alle Sätze erfüllt.

Der Gesamtzeitbedarf beträgt durchschnittlich etwas mehr als das Doppelte des nichtinkrementellen Zeitbedarfs. Abbildung 7.2 zeigt, wie sich das Verhältnis jeweils bei Aufbau des Suchraums und der sich darin anschließenden Suche darstellt. Der Aufbau des Suchraumes dauert im inkrementellen Fall, über alle inkrementellen Schritte addiert etwa $1,0$ bis $2,5$ mal so lange, mit einem Peak bei $1,5$. Bei der Suche ist der Zeitbedarf deutlich größer gestreut.

7.3.2. Vergleich mit dem pseudoinkrementellen Parsing

Ein weiterer Benchmark für das hier vorgestellte inkrementelle Vorgehen ist der Vergleich mit dem bereits vorher im CDG-System möglichen pseudoinkrementellen Parsing.

gleiches Endergebnis	$75,7\%$
integriert inkrementell besser	$12,8\%$
pseudoinkrementell besser	$11,5\%$
Verhältnis des Zeitbedarfs (Inkrementell / Pseudo, bei gleichem Ergebnis)	0.533

Tabelle 7.2.: Vergleich inkrementelles VS pseudoinkrementelles Parsing

Wie Tabelle 7.2 zeigt, sind die Ergebnisse der beiden Verfahren von ähnlicher Qualität. Sie stimmen für 75% der Fälle überein, die restlichen 25% zeigen etwa gleich verteilt auf die beiden Verfahren ein jeweils besseres Ergebnis. Das inkrementelle Vorgehen zeigt ein besseres Zeitverhalten, was insbesondere auf Einsparungen beim Aufbau des Suchraumes zurückzuführen ist. Abbildung 7.3 zeigt diesen Unterschied im Anteil des Suchraumaufbaus.

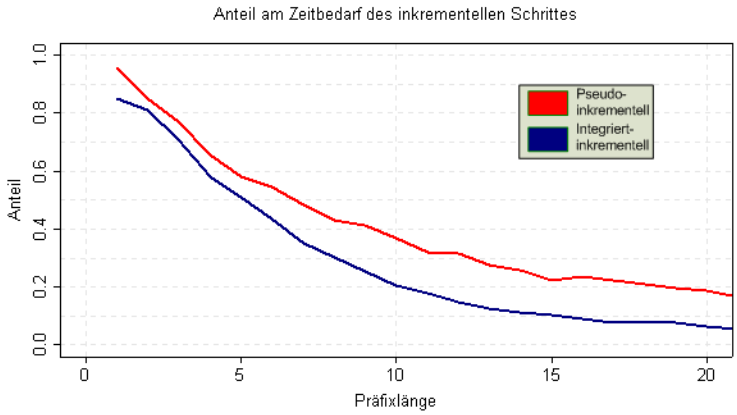

Abbildung 7.3.: Anteil des Suchraumaufbaus am Zeitbedarf

Das bessere Zeitverhalten des inkrementellen Vorgehens war zu erwarten, da durch die erhöhte Wiederverwendung bereits vorhandener Daten im Programmablauf, insbesondere den Elementen im Suchraum, der Neuaufbau dieser Daten eingespart wird. Die Ergebnisse ähnlicher Qualität waren ebenfalls zu erwarten, da im Wesentlichen in jedem Schritt die gleiche Suche im gleichen Suchraum stattfindet. Die verbliebenen Unterschiede sind auf Unterschiede bei der Auswahl der Anbindung des aktuell hinzugekommenen Tokens zurückzuführen.

7.3.3. Vergleich der verschiedenen Heuristiken

Konservative Heuristik

Eine der in Kapitel 6 eingeführten Heuristiken beruhte darauf, ein konservatives Verhalten zu erzwingen, indem Alternativen zu bestehenden Kanten eingefroren, also aus dem Suchraum entfernt werden.

Tabelle 7.3 zeigt, dass dieses Vorgehen in seiner reinen Form (Spalte "K") verheerende Auswirkungen auf die Ergebnisqualität zeigt. Für $78,2\%$ der Sätze wird ohne Verwendung dieser Heuristik ein besseres Ergebnis erzielt. Wir hatten zwei Möglichkeiten eingeführt, die Konservativität einzuschränken. Zum einen können Kanten am rechten Rand der Struktur vom Einfrieren ausgenommen werden, was einem strukturellen Lookahead entspricht. Zum anderen kann für eine Reanalyse das Einfrieren rückgängig gemacht werden.

Beide Heuristiken verbessern die Ergebnisqualität erheblich, der Anteil der Sätze für die das Ergebnis mit Heuristik schlechter ist sinkt von $78,2\%$ auf $28,3\%$ (Reanalyse) bzw. $15,3\%$ (strukturelles Lookahead). Dafür steigt in beiden Fällen der Zeitbedarf. Im direkten Vergleich dieser Erweiterungen erweist sich das Lookahead dabei in beiden Aspekten als die bessere. Die konservative Heuristik mit jeweils einer dieser Erweiterungen führt jedoch auch weiterhin zu schlechteren Ergebnissen als ganz ohne die

konservative Heuristik. Zum Vergleich ist ebenfalls ein Ergebnis für ein Lookahead mit fester Wortanzahl angegeben.

Bemerkenswert ist, dass die Kombination von Reanalyse und strukturellem Lookahead die Heuristik derart verbessert, dass sie sowohl von der Ergebnisqualität als auch vom Zeitbedarf besser abschneidet als das inkrementelle Parsing ohne diese Heuristik. Die besseren Ergebnisse lassen sich darauf zurückführen, dass die Reanalyse nicht nur durch eine Reaktivierung eingefrorener Kanten bessere Analysen, die diese Kanten enthalten, ermöglicht, sondern auch generell eine zweite Suche über dem selben Satzpräfix initiieren. Diese zweite Suche geht von der Analyse aus, die das Ergebnis des ersten Suchlaufes ist. Konflikte, die im ersten Lauf nicht behoben werden konnten, können im zweiten Lauf unter geänderten Ausgangsbedingungen evtl. behoben werden. Der zusätzliche Zeitbedarf für diesen zweiten Suchlauf steht der Zeitersparnis in allen inkrementellen Schritten gegenüber, in denen auch im verkleinerten Suchraum eine gut bewertete Analyse gefunden werden konnte. Das Lookahead scheint notwendig zu sein, um die Qualität des ersten Ergebnisses vor der Reanalyse zu gewährleisten. Ist diese durch das lookahead-freie Einfrieren nicht gegeben, kann die Reanalyse nur diese mangelnde Ergebnisqualität des ersten Suchlaufes ausgleichen, nicht jedoch den doppelten Suchlauf dazu nutzen das Ergebnis gegenüber dem einfachen inkrementellen Parsing verbessern.

Die Steuerung der Reanalyse initiiert über den Abfall der Bewertung auch dann eine Reanalyse, wenn der Einbruch gar nicht auf eingefrorene aber für ein besseres Ergebnis notwendige Kanten zurückzuführen ist. Diese zusätzlichen Reanalysen sind vor dem oben beschriebenen Hintergrund nicht als redundant, sondern als zentral für eine Verbesserung der Analyseergebnisse zu betrachten.

	K	K-RA	K-LA3	K-SLA	K-RA-SLA
Bewertung des Endergebnisses:					
gleich	$20,4\%$	$58,2\%$	$44,9\%$	$73,5\%$	$77,1\%$
mit Heuristik besser	$1,4\%$	$13,5\%$	$6,6\%$	$11,3\%$	$13,7\%$
ohne Heuristik besser	$78,2\%$	$28,3\%$	$48,4\%$	$15,3\%$	$10,3\%$
Zeitbedarf insgesamt					
mit/ohne Heuristik	$0,45$	$0,90$	$0,58$	$0,75$	$0,94$
von Zeitbegrenzung betroffen	$< 1\%$	$< 1\%$	$< 1\%$	1%	1%

Tabelle 7.3.: Vergleich der Ergebnisse mit und ohne konservative Heuristik, K = konservativ, RA = mit Reanalyse (Faktor $0,5$), LA3 = Lookahead um drei Wörter ,SLA = mit Strukturellem Lookahead

Mit der Reanalyse ist ein Parameter verbunden, der steuert, um welchen Faktor die Bewertung zwischen zwei Analysen mindestens abfallen muss, damit eine Reanalyse initiiert wird. Dieser Faktor ist bei den Ergebnissen von Tabelle 7.3 auf den Wert $0,5$ eingestellt. Tabelle 7.4 zeigt, wie sich eine Veränderung dieses Wertes auswirkt. Je höher der Faktor, um so öfter findet eine Reanalyse statt. Die Reanalyse wirkt sich positiv auf die Ergebnisbewertung aus, kostet jedoch Zeit. Über eine Veränderung des Reanalysefaktors lässt sich also Zeitersparnis gegen Ergebnisqualität eintauschen.

Faktor $f =$	$0,3$	$0,7$	$0,9$
Bewertung des Endergebnisses:			
gleich	$99,2\%$	$98,8\%$	$95,7\%$
mit f besser	$0,0\%$	$1,0\%$	$3,5\%$
mit $0,5$ besser	$0,8\%$	$0,2\%$	$0,8\%$
Zeitbedarf insgesamt			
anderer Faktor zu f=$0,5$	$0,98$	$1,02$	$1,13$
von Zeitbegrenzung betroffen	$1,5\%$	$1,5\%$	$1,5\%$

Tabelle 7.4.: Vergleich verschiedener Werte für den Reanalysefaktor bei konservativer Heuristik plus Reanalyse; der Reanalysefaktor gibt den relativen Bewertungseinbruch an, ab dem eine Reanalyse initiiert wird.

Einfrier-Heuristik

Die zweite in Kapitel 6 vorgestellte Heuristik ist die Einfrier-Heuristik, bei der möglichst nur solche Kanten und Lexeme aus dem Suchraum ausgeschlossen werden, die später nicht für eine Analyse gebraucht werden. Dafür werden Strukturen in den Zwischenergebnissen ermittelt, die als stabil gelten können. Ziel des gegenüber der konservativen Heuristik vorsichtigeren Vorgehens war es, die Reaktivierung von Kanten in einer Reanalyse unnötig zu machen. Ein weiterer Unterschied zur konservativen Heuristik besteht darin, dass auch von dem neu hinzugekommenen Wort ausgehende Kanten bereits eingefroren werden können, wenn sie zu Wörtern in einer als stabil geltenden Struktur hinführen.

Tabelle 7.5 zeigt, dass ohne Reanalyse die Einfrier-Heuristik zu einer großen Zeitersparnis bei einer leichten Verschlechterung der Ergebnisse führt, in der Differenz für 4% der Sätze. Das Ziel, auch ohne Reaktivierung von Kanten die Ergebnisqualität zu halten und gleichzeitig den Zeitaufwand zu verringern, wurde also nicht vollständig erreicht. Wie in der zweiten Spalte zu sehen, führt die Hinzunahme eines Reanalyseprozesses, also Reaktivierung eingefrorener Kanten und erneuter Suchlauf, wenn die Analysebewertung zu stark abfällt, zu einer Verbesserung der Ergebnisqualität. Dies geht aber wie oben besprochen nur zum Teil auf die Reaktivierung der Kanten zurück und ist auch durch den zweiten Suchlauf selbst bedingt. Wie Tabelle 7.6 zeigt, ist die Performanz der Einfrier-Heuristik der konservativen Heuristik mit strukturellem Lookahead sehr nahe.

Zielvorgaben

Ziel dieser Heuristik war es, die Bearbeitung abzubrechen und zum nächsten Inkrement überzugehen, wenn eine aus der Bewertung des letzten Zwischenergebnisses abgeleitete Zielvorgabe nicht mehr erreichbar ist.

Wie Tabelle 7.7 zeigt, ist die auf diese Weise eingesparte Zeit durch eine leicht gesunkene Ergebnisqualität erkauft und nicht wie erhofft ohne negative Auswirkung auf die Qualität. Durch ein Anheben des Faktors, aus dem sich die Zielvorgabe berechnet, kann also Ergebnisqualität gegen Zeitersparnis eingetauscht werden. Dieser Einbuße

	Einfrieren	mit Reanalyse
Bewertung des Endergebnisses:		
gleich	$70,3\%$	$73,6\%$
mit Heuristik besser	$13,3\%$	$14,7\%$
ohne Heuristik besser	$16,4\%$	$11,8\%$
Zeitbedarf insgesamt		
mit zu ohne Heuristik	$0,78$	$0,98$
von Zeitbegrenzung betroffen	$< 1\%$	$< 1\%$

Tabelle 7.5.: Vergleich mit und ohne Einfrier-Heuristik

	ohne Reanalyse	mit Reanalyse
Bewertung des Endergebnisses:		
gleich	$67,7\%$	$74,1\%$
Einfrieren besser	$16,0\%$	$13,4\%$
konservativ besser	$16,2\%$	$12,4\%$
Zeitbedarf insgesamt		
mit zu ohne Heuristik	$1,02$	$0,99$
von Zeitbegrenzung betroffen	$< 1\%$	$< 1\%$

Tabelle 7.6.: Vergleich Einfrier- VS konservative Heuristik

bei der Qualität lässt sich darauf zurückführen, dass ein vergeblicher Reparaturversuch kein verlässlicher Indikator für die Unbehebbarkeit des betroffenen Konfliktes ist. Der Konflikt kann zum einen später bei der Reparatur eines anderen Konfliktes behoben werden, zum anderen könnte in einem zusätzlicher Suchlauf mit veränderter Ausgangslage die Reparatur gelingen. Wie die Auswertung der Reanalyse in den vorangegangenen Abschnitten zeigt, ist die gezielte Initiierung weiterer Suchläufe auf einen Bewertungseinbruch.

Im verstärktem Maße wird die Zeitbegrenzung überschritten, die pro Schritt und nicht für den gesamten Prozess gilt. Daran ist zu erkennen, dass durch die Zielvorgabe vermehrt Parsingaufwand an das Ende des Satzes verschoben wird, was zu erwarten war, jedoch auch dem Ziel der inkrementellen Zeitüberlegenheit entgegen läuft.

Faktor der Zielvorgabe	0,7	0,8	0,9
Bewertung des Endergebnisses:			
gleich	87,1%	87,2%	86,5%
mit Zielvorgabe besser	5,2%	4,9%	4,9%
ohne Zielvorgabe besser	7,8%	7,8%	8,5%
Zeitbedarf insgesamt			
mit zu ohne Heuristik	0,87	0,86	0,85
von Zeitbegrenzung betroffen	1,4%	1,5%	2,0%

Tabelle 7.7.: Vergleich: mit und ohne Zielvorgabe

8. Ausblick

8.1. Tagging

Wie im Kapitel Implementation beschrieben, wird beim Tagging viel Wiederverwendungspotential des inkrementellen Ansatzes verschenkt. Die Ausgabe des Taggers ist eine der wichtigsten Grundlagen bei der Kantenbewertung und fließt bereits bei der initialen Bewertung einer Kante ein. Dort werden nur die einstelligen, nicht kontextsensitiven Constraints ausgewertet, also gerade jene die nur die Eigenschaften der Kante selbst (gebundene Lexeme, Kantenbeschriftung, Richtung der Kante) mit einbeziehen. Andere Constraints können nur an einer konkreten Analyse ausgewertet werden, da dort der Kontext bekannt ist. Diese initiale Auswertung der unären Constraints hat beim inkrementellen Parsing den Vorteil, dass sie nicht wiederholt werden muss, da sich die relevanten Eigenschaften der Kanten nicht ändern. Es gibt jedoch Constraints, die auf externen statistischen Prädiktoren beruhen, insbesondere dem POS-Tagger und dem PP-Attacher. Diese ziehen bei ihrer Bewertung sehr wohl auch Informationen aus dem Kontext der Kante in Betracht, wenn auch nicht über andere Kanten, so doch über die anderen Wörter im Satz. Es gibt zwei Gründe, warum diese Informationen beim Aufbau einer Kante mit ausgewertet werden, die binären Constraints jedoch nicht. Zum einen liegen die Informationen über andere Wörter im Satz im Gegensatz zu solchen über andere Kanten im selben Dependenzbaum zu diesem Zeitpunkt bereits vor, es ist also möglich sie auszuwerten. Zum anderen hat insbesondere der POS-Tagger einen wichtigen Anteil bei der Bewertung einer Kante, der so früh wie möglich in die Steuerung der Suche eingehen sollte, es ist also auch sinnvoll.

Beim inkrementellen Parsing ist nun aber eine dieser Voraussetzungen nicht mehr unbedingt erfüllt, beim Erzeugen einer Kante liegen nicht alle Wörter des Satzes vor. Der POS-Tagger kann also zu einem späteren Zeitpunkt zu einer anderen Bewertung kommen. Das hat zwei Konsequenzen für das inkrementelle Parsing. Die alten Kanten müssen nach jeder inkrementellen Erweiterung neu bewertet werden, was einen erneuten Aufruf des Taggers notwendig macht. Außerdem kann der Tagger mit falschen Entscheidungen falsche Strukturen erzwingen, die später wieder revidiert werden müssen.

8.1.1. Neubewertung

Die Neubewertung durch den Tagger wird dann notwendig, wenn der Tagger Wörter rechts vom aktuellen Wort mit einbezieht, entweder direkt durch ein Fenster in einem n-Gramm-Tagger, durch Pfadoptimierung in einem Viterbi-Algorithmus oder durch eine regelgesteuerte globale Optimierung. Ein Tagger der nur weiter links liegende Wörter betrachtet, wäre in seiner Bewertung über die inkrementellen Schritte konstant.

8.1.2. Falsche Entscheidungen des Taggers

Ein auf ganzen Sätzen trainierter Tagger begeht bestimmte systematische Fehler, da er die Möglichkeit weiterer Wörter außer Acht lässt und ein Präfix wie einen ganzen Satz behandelt. Ein Beispiel dafür ist das Wort "als" in seiner komparativen und seiner zeitlichen Bedeutung. Eine Sequenz wie "..., als in Frankfurt " wird vom Tagger in der komparativen Lesart getaggt, auch wenn der Satz später zu einem Temporalsatz ("..., als in Frankfurt noch ... wurde") erweitert wird und diese zweite Lesart bereits durch das Komma angedeutet wurde. Eine Erweiterung des Taggers um eine globale Regel, die nach einem Komma die zeitliche Lesart bevorzugt, hat auf einem Korpus von vollständigen Sätzen eine negative Auswirkung auf die Korrektheit der POS-Tags, verbessert jedoch das Tagging der Präfixe.

8.1.3. Inkrementelles Tagging

Es sind verschiedene Strategien denkbar, um die in dieser Arbeit beschriebenen Probleme mit sich verändernden Taggingergebnissen zu vermeiden. Diese Strategien wurden in dieser Arbeit nicht weiter verfolgt.

Tagger ohne Lookahead, ein Tagger der nur auf links vom aktuellen Wort stehende Wörter berücksichtigt, würde das Tag für dieses Wort unabhängig vom späteren Erweiterungen des Satzes wählen. Die Tags wären also zwischen den inkrementellen Schritten konstant, wie groß die Korrektheitseinbußen gegenüber dem nichtinkrementllem Tagging sind, ist zu evaluieren.

Training auf Präfixen, ein auf Präfixen trainierter Tagger wird in seiner Bewertung der Präfixe typische Erweiterungen des Präfixes berücksichtigen und somit Tags vergeben, die seltener revidiert werden müssen. Ausgeschlossen ist eine solche Revision jedoch nicht, so dass das Problem der zeitaufwändigen Neubewertung der Kanten weiterhin besteht. Inwieweit besseres und stabileres POS-Tagging auf Präfixen im Tradeoff mit evtl. schlechterem POS-Tagging auf ganzen Sätzen sich auf die Qualität und Geschwindigkeit des inkrementellen Parsings auswirkt, ist zu evaluieren.

Isolation fluktuierender Bewertungsanteile Ein Großteil des Zeit-Overheads durch über inkrementelle Schritte hinweg instabiles POS-Tagging resultiert daraus, dass vom Tagger verworfene Lexeme und damit auch Kanten nicht endgültig verworfen werden konnten. Da die Taggerbewertung in einem späteren inkrementellen Schritt anders ausfallen kann, müssen alle durch den Tagger verworfenen Kanten in jedem Schritt reevaluiert werden. Wenn es gelänge, solche Lexeme, die in allen möglichen Erweiterungen des Satzpräfixes vom Tagger verworfen werden, zu identifizieren und von solchen zu trennen, die evtl. reaktiviert werden müssen, kann dieser Overhead reduziert werden.

8.2. Grammatik

Die hier verwendete Grammatik ist, abgesehen von den Erweiterungen für den nur für unvollständige Sätze relevanten NONSPEC-Knoten, eine für vollständige Sätze optimierte Grammatik. Dies äußert sich in verschiedenen Phänomenen. Zum einen bestrafen die Constraints der Grammatik eine ungesättigte Valenz derart, dass bei einem unvollständigen Satzpräfix diese Valenz mit einem ungeeigneten Komplement erfüllt wird, was im Vergleich zur ungesättigten Valenz weniger stark bestraft wird. Ein Beispiel sind Adjektive, die als substantiviertes Adjektiv die Rolle des Nomens übernehmen können. Solche Substantivierungen kommen vor, sie weniger hart als eine Valenzverletzung zu bestrafen ist prinzipiell grammatikalisch richtig. Der Fall, dass das Nomen, da es nach dem Adjektiv im Satz steht, nur noch nicht im aktuellen Präfix enthalten ist, ist jedoch deutlich häufiger. Auf diese Weise wird jedes Adjektiv bei offener Valenz kurzzeitig, d.h. in einem Zwischenergebnis, substantiviert. Die von einem Wort am rechten Rand ausgehenden Kanten werden also in den ersten Schritten nach Erweiterung um dieses Wort häufig im jeweils nächsten Zwischenergebnis ersetzt. Inwieweit eine verzögerte Anbindung solcher suboptimaler Komplemente sich auf Geschwindigkeit und Korrektheit des inkrementellen Parsers auswirkt, ist zu evaluieren.

Ein anderes Phänomen sind Constraints, die die Anwesenheit von bestimmten Wörtern, Satzzeichen oder Konstruktionen weiter rechts im Satz verlangen, damit eine andere Konstruktion erlaubt ist. Ein Beispiel ist die Detektion von Fragen anhand des Fragezeichens am Ende des Satzes. Die Abwesenheit eines Fragezeichens in einem Präfix lässt keine Rückschlüsse auf dessen An- oder Abwesenheit im ganzen Satz zu. Um die Grammatik für unvollständige Sätze, wie sie im inkrementellen Parsing vorkommen, anzupassen, müssten Constraints, die implizit voraussetzen, dass ein vollständiger Satz vorliegt, verändert werden. So kann z.B. statt der negativen Evidenz, also der Abwesenheit eines Fragezeichens, die positiven Evidenz, also die Anwesenheit eines *anderen* Satzendzeichens als einem Fragezeichen, als Indiz dafür, dass der vorliegende Satz keine Frage ist, herangezogen werden. Die Annahme, das ein Satzendzeichen und somit ein vollständiger Satz vorliegt, wird dadurch explizit überprüft.

Die Grammatik wurde in der vorliegenden Arbeit als gegeben betrachtet und ihre Eignung für den inkrementellen Einsatz überprüft. Ein nächster Schritt wäre es, die Grammatik systematisch auf impliziten Annahmen wie die oben erwähnte hin zu überprüfen und anzupassen.

8.3. Verwendung der Zwischenergebnisse

In der vorliegenden Arbeit wurden die Zwischenergebnisse der inkrementellen Schritte nicht als Teil der Ausgabe betrachtet, ihre Aufgabe im Parsingprozess besteht darin, möglichst viele früh getroffene Entscheidungen in Form von Kanten in der Analyse an spätere inkrementelle Schritte weiterzugeben. Auf diese Weise sparen sie im weiteren Verlauf der Verarbeitung Zeit ein, was zu inkrementeller Zeitüberlegenheit gegenüber einem nichtinkrementellen Parser führen soll.

Für eine Anwendung im dynamischen Kontext ist es jedoch interessant, auch die

Analysen der einzelnen Satzpräfixe als Ausgabe des syntaktischen Parsings zu betrachten. Soll ein System bereits auf unvollständige Sätze reagieren können, muss es diese Zwischenergebnisse zur Verfügung haben. Im Zuge einer Syntax-Semantik-Integration könnte eine Voraussage über den Fortgang des Satzes gemacht werden. Dies ist nicht nur nützlich, um eine die weitere Verarbeitung zu beschleunigen, sollte diese Voraussage eintreffen, sondern auch um bereits auf die erwartete ganze Äußerung zu reagieren. Eine solche Reaktion könnte eine Planung der Antwort oder ein Planen oder bereits Ausführung der Handlung eines Roboters sein. Denkbar ist auch eine Fokussierung der sensorischen Aufmerksamkeit eines Roboters auf Gegenstände, auf die der Satz inhaltlich Bezug nimmt. Eine solche synchronisierte Fokussierung kann zum schnelleren Verständnis des Satzes beitragen, wenn auf diese Weise Informationen zeitnah aufgenommen werden können, mit denen dann Referenzen oder Ambiguitäten im weiteren Satzverlauf aufgelöst werden können.

Eine Aufnahme der Zwischenergebnisse in die Ausgabe des Parsers und eine Syntax-Semantik-Integration bereits auf Präfixebene wäre also unter verschiedenen Gesichtspunkten interessant.

8.4. Präfixkorpus

Für die bisher in diesem Kapitel angesprochenen Themen wäre ein Präfixkorpus, also eine Sammlung von Annotationen zu Satzpräfixen vorteilhaft. Eine Grammatik könnte darauf optimiert und Zwischenergebnisse evaluiert werden. Linguistische Korpora enthalten jedoch für gewöhnlich nur vollständige Sätze. Ein Präfixkorpus durch systematische Beschneidung der Syntaxbäume aus einem Satzkorpus zu gewinnen, ist nur bedingt zielführend, da eine global im Kontext des ganzen Satzes optimale Anbindung lokal im Präfix und insbesondere für andere mögliche Fortsetzungen des Präfixes nicht optimal sein muss. Befinden sich zwei unterschiedliche Sätze mit einem gemeinsamen Präfix im Ausgangskorpus, könnte es gar passieren, dass für dieses Präfix zwei verschiedene Annotationen im resultierenden Präfixkorpus enthalten sind. Eine Alternative ist die manuelle Erstellung eines solchen Korpus.

8.5. Andere Inkrementgrößen

In dieser Arbeit war die Inkrementgröße auf einzelne Worte beschränkt, d.h. jedes einzelne Wort initiiert einen eigenen Parsingschritt. Mit einer anderen Granularität, etwa auf geeignete Weise ermittelten Phrasen, lassen sich evtl. viele der hier vorgestellten Probleme mit temporären Konflikten am rechten Rand des bearbeiteten Präfixes vermeiden.

8.6. Mehrere nicht spezifizierte Knoten

Als Platzhalter für noch nicht erschienene Wörter wurde der Knoten NONSPEC als Regent eingeführt. Dieser Knoten hat außer seiner Position keine spezifizierten Eigen-

schaften. Statt hier einen generischen Knoten zu verwenden, wäre es denkbar, jedes Mal, wenn ein solcher Platzhalter benötigt wird, einen neuen Knoten zu erschaffen und bereits alle durch Constraints angeforderten Eigenschaften zu spezifizieren. Ein solcher Platzhalter könnte dann bereits vor der transformationsbasierten Suche mit dem neu hinzugekommenen Wort verglichen und evtl. ersetzt werden. Auch ließen sich auf diese Weise Erwartungen an den Rest des Satzes explizit machen. Eine der dabei zu lösenden Fragen ist, wann ein neuer Platzhalter erzeugt werden muss und wann ein alter wiederverwendet werden kann.

9. Fazit

In den vorangegangenen Kapiteln wurde beschrieben, wie Parsing mit Constraint-Dependency-Grammatiken inkrementell durchgeführt werden kann. Definiert wurde Inkrementalität des Parsings dabei über die inkrementelle Abbarbeitung der Eingabe, die es ermöglicht, den Prozess bereits bei unvollständig vorliegender Eingabe zu beginnen.

Im ersten Schritt wurde ein integriertes Vorgehen für das CDG-System implementiert. Gegenüber einem pseudoinkrementellen Vorgehen, bei dem zwischen verschiedenen Aufrufen des Parsers für die einzelnen Inkremente lediglich die jeweils letzte Ausgabe weitergegeben wurde, ist der neu implementierte Modus voll integriert, d.h. Daten werden großteils wiederverwendet und nicht für jedes Inkrement neu ermittelt. Darüber hinaus wurde versucht, Informationen aus dem Ablauf des Parsingprozesses für den nächsten inkrementellen Schritt zu verwenden, also insbesondere, welche Konflikte nicht behoben werden konnten und die Abschätzungen für das Potential von Kanten und Lexemen. Die Wiederverwendung dieser Informationen hat sich als nicht praktikabel erwiesen, da sie in vielen Fällen nicht mehr gültig sind, wenn der Satz und damit der Raum der Hypothesen erweitert wird.

Der integrierte inkrementelle Modus bringt deutliche Zeitersparnisse um 47%, jedoch kaum Verbesserungen bei der Genauigkeit. Dies war in Anbetracht der Tatsache, dass über die in beiden Verfahren vorkommende Weitergabe des letzten Zwischenergebnisses keine zusätzlichen Informationen übernommen werden konnten, zu erwarten. Die Zeitersparnis ist auf die stark verbesserte Wiederverwendung der Daten zurückzuführen.

Die Grammatik hat sich als robust genug erwiesen, um prinzipiell auch mit unvollständigen Satzpräfixen umgehen zu können, auch wenn sie nicht für solche optimiert ist. Eine solche Optimierung lässt jedoch weitere Verbesserung der Performanz erwarten. An der Grammatik mussten jedoch einige Anpassungen vorgenommen werden, um mit temporären Anbindungen, also mit Kanten mit einem nicht spezifizierten Regenten umgehen zu können. Der POS-Tagger hingehen konnte nur mit größerem Aufwand inkrementell verwendet werden. Dadurch, dass der POS-Tagger ein Wort in einem Präfix evtl. anders bewertet als im vollständigen Satz, musste die Aktualität der Bewertung durch eine Zeitaufwändige Reevaluation gewährleistet werden. Dieses Problem tritt im nicht-inkrementellen Vorgehen nicht auf, da dort der Tagger nur einmal am Anfang des Prozesses aufgerufen werden muss.

Im einem zweiten Schritt wurden verschiedene Möglichkeiten ausgelotet, Eigenschaften des inkrementellen Vorgehens zu nutzen, um Zeitbedarf und Ergebnisqualität weiter zu verbessern. Verschiedene Heuristiken zur Beschneidung des Suchraumes und zum Abbruch der Suche haben sich als geeignet erwiesen, den Zeitbedarf zu senken, sind jedoch mit leichten Einbußen bei der Genauigkeit verbunden. Um die Ergebnis-

qualität zu verbessern, kann die eingesparte Zeit dazu genutzt werden, ausgehend von Analysen mit schlechter Bewertung gezielt weitere Suchläufe zu initiieren. Die Eignung der verschiedenen Heuristiken hängt stark vom Verwendungszweck des inkrementellen Parsers ab, da verschiedene Anforderungen an das Zeitverhalten gestellt werden können.

Das Ziel, in der Zeit bis zum Vorliegen des gesamten Satzes, durch Analyse der unvollständigen Eingabe einen Verarbeitungsvorsprung gegenüber einem nichtinkrementellen Parser aufzubauen, ist bei ausreichend Zeit bis zum Abschluss der Eingabe erreichbar.

Danksagungen

An dieser Stelle möchte ich einige Menschen erwähnen, die mir bei der Erstellung dieser Diplomarbeit geholfen haben.

Danken möchte ich

- meinem Erstbetreuer Prof. Dr. Wolfgang Menzel für die Vergabe des Themas, die aufschlussreichen Diskussionen, seine stets offene Tür und sein offenes Ohr für alle Fragen und Probleme sowie seine stets konstruktive Kritik.
- meinem Zweitbetreuer Prof. Dr. Reinhard Moratz dafür, dass er als Gutachter für meine Arbeit zur Verfügung steht, für seine Unterstützung und für seine unkomplizierte Hilfe bei allen Problemen die ich an ihn herangetragen habe.
- Dr. Kilian Foth für seine stete Bereitschaft, alle meine Fragen zum CDG-System ausführlich zu beantworten. Auch für die Zeit, die er sich für Diskussionen und Beschäftigung mit dem Programmcode genommen hat, möchte ich ihm danken.
- meinem Kommilitonen Martin Burmester für seine Hilfe bei Latex-Problemen und seine vielen formalen und inhaltlichen Anregungen.
- meinem Kommilitonen Arne Köhn für die Diskussionen und seine technische Unterstützung.
- Reinhard Zierke für seine technische Unterstützung.
- meiner Mutter Annette Callies-Beuck für ihre Hilfe beim Korrekturlesen und ihre Unterstützung meines Studiums im Allgemeinen.
- allen Anderen, die mir durch Anmerkungen, Antworten, technische Handgriffe oder moralische Unterstützung behilflich waren.

Danke!

A. Beispieloutput

Nachfolgend eine exemplarische Ausgabe des Parsers für den Satz:
"Die Stadt Frankfurt lehnte es ab , Kosten von '68000' Mark zu übernehmen ."
Dependenzen sind dabei angegeben in der Form:

```
[Domäne] [Ebene]: [Dependent, Lexem] -- [Kantenbeschriftung] --> [Regent, Lexem]
```

Konflikte sind angegeben in der Form:

```
[betroffene Domäne(n)] : [Strafwert] : [verletztes Constraint]
```

```
Incremental step for token: die
   total tokens: 1
   time needed : 50ms
   000        SYN: die_PDS_pl_acc(0-1)--OBJA-->NONSPEC

   Violations:
      000 : 9.500e-01 : NONSPEC-Kante (SYN)


-------------------------------------------
Incremental step for token: Stadt
   total tokens: 2
   time needed : 180ms
   000        SYN: die_ART_sg_nom(0-1)--DET-->Stadt_NN(1-2)
   002        SYN: Stadt_NN(1-2)--SUBJ-->NONSPEC

   Violations:
      002 : 9.500e-01 : NONSPEC-Kante (SYN)
```

```
-------------------------------------------
Incremental step for token: Frankfurt
   total tokens: 3
   time needed : 430ms
   000        SYN: die_ART_sg_nom(0-1)--DET-->Stadt_NN(1-2)
   002        SYN: Stadt_NN(1-2)--SUBJ-->NONSPEC
   004        SYN: Frankfurt_NE(2-3)--APP-->Stadt_NN(1-2)

   Violations:
      002 : 9.500e-01 : NONSPEC-Kante (SYN)


-------------------------------------------
Incremental step for token: lehnte
   total tokens: 4
   time needed : 1030ms
   000        SYN: die_ART_sg_nom(0-1)--DET-->Stadt_NN(1-2)
   002        SYN: Stadt_NN(1-2)--SUBJ-->lehnte_VVFIN_third_past(3-4)
   004        SYN: Frankfurt_NE(2-3)--APP-->Stadt_NN(1-2)
   006        SYN: lehnte_VVFIN_third_past(3-4)--S-->NIL

   Violations:
      006 : 9.000e-01 : Transitivität


-------------------------------------------
Incremental step for token: es
   total tokens: 5
   time needed : 430ms
   000        SYN: die_ART_sg_nom(0-1)--DET-->Stadt_NN(1-2)
   002        SYN: Stadt_NN(1-2)--SUBJ-->lehnte_VVFIN_third_past(3-4)
   004        SYN: Frankfurt_NE(2-3)--APP-->Stadt_NN(1-2)
   006        SYN: lehnte_VVFIN_third_past(3-4)--S-->NIL
   008        SYN: es_PPER_acc(4-5)--OBJA-->lehnte_VVFIN_third_past(3-4)

   Violations:


-------------------------------------------
Incremental step for token: ab
   total tokens: 6
   time needed : 430ms
   000        SYN: die_ART_sg_nom(0-1)--DET-->Stadt_NN(1-2)
```

```
   002        SYN: Stadt_NN(1-2)--SUBJ-->lehnte_VVFIN_third_past(3-4)
   004        SYN: Frankfurt_NE(2-3)--APP-->Stadt_NN(1-2)
   006        SYN: lehnte_VVFIN_third_past(3-4)--S-->NIL
   008        SYN: es_PPER_acc(4-5)--OBJA-->lehnte_VVFIN_third_past(3-4)
   010        SYN: ab_PTKVZ(5-6)--AVZ-->lehnte_VVFIN_third_past(3-4)

   Violations:


-------------------------------------------
Incremental step for token: ,
   total tokens: 7
   time needed : 210ms
   000        SYN: die_ART_sg_nom(0-1)--DET-->Stadt_NN(1-2)
   002        SYN: Stadt_NN(1-2)--SUBJ-->lehnte_VVFIN_third_past(3-4)
   004        SYN: Frankfurt_NE(2-3)--APP-->Stadt_NN(1-2)
   006        SYN: lehnte_VVFIN_third_past(3-4)--S-->NIL
   008        SYN: es_PPER_acc(4-5)--OBJA-->lehnte_VVFIN_third_past(3-4)
   010        SYN: ab_PTKVZ(5-6)--AVZ-->lehnte_VVFIN_third_past(3-4)

   Violations:


-------------------------------------------
Incremental step for token: Kosten
   total tokens: 8
   time needed : 4910ms
   000        SYN: die_ART_sg_nom(0-1)--DET-->Stadt_NN(1-2)
   002        SYN: Stadt_NN(1-2)--SUBJ-->lehnte_VVFIN_third_past(3-4)
   004        SYN: Frankfurt_NE(2-3)--APP-->Stadt_NN(1-2)
   006        SYN: lehnte_VVFIN_third_past(3-4)--S-->NIL
   008        SYN: es_PPER_acc(4-5)--OBJA-->lehnte_VVFIN_third_past(3-4)
   010        SYN: ab_PTKVZ(5-6)--AVZ-->lehnte_VVFIN_third_past(3-4)
   014        SYN: Kosten_NN(7-8)--SUBJ-->NONSPEC

   Violations:
      014 : 9.500e-01 : NONSPEC-Kante (SYN)


-------------------------------------------
Incremental step for token: von
   total tokens: 9
   time needed : 11340ms
   000        SYN: die_ART_sg_nom(0-1)--DET-->Stadt_NN(1-2)
   002        SYN: Stadt_NN(1-2)--SUBJ-->lehnte_VVFIN_third_past(3-4)
```

```
   004        SYN: Frankfurt_NE(2-3)--APP-->Stadt_NN(1-2)
   006        SYN: lehnte_VVFIN_third_past(3-4)--S-->NIL
   008        SYN: es_PPER_acc(4-5)--OBJA-->lehnte_VVFIN_third_past(3-4)
   010        SYN: ab_PTKVZ(5-6)--AVZ-->lehnte_VVFIN_third_past(3-4)
   014        SYN: Kosten_NN(7-8)--SUBJ-->NONSPEC
   016        SYN: von_APPR(8-9)--PP-->Kosten_NN(7-8)

   Violations:
      016 : 1.000e-02 : PN fehlt
      014 : 9.500e-01 : NONSPEC-Kante (SYN)


-------------------------------------------
Incremental step for token: 68000
   total tokens: 10
   time needed : 5920ms
   000        SYN: die_ART_sg_nom(0-1)--DET-->Stadt_NN(1-2)
   002        SYN: Stadt_NN(1-2)--SUBJ-->lehnte_VVFIN_third_past(3-4)
   004        SYN: Frankfurt_NE(2-3)--APP-->Stadt_NN(1-2)
   006        SYN: lehnte_VVFIN_third_past(3-4)--S-->NIL
   008        SYN: es_PPER_acc(4-5)--OBJA-->lehnte_VVFIN_third_past(3-4)
   010        SYN: ab_PTKVZ(5-6)--AVZ-->lehnte_VVFIN_third_past(3-4)
   014        SYN: Kosten_NN(7-8)--SUBJ-->NONSPEC
   016        SYN: von_APPR(8-9)--PP-->Kosten_NN(7-8)
   018        SYN: 68000_CARD(9-10)--PN-->von_APPR(8-9)

   Violations:
      014 : 9.500e-01 : NONSPEC-Kante (SYN)


-------------------------------------------
Incremental step for token: Mark
   total tokens: 11
   time needed : 9070ms
   000        SYN: die_ART_sg_nom(0-1)--DET-->Stadt_NN(1-2)
   002        SYN: Stadt_NN(1-2)--SUBJ-->lehnte_VVFIN_third_past(3-4)
   004        SYN: Frankfurt_NE(2-3)--APP-->Stadt_NN(1-2)
   006        SYN: lehnte_VVFIN_third_past(3-4)--S-->NIL
   008        SYN: es_PPER_acc(4-5)--OBJA-->lehnte_VVFIN_third_past(3-4)
   010        SYN: ab_PTKVZ(5-6)--AVZ-->lehnte_VVFIN_third_past(3-4)
   014        SYN: Kosten_NN(7-8)--SUBJ-->NONSPEC
   016        SYN: von_APPR(8-9)--PP-->Kosten_NN(7-8)
   018        SYN: 68000_CARD(9-10)--ATTR-->Mark_NN_pl(10-11)
   020        SYN: Mark_NN_pl(10-11)--PN-->von_APPR(8-9)
```

```
   Violations:
      014 : 9.500e-01 : NONSPEC-Kante (SYN)


-----------------------------------------------
Incremental step for token: zu
   total tokens: 12
   time needed : 28680ms
   000         SYN: die_ART_sg_nom(0-1)--DET-->Stadt_NN(1-2)
   002         SYN: Stadt_NN(1-2)--SUBJ-->lehnte_VVFIN_third_past(3-4)
   004         SYN: Frankfurt_NE(2-3)--APP-->Stadt_NN(1-2)
   006         SYN: lehnte_VVFIN_third_past(3-4)--S-->NIL
   008         SYN: es_PPER_acc(4-5)--OBJA-->lehnte_VVFIN_third_past(3-4)
   010         SYN: ab_PTKVZ(5-6)--AVZ-->lehnte_VVFIN_third_past(3-4)
   014         SYN: Kosten_NN(7-8)--SUBJ-->NONSPEC
   016         SYN: von_APPR(8-9)--PP-->Kosten_NN(7-8)
   018         SYN: 68000_CARD(9-10)--ATTR-->Mark_NN_pl(10-11)
   020         SYN: Mark_NN_pl(10-11)--PN-->von_APPR(8-9)
   022         SYN: zu_PTKVZ(11-12)--PART-->NONSPEC

   Violations:
      022 : 3.386e-01 : tagger
      014 : 9.500e-01 : NONSPEC-Kante (SYN)
      022 : 9.500e-01 : NONSPEC-Kante (SYN)


-----------------------------------------------
Incremental step for token: übernehmen
   total tokens: 13
   time needed : 3230ms
   000         SYN: die_ART_sg_nom(0-1)--DET-->Stadt_NN(1-2)
   002         SYN: Stadt_NN(1-2)--SUBJ-->lehnte_VVFIN_third_past(3-4)
   004         SYN: Frankfurt_NE(2-3)--APP-->Stadt_NN(1-2)
   006         SYN: lehnte_VVFIN_third_past(3-4)--S-->NIL
   008         SYN: es_PPER_nom(4-5)--EXPL-->lehnte_VVFIN_third_past(3-4)
   010         SYN: ab_PTKVZ(5-6)--AVZ-->lehnte_VVFIN_third_past(3-4)
   014         SYN: Kosten_NN(7-8)--OBJA-->übernehmen_VVINF(12-13)
   016         SYN: von_APPR(8-9)--PP-->Kosten_NN(7-8)
   018         SYN: 68000_CARD(9-10)--ATTR-->Mark_NN_pl(10-11)
   020         SYN: Mark_NN_pl(10-11)--PN-->von_APPR(8-9)
   022         SYN: zu_PTKZU(11-12)--PART-->übernehmen_VVINF(12-13)
   024         SYN: übernehmen_VVINF(12-13)--OBJI-->lehnte_VVFIN_third_past(3-4)

   Violations:
```

```
-------------------------------------------
Incremental step for token: .
   total tokens: 14
   time needed : 510ms
   000         SYN: die_ART_sg_nom(0-1)--DET-->Stadt_NN(1-2)
   002         SYN: Stadt_NN(1-2)--SUBJ-->lehnte_VVFIN_third_past(3-4)
   004         SYN: Frankfurt_NE(2-3)--APP-->Stadt_NN(1-2)
   006         SYN: lehnte_VVFIN_third_past(3-4)--S-->NIL
   008         SYN: es_PPER_nom(4-5)--EXPL-->lehnte_VVFIN_third_past(3-4)
   010         SYN: ab_PTKVZ(5-6)--AVZ-->lehnte_VVFIN_third_past(3-4)
   014         SYN: Kosten_NN(7-8)--OBJA-->übernehmen_VVINF(12-13)
   016         SYN: von_APPR(8-9)--PP-->Kosten_NN(7-8)
   018         SYN: 68000_CARD(9-10)--ATTR-->Mark_NN_pl(10-11)
   020         SYN: Mark_NN_pl(10-11)--PN-->von_APPR(8-9)
   022         SYN: zu_PTKZU(11-12)--PART-->übernehmen_VVINF(12-13)
   024         SYN: übernehmen_VVINF(12-13)--OBJI-->lehnte_VVFIN_third_past(3-4)

   Violations:
```

Literaturverzeichnis

Thorsten Brants. Tnt - a statistical part-of-speech tagger. In *In Proceedings of the Sixth Applied Natural Language Processing Conference ANLP-2000*, 2000.

cdg. Constraint-Dependency-Grammar. URL `http://nats-www.informatik.uni-hamburg.de/view/CDG/WebHome`.

Kilian A. Foth. Disjunktive lexikoninformation im eliminativen parsing. Studienarbeit, 1999.

Kilian A. Foth. *Hybrid Methods of Natural Language Analysis*. PhD thesis, Universität Hamburg, Department Informatik, 2007.

Daniel Jurafsky. *Speech And Language Processing : An Introduction to Natural Language Processing , Computational Linguistics, and Speech Recognition*. Prentice Hall, 2008.

Hiroshi Maruyama. Structural disambiguation with constraint propagation. In *Proceedings of the 28th annual meeting on Association for Computational Linguistics*, 1990.

Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 91–98, Morristown, NJ, USA, 2005. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1219840.1219852.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 216–220, 2006.

Wolfgang Menzel and Ingo Schröder. Decision procedures for dependency parsing using graded constraints. In *Proceedings of the Workshop on Processing of Dependency-Based Grammars*, 1998.

negra. Negra-Korpus. URL `http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/negra-corpus.html`.

Joakim Nivre. Dependency grammar and dependency parsing. Technical report, Växjö University: School of Mathematics and Systems Engineering, 2005. URL `http://stp.lingfil.uu.se/~nivre/docs/05133.pdf`.

Joakim Nivre. *Inductive Dependency Parsing*. Springer, 2006.

Joakim Nivre. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553, 2008.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryigit, and Svetoslav Marinov. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225, 2006.

Adrian Staub and Keith Rayner. Effects of syntactic processing on eye movements. In *Oxford Handbook of Psycholinguistics*. 2007.

Roger P. G. van Gompel and Martin J. Pickering. Syntactic parsing. In *Oxford Handbook of Psycholinguistics*. 2007.

Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Ich bin mit einer Einstellung in den Bestand der Bibliothek des Departments Informatik einverstanden.

Niels Beuck, Hamburg d. 22.5.09