Disjunktive Lexikoninformation im eliminativen Parsing

Kilian A. Foth

18. März 1999

Kapitel 1

Einleitung

1.1 Überblick

Diese Arbeit behandelt ein sehr spezielles Teilthema des maschinellen Sprachanalyse. Diese **Einleitung** erläutert daher zunächst die im Titel verwendeten Fachbegriffe, bevor in Abschnitt 1.6 des konkrete Ziel der Arbeit vorgestellt wird. Der **allgemeine Teil** führt in die Formalismen ein, die im vorliegenden System verwendet werden. Der **spezielle Teil** erläutert die Ausgangslage und die Lösung der speziellen Aufgabe.

1.2 Was bedeutet "Parsing"?

Obwohl der Mensch das Vorbild aller Systeme zur Verarbeitung natürlicher Sprache ist, sind seine Fähigkeiten in vielen Punkten bis heute unerreicht: Eingaben in natürlicher Sprache können von Maschinen entweder nur oberflächlich oder nur in sehr kleinen Teilgebieten sinngemäß verarbeitet werden. Aus diesem Grund ist es zumeist nicht angemessen, von einem wirklichen Verständnis der Eingabe zu sprechen. Ein weniger ambitioniertes Ziel wird durch den Begriff des **Parsing** gefaßt.

Als Parsing bezeichnet man im engeren Sinne die Analyse von Eingaben in natürlicher Sprache mit dem Ziel, aus der zeitlichen Struktur einer Äußerung (Sequenz oder Graph von Wortformen) die zugrundeliegende sprachliche Struktur abzuleiten. Ein solche Analyse muß etwa erkennen, daß in dem Satz

(1) Hängt den König auf!

die Wörter "hängt" und "auf" eine Einheit bilden und folglich in engerer Beziehung stehen als die Wörter "hängt" und "den", obwohl diese einander im Satz näher stehen. Die Struktur einer ganzen Äußerung läßt sich sehr gut durch eine Menge von Beziehungen zwischen ihren Elementen darstellen. Sollen diese Beziehungen maschinell gefunden werden, so erkennt man recht schnell Regeln, die verschiedene Äußerungen in einheitlicher Weise erklären. So würde man etwa in dem Satz

(2) Störche fressen Frösche.

dieselben Beziehungen zwischen den einzelnen Wörtern annehmen wie in dem Satz

(3) Frösche fressen Fliegen.

und noch unzähligen anderen Aussagen dieser Art. Schon mit sehr wenigen Regeln (und einer Liste von Tierarten) kann also ein Parser eine sehr große Zahl von einfachen Sätzen analysieren, und allgemein wird angenommen, daß die dazu verwendete Regel tatsächlich auch einen kleinen Teil der menschlichen Sprachfähigkeit darstellt. Das Prinzip, daß durch eine endliche Anzahl von Regeln komplexe Strukturen aufgebaut werden können, deren Bedeutung sich aus ihrem Aufbau ergibt, wird als *Kompositionalität* bezeichnet. Nur durch dieses Prinzip ist es möglich, eine prinzipiell unbegrenzte Anzahl von Äußerungen zu produzieren und zu verstehen.

Mit der Einführung vieler grammatischer Regeln und sehr vieler Wörter scheint sich das Problem zunächst nicht grundlegend zu verändern. So kann auch ein realistischer Satz wie etwa

(4) Der ehemalige Rebellenführer hat drei Tage nach seiner Machtergreifung in einem unblutigen Putsch die Bildung sieben neuer Bundesstaaten angekündigt.

prinzipiell durch einen Parser analysiert werden, wenn er auf alle benötigten grammatischen Regeln und alle verwendeten Wörter zugreifen kann. Sehr schnell zeigt sich jedoch, daß die grammatischen Regeln oft mehrere Analysen zulassen. Sie erlauben es zum Beispiel nicht, zu entscheiden, ob der Referent von "seiner" der Rebellenführer oder der Putsch ist und ob sich die Präpositionalphrase "in einem unblutigen Putsch" auf das Hauptverb des Satzes bezieht oder auf die vorangehende Präpositionalphrase: Die Äußerung erweist sich als mehrdeutig.

Mehrdeutigkeit entsteht nicht etwa durch außergewöhnliche Umstände beim Zusammenwirken vieler Regeln; vielmehr ist sie der Regelfall selbst in sehr kurzen Äußerungen. Schon der Satz

(5) Frösche fressen sie.

wird ganz verschieden interpretiert werden, je nachdem, welcher dieser beiden Kontexte ihm vorausgeht:

(6) Fressen Störche Gras? — Nein.

(7) Fliegen haben ein schweres Leben. Kühe verjagen sie. Menschen erschlagen sie.

Der Diskurskontext legt hier strukturell verschiedene Deutungen derselben Äußerung nahe. Um realistische Texte zu verarbeiten, muß ein System also nicht nur Wissen über grammatische Regeln besitzen, sondern auch Wissen über den Geprächsgegenstand. Erschwerend kommt hinzu, daß dies Wissen nicht immer aus dem Kontext ermittelbar ist, sondern oft stillschweigend vorausgesetzt wird. So kann ein Zeitungsleser Äußerung (4) nur deshalb verstehen, weil ein Militärputsch üblicherweise der Machtergreifung dient und nicht nur der Ankündigung von Verwaltungsreformen.

Die Repräsentation solchen *semantischen* (inhaltlichen) Wissens zusätzlich zum *syntaktischen* (strukturellen) Wissen ist für maschinelle Systeme außerordentlich schwierig. Zum einen herrscht in der Linguistik kein Einverständnis über die geeignete Art, inhaltliche Sachverhalte zu repräsentieren. Zum anderen ist die Menge inhaltlicher Sachverhalte um Größenordnungen mächtiger als die der grammatischen Regeln: Während grammatische Regeln oft nur die Kategorie von Wörtern betrachten und daher gleichbedeutend auf sehr viele Wörter zutreffen, kann es prinzipiell ebenso viele verschiedene Konzepte wie Wörter geben—mehr noch, denn auch Kombinationen mehrerer Wörter können spezifische Konzepte bezeichnen. Darüberhinaus kann sich die Menge der inhaltlichen Konzepte viel schneller verändern als die Menge der grammatischen Regeln. Insbesondere kann ein Sprecher neue Konzepte selbst einführen und sofort danach verwenden. Ein System, daß verläßlich realistische Texte verstehen könnte, müßte also sowohl einen umfassenden Schatz an Weltwissen als auch eine äußerst flexible Lernfähigkeit besitzen. Beide Ziele sind bislang unerreicht.

Syntaktische Verfahren bilden daher nach wie vor den Schwerpunkt vieler Sprachverarbeitungssysteme. Tatsächlich wird der Begriff Parsing oft gleichbedeutend mit "Verfahren zur Sprachverarbeitung" verwendet. Eine adäquate inhaltliche Repräsentation von Sprachdaten ist bislang nur im Einzelfall möglich.

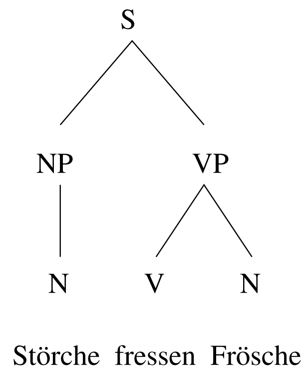
1.3 Was bedeutet "eliminativ"?

Der überwiegende Teil der derzeit eingesetzten Verfahren in der Sprachverarbeitung ist *konstruktiver* Natur. Die wesentliche Eigenschaft eines konstruktiven Verfahrens ist es, daß im Verlauf der Problemlösung zusätzlich zu den Eingabedaten neue Strukturen aufgebaut werden, aus denen schließlich die Lösung berechnet wird. Das verbreitete *divide-et-impera*-Verfahren ist hierfür ein gutes Beispiel: Das Ausgangsproblem wird in kleinere Teilprobleme unterteilt, die einfacher zu lösen sind. Jeder Subprozeß liefert eine Teil-Lösung zurück, aus denen die Gesamtlösung zusammengesetzt wird.

Beim **eliminativen** Vorgehen dagegen werden im Verlauf der Abarbeitung keine neuen Objekte in den Problemraum eingeführt. Stattdessen betrachtet man systematisch alle Funktionen, die ein Teil der Eingabe ausfüllen könnte, und schließt solche Funktionen aus, die zu den gegebenen Bedingungen im Widerspruch stehen. Das Problem ist gelöst, wenn eine Teilmenge der betrachteten Möglichkeiten verbleibt, die den Bedingungen genügt, die an eine Lösung gestellt werden. Die Lösung muß also unmittelbar aus den Elementen der Eingabe zusammengesetzt werden.

Die beiden Methoden seien hier am Beispiel von Satz (2) gegenübergestellt:

a)



b)

Störche: Subjekt, ~~Objekt,~~ ~~Attribut~~

fressen: ~~Infinitiv,~~ finites Verb

Frösche: ~~Subjekt,~~ Objekt, ~~Attribut~~

Das konstruktive Verfahren (a) betrachtet den Satz als Endprodukt einer generativen Grammatik. Es klassifiziert zunächst die gegebenen Wörter nach ihrer syntaktischen Funktion. Aus den gefundenen Kategorien werden eine Hauptwortgruppe und eine Verbgruppe gebildet, die dann zum Satz zusammengefaßt werden. Das Ergebnis ist eine Baumstruktur. Das Verfahren ist charakterisiert durch die Produktionen, die es verwendet: hier etwa die kontextfreie Regel S $\rightarrow$ NP VP.

Das eliminative Verfahren (b) stellt zunächst die Listen aller Funktionen zusammen, die jedes Wort ausüben könnte. Danach werden einzelne Elemente der Listen gestrichen, bis jeweils eine Funktion übrig bleibt. Das Ergebnis ist eine kürzere Liste von Funktionen. Das Verfahren ist charakterisiert durch die Verträglichkeitsregeln, die es verwendet: hier etwa die Regel "Es kann nicht zwei Subjekte geben".

1.4 Was bedeutet "Lexikon"?

Als **Lexikon** kann im weiteren Sinne jede Liste bezeichnet werden, die systematisch Information mit den Wörtern einer Sprache assoziiert. Auch die meisten sprachverarbeitenden Systeme besitzen eine Art von Lexikon, also eine Komponente, die die Funktionalität eines Nachschlagewerkes bietet. Insbesondere wird für den Menschen ein mentales Lexikon postuliert. Dessen genauer Aufbau ist nicht bekannt; ein wesentlicher Unterschied zu maschinellen Speichern ist die *assoziative* Zugriffsweise: Auf ein Wort kann sowohl anhand der Form als auch durch inhaltliche Konzepte zugegriffen werden. So kann ein Sprecher ohne Probleme Wörter aufzählen, die einem genannten Begriff inhaltlich nahestehen, aber auch solche, die sich auf ein gegebenes Wort reimen. Dagegen wäre es sehr aufwendig, einen Abschnitt des mentalen Lexikons alphabetisch aufzuzählen.

Nicht schlüssig geklärt ist die Frage, ob das mentale Lexikon nur Grundformen ("Storch") repräsentiert oder Vollformen ("Storch", "Storches", "Störche"). Insbesondere in stark flektierenden oder agglutinierenden Sprachen erscheint es wahrscheinlicher, daß nur Grundformen gelernt werden und für die Interpretation von Vollformen allgemeine Regeln zuständig sind. Oft wird der in [Kempen,Huijbers83] aufgestellten Theorie gefolgt, daß Einträge des mentalen Lexikons unterteilt sind in einen Eintrag, der inhaltliche und syntaktische Information enthält (*Lemma*), und eine Menge von Wortformen. Ungeklärt ist, ob auf Lemma und Formeintrag gleichzeitig zugegriffen wird oder nicht. Ein Gegenargument ist das Zungenspitzen-Phänomen, bei dem die Form des gewünschten Wortes sekundenlang nicht verfügbar ist, wohl aber seine Bedeutung und Syntax.

Ein Lexikon von Grundformen oder *lexikalischen Wörtern* ist kognitiv angemessen und effizient, weil die mit den Formen assoziierte Information nicht mehrfach gespeichert werden muß. Andererseits muß der Benutzer des Lexikons die Flektionsregeln der Arbeitssprache beherrschen, um die gespeicherte Information zu erhalten. Viele Parsingverfahren sind dazu nicht ohne weiteres in der Lage. Ein Lexikon aus Vollformen oder *phonologischen Wörtern* aufzustellen ist aufwendig und fehleranfällig, weil jede mögliche Form aufgeführt werden muß. Dafür kann es auch von solchen Parsern verwendet werden, die gar kein morphologisches Wissen besitzen. Insbesondere Erkenner von gesprochener Sprache sind gegenwärtig noch auf die Deklaration jeder einzelnen Form angewiesen.

1.5 Was bedeutet "disjunktiv"?

Auch ein Vollformenlexikon kann denselben Sprachausschnitt auf verschiedene Weise darstellen. Im Beispielsatz (2) mußten beide Verfahren die mehrdeutige Form "fressen" verarbeiten, die entweder Infinitiv oder finiter Plural sein kann. Die Pluralform kann wiederum in der ersten oder der dritten Person stehen. Mehr noch als bei orthographisch verschiedenen Formen liegt es nahe, diese Varianten als Ausprägungen nur eines lexikalischen Wortes anzusehen und durch nur einen Lexikoneintrag zu repräsentieren. Da aber verschiedene

Lesarten einer Form einander ausschließen, muß der entstehende Eintrag **disjunktiv** aufgebaut sein, also gleichwertige Alternativen anbieten. Zum Eintrag “fressen” muß etwa folgende Information dargestellt werden:

“Entweder liegt ein Infinitiv vor, oder es liegt eine Präsensform vor, die im Plural und in der ersten oder dritten Person steht.”

Wieder liegt der Vorteil dieser Lösung darin, daß die Information, die allen Varianten gemeinsam ist, nicht verdoppelt werden muß. Dafür wird aber dem Parsingverfahren der Aufwand aufgebürdet, aus der gegebenen Disjunktion die Menge der möglichen Formen zu expandieren. Gewöhnlich ist dazu eine spezielle Anpassung des Parsers notwendig.

1.6 Ziel der Arbeit

Diese Arbeit beschreibt ein Verfahren, durch Verwendung disjunktiver Information die Effizienz eines eliminativen Parsingverfahrens zu erhöhen. Gegenstand ist ein bereits existierendes prototypisches System, das natürliche Sprache durch *Constraint-Netzwerke* (vgl. Abschnitt 2.4) analysiert.

Bislang verwendete das System ein reines Wortvollformenlexikon, in dem jede Variante einer mehrdeutigen Form einen eigenen Lexikoneintrag benötigt. Der eliminative Ansatz berücksichtigt dabei zunächst alle möglichen Ausprägungen einer Form. Damit führt die Verwaltung homonymer Varianten zu beträchtlichem Mehraufwand. Da das Ziel der Analyse die Auffindung von Beziehungen der Formen untereinander ist, hängen Zeit- und Platzkomplexität des Verfahrens polynomiell von der mittleren Mehrdeutigkeit im Lexikon ab.

Das System ist dahingehend zu verändern, daß alle grammatischen Varianten in disjunktiven Lexikoneinträgen repräsentiert werden. Das System soll in die Lage versetzt werden, eine mehrdeutige Wortform genau dann zu expandieren, wenn es für den aktuellen Verarbeitungsschritt notwendig ist, und sonst mit unterspezifizierten Werten zu arbeiten. Wenn die unterschiedlichen Merkmale nicht immer relevant sind, kann damit in vielen Fällen die mehrfache Auswertung äquivalenter Daten verhindert werden. Durch die Einführung eines disjunktiven Lexikons könnte also die Effizienz des Systems merklich gesteigert werden. Das sonstige Verhalten des Systems soll gänzlich unverändert bleiben.

Kapitel 2

Allgemeiner Teil

2.1 Robuste Verarbeitung natürlicher Sprache

2.1.1 Ein enttäuschendes Ergebnis

Damit ein Verfahren zur Lösung eines Problemes von einem maschinellen Prozessor ausgeführt werden kann, muß es durch strikte Regeln beschrieben werden können. Vergleicht man die Performanz verschiedener Sprecher derselben Sprache, so stellt man fest, daß eine große Zahl von Regeln existiert, die jeder von ihnen befolgt. Dieses Regelsystem ist so komplex, daß offenbar nur der Mensch in der Lage ist, es zu erlernen. Selbst beim Menschen wird verschiedentlich ein spezieller Instinkt oder ein mentales Organ postuliert, um den Spracherwerb zu erklären ([Pinker96]).

Da programmierbare Rechenanlagen sehr gut geeignet sind, einer großen Zahl von Regeln zu gehorchen, wären sie also ideale Sprachverarbeiter. Tatsächlich ist die maschinelle Sprachverarbeitung aber der menschlichen noch weit unterlegen. Ein Grund dafür ist das schon erwähnte Phänomen der Mehrdeutigkeit: Maschinen können zwar in großer Zahl mögliche Analysen produzieren, aber nur sehr schlecht die angemessenste davon auswählen. Wird die Datenbasis zu groß, so ist nicht einmal mehr das Aufzählen aller Analysen möglich. Offenbar verwendet der Mensch neben den allgemeinen Regeln seiner Sprache weiteres Wissen, um die Analyse systematisch zu beschleunigen.

Umgekehrt kann der Mensch aber auch Regeln ignorieren, wenn sie eine Analyse zu verbieten drohen. Im Deutschen gilt etwa die Regel, daß ganze Sätze nach Satztypen klassifiziert werden. Es gibt dabei den *Indikativ*, den *Interrogativ* und den *Imperativ*, die durch die Wortstellung und die Sprachmelodie markiert werden. Die allgemeine Regel ist, daß der Indikativ für Aussagen verwendet wird, der Interrogativ für Fragen und der Imperativ für Aufforderungen:

(8) Christian hofft, nicht erwischt zu werden. (Aussage)

(9) Kommst du auch mit? (ja/nein-Frage)

(10) Wo soll ich denn das hinstellen? (allgemeine Frage)

(11) Folgen Sie mir! (Aufforderung)

In geeigneter Umgebung können aber dieselben grammatischen Strukturen auch andere Absichten ausdrücken:

(12) Der Bischof wünscht nicht gestört zu werden. (Aufforderung)

(13) Hörst du wohl auf? (Aufforderung)

(14) Was soll ich denn davon halten? (Aussage)

(15) Identifizieren Sie sich! (allgemeine Frage)

In jedem dieser Fälle wird die Regel über die Verwendung der Satzarten verletzt, weil inhaltliche Überlegungen Vorrang haben. Es müssen also nicht alle Regeln erfüllt sein, damit eine Äußerung verständlich ist.

Das verbreitete Stilmittel der *Ironie* verletzt sogar das Prinzip der kompositionalen Bedeutung:

(16) Das kann ja heiter werden.

Hierbei wird die Gesamtbedeutung zwar aus den Einzelbedeutungen gebildet, aber ausgedrückt wird stattdessen die umgekehrte Bedeutung. (Nicht zufällig werden ironische Äußerungen oft auch von Menschen falsch analysiert.) Die Liste der Beispiele kann fast unbegrenzt verlängert werden. All diesen Phänomenen ist gemeinsam, daß eine allgemeine Interpretationsregel gebrochen werden kann, ohne daß das Verständnis darunter leidet, solange nicht zu viele Regeln zugleich gebrochen werden.

2.1.2 Versuch einer Erklärung

Diese Art fehlertoleranter Analyse maschinell nachzubilden, erweist sich als äußerst schwierig. Es genügt natürlich nicht, aus einem komplexen Regelwerk alle verletzbaren Regeln zu entfernen, denn in den meisten Fällen tragen sie ja durchaus zur richtigen Analyse bei. Vielmehr muß der Formalismus immer, wenn sich zwei Regeln widersprechen, eine Entscheidung zwischen ihnen treffen. Welche von zwei Regeln die "wichtigere" ist, läßt sich aber nicht im voraus festlegen. So ist es beispielsweise durchaus nicht immer der Fall, daß inhaltliche Regeln den Vorrang vor sprachlichen Regeln erhalten sollen; selbst für die Verarbeitung völlig regelgerechter Sprache ist umstritten, ob inhaltliches Wissen nur zur Disambiguierung eingesetzt wird oder ob es umgekehrt die Analyse steuert und die Syntax nur als "letztes Mittel" zur Anwendung kommt ([Herrmann90]).

Die besondere *Robustheit* der menschlichen Sprachverarbeitung beruht auf der Fähigkeit, verletzbare Regeln genau dann zu ignorieren, wenn es notwendig ist. Insbesondere drei eng zusammenhängende Eigenschaften der Kognition begünstigen diese Fähigkeit:

- *Parallelität:* Der Mensch nutzt parallele Informationsverarbeitung nicht nur zur Erhöhung der Gesamtleistung. Vielmehr hilft die nebenläufige Analyse dabei, Regelverletzungen auf einer bestimmten Sprachebene durch Regelerfüllungen einer anderen Ebene zu kompensieren.

- *Inkrementalität:* Ein menschlicher Hörer beginnt mit der Analyse einer Äußerung, sobald sie ihm als solche bewußt wird, und führt sie ziemlich genau so lange fort, wie das Sprachsignal andauert. Große Teile der Analyse werden dabei fertiggestellt, bevor ein vollständiger Satz erkannt wurde. Damit können sowohl bruchstückhafte als auch potentiell unendliche Äußerungen analysiert werden. Auch die Produktion von Sprache verläuft auf vielen Ebenen inkrementell ([Levelt89]).

- *Adaptivität:* Der Mensch kann dem Sprachsignal mehr oder weniger Aufmerksamkeit schenken und dadurch besseres Verständnis gegen geringere Beanspruchung abwägen. So können beim normalen Sprachverstehen gleichzeitig andere kognitive Leistungen erbracht werden (wie etwa Dolmetschen). Schwierige Konstruktionen oder undeutliche Aussprache ziehen zusätzliche Aufmerksamkeit auf sich und können im Extremfall jede andere Aktivität verhindern.

2.1.3 Aussichten für die maschinelle Verarbeitung

Die Parallelisierbarkeit eines Algorithmus ist das Ausmaß, in dem die Verarbeitungszeit gesenkt werden kann, wenn mehrere Prozessoren verwendet werden. Bei vielen wichtigen Algorithmen ist es nicht möglich, durch die Verwendung von n Prozessoren eine n-fache Beschleunigung zu erreichen, weil zwischen vielen Verarbeitungsschritten eine strikte Ordnung eingehalten werden muß. Auch wenn ein Problem prinzipiell nebenläufige Verarbeitung zuläßt, ist die Programmierung eines parallelen Algorithmus oft sehr schwierig ([Hayes88]). Nebenläufiges Parsing setzt also eine Formalisierung voraus, die nicht von vornherein aufeinander aufbauende Verarbeitungsschritte verlangt.

Auch der inkrementelle Charakter menschlicher Sprachverarbeitung kann von maschinellen Systemen nur zum Teil nachgebildet werden. So kann etwa ein System, das natürliche Sprache als Produkt einer generativen Grammatik betrachtet, unvollständige Äußerungen nur dann analysieren, wenn sie gerade einem Teilergebnis der verwendeten Grammatik entsprechen. Der Mensch kann dagegen die meisten Teiläußerungen im Geiste ergänzen, selbst wenn sie mitten im Wort abbrechen. Auch hier muß also ein Formalismus gefunden werden, der zwar beliebig lange Konstruktionen erlaubt, deshalb aber nicht blockiert wird, bis die Eingabe vollständig ist. Auch sind viele Systeme nicht in der Lage, selbständig die Grenzen zwischen verschiedenen Sätzen zu erkennen; für den Menschen ist diese Entscheidung untrennbarer Bestandteil des Verstehensprozesses.

Die Adaptivitäts-Eigenschaft schließlich läßt sich durch den in [Russel,Zilberstein91] formulierten Begriff der *anytime*-Eigenschaft formalisieren. Ein Algorithmus hat diese Eigenschaft, wenn seine Laufzeit variierbar ist und die Qualität des Ergebnisses monoton mit der aufgewandten Zeit wächst. Zwei Spielarten von anytime-Verhalten werden unterschieden:

1. *unterbrechbare Algorithmen:* Der Algorithmus muß jederzeit damit rechnen, unterbrochen zu werden, und dennoch eine Lösung liefern können.
2. *Kontrakt-Algorithmen:* Dem Algorithmus wird eine vom Benutzer bestimmte Zeit für die Lösung des Problemes zur Verfügung gestellt. Erst nach Ablauf dieser Zeit wird eine Lösung erwartet.

Die Anwendung von anytime-Algorithmen ist immer dann sinnvoll, wenn angenommen werden muß, daß der Nutzen einer Problemlösung um so geringer ist, je später sie verfügbar ist. In diesem Fall muß der aufgewendete Arbeitsaufwand gegen den zu erwartenden Nutzen abgeschätzt werden.

[Menzel94] wendet die Definition auf verbreitete Parsingverfahren an und stellt zwei grundsätzliche Defizite fest:

1. Die Qualität der Lösung steigt gewöhnlich nicht gleichmäßig an, sondern in diskreten Schritten (oft nur einem).

2. Bei solchen Verfahren, die tatsächlich mehr oder weniger disambiguierte Lösungen erzeugen können, ist der Zusammenhang zwischen Qualität der Lösung und Zeitbedarf nicht im voraus abschätzbar.

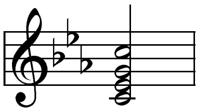
Das Profil der Relation zwischen Zeit und Qualität ist also nicht einheitlich. Ein solcher Algorithmus wird als *schwacher* anytime-Algorithmus bezeichnet. Ein *starker* anytime-Algorithmus muß dagegen ein Leistungsprofil aufweisen, das nicht nur monoton, sondern auch annähernd vorhersagbar ist. Für Algorithmen mit schwacher unterbrechbarer anytime-Eigenschaft erscheint das Modell des *constraint satisfaction* als geeignet.

2.2 Constraint-Satisfaction-Probleme

Sehr viele Probleme der Informatik und anderer Gebiete lassen sich als *constraint satisfaction problem (CSP)* formulieren. Das allgemeine CSP ist wie folgt definiert: Gegeben n Variablen V_i mit jeweils endlichem[1] Wertebereich D_i sowie eine Menge von *constraints* (Bedingungen) C_j an die Wertebelegung, ist eine Belegung $B \in D_1 \times \cdots \times D_n$ zu finden, die alle Bedingungen erfüllt.

2.2.1 Ein Minimalbeispiel

Ein einführendes Beispiel eines CSP: Auf einer Tastatur soll mit der rechten Hand der Akkord



angeschlagen werden. Dazu ist der Fingersatz zu bestimmen, also die Zuordnung von Tasten zu Fingern. Wir wählen als Variable die vier Tasten, die wir mnemonisch mit C, e, g und c bezeichnen wollen. Jede von ihnen hat den Wertebereich $\{1, 2, 3, 4, 5\}$, wobei die 1 für den Daumen steht, die 2 für den Zeigefinger etc. Der gesamte Griff muß nun verschiedene Bedingungen erfüllen:[2]

1. Der Daumen soll nicht auf Obertasten gesetzt werden.
2. Ein Finger kann nur eine Taste anschlagen.
3. Von zwei verwendeten Fingern muß der weiter rechts liegende die höhere Taste anschlagen.
4. Jeweils zwei Finger besitzen eine natürliche Griffweite und können nicht gleichzeitig zwei Tasten anschlagen, deren Abstand von dieser Griffweite wesentlich abweicht.

[1] Auch Probleme mit unendlichen, etwa numerischen, Domänen kommen vor. Diese sind jedoch streng genommen Gegenstand einer verwandten Disziplin, der *linearen Optimierung*.

[2] Alle vier Bedingungen können in fortgeschrittener Literatur verletzt werden, sind aber für dieses Beispiel angemessen.

Der Ausdruck $(C{:}1)$ soll eine (einfache) *Belegung* oder *label* bezeichnen und gibt an, daß die Variable C mit dem Wert 1 belegt wird, d.h. das tiefe C wird mit dem Daumen angeschlagen.

Die erste Bedingung schränkt die Belegung von Obertasten ein. Das C kann also mit allen Werten belegt werden. Das es ist eine Obertaste und kann daher nicht durch den Daumen ausgeführt werden. Das g und das c bleiben wiederum unbeschränkt.

Die Bedingung 1 ist also gerade die Menge

$$\begin{aligned} C_1 \quad = \{ \quad & (C{:}1), (C{:}2), (C{:}3), (C{:}4), (C{:}5), \\ & (e{:}2), (e{:}3), (e{:}4), (e{:}5), \\ & (c{:}1), (c{:}2), (c{:}3), (c{:}4), (c{:}5), \\ & (g{:}1), (g{:}2), (g{:}3), (g{:}4), (g{:}5) \quad \} \end{aligned}$$

Die Bedingung 2 schränkt die gleichzeitige Belegung von zwei Variablen ein. Eine solche *kombinierte Belegung* (*compound label*) notieren wir als $(g{:}1, c{:}2)$. In diesem Fall soll die Variable g mit 1 und zugleich die Variable c mit 2 belegt werden. Da zwei Variablen gebunden werden, liegt eine *zweifache* Belegung vor. Obwohl Belegungen hier zur Übersichtlichkeit als Tupel notiert werden, sind sie formal wiederum als Mengen anzusehen, es kann also kein Element mehrfach vorkommen, und die Anordnung der Elemente ist unbedeutend.

Angewendet auf die Variablen g und c erlaubt die Bedingung 2 diese kombinierten Belegungen:

$$\begin{aligned} C_2 \quad \supset \{ \quad & (g{:}1, c{:}2), (g{:}1, c{:}3), (g{:}1, c{:}4), (g{:}1, c{:}5), \\ & (g{:}2, c{:}1), (g{:}2, c{:}3), (g{:}2, c{:}4), (g{:}2, c{:}5), \\ & (g{:}3, c{:}1), (g{:}3, c{:}2), (g{:}3, c{:}4), (g{:}3, c{:}5), \\ & (g{:}4, c{:}1), (g{:}4, c{:}2), (g{:}4, c{:}3), (g{:}4, c{:}5), \\ & (g{:}5, c{:}1), (g{:}5, c{:}2), (g{:}5, c{:}3), (g{:}5, c{:}4) \quad \} \end{aligned}$$

Die anderen 5 Paare von Variablen erlauben die entsprechenden Belegungen. Die allgemeine Bedingung 2 ist die Vereinigung dieser sechs Mengen.

Bedingung 3 verlangt, daß zwei gleichzeitige Belegungen die natürliche Anordnung der Finger berücksichtigen. Angewendet auf die Variablen g und c erlaubt sie die kombinierten Belegungen

$$\begin{aligned} C_3 \quad \supset \{ \quad & (g{:}1, c{:}2), (g{:}1, c{:}3), (g{:}1, c{:}4), (g{:}1, c{:}5), \\ & (g{:}2, c{:}3), (g{:}2, c{:}4), (g{:}2, c{:}5), \\ & (g{:}3, c{:}4), (g{:}3, c{:}5), \\ & (g{:}4, c{:}5) \quad \} \end{aligned}$$

Wieder sind die entsprechenden Belegungen für die anderen Paare von Variablen zulässig.

Bedingung 4 ist ebenfalls die Vereinigung von 6 Mengen, die aber nicht alle dieselbe Struktur aufweisen. Drei dieser Mengen seien hier angegeben: Die Variablen g und c erlauben die Belegungen

$$\begin{aligned} C_4 \quad \supset \{ \quad & (g{:}1, c{:}2), (g{:}1, c{:}3), (g{:}1, c{:}4), (g{:}1, c{:}5), \\ & (g{:}2, c{:}4), (g{:}2, c{:}5), \\ & (g{:}3, c{:}5) \quad \} \end{aligned}$$

Die Tasten C und e sind einander näher und erlauben daher die Belegungen

$$
\begin{aligned}
C_4 \quad \supset \{ \quad & (C{:}1, e{:}2), (C{:}1, e{:}3), (C{:}1, e{:}4), \\
& (C{:}2, e{:}3), (C{:}2, e{:}4), \\
& (C{:}3, e{:}4), (C{:}3, e{:}5), \\
& (C{:}4, e{:}5) \quad \}
\end{aligned}
$$

Die Tasten C und c hingegen erlauben nur die Belegungen

$$C_4 \supset \{ \quad (C{:}1, c{:}4), (C{:}1, c{:}5) \quad \}$$

Wenn eine Belegung eine Teilmenge einer anderen mehrfachen Belegung ist, so wird sie auch als deren *Projektion* bezeichnet. So ist etwa die Belegung $(g{:}1)$ eine Projektion der Belegung $(g{:}1, c{:}2)$. Da die Gesamtbelegung alle Variablen binden und alle Bedingungen erfüllen muß, ist also eine vierfache kombinierte Belegung zu finden, die in jeder Bedingung eine Projektion besitzt. Aus den gestellten Bedingungen ergibt sich, daß die einzige Lösung $(C{:}1, e{:}2, g{:}3, c{:}5)$ lautet.

Im Beispiel wurden Bedingungen nur an höchstens zwei Variablen gleichzeitig gestellt.[3] Eine Bedingung, die nur eine Variable beschränkt, wird als *unär* bezeichnet, eine Bedingung, die zwei Variablen beschränkt, als *binär*. Allgemein bezeichnet die *Stelligkeit* einer Bedingung die Anzahl der in Beziehung gesetzten Variablen; der *Grad* einer Variablen ist die Mächtigkeit ihrer Wertemenge. Die höchste Stelligkeit und der höchste Grad bestimmen Stelligkeit und Grad des gesamten Problems.

Je nachdem, ob es eine Lösung gibt, die alle Bedingungen erfüllt, bezeichnet man ein CSP als *lösbar* oder *unlösbar*. Da der Lösungsraum eines CSP stets endlich ist und die Bedingungen an die Lösung exakt formuliert sein müssen, ist es immer entscheidbar, ob ein CSP lösbar ist. Dazu müssen nur alle möglichen Belegungen erzeugt und jeweils auf Gültigkeit überprüft werden. Je nach Aufgabenstellung ist das Ergebnis entweder die erste gültige Lösung oder die Menge aller gültigen Lösungen. Diese Lösungsmethode (bekannt als *generate and test*) ist vollständig und korrekt, liefert also alle gültigen und keine ungültigen Lösungen.

2.2.2 Ein reales Beispiel

Für das Beispielproblem führte generate and test schnell zum Erfolg. In wirklichen Problemen sind aber gewöhnlich erheblich mehr Variablen gleichzeitig zu belegen. Ein realistisches Beispiel für Bedingungserfüllung: Jedes Jahr erstellt der Deutsche Fußball-Bund den Spielplan der ersten Fußball-Bundesliga. Die Liga L besteht dabei aus 18 Vereinen, zwischen denen genau $|L| \cdot (|L| - 1) = 306$ Begegnungen ausgetragen werden müssen. Zu jeder Begegnung ist nun festzulegen, an welchem der 34 Spieltage sie angesetzt wird. Es sind also 306 Variablen (Matches $m_i \in L \times L$) zu belegen, deren Domäne jeweils die Werte 1 bis 34 enthält. Ein guter Spielplan sollte folgende Bedingungen erfüllen:

1. An jedem Spieltag müssen genau neun Begegnungen stattfinden.
2. Eine Mannschaft kann nicht zwei Begegnungen gleichzeitig austragen.

[3] Realistischerweise müßten auch noch Bedingungen an die gleichzeitige Stellung von drei Fingern gestellt werden.

3. Die Rückrunde ist die Umkehrung der Hinrunde.

4. Jede Mannschaft soll abwechselnd Heimspiele und Auswärtsspiele austragen.

Die Mengen von kombinierten Belegungen sind hierbei zu groß, um schnell aufgeschrieben zu werden. Es ist daher üblich, sie in impliziter Form anzugeben, etwa als boolesche Matrizen. Bei wenig beschränkten Problemen kann es auch sinnvoll sein, alle Belegungen aufzuzählen, die *nicht* zulässig sind. Da die definierende Eigenschaft einer Menge ihre charakteristische Funktion ist, genügt es auch, ein berechenbares Prädikat anzugeben, das entscheidet, ob eine kombinierte Belegung eine Bedingung erfüllt. Bezeichnen wir Gastgeber und Gast der Partie m durch m.host bzw. m.guest und die Belegung von m_i durch day(m_i), so lauten die vier Bedingungen:

1. $\forall i \in \{1, \dots, 34\} : |\{m \in M \mid \text{day}(m) = i\}| = 9$

2. $\forall m_i, m_j \in M :$
   $i \neq j \wedge \text{day}(m_i) = \text{day}(m_j) \rightarrow$
   $\{m_i.\text{host}, m_i.\text{guest}\} \cap \{m_j.\text{host}, m_j.\text{guest}\} = \emptyset$

3. $\forall m_i, m_j \in M :$
   $m_i.\text{host} = m_j.\text{guest} \wedge m_i.\text{guest} = m_j.\text{host} \rightarrow$
   $|\text{day}(m_i) - \text{day}(m_j)| = 17$

4. $\forall m_i, m_j \in M :$
   $\text{day}(m_i) + 1 = \text{day}(m_j) \rightarrow$
   $m_i.\text{host} \neq m_j.\text{host} \wedge m_i.\text{guest} \neq m_j.\text{guest}$

Die Stelligkeit des Problemes wird von der ersten Bedingung dominiert. Trotz der natürlichsprachlichen Formulierung "genau neun" hat sie nicht die Stelligkeit neun, sondern zehn: Sie verbietet gerade alle jene Belegungen, die zehn Variablen denselben Wert zuweisen, und erlaubt (enthält) alle anderen zehnfachen Belegungen. Die anderen Bedingungen haben die Stelligkeit zwei. Der Grad des Problemes beträgt 34.

2.2.3 Constraint-Netzwerke

Zur Veranschaulichung eines gegebenen CSP ist der Begriff des *Constraint-Netzwerkes* hilfreich. Ein Constraint-Netzwerk ist ein ungerichteter Graph mit ebensovielen Knoten, wie das zugrundeliegende Problem Variablen enthält. Mit jedem Knoten ist eine Tabelle seiner möglichen Werte assoziiert. Kanten verbinden alle Paare von Knoten, zwischen denen Bedingungen bestehen. An jeder Kante ist eine boolesche Matrix zu denken, die angibt, welche Paare aus Werten der beiden Variablen erlaubt sind. Die Zelle t_{ij} der Tabelle T_{kl} enthält den Wert 1, wenn die Bedingungen gleichzeitig den i-ten Wert der Variablen V_k und den j-ten Wert der Variablen V_l erlauben, sonst den Wert 0.

Eine naheliegende Methode, den Suchaufwand in einem CSP zu reduzieren, beruht auf der Überlegung, daß gewöhnlich verschiedene Zweige im selben Suchbaum aus demselben Grund fehlschlagen. Bei uninformierter Belegung von Variablen entstehen oft Zuweisungen, die eine unäre Bedingung an die betreffende Variable verletzen. Ohne besondere Maßnahmen würden solche Zuweisungen in anderen Zweigen des Suchbaumes immer wieder versucht, obwohl sie eine Lösung in jedem Fall verhindert. Solche Werte können schon vor Suchbeginn aus der betreffenden Domäne entfernt werden, ohne daß sich die Lösungsmenge des Problemes verändert. Enthält eine Domäne keine solchen unmöglichen Werte mehr, so wird sie als *knotenkonsistent* bezeichnet. Sind alle Domänen eines CSP knotenkonsistent, so kann man es absuchen, ohne unäre Bedingungen auszuwerten.

Einen Schritt weiter geht die systematische Betrachtung der gleichzeitigen Belegung von zwei Variablen. Selbst wenn die verbleibenden Werte a_i der Variable A und b_i der Variable B alle unären Bedingungen erfüllen, sind i.A. nicht alle Paare (a_i, b_j) gültige Belegungen, weil sie zweistellige Bedingungen verletzen können. Im Beispiel kann etwa jede Partie prinzipiell alle 34 Werte annehmen, aber für Hin- und Rückspiel sind nur 34 der 34^2 möglichen Kombinationen gültig. Im Verlauf der Suche kann es nun dazu kommen, daß alle gültigen Werte der Variable A einen bestimmten Wert der Variable B ausschließen. Läge etwa die Ansetzung der Partie FC Bayern München – Borussia Dortmund am letzten Spieltag schon fest[4], so würde dies den Wert 34 für alle anderen Partien dieser beiden Mannschaften ungültig machen.

Im Constraint-Netzwerk drückt sich diese Situation dadurch aus, daß eine Zeile oder Spalte einer Tabelle ausschließlich Nullen enthält. Auch solche nicht *kantenkonsistenten* Werte können an keiner Lösung teilhaben und dürfen daher entfernt werden. Durch die Entfernung *eines* Wertes aus einem CSP verkleinern sich *alle* Tabellen, die der betreffenden Variablen zugeordnet sind, um eine Zeile oder Spalte. Dabei können insbesondere auch Zellen wegfallen, die eine 1 enthielten. Es ist also möglich, daß durch den Ausschluß einer Inkonsistenz eine neue Inkonsistenz entsteht, weil die letzte 1 einer Zeile entfernt wurde. Die Überprüfung muß so lange fortgesetzt werden, bis sich keine Domäne mehr verändert. Diese Verfahren heißt *constraint propagation* oder *filtering*.

Die Herstellung von Kantenkonsistenz in einem CSP ist wesentlich aufwendiger als die Herstellung von Knotenkonsistenz. Bei sorgfältiger Buchführung darüber, welche Kanten jeweils zu überprüfen sind, kann ein CSP mit n Variablen und e Kanten in $O(e \cdot n^2)$ Zeit kantenkonsistent gemacht werden. Dennoch kann im anschließenden Suchvorgang nicht auf die Auswertung binärer Bedingungen verzichtet werden, da inkonsistente Paare nach wie vor vorkommen können; sie können aber immer vermieden werden.

Kantenkonsistenz allein ist weder hinreichend noch notwendig für die Lösbarkeit eines CSP: Ein nicht kantenkonsistentes CSP kann dennoch gelöst werden, wenn man nur sorgfältig die richtigen (konsistenten) Werte auswählt. Umgekehrt kann ein kantenkonsistentes CSP dennoch unlösbar sein, wenn es mehr als zwei Variablen enthält. Um den Suchvorgang ganz zu umgehen, muß der Begriff der Konsistenz also noch weiter gefaßt werden. Im allgemeinen Fall bezeichnet man mit *k-Konsistenz* die Eigenschaft, daß bei einer gültigen Belegung von $k-1$ Variablen jede weitere Variable so belegt werden kann, daß alle höchstens k-stelligen Bedingungen erfüllt sind. Liegt k-Konsistenz für alle k bis zu einer Grenze m vor, so spricht man von *starker m-Konsistenz*[5]. Offenbar genügt es zur Lösung eines Problemes mit k Variablen, durch Entfernung inkonsistenter Werte starke k-Konsistenz herzustellen und dann aus den verbleibenden Werten ein beliebiges Tupel zusammenzustellen.[6] Im allgemeinen ist aber die Herstellung dieser Eigenschaft mindestens ebenso aufwendig wie eine vollständige Suche. Welcher Grad an Konsistenz vor der Suche hergestellt werden sollte, hängt maßgeblich vom betrachteten CSP ab.

[4] Entsprechende Bedingungen könnten etwa die Fernsehanstalten stellen.

[5] Der Begriff der starken k-Konsistenz ist notwendig, weil im allgemeinen Fall k-Konsistenz nicht die $k-1$-Konsistenz impliziert. Zur Erinnerung: 2-Konsistenz bedeutet, daß eine gültige einfache Belegung immer zu einer gültigen zweifachen Belegung erweitert werden kann. Das schließt aber nicht aus, daß es Domänen gibt, die nicht gültig belegt werden können.

[6] Tatsächlich genügt es, starke m-Konsistenz für einen evtl. kleineren Wert von m herzustellen, der von der Topologie des Constraint-Netzwerkes abhängt ([Freuder82]).

2.2.4 Lösungsverfahren

Die Zahl aller möglichen Belegungen im Beispielproblem ist $34^{306} \approx 4.28 \cdot 10^{468}$. Offenkundig ist es also nicht mehr möglich, alle kombinierten Belegungen zu erzeugen und zu überprüfen.[7] Auch die Anwendung von constraint propagation ist nicht sinnvoll: Da es keine unären Bedingungen gibt, kann auch kein Wert aus einer Domäne ausgeschlossen werden. Nur mehrfache, in sich widersprüchliche Belegungen können ausgeschlossen werden, aber auch dafür wäre unpraktikabler Aufwand notwendig, ohne daß das Problem wesentlich einfacher wird. Wir kommen also nicht umhin, Variablen willkürlich zu belegen, ohne zunächst zu wissen, welche Wahl zu einer Lösung führen wird: Es muß ein Suchprozeß ausgeführt werden.

Die Lösung eines CSP durch Suche auf dem Lösungsraum muß kein blindes generate and test sein. Oft ist es möglich, schon nach wenigen Belegungen zu erkennen, daß eine Bedingung verletzt ist. Beim *backtracking* wird diese Möglichkeit systematisch ausgenutzt. Dabei wird eine vollständige Ordnung der Variablen und aller Domänen angenommen. Die Variablen werden gemäß ihrer Reihenfolge nacheinander instantiiert; sobald eine Bedingung verletzt ist, wird die zuletzt getroffene Entscheidung revidiert. Für Probleme, die nicht sehr stark eingeschränkt sind, führt dieser Weg oft zum Erfolg.

Die Lösung eines CSP durch backtracking benötigt freilich im schlimmsten Fall noch immer exponentiellen Zeitaufwand, obwohl der weitaus größte Teil der falschen Lösungen nicht vollständig aufgebaut werden muß. Tatsächlich ist das allgemeine CSP $\mathcal{NP}$-vollständig, was bedeutet, daß vermutlich jeder korrekte Algorithmus in bestimmten Fällen exponentiell viel Zeit benötigen wird.

Auch das Beispielproblem dürfte sich nicht durch reines backtracking lösen lassen. Bei der Suche ergeben sich sehr oft fast vollständige Teillösungen, die aber in einem der letzten Schritte scheitern. Es liegt hier der besonders ungünstige Fall vor, daß alle Bedingungen alle Variablen beschränken, ihre Erfüllung aber erst spät geprüft werden kann. Man betrachte nur die Bedingung C_3: Einerseits stellt sie eine sehr strikte Bedingung an die gemeinsame Belegung jeweils zweier Variablen, denn nach erfolgter Ansetzung einer Begegnung ist für die Ansetzung der inversen Begegnung nur noch ein Wert zulässig. Die Menge der von C_3 erlaubten Tupel ist also sehr klein. Bei zufälliger Instantiierungsreihenfolge wird der einzig erlaubte Wert der inversen Partie aber erst nach vielen Fehlversuchen gefunden. Dabei gilt C_3 zwischen allen Paaren von zwei Variablen, muß also bei jeder Belegung wieder überprüft werden.

Es liegt nahe, die Bedingung C_3 auf andere Weise sicherzustellen. Beispielsweise könnte gefordert werden, daß zunächst die Menge M derart in zwei gleichgroße Mengen partitioniert wird, daß zueinander inverse Partien in verschiedenen Teilmengen liegen. Danach wird über der ersten Teilmenge die Hinrunde geplant, wofür nur 153 Variablen mit einem Wert aus der Menge $\{1 \dots 17\}$ belegt werden müssen. Die Rückrunde entsteht danach durch triviale Transformation der Hinrunde. Der zusätzliche Aufwand für die Partitionierung wird gegenüber den ersparten Fehlversuchen vernachlässigbar sein.

Es verbleiben allerdings weiterhin $17^{153} \approx 1.81 \cdot 10^{188}$ mögliche Lösungen. Wenn wie in diesem Problem prinzipiell alle Domänen und alle Werte äquivalent sind und viele Lösungen möglich sind, ist es oft lohnend, eine bestimmte Regelmäßigkeit der Belegungen vorauszusetzen, also den Lösungsraum künstlich zu beschränken. Wir könnten etwa versuchen, das Programm aller Mannschaften ähnlich zu gestalten. Die Abfolge der Gegner von Mannschaft 1 sei o.B.d.A.

$$(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18)$$

[7] Die Zahl der Atome im beobachtbaren Universum wird auf 10^{80} geschätzt.

Mannschaft 1 tritt also am 1. (und am 18.) Spieltag gegen Mannschaft 2 an, danach gegen Mannschaft 3 etc. Daraus folgt, daß Mannschaft 2 am 1. und 18. Spieltag gegen Mannschaft 1 antritt. Will man auch Mannschaft 2 der Reihe nach gegen alle Gegner antreten lassen, so würde ihr Programm lauten

$$(1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18)$$

Das ist nicht möglich, weil dann die Mannschaften 3–18 jeweils zwei Spiele gleichzeitig austragen müßten. Das Problem kann gelöst werden, wenn eine spätere Begegnung vorgezogen wird:

$$(1, 18, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17)$$

Mannschaft 3 hat dann das Programm

$$(17, 1, 2, 18, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16)$$

Das allgemeine Prinzip ist: Mannschaft m tritt am Spieltag t gegen Mannschaft $(t - m + 2) \bmod 17$ an. Die Begegnung mit Mannschaft 18 tritt an die Stelle der nicht ausgetragenen Begegnung mit sich selbst. Das Programm von Mannschaft 18 spiegelt diese Ausnahmerolle wieder:

$$(10, 2, 11, 3, 12, 4, 13, 5, 14, 6, 15, 7, 16, 8, 17, 9, 1)$$

Durch geschickte Wahl zusätzlicher Bedingungen läßt sich also sehr schnell ein symmetrisches Schema aufstellen, in dem nur noch die Verteilung der Heim- und Auswärtsspiele zu finden ist. Da alle Mannschaften abwechselnd Heim- und Auswärtsspiele bestreiten sollen, muß nur festgelegt werden, wo die neun Spiele des ersten Spieltages ausgetragen werden; daraus wird sich der weitere Reiseplan aller Mannschaften eindeutig ergeben. Das Restproblem enthält nur noch 9 Variable vom Grad 2.

Versucht man jedoch, das neue Problem durch backtracking zu lösen, so findet sich keine Lösung. Haben wir eine ungeeignete Symmetriebedingung gestellt, oder ist das Ausgangsproblem überhaupt unlösbar? Es läßt sich mit einem einfachen Paritätsargument zeigen, daß *keine* Belegung das Ausgangsproblem lösen kann: Man betrachte die beiden ersten Begegnungen der Saison. Die Gastgeber dieser beiden Partien tragen gleichzeitig Heimspiele aus und müssen laut Bedingung 4 also die ganze Saison über denselben Rhythmus von Heim- und Auswärtsspielen beibehalten. Damit gibt es keinen Spieltag, an dem der direkte Vergleich stattfinden könnte, weil dazu eine der beiden Mannschaften ihren Rhythmus unterbrechen müßte.

Es stellt sich also heraus, daß die keine Belegung die Bedingung C_4 erfüllen kann. Damit ist natürlich auch die Schnittmenge der Bedingungen leer: Das Problem ist unlösbar. Aufgrund der nur impliziten Angabe der Mengen C_i war diese Situation bei der Problemstellung nicht zu erkennen.

2.2.5 Partielle Constraint-Satisfaction-Probleme

Um die Saison dennoch austragen zu können, muß man weniger strikte Bedingungen wählen. Ein CSP, in dem eventuell nicht alle Bedingungen erfüllt werden müssen, wird als *partial constraint satisfaction problem* bezeichnet. Die optimale Lösung läßt sich in verschiedener Weise definieren:

- Eine Belegung darf keine Bedingungen verletzen. Wenn es keine Lösung gibt, wird diejenige Teilbelegung gesucht, die die meisten Variablen bindet.
- Eine Belegung darf Bedingungen verletzen. Die beste Lösung ist diejenige, die am wenigsten Verletzungen verursacht.
- Die Bedingungen werden in Gruppen verschiedener Wichtigkeit eingeteilt. Die beste Lösung ist die, die nur Bedingungen von möglichst geringer Wichtigkeit verletzt.
- Jede Bedingung erhält eine individuelle Bewertung zugeordnet. Die Bewertung einer Belegung ist eine Funktion (z.B. die Summe) der Bewertungen aller verletzten Bedingungen. Die beste Lösung ist diejenige mit der besten Bewertung.

Im Beispiel wird man zweckmäßig die Bedingung C_4 als verletzbar erklären und eine Lösung suchen, die möglichst wenige Verletzungen verursacht. Damit bietet es sich an, noch weitere Bedingungen zu stellen, die nicht notwendig, sondern nur wünschenswert sind:

- Eine Mannschaft soll möglichst abwechselnd gegen schwache und starke Gegner antreten.
- Die attraktivsten Begegnungen sollen möglichst spät stattfinden.
- Mehrere attraktive Begegnungen sollten nicht gleichzeitig stattfinden.

Eine konkrete Bundesliga-Saison unterliegt überdies weiteren, nicht systematischen Beschränkungen:

- Der SC Freiburg kann die Saison nicht mit einem Heimspiel beginnen, da sein Stadion erst in der folgenden Woche betriebsfertig sein wird.
- Die beiden Münchner Mannschaften teilen sich ein Stadion und können darum nicht gleichzeitig zu Hause antreten.
- Im Ruhrgebiet können nicht mehr als vier Spiele gleichzeitig stattfinden, da nur begrenzt Polizeikräfte zur Verfügung stehen.
- Der SV Werder Bremen wird in der Winterpause sein Stadion ausbauen und will daher die Heimspiele gegen attraktive Gegner erst in der Rückrunde austragen.

Einige dieser Bedingungen sind unverletzbar, andere sind nur wünschenswert, können aber finanzielle Folgen in Millionenhöhe haben. Das reale Problem wäre also zweckmäßig als partial CSP mit verschieden priorisierten Bedingungen zu modellieren.[8]

Einen allgemein besten Weg zur Lösung eines CSP gibt es also nicht. Im Einzelfall können verschiedene Ansätze angebracht sein. Eine bisher nicht erwähnte Möglichkeit ist es z.B.,

[8] Tatsächlich wird die Fußball-Bundesliga nach dem oben vorgestellten Symmetrieprinzip ausgetragen. Jede Mannschaft verletzt einmal die Bedingung 4, nämlich gerade bei der Begegnung mit Mannschaft 18. Dieser abstrakte Spielplan verändert sich nie. Die Saisonplanung beschränkt sich auf die Zuordnung der Teamnamen zu den 18 Zahlen. Es werden also "nur" 18! =6.402.373.705.728.000 Lösungen überhaupt in Betracht gezogen. Dieser Lösungsraum ist gewöhnlich zu klein, um weitere Bedingungen zu berücksichtigen wie etwa die obengenannten. Eine ausführlichere Suche wäre zwar möglich, ist dem DFB aber erklärtermaßen zu teuer ([Dworschak97]).

ein CSP in ein äquivalentes CSP der Stelligkeit 2 zu transformieren. Im allgemeinen Fall erhöht sich dadurch jedoch der Grad des Problemes exponentiell. Diese Umwandlung ist also nur dann nützlich, wenn die Auswertung vielstelliger Bedingungen aufwendiger ist als das Durchsuchen großer Domänen.

Schließlich ist auch die Art der Formalisierung für den Erfolg maßgeblich: Ein Problem kann oft auf verschiedene Arten als CSP dargestellt werden. So hätte das Textbeispiel auch als Problem mit 34 Variablen (den Spieltagen) modelliert werden können. Jede Domäne wäre dann die Menge aller gültigen Spieltage. An einem Spieltag muß jede Mannschaft genau einmal beteiligt sein, so daß es rechnerisch 18! Möglichkeiten gibt, einen Spieltag zu planen. Da aber die Reihenfolge der 9 Begegnungen gleichgültig ist, sind jeweils 9! dieser Möglichkeiten äquivalent, so daß die Größe jeder Domäne $18!/9! = 17.643.225.600$ beträgt. Damit wären freilich immer noch $2.43 \cdot 10^{348}$ Belegungen möglich; außerdem wäre es äußerst aufwendig, die Verträglichkeit zweier Variablenbelegungen zu überprüfen, und so frühzeitig widersprüchliche Werte auszuschließen.

2.3 Dependenzgrammatiken

Ein Problem als CSP zu modellieren, ist nur dann sinnvoll, wenn seine Variablen einen bestimmten Grad an gegenseitiger Abhängigkeit aufweisen. Bestehen sehr wenig Bedingungen, so kann eine Lösung oft schneller durch bloßes Ausprobieren gefunden werden; bestehen sehr viele Einschränkungen, so ist es sinnvoller, mit problemspezifischem Wissen eine Lösung von Anfang an zu planen, als eine Suche durchzuführen, die fast ausschließlich sinnlose Alternativen betrachtet.

In der natürlichen Sprache besteht gewöhnlich ein mittlerer Grad an gegenseitiger Abhängigkeit. Im allgemeinen unterliegen Gruppen zusammengehöriger Elemente dabei strengen Regeln, während die Gruppen untereinander freier kombiniert werden können. So könnte etwa der letzte Satz ohne Bedeutungsänderung auch geschrieben werden als

(17) Dabei erlauben die Gruppen zusammengehöriger Elemente untereinander eine freiere Kombination, wogegen für die Gruppen selbst strenge Regeln gelten.

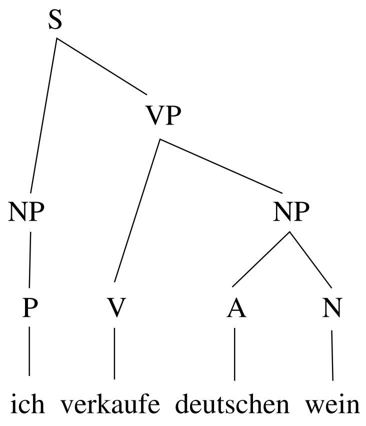
Grundsätzlich ist es also möglich, das Parsingproblem durch Anwendung lokaler Bedingungen zwischen benachbarten Elementen oder bereits gefundenen Gruppen zu lösen.

Diese Gruppen von Formen werden gewöhnlich als *Phrasen* bezeichnet. Jede Phrase besteht aus zumindest einer Wortform und gehört einem bestimmten Typ an, ist also z.B. eine Nominalphrase, eine Verbalphrase etc. Auch der ganze Satz kann als Phrase angesehen werden. Eine typische Phrasenstrukturanalyse könnte etwa der Äußerung

(18) ich verkaufe deutschen wein[9]

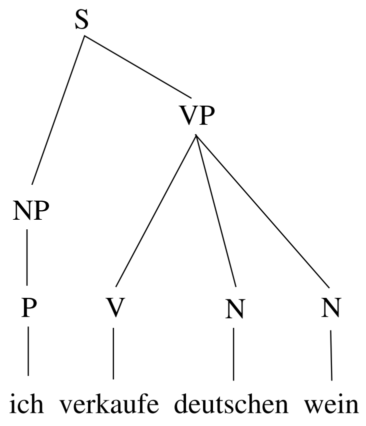
in etwa folgende Struktur zuweisen:

[9]Kleinschreibung soll hier andeuten, daß eine gesprochene Äußerung analysiert werden soll.



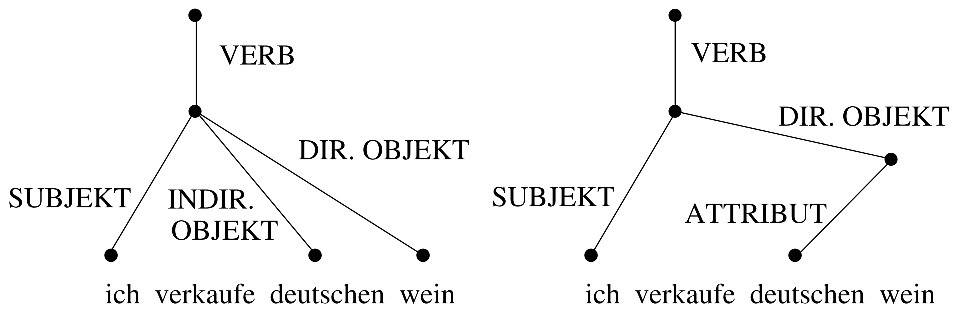
Hier wird die Struktur des Satzes durch die Annahme von acht Phrasen erklärt, von denen vier einzelne Worte repräsentieren und vier die Gruppierung von anderen Phrasen. Jede solche Phrase ist gerechtfertigt durch eine Phrasenstrukturregel wie etwa "S $\rightarrow$ NP VP". Deutlich ist zu erkennen, daß die acht grundsätzlichen Schritte dieser Analyse nicht alle parallel ausgeführt werden können, da die letzte Regel erst angewendet werden kann, sobald die NP und die VP analysiert worden sind.

Es wurde bereits erwähnt, daß eine Phrasenstrukturgammatik, die alle sprachlichen Formen erklären soll, zwangsläufig an verschiedenen Stellen mehrdeutig ist. Sie wird also verschiedene Strukturen zulassen, die dieselbe Äußerung erklären können. So könnte etwa dem Beispielsatz auch eine andere Struktur zugrunde liegen (man beachte die Sinnänderung):



Diese Struktur erklärt die Äußerung durch die Annahme von neun Phrasen, von denen vier Worten entsprechen und fünf komplexen Phrasen. Es fällt auf, daß die beiden Strukturen nicht nur durch unterschiedliche Regeln erzeugt wurden, sondern auch durch unterschiedlich viele. Es ist also im allgemeinen nicht möglich vorherzusagen, wieviele Phrasen zur Erklärung einer Äußerung nötig sein werden. In einem CSP muß aber die Struktur der Lösung von Beginn an bekannt sein, damit die üblichen Lösungsverfahren Anwendung finden können. Insbesondere ist es nicht möglich, nachträglich weitere Variablen in ein Problem einzuführen.

Um Methoden des constraint satisfaction auf das Parsingproblem anzuwenden, muß man also einen Formalismus wählen, der es erlaubt, die Anzahl der notwendigen Entscheidungen im voraus zu bestimmen. Dies ist der Fall im Modell der *Dependenzgrammatik*, die auf [Tesnière59] zurückgeht. Hierbei wird an Stelle der sonst üblichen *Konstituenz* systematisch die Beziehung der *Dependenz* (oder *Konnexion*) verwendet. Zwei Worte A und B stehen in Dependenz, wenn der Bedeutungsbeitrag des Wortes A zum Satz sich auf die Bedeutung des Wortes B bezieht. In grober Näherung kann man das Bestehen von Dependenz zwischen zwei Worten dadurch nachweisen, daß das Wort A nur dann weggelassen werden kann, wenn auch das Wort B wegfällt. Die finiten Verben dienen als Wurzel der gesamten Struktur. Die graphische Darstellung aller Dependenzen in einem Satz wird als *Stemma* bezeichnet. Die beiden Bedeutungsvarianten des Beispielsatzes können wie folgt durch Stemmata dargestellt werden:



Dabei sind verschiedene Unterschiede zur Phrasenstruktur wesentlich:

- Die Dependenzrelation macht im Gegensatz zur Phrasenstrukturregel keine Aussage über die Position der beteiligten Wörter. Varianten derselben Äußerung, die sich lediglich durch Umstellung voneinander unterscheiden, können durch dasselbe Stemma repräsentiert werden.
- Bezeichnungen wie “NP” oder “VP” beschreiben immer nur die Struktur jeweils einer Phrase. Die Konstituenzbeziehung muß im Diagramm eigens angegeben werden. *Label* wie “Subjekt” oder “Objekt” beschreiben dagegen immer das Verhältnis von genau zwei Formen zueinander. Weitere Beziehungen zwischen Formen werden nicht repräsentiert.
- “Knoten, die allein logisch motiviert sind, denen also keine Wörter im Satz zuzuordnen sind, werden nicht erlaubt.” ([Weber92]) Die gesamte Struktur wird durch genau eine Unterordnung pro Form dargestellt. Damit ist garantiert, daß sich die Zahl der Variablen vor der Verarbeitung bestimmen läßt.

2.4 Eliminatives Parsing mit SCLNs

Das Parsingproblem wurde erstmals in [Maruyama91] als Instanz des allgemeinen CSP dargestellt. Das besondere Ziel war dabei die Realisierung von Parsing durch *strukturelle Disambiguierung*: zu einer gegebenen Äußerung soll die Struktur gefunden werden, die dem Sinn der Äußerung entspricht.

Nach Maruyama hat eine *Constraint-Dependenz-Grammatik (CDG)* drei Datenstrukturen bereitzustellen:

1. ein Lexikon erlaubter Wortformen

2. eine Menge L erlaubter Unterordnungsbeziehungen oder Labels
3. Bedingungen[10], die die erlaubten Unterordnungen der auftretenden Wortformen regeln.

Zu diesem Zweck wird das Parsen einer Sprachäußerung wie folgt als CSP formalisiert: Zu jeder Wortform $w \in W$ wird eine Variable $V_i \in V$ erstellt, deren Werte die Form (l, w) mit $l \in L$, $w \in W \cup \{\mathrm{NIL}\}$ annehmen. Jedem Wort ist also eine Beschriftung zuzuordnen, die angibt, welche Funktion es erfüllt, und welchem anderen Wort es untergeordnet ist. Die nicht untergeordneten finiten Verben werden formal mit $(\mathrm{VERB}, \mathrm{NIL})$ beschriftet.

Viele Elemente natürlicher Sprache erlauben eine "freie" Anbindung, d.h. sie können gleichbedeutend an verschiedenen Stellen der Äußerung auftreten. Hierzu zählen in vielen indoeuropäischen Sprachen die Präpositionalphrasen: Als Beispiel für dieses Phänomen wird die Äußerung

(19) Put the block on the floor on the table in the room.

genannt. Treten mehrere solche Präpositionalphrasen in einer Äußerung auf, so kann die Zahl der möglichen Interpretationen überexponentiell sein.

In dieser Situation ist es sinnvoll, die Parse-Bäume, die den verschiedenen Varianten einer Äußerung entsprechen, nicht alle einzeln aufzubauen, sondern implizit in einer kompakten Datenstruktur zu repräsentieren. Zu diesem Zweck wird ein initiales Constraint-Netzwerk aufgebaut, das in polynomieller Zeit herstellbar ist. Die Mehrdeutigkeit einer Äußerung wird dann durch die lokale Propagation von Bedingungen eingeschränkt, wobei nur so viele Bedingungen zur Anwendung kommen sollen, wie im Einzelfall nötig sind.

Die Gesamtkomplexität des Verfahrens wird dominiert durch den Propagationsalgorithmus. Da die Zahl der Kanten in einem CDG-Netzwerk quadratisch von der Zahl der Variablen abhängt, vereinfacht sich dessen Komplexität von $O(n^2e)$ zu $O(n^4)$ in Abhängigkeit von der Länge n der Äußerung. Gleichzeitig ist die Menge der durch CDG beschreibbaren Sprachen echt größer als die Menge der kontextfreien Sprachen.

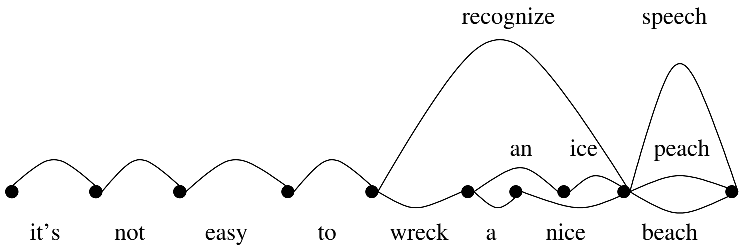
Harper ([Harper94]) zeigte, daß sich der Formalismus der CDG gut dafür eignet, die parallele Nutzung verschiedener Wissensquellen zu realisieren. Es wird ein System zur automatischen Anfragebeantwortung vorgestellt, welches Ambiguität auf mehreren Ebenen bewältigt:

- *akustische* Mehrdeutigkeit: Wegen der Verarbeitung gesprochener Sprache ist die Frage, welche Wörter zu verarbeiten sind, von Anfang an mit Unsicherheit behaftet.
- *lexikalische* Mehrdeutigkeit: Auch eine sicher erkannte gesprochene Form kann verschiedene Wortarten repräsentieren (so etwa *show* ein Nomen oder Verb).[11]
- *grammatische* Mehrdeutigkeit: Auch bei gleicher Wortart kann eine Form verschiedene Ausprägungen eines Merkmals realisieren (so etwa *take* die erste, zweite oder dritte Person Plural und noch andere Formen).
- *pragmatische* Mehrdeutigkeit: Anfragen können zu verschiedenen Themengebieten gestellt werden. Das System muß selbst entscheiden, welche Wissensbasis angesprochen werden soll.

[10] Maruyama spricht von *einer* Bedingung, die die Lösung erfüllen muß, wobei aber angenommen wird, daß diese Bedingung eine Konjunktion verschiedener Bedingungen sein kann.

[11] Das Phänomen ist im Englischen wegen der häufigen Null-Derivation besonders verbreitet.

Der akustischen Mehrdeutigkeit gesprochener Sprache wird durch *Wortgraphen* begegnet: In einem Wortgraphen können mehrere Hypothesen über dasselbe Zeitintervall aufgestellt werden. Jeder Pfad durch den Graphen entspricht einer möglichen Deutung der Eingabe. Es ist Aufgabe des Parsingverfahrens, zu jedem Intervall die richtige Deutung zu bestimmen. Der folgende Wortgraph stellt ein bekanntes Beispiel für akustische Mehrdeutigkeit dar:



13 Kanten repräsentieren hier insgesamt 9 mögliche Deutungen des Sprachsignals.

Durch die Verwaltung von alternativen Worthypothesen erweitert sich das Constraint-Netzwerk zum *spoken language constraint network (SCLN).* Ein solches Netzwerk enthält weiterhin einen Constraintknoten für jede Form; es gibt aber insgesamt mehr Wortformen und damit auch mehr Knoten.

Einerseits wird die Disambiguierung durch diese Änderung aufwendiger: Da in der Lösung nur eine Deutung vorkommen kann, müssen alle bis auf einen dieser Knoten im Lauf der Analyse wieder aus dem Netzwerk entfernt werden. In einem solchen Netzwerk wird sich die Zahl der Variablen also fortwährend verringern. Formal kann ein gelöschter Knoten mit dem speziellen Wert "ungültig" belegt werden. Das Lösungsverfahren muß außerdem sicherstellen, daß die Menge der ausgewählten Variablen über genau einen Pfad im Wortgraphen führt. Andererseits steigt der Berechnungsaufwand nicht im gleichen Maß an wie die Knotenzahl, denn zwischen Knoten, die Varianten zueinander darstellen, brauchen keine Bedingungen ausgewertet zu werden, da solche Paare von Variablen nicht in derselben Lösung vorkommen können. Bei geeigneter Anordnung der Knoten im Netz kann der Propagationsalgorithmus unverändert bleiben. Mit Hilfe von graphentheoretischen Überlegungen ist es dann möglich, die Propagation von binären Bedingungen über ein SCLN weiterhin in $O(n^4)$ Zeit vorzunehmen.

Ganz analog zur akustischen Mehrdeutigkeit wird lexikalische Mehrdeutigkeit behandelt: Für jede lexikalische Variante wird ein eigener Knoten angelegt. Genau einer dieser Knoten muß in der Lösung vorkommen, und zwischen alternativen Knoten brauchen keine Bedingungen berechnet zu werden.

Grammatische Mehrdeutigkeit wird dagegen anders behandelt. Verschiedene Ausprägungen von Merkmalen eines Wortes, die nicht an der Wortform erkennbar sind, werden nicht durch verschiedene Knoten modelliert. Stattdessen wird ein unterspezifizierter Knoten angelegt, der nur eine Liste der möglichen Ausprägungen enthält. Erst sobald eine Bedingung tatsächlich auf das betreffende Merkmal zugreift, wird der Knoten in mehrere neue Knoten aufgespalten, von denen jeder eine andere Ausprägung dieses Merkmals repräsentiert. Von den neuen Knoten können gewöhnlich einige sogleich wieder ausgeschlossen werden, da ja soeben eine bestimmte Bedingung an eben dieses Merkmal gestellt wurde.

Semantische oder pragmatische Mehrdeutigkeit schließlich wird durch den Begriff des *Kontextes* modelliert: Jedes bekannte Wort besitzt eine Liste der Kontexte, in denen es auftreten kann. In einer gültigen Lösung müssen sich alle Worte auf denselben Kontext beziehen. In einem eigenen Verarbeitungsschritt können damit noch vor der Propagation alle Worte entfernt werden, deren Kontexte keinen vollständigen Pfad durch das Netzwerk bilden können.

Durch eine andere Erweiterung des Verfahrens wird das Phänomen dargestellt, daß in der natürlichen Sprache ein Wort oft mehr als eine Funktion hat. So wird die äußere Form oder *Syntax* von Äußerungen gewöhnlich durch Funktionen wie "Subjekt", "Prädikat" und "Objekt" erklärt. In der Äußerung

(20) Der Architekt baut das Haus

erfüllt dann das Wort "Architekt" die syntaktische Funktion des Subjektes. In der alternativen Formulierung

(21) Das Haus wird vom Architekten gebaut

erfüllt es die Funktion einer Präpositionalanbindung, obwohl der beschriebene Sachverhalt derselbe ist. Es ist sinnvoll anzunehmen, daß auf einer anderen Betrachtungsebene die Funktion des Wortes "Architekt" tatsächlich in beiden Sätzen dieselbe ist, nämlich der der Bedeutungsrepräsentation oder *Semantik*.

Um diese verschiedenen Funktionen darzustellen, werden zwei Variablen pro Wortform benötigt. Die gesamte Interpretation der Äußerung besteht dann aus zwei unverbundenen Stemmata, von denen das eine nur die syntaktische und das andere nur die inhaltliche Struktur darstellt. Im weiteren werden diese verschiedenen Funktionen eines Wortes als unterschiedliche *Rollen* bezeichnet.

In einer CDG können sowohl Bedingungen an die Rollen verschiedener Worte gestellt werden als auch Bedingungen an die verschiedenen Rollen eines Wortes: Eine Bedingung kann lauten "Nur ein Wort kann das grammatische Subjekt sein" oder auch "Ein grammatisches Objekt kann nicht die semantische Funktion des Agens ausüben." Die Bedingungen verschiedener Rollen können prinzipiell in beliebiger Reihenfolge angewandt und propagiert werden.

Das Parsen von SCLN ist also ein Spezialfall des allgemeinen CSP, wobei die Domänen der Variablen die spezielle Form $\{\text{invalid}\} \cup (L \times (V \cup \{\text{NIL}\}))$, besitzen, d.h. die Größe der Domänen wächst mit der Zahl der Variablen. Im Verlauf der Analyse kann sich die Zahl der Variablen verringern.

Im Verhältnis zu anderen Parsingverfahren besitzt das Parsen von SCLN den Vorteil einer sehr guten Parallelisierbarkeit: Bei Verwendung von $O(n^4)$ Prozessoren in Abhängigkeit von der Länge einer Äußerung kann die Lösung in einer Zeit berechnet werden, die vornehmlich von der Größe der Grammatik abhängt, und nur noch logarithmisch von der Länge der Äußerung ([Helzermann,Harper92]).

Dabei sind verschiedene Zwischenstufen zwischen serieller und massiv paralleler Abarbeitung möglich: Zum Beispiel kann jeder Bedingung ein eigener Prozessor zugeordnet sein, so daß alle Belegungen einer Lösung gleichzeitig ausgewertet werden; umgekehrt können auch verschiedene alternative Lösungen erzeugt und von je einem Prozessor bewertet werden. Verschiedene Arten von Wissen können in gleicher Form verfügbar gemacht werden; trotzdem ist die Reihenfolge ihrer Anwendung beliebig. Schließlich läßt sich in dem Formalismus ein Algorithmus mit der gewünschten anytime-Eigenschaft realisieren.[12] Aus diesen Gründen ist das Parsen von SCLN gut geeignet für die robuste Verarbeitung realer natürlicher Sprache.

[12] Über die Möglichkeiten des inkrementellen Parsings mit SCLN vergl. [Schulz98].

Kapitel 3

Spezieller Teil

3.1 Ausgangslage

Als Ausgangspunkt diente das in [Schröder96] vorgestellte Parsing-System `cdg`; die Syntax der gültigen Eingaben wird in [Schröder97a] erläutert. Es diente als erstes Demonstrationssystem des Forschungsprojektes DAWAI, das das Parsen natürlicher Sprache unter beschränkten Ressourcen untersucht. Es stellt Methoden zur Verwaltung und Verarbeitung von Constraint-Grammatiken und Testdaten bereit.

3.1.1 Datenstrukturen

Eine Grammatik besteht aus einer endlichen Menge von Wortformen mit assoziierter sprachlicher Information (dem Lexikon) und Regeln über die möglichen Verbindungen der Wortformen untereinander (den Rollen und Bedingungen). Ein typischer Lexikoneintrag kann wie folgt eingegeben werden:

```
fressen_3p := fressen :
 [ syn:[ cat:verb, num:pl, pers:3, needs_subj:yes, needs_aobj:yes],
   sem:[ cat:EAT, needs_agent:SUBJ, needs_patient:OBJA]
 ];
```

Die erste Angabe ist ein eindeutiger Bezeichner, durch den Homonyme unterschieden werden können. Die zweite Angabe bezeichnet die Wortform, die im Analysesatz auftreten kann.

Alle verfügbare Information über die Merkmale einer Wortform wird in einer Merkmalsstruktur niedergelegt, die aus Paaren aus Attribut und zugeordnetem Wert besteht. Der Wert eines Attributes kann wiederum eine Merkmalsstruktur sein. Dadurch entstehen verschachtelte Werte, auf die durch Konkatenation der betreffenden Attribute zugegriffen werden kann. Im obigen Beispiel gilt etwa `fressen_3p:syn:cat = verb`.

Eine Grammatik kann beliebig viele Rollen definieren, die durch Aufzählung ihrer Beschriftungen deklariert werden:

```
SEM # ROOT, AGENT, THEME, PATIENT;
```

Das System erlaubt die Formulierung von unären und binären Bedingungen. Die Form der Bedingungen zeigen zwei Beispiele:

```
{X:SYN} : syn_subj_case : syn_subjekt : 0.05 :
  X.label=SUBJ -> X@syn:case=nom;

{X:SEM, Y:SEM} : sem_label_einmal : sem_init : 0.0 :
   X^id=Y^id  -> X.label!=Y.label;
```

Dabei erscheinen nacheinander die Liste der in Beziehung gesetzten Variablen, der Name der Bedingung, ihre Gruppe, ihre Bewertung sowie die auszuwertende Formel, deren Form der Aussagenlogik ähnelt. Der Ausdruck `X@` bezeichnet den Modifikator der Dependenzbeziehung `X`, der Ausdruck `X^` ihren Modifikanden. Die erste Bedingung besagt also, daß alle Subjekte (genauer: Alle Formen, die einer anderen Form als "SUBJ" untergeordnet sind) im Nominativ stehen. Die zweite Bedingung fordert, daß an jeden Knoten jedes semantische Label nur einmal untergeordnet werden kann.

Von besonderer Wichtigkeit ist die Bewertung der Bedingung. Besonderes Ziel des Demonstrationssystems war die robuste Verarbeitung auch abweichender sprachlicher Strukturen[1]. Um solche Regelverstöße zu benachteiligen, aber doch nicht ganz auszuschließen, ist die Formulierung verletzbarer Bedingungen notwendig. Je niedriger die Bewertung einer Regel, desto strenger soll sie gelten.[2] Die Regel, daß Subjekte im Nominativ stehen, kann damit verletzt werden, wenn die Alternative eine noch schwerwiegendere Verletzung wäre. Formal wird also ein partielles CSP aufgestellt, wobei die beste Lösung diejenige mit der höchsten Bewertung sein soll.

Die Bewertung einer Äußerung ist 1, wenn sie keine Bedingung verletzt, anderenfalls das Produkt der Bewertungen aller verletzten Bedingungen. Verletzt also eine Belegung eine mit 0 bewertete Bedingung, so haben auch alle Lösungen, die sie enthalten, die Bewertung 0. Das Ergebnis ist, daß das System unter allen Umständen versucht, eine solche Lösung zu vermeiden. Eine mit 1 bewertete Bedingung ändert die Bewertung einer Lösung überhaupt nicht und ist daher wirkungslos. Durch die Wahl der Bewertung von Bedingungen kann festgelegt werden, wie wünschenswert die Einhaltung einer Regel ist.

Die Regel, daß Subjekte im Nominativ stehen, ist daher mit Bedacht mit 0.05 bewertet worden, da ihre Verletzung ein grober, aber immerhin möglicher Fehler ist. Die Regel, daß es nur ein Subjekt gibt, sollte mit 0 bewertet werden; die Regel, daß das Subjekt vor dem Verb steht, ist dagegen sehr viel schwächer und könnte mit 0.5 bewertet werden, da ihre Verletzung auch als Stilmittel dienen kann.

Sprachäußerungen können in Form von Wortgraphen eingegeben werden:

[1] Eine der ersten Anwendungen war die Diagnose von Fehlern beim Spracherwerb.

[2] Die Konvention, daß "starke" Bedingungen mit niedrigen Bewertungen versehen werden und umgekehrt, mag zunächst widersinnig erscheinen. Die Verwendung solcher *negativer Evidenz* hat aber den Vorteil, daß die Wirkung einer Bedingung strikt begrenzt wird auf solche Äußerungen, für die sie überhaupt relevant ist. So bestraft etwa eine Bedingung über die Stellung von Objekten nur solche Äußerungen, in denen tatsächlich ein Objekt falsch gestellt ist. In einem System mit *positiver Evidenz* würden dagegen alle Äußerungen belohnt, in denen kein Objekt falsch gestellt ist — selbst wenn gar kein Objekt vorkommt.

```
0 1 wir,
1 2 gehe    0.6,
2 3 in      0.6,
1 3 sehen   0.4,
3 4 die,
4 5 kirche;
```

In diesem Fall konnte der Spracherkenner das Intervall $[1,3]$ der Äußerung nicht eindeutig auflösen und gibt beide Varianten des Satzes zur Verarbeitung weiter.

3.1.2 Algorithmen

Aus einem gegebenen Wortgraphen kann in zwei Schritten ein SCLN aufgebaut werden. Der Wortgraph wird zunächst zum *Lexemgraphen* angereichert. Dabei wird jede Kante des Wortgraphen durch so viele Knoten ersetzt, wie die betreffende Wortform Interpretationen besitzt. Jeder der entstehenden Lexemknoten enthält lexikalische Information über eine der Lesarten. Aus dem Lexemgraphen wird das Constraintnetz aufgebaut, indem zu jedem Lexemknoten und jeder verwendeten Rolle ein Constraintknoten gebildet wird.

Zur Disambiguierung eines SCLN stehen drei Methoden zur Verfügung:

1. *Suche:* Ein SCLN kann mittels *branch & bound*-Suche nach der bestmöglichen Lösung abgesucht werden. Dieses Suchverfahren macht sich den Umstand zunutze, daß die Bewertung einer Teillösung sich im Laufe der Analyse niemals verbessern kann, wenn alle Bedingungen Bewertungen aus dem Intervall $[0,1]$ aufweisen. Sobald also eine vollständige Lösung gefunden wurde, können alle Teilergebnisse verworfen werden, die bereits schlechter bewertet sind. Dadurch werden große Teile des Suchraumes von der Suche ausgeschlossen; die worst-case-Zeitkomplexität bleibt allerdings exponentiell. Existiert keine eindeutige beste Lösung, so werden alle möglichen Lösungen ausgegeben.

2. *constraint propagation:* Es werden Algorithmen zur Herstellung von Knoten- und Kantenkonsistenz angeboten. Im allgemeinen reicht die Herstellung von starker 2-Konsistenz jedoch nicht zur völligen Disambiguierung eines SCLN aus. Überdies können bei der Arbeit mit verletzlichen Bedingungen immer nur wenige Belegungen mit Sicherheit ausgeschlossen werden. Aus diesem Grund wird der Propagationsalgorithmus im weiteren nicht betrachtet.

3. *pruning:* Es werden systematisch Bedingungen ausgewertet und bestimmte Werte von der Verarbeitung ausgeschlossen. Durch pruning werden aber nicht nur die seltenen unmöglichen Werte entfernt, sondern auch solche, deren Bewertung sehr niedrig ist. Ausgehend vom maximal mehrdeutigen Zustand des SCLN wird wiederholt der Wert mit der schlechtesten Bewertung ermittelt und von der weiteren Verarbeitung ausgeschlossen.

   Welcher der Werte der schlechteste ist, läßt sich auf verschiedene Weise definieren. Die einfachste Methode wäre, denjenigen Wert zu wählen, der für sich genommen den kleinsten Konfidenzwert aufweist. Dadurch würden aber die Beziehungen zwischen verschiedenen Worten vernachlässigt. Ein Wert sollte vielmehr auch danach beurteilt werden, wie gut er im Mittel mit den anderen Werten des Problems verträglich ist. Die genaue Auswahlfunktion kann das Verfahren erheblich beeinflussen und ist noch Gegenstand der Forschung. Standardmäßig bewertet `cdg` einen Wert mit der Summe der Quadrate aller Bewertungen von Paaren, denen er angehört.

Bei diesem Verfahren besteht die Gefahr, daß eine Belegung ausgeschlossen wird, die zwar lokal schlecht bewertet ist, aber dennoch einen Teil der insgesamt besten Lösung bildet. Das Verfahren ist also nicht korrekt, dafür aber erheblich effizienter als die vollständige Suche. Insbesondere liefert es die gewünschte anytime-Eigenschaft: Zu jeder Zeit der Analyse ist eine unterspezifizierte Beschreibung der Lösung verfügbar, die Bearbeitung kann jederzeit unterbrochen werden, und die mittlere verbleibende Ambiguität kann als leicht berechenbares Maß für die noch zu leistende Arbeit dienen (vgl. [Schröder97b]).

3.2 Ziel der Änderungen

In [Maruyama91] wurde gezeigt, daß CDG eine echt höhere Ausdrucksmacht besitzen als reine CFG. Ob sie auch mächtiger sind als die kontextsensitiven Sprachen, ist noch unbekannt. Viele andere Parsingverfahren besitzen diese Eigenschaft; die ursprüngliche Definition der Transformationsgrammatik ist sogar Turing-vollständig ([Peters,Ritchie73]). Es ist also nicht sicher, daß CDG tatsächlich dieselben sprachlichen Phänomene beschreiben können wie Transformationsgrammatiken. So können etwa Erscheinungen wie *Position*, einfache *Kongruenz* oder *Rektion* von Wortformen leicht als unäre Bedingungen formuliert werden:

```
// Das Subjekt steht vor dem Verb.
{X:SYN} : syn_subj_pos : syn_subjekt : 0.5 :
  X.label=SUBJ -> distance(X@id, X^id) > 0;

// Verb und Subjekt kongruieren im Numerus.
{X:SYN} : syn_sv_agr_num : syn_subjekt : 0.1 :
  X.label=SUBJ -> X@syn:num=X^syn:num;

// Subjekte stehen im Nominativ.
{X:SYN} : syn_subj_case : syn_subjekt :  0.05 :
  X.label=SUBJ -> X@syn:case=nom;
```

Dagegen ist es schwierig, Kongruenz in beliebigen Strukturen zu garantieren, ohne Bedingungen mit beliebiger Stelligkeit zuzulassen. Auch die Sättigung von Valenzen wirft ein Problem auf; fast alle Sprachen kennen Bedingungen wie die folgende:

"Ein vollständiger Satz muß ein Subjekt besitzen."

In einem Constraint-Netzwerk entspräche diese Bedingung der Forderung, daß es *ein* Wort im Satz geben muß, von dem eine Kante mit dem Label "Subjekt" ausgeht. Um diese Bedingung zu überprüfen, müssen aber alle Variablen im Netz betrachtet werden. Sie hat also eine Stelligkeit, die von der Größe des Netzes abhängt. Wenn aber die Überprüfung einer Bedingung nicht in konstanter Zeit erfolgen kann, erhöht sich die Zeitkomplexität des Parsingverfahrens beträchtlich. Aus diesem Grund wird in CDG gewöhnlich kein Existenzquantor erlaubt.

Wie in Abschnitt 2.2 erwähnt, lassen sich alle CSP auf die Stelligkeit 2 transformieren. Für unser Problem bedeutet das, daß auch eine Existenzforderung sich mit binären Bedingungen beschreiben läßt, wenn man sie auf mehrere Ebenen verteilt. Anstelle der formal nicht erlaubten Bedingung

```
// Ein Verb muß ein Subjekt haben
X:SYN : syn_require_subj : syn_subjekt : 0.1 :
  X@syn:cat=verb -> ∃Y: Y^id=X@id & Y.label=SUBJ;
```

kann die Existenz eines Subjektes durch eine eigene Ebene `AUX` wie folgt gesichert werden:

```
AUX # LINK;

// Alle AUX-Kanten verlaufen vom Verb zum Nomen (oder nach NIL).
{X:AUX} : aux_def : syn_subjekt : 0.0 :
  ~root(X^id) -> X@syn:cat=verb & X^syn:cat=noun;

// Die AUX-Kante ist die Umkehrung einer SYN-Kante.
{X:SYN, Y:AUX} : aux_syn_map : syn_subjekt : 0.0 :
  X@id=Y^id -> X.label=SUBJ & X^id=Y@id;

// Verben modifizieren nicht NIL.
{X:AUX} : aux_require: syn_subjekt : 0.0 :
  X@syn:cat=verb <-> ~root(X^id);
```

Alle Kanten der Ebene `AUX` modifizieren entweder den Knoten NIL, oder sie verlaufen vom Verb zum Nomen. Eine solche Kante muß außerdem gerade die Umkehrung einer `SYN`-Kante sein. Indem man nun noch verbietet, daß Verben den Knoten NIL modifizieren, stellt man sicher, daß es eine Kante mit dem Label `SUBJ` gibt, die das Verb modifiziert.

Diese Methode hat den Nachteil, daß *alle* Wortformen eine zusätzliche Variable erhalten, auch solche, die mit der betreffenden Valenz nichts zu tun haben. Die Belegung dieser Variablen ist freilich unproblematisch, weil sie nur den Wert (LINK,NIL) zulassen. Es erhöht sich allerdings der Verarbeitungsaufwand bei der Auswertung binärer Constraints: Jede der neuen Variablen muß ja alle zweistelligen Bedingungen im Verhältnis zu den alten Variablen erfüllen.

Eine Vergrößerung der Variablenzahl droht auch aus einem anderen Grund: In einer typischen Constraint-Dependenz-Grammatik verwendet die Mehrzahl der verwendeten Rollen nur eine kleine Teilmenge aller definierten Merkmale über Worten. Beispielsweise wird die semantische Rolle alle Oberflächenmerkmale eines Adjektivs ignorieren und nur der semantischen Kategorie Beachtung schenken. Umgekehrt zeigen die verschiedenen Lesarten von Homonymen gewöhnlich identische syntaktische Formen bei verschiedenen semantischen Kategorien. Da aber beim Auftreten mehrdeutiger Wortformen zunächst immer alle Varianten erlaubt werden müssen, multiplizieren sich die Zahl der Rollen und die Länge des Eingabesatzes noch mit der mittleren Mehrdeutigkeit der Arbeitssprache, obwohl im Ergebnis nur jeweils eine der betreffenden Formen benötigt wird.

Diese mittlere Mehrdeutigkeit ist gewöhnlich unannehmbar hoch. Insbesondere das Neuhochdeutsche nimmt in dieser Hinsicht eine unglückliche Zwischenstellung zwischen flektierenden und isolierenden Sprachen ein: Sah beispielsweise das Altgriechische noch über 200 verschiedene Formen zur Konjugation eines regelmäßigen Verbums vor, so sind es im modernen

Deutsch gewöhnlich nur acht, im Englischen vier, im Mandarin nur eine. Auch die deutschen Adjektive zeigen eine hohe mittlere Mehrdeutigkeit: die rechnerisch möglichen 72 Kombinationen von Merkmalen werden von nur sechs Formen realisiert.

In den ursprünglichen Arbeitssprachen für CDG (Japanisch und Englisch) stellt sich dieses Problem nicht, da gewöhnlich *alle* Merkmalsausprägungen in einem Homonym zusammenfallen. Ein Wort wie "the" braucht nicht mit Information über Kasus, Numerus etc. assoziiert zu werden; stattdessen kann man annehmen, daß es jeden Kongruenztest automatisch besteht. Damit kann man auf Bedingungen über die Kongruenz zwischen Artikeln und Nomina gänzlich verzichten. Die Mehrdeutigkeit in Bezug auf Kasus, Numerus und Genus ist hier also *irrelevant*.

Das Deutsche als "leicht flektierende" Sprache erlaubt diesen Weg nicht. Die Merkmale einer Wortform wie "der" können im allgemeinen nicht einfach vernachlässigt, aber auch nicht eindeutig ermittelt werden. Eine mehrdeutige Form wird also für den Parser immer eine lexikalische Entscheidung nötig machen: Die Mehrdeutigkeit ist stark *relevant*.

Solange ausschließlich korrekte Verfahren zur Disambiguierung angewandt werden, bewirkt die Mehrdeutigkeit der Sprache nur einen höheren Verarbeitungsaufwand, der allerdings erheblich sein kann. Noch schwerwiegender ist die Auswirkung auf das Pruning. Die Bindung eines Constraintknotens in einem SCLN bedeutet immer auch eine Entscheidung auf der Ebene der Lesarten. Eine typische Unterordnung wird folgendermaßen dargestellt:

```
stoerche_n(1-2)/SYN --SUBJ--> fressen_3(2-3) [1]
```

Diese Unterordnung gehört der Ebene `SYN` an und trägt das Label `SUBJ`, drückt also eine Subjektanbindung aus. Die Form "Störche" steht dabei im Nominativ und die Form "fressen" in der dritten Person. Die Zahlen hinter den Formen geben an, über welchen Zeitraum sich die Form im Wortgraphen erstreckt; dadurch wird Mehrdeutigkeit vermieden, wenn eine Äußerung mehrere gleichlautende Formen enthält. Die Bewertung ist `[1]`, die Unterordnung genügt also allen unären Bedingungen.

Wenn diese Bindung ausgewählt wird, so wird zugleich festgelegt, daß die Lösung die Nominativ-Lesart des Nomens und die Singular-Lesart des Verbums enthält. Alle Bindungen, die andere Lesarten enthalten, etwa "fressen" als erste Person Plural deuten, werden damit aufgrund des Pfad-Kriteriums sofort unmöglich. Gleichzeitig werden auch alle anderen Formen, die sich im Wortgraph mit einer der beiden ausgewählten Formen zeitlich überschneiden, unmöglich.

Derselbe Mechanismus muß aber auch bei der semantischen Analyse zur Anwendung kommen. So kann etwa die semantische Beziehung zwischen den Wörtern "Störche" und "fressen" nur durch eine Unterordnung dieser Art dargestellt werden:

```
stoerche_n(1-2)/SEM --AGENT--> fressen_3(2-3) [1]
```

Auch bei der semantischen Analyse muß also sofort eine Entscheidung über Lesarten getroffen werden, die aber durch die semantischen Bedingungen nicht unterstützt wird: Eine Verbalisierung desselben Sachverhaltes könnte ja ebensogut im Passiv oder im Singular stehen ("Der Frosch wird vom Storch gefressen"). Wird die Äußerung also zuerst oder konkurrent semantisch untersucht, so muß die Entscheidung über Lesarten zufällig getroffen werden und liefert oft falsche Ergebnisse. Bei rein semantischer Analyse bedeutet das nur, daß die Lösung willkürlich ausgewählte Lexeme enthält. Werden aber später noch syntaktische Bedingungen an die Teillösung gestellt, so kann die falsche Entscheidung dazu führen, daß überhaupt keine Lösung gefunden wird, obwohl eine fehlerfreie Analyse möglich wäre. Wesentlich sinnvoller ist also ein Verfahren, das die Entscheidung über lexikalische Alternativen allein der dafür zuständigen Ebene überläßt. Zum einen verhindert man so ein uninformiertes Beschneiden des Lexemgraphen. Zum anderen können dadurch viele Constraintknoten zusammengefaßt werden, so daß sich die Größe des SCLN verringert.

Will man die Entscheidung über Lexemauswahl z.B. von der semantischen Analyse unabhängig machen, so muß man fordern, daß die semantische Ebene Unterordnungsbeziehungen nicht zwischen Lexemen, sondern zwischen Mengen von Lexemen herstellt. Die oben angeführte AGENT-Beziehung muß also dargestellt werden als

```
stoerche_n/stoerche_a(1-2)/SEM --AGENT--> fressen_1/fressen_3/fressen_inf(2-3) [1]
```

Damit diese Beziehung durch nur eine Unterordnung dargestellt werden kann, muß die Struktur der Unterordnung disjunktiv werden, also für mehrere Formen zugleich gelten. Dasselbe gilt für den Constraintknoten, der diese Unterordnung als einen seiner Werte enthält.

3.3 Maßnahmen

In der ursprünglichen Darstellung der SCLN wurde angenommen, daß nur lexikalisch mehrdeutige Formen durch mehrere Variable repräsentiert werden. Grammatische Alternativen innerhalb einer Wortform bleiben dagegen zunächst unberücksichtigt: Alle Wortformen, die keine äußerlichen Unterschiede aufweisen, werden demselben Knoten zugeordnet. Erst wenn der Wert des differenzierenden Merkmales tatsächlich abgefragt wird, wird der Knoten in mehrere Alternativen aufgespalten. Da diese Operation aufwendig ist, wird der Grammatikschreiber ermutigt, die Bedingungen einer Grammatik so anzuordnen, daß Tests auf Merkmale möglichst spät notwendig werden. Rein strukturelle Bedingungen wie etwa das Verbot von Dependenz-Zyklen sollten also vor sprachspezifischen Bedingungen wie Rektions- oder Kongruenzregeln angewandt werden.

Im vorliegenden `cdg`-System werden dagegen von Anfang an alle möglichen Alternativen explizit aufgestellt. Folgende Überlegungen lassen dies sinnvoll erscheinen:

- Im Englischen herrscht große lexikalische Mehrdeutigkeit: Da Wortarten nicht morphologisch markiert werden, können sehr viele Inhaltswörter Verb, Adjektiv oder Nomen sein. Die grammatische Mehrdeutigkeit kann dagegen zumeist ignoriert werden, da fast alle Ausprägungen eines lexikalischen Wortes zusammenfallen. Die verbleibenden grammatisch mehrdeutigen Formen haben gewöhnlich nur ein unterspezifiziertes Merkmal. Die Aufspaltung von Knoten bleibt also der Ausnahmefall.

  Im Deutschen tritt dagegen nur sehr wenig lexikalische Mehrdeutigkeit auf, aber um so mehr grammatische Mehrdeutigkeit. Von dieser Mehrdeutigkeit sind gewöhnlich mehrere grammatische Merkmale betroffen, und sie kann nicht ignoriert werden. Die Aufspaltung von Knoten würde also zum Normalfall.

- Die Expansion mehrdeutiger Knoten kann vermieden werden, wenn ein Knoten bereits vor der Propagation von der Verarbeitung ausgeschlossen werden kann. Bei systematischer Verwendung verletzbarer Bedingungen kann aber nur selten eine Lesart mit Sicherheit ausgeschlossen werden. In den meisten Fällen wird also alle verfügbare Information auch tatsächlich relevant.

- Die Expansion kann auch dann vermieden werden, wenn die Disambiguierung gelingt, noch bevor alle Bedingungen angewandt wurden. Eine solche frühzeitige Lösung ist aber bei der Analyse fehlerbehafteter Äußerungen selten.
- Sollte im ursprünglichen Ansatz Mehrdeutigkeit allein durch Propagation von Bedingungen aufgelöst werden, so kann im `cdg`-System eine Suche zur Aufzählung aller Lösungen nötig werden. Bei der Tiefensuche werden zur selben Zeit viele Verarbeitungszustände desselben Netzes gespeichert, in denen die Suche nötigenfalls fortgesetzt werden kann. Die Expansion eines Knotens würde also oft nicht nur einmal vorgenommen, sondern an vielen verschiedenen Stellen des Suchraumes. Außerdem müßte in jedem Zustand genau vermerkt werden, welche und wieviele Knoten das Netz enthält.

Es ist also zu erwarten, daß eine große Zahl von Constraintknoten tatsächlich dupliziert werden müßte, und zwar eventuell in mehreren Schritten oder (bei der Tiefensuche) sogar mehrmals in gleicher Weise. Aus diesem Grund wurde das grundsätzliche Verfahren, mehrdeutige Wortformen von Beginn an durch mehrere Knoten zu repräsentieren, beibehalten. Zusammengefaßt werden lediglich solche Knoten, deren Mehrdeutigkeit niemals relevant werden kann. Alle anderen Knoten werden zu Beginn erzeugt und bei der Verarbeitung ggf. als gelöscht markiert.

Zwei wesentliche Datenstrukturen des bestehenden Systems werden erweitert, nämlich die Strukturen “Lexikoneintrag” und “Constraintknoten”.

3.3.1 Lexikoneinträge

Die bisherigen Lexikoneinträge folgten dieser Definition:

⟨Lexikoneintrag⟩	::=	⟨String⟩ ':=' ⟨String⟩ ':' ⟨Wert⟩ ';'
⟨Wert⟩	::=	⟨Zahl⟩ | ⟨String⟩ | '[' ⟨Attributliste⟩ ']'
⟨Attributliste⟩	::=	⟨Attributliste⟩ ',' ⟨Attribut+Wert⟩ | ⟨Attribut+Wert⟩
⟨Attribut+Wert⟩	::=	⟨String⟩ ':' ⟨Wert⟩

Zu diesen Möglichkeiten tritt nun die Disjunktion hinzu:

⟨Wert⟩	::=	'(' ⟨Disjunktion⟩ ')'
⟨Disjunktion⟩	::=	⟨Disjunktion⟩ '|' ⟨Wert⟩ | ⟨Wert⟩

Diese neue Bildungsregel erlaubt es, einfache und geschachtelte Alternativen in den Merkmalsmengen auszudrücken:

```
storch := Storch :
 [ syn:[ cat:noun, pers:3, num:sg, gen:mas, case:(nom|dat|acc) ],
   sem:[ cat:ANIMAL ]
 ];

sich := sich :
 [ syn:[ cat:pron, num:(sg|pl), pers:3, case:(dat|acc) ],
   sem:[ cat:NIL ]
 ];

der := der :
```

```
[ syn:[ cat:def_article,
       ([ num:pl, gen:(mas|fem|neu), case:gen ] |
        [ num:sg, ([ gen:mas, case:nom ] |
                   [ gen:fem, case:(gen|dat) ]) ])
      ],
  sem:[ cat:NIL ]
];
```

Diese Einträge entsprechen drei, vier bzw. sechs verschiedenen Merkmalskombinationen.

Bisweilen hängen in einer mehrdeutigen Form zwei Merkmale zusammen, die nicht durch eine Disjunktion verbunden werden können. So kann etwa die folgende Situation nicht disjunktiv dargestellt werden:

```
tor_gebäude := Tor :                tor_person := Tor :
 [ syn:[ cat:noun, gen:neu ],        [ syn:[ cat:noun, gen:mas ],
   sem:[ cat:BUILDING ]                sem:[ cat:PERSON ]
 ];                                  ];
```

Hier liegen zwei binäre Alternativen vor, die aber nicht unabhängig voneinander sind. Um diese Beziehung über verschiedene Ebenen der Struktur hinweg auszudrücken, ist noch eine weitere Bildungsregel nötig:

⟨Wert⟩ ::= '#' ⟨Zahl⟩ ⟨Disjunktion⟩

Damit ist auch die Beispielsituation erfaßbar:

```
tor := Tor :
 [ syn:[ cat:noun, gen: #1(neu|mas) ],
   sem:[ cat: #1(BUILDING|PERSON) ]
 ];
```

Der Index 1 drückt hier aus, daß beide auftretenden Disjunktionen Teile einer sogenannten *verteilten Disjunktion* sind und folglich immer in gleicher Weise aufgelöst werden müssen. Der disjunktive Lexikoneintrag repräsentiert also nicht vier Wortvollformen, sondern nur zwei.

3.3.2 Constraintknoten

Entsprach bislang jeder Constraintknoten eineindeutig einem Paar aus Rolle und Wortvollform, so kann nun ein Knoten die Funktion mehrerer Wortformen repräsentieren, wenn deren Unterschiede für ihn belanglos sind. Sinnvollerweise wird sich zum Beispiel die semantische Ebene nicht um Kongruenzphänomene kümmern, so daß alle Formen mit der Schreibung "Störche" dieselben semantischen Unterordnungen erlauben.

Um das bestehende System nicht zu sehr zu verändern, wurde entschieden, diese Äquivalenz nur für die Modifikatoren zu erlauben, die Modifikanden aber weiterhin jeweils einzeln zu behandeln. Dadurch wird die Zahl der Constraintknoten so weit reduziert wie möglich (denn jeder Constraintknoten entspricht einem Paar aus unterspezifiertem Modifikand und Rolle), während bei den Werten einige äquivalente übrigbleiben. Da der Aufbau des Constraint-Netzwerks aber von der Zahl der Knoten dominiert wird, liefert diese Lösung zumindest den vollen Effizienzvorteil.

Durch diese Änderung werden an verschiedenen Stellen der Verarbeitung Mengenoperationen nötig, wo zuvor skalare Operationen ausreichten. Da solche Mengenoperationen nicht in konstanter Zeit ausgeführt werden können, besteht prinzipiell wiederum die Gefahr, daß sich die Gesamtkomplexität des Verfahrens erhöht, und zwar abhängig von der mittleren Mehrdeutigkeit der Arbeitssprache. Da aber der Zeitbedarf der Gesamtanalyse vom Aufbau des Constraintnetzes dominiert wird, dessen Komplexität sogar biquadratisch von der Länge der zu analysierenden Äußerung abhängt, kann dieser Mehraufwand durch Reduzierung der Variablenanzahl aufgewogen werden.

Es bleibt die Frage, welche Wortformen als äquivalent zueinander gelten dürfen. Diese Entscheidung kann das Programm ausschließlich aus der eingebenen Grammatik ableiten: Die Bedingungen einer CDG betrachten die Stellung, den Wortlaut und die Merkmale der betreffenden Wortformen. Jeder Constraintknoten ist wiederum einer bestimmten Rolle zugeordnet. In einem Constraintknoten sind nun gerade solche Wortformen äquivalent, die dieselbe Schreibung aufweisen, an derselben Stelle auftreten und in allen relevanten Merkmalen gleich sind. Welche Merkmale für welche Rolle relevant sind, läßt sich eindeutig aus der verwendeten Grammatik ermitteln. Wird etwa folgende Bedingung verwendet:

```
{X:SYN} : syn_subj_case : syn_subjekt : 0.05 :
  X.label=SUBJ -> X@syn:case=nom;
```

so wird für die Knoten der Rolle `SYN` das Merkmal `syn:case` relevant sein. Alle Merkmale, die von einer Rolle niemals überprüft werden, können von ihr ignoriert werden.

3.3.3 Aufbau von Constraint-Netzwerken

Beim Aufbau des Constraintnetzes wird die Äußerung zunächst in einen *Graphemgraphen* umgewandelt. Dieser Graph spiegelt die Struktur des verarbeiteten Wortgraphen wieder, wobei nur die Schriftform (*Graphem*) durch die Liste aller bekannten Lexikoneinträge ersetzt wird, die dieselbe Schreibweise haben. Der Lexemgraph wird dann dadurch aus dem Graphemgraphen gewonnen, daß auch alle grammatisch unterschiedlichen Formen durch eigene Knoten repräsentiert werden. Dabei werden insbesondere alle disjunktive Lexikoneinträge vollständig expandiert. Da dabei möglicherweise die Namen der Lexikoneinträge nicht mehr eindeutig sind, werden aufgrund der differenzierenden Merkmale und der Schreibung der Formen neue Bezeichner für jedes Lexem erzeugt. Der Beispielsatz (2) kann wie folgt im Graphemgraphen repräsentiert werden:

```
T0 :
 0  1  Störche:    Störche_n/Störche_g/Störche_a       0.000000
 1  2  fressen:    fressen_?v/fressen_1v/fressen_3v    0.000000
 2  3  Frösche:    Frösche_n/Frösche_g/Frösche_a       0.000000
```

Die Suffixe `n`, `g` und `a` deuten die Fälle Nominativ, Genitiv und Akkusativ an, die Suffixe `1` und `3` die erste und dritte Person. Der Buchstabe `?` drückt aus, daß dieses Lexem für das Merkmal "Person" keinen Eintrag besitzt.

Im Lexemgraphen sind dagegen weiterhin acht Knoten notwendig:

```
T0 :
 0  1          Störche( 0)       Störche_n 0.000000
 0  1          Störche( 1)       Störche_g 0.000000
 0  1          Störche( 2)       Störche_a 0.000000
 1  2          fressen( 0)      fressen_?v 0.000000
 1  2          fressen( 1)      fressen_1v 0.000000
 1  2          fressen( 2)      fressen_3v 0.000000
 2  3          Frösche( 0)       Frösche_n 0.000000
 2  3          Frösche( 1)       Frösche_g 0.000000
 2  3          Frösche( 2)       Frösche_a 0.000000
```

Prinzipiell wäre es nicht notwendig, die Lexeme im Lexemgraphen vollständig zu disambiguieren; es würde genügen, jene Mehrdeutigkeit zu beseitigen, die bei der geplanten Analyse relevant werden kann. So könnte bei rein semantischer Analyse auch der Lexemgraph mit drei Knoten auskommen. Da aber beim Aufbau des Netzes nicht vorhergesagt werden kann, welche Bedingungen vielleicht einmal darauf angewendet werden, stellt der Lexemgraph vorsichtshalber jede Variante einzeln zur Verfügung. Die vollständige statt nur selektive Expansion fällt für den Verarbeitungsaufwand nicht sehr ins Gewicht, da die Mehrdeutigkeit eines Lexems sich gewöhnlich in Grenzen hält und die Expansion nur einmal zu Beginn der Analyse nötig ist.

Ein Constraintnetz wird nun anhand beider Graphen aufgebaut. Jede Rolle betrachtet nacheinander jeden Knoten des Graphemgraphen. Die Lexeme, die aus diesem Graphemknoten hervorgegangen sind, werden nach den Bedürfnissen dieser Rolle in Äquivalenzklassen partitioniert. Jede der Klassen gilt hernach für die Dependenzanalyse als nur ein Lexem.

Die Syntax-Rolle wird gewöhnlich sehr viele sprachliche Merkmale betrachten und ebenso viele Constraintknoten aufbauen, wie es Lexemknoten gibt. Die anderen Ebenen enthalten gewöhnlich sehr viel weniger Bedingungen und machen also weniger Unterschiede zwischen gleichlautenden Formen. Das Netz enthält also bei gleicher Aussagekraft wesentlich weniger Constraintknoten.

Als Beispiel hier das originale Constraint-Netzwerk für die Äußerung (2) bei Untersuchung von Syntax und Semantik:

```
-------------------------------------------------------------
    id: net0
 state: 0
 nodes:
 0 stoerche_nom(0,1)-SYN 4: SUBJ-fressen_1(1,2)[0.07]
   OBJA-fressen_1(1,2)[0.025] SUBJ-fressen_3(1,2)[1]
   OBJA-fressen_3(1,2)[0.025]
 1 stoerche_nom(0,1)-SEM 6: AGENT-fressen_inf(1,2)[1]
   PATIENT-fressen_inf(1,2)[1] AGENT-fressen_1(1,2)[1]
   PATIENT-fressen_1(1,2)[1] AGENT-fressen_3(1,2)[1]
   PATIENT-fressen_3(1,2)[1]
 2 stoerche_gen(0,1)-SYN 4: SUBJ-fressen_1(1,2)[0.0035]
   OBJA-fressen_1(1,2)[0.025] SUBJ-fressen_3(1,2)[0.05]
```

```
   OBJA-fressen_3(1,2)[0.025]
 3 stoerche_gen(0,1)-SEM 6: AGENT-fressen_inf(1,2)[1]
   PATIENT-fressen_inf(1,2)[1] AGENT-fressen_1(1,2)[1]
   PATIENT-fressen_1(1,2)[1] AGENT-fressen_3(1,2)[1]
   PATIENT-fressen_3(1,2)[1]
 4 stoerche_acc(0,1)-SYN 4: SUBJ-fressen_1(1,2)[0.0035]
   OBJA-fressen_1(1,2)[0.5] SUBJ-fressen_3(1,2)[0.05]
   OBJA-fressen_3(1,2)[0.5]
 5 stoerche_acc(0,1)-SEM 6: AGENT-fressen_inf(1,2)[1]
   PATIENT-fressen_inf(1,2)[1] AGENT-fressen_1(1,2)[1]
   PATIENT-fressen_1(1,2)[1] AGENT-fressen_3(1,2)[1]
   PATIENT-fressen_3(1,2)[1]
 6 fressen_inf(1,2)-SYN 0:
 7 fressen_inf(1,2)-SEM 1: ROOT-NIL[1]
 8 fressen_1(1,2)-SYN 1: VFIN-NIL[1]
 9 fressen_1(1,2)-SEM 1: ROOT-NIL[1]
10 fressen_3(1,2)-SYN 1: VFIN-NIL[1]
11 fressen_3(1,2)-SEM 1: ROOT-NIL[1]
12 froesche_nom(2,3)-SYN 4: SUBJ-fressen_1(1,2)[0.035]
   OBJA-fressen_1(1,2)[0.05] SUBJ-fressen_3(1,2)[0.5]
   OBJA-fressen_3(1,2)[0.05]
13 froesche_nom(2,3)-SEM 6: AGENT-fressen_inf(1,2)[1]
   PATIENT-fressen_inf(1,2)[1] AGENT-fressen_1(1,2)[1]
   PATIENT-fressen_1(1,2)[1] AGENT-fressen_3(1,2)[1]
   PATIENT-fressen_3(1,2)[1]
14 froesche_gen(2,3)-SYN 4: SUBJ-fressen_1(1,2)[0.00175]
   OBJA-fressen_1(1,2)[0.05] SUBJ-fressen_3(1,2)[0.025]
   OBJA-fressen_3(1,2)[0.05]
15 froesche_gen(2,3)-SEM 6: AGENT-fressen_inf(1,2)[1]
   PATIENT-fressen_inf(1,2)[1] AGENT-fressen_1(1,2)[1]
   PATIENT-fressen_1(1,2)[1] AGENT-fressen_3(1,2)[1]
   PATIENT-fressen_3(1,2)[1]
16 froesche_acc(2,3)-SYN 4: SUBJ-fressen_1(1,2)[0.00175]
   OBJA-fressen_1(1,2)[1] SUBJ-fressen_3(1,2)[0.025]
   OBJA-fressen_3(1,2)[1]
17 froesche_acc(2,3)-SEM 6: AGENT-fressen_inf(1,2)[1]
   PATIENT-fressen_inf(1,2)[1] AGENT-fressen_1(1,2)[1]
   PATIENT-fressen_1(1,2)[1] AGENT-fressen_3(1,2)[1]
   PATIENT-fressen_3(1,2)[1]

#nodes: 16/18
#paths: 18
values: #min 1, #max 6, #total 65, average  4.06
#edges: 306
-------------------------------------------------------------
```

Für jeden Constraintknoten sind alle Belegungen aufgeführt, deren Bewertung nicht 0 ist. Da sie alle denselben Modifikator haben, sind jeweils nur das Label und der Modifikand angegeben. Im Knoten 0 sind etwa noch vier Belegungen möglich, von denen nur eine alle unären Bedingungen erfüllt. Deutlich ist zu sehen, daß sich die Varianten von Wortformen und

Labeln im wesentlichen ausmultiplizieren, so daß schon für eine Äußerung von drei Worten 65 Werte erzeugt werden.

Nach der Einführung von disjunktiven Werten und Knoten wird in derselben Situation folgendes Netz aufgebaut:

```
------------------------------------------------------------
    id: net0
 state: 0
 nodes:
 0 Störche_n(0-1)/SYN 4: SUBJ-fressen_1v(1,2)[0.07]
   OBJA-fressen_1v(1,2)[0.025] SUBJ-fressen_3v(1,2)[1]
   OBJA-fressen_3v(1,2)[0.025]
 1 Störche_g(0-1)/SYN 4: SUBJ-fressen_1v(1,2)[0.0035]
   OBJA-fressen_1v(1,2)[0.025] SUBJ-fressen_3v(1,2)[0.05]
   OBJA-fressen_3v(1,2)[0.025]
 2 Störche_a(0-1)/SYN 4: SUBJ-fressen_1v(1,2)[0.0035]
   OBJA-fressen_1v(1,2)[0.5] SUBJ-fressen_3v(1,2)[0.05]
   OBJA-fressen_3v(1,2)[0.5]
 3 Störche_n/Störche_g/Störche_a(0-1)/SEM 6: AGENT-fressen_?v(1,2)[1]
   PATIENT-fressen_?v(1,2)[1] AGENT-fressen_1v(1,2)[1]
   PATIENT-fressen_1v(1,2)[1] AGENT-fressen_3v(1,2)[1]
   PATIENT-fressen_3v(1,2)[1]
 4 fressen_?v(1-2)/SYN 0:
 5 fressen_1v(1-2)/SYN 1: VFIN-NIL[1]
 6 fressen_3v(1-2)/SYN 1: VFIN-NIL[1]
 7 fressen_?v/fressen_1v/fressen_3v(1-2)/SEM 1: ROOT-NIL[1]
 8 Frösche_n(2-3)/SYN 4: SUBJ-fressen_1v(1,2)[0.035]
   OBJA-fressen_1v(1,2)[0.05] SUBJ-fressen_3v(1,2)[0.5]
   OBJA-fressen_3v(1,2)[0.05]
 9 Frösche_g(2-3)/SYN 4: SUBJ-fressen_1v(1,2)[0.00175]
   OBJA-fressen_1v(1,2)[0.05] SUBJ-fressen_3v(1,2)[0.025]
   OBJA-fressen_3v(1,2)[0.05]
10 Frösche_a(2-3)/SYN 4: SUBJ-fressen_1v(1,2)[0.00175]
   OBJA-fressen_1v(1,2)[1] SUBJ-fressen_3v(1,2)[0.025]
   OBJA-fressen_3v(1,2)[1]
11 Frösche_n/Frösche_g/Frösche_a(2-3)/SEM 6: AGENT-fressen_?v(1,2)[1]
   PATIENT-fressen_?v(1,2)[1] AGENT-fressen_1v(1,2)[1]
   PATIENT-fressen_1v(1,2)[1] AGENT-fressen_3v(1,2)[1]
   PATIENT-fressen_3v(1,2)[1]

#solutions(s): 0
#nodes: 10/12
#paths: 18
values: #min 1, #max 6, #total 39, average  3.90
#edges: 132
------------------------------------------------------------
```

Alle Unterordnungen aus dem ersten Netz sind weiterhin möglich und tragen dieselben Bewertungen. Solche Unterordnungen, deren Modifikatoren zueinander äquivalent sind, wurden zu disjunktiven Unterordnungen zusammengefaßt. Dadurch verringert sich die Anzahl der Constraintknoten. So waren etwa im ursprünglichen Netz die drei Constraintknoten 13, 15 und 17 für die semantische Interpretation des Graphems "Frösche" zuständig; nun ist es allein der Knoten 11. Auch die Zahl der Unterordnungen schrumpft von 18 auf 6.

Bei konsequent disjunktiver Darstellung könnte eine Unterordnung eine n-zu-m Beziehung zwischen Lexemen ausdrücken. Wie oben erwähnt, wurde aber nur eine n-zu-1-Beziehung implementiert. Es wird also nur die Anzahl der Constraintknoten auf ein Minimum reduziert, während innerhalb der Knoten noch immer mehrere gleichwertige Werte als verschieden betrachtet werden, hier etwa in Knoten Nr. 3. Jedoch verringert sich auch die Gesamtzahl der Werte merklich.

Eine weitere Veränderung ist zu sehen: Wenn mehrere Varianten einer Wortform im Lexikon disjunktiv gespeichert werden, so tragen sie alle denselben Bezeichner. Dieser Bezeichner ist also nicht mehr eindeutig, wenn eine mehrdeutige Form in mehrere Formen expandiert wird. Deshalb wird jeder expandierten Form automatisch ein neuer Bezeichner zugeordnet, der aus der betreffenden Wortform und den Werten der relevanten Merkmale gebildet wird. So wird etwa das disjunktive Lexem `Frösche` in drei disambiguierte Lexeme expandiert, die sich nur im Merkmal "Kasus" unterscheiden; daher erhalten sie die Bezeichner `Frösche_n`, `Frösche_g` und `Frösche_a` (vgl. Abschnitt 3.3.3).

3.3.4 Pruning von Constraint-Netzwerken

Das Pruning in SCLN bleibt von der Veränderung fast unberührt. Der grundsätzliche Ablauf des Pruning läßt sich vereinfacht wie folgt darstellen:

- bewerte jeden Wert im Constraint-Netzwerk mit einer Funktion f, die seine Verträglichkeit mit anderen Werten mißt
- lösche den am schlechtesten bewerteten Wert
- wenn noch ein Wert gelöscht werden konnte, beginne von vorn

Unabhängig davon, welche Auswahlfunktion gerade verwendet wird, ist die Bewertung einer Belegung nur von der gegenseitigen Verträglichkeit zweier Belegungen abhängig. Diese Werte werden weiterhin den Tabellen an den Kanten des Constraint-Netzwerkes entnommen. Die Zahl und Größe dieser Tabellen kann sich freilich verringern, was aber den Auswahlalgorithmus nicht verändert: Er betrachtet nach wie vor alle Kombinationen der existierenden Belegungen.

Eine Veränderung ergibt sich lediglich, wenn ein Wert gelöscht wird: Im ursprünglichen Pruning-Algorithmus muß darauf geachtet werden, daß nicht der letzte Wert eines Constraintknotens gelöscht wird, da dies der Löschung des gesamten Knotens gleichkäme. Wird aber ein Wert gelöscht, so kann auch das Lexem, das er repräsentiert, gelöscht werden. Andere Werte, die dasselbe Lexem binden, fallen danach in den nächsten Durchgängen automatisch weg, da sie mit 0 bewertet werden.

Diese beiden Operationen werden für disjunktive Werte angepaßt: ein Wert darf nicht gelöscht werden, wenn es zu den von ihm repräsentierten Lesarten keine Alternativen mehr gibt, wenn ihre Löschung also unmittelbar das Pfadkriterium verletzen würde. Wird aber ein disjunktiver Wert gelöscht, so können alle von ihm vertretenen Lesarten zugleich gelöscht werden, da keines von ihnen der Lösung angehören kann.

3.3.5 Suche in Constraint-Netzwerken

In der vorliegenden Implementation von `cdg` werden die Belegungen, die zusammengenommen die Lösung bilden, nacheinander vorgenommen. Ausgehend vom aufgebauten Constraintnetz werden der Reihe nach alle enthaltenen Variablen belegt. Bei jedem Versuch, eine neue Bindung vorzunehmen, muß entschieden werden, ob diese Bindung mit den bereits vorgenommenen Bindungen verträglich ist, d.h. das Pfad-Kriterium und die Bedingungen der Grammatik erfüllt sind. Eine Bindung wird also vorgenommen, wenn

- sie keine mit 0 bewertete Bedingung verletzt und
- der Modifikator und der Modifikand der Dependenz nicht gelöscht sind.

Sobald eine Bindung vorgenommen wurde, liegt fest, daß die beiden beteiligten Formen in der Lösung vorkommen werden. Alle anderen Formen, die sich mit diesen Formen zeitlich überschneiden, werden (in diesem Zweig des Suchbaumes) sofort gelöscht.

Anstatt eine Bindung vorzunehmen, kann das System aber auch versuchen, den Constraintknoten selbst aus dem Netz zu löschen, was der Hypothese entspricht, daß das Lexem des Knotens nicht in der Lösung vorkommt. Dies kann nur dann der Fall sein, wenn

- das Lexem noch nicht in der Teillösung vorkommt und
- das Lexem nicht nötig für den Zusammenhang des Lexemgraphen ist.

Wird ein Constraintknoten gelöscht, so wird auch das von ihm vertretene Lexem aus dem Lexemgraphen gelöscht. Alle anderen Constraintknoten, die dasselbe Lexem vertreten, werden dann in diesem Suchzweig nie wieder berücksichtigt.

Die Lösung des Problemes wird ausschließlich durch diese beiden Schritte aufgebaut. Damit ist sichergestellt, daß die Modifikatoren und Modifikanden der in der Lösung auftretenden Dependenzen genau die Menge der verbleibenden Lexeme sind.

Zur Behandlung disjunktiver Constraintknoten müssen beide Kriterien verändert werden. Eine Dependenzbeziehung kann nun mehrere äquivalente Modifikatoren einem Modifikanden unterordnen. Eine weitere Dependenz kann also genau dann postuliert werden, wenn

- sie keine mit 0 bewertete Bedingung verletzt
- ihr Modifikand nicht gelöscht ist und
- zumindest einer ihrer Modifikatoren nicht gelöscht ist.

Nach der Postulierung einer Dependenz werden alle Varianten des Modifikators, die von dem gebundenen Knoten nicht unterstützt werden, sofort gelöscht.

Wird ein disjunktiver Constraintknoten gelöscht, so bedeutet das, daß alle durch ihn repräsentierten Modifikatoren nicht in der Lösung vorkommen werden. Ein Knoten kann also genau dann gelöscht werden, wenn

- durch das Löschen all seiner Modifikatoren der Lexemgraph nicht in zwei Teile zerfällt
- keiner seiner Modifikatoren schon in der Teillösung benötigt wird.

Dabei wird ein Lexemknoten von der Teillösung "benötigt", wenn er der Modifikand oder der einzige ungelöschte Modifikator einer bereits bestehenden Bindung ist.

3.3.6 Weitere Änderungen

Einige kleinere Änderungen sind an anderen Stellen des Programmes notwendig. Zum Beispiel kann der Term `X@ = Y@` in einer Bedingung nicht mehr durch einen einfachen Vergleich der beiden Modifikatoren entschieden werden; vielmehr bezeichnen `X@` und `Y@` nun Mengen von Lexemen, deren Schnittmenge nicht leer sein darf. Hier ist es allerdings unnötig, tatsächlich beide Mengen abzusuchen; es reicht aus zu prüfen, ob beide Mengen aus demselben Graphem hervorgegangen sind, also bezüglich Schreibung und Position im Satz übereinstimmen.

Jede vollständige Lösung muß außerdem darauf überprüft werden, ob die postulierten Stemmata tatsächlich Bäume, also azyklisch, sind.[3] Dazu wird auf jeder Ebene geprüft, ob von jedem Knoten des Stemmas aus in endlich vielen Schritten die Wurzel erreicht werden kann. Die Frage, ob eine Dependenz direkt einer anderen übergeordnet ist oder nicht, entspricht der Bedingung `X^ = Y@` und muß ebenfalls auf der Ebene der Grapheme entschieden werden. Dies ist eine der Stellen des Programmes, an denen skalare Operationen durch Mengenoperationen ersetzt wurden. Die Überprüfung von Lösungen nimmt aber gegenüber dem Aufbau und der Lösung von Netzwerken nur geringe Zeit in Anspruch.

3.4 Auswertung

Die Semantik des gesamten Programmes, also die Abbildung von Eingabe- auf Ausgabedaten, verändert sich durch die Veränderung in einem unwesentlichen Punkt. Bislang bestimmten die in der Lösung gebundenen Constraintknoten darüber, welche Lesarten für die Wörter des Analysesatzes angenommen wurden: Jeder Constraintknoten repräsentierte ein Lexem, folglich wurden gerade die Lesarten angenommen, die den ungelöschten Constraintknoten entsprachen. Mit der Erfindung disjunktiver Constraintknoten ist nicht mehr gewährleistet, daß jede grammatische Ambiguität durch geeignetes Löschen von Constraintknoten aufgelöst werden kann. Würde etwa Satz 2 rein semantisch analysiert, so gäbe es keinen Grund, etwa die Nominativ-Lesart von "Störche" vorzuziehen. Die Lösung macht in diesem Fall keine Aussage über die Lesart des Graphems.

Diese Veränderung ist jedoch kein wirklicher Verlust, denn auch das bisherige System konnte ja keine informierte Entscheidung darüber treffen, welche Lesart in einer solchen Situation anzunehmen sei. Da es aber formal dazu gezwungen wurde, Lesarten zu disambiguieren, wurden einfach alle Disambiguierungsmöglichkeiten aufgezählt, so daß kombinatorisch viele ähnliche Versionen derselben Lösung entstanden (die natürlich alle gleich bewertet waren). Das disjunktive System gibt stattdessen nur eine Lösung mit unterspezifizierten Lesarten aus, die denselben Sachverhalt kompakt ausdrückt; aus ihr können die disambiguierten Lösungen leicht durch Aufzählen erzeugt werden.

Die Auswirkungen der Veränderungen wurden anhand von zwei verschiedenen Constraint-Dependenz-Grammatiken untersucht: **Grammatik A** wurde zugleich mit dem ursprünglichen `cdg`-System erstellt (vgl. [PJS97]). Ihr Zweck war die Diagnose von Fehlern auf syntaktischer, inhaltlicher und pragmatischer Ebene in einem kleinen Sprachausschnitt des Deutschen, wie er für den grundlegenden Fremdsprachenerwerb angemessen ist. Dabei sollten alle begangenen Fehler klassifiziert und dem Benutzer erklärt werden (zu diesem Zweck wurde

[3] Diese Überprüfung muß vom Programm eigens vorgenommen werden, weil binäre Bedingungen nicht jede Art von Zyklus ausschließen können.

das Programm mit einem Expertensystem gekoppelt, das die verletzten Bedingungen in Hilfemeldungen übersetzte). Verarbeitet werden sollten in gleicher Weise ungrammatische, semantisch unsinnige und auch semantisch gültige, aber in der beschriebenen Welt falsche Äußerungen. Die Grammatik umfaßt neun Ebenen mit 167 Bedingungen, von denen 44 zweistellig sind.

Grammatik B modelliert einen größeren Ausschnitt des Deutschen bis hin zu einfachen Nebensätzen. Der Schwerpunkt liegt dabei auf der syntaktischen Analyse. Die Grammatik umfaßt 10 Ebenen mit 154 Bedingungen, von denen 53 zweistellig sind. Insgesamt sind die Mengen der für jede Ebene relevanten Merkmale größer, zeigen aber weniger Überschneidungen als bei Grammatik A.

Beide Grammatiken wurden für die Analyse einer Auswahl von korrekten und inkorrekten Testsätzen verwendet. Für Grammatik A wurde sowohl eine vollständige Analyse durchgeführt als auch eine nur syntaktische und eine nur semantische Analyse. Für Grammatik B wurde eine vollständige Analyse durchgeführt sowie eine nur syntaktische Analyse. Die Ergebnisse sind nachstehend angegeben:

Zeitbedarf:	Netzaufbau	Suche
A	-73%	-46%
A, Syntax	-20%	-3%
A, Semantik	-52%	-94%
B	-98%	-95%
B, Syntax	-32%	-27%

Platzbedarf:	Netzaufbau	Suche
A	-67%	-55%
A, Syntax	-	-5%
A, Semantik	-83%	-95%
B	-80%	-57%
B, Syntax	-8%	-59%

Bei der Suche erreicht das Programm dabei stets unveränderte Ergebnisse, abgesehen davon, daß statt mehrerer gleichwertiger Lösungen eine unterspezifizierte Lösung geliefert werden kann. Beim Pruning ergibt sich insgesamt eine leichte Verbesserung der Qualität der Lösung; nur bei der Einschränkung von Grammatik A auf ihre Syntax-Rolle ergeben sich sowohl Verbesserungen als auch Verschlechterungen. Diese Ebene betrachtet nämlich alle vorkommenden Merkmale als relevant und zieht daher aus der Möglichkeit disjunktiver Werte keinen Vorteil; da sich aber durch den Expansionsalgorithmus die scheinbare Reihenfolge der gleichlautenden Lexikoneinträge in unvorhersehbarer Weise ändert, kann sich auch das Verhalten des Pruning-Verfahrens in zufälliger Weise ändern.

Die Eingabesprache des Programmes ist dabei eine echte Obermenge der bislang gültigen Eingaben: Für den Benutzer erweitert sich lediglich die Syntax der Lexikoneinträge um eine neue Möglichkeit. Damit können alle bestehenden CDG unverändert verarbeitet werden. Dabei wird aber selbst bei nicht disjunktiver Formulierung des Lexikons der volle Effizienzgewinn erreicht, denn im Graphemgraphen werden *alle* bekannten gleichlautenden Wortformen zusammengefaßt, ob sie nun disjunktiv formuliert sind oder nicht. Der hierfür nötige Aufwand ist in jedem Fall vernachlässigbar gegenüber den eingesparten Operationen beim Aufbau des Constraint-Netzwerkes.

Damit erweist sich die Verwaltung von disjunktiven Lexemen als Idealfall einer nachträglichen Verbesserung: Durch automatische Analyse der Eingabedaten kann die Effizienz des Verfahrens deutlich erhöht werden. Zusätzlich besteht die Möglichkeit, die Eingabe in angemessenerer Form vorzunehmen (als disjunktive Struktur), und die Ausgabe spiegelt diese disjunktive Form ggf. wieder. Gleichzeitig besteht vollständige Abwärtskompatibilität zum bisherigen Formalismus.

Das Ausmaß der erzielbaren Verbesserung ist von verschiedenen Eigenschaften der Eingabedaten abhängig. Zum einen hängt der Effizienzgewinn entscheidend von der relevanten Mehrdeutigkeit der Arbeitssprache ab, die im Deutschen besonders hoch ist. Außerdem können unterschiedliche Formen nur dann zusammengefaßt werden, wenn die unterscheidenden Merkmale nicht für jede Verarbeitungsebene relevant sind. Ideal sind also Lexika, in denen die Information zu einer Wortform in vielen ausdifferenzierten Merkmalen festgehalten wird.

Da beide Grammatiken nur einen stark eingeschränkten Sprachausschnitt behandeln, der eine geringere Mehrdeutigkeit aufweist als die gesamte deutsche Sprache,[4] läßt sich sagen, daß das Verfahren zum Parsen von SCLN durch die geeignete Verwaltung disjunktiver Lexeme deutlich effizienter gemacht werden kann. Insbesondere im Deutschen kann hierdurch die Verarbeitungsdauer in einem Ausmaß gesenkt werden, das die eliminative Verarbeitung komplexer Äußerungen überhaupt erst gangbar macht.

[4] Insbesondere die extrem mehrdeutigen Adjektive sind nur vereinfacht repräsentiert, und auch der eingangs erwähnte Gleichklang von Artikeln, Demonstrativpronomen und Relativpronomen wurde nicht modelliert.

Literaturverzeichnis

[Dworschak97] Dworschak, Manfred. 1997. Fußballbundesliga: Die schwierige Tüftelei um den Spielplan könnte jetzt ein neues Computerprogramm übernehmen. In: Die Zeit Nr. 23, Axel Springer Verlag, Hamburg.

[Freuder82] Freuder, Eugene C. 1985. A sufficient condition for backtrack-free search. In: Journal of the ACM, 29:24–32.

[Harper94] Harper, Mary P. und Randall A. Helzerman. 1994. Managing multiple knowledge sources in constraint-based parsing of spoken language. Technical Report EE 94–16, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907–1285

[Hayes88] Hayes, John P. 1988. Computer Architecture and Organization. McGraw-Hill, New York.

[Helzermann,Harper92] Helzerman, Randall A. und Mary P. Harper. 1992. Semantics and constraint parsing of word graphs. In: Proceedings of the International Conference on Acoustics, Speech and Signal Processing, S. 63–66.

[Herrmann90] Herrmann, T. (1990). Sprechen und Sprachverstehen. In H. Spada (Hrsg.), Lehrbuch Allgemeine Psychologie (S.281-322). Bern: Huber.

[Kempen,Huijbers83] Kempen, G. und Huijbers, P. 1983. The lexicalization process in sentence production and naming: indirect election of words. In: Cognition, 14, S. 185–209.

[Kumar92] Kumar, Vipin. 1992. Algorithms for constraint satisfaction problems: A survey. AI Magazine, 13(1):32–44.

[Levelt89] Levelt, Willem J. M. 1989. Speaking — From Intention to Articulation. MIT Press, Cambridge, Massachussets.

[Maruyama91] Maruyama, Hiroshi. 1991. Constraint dependency grammar. Technical Report RT0044, IBM Research, Tokyo Research Laboratory.

[Menzel94] Menzel, Wolfgang. 1994. Parsing of spoken language under time constraints. In: A. Cohn, Hrsg., Proceedings of the 11th European Conference on Artificial Intelligence, S. 560–564, Amsterdam.

[Peters,Ritchie73] Peters, Stanley und Ritchie, Robert. 1973. On the Generative Power of Transformational Grammars. In: Informational Sciences 6:49–83.

[Pinker96] Pinker, Steven. Der Sprachinstinkt. 1996. Kindler Verlag, München.

[PJS97] Glockemann, Martin et al. 1997. Robuste Sprachanalyse. `http://nats-www.informatik.uni-hamburg.de/~glockema/pjs_robust`

[Russel,Zilberstein91] Russell, Stuart J. und Shlomo Zilberstein. 1991. Composing real-time systems. In: Proceedings of the IJCAI, S. 212–217, Sydney.

[Schröder95] Schröder, Ingo. 1995. Analyse natürlicher Sprache durch Beschränkungserfüllung. Studienarbeit Universität Hamburg.

[Schröder96] Schröder, Ingo. 1996. Integration statistischer Methoden in eliminative Verfahren zur Analyse von natürlicher Sprache. Diplomarbeit Universität Hamburg.

[Schröder97a] Schröder, Ingo. 1997. Syntax der Eingaben in den `cdg`-Parser 0.8. Memo HH-3/97, Projekt DAWAI, Fachbereich Informatik, Universität Hamburg.

[Schröder97b] Schröder, Ingo. 1997. Pruning-Strategien. Memo HH-8/97, Projekt DAWAI, Fachbereich Informatik, Universität Hamburg.

[Schulz98] Schulz, Michael. 1998. Inkrementelles Constraint-Parsing. Studienarbeit Universität Hamburg.

[Stede92] Stede, Manfred. 1992. The Search for Robustness in Natural Language Understanding. In: Artificial Intelligence Reviews, Kluwer Academic Publishers.

[Tesnière59] Tesnière, Lucien. 1959. Éléments de syntaxe structurale, Klincksieck, Paris.

[Tsang93] Tsang, Edward. 1993. Foundations of Constraint Satisfaction, Academic Press Limited, London.

[Weber92] Weber, Heinz Josef. 1992. Dependenzgrammatik: Ein Arbeitsbuch. Gunter Narr Verlag, Tübingen.