Universität des Saarlandes
Fachbereich 14
WS 1997/98

Proseminararbeit

Systemische Generierung mit TAGs

Jochen Bedersdorfer

18.Februar 1998

im Rahmen des Proseminars
"Effizientes Parsing und Generieren
mit Baumadjunktionsgrammatiken"
Wolfgang Wahlster und Tilman Becker

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung in das systemische Framework

Das systemische Framework, dessen Ursprünge auf Artikel von Halliday zurückgehen (z.B. [Halliday 69]), ist ein funktionaler Ansatz zur Beschreibung von Grammatiken. Es steht im Gegensatz zu klassischen Transformationsgrammatiken im Sinne von Chomsky. Dabei ist nicht nur die Repräsentation unterschiedlich, sondern vor allem die Ebene der Betrachtung.

Systemische Linguistik stellt Fragen wie: Wie kann ich erfolgreich kommunizieren? Wie ist die Beziehung zwischen Kontext und benutzter Sprache? Was kann ein Sprecher grammatikalisch tun, um ein bestimmtes Ziel zu erreichen?

Dies sind wichtige Fragen für ein System, das Text generieren soll. Eine Konsequenz dieser Fragen spiegelt sich in Systemen wieder, die den Text als grundlegende linguistische Einheit verstehen und nicht die Sätze eines Textes an sich. Hasan hat sich damit in den späten siebziger Jahren auseinandergesetzt.

Weitere Arbeiten von Hudson, Davey, Berry, Fawcett ziehen sich bis in die achtziger Jahre und belegen, daß das systemische Framework eine große Attraktivität besessen hat und immer noch besitzt, wie die Arbeiten von Kahtleen McCoy, J.Vijay-Shanker und Gijoo Yang zeigen [McCoy, Vijay-Shanker, Yang 1990] [McCoy, Vijay-Shanker, Yang 1992]

Eine neuere Implementierung ist im Projekt PENMAN zu finden, das auch erfolgreich in anderen Systemen eingesetzt wurde, z.B. siehe http://delite.darmstadt.gmd.de/delite/Projects/Speak/

1.1 Allgemeiner Aufbau

Das systemische Framework beschäftigt sich mit der Funktion und Bedeutung von Äußerungen, also ein Ansatz, der von einer semantische Beschreibung ausgeht.

Implementierungen verlangen zumeist eine logische Form als Eingabe, die evtl. lexikalische Einträge enthält wie z.B. in Abbildung 1. Eine systemische funktionale Grammatik an sich ist in mehrere Domänen verteilt und als eine Menge von Netzwerken realisiert. Diese Netzwerke bestehen aus grammatikalischen Auswahlmöglichkeiten, die mit Hilfe von Realisierungsoperatoren eine Struktur aufbauen. Abbildung 2 zeigt ein einfaches systemisches Netzwerk mit seinen Realisierungsoperatoren.

Eine mögliche Implementierung durchläuft dann eine Vielzahl dieser Netzwerke und sammelt dabei die Realisierungsoperatoren auf, die dann ausgeführt werden und eine finale Struktur generieren. Eine anfängliche semantische Repräsentation wird dabei in Komponenten zerlegt, die möglicherweise wieder als Eingabe für Netzwerke dienen, die rekursiv aufgerufen werden.

```
s-act : wh-question
wh-t : n1
tense : past
 proc : "think"
actor : n2 : [ "you" ]
           [proc : "hit"
           [tense: past
     phen: [ actee: n3 = [ "john" ]
           [ actor: n1 = [ type : person, id : quest]
```

Abbildung 1: *Beispieleingabe für einen systemischen Generierer*

Diese Zerlegung hat drei Dimensionen, nämlich das Strukturieren der Repräsentation, die metafunktionale Ebene und der Rang [Matthiessen, R. Kasper 1987]. Der Rang bezeichnet dabei die Zerlegung einer Äußerung in eine Skala von Einheiten wie z.B. Satz – Phrase – Wort – Morphem. Unter Strukturieren der Repräsentation verstehen wir die Organisation innerhalb der verschiedenen Ränge und funktionalen Domänen.

Ein Netzwerk besteht aus einzelnen Systemen, die jeweils einen Auswahlpunkt bieten und verzweigen. An den Verzweigungen werden dann Realisierungsoperatoren aufgesammelt.

Jetzt wollen wir uns die prinzipiellen Teile einer systemischen Grammatik näher anschauen.

1.2 Meta-Funktionale Domänen

Das Systemische Framework unterscheidet drei multi-funktionale Faktoren, die die unterschiedlichen Sichtweisen auf eine Äußerung zusammenfassen.

1.2.1 Interpersonelle Domäne

In dieser Domäne repräsentieren einzelne Netzwerke Fragen über die Interaktion zwischen Sprecher und Hörer, also den Sprechakt.

Beim Design einer systemischen Grammatik wird hier zumeist unterschieden zwischen:

- *Stimmung (mood)* oder Klassifikation des Sprechakts. Ist der zu generierende Satz eine Aufforderung oder eine Frage? Handelt es sich um einen einfachen Aussagesatz? Ein Beispiel für ein dazugehöriges Netzwerk findet sich in Abbildung 2.
- *Identität von Hörer/Sprecher* – Ist ein Hörer involviert?
- *Polarität* Ist der Satz eine Bejahung oder Verneinung einer Aussage? etc.
- *Aufrichtigkeit des Sprechakts* – Ist es notwenig die Aufrichtigkeit des Sprechers zum Ausdruck zu bringen?

Alle diese Aspekte ziehen möglicherweise eine Realisierung durch ein Entscheidungsnetzwerk nach sich.

1.2.2 Ideationale Domäne

Hier wird die Äußerung aus der Sichtweise unserer eigenen Erfahrung betrachtet. Wie ist die Charakteristik des Textes in der aktuellen Situation? Auswahlfragen in diesen Netzwerken können wie folgt aussehen:

- *Transitivität* – Muß ein Prozess geäussert werden, der Akteure hat? Welche Entitäten werden benutzt?
- *Zeitliche Abfolge (time)* – Wie ist die Beziehung zum Zeitpunkt der darzustellenden Situation zum Zeitpunkt der Äußerung? Dies ist wichtig, um die richtige Zeitform zu finden oder unmögliche Zeitformen auszuschliessen.

1.2.3 Textuelle Domäne

Hier wird der Zusammenhang von Äußerungen untersucht und Entscheidungen getroffen, die z.B. Realisierungen von Fokusänderungen nach sich ziehen. Einige der üblichen Netzwerke in dieser Domäne betreffen Fragen wie:

Um was geht es in dem Text? Ändert sich das Thema des Textes während der Äußerung?

1.3 Realisierungsoperatoren

Beim Durchwandern der einzelnen Netzwerke werden verschiedene Realisierungsoperatoren aufgesammelt, die an den Verzweigungen der Teilsysteme annotiert sind. Diese Operatoren steuern unter anderem die Zerlegung einer Formel in Komponenten und die Anordnung von Komponenten in der finalen Struktur.

In dem von Mann und Matthiessen beschriebenen System *Nigel* [Mann, Matthiessen 1983] werden u.a. folgende Realisierungsoperatoren verwendet:

Name	Abk.	Beschreibung
(Insert M)	+M	Füge die Funktion M als Konstituente in die Struktur ein
(Order M N)	M^N	Führe M vor N aus
(Expand M N)	M(N)	Führe Funktion M mit N aus (rekursiver Eintritt in Netzwerk M)
(Conflate M N)	M/N	Verschmelze die beiden Funktionen. Bspw.M = *Subject*, N = *Agent* bei Sätzen, in denen der *Agent* auch das *Subject* ist.
(Preselect M feature)	M: feature	Wähle ein Feature für diese Funktion aus, sodaß es im folgenden erhalten bleibt. Bsp: *Subject:singular*

Diese Definitionen beinhalten keinerlei Informationen darüber, wie eine logische Form im einzelnen zerlegt und auf eine lexikalische Struktur abgebildet wird. Aus linguistischer Sicht sind die einzelnen Netzwerke autark operierende Funktionen. Bei der Implementierung einer systemischen Grammatik wird man gezwungen, diese losen Definitionen zu realisieren, was zu einigen Problemen führen kann, wie wir später noch sehen werden.

Normalerweise wird die Eingabe (eine logische Form) so lange über den Netzwerken traversiert, bis die einzelnen Komponenten den gewünschten Rang (i.d. Regel Wörter) erreicht haben.

Zum besseren Verständnis, ein ...

1.4 Beispiel

Ein Textgenerierungssystem soll mit Hilfe einer systemischen Grammatik folgenden Satz generieren: *Had Sir Christopher Wren been going to build a cathedral ever since his youth?* Wir nehmen an, daß ein Planer sowie eine Wortauswahlkomponente aktiv waren und es jetzt um die Realisierung des Satzes geht.

In einer möglichen Realisierung übernimmt ein Teil des Systems, genannt *Chooser*, die Abarbeitung der einzelnen Netzwerke der meta-funktionalen Domänen (siehe [Matthiessen 1984] für eine detailiertere Darstellung des Beispiels). Der Chooser wird irgendwann den Aspekt des Sprachakts unter die Lupe nehmen in folgendem (Teil-)Netzwerk:

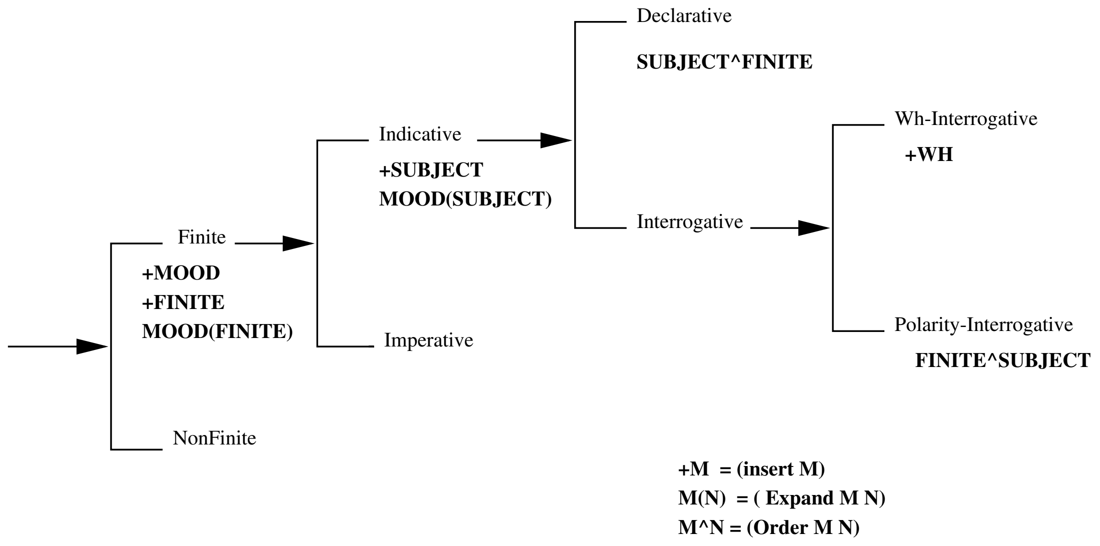

Abbildung 2: *Beispielnetzwerk zu Mood*

Wie man sieht, wird anhand der Entscheidungen, die der Chooser trifft, bestimmt, daß die Äußerung als Wh-Satz realisiert werden soll, wobei das Subject vor dem finiten Verb realisiert wird.

In den folgenden Iterationen wird die Funktion *Mood(Subject)* expandiert und rekursiv in Mood bearbeitet. Mit weiteren Aufrufen werden die restlichen semantischen Funktionen hoffentlich realisiert.

Beim Erzeugen der Zeitform muß vor allem beachtet werden, daß hier eine komplexe Zeitform (past continous) realisiert werden soll. Für diese Entscheidungen wäre dann das hier nicht näher erläuterte Time–Netzwerk verantwortlich.

1.5 Unabhängigkeitskriterium

Wie schon angedeutet ist die Abfolge der Abarbeitung in der Theorie nicht festgelegt. In realen Implementierungen kann aber von einer wichtigen Eigenschaft ausgegangen werden, nämlich dem Unabhängigkeitskriterium.

Man geht dabei von einer schrittweisen Dekomposition der Eingabe aus, indem man folgende Eigenschaften aufrecht erhält:

1. Realisierungsentscheidungen auf einem höheren Rang werden vor solchen auf einem niedrigeren Rang getroffen.

2. Die strukturelle Realisierung der einzelnen Komponenten einer zerlegten Einheit muß unabhängig gehandhabt werden.

Das Unabhängigkeitskriterium ist von Bedeutung für die Erweiterung des Frameworks durch lexikalisierte TAGs, wie wir noch sehen werden.

2 Probleme beim systemischen Generieren

Matthiessen beschreibt in [Matthiessen, R. Kasper 1987] einige Probleme im System *Nigel* [Mann, Matthiessen 1983], das auf dem systemischen Framework beruht. Eingehen will ich nur kurz auf einige Realisierungsprobleme auf Satzebene.

2.1 Diskontinuität

In dem Beispielsatz *They called the meeting off* haben wir eine eingebette Nominalphrase zwischen den Bestandteilen des Verbs *to call off*. Bei einer Realisierung des finiten Verbs ist es jedoch mit den bisherigen Realisierungsoperatoren nicht möglich, *the meeting* einzubetten, da die NP und das finite Verb unabhängig voneinander realisiert werden.

Als mögliche Lösungen werden Operatoren vorgeschlagen, die Komponenten splitten oder Phrasen einbetten können. Ebenso ist eine weitere Aufteilung von *called off* möglich, was aber recht unelegant erscheint.

Eine weitere Lösung eines ähnlichen Problems sehen wir weiter unten bei der Einbindung von TAGs in das Framework.

2.2 Agreement

Obwohl es durch die Operation *Preselect* möglich ist bei den semantischen Funktionen Subject und Finite Agreement zu erreichen, sind nominale Gruppen schwieriger zu handhaben. Beispielsweise sind Pluralformen problematisch, da sie auf unterschiedlichen Rängen zugewiesen werden können. Man betrachte dabei die vier Arten von Plural im Englischen:

- additive plurality: The title and the purport are ...
- plurality in disjunction: The titles or subtitles are ...
- multiplicative pluralibility: The titles are ...
- collective pluralibility: The crew are ...

Matthiessen schlägt als Lösung vor, Operatoren aufzunehmen, die spezifizieren, daß Person und Numerus von Subject und finitem Verb gleich sein müssen, ohne sie näher zu spezifizieren.

Hierbei sieht man die leider notwendige Verquickung rein syntaktischer Information mit semantischer Information und dieses Beispiel schreit geradezu nach der erweiterten Lokalitätsdomäne von TAG, die solche Agreement–Merkmale in den Bäumen kodiert.

3 TAGs im systemischen Framework

Wie wir gesehen haben, gibt es einige Probleme in Implementationen von systemischen Grammatiken, die sich meist auf die ungenügende Kapselung von syntaktischen Strukturen zurückführen lassen. Die meisten dieser Implementationen benutzen atomare Strings als Realisierungseinheiten des untersten Rangs.

Wegen dieser simplen Betrachtungsweise kann z.B. der Satz *It was Mary that John thought Bill liked* nicht ohne weiteres erzeugt werden. Eine natürliche Zerlegung in semantische Teilstücke würde *john thought* und der davon abhängige Satz *bill liked mary* darstellen.

Da in der weiteren Zerlegung diese Teile unabhängig voneinander realisiert werden und man keine Möglichkeit hat, Teilstrings zu mischen, wird man den Satz *John thought it was Mary who Bill liked* erzeugen können, aber nicht unsere ursprüngliche Äußerung.

Eine mögliche Lösung wäre, eine andere Art der Zerlegung zu wählen, was aber das Unabhängigkeitskriterium verletzen würde. Um eben dieses zu erfüllen müssen die gebildeten Strukturen in der Lage sein, alle Abhängigkeiten lokal zu halten. Außerdem sollen sie es ermöglichen, Teiläußerungen in andere einzustreuen.

Diese Überlegungen sowie die oben geschilderten Probleme motivierten die Autoren des zugrundeliegenden Papers [McCoy, Vijay-Shanker, Yang 1990] die Generierung mit systemischen Grammatiken mit TAGs anzureichern.

TAGs [Joshi 1985] erfüllen genau diese Bedingungen wegen der erweiterten Lokalitätsdomäne und der Adjunktionsoperation.

Um TAGs optimal in das systemische Framework einzubetten, werden lexikalisierte, auf Feature–Strukturen basierende TAGs gewählt. Damit kann man u.a. auch Agreement–Probleme in die TAG–Bäume verlagern und einige Netzwerke vereinfachen. Agreement ergibt sich dann bei der Adjunktion von Bäumen (hoffentlich) von selbst.

Lexikalisiert bedeutet, daß es einen lexikalischen Anker gibt, der mit einer Menge von Baumgruppen assoziiert ist. Jeder Baum in einer Baumgruppe hat die selbe logische Form, aber mehrere syntaktische Realisierungen.

Beispielsweise hat die Baumgruppe 'Transitivies Verb' die in Abbildung 3 gezeigten Einträge in dem Beispielsystem der Autoren von [McCoy, Vijay-Shanker, Yang 1990].

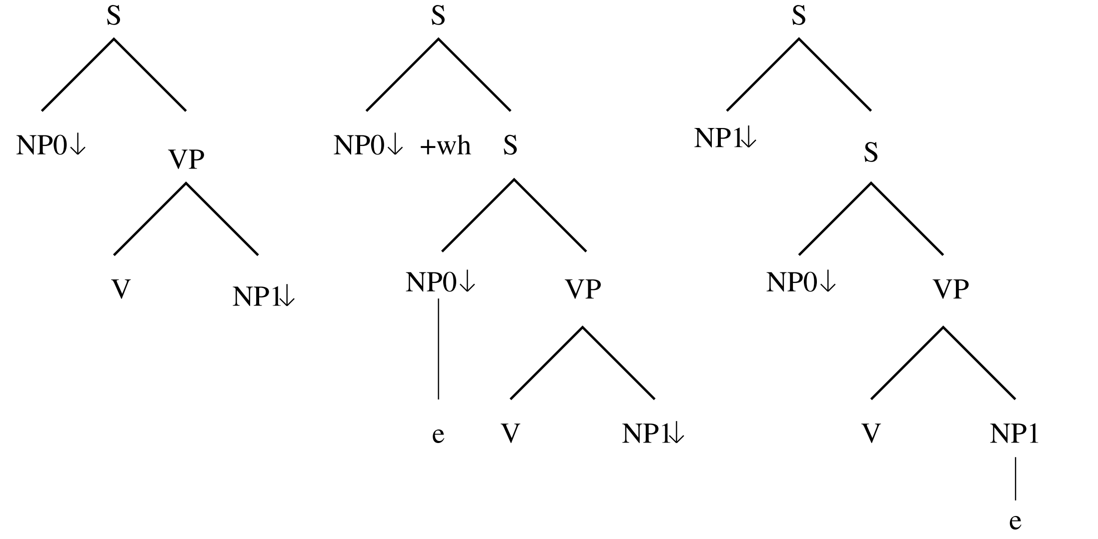

Abbildung 3: *Eine Baumgruppe für transitive Verben*

Eine möglichen Einbettung von TAGs in systemische Grammatiken besteht jetzt darin, die Funktionen, die beim Traversieren der Netzwerke in die systemische Struktur eingefügt werden, mit elementaren Bäumen zu verbinden.

Diese elementaren Bäume bilden dann den Bereich der Anwendung von Realisierungsoperatoren. Die Typen von Operatoren bleiben unverändert, während die Operatoren selbst auf den TAG–Formalismus zugeschnitten werden. Beispiele für Realisierungsoperatoren werden später gegeben.

Der grundlegende Prozess in einer S–TAG muß folgende Dinge berücksichtigen:

1. Meta-funktionale Domänen – Dort werden simultan Strukturen aufgebaut und beispielsweise durch bestimmte Operatoren verschmolzen.

2. Rang – Durch den Wechsel des Rangs werden semantische Strukturen letztendlich in einer Oberflächenform realisiert.

Wie schon in der Einführung angedeutet, werden innerhalb eines Rangs funktionale Einheiten aufgebaut, die durch Wiedereintritt in ein Netzwerk und bei bestimmten Operatoren in einen niedrigeren Rang gelangen. Es ist also zu entscheiden, in welchem Rang letztendlich Bäume ausgewählt werden, die dann durch Substitution oder Adjunktion die Oberflächenstruktur bilden. Um den Prozess der Abarbeitung eines Rangs besser steuern zu können, führen wir den Begriff einer **Region** ein.

Eine Region ist eine Datenstruktur, die Informationen darüber enthält, welche Realisierungsoperatoren benutzt wurden und welche TAG–Bäume ausgewählt wurden innerhalb *eines* Rangs. Beim Expandieren der Funktionen werden weitere Unterregionen erzeugt. Schließlichl führt dies zu einer Auswahl eines elementaren Baums für jede Region auf einem bestimmten Rang. Dies wird der Fall sein, wenn Operatoren das Einfügen eines Lexikoneintrags fordern. Anstatt direkt den Eintrag in die systemische Struktur einzufügen, werden elementare Bäume ausgewählt. Der Lexikoneintrag ist hierbei der lexikalische Anker.

Regionen dienen also als Abgrenzungsbereich für die Realisierungsoperatoren. Alle Operatoren in einer Region sind dann anwendbar auf die Features des lexikalischen Ankers oder den Baum, der dadurch ausgewählt wird.

Die Auswahl einer Baumgruppe für das lexikalische Item und eines Baums innerhalb einer Baumgruppe wird durch neu eingeführte Realisierungsoperatoren gesteuert.

Um zu einer endgültigen Struktur zu kommen, müssen dann nur noch die elementaren Bäume in den einzelnen Regionen mit Adjunktions- oder Substitutionsoperationen kombiniert werden.

3.1 Spezifische Realisierungsoperatoren

Laut Matthiessen [Matthiessen, R. Kasper 1987] können die Operatoren unter den drei Aspekten Struktur, Rang und meta-funktionale Domänen aufgeteilt werden. Für TAGs scheinen nur Rang und Struktur eine Rolle zu spielen. Wichtige Operatoren sind dabei diejenigen, die helfen, Bäume und Baumgruppen auszuwählen.

Hier eine Übersicht der im Artikel [McCoy, Vijay-Shanker, Yang 1990] benutzen Operatoren:

3.1.1 Einfügen eines lexikalischen Ankers

Insert(Lex-Item)

Wenn in einem Netwerk lexikalische Einträge identifiziert wurden (meist auf dem Wort–Rang), wird diese Operation benutzt, um das lexikalische Item und die damit verbundenen Baumgruppen, die als Paare *(feature-structure tree-group-name)* denotiert werden, in die Region einzubringen. Genauer gesagt wird die Baumgruppe nicht an sich eingefügt (Baumgruppen können von mehreren Lexikoneinträgen gleichzeitig benutzt werden), sondern eine Referenz auf dieselbige.

Die Verknüpfung des Lexikoneintrags mit einem ausgewählten Baum aus einer der Baumgruppen geschieht allerdings erst, wenn weitere Operatoren die Auswahl aus Baumgruppen und Bäumen auf eins verringert haben.

3.1.2 Filtern von Baumgruppen in einer Region

Select-Group(feature)

Die Auswahl darüber, welche Baumgruppe für einen lexikalischen Anker relevant ist wird durch aufgesammelte Features bestimmt. Ein relevantes Feature z.B. ist die semantische Funktion, die diese Region erfüllen soll.

Zum Beispiel wird bei der Auswahl von *like* unter anderem auch die Baumgruppe aus Abbildung 3 ausgewählt. Diese wird von Select-Group als endgültige Baumgruppe für diese Region bestimmt, wenn ein Netzwerk entscheidet, daß die semantische Funktion *recipient* realisiert werden muß.

Dieser Operator ähnelt in gewisser Weise der oben beschriebenen *Insert* Funktion im Nigel–System.

3.1.3 Auswahl eines Baums aus der Baumgruppe

Select-Tree(syntaxfeatures)

Bäume innerhalb Baumgruppen bestehen, wie oben gesagt, aus verschiedenen syntaktischen Ausprägungen einer Phrase. In Abbildung 3 sind in der transitiven Baumgruppe Deklarativsatz, Wh-Frage und it-Cleft-Konstruktion realisiert.

Wenn also beim Traversieren eines Netzwerks die Entscheidung fällt, das Objekt zu topikalisieren, wird der Select-Tree Operator benutzt, um einen bestimmten Baum auszuwählen.

Hier sieht man, daß je nach Implementation bestimmte Synchronisationsprobleme auftreten können. Wenn obige Entscheidung auf einem höhren Rang gefällt wurde, bevor für diese Region überhaupt eine Baumgruppe ausgewählt wurde, muß die Evaluation dieses Operators verzögert werden, bis die erforderlichen Bedingungen erfüllt sind.

3.1.4 Zusammensetzen von Bäumen

Wie wir gesehen haben, besteht eine Region aus mehreren Unterregionen, die elementare Bäume ausgewählt haben können. Um diese Bäume zusammensetzen zu können, muß das System sich merken, an welchen Knoten im übergeordneten Baum die Operation **Expand(function, features)** ausgeführt wurde. Diese

Knoten sind die potentiellen Verknüpfungsknoten für Adjunktion oder Substitution.

Details über diese Komposition werden in einem späteren Beispiel erläutert. Von den anderen Realisierungsoperatoren werden nur noch *Conflate* sowie *Preselect* benötigt.

Eine mögliche Realisierung der neuen Operatoren wird in [McCoy, Vijay-Shanker, Yang 1992] vorgeschlagen. Die Operatoren bewirken das Eintreten in ein Netzwerk, das aufgrund der bisher aufgesammelten Features Baum oder Baumgruppen auswählt.

Ein Schritt in einer Implementierung würde also aus der Traversierung der systemischen Netzwerke auf einem Rang und der Traversierung der sogenannten TAG–Netzwerke bestehen.

Die Argumente eines so ausgewählten Baums (Substitutionsknoten etc.) führen zu einem rekursiven Aufruf der systemischen Netzwerke, wiederum gefolgt von einem Durchlauf des TAG–Netzwerks. Die dort erhaltenen Informationen sind dann in einer Unterregion gespeichert.

Wenn alle Argumente realisiert wurden, wir also auf dem untersten Rang angekommen sind und für jeden Lexikoneintrag einen elementaren Baum gefunden haben, folgt die Komposition der einzelnen Bäume.

Wir haben es also mit zwei grundlegenden Prozessen zu tun:

1. Ein *absteigender* Prozess, in dem das systemische Netzwerk benutzt wird, bestimmte Features zu sammeln, die zur Auswahl eines TAG–Baums führen, der den lexikalischen Kopf realisiert und die Argumente einpasst. Dieser Prozess erstellt auch die Unterregionen, in denen die Argumente realisiert werden.

2. Und ein *aufsteigender* Prozess, in dem die Bäume der einzelnen Unterregionen mit dem Baum der Vaterregion kombiniert werden, um die Oberflächenstruktur zu bilden.

Um diese Vorgänge zu verdeutlichen, schauen wir uns am besten ein Beispiel an.

3.2 Beispielableitung

Als Beispiel versuchen wir, die in Abbildung 1 gegebene logische Form in den Satz *Who do you think hit John ?* zu verwandeln.

Im absteigenden Prozess werden, wie in den üblichen Implementierungen einer systemischen Grammatik, verschiedene Netzwerke durchlaufen. Wie wir schon gesehen haben, entscheidet das Mood–Netzwerk darüber, welcher Sprechakt realisiert werden soll. Es muß erkennen, daß es sich in unserem Beispiel zwar um einen Wh-Fragesatz handelt, aber das *befragte* Item nicht Teil der dominierenden Phrase *you think* ist. Daher wird das Netzwerk wohl einen Aussagesatz auswählen.

In rein systemischen Implementationen würde hier der *Order* Operator versuchen, Oberflächenstrings entsprechend des Features $Subject\hat{\ }Finite$ zu ordnen. Durch die Hinzunahme von TAGs haben wir aber ein formaleres grammatisches System, das die Wortordnung eleganter löst.

Während dieser absteigenden Phase wird auch noch das Transitivy–Netzwerk durchlaufen, das mit der Identifikation des Heads oder der Argumente beauftragt ist (vgl. ideationale Domäne). Das Netzwerk würde evaluieren, daß wir es hier mit einem Prozess zu tun haben und deswegen einen Akteur (actor) und ein Phänomen (phen) realisieren müssen. Diese Argumente werden dann durch den *Preselect*–Operator realisiert, der einen rekursiven Netzwerkaufruf initiiert (auf einem niedrigeren Rang) und damit Unterregionen definiert.

Die Features, die während dieser Traversierungen aufgesammelt werden, werden in einem TAG–Netzwerk zur Auswahl von Bäumen und Baumgruppen benutzt.

Insgesamt wird dieser Prozess die Bäume in den Abbildungen 4, 5 und 6 auswählen.

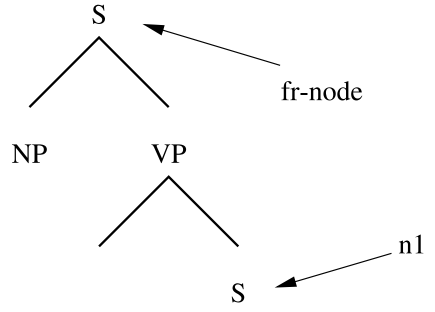

Abbildung 4: *Initialer Baum für Region r1*

Im folgenden aufsteigenden Prozess geht es nun darum, die gefundenen Bäume zu einem Baum zu vereinigen mit den üblichen Operationen Substitution und Adjunktion.

Dazu muß der TAG–Generator festhalten, an welchen Knoten diese Ersetzungen geschehen sollen. Es muß also für jeden Baum eine Abbildung der gramma-

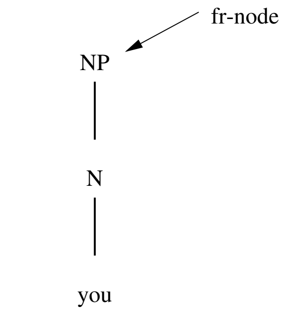

Abbildung 5: *Baum für den actor in Region r2*

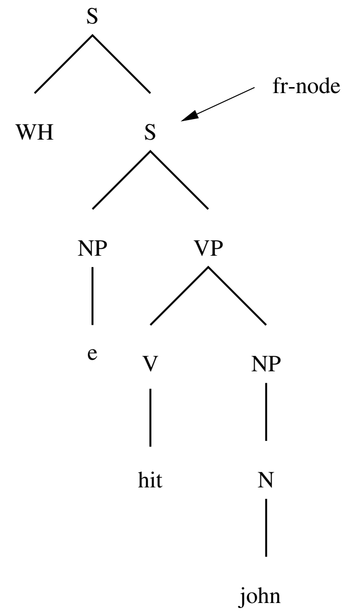

Abbildung 6: *Für Phenomenon ausgewählter Baum in Region r3*

tikalischen Funktion auf Knoten des Baums geben[1]. In Abbildung 4 wird beispielsweise der Knoten *n1* auf die semantische Funktion *phenomenon* abgebildet. Im einfachsten Fall wird dieser Knoten durch den Baum substituiert, der diese Funktion realisiert.

Ein komplexerer Fall tritt auf, wenn der Argumentknoten der Fußknoten eines Auxiliarbaums ist. Wenn also in einer Region ein Auxiliarbaum α ausgewählt wird und in der Unterregion ein Baum β, dann muß α in β adjungiert werden.

[1] Ähnliche Datenstrukturen werden auch in Synchronous–TAGs verwendet, um zwei Knoten mit einem Link zu versehen.

Allerdings bleibt dann zu klären an welchen Knoten in β. Dieser Knoten muß der Kopf der Unterregion sein, die das Argument realisiert. Deswegen hat jeder Baum in einer Region einen Zeiger, genannt *fr-Knoten*, der auf den Knoten eines Baums zeigt, der als Kopf fungiert. Dies muß nicht immer der Wurzelknoten sein, wie man in Abbildung 6 sieht, da hier *hit John* eingebettet ist.

In unserem Beispiel haben wir jetzt alle nötigen Informationen, um den finalen Baum zusammenzustellen. Dazu müssen wir Region r1 in Region r3 adjungieren aufgrund der Tatsache, daß n1 als Fußknoten realisiert wurde. Adjungiert in r3 wird, wie oben erläutert, an dem *fr-Knoten*. Letztendlich erhält man die Struktur in Abbildung 7 als Ergebnis der Generierung der logischen Form als Satz.

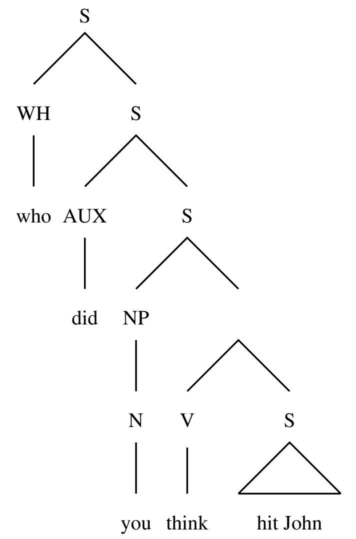

Abbildung 7: *Finale Struktur für: Who did you think hit John?*

Eine schöne Eigenschaft an dieser rang-abhängigen Komposition von Bäumen ist, daß man bei der Erstellung von Unterregionen nur funktionale Informationen nach unten reichen muß. Allerdings kann diese Eigenschaft nicht immer aufrecht erhalten werden, wie man an folgendem Beispiel erkennt:

John tried to win

Das Problem hier liegt in der Tatsache begründet, daß man mit TAGs diesen Satz als Kombination zweiter satzwertiger Komponenten betrachten muß, nämlich *John tried* und *John wins*. Allerdings soll dabei *John* nur einmal realisiert werden, sodaß wir den Baum in Abbildung 8 erhalten.

Wir müssen also diese Bedingung bei der Realisierung der Unterregion *to win* mit angeben. Dies ist aber eine reine syntaktische Beschränkung, die auf dem Verb *to try* basiert. Diese Beschränkung kann auch nicht ohne weiteres in einem

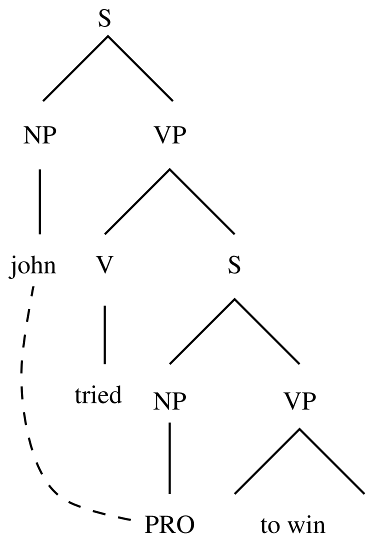

Abbildung 8: *Standardbaum für: John tried to win*

TAG lokalisiert werden, sodaß alle Generierungssysteme, bei denen der Kopf einer Phrase vor den Argumenten realisiert wird, diese syntaktische Information weiterreichen müssen.

4 Vergleiche mit anderen Systemen

Abschließend möchte ich noch die Thesen in [McCoy, Vijay-Shanker, Yang 1992] erläutern, um funktionales Generieren mit TAGs und systemischen Grammatiken mit anderen Methoden zu vergleichen.

4.1 Generieren mit Synchronous TAGs

Shieber und Schabes [Schieber, Schabes 1991] beschrieben das Generieren von Text mit Hilfe von STAG. Synchronous TAGs bestehen aus Baumpaaren, die über Links Verbindungen zwischen den Knoten der Bäume eines Paars herstellen. Angewendet in einem Generator hat man also eine explizite Verbindung einer syntaktischen Struktur mit ihrer korrespondierenden semantischen. Oben genannte Autoren haben in ihrem Artikel einen Algorithmus vorgestellt, der eine Eingabe in Form eines logischen Ausdrucks parst und die benötigten Substitutions– und Adjunktionsoperatoren aufzeichnet. Dieselben Operationen werden dann rückwärts auf die syntaktischen Strukturen angewendet, um einen Satz zu erhalten.

Während der Zusammenhang zwischen logischer Form und syntaktischer Form wichtig für ein Generierungssystem ist, so fehlen feinere Unterscheidungsmöglichkeiten bei Synchronous TAGs. Man kann nicht ohne weiteres aus mehreren syntaktischen Formen die am besten passende aussuchen.

S–TAG hingegen basieren auf der vollen formalen Ausdrucksmöglichkeit von systemischen Grammatiken und können komplexe Entscheidungen treffen. Eventuell lassen sich diese auch in Synchronous TAGs fällen, was aber zu einem Aufblähen der Baummenge führen würde. Ein weiterer Vorteil des S–TAG Systems ist die Trennung von Entscheidungen semantischer Art von Entscheidungen syntaktischer Art. Beide Teilsysteme können weitgehend unabhängig entwickelt werden (Nur weitgehend, weil die Verbindungsnaht zwischen semantischer Seite und der syntaktischen in einem systemischen Netzwerk getroffen werden muß. Der Aufbau eines solchen Netzwerk unterliegt natürlich Änderungen von beiden Seiten der Grammatik).

4.2 Mumble–86

In [Meteer, McDonald, Anderson, Forster, Gay, Huettner, Sibun 1987] beschreiben die Autoren ein Generierungssystem, das ebenfalls auf TAGs basiert, aber bei der Baumauswahl für die Oberflächenrealisierung andere Wege geht. Mumble–86 war einer der am weitesten entwickelten Generatoren, die in einer Vielzahl von Projekten eingesetzt wurden. Vor allem besitzt es ein genaues Maschinenmodell und eine eigene High-Level Sprache.

Mumble–86 erwartet als Eingabe eine sogenannte *L–Spec* (linguistische Spezifikation). Darin sind der zu generierende Inhalt, sowie die Ziele und rhetorischen Mittel enthalten. Ein Beispiel einer solchen L–Spec findet sich in Abbildung 9

Der hervorgehobene Realisierungsoperator kann eine einzelne Funktion sein oder eine Klasse von Operatoren bezeichnen. Mumble–86 benutzt dann einen Lookup–Mechanismus ähnlich eines Lexikons, um über den Realisierungsoperator einen elementaren TAG–Baum zu finden (Dies entspricht im S–TAG System die Traversierung der systemischen Netzwerke).

Wurde ein initialer Baum ausgewählt, arbeitet sich Mumle–86 von links nach rechts durch diesen Baum, um die Argumente zu realisieren. Grammatikroutinen werden dabei aufgerufen, um z.B. Agreement zu garantieren und um Kontexinformationen aufzusammeln, die bei der Realisierung der restlichen Argumente vonnöten sind (long–distance–dependencies).

Damit diese Vorgehensweise Erfolg hat, stellt Mumble–86 einige Anforderungen an die Eingabe. Da Mumble–86 immer bei einem initialen Baum anfängt, muß bei einem eingebetteten Satz die innerste Phrase zuerst abgearbeitet werden. Dies ist jedoch keine übliche Darstellungsweise von Satzkomplementen und stellt dem Generierungsmodul vorangestellte Module vor möglicherweise schwie-

```
                                   realization-function

(general-clause
  :head (CHASE/S-V-O_two-explicit-args
           (general-np
              :head (np-proper-name "Fluffy")
              :accessories (:number singular :gender masculin :person third
                                 :determiner-policy no-determiner))
           (general-np
              :head (np-common-noun "mouse")
              :accessories (:number plural :gender neuter :person third
                                 :determiner-policy initially-indefinite)
              :further-specifications
               ((:attachment-function restrictive-modifier
                  :specififcation (predication-to-be *self*
                                      (adjective "little"))))))
  :accessories (:tense-modal present :progressive :unmarked)))
```

Abbildung 9: *Beispieleingabe für Mumble–86*

rige Entscheidungen.

Ein Satz wie *I wonder who hit John* ist sogar nicht ohne weiteres mit Mumble–86 zu erzeugen, da das Hauptverb *wonder* bei der Auswahl eines geeigneten Baums für die eingebettete Phrase nicht zugreifbar wäre (wegen der Abfolge der Abarbeitung). Das Phänomen, das hier Probleme bereitet ist, daß der Fragesatz eingebettet bleiben muß !

Das S–TAG System stellt keine besonderen Anforderungen an die Eingabeform, da Bäume für Teilphrasen unabhängig gewählt werden und erst durch den aufsteigenden Prozess in eine Oberflächenform gepackt werden.

Ein weiterer problematischer Punkt in Mumble–86 ist die unklare Trennung von syntaktischen und semantischen Entscheidungen. Beide Arten werden in den Realisierungsoperatoren implementiert.

Beim S–TAG System haben wir wiederum eine klare Trennung der semantischen Entscheidungen (systemische Netzwerk der drei Domänen) und der syntaktischen (die TAG–Netzwerke zur Auswahl der Bäume).

5 Fazit

In einem kurzen Fazit möchte ich noch einmal die Stärken und Schwächen des mit TAGs angereicherten systemischen Generierens beleuchten.

Wie wir gesehen haben, haben normale systemische Generierer Probleme mit Long–distancy, Agreement und Diskontinuität. Diese rein syntaktischen Probleme lassen sich hervorragend mit TAGs lösen, da sie eine erweiterte Lokalitätsdomäne besitzen. Durch das Herausnehmen einiger Realisierungsoperatoren in rein systemischen Implementierungen kann eine Trennung von semantischer und syntaktischer Verarbeitung erreicht werden, die der Wartbar– und Erweiterbarkeit zugute kommt.

Bei näherer Betrachtung der Implementierung des Systems wird jedoch zumindest ein Schwachpunkt deutlich:

Obwohl wir eine Trennung von Syntax und Semantikkomponente haben, ist die Verbindungsstelle problematisch in der Handhabung. Sobald Änderungen an den Teilkomponenten stattfinden, muß diese Schnittstelle (die TAG–Netzwerke) überprüft werden. Man muß also immer den Überblick über beide Komponenten haben, um ein korrektes Mapping von semantischen Features auf die Bäume zu gewährleisten.

Nichtsdestotrotz erscheint mir und den Autoren von [McCoy, Vijay-Shanker, Yang 1990] die Einbindung von TAGs eine natürliche Erweiterung systemischer Systeme zu sein.

Literatur

[Halliday 69] M.A.K. Halliday Options and functions in the English clause Brno Studies in English, 1969, 81–88

[Joshi 1985] A.K. Joshi How much contex–sensitivity is necessary for characterizing structural descriptions : Tree Adjoining Grammar In D.Dowty, L. Karttunen & A. Zwicky (Eds.) Natural Language Processing : Theoretical, Computational and Psychological Perspectives

[McCoy, Vijay-Shanker, Yang 1990] K. McCoy and K. Vijay-Shanker and G. Yang Using Tree Adjoining Grammars in the Systemic Framework. In Proc. of the Fifth International Workshop on NL Generation, 1990

[McCoy, Vijay-Shanker, Yang 1992] K. McCoy and K. Vijay-Shanker and G. Yang A Functional Approach to Generatio with TAG ACL '92

[Matthiessen, R. Kasper 1987] Christian Matthiessen, Robert Kasper Representational Isses in Systemic Functional Grammar Repr. Proceedings of the 12th Intern. Systemic Workshop, ISI

[Matthiessen 1984] Christian M.I.M. Matthiessen Systemic Grammar in Computation: The Nigel Case Repr. Proceedings of the First Conference of the European Chapter of ACL, Pisa, 1983, ISI

[Mann, Matthiessen 1983] William C. Mann, Christian M.I.M. Matthiessen Nigel: A Systemic Grammar for Text Generation Publikation der ISI (Information Sciences Institute)

[Meteer, McDonald, Anderson, Forster, Gay, Huettner, Sibun 1987] Meteer, McDonald, Anderson, Forster, Gay, Huettner, Sibun Mumble–86: Design and Implementation COINS Technical Report 87–87

[Schieber, Schabes 1991] S.M. Shieber, Y. Schabes Generation and synchronous tree–adjoining grammars Computational Linguistics 7(4)