Prüfer: Prof. Dr. Christian Rohrer

Betreuer: Dr. Jürgen Wedekind

begonnen am: 15. November 1996

beendet am: 15. Mai 1997

Generierung in unifikationsbasierten Grammatiken

Dipl. Math. Detlef Prescher
Diplomarbeit

Institut für maschinelle Sprachverarbeitung
Universität Stuttgart
Azenbergstraße 12
D 70174 Stuttgart

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfaßt und nur die angegebenen Quellen benutzt zu haben.

_____

(Detlef Prescher)

Inhaltsverzeichnis

Kapitel 1

Einleitung

Unifikationsbasierte Grammatiken sind für die Beschreibung natürlicher Sprache sehr populär geworden. Ein Grund für diesen Erfolg ist, daß unifikationsbasierte Grammatiken *deklarativ* sind, d.h. die Grammatik legt lediglich Fakten über die Sprache fest, die sie beschreibt, macht aber keine Angaben darüber, wie sie für Parsing oder Generierung benutzt werden soll. Aus diesem Grund bewahren deklarative Grammatiken einen hohen Abstraktionsgrad für die Sprachbeschreibung; sie sind einfach zu verstehen, einfach zu erweitern und in anderen Anwendungen zu benutzen.
Da unifikationsbasierte Grammatiken keinen spezifischen Abarbeitungsprozeß vorgeben, liegt die Vorstellung nahe, unifikationsbasierte Grammatiken für *beides*, für Parsing *und* Generierung zu benutzen. Solche Grammatiken werden *reversibel* genannt, genauer: eine *reversible Grammatik* ist eine Grammatik, für die Parsing und Generierung terminieren.

Die Idee einer Grammatik, die sowohl für Parsing und Generierung benutzt werden kann, findet intuitiv sofort Anklang. Sie kann linguistisch, psychologisch und technisch motiviert werden:

- Linguistische Motivation:
  Sprache sollte durch eine einzelne Grammatik beschrieben werden (nicht durch zwei verschiedene Grammatiken für Verstehen und Produktion). Zum Beispiel sollte Deutsch charakterisiert werden, indem ein Linguist die möglichen deutschen Sätze und ihre korrespondierenden Bedeutungen in einer einzelnen Grammatik definiert.
  Durch diese Grammatik sollte die Satz-Bedeutungs-Relation so festgelegt werden, daß für einen gegebenen Satz berechenbar ist, welche Bedeutungen er hat. Und umgekehrt sollte für eine Bedeutung berechenbar sein, welche Sätze diese Bedeutung tragen, d.h. die Grammatik sollte reversibel sein.

- Psychologische Motivation:
  Es ist eine interessante Frage, ob menschliche Sprachproduktion und mensch-

liches Sprachverständnis auf einer einzelnen Grammatik basiert. Es würde erklären, warum Menschen dieselbe Sprache sprechen, die sie verstehen und umgekehrt. In der Praxis verstehen Sprecher jedoch oft Sätze, die sie niemals produzieren würden.

  - Viele dieser Unterschiede können erklärt werden, indem man berücksichtigt, daß Menschen ihnen sonst unklare Äußerungen aufgrund Kontext und Situation verstehen - indem sie Intelligenz, weniger grammatisches Wissen benutzen.
  - Alternativ kann angenommen werden, daß Menschen Sätze verstehen, die sie niemals benutzten, weil sie nicht fähig waren, die Bedeutung dieser Sätze *zu erzeugen* , sehr wohl aber fähig, die Bedeutung dieser Sätze *zu verstehen*. (Dies könnte etwa für die Zuhörer von Einsteins erstem Vortrag über die Relativitätstheorie gegolten haben.) Ebenso ist es denkbar, daß Menschen nicht *willens sind*, Sätze zu benutzen, die sie verstehen. (Zum Beispiel zwecks Betonung sozialer Unterschiede).

- Technische Motivation:

  - Eine Grammatik, die nur für Parsingzwecke geschrieben wird, wird typischerweise etwas übergenerieren, d.h. sie wird Sätzen eine Bedeutung zuweisen, obwohl diese ungrammatisch sind. Analog wird eine Grammatik, die nur für Generierungszwecke geschrieben wird, gewöhnlich untergenerieren, d.h. sie wird für eine gegebene Bedeutung *nicht alle* Sätze generieren, die diese Bedeutung tragen.
  - Ein System für natürliche Sprache muß *konsistent* sein, d.h. alle Sätze, die das System produziert, muß dieses System auch verstehen. Und umgekehrt! Liegt dem System eine reversible Grammatik zugrunde, so ist das System konsistent und der nötige *Konsistenz-Check* darf entfallen.
  - Entwicklung und Pflege einer reversiblen Grammatik sind billiger als die Entwicklung eines Systems, welches jeweils eine seperate Grammatik für Produktion und Verstehen benutzt (, da in einem solchen System zwei Lexika, zwei Phrasenstrukturregelwerke, usw. vorhanden sein müssen.)

Obwohl deklarative Grammatiken - im Prinzip - in reversibler Art und Weise benutzt werden können, müssen hierfür noch einige Probleme gelöst werden. Historisch betrachtet, wurden unifikationsbasierte Grammatiken nur für Parsing benutzt. Versuche, diese Grammatiken auch für Generierung zu benutzen, schlugen fehl.

Es gibt zwei Lösungsansätze, um dieses Dilemma zu beseitigen:

Der erste Ansatz zielt darauf ab, eine Architektur zu schaffen, in der Parsing und Generierung als zwei gleichrangige *Deduktionsprozesse* anszusehen sind, die sich nur durch die Instantiierung gewisser Parameter unterscheiden. Je nach Wahl der Parameter läuft der Deduktionsprozeß als Parsing oder Generierung ab. Desweiteren sind in der Architektur Strategien vorgesehen, die einen angestoßenen Deduktionsprozeß *steuern* können. Ziel ist es, solche Steuerungsstrategien zu finden, sodaß jeder Deduktionsprozeß (egal ob Parsing, Generierung oder Mischform) terminiert.

Der zweite Ansatz ist hiervon sehr verschieden. Er nimmt keine spezielle Steuerungsstrategien für den Parsing- und Generierungsvorgang an, sondern versucht vielmehr Eigenschaften der Grammatiken zu finden, sodaß Parsing und Generierung für Grammatiken, die diese Eigenschaften erfüllen, stets möglich ist.

Vergleichen wir beide Ansätze miteinander, so fällt zunächst beim ersten Ansatz die elegante Idee auf, Parsing und Generierung *nicht* als zwei unterschiedliche Prozesse aufzufassen, sondern im Gegenteil die Gemeinsamkeiten von Parsing und Generierung so zu bündeln, daß sie als zwei Ausprägungen eines einzelnen Deduktionsprozeßes anzusehen sind. Diese Idee stammt von Shieber [6] und wir werden auf sie in Kapitel 4 ausführlich eingehen.

Der Unterschied beider Ansätze wird klar, wenn wir die Idee, Parsing und Generierung als gleichartige Prozesse anzusehen, beiseite tun. Im ersten Ansatz werden Steuerungsstrategien gesucht, sodaß für eine gegebene Unifikationsgrammatik Parsing und Generierung möglich sind.
In der Vergangenheit sind sehr viele Parsingstrategien für unifikationsbasierte Grammatiken entwickelt worden (Earley-Algorithmus, SLR-Algorithmus, usw.). Alle zeichnen sich dadurch aus, daß sie *nicht* auf jede beliebige Grammatik anwendbar sind, sondern nur auf gewisse Grammatik-Klassen. Die Namen dieser Grammatik-Klassen leiten sich meistens direkt aus der gewählten Parsingstrategie ab (SLR-Grammatik, usw.)

VanNoord stellt in seiner Dissertation [5] Generierungsstrategien vor (Semantic-Head-Driven-Generation, Head-Corner-Parser), von denen er zeigt, daß sie ebenfalls nicht auf alle unifikationsbasierten Grammatiken anwendbar sind. Er führt aus, daß unifikationsbasierte Grammatiken gewissen Einschränkungen unterworfen sein müssen, die die Art und Weise betreffen, wie semantische und phonologische Strukturen kombiniert werden. In seinem Ansatz werden die semantischen Strukturen in lexikalischer und kopfgesteuerter Weise aufgebaut, die phonologischen Strukturen werden *nicht* konkatenativ aufgebaut. VanNoord zeigt allerdings nicht, daß in allen gemäß seinem Ansatz gebauten Grammatiken Parsing und Generierung möglich ist.

Shieber [6] kommt zu einem ähnlichen Ergebnis. Auch er ist von der Notwendigkeit von Steuerungsstrategien überzeugt, muß jedoch auch in seinem Ansatz gewisse Grammatik-Restriktionen hinnehmen. Von ihm stammt der Begriff der *semantisch monotonen Grammatiken*. Er formuliert:

> Eine Grammatik ist *semantisch monoton*, wenn für jede grammatische Phrase gilt: die semantische Struktur jeder direkten Subphrase subsummiert einen Teil der semantischen Struktur der ganzen Phrase.

Er führt aus, daß solche Grammatiken Generierung gestatten, wünscht sich aber eine schwächere Bedingung. Wir werden in Kapitel 4 detailliert auf diesen Punkt eingehen.

Fassen wir den ersten Lösungsansatz zusammen, so können wir festhalten, daß unifikationsbasierte Grammatiken, trotz der zusätzlichen Vorgabe gewisser Steuerungsstrategien, Restriktionen erfordern, damit Parsing und Generierung möglich werden. Die Eleganz des ersten Ansatzes zeigt sich vor allem darin, Parsing und Generierung als verschiedene Ausprägungen desselben Deduktionsvorganges anzusehen.

Wir wenden uns nun dem zweiten Ansatz zu. Hier werden keine speziellen Steuerungsstrategien für den Parsing- oder Generierungsprozeß angenommen, sondern es werden Grammatikeigenschaften gesucht, sodaß Parsing und Generierung terminieren. Die bisher wichtigsten Ergebnisse zu diesem Ansatz stammen von VanNoord [5] und Wedekind [8].

VanNoord führt in seiner Disseration [5] den Begriff des *Pfad-Parsings* ein. Hier eine kurze Erklärung: in unifikationsbasierten Grammatiken, besitzt jeder grammatische Satz eine Feature-Struktur. So besitzt z.B. der Satz:

the priest drinks

die Feature-Struktur:

$$\begin{bmatrix} \text{PHON} & \text{„the priest drinks“} \\ \text{SEM} & \begin{bmatrix} \text{PRED} & \text{DRINK} \\ \text{ARG1} & \begin{bmatrix} \text{PRED} & \text{priest} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

PHON-Parsing nennt VanNoord den Prozeß, bei dem zu vorgegebenem PHON-Wert, z.B. „the priest drinks“, alle grammatischen Sätze berechnet werden, deren Feature-Struktur diesen PHON-Wert tragen, also z.B. den Satz:

the priest drinks

Beim PHON-Parsing handelt es sich also um gewöhnliches Parsing.

SEM-Parsing nennt VanNoord den Prozeß, bei dem zu vorgegebenem SEM-Wert, z.B.

$$\begin{bmatrix} \text{PRED} & \text{DRINK} \\ \text{ARG1} & \begin{bmatrix} \text{PRED} & \text{priest} \end{bmatrix} \end{bmatrix}$$

alle grammatischen Sätze berechnet werden, deren Feature-Struktur diesen SEM-Wert tragen, also z.B. wieder den Satz:

the priest drinks

Beim SEM-Parsing handelt es sich also um gewöhnliche Generierung. SEM und PHON sind Attribute. VanNoord verallgemeinert an dieser Stelle und erlaubt anstelle von Attributen ganze Pfade. Zum Beispiel werden beim SEM-PRED-Parsing zu vorgegebenem SEM-PRED-Wert, etwa DRINK, alle grammatischen Sätze berechnet, deren Feature-Struktur den SEM-PRED-Wert DRINK besitzen, also z.B.

the priest drinks
the priest drinks wine
⋮

Wir sehen, daß VanNoord mit der Idee des Pfad-Parsing gewöhnliches Parsing und gewöhnliche Generierung als verschiedene Ausprägungen desselben Prozeßes dargestellt hat.
VanNoord zeigt dann in seiner Dissertation [5], daß für jeden beliebig vorgegebenen Pfad das Pfad-Parsing-Problem nicht entscheidbar ist. In Kapitel 4 werden wir seinen Beweis vorstellen. Durch dieses - negative - Ergebnis beeinflußt, führt er Restriktionen sowohl beim Pfad-Parsing (in Form von Steuerungsstrategien), als auch in der Grammatik (beim Aufbau semantischer und phonologischer Strukturen) ein. In Anbetracht dieses entmutigenden Faktums ist das folgende Ergebnis von Wedekind [9] umso erfreulicher.

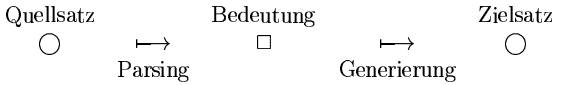
Wie wir eben ausgeführt haben, bedeutet Generierung für VanNoord, daß man zu einen vorgegebenen SEM-Wert alle grammatischen Sätze berechnen kann, die diesen SEM-Wert tragen. Seine wichtige Entdeckung ist allerdings, daß es nicht einmal entscheidbar ist, ob überhaupt *ein* grammatischer Satz existiert, der den vorgegebenen SEM-Wert trägt. Ausserdem gilt dies nicht nur für den SEM-Pfad, sondern für *alle* Pfade einer Feature-Struktur.
Den einzigen Ausweg beschreitet Wedekind, indem er Generierung als Prozeß auffaßt, alle grammatischen Sätze zu berechnen, die eine vorgegebene Feature-Strukur tragen. In [9] zeigt Wedekind, daß es zumindest entscheidbar ist, ob ein grammatischer Satz existiert, der eine vorgegebene Feature-Struktur trägt. Für dieses Resultat müssen keinerlei Restriktionen an die unifikationsbasierte

Grammatik gestellt werden. Ferner zeigt Wedekind, daß im allgemeinen unendlich viele grammatische Sätze existieren, die eine vorgegebene Feature-Struktur tragen (Pumping-Lemma). In Kapitel 4 werden wir diese Resultate ausführlich darstellen.

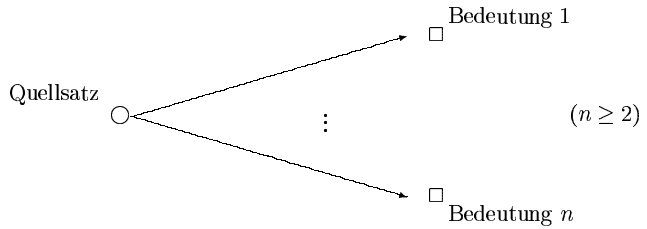
Generierung kann also im allgemeinen *nicht terminieren* - auch wenn man Wedekinds Auffassung von Generierung zugrunde legt. Als Ausweg bietet sich nur an, Restriktionen an die Grammatik anzulegen, die dieses Dilemma beseitigen.

Diesen Themenkreis abrundend, möchten wir auf ein Generierungsproblem hinweisen, das sich bei der Übersetzung ambiger Sätze ergeben hat. Es wird sich zeigen, daß dieses Problem lösbar ist, wenn wir eine Lösung für das Terminierungsproblem beim Generierungsvorgang finden können.

Übersetzung wird oft als Prozeß aufgefaßt, bei dem ein grammatischer Satz der Quellsprache auf seine Bedeutung reduziert wird und aus dieser dann ein grammatischer Satz der Zielsprache berechnet wird:



Bei dieser Vorgehensweise ergeben sich unter anderem immer dann Probleme, wenn der Quellsatz ambig ist, d.h. mehrere Bedeutungen trägt:



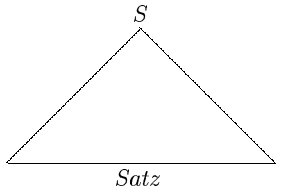
Ein Zielsatz ist nur dann die angemessene - eine ambiguitätserhaltende - Übersetzung des Quellsatzes, wenn er genau die Bedeutungen des Quellsatzes trägt. In ihrer Arbeit [10] haben Wedekind und Kaplan aber gezeigt, daß es nicht entscheidbar ist, ob zu mehreren vorgegebenen Bedeutungen ein grammatischer Satz existiert, der alle diese Bedeutungen trägt. In Kapitel 4 werden wir den sehr kurzen Beweis vorstellen.

Dieses Resultat ist also vollkommen negativ, aber die Lösung des Problems ist eng mit der Lösung unseres vorherigen Problems verknüpft. Angenommen, es wäre möglich, Grammatikeigenschaften zu finden, die garantieren, daß Generierung terminiert, etwa aus dem Grund, weil zu jeder vorgegebenen Bedeutung nur *endlich viele* grammatische Sätze existieren, die diese Bedeutung tragen...

Dann könnte man für jeden grammatischen Satz einer Quellsprache seine Bedeutungen errechnen [1]. Errechnet man dann für eine dieser Bedeutungen alle grammatischen Sätze der Zielsprache, die diese Bedeutung tragen (es sind nur endlich viele), und fährt man damit für alle weiteren Bedeutungen fort, so erhält man insgesamt nur eine endliche Anzahl von grammatischen Sätzen, die als Zielsatz in Frage kommen. Diese können dann nacheinander überprüft werden, ob sie die gewünschte Eigenschaft besitzen, den Quellsatz ambiguitätserhaltend zu übersetzen.

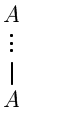
In der vorliegenden Arbeit werden für unifikationsbasierte Grammatiken - insbesondere für LFG - Eigenschaften angegeben, die garantieren, daß zu jeder vorgegebenen Feature-Struktur *nur endlich viele* grammatische Sätze existieren, die diese Feature-Struktur tragen.
D.h.: Generierung - im Wedekindschen Sinne - terminiert, wenn man sich auf Grammatiken beschränkt, die diese Eigenschaften tragen.
Fügt man ferner die bekannten Eigenschaften hinzu, die garantieren, daß Parsing terminiert, so erhält man echte, reversible Grammatiken.
Gemäß der vorhergehenden Bemerkung, kann man mit diesen reversiblen Grammatiken ambiguitätserhaltend übersetzen.

Die in der vorliegenden Arbeit vorgestellten Grammatikeigenschaften basieren auf den in Wedekind [9] vorgestellten Ideen. Wedekind führt in seiner Arbeit den Begriff der *Redundanz* ein:
Kommt in der Satzanlayse

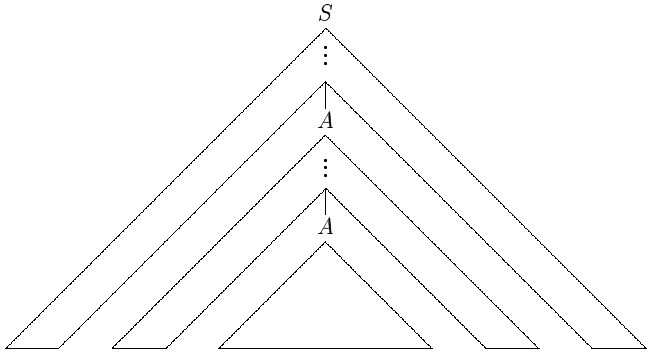


eines grammatisches Satzes eine Rekursion

[1] Vorausgesetzt, daß jeder Satz der Quellsprache nur endlich viele Bedeutungen traegt - d.h. Parsing sollte terminieren ...



vor, d.h.



so nennt Wedekind diese Rekursion *redundant*, wenn sie nichts zum Aufbau der Feature-Struktur des Satzes beiträgt.

Wedekind zeigt dann weiter, daß zu jeder vorgegebenen Feature-Struktur nur endlich viele grammatische Sätze existieren können, die diese Feature-Struktur besitzen *und* keine redundante Rekursionen in ihrer Satzanalyse aufweisen.

Damit ist Wedekind in der Lage, das Generierungsproblem zu entscheiden. Ferner weiß er, daß Generierung in allen Grammatiken terminiert, die nur Satzanalysen *ohne redundante Rekursionen* zulassen.

Solche Grammatiken könnten wir - in Anlehnung an Shiebers Forderung - semantisch monotone Grammatiken nennen. Es handelt sich hierbei um Grammatiken, deren Sätze nur Analysen besitzen, für die entweder die kategoriale Information (für Analyseteile ohne Rekursion) oder die semantische Informationen (für Analyseteile mit *nicht redundanter* Rekursion) monoton anwächst.
Als semantische Information sehen wir hierbei die Feature-Struktur selbst an (Generierung im Wedekindschen Sinne).

VanNoord gibt in seiner Dissertation Generierungsstrategien an, die von *semantischen Köpfen* gesteuert werden. Wir lehnen zwar die Idee von Generierungsstrategien ab, können aber den Begriff des *semantischen Kopfes* sehr wohl benutzen. Unter einem *semantischen Kopf* einer vorgegebenen Satzanalyse verstehen wir

einen Knoten des Analysebaums, an dem eine semantische Information eingeführt wird, die wir sonst an keiner anderen Stelle finden können.

Fordern wir für jede in der Grammatik mögliche Rekursion die Einführung eines semantischen Kopfes, *so kann keine Rekursion redundant sein*, d.h. keine Satzanlyse kann redundante Rekursionen aufweisen. Basierend auf Wedekind's Ergebnis muß Generierung terminieren.

Dies ist die simple Grundidee, die in Kapitel 5 detailliert ausgeführt wird. Schwierigkeiten ergeben sich technisch bedingt dadurch, daß es in unifikationsbasierten Grammatiken durch den Unifikationsmechanismus sehr schwer ist, einen Ort festzulegen, an dem semantische Information (d.h. gewisse Attributwerte) in eine Satzanalyse eingeführt wird.
In LFG werden wir die PRED-Werte als Indikatoren für semantische Köpfe ansehen. Unter Benutzung der Ergebnisse aus Kaplan & Bresnan[2] werden wir zu motivieren versuchen, daß es linguistisch durchaus sinnvoll ist, für alle Rekursionen die Einführung eines PRED-Wertes zu verlangen.

Diese Diplomarbeit ist wie folgt aufgebaut:
Nach der Einleitung werden die formalen Grundlagen von Constraints und unifikationsbasierten Grammatiken dargestellt. Mit diesem Formalismus werden anschließend die schon angesprochenen Arbeiten von VanNoord [5], Shieber [6], Wedekind[9] und Wedekind & Kaplan [10] vorgestellt.
Anschließend werden dann Grammatikeigenschaften vorgeschlagen, die Generierung terminieren lassen. Dies wird in einem etwas allgemeiner gehaltenen Teil für alle unifikationsbasierten Grammatiken - und sehr detailliert für LFG - ausgeführt.

Am Ende dieser Einleitung möchte ich Herrn Prof. Dr. Christian Rohrer danken, der es mir ermöglicht hat, diese mehr theoretisch orientierte Arbeit bei ihm zu schreiben.
Zu großem Dank bin ich Herrn Dr. Jürgen Wedekind verpflichtet, der mich in unifikationsbasierte Grammatiken eingearbeitet hat und der stets ein hilfsbereiter und kompetenter Ansprechpartner war.
Diese Arbeit entstand in weiten Teilen in Budapest. Für die in dieser Zeit gewährte Unterkunft danke ich Agnes Herold.
Karin danke ich dafür, daß sie viele Korrekturen las und sehr viel Geduld mit mir und meiner Arbeit hatte.

Kapitel 2

Die Constraintsprache $\mathcal{L}$

In diesem Kapitel werden wir die Constraintsprache $\mathcal{L}$ definieren. Formeln der Constraintsprache $\mathcal{L}$ bestehen aus den Formeln der Prädikatenlogik erster Stufe mit Gleichheitszeichen, Funktoren, jedoch ohne Relationszeichen. Constraints sind existentiell abgeschlossene Formeln, deren Skopus aus einer Konjunktion von Literalen besteht.
Jede Aussage in $\mathcal{L}$ ist äquivalent zu einer Disjunktion von Constraints. Dies erklärt die Wichtigkeit der Constraints. Die Erfüllbarkeit einer Formel in $\mathcal{L}$ ist entscheidbar, sobald wir gezeigt haben, daß die Erfüllbarkeit eines Constraints entscheidbar ist.
Zu diesem Zweck geben wir einen terminierenden Algorithmus an, mit dem wir einen Constraint in eine (nicht eindeutig bestimmte) Normalform überführen können und zeigen, daß ein Constraint ganau dann erfüllbar ist, wenn eine Normalform erfüllbar ist.
Für die Erfüllbarkeit eines normalisierten Constraints werden wir ein einfaches Kriterium angeben können.

2.1 Der Aufbau der Constraintsprache $\mathcal{L}$

In diesem Abschnitt werden wir die Constraintsprache $\mathcal{L}$ definieren. Wir benutzen hierzu die Prädikatenlogik erster Stufe (ohne Relationszeichen) und lehnen uns an die Darstellung von Prestel [4] an.

Definition: Das *logische Vokabular* der Sprache $\mathcal{L}$ besteht aus den Konnektiven $\neg$ (Negation), $\wedge$ (Konjunktion), dem Allquantor $\forall$, dem Gleichheitszeichen $\doteq$ und den Klammern (,). Das *nichtlogische Vokabular* der Sprache $\mathcal{L}$ besteht aus einer endlichen Menge Con von *Konstanten*, einer endlichen Menge $Attr$ von *Attributen* (oder *Funktionszeichen*) [1] und einer abzählbaren Menge Var von *Variablen* (Con, $Attr$ und Var paarweise disjunkt). Die Menge der *Terme* von $\mathcal{L}$ ist rekursiv definiert:

1. jede Konstante ist ein Term.
2. jede Variable ist ein Term.
3. ist f ein Attribut und t ein Term, so ist $f(t)$ ein Term.
4. keine weitere Zeichenreihe ist ein Term.

Als nächstes bilden wir die *Formeln* von $\mathcal{L}$:

1. sind t und d Terme, so ist $t \doteq d$ eine Formel.
2. ist ϕ eine Formel, so ist $(\neg\phi)$ eine Formel.
3. sind ϕ und ψ Formeln, so ist $(\phi \wedge \psi)$ eine Formel.
4. ist x eine Variable und ϕ eine Formel, so ist $\forall x\phi$ eine Formel.
5. keine weitere Zeichenreihe ist eine Formel.

Eine Formel der Gestalt $t \doteq d$ nennen wir *atomare Formel* oder *positives Literal*. Ein *negatives Literal* ist eine Formel der Gestalt $\neg t \doteq d$.

Abkürzungen:

$(\phi \vee \psi)$	steht für	$\neg(\neg\phi \wedge \neg\psi)$
$(\phi \rightarrow \psi)$	steht für	$\neg(\phi \wedge \neg\psi)$
$(\phi \leftrightarrow \psi)$	steht für	$\neg(\neg\phi \wedge \psi) \wedge \neg(\phi \wedge \neg\psi)$
$\exists x\phi$	steht für	$\neg\forall x\neg\phi$
$t \not\doteq d$	steht für	$\neg t \doteq d$
$t\ f$	steht für	$f(t)$

[1] Prestel benutzt beliebige Mengen Con und $Attr$. Wir beschränken uns (der Einfachheit halber) auf endliche Mengen

Ferner vereinbaren wir die üblichen Bindungs- und Klammerungs-Konventionen.

Im weiteren Verlauf interessieren uns primär nicht sämtliche Formeln der Sprache, sondern nur Formeln, die Constraints sind.

Definition: Eine Formel der Gestalt $\exists x_1 ... \exists x_k \phi_1 \wedge ... \wedge \phi_l$, wobei jedes ϕ_i ein positives oder negatives Literal ist und $Frei(\phi_1 \wedge ... \wedge \phi_l) = \{x_1, ..., x_k\}$ [2], heißt *Constraint*. Einen Constraint schreiben wir abk. als Konjunktion $\phi_1 \wedge ... \wedge \phi_l$ oder als Folge $\phi_1, ..., \phi_l$ oder, wenn es uns nicht auf die Reihenfolge der Literale ϕ_i ankommt, auch als Menge $\{\phi_1, ..., \phi_l\}$ auf. Aus der Notation eines Constraints, als Konjunktion oder Folge, kann offensichtlich wieder die ursprüngliche Formel hergestellt werden.

Beispiel 2.1.1: Für:

$$\begin{aligned} Attr &= \{PRED, SUBJ, OBJ, NUM, PER, AGR, \cdots\}, \\ Con &= \{1st, 2nd, 3rd, sg, pl, john, \cdots\}, \\ Var &= \{x_0, x_1, x_2, x_3, \cdots\} \end{aligned}$$

sind $x_0 SUBJ$ oder $x_0 PRED$ oder $john$ oder x_1 Terme. $x_0 SUBJ \doteq x_1$ oder $x_0 AGR\ NUM \doteq 3rd$ oder $x_0 \doteq x_1$ sind positive Literale. $x_0 PRED \not\doteq john$ ist ein negatives Literal und

$$\neg(x_0 AGR \doteq sg \ \wedge \ x_0 AGR \doteq 3rd)$$

ist zwar eine Formel, jedoch weder ein positives noch ein negatives Literal. Die Folge der Literale:

$$\begin{aligned} x_0 SUBJ &\doteq x_1 \\ x_0 &\doteq x_2 \\ x_1 PRED &\doteq john \end{aligned}$$

ist ein Constraint.

[2] $Frei(\phi)$ bezeichnet die Menge der freien Variablen von ϕ

Formale Beweise in der Constraintsprache $\mathcal{L}$

Für $\mathcal{L}$ wollen wir nun den Begriff eines formalen Beweises definieren.

Definition: Es sei Σ eine Menge von Formeln. Eine Folge $\phi_1, ..., \phi_l$ von Formeln heißt *ein Beweis von ϕ_l aus Σ*, falls für jedes ϕ_i gilt:

1. ϕ_i gehört zu Σ

oder 2. ϕ_i ist ein *logisches Axiom*

oder 3. ϕ_i ist durch Anwendung einer *logischen Regel* auf Folgenglieder mit kleinerem Index entstanden.

Wir sagen dann: ϕ_l ist aus Σ *ableitbar* und schreiben $\Sigma \vdash \phi_l$.

Die *logischen Axiome* bestehen aus den Beispielen aussagenlogischer Tautologien [3], den quantorenlogische Axiomen und den identitätslogischen Axiomen. Die logischen Regeln bestehen aus dem Modus Ponens und der Generalisierungsregel.

Für Details verweisen wir auf Prestel, führen wegen ihrer Wichtigkeit für die Constraintsprache $\mathcal{L}$ hier aber die *identitätslogischen Axiome* auf:

$$\begin{aligned} &(I_1) \quad x \doteq x \\ &(I_2) \quad x \doteq y \rightarrow (x \doteq z \rightarrow y \doteq z) \\ &(I_3) \quad x \doteq y \rightarrow xf \doteq yf \end{aligned}$$

Hierbei sind x, y, z Variablen und f ist ein Attribut.

[3] Ein *Beispiel einer aussagenlogischen Tautologie* in $\mathcal{L}$ erhalten wir dadurch, daß in einer aussagenlogischen Tautologie jede Aussagenvariable durch eine Formel in $\mathcal{L}$ ersetzt wird. (Selbstverständlich sind dabei gleiche Variablen durch gleiche Formeln zu ersetzen.)

2.2 Feature-Graphen

Die Semantik der Prädikatenlokik wird mit Hilfe von Modellen M und Belegungen α definiert [4]:
induktiv über den Term- und Formelaufbau erhält man eine Funktion $[\![.]\!]^{M,\alpha}$, die sämtliche Formeln interpretiert.

Da wir jedoch nicht an allen Formeln der Prädikatenlogik interessiert sind, sondern an Constraints (in Mengen-Notation), so können wir etwas einfacher vorgehen. Es ist üblich, die Semantik von Constraints mit Hilfe von Feature-Graphen, d.h. gelabelten, gerichteten, endlichen Graphen mit Wurzeln [5] zu definieren.

In diesem Abschnitt werden wir die Feature-Graphen, im nächsten Abschnitt die durch Feature-Graphen gegebene Semantik, definieren. Wir beginnen mit einem Beispiel.

Beispiel 2.2.1: Abbildung 2.2.1 zeigt einen Feature-Graphen mit Wurzeln a, b, e (bzw. c, b, e) und Kanten f, g, h, m, n, p.

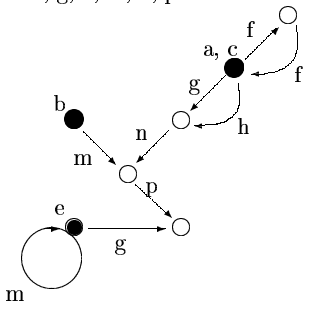


Abbildung 2.2.1

Zunächst fällt an diesem Beispiel auf, daß in einem Feature-Graphen von einem Knoten mehrere Kanten zu einem zweiten bestimmten Knoten führen dürfen. Dies müssen wir bei der Definition eines endlichen, gerichteten Graphen berücksichtigen.

[4] Ein Modell M besteht aus einem Universum U und einer Funktion $[\![.]\!]^M$, welche die Konstanten, Attribute und Relationen interpretiert. Eine Belegung α ist für die Interpretation der Variablen zuständig.

[5] Wir werden Wurzeln später definieren. Wurzeln sind besonders markierte Knoten: jeder Knoten, der selbst keine Wurzel ist, muß von einer Wurzel aus erreichbar sein; keine Wurzel sollte von einer anderen Wurzel aus erreichbar sein.

Definition: Ein *gerichteter, endlicher Graph* $< N, E, source, target >$ besteht aus einer endlichen Menge N von *Knoten*, einer endlichen Menge E von *Kanten*, sowie zweier Funktionen $source : E \to N$ und $target : E \to N$.

Diese beiden Funktionen geben für eine Kante $e \in E$ an, daß e vom Knoten $source(e)$ zum Knoten $target(e)$ führt. Wir wollen noch bemerken, daß die übliche Definition eines gerichteten, endlichen Graphens als Tupel $< N, E >$ mit $E \subseteq N \times N$ unseren Ansprüchen nicht genügt: in einem solchen Graphen kann höchstens eine Kante von einem Knoten zu einem bestimmten zweiten Knoten führen.

Beispiel 2.2.2: Abbildung 2.2.2 zeigt den gerichteten, endlichen Graphen, der dem Feature-Graphen aus Abbildung 2.2.1 zugrundeliegt.

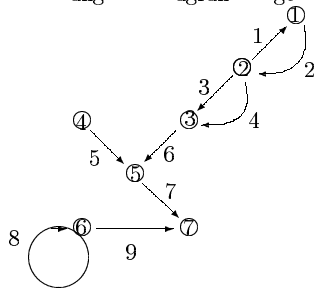


Abbildung 2.2.2

Die Knoten sind von 1 bis 7, die Kanten von 1 bis 9 durchnummeriert. Es gilt: $source(1) = 2, target(1) = 1, source(2) = 1, target(2) = 2, ...$

Definition: Ein *Pfad (von m nach n)* ist eine Folge $e_1, ..., e_k$ von Kanten mit:

$$
\begin{aligned}
& k \geq 1 \\
& target(e_i) = source(e_{i+1}) \text{ für } i = 1, ..., k-1 \\
& m = source(e_1) \\
& n = target(e_k)
\end{aligned}
$$

Mit $Pfad(m, n)$ bezeichnen wir die Menge aller Pfade von m nach n, mit P bezeichnen wir die Menge aller Pfade eines endlichen, gerichteten Graphen.

Die Funktionen *source* und *target* können wie folgt auf P fortgesetzt werden:

$$\begin{array}{rrcl} source: & P & \to & N \\ & e_1,...,e_k & \mapsto & source(e_1) \end{array}$$

$$\begin{array}{rrcl} target: & P & \to & N \\ & e_1,...,e_k & \mapsto & target(e_k) \end{array}$$

Definition: Eine Menge $S \subseteq N$ mit:

$(S1)$ zu jedem $n \in N \setminus S$ gibt es ein $s \in S$ und ein $p \in Pfad(s,n)$

heißt eine *Wurzelmenge* des gerichteten, endlichen Graphen.
Das Tupel $< N, E, source, target, S >$ heißt dann *gerichteter, endlicher Graph mit Wurzeln*.

Im Beispiel 2.2.2 ist etwa $S = \{2,4,6\}$ eine Wurzelmenge. I.A. wird man aus verschiedenen Gründen versuchen, eine möglichst kleine Menge zu finden, die $(S1)$ erfüllt. Dahingehend ist:

$(S2)$ für $s_1, s_2 \in S$ mit $s_1 \neq s_2$ gibt es kein $p \in Pfad(s_1, s_2)$

eine gute Eigenschaft. Wir nennen eine Menge S, die die Eigenschaften $(S1)$ und $(S2)$ besitzt, eine *minimale Wurzelmenge*. Im Beispiel 2.2.2 hat $S = \{2,4,6\}$ auch die Eigenschaft $(S2)$. Aus der Graphentheorie ist bekannt, daß eine minimale Wurzelmenge existiert und ihre Kardinalität (aber nicht die minimale Wurzelmenge selbst) eindeutig bestimmt ist.

Wir kommen nun zu gelabelten Graphen. Die Intention ist, daß wir die Knoten und Kanten eines Graphen mit Namen versehen möchten. Wie Abbildung 2.2.1 zeigt, soll jede Kante des Graphen genau einen Namen, jede Wurzel aber wenigstens einen Namen tragen (*obligatorische Labelung*). Die restlichen Knoten des Graphen dürfen, müssen aber nicht, Namen tragen (*fakultative Labelung*). Sei also L_N eine endliche Menge (von Knotennamen) und L_E eine endliche Menge (von Kantennamen), sowie $label_N$ und $label_E$ zwei Funktionen mit:

$$(N0) \qquad label_N : N \to \wp(L_N)$$
$$(E0) \qquad label_E : E \to L_E$$

Es ist offensichtlich, daß wir mittels eines Namens auf den Träger des Namens referieren wollen. Für die Knoten bedeutet dies, daß ihre Label paarweise disjunkt sein müssen.

$$(N1) \qquad label_N(n_1) \cap label_N(n_2) = \emptyset \quad \text{für} \quad n_1 \neq n_2 \in N$$

Entsprechendes soll für die auslaufenden Kanten eines Knoten gelten:

$$(E1) \qquad label_E(e_1) \neq label_E(e_2) \quad \text{für} \quad e_1 \neq e_2 \in E,\ source(e_1) = source(e_2)$$

Ausserdem haben wir noch nicht aufgeschrieben, daß jede Wurzel wenigstens einen Namen trägt:

$$(N2) \qquad label_N(s) \neq \emptyset \quad \text{für} \quad s \in S$$

Bevor wir allen Knoten $n \in N$ Namen zuweisen, bemerken wir noch, daß sich $label_E$ auf die Menge aller Pfade P fortsetzen lässt:

$$\begin{array}{rrcl} label_E: & P & \to & L_E^+ \\ & e_1,...,e_k & \mapsto & label_E(e_1),...,label_E(e_k) \end{array}$$

Induktiv über die Pfadlänge können wir leicht zeigen, daß $label_E$ eine $(E1)$ analoge Eigenschaft behält:

$$(E1') \qquad label_E(p_1) \neq label_E(p_2) \quad \text{für} \quad p_1 \neq p_2 \in P,\ source(p_1) = source(p_2)$$

Mittels dieser Eigenschaften sind wir in der Lage, allen Knoten $n \in N$ Namen so zuzuweisen, daß jeder Name auf höchstens einen Knoten des Graphen referiert. Ist $n \in S$, so trägt n bereits Namen; ist $n \in N \setminus S$, so liefert $(S1)$ ein $s \in S$ und ein $p \in Pfad(s,n)$, welche beide Namen tragen. Daher ist die folgende Definition sinnvoll:

Definition: Die Funktion $label_F : N \to \wp(L_N \times L_E^*)$ mit:

$$\begin{array}{rl} label_F(n) = \{\ l \in L_N \times L_E^* \mid & (i)\ \ l \in label_N(n) \quad \text{oder} \\ & (ii)\ \ l = l_0\pi \text{ und } \exists m \in N,\ p \in Pfad(m,n): \\ & \quad l_0 \in label_N(m) \\ & \quad \pi = label_E(p)\ \} \end{array}$$

heißt die *Labelfunktion* eines Feature-Graphen F.

Offensichtlich gilt für alle $n \in N$:

$$label_F(n) \supseteq label_N(n)$$

Wir zeigen nun, daß jeder Knotenname auf höchstens einen Knoten des Graphen referiert, d.h.:

$$(N1') \qquad label_F(n_1) \cap label_F(n_2) = \emptyset \quad \text{für} \quad n_1 \neq n_2 \in N$$

Beweis: Sei $n_1 \neq n_2 \in N$ und $l \in label_F(n_1) \cap label_F(n_2)$. Ist (i) $l \in label_N(n_1) \cap label_N(n_2)$, so liefert $(N1)$ einen Widerspruch. Sei also (ii) $l = l_0\pi$. Dann

gibt es $m_1, m_2 \in N$ und $p_1 \in Pfad(m_1, n_1)$, $p_2 \in Pfad(m_2, n_2)$, sodaß $l_0 \in label_N(m_1) \cap label_N(m_2)$ und $\pi = label_E(p_1) = label_E(p_2)$. Wegen $(N1)$ gilt zunächst $m_1 = m_2$, insbesondere also $source(p_1) = source(p_2)$. Wegen $(E1')$ gilt dann $p_1 = p_2$. Somit: $n_1 = target(p_1) = target(p_2) = n_2$. Widerspruch $\diamondsuit$

Lemma 2.2.1: Sei $t \in L_N \times L_E^*$ und $\ell \in L_E^+$. Dann gilt für alle $n \in N$:

$$\begin{array}{ll} t\ell \in label_F(n) & \text{gdw.} \\ \exists m \in N,\ p \in Pfad(m,n):\ \ t \in label_F(m),\ \ \ell = label_E(p) & \end{array}$$

Beweis: „$\Leftarrow$" ist trivial. „$\Rightarrow$": Sei $n \in N,\ \tau = t\ell \in label_F(n)$. Ist $t \in L_N$, so liefert die Definition von $label_F(n)$ direkt das Gewünschte.
Sei also $t = t_0\pi$ (mit $t_0 \in L_N,\ \pi \in L_E^+$). Die Definition von $label_F(n)$ liefert hier ein $m_\tau \in N,\ p_\tau \in Pfad(m_\tau, n)$ mit:

$$\begin{array}{l} t_0 \in label_N(m_\tau) \\ \pi\ell = label_E(p_\tau) \end{array}$$

Seien $p_t, p \in P$ mit:

$$\begin{array}{ll} p_\tau = p_t p & \text{(insbesondere: } target(p_t) = source(p)\text{)} \\ \pi = label_E(p_t) & \\ \ell = label_E(p) & \end{array}$$

Dann gilt zunächst: $target(p) = target(p_\tau) = n$.
Ferner gilt für $m = target(p_t)$:

$$\begin{array}{ll} t \in label_F(m) & \text{(denn: } source(p_t) = source(p_\tau) = m_\tau, \\ & \text{d.h. } p_t \in Pfad(m_\tau, m) \quad \text{)} \\ m = source(p) & \end{array}$$

Insgesamt also: $p \in Pfad(m,n)$ $\diamond$

Definition: Ein *gelabelter, gerichteter, endlicher Graph mit Wurzeln* oder kurz *Feature-Graph* $< N, E, source, target, S, L_N, L_E, label_N, label_E >$ besteht aus einem gerichteten, endlichen Graphen mit Wurzeln $< N, E, source, target, S >$, einer endlichen Menge L_N von *Knotennamen*, einer endlichen Menge L_E von *Kantennamen* und zwei *Labelfunktionen* $label_N$ und $label_E$ mit Eigenschaften $(N0), (N1), (N2), (E0), (E1)$.

2.3 Die Semantik der Constraintsprache $\mathcal{L}$

Die Semantik von Constraints wird mit Hilfe von Feature-Graphen definiert. Bevor wir uns der formalen Definition zuwenden, schauen wir uns Constraints in der üblichen Semantik an. Offensichtlich gilt:

$$\begin{array}{rlll} \exists M: & M & \models \exists x_1 ... x_k \phi_1 \wedge ... \wedge \phi_l & \text{gdw.} \\ \exists M, \alpha: & M, \alpha & \models \phi_1 \wedge ... \wedge \phi_l & \text{gdw.} \\ \exists M, \alpha: \forall i \in \{1, ..., l\}: & M, \alpha & \models \phi_i & \end{array}$$

Unter dem Gesichtspunkt *Erfüllbarkeit* müssen daher weder die $\exists$-Quantoren, noch die Reihenfolge der Literale eines Constraints berücksichtigt werden. (Die Mengen-Notation eines Constraints ist daher in diesem Fall am angemessensten.) In der folgenden Definition weisen wir Termen und Formeln aus $\mathcal{L}$ mit Hilfe eines Feature-Graphen F eine Interpretation $[\![.]\!]^F$ zu. Für Terme wird dies (der Einfachheit wegen) *nicht* induktiv über ihren Aufbau geschehen. Für Formeln verfahren wir in der gewohnten Weise induktiv über ihren Aufbau, verzichten aber - wegen der einleitenden Ausführung - auf den Quantoren-Induktionschritt.

Definition: Sei $F =< N, E, source, target, S, Var \cup Con, Attr, label_N, label_E >$ ein Feature-Graph. Wir definieren für einen Term t:

$$[\![t]\!]^F = \begin{cases} n \in N & \text{falls} \quad t \in label_F(n) \\ \text{undef.} & \text{sonst} \end{cases}$$

Dabei ist $label_F$ die Labelfunktion des Feature-Graphen F. $[\![t]\!]^F$ ist dann wegen $(N1')$ wohldefiniert. Für ein positives Literal $t \doteq d$ definieren wir:

$$[\![t \doteq d]\!]^F = \begin{cases} 1 & \text{falls} \quad [\![t]\!]^F = [\![d]\!]^F \\ 0 & \text{falls} \quad [\![t]\!]^F \neq [\![d]\!]^F \end{cases}$$

Ist $[\![t]\!]^F$ oder $[\![d]\!]^F$ undefiniert, so bleibt auch $[\![t \doteq d]\!]^F$ undefiniert. Ist ϕ eine Formel, so definieren wir:

$$[\![\neg\phi]\!]^F = \begin{cases} 1 & \text{falls} \quad [\![\phi]\!]^F = 0 \\ 0 & \text{falls} \quad [\![\phi]\!]^F = 1 \end{cases}$$

Ist $[\![\phi]\!]^F$ undefiniert, so bleibt auch $[\![\neg\phi]\!]^F$ undefiniert. Für zwei Formeln ϕ und ψ definieren wir:

$$[\![\phi \wedge \psi]\!]^F = \begin{cases} 1 & \text{falls} \quad [\![\phi]\!]^F = 1 \text{ und } [\![\psi]\!]^F = 1 \\ 0 & \text{falls} \quad [\![\phi]\!]^F = 0 \text{ oder } [\![\psi]\!]^F = 0 \end{cases}$$

Ist $[\![\phi]\!]^F$ oder $[\![\psi]\!]^F$ undefiniert, so bleibt auch $[\![\phi \wedge \psi]\!]^F$ undefiniert.

Für Formeln ϕ mit $[\![\phi]\!]^F = 1$ sagen wir: F *erfüllt* ϕ und schreiben $F \models \phi$.

Beispiel 2.3.1: Für den Feature-Graphen F aus Abbildung 2.2.1 (mit gerichtetem, endlichen Graphen aus Abbildung 2.2.2) gilt etwa:

$$\begin{aligned}
[\![cffgn]\!]^F &= 5 \\
[\![ag]\!]^F &= 3 \\
[\![ch]\!]^F &= 3 \\
[\![bm]\!]^F &= 5 \\
[\![bmn]\!]^F &= \text{undef.} \\
[\![en]\!]^F &= \text{undef.} \\
F &\models cffgn \doteq bm \\
F &\models cff \doteq a \\
F &\models e \not\doteq bm
\end{aligned}$$

Beispiel 2.3.2: Der Feature-Graph aus Abbildung 2.2.1 erfüllt den folgenden Constraint (a, b, c, e seien Variablen, f, g, h, m, n, p Konstanten):

$$\begin{aligned}
eg &\doteq bmp \\
e &\doteq em \\
bm &\doteq cffgn \\
c &\doteq a \\
ag &\doteq ah \\
a &\doteq aff \\
affgn &\not\doteq e
\end{aligned}$$

Wir wollen nun für Constraints zeigen, daß die Erfüllbarkeit in Feature-Graphen die Erfüllbarkeit in Modellen nach sich zieht. Später werden wir auch die umgekehrte Richtung erhalten. Wegen der Eigenschaften $(S1)$ und $(S2)$ ist es schwierig, den Feature-Graphen direkt aus dem Modell und der Belegung zu konstruieren.

Behauptung: Für einen Constraint C (in Folgen-Notation) gilt:

$$\begin{aligned}
\exists M, \alpha: \quad M, \alpha &\models C \qquad \text{falls} \\
\exists \text{ Feature-Graph } F: \quad F &\models C
\end{aligned}$$

Beweis: Sei $F =< N, E, source, target, S, Var \cup Con, Attr, label_N, label_E >$ ein Feature-Graph, der C erfüllt. Wir setzen:

$U = N$

für $c \in Con$: $\quad [\![c]\!]^M = \begin{cases} n & \text{falls} \quad c \in label_N(n) \\ \text{beliebig} & \text{sonst} \end{cases}$

für $f \in Attr$: $\quad [\![f]\!]^M: U \to U$

$$m \mapsto \begin{cases} n & \text{falls} \quad \text{es gibt ein } e \in E: \\ & \qquad source(e) = m \\ & \qquad target(e) = n \\ & \qquad label_E(e) = f \\ \text{beliebig} & \text{sonst} \end{cases}$$

für $x \in Var$: $\quad \alpha: Var \to U$

$$x \mapsto \begin{cases} n & \text{falls} \quad x \in label_N(n) \\ \text{beliebig} & \text{sonst} \end{cases}$$

Wegen $(N1)$ und $(E1)$ sind $[\![.]\!]^M$ und α wohldefiniert. Wir sind fertig, wenn wir zeigen können:

Beh.: Für alle Terme t gilt:

$$[\![t]\!]^{M,\alpha} = n \qquad \text{falls} \qquad [\![t]\!]^F = n$$

Beweis durch Induktion über den Termaufbau:
Sei also zunächst $t \in Var \cup Con$ und $[\![t]\!]^F = n$. Dann gilt per Definition von M und α:

$$\begin{aligned}
[\![t]\!]^{M,\alpha} &= \begin{cases} \alpha(t) & \text{für} \quad t \in Var \\ [\![t]\!]^M & \text{für} \quad t \in Con \end{cases} \\
&= n
\end{aligned}$$

Sei die Beh. nun für t bewiesen, sei $f \in Attr$ und $[\![tf]\!]^F = n$. Dann gilt zunächst:

$$tf \in label_F(n)$$

Mit Lemma 2.2.1 erhalten wir nun $m \in N, \ \ p \in Pfad(m, n)$ mit:

$$\begin{aligned}
t &\in label_F(m) \quad (\text{d.h. } [\![t]\!]^F = m) \\
f &= label_E(p)
\end{aligned}$$

Insbesondere ist $p \in E$ und wir erhalten:

$$[\![tf]\!]^{M,\alpha} = [\![f]\!]^M \left([\![t]\!]^{M,\alpha} \right)$$

$$
\begin{aligned}
&= [f]^M\left([t]^F\right)\\
&= [f]^M(m)\\
&= n
\end{aligned}
$$

Dabei haben wir in der ersten Zeile die Definition von $[.]^{M,\alpha}$, in der zweiten Zeile die Induktionsvoraussetzung, in der dritten Zeile die Definition von $[t]^F$ und in der vierten Zeile die Definition von $[f]^M$ benutzt $\diamond$

Das folgende Lemma ist eine unmittelbare Folgerung aus dem Lemma 2.2.1

Lemma 2.3.1: Seien t, d Terme, $\ell \in Attr^*$, F ein Feature-Graph. Dann gilt:

$$[t]^F = [d]^F \;\Rightarrow\; [t\ell]^F = [d\ell]^F \quad \text{oder } [t\ell]^F, [d\ell]^F \text{ sind beide undef.}$$

Beweis: Sei $\ell \in Attr^+$ (sonst wäre nichts zu zeigen). Sei $[t]^F = [d]^F$ und o.B.d.A. sei $[t\ell]^F$ definiert. Sei also $n \in N$ mit $[t\ell]^F = n$, d.h.

$$t\ell \in label_F(n)$$

Mit Lemma 2.2.1 („$\Rightarrow$") erhalten wir ein $m \in N,\ p \in Pfad(m,n)$ mit:

$$t \in label_F(m),\ \ \ell = label_E(p)$$

D.h. $[d]^F = [t]^F = m$. Somit:

$$d \in label_F(m)$$

Lemma 2.2.1 („$\Leftarrow$") liefert dann: $d\ell \in label_F(n)$. D.h.: $[d\ell]^F = n = [t\ell]^F \diamond$

Bemerkung: Es gilt übrigens nicht die Rückrichtung:

$$[t\ell]^F = [d\ell]^F \;\not\Rightarrow\; [t]^F = [d]^F$$

Beispiel 2.3.3: Der Featuregraph F aus Abbildung 2.3.1 ist ein Gegenbeispiel: $[xff]^F = [xf]^F$. Aber: $[xf]^F \neq [x]^F$.

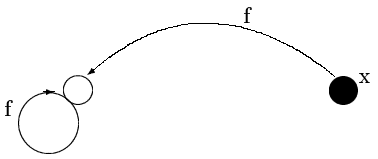


Abbildung 2.2.3

2.4 Die Normalform eines Constraints

In diesem Abschnitt definieren wir für einen (in Folgen-Notation gegebenen) Constraint eine Normalform, und zeigen, daß der Algorithmus mit dem wir eine Normalform herstellen, terminiert.

Definition: Sei C ein Constraint (in Mengen-Notation). Die *Terme* und *Subterme von C* sind:

$$
\begin{aligned}
Term(C) &= \{\ t \mid \text{es gibt einen Term } d \text{ mit :}\\
&\qquad t \doteq d \in C \text{ oder } t \not\doteq d \in C \text{ oder } d \doteq t \in C \text{ oder } d \not\doteq t \in C\}\\
Subterm(C) &= \{\ t \mid \text{es gibt ein } \ell \in Attr^* \text{ mit : } \ t\ell \in Term(C)\}
\end{aligned}
$$

Ferner benötigen wir für Constraints eine syntaktische Operation: die *obligatorische Ersetzung eines Terms t durch einen Term d*. Für einen Term τ definieren wir:

$$\tau[t|d] = \begin{cases} d\ell & \text{falls} \quad \tau = t\ell \text{ mit } \ell \in Attr^* \\ \tau & \text{sonst} \end{cases}$$

Diese Definition wird dann in der gewohnten Weise für Formeln (ohne Quantoren) induktiv über ihren Aufbau fortgesetzt. Wir führen die Details nicht aus, sondern kommen zu einem für die Herstellung einer Normalform wichtigen Begriff:

Eine lexikographische Ordnung [6] $\prec$ auf der Menge aller Terme, die die folgende Eigenschaft besitzt:

$$(T0) \qquad x \prec c \qquad \text{falls} \qquad x \in Var,\ \ c \in Con$$

heißt *Termordnung*.

Die Existenz und Konstruierbarkeit einer Termordnung ist gesichert, da die Menge aller Terme eine Produktmenge $(Var \cup Con) \times Attr^*$ mit Faktoren $Var \cup Con$ und $Attr$ ist. Da jeder Faktor endlich oder abzählbar ist, kann er wohlgeordnet werden. Dies sichert uns zunächst die Existenz einer lexikographischen Ordnung auf der Menge der Terme. Sie erfüllt $(T0)$, wenn die Wohlordnung auf $Var \cup Con$ so gewählt wird, daß alle Variablen vor den Konstanten angeordnet werden.

[6] In der Literatur findet man verschiedene Definitionen für eine lexikographische Ordnung. Wir benutzen:
$< W_i, \prec_i >$ seien wohlgeordnete Mengen $(i = 1, 2, 3, ...)$.
Sei $W = \{w_1 ... w_p | p \geq 1 \text{ und } w_i \in W_i \text{ für } i = 1, ..., p\}$. Die *lexikographische Ordnung* $\prec$ für W ist dann wie folgt definiert:
$v_1 ... v_p \prec w_1 ... w_q$ falls (i) $p < q$ oder (ii) $p = q$ und es gibt ein $r \geq 1$ mit: $v_r <_r w_r$ und $v_i = w_i$ für alle $i < r$.
Wir bemerken noch, daß für $v, w \in W$ entscheidbar ist, ob $v \prec w$.

Wir listen nun einige Eigenschaften einer Termordnung auf. Seien d, t Terme, $\ell \in Attr^*$. Dann gilt:

$(T1)$ $d \prec t$ oder $d = t$ oder $t \prec d$
$(T2)$ es gibt einen kleinstes Element
$(T3)$ $d \prec t$ falls $|d| < |t|$
$(T4)$ $d\ell \prec t\ell$ falls $d \prec t$

(Dabei bezeichnet $|t|$ die *Länge* eines Terms t, d.h. $|sf_1...f_k| = k+1$ für $s \in Var \cup Con$.)

Beispiel 2.4.1: Sind $x \prec y$ Variablen, $a \prec b$ Konstanten und $f \prec g \prec h$ Attribute eines Constraints, so gelten etwa:

$$\begin{aligned} x &\prec a \\ b &\prec xf \\ xf &\prec yf \\ xf &\prec xg \\ yf &\prec xfg \\ xgf &\prec yhg \\ xfg &\prec xfh \quad \text{u.s.w.} \end{aligned}$$

Definition: Wir überführen einen Constraint C in *Normalform*, indem wir eine Termordnung $\prec$ festlegen und die vier folgenden Regeln solange anwenden, bis keine Regel mehr anwendbar ist (wir notieren Variablen mit x, Terme mit t oder d und Constraints mit Δ oder Γ):

1. $$\frac{\Delta,\ t \doteq x,\ \Gamma}{\Delta[x \mid t],\ t \doteq t,\ \Gamma[x \mid t]}\ (x \notin t)$$

2. $$\frac{\Delta,\ t \doteq t,\ \Gamma}{\Delta,\ \Gamma}\ (t \in Subterm(\Delta \cup \Gamma)\ )$$

3. $$\frac{\Delta,\ d \doteq t,\ \Gamma}{\Delta,\ t \doteq d,\ \Gamma}\ (d \prec t)$$

4. $$\frac{\Delta,\ t \doteq d,\ \Gamma}{\Delta[t \mid d],\ t \doteq d,\ \Gamma[t \mid d]}\ (d \prec t \text{ und } t \in Subterm(\Delta \cup \Gamma)\ )$$

Die Regel heissen (der Reihe nach): *Eliminationsregel*, *Subtermregel*, *Ordnungsregel* und *Verkürzungsregel*.

Beispiel 2.4.2: Wir werden nun den Constraint aus Beispiel 2.3.2 in eine Normalform überführen. Wir wollen zunächst die Eliminationsregel anwenden. Da $c \doteq a$ die Seitenbedingung der Eliminationsregel erfüllt, ersetzen wir in allen Literalen des Constraints a durch c. Wir erhalten:

$$\begin{aligned} eg &\doteq bmp \\ e &\doteq em \\ bm &\doteq cffgn \\ c &\doteq c \\ cg &\doteq ch \\ c &\doteq cff \\ cffgn &\not\doteq e \end{aligned}$$

In diesem Constraint erfüllt kein Literal die Seitenbedingung der Eliminationsregel. Wir wenden uns deshalb der Subtermregel zu. Das Literal $c \doteq c$ erfüllt die Seitenbedingung der Subtermregel, da c beispielsweise Subterm von $cg \doteq ch$ ist. Nach Anwendung der Subtermregel erhalten wir:

$$\begin{aligned} eg &\doteq bmp \\ e &\doteq em \\ bm &\doteq cffgn \\ cg &\doteq ch \\ c &\doteq cff \\ cffgn &\not\doteq e \end{aligned}$$

Da nunmehr kein Literal der Form $d \doteq d$ im Constraint vorhanden ist, wenden wir nun die Ordnungsregel an. Wir nehmen an, daß $a \prec b \prec c \prec e$ und $f \prec g \prec h \prec m \prec n \prec p$ ist, und erhalten:

$$\begin{aligned} bmp &\doteq eg \\ em &\doteq e \\ cffgn &\doteq bm \\ ch &\doteq cg \\ cff &\doteq c \\ cffgn &\not\doteq e \end{aligned}$$

Nun versuchen wir die Verkürzungsregel anzuwenden. Das Literal $cff \doteq c$ erfüllt die Seitenbedingung der Verkürzungsregel, da cff ein Subterm des Literals $cffgn \doteq$

bm ist. Wir erhalten:

$$\begin{aligned} bmp &\doteq eg \\ em &\doteq e \\ cgn &\doteq bm \\ ch &\doteq cg \\ cff &\doteq c \\ cgn &\not\doteq e \end{aligned}$$

Das Literal $cgn \doteq bm$ erfüllt ebenfalls die Seitenbedingung der Verkürzungsregel, da cgn ein Subterm des Literals $cgn \not\doteq e$ ist. Wir erhalten:

$$\begin{aligned} bmp &\doteq eg \\ em &\doteq e \\ cgn &\doteq bm \\ ch &\doteq cg \\ cff &\doteq c \\ bm &\not\doteq e \end{aligned}$$

Da nun keine der vier Regeln mehr anwendbar ist, haben wir eine Normalform hergestellt $\diamond$

Abschliessend zeigen wir nun:

Behauptung: Der Algorithmus zur Herstellung einer Normalform terminiert, d.h. die vier Normalformregeln müssen für jeden Constraint C nur endlich oft angewandt werden.

Beweis: Die Eliminationsregel substituiert in einem Constraint C eine Variable x durch einen Term t, in dem diese Variable nicht vorkommt. Nach jeder Anwendung der Eliminationsregel verringert sich also die Anzahl der Variablen. Da in C nur endlich viele Variablen vorkommen und keine der vier Regeln eine neue Variable einführt, kann die Eliminationsregel höchstens endlich oft angewandt werden.
Wir zeigen nun, daß die Verkürzungsregel nur endlich oft angewandt wird: Ist $\tau_1 \doteq \tau_2$ ein Literal aus $\Delta \cup \Gamma$ und t Subterm von τ_1 oder τ_2, so ersetzt die Verkürzungsregel das Literal $\tau_1 \doteq \tau_2$ durch ein Literal $\tau_1' \doteq \tau_2'$, für das $\tau_1' \prec \tau_1$ oder $\tau_2' \prec \tau_2$ gilt (t wird durch einen Term d mit $d \prec t$ ersetzt). Entsprechendes gilt auch für negative Literale. D.h. jede Anwendung der Verkürzungsregel ersetzt

im Constraint wenigstens ein Literal ϕ durch ein Literal ϕ', welches mindestens eine kleinere Termseite besitzt. Da die Termordnung aber ein kleinstes Element besitzt (dies ist die kleinste Variable), kann der Verkürzungsprozess nur dann *ad infinitum* fortgeführt werden, wenn die drei anderen Normalformregeln ständig neue Literale mit grösseren Termen liefern. Da aber nur die Eliminationsregel Literale mit eventuell grösseren Termen liefert, sie jedoch nur endlich oft angewandt wird, kann dieser Fall nicht eintreten.
Die Subtermregel wird nur endlich oft angewandt, da die einzigen Regeln, die neue Subterme liefern können, die Eliminationsregel und die Verkürzungsregel sind. Beide werden aber nur endlich oft angewandt. D.h. im Verlauf des Normalformalgorithmus' entstehen nur endlich viele neue Subterme. Da die Subtermregel nur dann auf ein Literal $t \doteq t$ angewandt wird, wenn t ein Subterm von $\Delta \cup \Gamma$ ist, kann sie nur endlich oft angewandt werden.
Die Ordnungsregel wird ebenfalls nur endlich oft angewandt. Sie ersetzt ein Literal $d \doteq t$ mit $d \prec t$ durch das Literal $t \doteq d$. Auf das neue Literal $t \doteq d$ ist die Ordnungsregel nicht wieder anwendbar. Die Ordnungsregel ist somit nur auf solche Literale anwendbar, die nicht schon vorher einmal von der Ordnungsregel aufgegriffen wurden. Somit kommen für die Ordnungsregel nur die Ausgangsliterale und die durch die Eliminationsregel und Verkürzungsregel ersetzten Literale in Frage. Diese bilden eine endliche Menge von Literalen.

2.5 Erfüllbarkeit eines Constraints

Wir werden in diesem Abschnitt zeigen, daß ein Constraint genau dann erfüllbar ist, wenn eine Normalform erfüllbar ist. Für die Erfüllbarkeit einer Normalform werden wir ein einfaches Kriterium angeben.

Wir beginnen mit einer einfachen Folgerung aus dem Lemma 2.3.1.

Lemma 2.5.1: Sei F ein Featuregraph, ϕ eine quantorenfreie Formel, d, t Terme mit $[\![t]\!]^F = [\![d]\!]^F$. Dann gilt:

$$F \models \phi \quad \Leftrightarrow \quad F \models \phi[t|d]$$

Beweis: (i) sei τ ein Term. Wir zeigen für ein beliebiges $n \in N$:

$$[\![\tau]\!]^F = n \quad \Leftrightarrow \quad [\![\tau[t|d]]\!]^F = n$$

Für $\tau[t|d] = \tau$ ist nichts zu zeigen. Sei also $\tau = t\ell$ (mit $\ell \in Attr^*$). Dann ist zu zeigen:

$$[\![t\ell]\!]^F = n \quad \Leftrightarrow \quad [\![d\ell]\!]^F = n$$

Lemma 2.3.1 liefert das Gewünschte.
(ii) induktiv über den Aufbau von ϕ zeigt man nun alles weitere. Wir bemerken noch, daß wir den Quantorenschritt auslassen dürfen, da ϕ eine quantorenfreie Formel ist $\diamond$

Behauptung: Sei NF eine Normalform eines Constraints C. Dann gilt:

$$C \text{ erfüllbar} \quad \Leftrightarrow \quad NF \text{ erfüllbar}$$

Beweis:Da NF aus C durch endlich viele Anwendungen der Eliminationsregel, der Subtermregel, der Ordnungsregel und der Verkürzungsregel entsteht, zeigen wir, daß jede Regelanwendung diese Eigenschaft bewahrt.

4. Verkürzungsregel: „$\Rightarrow$": Sei F ein Feature-Graph mit $F \models \Delta, t \doteq d, \Gamma$. Dann gilt:

$$F \models \Delta, \quad [\![t]\!]^F = [\![d]\!]^F, \quad F \models \Gamma$$

Mit Lemma 2.5.1 erhalten wir:

$$F \models \Delta[t|d], \quad [\![t]\!]^F = [\![d]\!]^F, \quad F \models \Gamma[t|d]$$

D.h. $F \models \Delta[t|d], t \doteq d, \Gamma[t|d]$.
„$\Leftarrow$": analog.

3. Ordnungsregel: wegen $[\![d \doteq t]\!]^F = [\![t \doteq d]\!]^F$ trivial.

2. Subtermregel: „$\Rightarrow$": trivial. „$\Leftarrow$": Sei F ein Feature-Graph mit $F \models \Delta, \Gamma$. Wir benutzen nun die Seitenbedingung $t \in Subterm(\Delta \cup \Gamma)$. Sei also $\ell \in Attr^*$ mit $t\ell \in Term(\Delta \cup \Gamma)$. Somit ist $[\![t\ell]\!]^F$ definiert. Es gibt also ein $n \in N$ mit $t\ell \in label_F(n)$. Für $\ell \in Attr^+$ liefert Lemma 2.2.1 dann ein $m \in N$ mit $t \in label_F(m)$ (für $\ell \notin Attr^+$ setzen wir $m = n$). D.h. $[\![t]\!]^F = m$. Somit $F \models \Delta, t \doteq t, \Gamma$.

1. Eliminationsregel:„$\Rightarrow$": Sei F ein Feature-Graph mit $F \models \Delta, t \doteq x, \Gamma$. Dann gilt:

$$F \models \Delta, \quad [\![t]\!]^F = [\![x]\!]^F, \quad F \models \Gamma$$

Mit Lemma 2.5.1 erhalten wir:

$$F \models \Delta[x|t], \quad F \models \Gamma[x|t]$$

$F \models t \doteq t$ ist trivial. Zusammenfassend: $F \models \Delta[x|t], t \doteq t, \Gamma[x|t]$.
„$\Leftarrow$": Sei F ein Feature-Graph mit $F \models \Delta[x|t], t \doteq t, \Gamma[x|t]$. Wegen der Seitenbedingung $x \notin t$ gilt:

$$x \notin Subterm(\Delta[x|t], t \doteq t, \Gamma[x|t])$$

O.B.d.A. nehmen wir deshalb an:

$$[\![x]\!]^F = \text{ undef.}$$

D.h. für alle $n \in N$ gilt:

$$x \notin label_N(n)$$

Wegen $F \models t \doteq t$ ist $[\![t]\!]^F = n_t \in N$. Wir versehen nun den Knoten n_t mit dem zusätzlichen Namen x. Sei:

$$label'_N : \begin{array}{lcl} N & \to & \wp(Var \cup Con) \\ n & \mapsto & \begin{cases} label_N(n) & \text{für} \quad n \neq n_t \\ label_N(n) \cup \{x\} & \text{für} \quad n = n_t \end{cases} \end{array}$$

Offensichtlich erfüllt $label'_N$ die Bedingungen $(N0) - (N2)$.
$F' =< N, E, source, target, S, Var \cup Con, Attr, label'_N, label_E >$ ist somit ein Feature-Graph. Wegen $label_{F'}(n) \supseteq label_F(n)$ (für alle $n \in N$), erfüllt F' jede Formel, die F erfüllt. Insbesondere gilt:

$$F' \models \Delta[x|t], \Gamma[x|t]$$

Gemäss Konstruktion gilt aber auch:

$$F' \models t \doteq x$$

Lemma 2.5.1 liefert dann das Gewünschte $\diamond$

Bemerkung: Genauer haben wir bewiesen:

$$\begin{array}{lcl} F \models C & \Rightarrow & F \models NF \\ F \models NF & \Rightarrow & F' \models C \end{array}$$

Hierbei unterscheidet sich F' nur durch die Knotenlabelung von F und hier wiederum nur um Variablenlabel $\diamond$

Das folgende Lemma ist trivial, wird aber oft benötigt.

Lemma 2.5.2: Sei NF ein normalisierter Constraint. Dann gilt:

$$t \doteq d \in NF \text{ und } t \neq d \;\Rightarrow\; t \notin Subterm(NF \setminus \{t \doteq d\})$$

Beweis: Aus $t \doteq d \in NF$ und $t \neq d$ folgt $d \prec t$ (für $t \prec d$ könnten wir die Ordnungsregel auf NF anwenden. Widerspruch).
Angenommen: $t \in Subterm(NF \setminus \{t \doteq d\})$. Dann können wir die Verkürzungsregel auf NF anwenden. Widerspruch $\diamond$.

Behauptung: Enthält ein normalisierter Constraint NF keine Literale der Form

$$t \not\doteq t$$

so ist er erfüllbar.

Wir liefern zwei Beweise für diese Behauptung. Im ersten Beweis schauen wir uns in der Semantik der Prädikatenlogik erster Stufe an, was zu tun ist. Hierbei lehnen wir uns an die Darstellung von Wedekind [9] an. Im zweiten Beweis geben wir - vom ersten Beweis ausgehend - einen Feature-Graphen an, der NF erfüllt.

1. Beweis: Sei NF ein normalisierter Constraint ohne Literale der Form $t \not\doteq t$. Wir definieren:

$$\bar{U} = \{t \in Term(NF) \mid \text{ es gibt einen Term } d \neq t \text{ mit } t \doteq d \in NF\}$$

Sei $U = Subterm(NF) \setminus \bar{U}$ und:

$$\begin{array}{rcl} \gamma: \; Subterm(NF) & \to & U \\ t & \mapsto & \begin{cases} d & \text{falls} \quad d \neq t \text{ und } t \doteq d \in NF \\ & \quad (\text{ insbesondere also } t \in \bar{U}) \\ t & \text{falls} \quad t \in U \end{cases} \end{array}$$

Hierfür müssen wir zeigen:

$$\begin{array}{ll} (i) & \gamma \text{ ist wohldefiniert} \\ (ii) & \gamma(t) \in U \text{ für } t \in \bar{U} \end{array}$$

zu (i): Angenommen $\gamma(t) = d$ und $\gamma(t) = d'$ mit $d \neq d'$. Dann ist zunächst $t \in \bar{U}$, d.h.

$$\begin{array}{l} t \doteq d \in NF \\ t \doteq d' \in NF \end{array}$$

Also ist $t \in Subterm(NF \setminus \{t \doteq d\})$. Wegen $d \neq t$ steht dies im Widerspruch zu Lemma 2.5.2.

zu (ii): Angenommen $t \in \bar{U}$ mit $\gamma(t) = d \in \bar{U}$. Dann gilt also:

$$\begin{array}{ll} t \doteq d \in NF & (\text{ mit } d \prec t) \\ d \doteq d' \in NF & (\text{ mit } d' \prec d) \end{array}$$

Es liegen also wieder zwei verschiedene [7] Literale vor. D.h. $d \in Subterm(NF \setminus \{d \doteq d'\})$ erneut im Widerspruch zu Lemma 2.5.2.

Wir definieren nun ein Modell M und eine Belegung α, welche NF erfüllt. Da U nie leer ist, können wir definieren:

$$\text{für } c \in Con: \quad [\![c]\!]^M = \begin{cases} \gamma(c) & \text{falls} \quad c \in Subterm(NF) \\ \text{beliebig} & \text{sonst} \end{cases}$$

$$\text{für } f \in Attr: \quad [\![f]\!]^M: \; U \to U, \quad t \mapsto \begin{cases} \gamma(tf) & \text{falls} \quad tf \in Subterm(NF) \\ \text{beliebig} & \text{sonst} \end{cases}$$

$$\text{für } x \in Var: \quad \alpha: \; Var \to U, \quad x \mapsto \begin{cases} \gamma(x) & \text{falls} \quad x \in Subterm(NF) \\ \text{beliebig} & \text{sonst} \end{cases}$$

Beh.: Für $t \in Subterm(NF)$ gilt:

$$[\![t]\!]^{M,\alpha} = \gamma(t)$$

Beweis durch Induktion über den Termaufbau:
I.A.: trivial
I.S.: sei also $tf \in Subterm(NF)$ und die Beh. für den Term t bewiesen. Dann gilt:

$$\begin{array}{rcl} [\![tf]\!]^{M,\alpha} & = & [\![f]\!]^M \left([\![t]\!]^{M,\alpha} \right) \\ & = & [\![f]\!]^M(\gamma(t)) \\ & = & [\![f]\!]^M(t) \\ & = & \gamma(tf) \end{array}$$

[7] Wir bemerken, daß dieser Schluß nicht funktionieren würde, wenn auch $d = t$ und $d' = d$ zugelassen wären.

Dabei haben wir in der ersten Zeile die Definition von $[\![.]\!]^{M,\alpha}$, in der zweiten Zeile die Induktionsvoraussetzung (weil $t \in Subterm(NF)$) und in der vierten Zeile die Definition von $[\![f]\!]^M$ benutzt. Es bleibt zu zeigen:

$$\gamma(t) = t$$

Wir zeigen hierfür, daß $t \in U$. Angenommen $t \in \bar{U}$. Dann gibt es einen Term $d \neq t$ mit $t \doteq d \in NF$. Wegen Lemma 2.5.2 gilt dann:
$t \notin Subterm(NF \setminus \{t \doteq d\})$. Gemäss unserer Voraussetzung ist aber $tf \in Subterm(NF)$. Da $tf \notin Subterm(t \doteq d)$, gilt also $tf \in Subterm(NF \setminus \{t \doteq d\})$. Also auch $t \in Subterm(NF \setminus \{t \doteq d\})$. Widerspruch.

Wir müssen nun noch zeigen, daß alle positiven und negativen Literale aus NF durch M und α erfüllt werden:

(i) Sei $\phi = (t \doteq d) \in NF$. Dann gilt also: $t, d \in Subterm(NF)$. D.h.

$$\begin{array}{rcl} [\![t]\!]^{M,\alpha} & = & \gamma(t) \\ [\![d]\!]^{M,\alpha} & = & \gamma(d) \end{array}$$

Wir müssen also zeigen, daß $\gamma(t) = \gamma(d)$.

Dies ist trivial, falls $t = d$. Sei also $d \prec t$. Dann gilt nach Definition von γ, daß $\gamma(t) = d$. Wir müssen also zeigen, daß $\gamma(d) = d$, beziehungsweise $d \in U$.

Angenommen $d \in \bar{U}$. Dann gibt es ein $d' \neq d$ mit:

$$d \doteq d' \in NF$$

Somit $d \in Subterm(NF \setminus \{t \doteq d\})$ im Widerspruch zu Lemma 2.5.2.

(ii) Sei $\phi = (t \not\doteq d) \in NF$. Dann gilt wieder: $t, d \in Subterm(NF)$. Wir müssen nun zeigen, daß $\gamma(t) \neq \gamma(d)$.

Wir bemerken zunächst, daß $t, d \in U$. Angenommen: $t \in \bar{U}$. Dann gibt es ein $t' \neq t$ mit $t \doteq t' \in NF$. Nach Lemma 2.5.2 somit: $t \notin Subterm(NF \setminus \{t \doteq t'\})$. Dies steht im Widerspruch zu $t \in Subterm(\phi)$. Ebenso führen wir $d \in \bar{U}$ zum Widerspruch.

Angenommen: $\gamma(t) = \gamma(d)$. Nach Definition von γ gilt dann (wegen $t, d \in U$): $t = d$. Also: $\phi = (t \not\doteq t) \in NF$. Widerspruch $\diamond$

Bemerkung: Diesem Beweis entnehmen wir, daß ein erfüllbarer normalisierter

Constraint NF die folgenden Literaltypen enthält:

Typ	(I) :	$t \doteq d$	mit	$t \neq d$	und	$t \in \bar{U}$,	$d \in U$
Typ	(II) :	$t \doteq t$				$t \in U$	
Typ	(III) :	$t \not\doteq d$		$t \neq d$		$t \in U$,	$d \in U$

Ferner haben wir eine weitere Eigenschaft bewiesen:

$$tf \in Subterm(NF) \;\Rightarrow\; t \in U$$

2. Beweis: Wir konstruieren einen Feature-Graphen F, der NF erfüllt. Hierzu übernehmen wir U und γ aus dem 1. Beweis und definieren:

$N = U$

$E = \{ <t, d, f> \in N \times N \times Attr \mid \gamma(tf) = d \}$

$$\begin{array}{lrcl} source: & E & \to & N \\ & <t, d, f> & \mapsto & t \end{array}$$

$$\begin{array}{lrcl} target: & E & \to & N \\ & <t, d, f> & \mapsto & d \end{array}$$

$S = N \cap (Var \cup Con)$

$$\begin{array}{lrcl} label_E: & E & \to & Attr \\ & <t, d, f> & \mapsto & f \end{array}$$

$$\begin{array}{lrcl} label_N: & N & \to & \wp(Var \cup Con) \\ & n & \mapsto & \{ s \in Var \cup Con \mid s \in Subterm(NF) \text{ mit } \gamma(s) = n \} \end{array}$$

Beh.: $F =< N, E, source, target, S, Var \cup Con, Attr, label_N, label_E >$ ist ein Feature-Graph.

Beweis: Wir überprüfen nacheinander $(S1), (N1), (E1)$ und $(N2)$.

zu $(S1)$: Wir beweisen dies induktiv über den Termaufbau:
I.A.: Sei also $n \in Var \cup Con$ und $n \in N$. Dann ist $n \in S$, also nichts zu zeigen.
I.S.: Sei $n = tf \in N \setminus S$. Wir müssen ein $s \in S$ und ein $p \in Pfad(s, n)$ finden. Sei dazu:

$$e =< t, n, f >$$

Gemäss der Bemerkung, die dem 1. Beweis folgte, ist $t \in N$. Ferner gilt $\gamma(tf) = \gamma(n) = n$. Also per Definition: $e \in E$. Somit:

Fall 1) Für $t \in S$ setzen wir $s = t$ und $p = e$.
Fall 2) Für $t \notin S$ ist $t \in N \setminus S$. Per Induktionsvoraussetzung gibt es daher ein $s \in S$ und ein $p' \in Pfad(s,t)$. Wir setzen $p = p'e$ und erhalten das Gewünschte.

zu $(N1)$: Seien $n_1 \neq n_2 \in N$. Angenommen: $s \in label_N(n_1) \cap label_N(n_2)$. Dann gilt: $n_1 = \gamma(s) = n_2$. Widerspruch.

zu $(E1)$: Seien $e_1 \neq e_2 \in E, source(e_1) = source(e_2)$.
Zu zeigen ist: $label_E(e_1) \neq label_E(e_2)$. Seien dazu:

$$e_1 = <t, d_1, f_1>$$
$$e_2 = <t, d_2, f_2>$$

Angenommen: $label_E(e_1) = label_E(e_2)$. Dann folgt $f_1 = f_2$. Also $d_1 = \gamma(tf_1) = \gamma(tf_2) = d_2$. D.h. $e_1 = e_2$. Widerspruch.

zu $(N2)$: Für $s \in S$ gilt $\gamma(s) = s$. Somit $s \in label_N(s)$. D.h. $label_N(s) \neq \emptyset$.

Damit haben wir alle Feature-Graph-Eigenschaften nachgeprüft. Wenn wir noch die folgende Behauptung zeigen, können wir wie im 1. Beweis folgern, daß F alle positiven und negativen Literale aus NF erfüllt.

Beh.: $[\![t]\!]^F = \gamma(t)$ für $t \in Subterm(NF)$.

Beweis durch Induktion über den Termaufbau:
I.A. Sei $t \in Var \cup Con$ und $t \in Subterm(NF)$. Per Definition von $label_N$ gilt (wegen $\gamma(t) = \gamma(t)$):

$$label_N(\gamma(t)) \ni t$$

D.h. $[\![t]\!]^F = \gamma(t)$.

I.S. Sei $tf \in Subterm(NF)$. Dann ist $t \in N$, d.h. nach Induktionsvoraussetzung: $[\![t]\!]^F = \gamma(t) = t$. D.h.

$$t \in label_F(t)$$

Für $p = <t, \gamma(tf), f>$ gilt offensichtlich:

$$p \in Pfad(t, \gamma(tf))$$
$$label_E(p) = f$$

Mit Lemma 2.2.1 („$\Leftarrow$") erhalten wir:

$$tf \in label_F(\gamma(tf))$$

D.h. $[\![tf]\!]^F = \gamma(tf)$ $\diamond$

Wir vervollständigen nun unsere Behauptung aus Abschnitt 2.3.

Behauptung:

$$\begin{array}{ll} \exists \text{ Feature-Graph F} : & F \models C \quad \text{falls} \\ \exists M, \alpha : & M, \alpha \models C \end{array}$$

Beweis: Seien also M, α mit:

$$M, \alpha \models C$$

Aus dem Substitutionslemma der Prädikatenlogik erster Stufe erhalten wir induktiv über die Konstruktion von NF:

$$M, \alpha \models NF$$

Somit enthält NF keine Literale der Form $t \not\equiv t$. D.h. es gibt einen Feature-Graphen F_{NF} mit:

$$F_{NF} \models NF$$

D.h. es gibt einen Feature-Graphen F mit:

$$F \models C \qquad \diamond$$

Beispiel 2.5.1: Wir führen für den normalisierten Constraint NF aus Beispiel 2.4.2 die Feature-Graph-Konstruktion des 2. Beweises vor:

$$Subterm(NF) = \{b, bm, bmp, e, eg, em, c, cg, cgn, ch, cf, cff\}$$

$$\bar{U} = \{bmp, em, cgn, ch, cff\}$$

$$N = U = \{b, bm, e, eg, c, cg, cf\}$$

$$\begin{array}{lrcll} \gamma: & Subterm(NF) & \to & U & \\ & bmp & \mapsto & eg & \\ & em & \mapsto & e & \\ & cgn & \mapsto & bm & \\ & ch & \mapsto & cg & \\ & cff & \mapsto & c & \\ & t & \mapsto & t & \text{für } t \in U \end{array}$$

$$\begin{array}{ll} E = \{ & <b, bm, m>, \\ & <bm, eg, p>, \\ & <e, eg, g>, \\ & <e, e, m>, \\ & <c, cg, g>, \\ & <cg, bm, n>, \\ & <c, cg, h>, \\ & <c, cf, f>, \\ & <cf, c, f> \quad \} \end{array}$$

$$S = \{b, e, c\}$$

Die Abbildung 2.5.1 zeigt den bis hierher konstruierten gerichteten, endlichen Graphen mit Wurzeln. Die Wurzeln sind durch doppelte Kreise markiert.

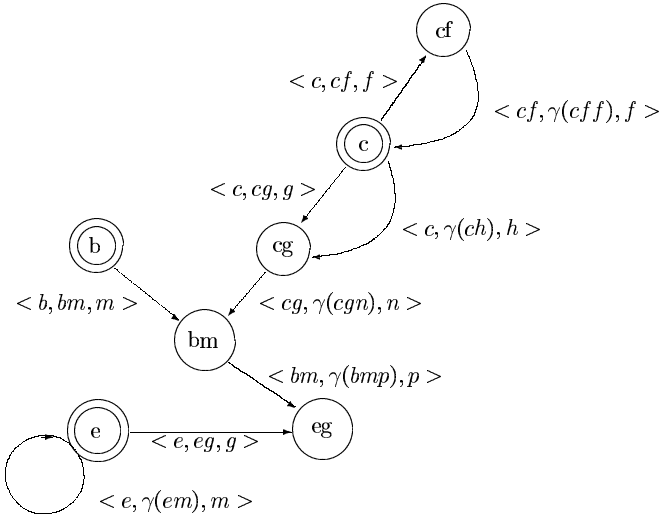


Abbildung 2.5.1

Ferner gilt nach Konstruktion:

$$\begin{array}{lrcll} label_N: & N & \to & \wp(Var \cup Con) & \\ & b & \mapsto & \{b\} & \\ & e & \mapsto & \{e\} & \\ & c & \mapsto & \{c\} & \\ & t & \mapsto & \emptyset & \text{für } t \notin S \end{array}$$

Die Kanten $e \in E$ werden mit ihrer 3. Koordinate gelabelt.

Offenbar erhalten wir mit unserer Konstruktion wieder den Feature-Graphen aus Abbildung 2.2.1 (bis auf das fehlende Label a). $\diamond$

Definition: Sei C ein Constraint und NF eine Normalform. Sei F der im zweiten Beweis aus NF konstruierte Featuregraph. Sei F' der Featuregraph, den wir aus F durch die im Beweis für „ NF erfüllbar $\Longrightarrow$ C erfüllbar “ vorgeführte Variablenlabelerweiterung erhalten. Dann heißt F' *Minimalmodell* für C $\square$

Bemerkung: Das Minimalmodell eines Constraints C ist *nicht* der kleinste Fea-

turegraph [8], der C erfüllt. Es ist aber der kleinste Featuregraph, der C erfüllt *und* keinen Constraint, der nicht aus C folgt.
Jeder Featuregraph mit diesen Eigenschaften ist (bis auf Isomorphie) eindeutig bestimmt. □

[8] d.h. der Featuregraph mit der wenigsten Anzahl Kanten

Kapitel 3

Unifikationsbasierte Grammatiken

Unifikationsbasierte Grammatiken, wie LFG und PATR-II weisen einem grammatischen Satz nicht nur eine Konstituentenstruktur, sondern auch eine Feature-Struktur zu.
Ein String (eine Abfolge von Wörtern) wird nur dann als grammatischer Satz betrachtet, wenn ihm durch die Grammatik eine (wohlgeformte) Feature-Struktur zugewiesen wird. Aus dieser Sichtweise ergeben sich zwei zueinander inverse Entscheidungsprobleme:

- Für eine gegebene Grammatik und einem gegebenen String müssen wir *entscheiden* können, ob es eine Feature-Struktur gibt, die dem String durch die Grammatik zugewiesen wird (Parsingproblem).
- Für eine gegebene Grammatik und eine gegebene Feature-Struktur müssen wir entscheiden können, ob es einen grammatischen Satz gibt, der diese Feature-Struktur trägt (Generierungsproblem).

Es ist seit langer Zeit bekannt, daß das Parsingproblem für unrestringierte unifikationsbasierte Grammatiken nicht entscheidbar ist (Kaplan und Bresnan [2], Johnson [3]). Wedekind [9] hat gezeigt, daß das Generierungsproblem für beliebige (unrestringierte) unifikationsbasierte Grammatiken entscheidbar ist.

In diesem Kapitel werden wir unifikationsbasierte Grammatiken formal beschreiben und das Generierungsproblem in diesem Formalismus definieren.

3.1 Formale Grundlagen unifikationsbasierter Grammatiken

Eine unifikationsbasierte Grammatik besteht aus einer kontextfreien Grammatik, deren Produktionen jeweils zusätzlich mit einer Menge von Annotationen versehen sein können, die sich auf die Konstituenten der zugehörigen Grammatikproduktion beziehen dürfen. Typische Beispiele aus LFG und PATR-II sind etwa:

$$\begin{array}{lcc} \text{S} & \longrightarrow & \begin{array}{cc} \text{NP} & \text{VP} \\ (\uparrow SUBJ) = \downarrow & \uparrow = \downarrow \end{array} \end{array} \qquad \begin{array}{lcl} \text{S} & \longrightarrow & \text{NP VP} \\ & & <\text{VP } AGR> = <\text{NP } AGR> \end{array}$$

$$\begin{array}{lcc} \text{NP} & \longrightarrow & \begin{array}{c} \text{John} \\ (\uparrow PRED) = JOHN \end{array} \end{array} \qquad \begin{array}{lcl} \text{NP} & \longrightarrow & \text{John} \\ & & <\text{NP}AGR\ NUM> = SG \\ & & <\text{NP}AGR\ PER> = 3RD \end{array}$$

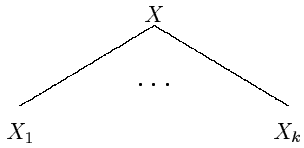
Für die formale Definition solcher Grammatiken werden wir die Annotationen als Formeln der in Kapitel 2 vorgestellten Constraintsprache $\mathcal{L}$ rekonstruieren.

Für die Übersetzung der LFG- und PATR-II-Annotationen betrachten wir deren Attribute *(SUBJ, PRED, AGR, NUM, PER)* als Elemente einer endlichen Menge *Attr* von Attributen aus $\mathcal{L}$. Die atomaren Werte *(JOHN, SG, 3RD)* der Annotationen betrachten wir als Elemente einer endlichen Menge *CON* von Konstanten aus $\mathcal{L}$.

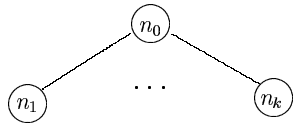
Eine Produktion

$$X \longmapsto X_1 ... X_k$$

einer kontextfreien Grammatik kann durch den folgenden Syntaxbaum repräsentiert werden:



Ist n_0 der Mutterknoten und sind $n_1, ..., n_k$ die Tochterknoten dieses Syntaxbaumes, so erhalten wir den Baum:



In LFG verweist das Auftauchen von $\downarrow$ in der Annotation von X_i $(1 \leq i \leq k)$ auf die am Tochterknoten n_i vorhandene Feature-Struktur. Jedes Auftauchen von $\uparrow$ verweist auf die am Mutterknoten n_0 vorhandene Feature-Struktur.

In PATR-II verweist das Auftauchen einer Kategorie X_i $(1 \leq i \leq k)$ auf die am Tochterknoten n_i vorhandene Feature-Struktur. Das Auftauchen der Kategorie X verweist auf die am Mutterknoten n_0 vorhandene Feature-Struktur.

Stehen nun die Variablen

$$x_{n_0}, x_{n_1}, ..., x_{n_k}$$

für die Feature-Strukturen an den Knoten

$$n_0, n_1, ..., n_k$$

so können die Annotationen der Produktion

$$X \longrightarrow X_1 ... X_k$$

als Constraints der Constraintsprache $\mathcal{L}$ rekonstruiert werden. Für unsere typischen LFG-Beispiele erhalten wir:

$$\begin{array}{lcll} \text{S} & \longrightarrow & \text{NP VP} & \{x_{n_0} SUBJ \doteq x_{n_1}, \quad x_{n_0} \doteq x_{n_2}\} \\ \text{NP} & \longrightarrow & \text{John} & \{x_{n_0} PRED \doteq JOHN\} \end{array}$$

Für die PATR-II-Beispiele erhalten wir:

$$\begin{array}{lcll} \text{S} & \longrightarrow & \text{NP VP} & \{x_{n_2} AGR \doteq x_{n_1} AGR\} \\ \text{NP} & \longrightarrow & \text{John} & \{x_{n_0} AGR\ NUM \doteq SG, \quad x_{n_0} AGR\ PER \doteq 3RD\} \end{array}$$

In der folgenden Definition werden wir formal festlegen, was wir unter einer *unifikationsbasierten* Grammatik verstehen wollen. Wir werden hierbei den Begriff der *kontextfreien* Grammatik als Ausgangsbasis benutzen.

Definition Eine *unifikationsbasierte Grammatik* $< V_N, V_T, S, \mathcal{L}, Prod >$ besteht aus einer endlichen Menge V_N von Nonterminalsymbolen, einer endlichen Menge V_T von Terminalsymbolen, einem Startsymbol $S \in V_N$, einer Constraintsprache

$\mathcal{L}$ und einer endlichen Menge $Prod$ von Produktionen. Jede Produktion $p \in Prod$ ist von der Gestalt:

$$X \longrightarrow X_1 ... X_k, \quad C[x_{n_0}, x_{n_1}, ..., x_{n_k}]$$

Hierbei ist $X \longrightarrow X_1 ... X_k$ die Produktion einer *kontextfreien Grammatik*, d.h.

$k \in \mathbb{N}$ ($k = 0$ für ϵ-Produktionen)
$X \in V_N$
$X_1, ..., X_k \in V_N \cup V_T$

Ferner notiert $C[x_{n_0}, x_{n_1}, ..., x_{n_k}]$ einen Constraint [1] aus $\mathcal{L}$, der höchstens die Variablen $x_{n_0}, x_{n_1}, ..., x_{n_k} \in Var$ enthält. Ferner vereinbaren wir, daß $C[x_{n_0}, x_{n_1}, ..., x_{n_k}]$ auch die leere Literalmenge sein darf. Alle Vokabularien ($V_N, V_T, Attr, Con$ und Var) werden als disjunkt vorausgesetzt □

Mit dieser Definition haben wir unifikationsbasierte Grammatiken im wesentlichen als kontextfreie Grammatiken charakterisiert. Der Unterschied besteht einfach darin, daß wir für jede Produktion einer unifikationsbasierten Grammatik die zusätzliche Angabe eines Constraints zulassen. Fehlt die Angabe dieser Constraints, so handelt es sich bei der unifikationsbasierten Grammatik lediglich um eine gewöhnliche kontextfreie Grammatik. Das folgende Beispiel illustriert dies.

Beispiel:

S	$\longrightarrow$	NP VP	$\{x_{n_0} SUBJ \doteq x_{n_1} \ , \quad x_{n_0} \doteq x_{n_2}\}$
NP	$\longrightarrow$	DET N	$\{x_{n_0} \doteq x_{n_1} \ , \quad x_{n_0} \doteq x_{n_2}\}$
VP	$\longrightarrow$	V NP	$\{x_{n_0} \doteq x_{n_1} \ , \quad x_{n_0} OBJ \doteq x_{n_2}\}$
DET	$\longrightarrow$	a	$\{x_{n_0} SPEC \doteq A \ , \quad x_{n_0} NUM \doteq SG\}$
DET	$\longrightarrow$	the	$\{x_{n_0} SPEC \doteq THE\}$
V	$\longrightarrow$	loved	$\{x_{n_0} TENSE \doteq PAST \ , \quad x_{n_0} PRED \doteq LOVE\}$
N	$\longrightarrow$	boy	$\{x_{n_0} NUM \doteq SG \ , \quad x_{n_0} PRED \doteq BOY\}$
N	$\longrightarrow$	boys	$\{x_{n_0} NUM \doteq PL \ , \quad x_{n_0} PRED \doteq BOY\}$
N	$\longrightarrow$	girl	$\{x_{n_0} NUM \doteq SG \ , \quad x_{n_0} PRED \doteq GIRL\}$ □

Bemerkung: Wir wollen bemerken, daß wir in der Definition für unifikationsbasierte Grammatiken auch Produktionen der etwas allgemeineren Form

$$X \longrightarrow X_1 ... X_k, \quad \phi[x_{n_0}, x_{n_1}, ..., x_{n_k}]$$

zulassen könnten, wobei $\phi[x_{n_0}, x_{n_1}, ..., x_{n_k}]$ eine beliebige Aussage aus $\mathcal{L}$ ist.
Da $\phi[x_{n_0}, x_{n_1}, ..., x_{n_k}]$ äquivalent zu einer Disjunktion

$$C_1[x_{n_0}, x_{n_1}, ..., x_{n_k}] \vee ... \vee C_l[x_{n_0}, x_{n_1}, ..., x_{n_k}]$$

[1] in Mengen-Notation

ist, könnten wir die oben angegebene Produktion durch die Produktionen

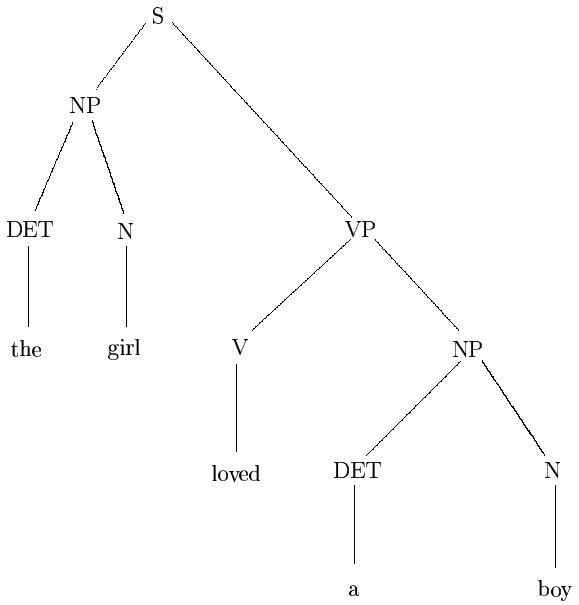
$$\begin{array}{ll} X \longrightarrow X_1 ... X_k, & C_1[x_{n_0}, x_{n_1}, ..., x_{n_k}] \\ & \vdots \\ X \longrightarrow X_1 ... X_k, & C_l[x_{n_0}, x_{n_1}, ..., x_{n_k}] \end{array}$$

ersetzen. Obwohl wir noch nicht in der Lage sind, diesen Punkt formal zu begründen, ist es uns intuitiv klar, daß beide Grammatiken dieselbe Sprache erzeugen.
Die Forderung, daß die Produktionen mit Constraints (und nicht mit Aussagen) aus $\mathcal{L}$ annotiert sein sollen, bedeutet unter diesem Blickwinkel also keine wirkliche Einschränkung. Aus theoretischen Gründen ist unsere Definition einer unifikationsbasierten Grammatik sehr gut geeignet - für praktische Anwendungen können wir sie leicht verallgemeinern. □

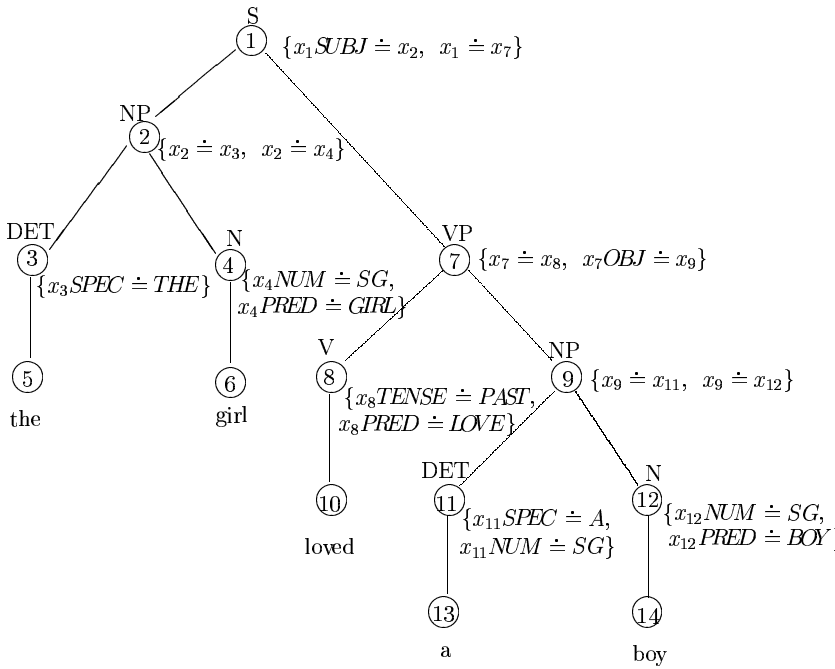
Wir wenden uns nun der Frage zu, wann ein String in einer unifikationsbasierten Grammatik ein grammatischer Satz ist. Da das Grundgerüst einer unifikationsbasierten Grammatik aus einer kontexfreien Grammatik besteht, wollen wir so vorgehen, daß wir uns an der entsprechenden Fragestellung für kontextfreie Grammatiken orientieren.
In einer kontextfreien Grammatik ist ein String ein grammatischer Satz, wenn es für ihn einen Syntaxbaum gibt. In der kontextfreien Grammatik, die das Gerüst unserer unifikationsbasierten Beispielgrammatik bildet, ist etwa der folgende Satz

the girl loved a boy

grammatisch, denn es gibt für ihn einen Syntaxbaum:



Wir wollen Syntaxbäume auch für unifikationsbasierte Grammatiken benutzen. Die sehr einfache Idee besteht darin, die formale Definition eines Syntaxbaumes so zu erweitern, daß in einem Syntaxbaum auch die Constraints einer jeden angewandten Produktion notiert werden können. Da die Constraints sich auf die Mutter- und Tochterknoten ihrer zugeordneten Produktion beziehen, ist es nötig, diese Knoten explizit anzugeben. Die Constraints selber können dann am jeweiligen Mutterknoten eingetragen werden. Für unseren obigen Satz könnten wir z.B. erhalten:

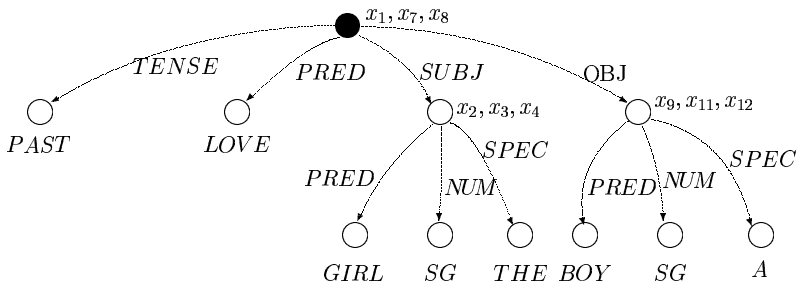


Die *Feature-Beschreibung* dieses Syntaxbaumes ist die Vereinigung aller in ihm

vorkommenden Constraints, also:

$$
\begin{array}{rcll}
x_1 SUBJ & \doteq & x_2 & \text{(Knoten 1)} \\
x_1 & \doteq & x_7 & \\
x_2 & \doteq & x_3 & \text{(Knoten 2)} \\
x_2 & \doteq & x_4 & \\
x_3 SPEC & \doteq & THE & \text{(Knoten 3)} \\
x_4 NUM & \doteq & SG & \text{(Knoten 4)} \\
x_4 PRED & \doteq & GIRL & \\
x_7 & \doteq & x_8 & \text{(Knoten 7)} \\
x_7 OBJ & \doteq & x_9 & \\
x_8 TENSE & \doteq & PAST & \text{(Knoten 8)} \\
x_8 PRED & \doteq & LOVE & \\
x_9 & \doteq & x_{11} & \text{(Knoten 9)} \\
x_9 & \doteq & x_{12} & \\
x_{11} SPEC & \doteq & A & \text{(Knoten 11)} \\
x_{11} NUM & \doteq & SG & \\
x_{12} NUM & \doteq & SG & \text{(Knoten 12)} \\
x_{12} PRED & \doteq & BOY &
\end{array}
$$

In Kapitel 2 haben wir ein Verfahren kennengelernt, mit dem wir entscheiden können, ob dieses Constraint erfüllbar ist.
Nach dessen Anwendung erhalten wir den folgenden *Feature-Graphen*, der die Feature-Beschreibung erfüllt:



Indem wir auf die Notation der Wurzeln und Variablen verzichten, sprechen wir von der *Feature-Struktur* unseres Satzes. Wenn möglich, wird die Feature-Struktur eines Satzes in einer *Matrix-Repräsentation* angegeben. Sie wird in der Regel als

übersichtlicher empfunden. Wir erhalten:

$$
\begin{bmatrix}
\text{PRED} & \text{LOVE} & \\
\text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{GIRL} \\ \text{SPEC} & \text{THE} \\ \text{NUM} & \text{SG} \end{bmatrix} \\
\text{OBJ} & \begin{bmatrix} \text{PRED} & \text{BOY} \\ \text{SPEC} & \text{A} \\ \text{NUM} & \text{SG} \end{bmatrix} \\
\text{TENSE} & \text{PAST}
\end{bmatrix}
$$

Ein String ist in einer unifikationsbasierten Grammatik ein grammatischer Satz, wenn er einen Syntaxbaum besitzt, dessen Feature-Beschreibung erfüllbar ist. Zusätzlich muß die Feature-Struktur in den diversen Grammatik-Formalismen noch gewisse *Wohlgeformtheitsbedingungen* erfüllen. Um diesen Punkt zu illustrieren betrachten wir den folgenden Satz:

the girl loved a boys

Im kontextfreien Teil unserer Beispielgrammatik ist dieser Satz grammatisch, da er eine Syntaxanalyse zuläßt. Für seine Feature-Beschreibung erhalten wir anstelle des Literals

$$x_{12} NUM \doteq SG$$

das Literal

$$x_{12} NUM \doteq PL$$

Die Feature-Beschreibung ist weiterhin erfüllbar. Wir erhalten als Feature-Struktur (nur in relevanten Teilen) :

$$
\begin{bmatrix}
\text{PRED} & \text{...} & \\
\text{SUBJ} & \text{...} & \\
\text{OBJ} & \begin{bmatrix} \text{PRED} & \text{...} \\ \text{SPEC} & \text{...} \\ \text{NUM} & \{\text{SG, PL}\} \end{bmatrix} \\
\text{TENSE} & \text{...} &
\end{bmatrix}
$$

Sie ist nicht wohlgeformt, da der OBJ NUM-Pfad *zwei* Werte trägt: die Konstanten SG und PL.

Wir werden an dieser Stelle nicht auf weitere Wohlgeformtheitsbedingungen eingehen, sondern verweisen auf das Ende des Kapitels. Wir halten aber fest, daß der Satz

the girl loved a boys

in unserer unifikationsbasierten Grammatik als ungrammatisch anzusehen ist, weil seine Feature-Struktur nicht wohlgeformt ist. Wenn wir unsere Beispiel-Grammatik leicht abändern, indem wir die Produktion

$$\text{N} \longrightarrow \text{boys} \quad \{x_{n_0}NUM \doteq PL \ , \quad x_{n_0}PRED \doteq BOY\}$$

durch die Produktion

$$\text{N} \longrightarrow \text{boys} \quad \{x_{n_0}NUM \not\doteq SG \ , \quad x_{n_0}PRED \doteq BOY\}$$

ersetzen, so ist unser obiger Satz ungrammatisch, weil sein Syntaxbaum eine Feature-Beschreibung trägt, *die nicht erfüllbar ist.* Wir prüfen dies nach: die Feature-Beschreibung enthält die Literale:

$$\begin{array}{rcl} \vdots & & \\ x_9 & \doteq & x_{11} \\ x_9 & \doteq & x_{12} \\ \vdots & & \\ x_{11}NUM & \doteq & SG \\ x_{12}NUM & \not\doteq & SG \\ \vdots & & \end{array}$$

Variablen-Elimination von x_{11} und x_{12} führt zu:

$$\begin{array}{rcl} \vdots & & \\ x_9NUM & \doteq & SG \\ x_9NUM & \not\doteq & SG \\ \vdots & & \end{array}$$

Die Anwendung der Verkürzungsregel ergibt:

$$\begin{array}{rcl} \vdots & & \\ SG & \not\doteq & SG \\ \vdots & & \end{array}$$

Das Normalisierungsverfahren zeigt also, daß die Feature-Beschreibung nicht erfüllbar ist. Der betrachtete Satz ist in der gewählten unifikationsbasierten Grammatik ungrammatisch.[2]

Im folgenden werden wir das anhand von Beispielen illustrierte Entscheidungsverfahren, ob ein String in einer Unifikationsgrammatik ein grammatischer Satz

[2]Genau genommen müssen wir uns noch davon überzeugen, daß es keine weiteren Syntaxanalysen für den Satz gibt. Offensichtlich ist dies aber der Fal.

ist oder nicht, formal definieren. Hierzu benötigen wir den Begriff des *Syntaxbaumes* und seines Grundgerüstes - den *Baum* . Wir bedienen uns hierfür - der Einfachheit halber - der in Kapitel 2 definierten gerichteten, endlichen Graphen mit Wurzeln.

Definition Ein $Baum < N, E, source, target, \{s\}, daughters >$ besteht aus einem gerichteten, endlichen Graphen mit Wurzel

$$< N, E, source, target, \{s\} >$$

und einer *Tochterfunktion*

$$daughters : N \longrightarrow N^*$$

sodaß:

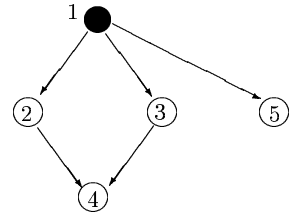
($B1$) für $n, m \in N$ mit $n \neq m$ gilt: $card\ \ Pfad(n, m) \leq 1$

($B2$) für $n \in N$ und $daughters(n) = m_1...m_k$ gilt:
$\{m_1, ..., m_k\} = \{m \in N | source(e) = n, target(e) = m \text{ für ein } e \in E\}$ □

Notation: Für $n, m \in N$ mit $card(Pfad(n, m)) = 1$ sagen wir:

$$n\ dominiert\ \ m \qquad \square$$

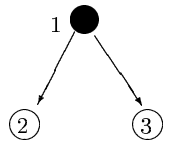
Die Bedingung ($B1$) schließt echte Feature-Graphen als Bäume aus. Zum Beispiel ist



kein Baum, weil es mehr als einen Pfad von Knoten 1 nach Knoten 4 gibt.

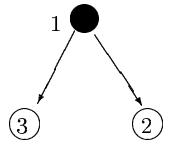
Die Wurzelbedingung ($S1$) garantiert uns, daß es zu jedem Knoten $n \in N$ mit $n \neq s$ einen Pfad von der Wurzel $s \in N$ nach n gibt. Es treten also keine *isolierte* Knoten auf.

Die Tochterfunktion $daughters : N \longrightarrow N^*$ wird gebraucht, um alle Tochterknoten eines Knotens $n \in N$ aufzuzählen. Dies ist nötig, um etwa den Baum



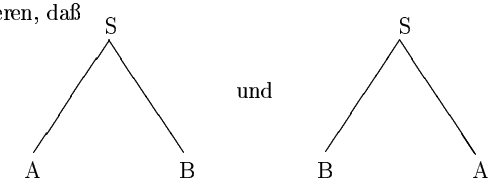
$daughters(1) = 2,3$

vom Baum



$daughters(1) = 3,2$

zu unterscheiden. Betrachten wir Syntaxbäume, wird dies klarer. Wir müssen irgendwie codieren, daß



zwei verschiedene Syntaxbäume sind. Hierfür werden wir benutzen, daß wir mittels der Funktion *daughters* in der Lage sind, von einem ersten Tochterknoten und einem zweiten Tochterknoten, usw. zu sprechen. Der erste Tochterknoten ist im linken Syntaxbaum mit der Konstituente A und im rechten Syntaxbaum mit der Konstituente B versehen. Der zweite Tochterknoten ist im linken Syntaxbaum mit der Konstituente B und im rechten Syntaxbaum mit der Konstituente A versehen.

Die Bedingung $(B2)$ garantiert, daß $daughters(n)$ *nur* Tochterknoten und *alle* Tochterknoten von $n \in N$ aufzählt.

Bemerkung: Für einen Baum $< N, E, source, target, \{s\}, daughters >$ sind $E, source, target$ durch die Angabe von N und $daughters$ (bis auf Isomorphie) eindeutig bestimmt. Die Wurzel $s \in N$ ebenfalls. O.b.d.A. werden wir deshalb in Zukunft von einem Baum $< N, daughters >$ sprechen $\square$

Defintion: Ein *Syntaxbaum* $< N, daughters, cat, constraint >$ einer unifikationsbasierten Grammatik $< V_N, V_T, S, \mathcal{L}, Prod >$ besteht aus einem Baum $< N, daughters >$, sowie zweier Funktionen

$$\begin{aligned} cat: &\ N \longrightarrow V_N \cup V_T \\ constraint: &\ N \longrightarrow \mathcal{L} \end{aligned}$$

sodaß:

$(SYN0)$ Für den Wurzelknoten $s \in N$ gilt: $cat(s) = S$

Ferner fordern wir, daß es zu jedem Knoten $n \in N$ mit Tochterknoten

$$\begin{aligned} &daughters(n) = m_1 ... m_k \\ &k \geq 1 \end{aligned}$$

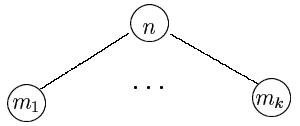
eine Produktion

$$X \longrightarrow X_1 ... X_k \ , \ C[x_{n_0}, x_{n_1}, ..., x_{n_k}]$$

aus $Prod$ gibt, mit:

$(SYN1)$ $cat(n) = X$ und $cat(m_i) = X_i \quad (i = 1, ..., k)$
$(SYN2)$ $constraint(n) = C[x_{n_0}, x_{n_1}, ..., x_{n_k}][x_{n_0}|x_n, x_{n_1}|x_{m_1}, ..., x_{n_k}|x_{m_k}]$ $\square$

Die zwei Funktionen *cat* und *constraint* versehen jeden Knoten des Syntaxbaumes mit einer Kategorie der unifikationsbasierten Grammatik und einem Constraint der Sprache $\mathcal{L}$.
Die Bedingung $(SYN0)$ sorgt dafür, daß der Wurzelknoten $s \in N$ mit dem Startsymbol der Grammatik gelabelt wird.
Die Bedingung $(SYN1)$ sorgt dafür, daß es zu jedem Teilbaum



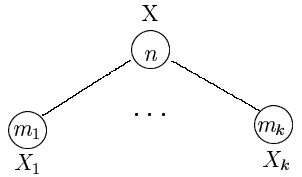
$k \geq 1$

$daughters(n) = m_1 ... m_k$

eine Produktion

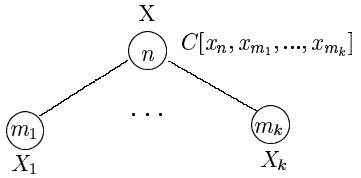
$$X \longrightarrow X_1 ... X_k \ , \ C[x_{n_0}, x_{n_1}, ..., x_{n_k}]$$

aus $Prod$ gibt, sodaß wir den folgenden Teilsyntaxbaum erhalten:



Schließlich sorgt die Bedingung $(SYN2)$ dafür, daß der Constraint $C[x_{n_0}, x_{n_1}, ..., x_{n_k}]$

der Produktion $X \longrightarrow X_1 ... X_k$ am Mutterknoten $n \in N$ eingetragen wird. Hierbei ist darauf zu achten, daß der Constraint richtig *instantiiert* wird. Wir deuten dies in der folgenden Skizze so an, daß wir $C[x_n, x_{m_1}, ..., x_{m_k}]$ für $C[x_{n_0}, x_{n_1}, ..., x_{n_k}][x_{n_0}|x_n, x_{n_1}|x_{m_1}, ..., x_{n_k}|x_{m_k}]$ schreiben:



Die Bedingungen $(SYN1)$ und $(SYN2)$ greifen selbstverständlich nicht für die *Blätter* $n \in N$ des Syntaxbaumes, d.h. für Knoten $n \in N$, die keine Töchter haben ($k = 0$ oder $daughters(n) \notin N^+$). Demnach sind die Blätter eines Syntaxbaumes nie mit einem Constraint - wohl aber mit einer Kategorie - gelabelt.

Mit dem Begriff *Syntaxbaum einer unifikationsbasierten Grammatik* haben wir nun die Voraussetzungen geschaffen, um formal zu definieren, wann ein String $w \in {V_T}^*$ ein grammatischer Satz in einer unifikationsbasierten Grammatik ist.

Wir rufen uns in Erinnerung, daß hierzu ein Syntaxbaum Syn existieren muß, sodaß

- die *Feature-Beschreibung* von Syn durch einen (wohlgeformten) Feature-Graphen erfüllt wird
- *der durch Syn abgeleitete String* mit w übereinstimmt

Wir wenden uns zunächst der formalen Definition der *Feature-Beschreibung* eines Syntaxbaumes zu. Sie ist sehr einfach:

Definition: Für einen Syntaxbaum

$$Syn =< N, daughters, cat, constraint >$$

heißt

$$\bigcup_{n \in N} constraint(n)$$

die *Feature-Beschreibung* von Syn. Wir notieren sie mit:

$$f\text{-}description(Syn) \qquad \square$$

Wir müssen nun noch definieren, wann ein Syntaxbaum Syn als Syntaxanalyse eines Strings $w \in {V_T}^*$ anzusehen ist. Obwohl dies anschaulich sehr einfach ist - die Blattkategorien des Syntaxbaumes müssen, von links nach rechts gelesen, den String w ergeben - bereitet dieses Vorhaben doch einige formale Schwierigkeiten. Es ist zwar sehr einfach die Blätter eines Syntaxbaumes zu identifizieren - es sind die Knoten, die keine auslaufenden Kanten besitzen - aber es ist relativ unangenehm sie von links nach rechts anzuordnen. Hierbei wird man sich offensichtlich der Funktion $daughters$ bedienen müssen.

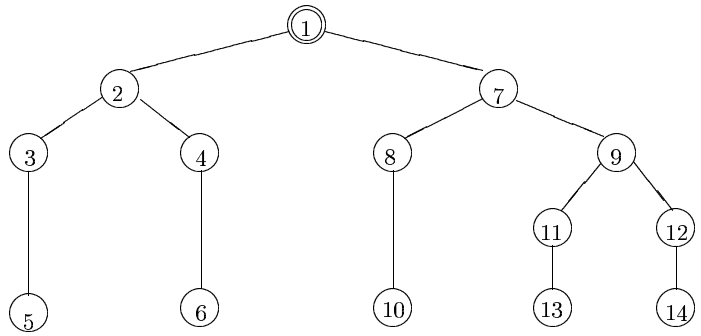
Wir werden im folgenden *den durch Syn abgeleiteten String* definieren. In gewohnter Weise reduzieren wir das Problem, indem wir zunächst das Grundgerüst des Syntaxbaumes - den Baum - betrachten. Der folgende (nicht-deterministische) Algorithmus berechnet für einen Baum seine *Blattfolge*. Wir können sehr einfach nachprüfen, daß der Algorithmus terminiert und ein eindeutiges Ergebnis liefert.

Definition: Sei $< N, daughters >$ ein Baum. Wir überführen seine Wurzel $s \in N$ in seine *Blattfolge*, indem wir die folgende Regel solange anwenden, bis sie nicht mehr anwendbar ist (wir notieren Knoten mit n, Knotenfolgen mit p und q):

$$\frac{p,\ n,\ q}{p,\ daughters(n),\ q}\ (daughters(n) \in N^+)$$

$\square$

Beispiel: Für den Baum



erhalten wir etwa:

1.	Schritt	1	(Wurzel)
2.	-„-	$2, 7$	$(n = 1)$
3.	-„-	$3, 4, 7$	$(n = 2)$
4.	-„-	$3, 4, 8, 9$	$(n = 7)$
5.	-„-	$3, 4, 8, 11, 12$	
		$\vdots$	
10.	Schritt	$5, 6, 10, 13, 14$	

Der Algorithmus liefert also eine mit unserer Anschauung übereinstimmende Blattfolge. □

Definition: Sei $Syn =< N, daughters, cat, constraint >$ ein Syntaxbaum. Sei $n_1 ... n_k \in N^+$ seine Blattfolge. Dann heißt

$$cat(n_1)...cat(n_k) \in (V_N \cup V_T)^*$$

der *durch Syn abgeleitete String*. Wir notieren ihn mit

$$c\text{–}string(Syn) \qquad \square$$

Wir wollen noch bemerken, daß der durch Syn abgeleitete String $c\text{–}string(Syn)$ unabhängig von der Knotenwahl N ist. D.h. sind $Syn \cong Syn'$ zwei isomorphe Syntaxbäume, so gilt:

$$c\text{–}string(Syn) = c\text{–}string(Syn')$$

Wir werden dieses Ergebnis nicht beweisen (auch keinen Syntaxbaumisomorphismus $\Phi : N \longrightarrow N'$ definieren), sondern vielmehr darauf hinweisen, daß ein ähnlich strenges Ergebnis für die Feature-Beschreibungen nicht gilt. Im allgemeinen gilt:

$$f\text{–}description(Syn) \neq f\text{–}description(Syn')$$

Dies liegt daran, weil bei der Variablen-Instantiierung in $(SYN2)$ die Variablen mit den Knoten des entsprechenden Teilbaumes indiziert werden. Es kann aber leicht gezeigt werden, daß:

$$f\text{–}description(Syn) = f\text{–}description(Syn') \quad \text{modulo Variablenumbenennung respektive } \Phi : N \longrightarrow N'$$

Wir verlassen dieses Thema und führen das Ableitungs-Symbol $\Longrightarrow^*$ ein, welches uns bereits durch die kontextfreien Grammatiken bekannt ist. Wir verallgemeinern in naheliegender Weise:

Notation: Sei C ein Constraint aus $\mathcal{L}$, $w \in {V_T}^*$. Wir schreiben:

$S \Longrightarrow^* w, C$
gdw. es gibt einen SyntaxbaumSyn mit:
(i) $c\text{–}string(Syn) = w$
(ii) $f\text{–}description(Syn) = C$ □

Für kontextfreie Grammatiken bedeutet die Notation $S \Longrightarrow^* w$, daß der String $w \in {V_T}^*$ ein grammatischer Satz ist. Wir wollen diese Notation auch für unifikationsbasierte Grammatiken übernehmen und definieren:

Definition: Sei $w \in {V_T}^*$. Dann gilt:

w ist ein *grammatischer Satz* oder
$S \Longrightarrow^* w$
gdw. es gibt einen Constraint$C \in \mathcal{L}$ mit:
(i) $S \Longrightarrow^* w, C$
(ii) $F \models C$ für einen (wohlgeformten) Featuregraphen F □

Bemerkung: Offensichtlich gilt:

$S \Longrightarrow^* w$
gdw. es gibt einen Syntaxbaum Syn mit:
(i) $w = c\text{–}string(Syn)$
(ii) $F \models f\text{–}description(Syn)$
für einen (wohlgeformten) Featuregraphen F

(Da wir als Annotation einer Produktion auch die leere Literalmenge zugelassen haben, kann u.U. $f\text{–}description(Syn) = \emptyset$ sein. Hierfür vereinbaren wir, daß jeder Featuregraph die leere Literalmenge erfüllt.) □

Damit haben wir unser Ziel erreicht. Die Bedingung, daß der Feature-Graph F *wohlgeformt* ist, steht in der Definition in Klammern (), weil nicht in allen Grammatikformalismen Wohlgeformtheitsbedingungen gefordert werden. Ferner haben wir gesehen, daß auf Wohlgeformtheitsbedingungen verzichtet werden können, wenn die Grammatik so verändert wird, daß ungrammatische Sätze eine *nicht erfüllbare* Feature-Beschreibung erhalten.
Trotzdem wollen wir an dieser Stelle - wie versprochen - die drei prominentesten Wohlgeformtheitsbedingungen für Feature-Graphen vorstellen:

- kein Knoten des Feature-Graphen darf mit zwei (oder mehr) Konstanten gelabelt sein (**constant clash-free**)
- kein Knoten des Feature-Graphen, der mit einer Konstanten gelabelt ist, darf eine auslaufende Kante besitzen (**constant/compound clash-free**)

- kein Knoten des Feature-Graphen darf Ausgangs- und Zielknoten desselben Pfades sein (**cycle-free**)

LFG besitzt darüber hinaus eine Vielzahl von weiteren Wohlgeformtheitsbedingungen. Wir werden bei Bedarf hierauf eingehen. Schließen wollen wir mit der Bemerkung, daß die vorgestellten drei Wohlgeformtheitsbedingungen oft in der jeweils gewählten Constraintsprache integriert werden. Wir haben für unsere Constraintsprache $\mathcal{L}$ darauf verzichtet und sind der Meinung, daß unser Vorgehen den Grammatikentwicklern einen etwas größeren Freiheitsgrad beläßt.

3.2 Parsing und Generierung in unifikationsbasierten Grammatiken

Obwohl wir es in der Definition eines in einer unifikationsbasierten Grammatik *grammatischen Satzes* nicht explizit ausgeführt haben, ist es klar, daß wir einem grammatischen Satz (wenigstens) eine Feature-Struktur zuweisen wollen. Hierfür müssen wir zunächst den Begriff der Feature-Struktur einführen. Unter einer *Featurestruktur* verstehen wir einen noch nicht völlig bestimmten Featuegraphen: die Wahl von Wurzeln und Variablenlabel soll noch offen sein. Wir definieren deshalb:

Definition: Sei $F =< N, E, source, target, R, Con \cup Var, Attr, label_N, label_E >$ ein Feature-Graph. [3] Dann heißt

$$< N, E, source, target, Con, Attr, label'_N, label_E >$$

mit:

$$label'_N(n) =_{def} label_N(n) \cap Con \qquad \forall n \in N$$

seine *Featurestruktur*. Wir notieren sie mit $f\text{–}structure(F)$ □

Wir wollen den Begriff der Featurestruktur auch für Constraints und Syntaxbäume benutzen und definieren deshalb:

Definition Sei C ein erfüllbarer Constraint und F ein Minimalmodell für C. Dann schreiben wir:

$$f\text{–}structure(C) =_{def} f\text{–}structure(F)$$

Ist Syn ein Syntaxbaum mit erfüllbarer Feature-Beschreibung und F ein Minimalmodell für $f\text{–}description(Syn)$, so schreiben wir:

$$f\text{–}structure(Syn) =_{def} f\text{–}structure(F) \qquad \square$$

Bemerkung: In Kapitel 2 haben wir ein Verfahren kenngelernt, mit dessen Hilfe wir entscheiden können, ob ein Constraint C erfüllbar ist. Zusätzlich haben wir ein Verfahren kennengelernt, mit dem wir aus jeder Normalform eines erfüllbaren Constraints einen Featuregraphen konstruieren können, der die Normalform erfüllt. Damit dieser Featuregraph den Constraint selbst erfüllt, ist es nötig, die Knotenlabelung des Featuregraphen um die im Normalisierungsverfahren eventuell eliminierten Variablen zu erweitern. Somit sind wir in der Lage, erstens zu entscheiden, ob ein Constraint erfüllbar ist und zweitens, für einen erfüllbaren Constraint ein Minimalmodell anzugeben. Wir wollen an dieser Stelle noch

[3] um die Wurzelmenge des Featuregraphen nicht mit dem Startsymbol S zu verwechseln, bezeichnen wir sie im folgenden mit R (ftir englisch *Root*)

anmerken (ohne es zu beweisen), daß für einen Constraint C aus $\mathcal{L}$ alle Minimalmodelle zueinander isomorph sind - abgesehen von der speziellen Wahl einer Wurzelmenge. Also sind die Featurestrukturen eines Constraints und die Featurestruktur eines Syntaxbaumes (bis auf Isomorphie) eindeutig bestimmt □

Somit liegt die folgende Definition nahe:

Definition: Ein String $w \in {V_T}^*$ ist *mit Feature-Struktur F ableitbar*

gdw. (i) $S \Longrightarrow^* w, C$
(ii) C ist erfüllbar
(iii) $F = f\text{–}structure(C)$ □

Offensichtlich äquivalent ist:

Definition: Ein String $w \in {V_T}^*$ ist *mit Feature-Struktur F ableitbar*

gdw. es gibt einen Syntaxbaum Syn mit:
(i) $w = c\text{–}string(Syn)$
(ii) $f\text{–}description(Syn)$ ist erfüllbar
(iii) $F = f\text{–}structure(Syn)$ □

Bemerkung: Die Featurestruktur ist unabhängig von der speziellen Wahl der Knotenmenge des Syntaxbaumes Syn (d.h. isomorphe Syntaxbäume haben isomporphe Featurestrukturen). □

Bezeichnet $\mathcal{F}$ die Menge aller Feature-Strukturen - wobei zueinander isomorphe Feature-Strukturen als gleiche Feature-Strukturen angesehen werden - so können wir die Satz-Bedeutungs-Relation einer unifikationsbasierten Grammatik folgendermaßen als binäre Relation $\Delta \subseteq {V_T}^* \times \mathcal{F}$ formalisieren:

Definition: Sei $w \in {V_T}^*$ und F eine Feature-Struktur. Wir schreiben:

$\Delta(w, F)$ **gdw.** w ist mit Feature-Struktur F ableitbar □

Für unifikationsbasierte Grammatiken können wir nun in relativ abstrakten Termen definieren, was wir unter einem *Parser* und einem *Generator* verstehen:

- ein *Parser* ist ein Algorithmus, der für vorgegebenes $w \in {V_T}^*$ die Menge

  $$\Delta_w =_{def} \{F \in \mathcal{F} | \Delta(w, F)\}$$

  aufzählt

- ein *Generator* ist ein Algorithmus, der für vorgegebenes $F \in \mathcal{F}$ die Menge

  $$\Delta_F =_{def} \{w \in {V_T}^* | \Delta(w, F)\}$$

  aufzählt

Ob adäquate Algorithmen gefunden werden können, hängt von der Entscheidbarkeit des korrespondierenden Parsing- und Generierungproblems ab. Das *Generierungsproblem* besteht darin, für eine vorgegebene Feature-Struktur F zu entscheiden, ob

$$\Delta_F \neq \emptyset$$

Die Entscheidbarkeit des Generierungsproblems garantiert die Existenz eines Algorithmus, der in jedem Fall terminiert und einen Output liefert, obwohl er (natürlich) nicht in der Lage sein muß, alle auftretenden Lösungen zu liefern (z.B. wenn Δ_F keine endliche Menge ist). Wedekind hat in [9] gezeigt, daß das Generierungsproblem entscheidbar ist.

Wir werden in dieser Arbeit Grammatikeigenschaften formulieren, sodaß für beliebig vorgegebenes $F \in \mathcal{F}$ die Menge Δ_F endlich ist. Diese Grammatikeigenschaften garantieren also die Existenz eines terminierenden Generators.

Kapitel 4

Generierung in unifikationsbasierten Grammatiken - ein Überblick

In diesem Kapitel werden wir die folgenden vier Themen besprechen:

- Unentscheidbarkeit des Pfad-Parsing-Problems, VanNoord [5]
- Semantisch monotone Grammatiken, Shieber [6]
- Entscheidbarkeit des Generierungsproblems, Wedekind [9]
- Unentscheidbarkeit ambiguitätserhaltender Übersetzung, Wedekind und Kaplan [10]

Wir sind auf die Bedeutung dieser vier Arbeiten - im Hinblick auf Generierung in unifikationsbasierten Grammatiken - schon in der Einleitung eingegangen. Wir fassen die für uns wichtigen Punkte kurz zusammen:

- Shieber und VanNoord fassen Parsing und Generierung als Deduktionsprozess auf. Je nach Wahl der Parameter des Deduktionsprozess läuft er als Parsing oder Generierung ab. (In VanNoords Ansatz ist SEM-Parsing der Generierungsprozess und PHON-Parsing der Parsingprozess; je nach Wahl des Parameters *Pfad* läuft Pfad-Parsing als Generierung oder Parsing ab.)
- Shieber führt den Begriff der *semantisch monotonen* Grammatiken ein und führt aus, daß solche Grammatiken Generierung gestatten könnten.
- VanNoord zeigt, daß das Pfad-Parsing-Problem unentscheidbar ist. Aus diesem Grund sucht er sowohl nach Steuerungsstrategiuen, die den Deduktionsprozess terminieren lassen, als auch nach Grammatikrestriktionen, die den Aufbau phonologischer und semantischer Strukturen betreffen.

- Wedekind fasst - im Gegensatz zu VanNoord - Generierung nicht als Prozess auf, Sätze zu finden, die eine gegebene Teil-Featurestruktur tragen, sondern als Prozess, Sätze zu erzeugen, die eine vorgegebene Voll- Featurestruktur tragen. Wedekind zeigt, daß in seinem Ansatz das Generierungsproblem entscheidbar ist. Ein *Generator*, d.h. ein Algorithmus, welcher alle Sätze aufzählt, die eine vorgegebene Featurestruktur tragen, muß im allgemeinen jedoch nicht existieren.
- Wedekind und Kaplan zeigen in einem einfachen Beweis, daß es unentscheidbar ist, ob zu zwei (oder mehr) vorgegebenen Featurestrukturen ein Satz existiert, der diese Bedeutungen trägt. Aus diesem Grund kann ambiguitätserhaltende Übersetzung ebenfalls nicht entscheidbar sein.

Unrestringierte unifikationsbasierte Grammatiken erlauben also die Entscheidbarkeit des Generierungsproblems, vorausgestzt man betreibt die Generierung im Wedekind'schen Sinne (was wegen VanNoords Ergebnis naheliegend ist). Ein Generator wird aber selbst dann nicht immer existieren. Ferner halten wir fest, daß mit unrestringierten unifikationsbasierten Grammatiken keine ambiguitätserhaltende Übersetzung möglich ist.

4.1 Unentscheidbarkeit des Pfad-Parsing-Problems

In seiner Dissertation [5] (Seiten 54-62) betrachtet VanNoord das *unrestringierte Parsing-Problem* und das *Pfad-Parsing-Problem*. Trotz des Nomens „Parsing“ handelt es sich bei beiden Problemen sowohl um Parsing als auch um Generierung.

Im Abschnitt 4.1.1 werden wir VanNoords Auffassung von Parsing und Generierung kennenlernen. Dabei auftretende Schwierigkeiten motivieren ihn, seine Auffassung zu modifizieren und den Begriff des Pfad-Parsing-Problems einzuführen. Wir werden in Abschnitt 4.1.2 diese Modifikation kennenlernen. Im letzten Abschnitt werden wir dann VanNoords Beweis, daß das Pfad-Parsing-Problem unentscheidbar ist, nachvollziehen.

Der Einfachheit halber werden wir dafür *nicht* den VanNoord'schen, sondern den in den vorangehenden Kapiteln vorgestellten Formalismus benutzen.

4.1.1 Das unrestringierte Parsing-Problem

In einer unifikationsbasierten Grammatik des Englischen könnte der Satz

Mexican priests drink whisky

die folgende Featurestruktur tragen:

$$\begin{bmatrix} \text{SYN} & \text{S} \\ \text{PHON} & \text{„mexican priests drink whisky“} \\ \text{SEM} & \text{drink([mexican](priests), whisky)} \end{bmatrix}$$

Für eine gegebene Grammatik besteht das *Parsing-Problem* darin, einen String zu spezifizieren. Ein *Parser* zählt alle Sätze auf, die diesen String als Wert des PHON-Attributs tragen.
Die Spezifikation einer semantischen Repräsentation ist ein *Generierungs-Problem*. Ein *Generator* zählt alle Sätze auf, die diese semantische Repräsentation als Wert des SEM-Attributs tragen.
Zum Beispiel besteht das Parsen des Strings „mexican priests drink whisky“ darin, alle Sätze aufzuzählen, die eine Featurestruktur

$$\begin{bmatrix} \text{...} & \\ \text{PHON} & \text{„mexican priests drink whisky“} \\ \text{...} & \end{bmatrix}$$

besitzen. Ein Generierungsvorgang könnte darin bestehen, alle Sätze aufzuzählen, die eine Featurestruktur

$$\begin{bmatrix} \text{...} & \\ \text{SEM} & \text{drink([mexican](priests), whisky)} \\ \text{...} & \end{bmatrix}$$

tragen. Anstelle der Pfade PHON und SEM können wir natürlich auch andere Pfade benutzen. Ebenso ist es uns freigestellt, weitere Constraints hinzuzufügen, etwa die syntaktische Kategorie der Sätze, an der wir interessiert sind. Ein typischer Generierungsvorgang ist z.B. die Aufzählung aller Sätze mit einer Featurestruktur

$$\begin{bmatrix} \text{...} & \\ \text{CAT} & \text{S} \\ \text{SUBCAT} & <> \\ \text{SEM} & \text{see(graham, drink([mexican](priests), [strong](whisky)))} \\ \text{...} & \end{bmatrix} .$$

Wir stehen nun vor der Aufgabe, diese (einheitliche) Sichtweise von Parsing und Generierung zu formalisieren. Wir erinnern uns, daß wir einen grammatischen Satz $w \in {V_T}^*$ vor uns haben, wenn:

(i) $\quad S \Longrightarrow^* w, C$
(ii) $\quad F \models C$ für einen (wohlgeformten) Featuregraphen F

Hierbei ist C die Feature-Beschreibung und w der abgeleitete String eines (geeigneten) Syntaxbaumes. Ist $s \in N$ die Wurzel dieses Syntaxbaumes, so steht x_s für die dort vorhandene Featurestruktur. Die entscheidende Beobachtung ist nun, daß VanNoord Constraints *immer* so formuliert, daß sie mit x_s annotiert werden können. Für unser letztes Beispiel erhalten wir etwa den folgenden Constraint:

$$\begin{aligned} x_s \ CAT &\doteq S \\ x_s \ SUBCAT &\doteq <> \\ x_s \ SEM &\doteq see(...) \end{aligned}$$

Das unrestringierte Parsing-Problem besteht für VanNoord darin, zu einem vorgegebenen Constraint $\Phi[x_s]$ zu entscheiden, ob es einen grammatischen Satz gibt, dessen Featurestruktur mit $\Phi[x_s]$ *kompatibel* ist. Wir definieren deshalb:

Definition: Das *unrestringierte Parsing-Problem* bestehet darin, für einen vorgegebenen Constraint $\Phi[x] \in \mathcal{L}$ zu entscheiden, ob ein $w \in {V_T}^*$ existiert, sodaß:

(i) $\quad S \Longrightarrow^* w, C$
(ii) $\quad F \models C \cup \Phi[x|x_s]$ für einen (wohlgeformten) Featuregraphen F

Hierbei ist $s \in N$ die Wurzel eines Syntaxbaumes Syn mit $w = c\text{–}string(syn)$ und $C = f\text{–}description(Syn)$ $\quad \square$

Beispiele: In Abhängigkeit von der speziellen Wahl von Φ werden wir generieren oder parsen:

- Für $\Phi = \{xPHON \doteq$ „priests drink mexican whisky“ $\}$ liegt ein Parsing-Problem vor
- Für $\Phi = \{xSEM \doteq \text{drink(priests, [mexican](whisky))}\}$ liegt ein Generierungs-Problem vor $\quad \square$

Wir wenden uns nun der Frage zu, wie gut dieser Ansatz ist. Angenommen wir haben eine Grammatik vorliegen, die grammatische Sätze mit den folgenden Featurestrukturen zulässt:

$$\begin{bmatrix} \text{PHON} & \text{„the priest drinks“} \\ \text{SEM} & \begin{bmatrix} \text{PRED} & \text{drink} \\ \text{ARG1} & \begin{bmatrix} \text{PRED} & \text{priest} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

und:

$$\begin{bmatrix} \text{PHON} & \text{„the priest drinks whisky“} \\ \text{SEM} & \begin{bmatrix} \text{PRED} & \text{drink} \\ \text{ARG1} & \begin{bmatrix} \text{PRED} & \text{priest} \end{bmatrix} \\ \text{ARG2} & \begin{bmatrix} \text{PRED} & \text{whisky} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Für eine vorgebene logische Form

$$\begin{bmatrix} \text{PRED} & \text{drink} \\ \text{ARG1} & \begin{bmatrix} \text{PRED} & \text{priest} \end{bmatrix} \end{bmatrix}$$

würde ein Generator die Sätze

the priest drinks
the priest drinks whisky

und vielleicht auch noch den Satz

the priest drinks strong cheap whisky from a brown paper bag

aufzählen. Andererseits würde er den Satz

the priest drinks

für die logische Form

$$\begin{bmatrix} \text{PRED} & \text{drink} \\ \text{ARG1} & \begin{bmatrix} \text{PRED} & \text{priest} \end{bmatrix} \\ \text{ARG2} & \begin{bmatrix} \text{PRED} & \text{whisky} \end{bmatrix} \end{bmatrix}$$

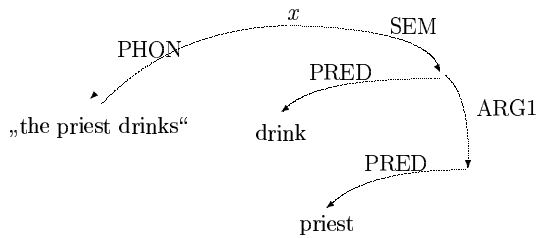
ermitteln. Alle vom Generator ermittelten Sätze haben eine Featurestruktur, deren SEM-Wert mit dem vorgegebenen SEM-Wert kompatibel ist. Dieses Verhalten ist sicherlich nicht erwünscht. Vielmehr wünschen wir uns einen Generator, der nur die Sätze aufzählt, deren SEM-Wert exakt mit dem vorgegebenen SEM-Wert übereinstimmt.

4.1.2 Das Pfad-Parsing-Problem

Im vorangehenden Abschnitt haben wir ausgeführt, daß wir uns einen Generator wünschen, der alle Sätze aufzählt, deren semantische Repräsentation exakt mit einer vorgegebenen semantischen Repräsentation übereinstimmt. Wir haben bemerkt, daß unrestringiertes Parsing dies nicht zu leisten vermag.
In Wedekind [7] wird Generierung in LFG betrachtet. Dort wird formal ausgeführt, daß die Input-Struktur die generierte Struktur *subsummieren* [1] sollte. Diese Bedingung nennt Wedekind *Completeness*. Andererseits sollte die generierte Struktur auch die Input-Struktur subsummieren. Diese Bedingung wird von Wedekind *Coherence* genannt.
VanNoord kritisiert diesen Ansatz insoweit, als er ausführt, daß Completeness und Coherence nur für *den* Pfad verlangt werde sollte, der innerhalb der Featurestruktur zu der semantischen Repräsentation führt.
VanNoord überträgt diesen Gedanken dann in naheliegender Weise auf Parsing. Hier fordert er Completeness und Coherence für *den* Pfad, der innerhalb der Featurestruktur zur String-Repräsentation führt.
Die Verallgemeinerung der Forderungen Completeness und Coherence auf beliebige (aber festgelegte) Pfade führt dann zum Begriff des Pfad-Parsings.

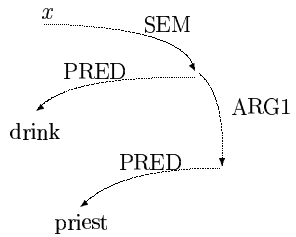
Bevor wir uns der formalen Definition des Pfad-Parsing-Problems zuwenden können, müssen wir uns mit Featuregraphen, Pfaden und Restriktionen von Featuregraphen auf Pfaden beschäftigen.

Beispiel: Für den folgenden Featuregraphen F

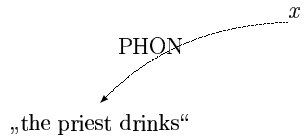


wollen wir unter der Restriktion von F auf den Pfad $xSEM$ den Featuregraphen

[1] In etwas vagen Termen könnten wir formulieren: a subsummiert b **gdw.** a ist allgemeiner, weniger informativ als b



verstehen. Die Restriktion von F auf den Pfad $xPHON$ ist der Featuregraph:



Wir bemerken, daß beide Restriktionen *Subgraphen* [2] von F sind und existieren (definiert sind), weil $[\![xSEM]\!]^F$ und $[\![xPHON]\!]^F$ definiert sind □

Formal können wir definieren:

Definition: Sei F ein Featuregraph und p ein Term, sodaß $[\![p]\!]^F$ definiert ist. Dann sei $p{\sim}F$ der kleinste Subgraph von F, sodaß:

$$(REST) \qquad [\![p\ell]\!]^F \text{ definiert }, \ell \in Attr^* \Longrightarrow [\![p\ell]\!]^{p\sim F} \text{ definiert}$$

$p{\sim}F$ heißt *Restriktion von F auf p* □

Da $[\![p]\!]^F$ definiert ist, liefert $(REST)$ insbesondere, daß $[\![p]\!]^{p\sim F}$ definiert ist. Dies liefert die Existenz des Featuregraphen $p \sim F$. Die Minimalitätsforderung „kleinster Subgraph von F “ liefert die Eindeutigkeit von $p \sim F$. Wir gehen hierauf nicht näher ein, sondern wenden uns nun der formalen Definition des Pfad-Parsing-Problems zu.

Definition: Sei $\ell \in Attr^*$. Das *ℓ-Parsing-Problem* besteht darin, für einen vorgegebenen Constraint $\Phi[x] \in \mathcal{L}$ zu entscheiden, ob ein $w \in {V_T}^*$ existiert, sodaß:

$$\begin{array}{ll} (i) & S \Longrightarrow^* w, C \\ (ii) & C \cup \Phi[x|x_s] \text{ ist erfüllbar} \\ (iii) & p \sim F^C = p \sim F^\Phi \end{array}$$

Hierbei ist $s \in N$ die Wurzel eines Syntaxbaumes Syn mit $w = c{-}string(Syn)$ und $C = f{-}description(Syn)$. Ferner ist F^C ein Minimalmodell für C, F^Φ ein Minimalmodell für $\Phi[x|x_s]$ und $p = x_s\ell$ □

Bemerkungen:

- Offensichtlich ist jede Lösung $w \in {V_T}^*$ eines ℓ-Parsing-Problems stets auch Lösung des unrestringierten Parsing-Problems (zum selben vorgegebenen $\Phi[x] \in \mathcal{L}$)
- In den nachfolgenden Beispielen wird stets $p \sim F^\Phi = F^\Phi$ gelten. (Wir können uns unter (iii) also darauf beschränken, die Gleichung $p \sim F^C = F^\Phi$ nachzuprüfen.) VanNoord lässt aber ausdrücklich Constraints Φ zu, deren Minimalmodell *kein* Subgraph von $p \sim F^\Phi$ ist.

Wir wollen anhand unserer Beispiele aus Abschnitt 4.1.1 nachprüfen, ob SEM-Parsing tatsächlich nur Sätze aufzählt, deren semantische Repräsentation *exakt* mit einer vorgegebenen semantischen Repräsentation übereinstimmt.

Coherence:

$$\begin{array}{lll} \text{Sei:} & \Phi & = \{ \ xSEM \ \ PRED \doteq drink, \\ & & \quad\ xSEM \ \ ARG1 \ \ PRED \doteq priest\} \end{array}$$

Der Constraint $\Phi[x|x_s]$ hat das Minimalmodell F^Φ:

$$\underset{x_s}{\left[\text{SEM}\left[\begin{array}{ll}\text{PRED} & \text{drink}\\ \text{ARG1} & \left[\ \text{PRED} \quad \text{priest}\ \right]\end{array}\right]\right]}$$

Unsere Beispielgrammatik war so gewählt, daß unrestringiertes Parsing die folgenden Sätze lieferte:

the priest drinks
the priest drinks whisky
the priest drinks strong cheap whisky from a brown paper bag

SEM-Parsing liefert den Satz:

the priest drinks

[2] Seien F, F' zwei (Feature-)Graphen. F' heißt *Subgraph* von F, falls jede Kante und jeder Knoten von F' (inclusive Label) auch in F vorkommt.

mit Minimalmodell F^C:

$$\left[x_s \left[\begin{array}{ll} \text{PHON} & \text{„the priest drinks“} \\ \text{SEM} & \left[\begin{array}{ll} \text{PRED} & \text{drink} \\ \text{ARG1} & \left[\text{PRED} \quad \text{priest} \right] \end{array} \right] \end{array} \right] \right]$$

Zur Begründung rechnen wir einfach nach, daß:

$$x_s SEM \sim F^C = x_s SEM \sim F^\Phi \quad .$$

Die Sätze:

the priest drinks whisky
the priest drinks strong cheap whisky from a brown paper bag

sind hingegen keine Lösungen des SEM-Parsing-Problems, da die Restriktionen ihrer Minimalmodelle auf $x_s SEM$ nicht mit F^Φ übereinstimmen. (Wir rechnen leicht nach, daß sie *informativer* als F^Φ sind:

F^Φ ist echter Subgraph von $x_s SEM \sim F^C$.

Diese Sätze verletzen also die Coherence-Bedingung, daß die generierte Struktur (in relevanten Teilen) weniger informativ als die Input-Struktur sein soll.)

Completeness:

$$\begin{array}{lll} \text{Sei:} & \Phi & = \{ \; xSEM \;\; PRED \doteq drink, \\ & & \quad\; xSEM \;\; ARG1 \;\; PRED \doteq priest\} \\ & & \quad\; xSEM \;\; ARG2 \;\; PRED \doteq whisky\} \end{array}$$

Der Constraint $\Phi[x|x_s]$ hat das Minimalmodell F^Φ:

$$\left[x_s \left[\text{SEM} \left[\begin{array}{ll} \text{PRED} & \text{drink} \\ \text{ARG1} & \left[\text{PRED} \quad \text{priest} \right] \\ \text{ARG2} & \left[\text{PRED} \quad \text{whisky} \right] \end{array} \right] \right] \right]$$

Unrestringiertes Parsing lieferte hier die Sätze:

the priest drinks
the priest drinks whisky

SEM-Parsing liefert den Satz:

the priest drinks whisky

jedoch nicht den Satz:

the priest drinks

da dessen Minimalmodell F^C:

$$\left[x_s \left[\begin{array}{ll} \text{PHON} & \text{„the priest drinks“} \\ \text{SEM} & \left[\begin{array}{ll} \text{PRED} & \text{drink} \\ \text{ARG1} & \left[\text{PRED} \quad \text{priest} \right] \end{array} \right] \end{array} \right] \right]$$

nicht die Bedingung

$$x_s SEM \sim F^C = x_s SEM \sim F^\Phi$$

erfüllt. (Die Restriktion von F^C auf $x_s SEM$ ist ein echter Subgraph von F^Φ, also weniger informativ als F^Φ. Bei diesem Satz ist also die Completeness-Bedingung verletzt.)

Bemerkung: Die Bedingung (*iii*) im ℓ-Parsing-Problem lässt sich folgendermassen in die Coherence- und Completeness-Bedingung aufspalten:

(Coherence)	$p \sim F^C$ ist Subgraph von $p \sim F^\Phi$
(Completeness)	$p \sim F^\Phi$ ist Subgraph von $p \sim F^C$

4.1.3 Post's Correspondence Problem

In diesem Abschnitt werden wir VanNoords Beweis nachvollziehen, daß das Pfad-Parsing-Problem unentscheidbar ist.
Ein *Ja-Nein-Problem* ist *unentscheidbar*, wenn es keinen Algorithmus gibt, der als Input eine Instanz des Problems nimmt und bestimmt, ob die Antwort auf die Frage nach einer Lösung dieser Instanz „ja“ oder „nein“ lautet.
Das folgende Problem ist als *Post's Correspondence Problem* (PCP) bekannt. Hopcroft und Ullmann haben 1979 in [1] bewiesen, daß es unentscheidbar ist. VanNoord benutzt es, um zu zeigen, daß das Pfad-Parsing-Problem unentscheidbar ist.

Definition: Eine Instanz des PCP besteht aus zwei Folgen

$$A = v_1 ... v_k$$
$$B = w_1 ... w_k$$

gleicher Länge $k \in \mathbb{N}$. Hierbei sind $v_1, ..., v_k, w_1, ..., w_k$ Strings über einem Alphabet Σ. Die Instanz hat eine Lösung, falls es irgendeine Folge

$$i_1 ... i_m \qquad (m \geq 1)$$

gibt, sodaß:

$$v_{i_1} + ... + v_{i_m} = w_{i_1} + ... + w_{i_m}$$

(Hierbei steht „+“ für die Stringkonkatenation.) □

Beispiel: Sei $A = v_1 ... v_3$, $B = w_1 ... w_3$ mit:

$$\begin{array}{ll} v_1 = \text{„1“} & w_1 = \text{„111“} \\ v_2 = \text{„10111“} & w_2 = \text{„10“} \\ v_3 = \text{„10“} & w_3 = \text{„0“} \end{array}$$

Die Folge $2, 1, 1, 3$ ist eine Lösung dieser Instanz, da:

$$\begin{array}{rl} v_2 + v_1 + v_1 + v_3 & = \text{„10111“} + \text{„1“} + \text{„1“} + \text{„10“} \\ & = \text{„10“} + \text{„111“} + \text{„111“} + \text{„0“} \\ & = w_2 + w_1 + w_1 + w_3 \end{array}$$

(Natürlich hat nicht jede Instanz des PCP eine Lösung. Z.B. ist dies offensichtlich für $A = \text{„1“}$ und $B = \text{„0“}$ der Fall.) □

Wir werden nun angeben, wie wir jede Instanz des PCP als unifikationsbasierte Grammatik encodieren können. Die Frage nach einer Lösung dieser Instanz wird äquivalent zu einem Pfad-Parsing-Problem sein. Wir definieren:

PCP-Grammatik:

$$\begin{array}{ll} I \longrightarrow 1 & \{x_{n_0} A \doteq v_1,\ x_{n_0} B \doteq w_1\} \\ \vdots & \\ I \longrightarrow k & \{x_{n_0} A \doteq v_k,\ x_{n_0} B \doteq w_k\} \\ I \longrightarrow II & \{x_{n_0} A \doteq x_{n_1} A + x_{n_2} A,\ x_{n_0} B \doteq x_{n_1} B + x_{n_2} B\} \\ S \longrightarrow I & \{x_{n_0} SOLUTION \doteq YES,\ x_{n_1} A \doteq x_{n_1} B\} \end{array}$$

(Hierbei sind $A, B, SOLUTION$ Attribute und YES eine Konstante.) □

Zunächst möchten wir bemerken, daß wir in dieser Grammatik nicht definierte Constraints benutzt haben:

- Für einen Term t und einen String $\sigma \in \Sigma^*$ ist

$$t \doteq \sigma$$

nicht definiert

- Für Terme t, d_1, d_2 ist

$$t \doteq d_1 + d_2$$

nicht definiert

Da die Benutzung von Strings und der Stringkonkatenation in unifikationsbasierten Grammatiken durchaus üblich ist und deren formale Definition wenig zum Verständnis des folgenden beiträgt, möchten wir an dieser Stelle auf den Anhang verweisen, in welchem wir ausführlich auf die Encodierung von Strings in der Constraintsprache $\mathcal{L}$ eingehen.

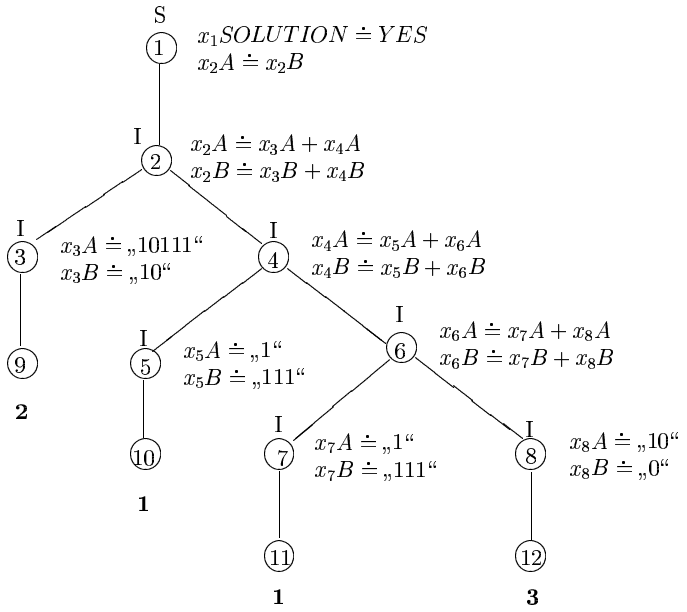
Beispiel: Für unsere Beispiel-Instanz des PCP erhalten wir etwa das PCP-Grammatik-Lexikon:

$$\begin{array}{lll} I \longrightarrow 1 & \{x_{n_0} A \doteq \text{„1“}, & x_{n_0} B \doteq \text{„111“}\} \\ I \longrightarrow 2 & \{x_{n_0} A \doteq \text{„10111“}, & x_{n_0} B \doteq \text{„10“}\} \\ I \longrightarrow 3 & \{x_{n_0} A \doteq \text{„10“}, & x_{n_0} B \doteq \text{„0“}\} \end{array}$$

Wir werden nun zeigen, daß

2113

ein grammatischer Satz für diese PCP-Grammatik ist. Betrachte dazu den folgenden Syntaxbaum:



Seine Feature-Beschreibung:

$x_1 SOLUTION \doteq YES$
$x_2 A \doteq x_2 B$
$x_2 A \doteq x_3 A + x_4 A$
$x_2 B \doteq x_3 B + x_4 B$
$x_3 A \doteq$ „10111“
$x_3 B \doteq$ „10“
$x_4 A \doteq x_5 A + x_6 A$
$x_4 B \doteq x_5 B + x_6 B$
$x_5 A \doteq$ „1“
$x_5 B \doteq$ „111“
$x_6 A \doteq x_7 A + x_8 A$
$x_6 B \doteq x_7 B + x_8 B$
$x_7 A \doteq$ „1“
$x_7 B \doteq$ „111“
$x_8 A \doteq$ „10“
$x_8 B \doteq$ „0“

ist erfüllbar.Wir erhalten (als Lösung des Normalisierungsverfahrens) den folgenden Featuregraphen F (in Matrixrepräsentation):

x_1[SOLUTION YES]

x_2[A „101111110“ ; B „101111110“]

x_3[A „10111“ ; B „10“]

x_4[A „1110“ ; B „1111110“]

x_5[A „1“ ; B „111“]

x_6[A „110“ ; B „1110“]

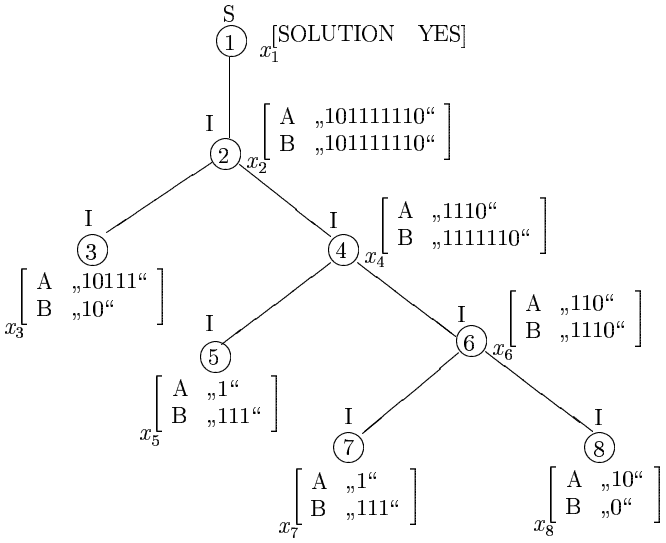
x_7[A „1“ ; B „111“]

x_8[A „10“ ; B „0“]

(F hat die Wurzeln $x_1, ..., x_8$.) Einen sehr guten Eindruck erhalten wir, wenn wir die Subgraphen

$$x_i \sim F \qquad (i = 1, 2, ..., 8)$$

in Matrixräpresentation an den entsprechenden Knoten des Syntaxbaumes eintragen:



(Die Blätter des Syntaxbaumes haben wir nicht in die Abbildung eingetragen.)

Das folgende Lemma ist leicht zu verifizieren:

Lemma: $i_1 ... i_m$ ist Lösung der PCP-Instanz:
$A = v_1 ... v_k$
$B = w_1 ... w_k$
gdw. Für die (dieser Instanz zugeordnete) PCP-Grammatik gilt:
$S \Longrightarrow^* i_1 ... i_m$ □

Wir beschliessen nun diesen Abschnitt mit VanNoords Ergebnis:

Satz: Das ℓ-Parsing-Problem ist unentscheidbar $\quad \square$

Beweis: Angenommen, das ℓ-Parsing-Problem ist entscheidbar. Dann gibt es einen Algorithmus, der für jede Instanz - d.h. für jede unifikationsbasierte Grammatik und jedes $\Phi[x] \in \mathcal{L}$ - entscheidet, ob es ein $w \in {V_T}^*$ gibt, sodaß:

$$\begin{array}{ll} (i) & S \Longrightarrow^* w, C \\ (ii) & C \cup \Phi[x|x_s] \text{ ist erfüllbar} \\ (iii) & p \sim F^C = p \sim F^\Phi \end{array}$$

(Hierbei ist $s \in N$ die Wurzel eines Syntaxbaumes Syn mit $w = c\text{–}string(Syn)$ und $C = f\text{–}description(Syn)$. F^C ist ein Minimalmodell für C, F^Φ ein Minimalmodell für $\Phi[x|x_s]$ und $p = x_s\ell$)

Wir setzen:

$$\begin{array}{l} \ell =_{def} SOLUTION \\ \Phi =_{def} \{xSOLUTION \doteq YES\} \end{array}$$

und zeigen, daß für jede PCP-Grammatik gilt:

$$(i), (ii), (iii) \quad \textbf{gdw.} \quad S \Longrightarrow^* w$$

„$\Rightarrow$": trivial (aus (ii) folgt insbesondere, daß C erfüllbar ist)
„$\Leftarrow$": Sei also $S \Longrightarrow^* w$. Dann gibt es einen Syntaxbaum Syn, sodaß:

$$\begin{array}{l} w = c\text{–}string(Syn) \\ C = f\text{–}description(Syn) \text{ ist erfüllbar} \end{array}$$

Somit folgt insbesondere (i). Wegen der Grammatikregel:

$$S \longrightarrow I \qquad \{x_{n_0}SOLUTION \doteq YES,\ \ x_{n_1}A \doteq x_{n_1}B\}$$

gilt:

$$\{x_sSOLUTION \doteq YES\} \in C$$

D.h. es gilt (ii).
Da das Grammatiksymbol S in jedem Syntaxbaum einer PCP-Grammatik genau einmal vorkommt, kann $x_sSOLUTION$ kein Subterm eines weiteren Literals von C sein. Hieraus folgt (iii).

Somit haben wir gezeigt, daß es einen Algorithmus gibt, der für jede PCP-Grammatik entscheidet, ob es ein $w \in {V_T}^*$ gibt, sodaß:

$$S \Longrightarrow^* w$$

Mit dem Lemma erhalten wir einen Algorithmus, der für jede PCP-Instanz entscheidet, ob es eine Lösung für diese PCP-Instanz gibt. Widerspruch $\square$

Bemerkung: Am Anfang des Abschnittes hatten wir argumentiert, daß das unrestringierte Parsing-Problem nicht unseren linguistischen Ansprüchen genügt. Wir können nun nachtragen, daß es auch formalen Ansprüchen nicht genügt: mit derselben Argumentation wie im eben vorgestellten Beweis können wir zeigen, daß auch das unrestringierte Parsing-Problem unentscheidbar ist. Wir überlassen die Details dem Leser $\square$

4.2 Semantisch monotone Grammatiken

In diesem Abschnitt werden wir Shiebers Arbeit [6] besprechen. Shieber stellt einen Deduktionsprozess vor, dessen Parameter so gewählt werden können, daß der Deduktionsprozess entweder als Parsing oder Generierung abläuft. Den Generierungsvorgang kann Shieber allerdings nur für *semantisch monotone* Grammatiken anwenden. Wir werden den Deduktionsprozess vorstellen, Parsing- und Generierungs-Beispiele rechnen und hierbei insbesondere herauszustellen versuchen, warum Shieber die Restriktion auf semantisch monotone Grammatiken benötigt.

Wir beginnen mit einer kurzen Zusammenfassung; jedes *Deduktionssystem* besteht grob gesagt aus:

- Formeln, Axiomen, Inferenzregeln, Aussagen
- einer Interpretationen (der Formeln) und einer Wahrheitsfunktion für (interpretierte) Formeln

Die *Axiome* bestehen aus einer Teilmenge der Formeln. Die *Inferenzregeln* überführen Formeln in Formeln. *Aussagen* sind Formeln, die entweder Axiome sind, oder durch Anwendung von Inferenzregeln auf Aussagen entstanden sind. Ein Deduktionssystem ist *korrekt*, wenn jede Aussage wahr ist. Ein Deduktionssystem ist *vollständig*, wenn jede wahre Formel eine Aussage ist.

Wir wollen nun Shiebers Deduktionssystem kennenlernen. Als Formeln benutzt er die aus dem *Earley-Algorithmus* bekannten *Items*:

$$[i, N \longrightarrow V_1 ... V_{m-1} \bullet V_m ... V_n, j]$$

(Hierbei sind $i \leq j$ und $N, V_1, ... V_n$ Grammatiksymbole.) Shieber *interpretiert* diese Formeln (für einen gegebenen String) als *wahr*, falls:

> Es existiert eine Stringposition $l \geq j$, sodaß der Substring zwischen i und l als N klassifiziert werden kann, vorausgesetzt daß der Substring zwischen j und l als $V_m ... V_n$ klassifiziert werden kann.

Notation: Die konditionale Natur dieser Interpretation und ihre Unabhängigkeit von $V_1, ..., V_{m-1}$ drückt Shieber durch die folgende Notation aus:

$$[i, N \longleftarrow V_m ... V_n, j]$$

Die *nicht-konditionalen* Formeln $[i, N \longrightarrow V_1 ... V_{m-1} \bullet, j]$ schreibt Shieber als:

$$[i, N, j]$$

(Ihre Interpretation lautet: Der Substring zwischen i und j kann als N klassifiziert werden.) □

Bevor wir Shiebers Inferenzregeln vorstellen, machen wir uns klar, daß Shieber Grammatikformalismen wie DCG, PATR, HPSG, u.s.w. im Sinn hat. Grammatiksymbole stehen hier für *strukturierte linguistische Objekte*. Für unsere Zwecke genügt es, daß wir ein Grammatiksymbol als Träger einer kategorialen und semantischen Information ansehen. In den folgenden Inferenzregeln sind dann B, B' von derselben Kategorie. Ist $\Theta = mgu(B, B')$ und X ein Grammatiksymbol, so steht $X\Theta$ für dasjenige linguistische Objekt mit:

- kategorialer Information von X
- (vereinigter) semantischer Information von B, B' und X

(Eine präzise Definition ist natürlich vom zugrundegelegten Grammatikformalismus abhängig.)

Als Inferenzregeln benutzt Shieber *Prediction* und *Completion*:

$$\textbf{(Prediction)} \quad \frac{[i, A \longleftarrow BC_1 ... C_m, j]}{[j, B'\Theta \longleftarrow D_1\Theta ... D_n\Theta, j]} \; (B' \longrightarrow D_1 ... D_n, \;\; \Theta = mgu(B, B$$

$$\textbf{(Completion)} \quad \frac{[i, A \longleftarrow BC_1 ... C_m, j] \;\; [j, B', k]}{[i, A\Theta \longleftarrow C_1\Theta ... C_m\Theta, k]} \; (\Theta = mgu(B, B'))$$

Shiebers *Parametrisierung* des Deduktionssystems besteht im wesentlichen darin, offenzulassen, welche Formelteilmenge die Axiome bilden sollen. Ausserdem läßt er offen, wann der Deduktionsprozess als erfolgreich beendet gelten soll. Hierfür sind gewisse *Erfolgskriterien* zuständig. Eine weitere Feinheit von Shiebers Deduktionsprozess besteht in der Einführung einer sogenannten *Prioritätsfunktion*. Hierbei handelt es sich um eine (totale oder partiale) Funktion auf der Menge aller Aussagen. Ein Deduktionsprozess ist in der Regel ein nicht-deterministischer Prozess. Die Prioritätsfunktion hat die Aufgabe, diesen Nicht-Determinismus zu beseitigen. Sie bestimmt, welche Aussagen als nächste zur Weiterverarbeitung (durch die Inferenzregeln) anstehen, bzw. welche Aussagen garnicht mehr zur Weiterverarbeitung herangezogen werden sollen.
Die Prioritätsfunktion stellt also ein Steuerungsinstrument dar. Shieber benutzt es, um für Parsing gewisse psycholinguistische Phänomene nachzubilden. Ungleich wichtiger ist die Prioritätsfunktion aber für Generierung. Hier wird sie benutzt, um ein *semantisches Filter* in den Deduktionsprozess einzuführen. Wir kommen später darauf zurück und halten jetzt fest:

Eine **Instantiierung** von Shieber's Deduktionsprozess besteht in der Vorgabe:

- der Axiome
- des Erfogskriteriums
- der Prioritätsfunktion

Wir werden nun demonstrieren, daß wir durch Variation der Parameter, sowohl parsen als auch generieren können:

Parsing-Instanz:
Es sei der folgende (ambige) Satz:

Castillo said Sonny was shot yesterday

zu parsen. Wir parametrisieren:

Axiome:	$[0, castillo, 1]$	
	$[1, said, 2]$	
	$[2, sonny, 3]$	
	$[3, was, 4]$	
	$[4, shot, 5]$	
	$[5, yesterday, 6]$	
	$[0, S \longleftarrow V_1 ... V_n, 0]$	$\forall S \longrightarrow V_1 ... V_n$
Erfolgskriterium:	$[0, S, 6]$ ist eine Aussage	
Prioritätsfunktion:	ohne	

Der Parsevorgang könnte dann wie folgt aussehen:

Das Item	$[0, S \longrightarrow \bullet NP\ VP, 0]$	"
könnte ein Axiom sein.		
Prediction auf das Axiom	$[0, castillo, 1]$	'Castillo'
könnte die Aussage	$[1, NP \longrightarrow \bullet castillo, 1]$	"
liefern.		
Completion liefert dann:	$[0, NP \longrightarrow castillo\bullet, 1]$	'Castillo'
Erneute Completion liefert:	$[0, S \longrightarrow NP \bullet VP, 1]$	'Castillo'
Prediction könnte dann	$[1, VP \longrightarrow \bullet VP\ AdvP, 1]$	"
liefern.		
Irgendwann könnte das Item	$[1, VP \longrightarrow VP\ AdvP\bullet, 6]$	'said Sonny was shot yesterday'
entstehen.		
Completion liefert dann:	$[1, S \longrightarrow NP\ VP\bullet, 6]$	'Castillo said Sonny was shot yesterday'

Diese Aussage erfüllt aber das Parsing-Erfolgskriterium. D.h. der Deduktionsprozess bricht erfolgreich ab □

Wir wenden uns nun der Generierungs-Instanz zu und stellen drei Ansätze vor. Jeder Ansatz wird dem vorhergehenden Ansatz etwas überlegen sein. Wir beginnem mit:

Generierungs-Instanz (1. Ansatz): Wenn wir beispielsweise einen Satz generieren wollen, der die logische Form

passionately(love(sonny, kait))

trägt, so können wir den Deduktionsprozess wie folgt parametrisieren:

Axiome:	$[i, w, i+1]$	$\forall w \in {V_T}^*$
		$\forall i \in \{0, 1, 2, 3, ...\}$
	$[0, S \longleftarrow V_1 ... V_n, 0]$	$\forall S \longrightarrow V_1 ... V_n$
Erfolgskriterium:	$\exists n \in N$: $[0, S, n]$ ist eine Aussage und S trägt die logische Form: *passionately(love(sonny, kait))*	
Prioritätsfunktion:	ohne	

Wenn es einen grammatischen Satz gibt, der die gewünschte logische Form trägt, so können wir ihn offensichtlich in der vorgestellten Generierungsinstanz ableiten. Aus zwei Gründen ist der vorgestellte Ansatz jedoch unbefriedigend:

- Das gesamte Grammatiklexikon (d.h jedes $w \in {V_T}^*$) muß für jede Stringposition ($i = 0, 1, 2, 3, ...$) in die Axiomenmenge übernommen werden. Eine Implementation des Deduktionssystems (etwa als Chart-basierter Theorembeweiser) ist somit undurchführbar.
- Beim Generierungsvorgang treten massive Redundanzen auf, da jede Aussage $[0, N, n]$ auch als *geshiftete* Aussage $[i, N, j]$ (mit $j = i + n$) abgeleitet werden kann. Wenn wir uns klarmachen, daß z.B. der Earley-Parser gerade deswegen so effizient arbeitet, weil er zu jedem Teilstring exakt verwaltet, wie dieser Teilstring zu klassifizieren ist, so wird deutlich, daß der vorgestellte Ansatz nicht effizient sein kann.

Generierungs-Instanz (2. Ansatz): Eine einfache Lösung des ersten Problems besteht darin, die Stringpositionen zu ignorieren. Wir können dies sehr einfach

tun, indem wir (für eine vorgegebene logische Form Φ) wie folgt parametrisieren:

Axiome:	$[0, w, 0]$	$\forall w \in {V_T}^*$
	$[0, S \longleftarrow V_1 ... V_n, 0]$	$\forall S \longrightarrow V_1 ... V_n$
Erfolgskriterium:	$[0, S, 0]$ ist eine Aussage und S trägt die logische Form Φ	
Prioritätsfunktion:	ohne	

Dies führ aber leider zu keiner Lösung des zweiten Problems, da die Stringpositionen (nun stets 0) wieder nicht zu einer effizienten Klassifizierung herangezogen werden können. Als Ausweg aus diesem Dilemma schlägt Shieber vor, daß statt der Stringpositionen nun die sematische Information dazu dienen soll, die Items zu indizieren. Hierzu machen wir uns klar, daß ein Item

$$[i, ..., j]$$

unter anderem encodierte, daß zwischen den Stringpositionen i und j ein *Substring* des vorgegebenen Strings vorlag. Analog hierzu fordert Shieber für Generierung:

> Die einem Item assoziierte Bedeutung muß einen Teil der vorgegebenen Bedeutung subsummieren.

(Unter der einem Item $[i, N \longrightarrow ..., j]$ assoziierten Bedeutung könnten wir die semantische Information des strukturierten linguistischen Objekts N verstehen.) In Shiebers Deduktionsprozess kann die Einhaltung dieser Bedingung durch eine geschickte Wahl der Prioritätsfunktion erreicht werden:

> Sei $I(\Phi)$ die Menge aller Items, deren Bedeutung einen Teil von Φ subsummiert. Wird die Prioritätsfunktion so gewählt, daß nur Items aus $I(\Phi)$ zur Weiterverarbeitung (durch die Inferenzregeln) aufgegriffen werden, so wirkt die Prioritätsfunktion gewissermaßen als *semantisches Filter*: dieses Filter können nur Items aus $I(\Phi)$ passieren.

Dies führt zu:

Generierungs-Instanz (3. Ansatz): Sei Φ eine vorgegebene logische Form.

Wir parametrisieren dann:

Axiome:	$[0, w, 0] \in I(\Phi)$	$(w \in {V_T}^*)$
	$[0, S \longleftarrow V_1 ... V_n, 0] \in I(\Phi)$	$(S \longrightarrow V_1 ... V_n)$
Erfolgskriterium:	$[0, S, 0]$ ist eine Aussage, $[0, S, 0] \in I(\Phi)$ Φ subsummiert die Bedeutung von S	
Prioritätsfunktion:	nur Items aus $I(\Phi)$ werden weiterverarbeitet	

Shieber bemerkt, daß auch diese Generierungsinstanz einen schwerwiegenden Nachteil hat: sie ist nicht *vollständig*. Der Grund ist in der Einschränkung zu suchen, daß zur Deduktion nur *die* Items zugelassen sind, deren Bedeutung einen Teil von Φ subsummieren. Es ist nun aber durchaus vorstellbar, daß es Phrasen gibt, die eine Teilphrase besitzen, sodaß die Bedeutung dieser Teilphrase *nicht* die Bedeutung der ganzen Phrase subsummiert. Somit könnte es also der Fall sein, daß die Phrase zwar von der Grammatik (als Satz) akzeptiert wird, jedoch nicht durch Shiebers Deduktionsprozess generiert werden kann (da eine Teilphrase nicht das semantische Filter passiert.) Shieber ist daher gezwungen, die Anwendung der Generierungsinstanz auf *semantisch monotone Grammatiken* zu beschränken:

> Eine Grammatik ist *semantisch monoton*, wenn für jede (grammatische) Phrase gilt: die semantische Struktur einer direkten Subphrase subsummiert einen Teil der semantischen Struktur der ganzen Phrase.

Beispiel: Wir wollen uns nun davon überzeugen, daß seine Generierungsinstanz sehr effektiv arbeitet. Wir betrachten die Generierung eines Satzes mit der vorgegebenen logischen Form:

> *passionately(love(sonny, kait))*

Die Grammatik sei so geartet, daß die Semantik von Hilfsverben mit der Semantik der zugehörigen Vollverben identifiziert wird. Ferner ignoriere sie Zeit und Aspekt.

Die folgenden Items könnten Axiome sein:

$[0, sonny, 0]$	'Sonny '
$[0, kait, 0]$	'Kait '
$[0, to, 0]$	'to '
$[0, was, 0]$	'was '
$[0, were, 0]$	'were'
...	
$[0, loves, 0]$	'loves '
$[0, love, 0]$	'love '
$[0, loved, 0]$	'loved '
$[0, passionately, 0]$	'passionately '
$[0, S \longrightarrow \bullet NP\ VP, 0]$	''

Die Hilfsverben wurden aufgenommen, weil ihre semantische Struktur (im weiteren Verlauf des Deduktionsprozesses) mit der semantischen Struktur ihrer Vollverben identifiziert wird. Die Nomen 'Sonny' und 'Kait' wurden aufgenommen, da ihre semantischen Strukturen (*sonny* bzw. *kait*) einen Teil der vorgegebenen logischen Form subsummieren. Einige Formen des Verbs 'love' wurden aus demselben Grund hinzugefügt. (Hierbei ist zu beachten, daß die Grammatik keine Zeit/Aspekt-Unterschiede vornimmt.) Der wichtige Punkt ist jedoch, daß *keine weiteren* Verben oder Nomen des Grammatiklexikons Axiome sein können, da sie *nicht* das semantische Filter passieren ! Durch Prediction und Completion erhält Shieber die folgende Item-Menge:

(1)	$[0, NP \longrightarrow sonny\bullet, 0]$	'Sonny'
(2)	$[0, NP \longrightarrow kait\bullet, 0]$	'Kait'
	$[0, V \longrightarrow to\bullet, 0]$	'to'
	$[0, V \longrightarrow was\bullet, 0]$	'was'
	$[0, V \longrightarrow were\bullet, 0]$	'were'
	...	
	$[0, V \longrightarrow loves\bullet, 0]$	'loves'
	$[0, V \longrightarrow love\bullet, 0]$	'love'
	$[0, V \longrightarrow loved\bullet, 0]$	'loved'
	$[0, AdvP \longrightarrow passionately\bullet, 0]$	'passionately'
(3)	$[0, S \longrightarrow \bullet NP\ VP, 0]$	''

Die NP 'Sonny' kann als Satzsubjekt benutzt werden, indem wir die Items (1) und (3) kombinieren:

(4) $[0, S \longrightarrow NP \bullet VP, 0]$ 'Sonny'

(Das korrespondierende Item mit dem Subjekt 'Kait' wird später generiert.) Prediction liefert dann:

$[0, VP \longrightarrow \bullet VP\ AdvP, 0]$ ''
$[0, VP \longrightarrow \bullet V, 0]$ ''

Die verschiedenen Verben können mit dem letzen Item durch Completion kombiniert werden:

	$[0, VP \longrightarrow V\bullet, 0]$	'to'
	$[0, VP \longrightarrow V\bullet, 0]$	'is'
	$[0, VP \longrightarrow V\bullet, 0]$	'was'
	$[0, VP \longrightarrow V\bullet, 0]$	'were'
(5)	$[0, VP \longrightarrow V\bullet, 0]$	'loves'
	$[0, VP \longrightarrow V\bullet, 0]$	'love'
	$[0, VP \longrightarrow V\bullet, 0]$	'loved'

Die Passivform des Verbs 'loved' könnte mit dem Adverb kombiniert werden:

$[0, VP \longrightarrow VP \bullet AdvP, 0]$	'loved'
$[0, VP \longrightarrow VP\ AdvP\bullet, 0]$	'loved passionately'
...	

Das letzte Item könnte in einem Satz 'Kait was loved passionately' benutzt werden. Dieser Satz würde eventuell generiert werden, jedoch nicht das Erfolgskriterium bestehen, da seine Semantik unzureichend ist.
Prediction auf Item (4) liefert (mit einer Grammatikregel, welche Komplemente zu Verbphrasen addiert) das Item:

$[0, VP \longrightarrow \bullet VP\ X, 0]$ ''

Completion mit Items (5) und (2) liefert:

...	
$[0, VP \longrightarrow VP \bullet NP, 0]$	'loves'
$[0, VP \longrightarrow VP\ NP\bullet, 0]$	'loves Kait'

Damit könnten dann die folgenden Items generiert werden:

$[0, VP \longrightarrow VP \bullet AdvP, 0]$	'loves Kait'
$[0, VP \longrightarrow VP\ AdvP\bullet, 0]$	'loves Kait passionately'
$[0, S \longrightarrow NP\ VP\bullet, 0]$	'Sonny loves Kait passionately'

Das letzte Item erfüllt das Erfolgskriterium und es ist das einzige solche Item. Daher wird für die logische Form $passionately(loves(sonny, kait))$ der Satz 'Sonny loves Kait passionately' generiert.
Ein Rückblick auf den gesamten Generierungsprozess zeigt, daß vom Generator die Sätze 'Kait is loved', 'Kait is loved passionately' und ähnliche Passivkonstruktionen, ausserdem 'Sonny loves Kait' und diverse Subphrasen erzeugt wurden. Keiner dieser Sätze bestand das Erfolgskriterium. Sehr erfreulich ist hingegen, daß Sätze wie 'Kait loves Kait', 'Sonny is loved' u.s.w. frühzeitig vom semantischen Filter eliminiert wurden □

4.3 Entscheidbarkeit des Generierungsproblems

In diesem Abschnitt werden wir uns mit den Ergebnissen von Wedekind [9] beschäftigen. In Abschnitt 3.2 haben wir festgelegt, was wir unter der (oder den) Featurestruktur(en) eines grammatischen Satzes verstehen wollen. Wir rufen uns in Erinnerung:

> Ein String $w \in {V_T}^*$ ist mit Feature-Struktur F ableitbar
>
> **gdw.** (i) $S \Longrightarrow^* w, C$
> (ii) C ist erfüllbar
> (iii) $F = f\text{–}structure(F^C)$
>
> Hierbei ist F^C ein Minimalmodell für C.

Ferner hatten wir angegeben:

> Das Generierungsproblem besteht darin, für eine (vorgegebene unifikationsbasierte Grammatik und) eine vorgegebene Featurestruktur F zu entscheiden, ob ein String $w \in {V_T}^*$ existiert, sodaß w mit Featurestruktur F ableitbar ist.

Bevor wir uns Wedekinds Ergebnissen zuwenden, wollen wir uns ein Beispiel anschauen. In Abschnitt 4.1 haben wir uns mit der Unentscheidbarkeit des Pfad-Parsing-Problems beschäftigt. Für den Beweis der Unentscheidbarkeit spielten die PCP-Grammatiken eine wesentliche Rolle. Für die PCP-Grammatik:

$$
\begin{array}{lll}
I \longrightarrow 1 & \{x_{n_0}A \doteq \text{„}1\text{“}, & x_{n_0}B \doteq \text{„}111\text{“}\} \\
I \longrightarrow 2 & \{x_{n_0}A \doteq \text{„}10111\text{“}, & x_{n_0}B \doteq \text{„}10\text{“}\} \\
I \longrightarrow 3 & \{x_{n_0}A \doteq \text{„}10\text{“}, & x_{n_0}B \doteq \text{„}0\text{“}\} \\
I \longrightarrow II & \{x_{n_0}A \doteq x_{n_1}A + x_{n_2}A, & x_{n_0}B \doteq x_{n_1}B + x_{n_2}B\} \\
S \longrightarrow I & \{x_{n_0}SOLUTION \doteq YES, & x_{n_1}A \doteq x_{n_1}B\}
\end{array}
$$

hatten wir gefunden, daß

2113

ein grammatischer Satz ist: wir hatten einen Syntaxbaum angeben können, dessen Feature-Beschreibung C durch das folgende Minimalmodell F^C erfüllt wurde:

$$x_1 \left[\text{SOLUTION} \quad \text{YES} \right]$$

$$x_2 \left[\begin{array}{ll} \text{A} & \text{„}101111110\text{“} \\ \text{B} & \text{„}101111110\text{“} \end{array} \right]$$

$$x_3 \left[\begin{array}{ll} \text{A} & \text{„}10111\text{“} \\ \text{B} & \text{„}10\text{“} \end{array} \right]$$

$$x_4 \left[\begin{array}{ll} \text{A} & \text{„}1110\text{“} \\ \text{B} & \text{„}1111110\text{“} \end{array} \right]$$

$$x_5 \left[\begin{array}{ll} \text{A} & \text{„}1\text{“} \\ \text{B} & \text{„}111\text{“} \end{array} \right]$$

$$x_6 \left[\begin{array}{ll} \text{A} & \text{„}110\text{“} \\ \text{B} & \text{„}1110\text{“} \end{array} \right]$$

$$x_7 \left[\begin{array}{ll} \text{A} & \text{„}1\text{“} \\ \text{B} & \text{„}111\text{“} \end{array} \right]$$

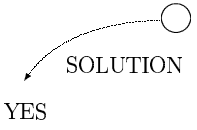
$$x_8 \left[\begin{array}{ll} \text{A} & \text{„}10\text{“} \\ \text{B} & \text{„}0\text{“} \end{array} \right]$$

Die Indizees der Variablen $x_1, ..., x_8$ korrespondierten hierbei mit den Knoten des Syntaxbaumes *Syn*. (Der Knoten 1 war die Wurzel von *Syn*.) Wir bemerken also, daß eine Umbenennung der Knoten $\{1, ..., 8\}$ des Syntaxbaumes , etwa durch eine Bijektion $\Phi: \ \{1, ..., 8\} \longrightarrow \{1, ..., 8\}$ zwar ein anderes Minimalmodell nach sich zieht, nämlich

$$F^{C[x_1|x_{\Phi(1)}, ..., x_8|x_{\Phi(8)}]}$$

aber die Featurestruktur erhalten bleibt (da diese unabhängig von den Variablenlabeln ihres Featuregraphen ist). Diese mehr oder weniger triviale Überlegung mag als nachträgliche Motivation dienen, den grammatischen Sätzen Featurestrukturen und nicht Featuregraphen zuzuweisen. Wir wenden uns nun dem Generierungsproblem zu.

VanNoord hat in seiner Dissertation [5] gezeigt, daß für PCP-Grammatiken das *SOLUTION*-Parsing-Problem unentscheidbar ist. Das *SOLUTION*-Parsing-Problem besteht grob gesagt darin, für PCP-Grammatiken zu entscheiden, ob ein $w \in {V_T}^*$ existiert, sodaß die Featurestruktur von $w \in {V_T}^*$ den folgenden Subgraphen enthält:



In diesem Subgraphen ist nun keinerlei Information enthalten, außer der Forderung, daß die einer jeden PCP-Grammatik zugeordnete PCP-Instanz lösbar sein soll.

Anders im Generierungsproblem (für PCP-Grammatiken): hier muß die komplette Featurestruktur eines Strings $w \in {V_T}^*$ vorgegeben werden. In unserem Beispiel oben ist in der Featurestruktur aber die komplette Zerlegung von „10111111110“ in A: „10111“ + „1“ + „1“ + „10“ und B: „10“ + „111“ + „111“ + „0“ codiert. Trotz des VanNoordschen Ergebnisses erscheint es also durchaus naheliegend, daß das Generierungsproblem entscheidbar sein könnte, da hier komplette Featurestrukturen - und nicht nur Sub-Featurestrukturen - vorgegeben werden müssen.

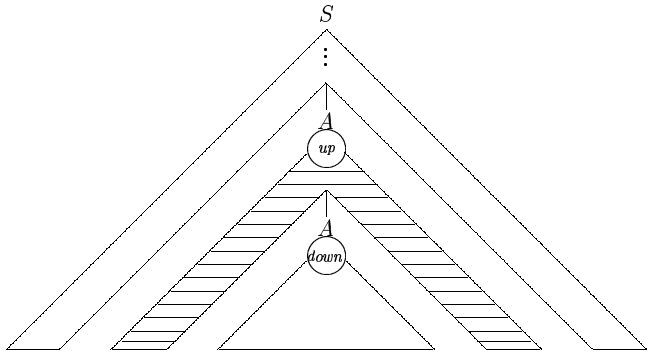
Wir wenden uns nun Wedekinds Arbeit [9] zu, in der er erstens beweist, daß das Generierungsproblkem entscheidbar ist und zweitens, daß ein Generator im allgemeinen nicht existieren wird. Da Wedekinds Beweise relativ komplex sind, werden wir sie nicht detailliert nachvollziehen. Für die Vorstellung der Ergebnisse werden wir den in den vorangehenden Kapiteln eingeführten Formalismus verwenden.

Wedekinds grundlegende Idee ist, daß es in einer unifikationsbasierten Grammatik nur endlich viele Sätze geben kann, die eine vorgegebene Featurestruktur tragen - vorausgesetzt, daß wir nur Syntaxbäume ohne *redundante Rekursionen* zulassen. Wir definieren dazu:

Definition: Sei $Syn =< N, daughters, cat, constraint >$ ein Syntaxbaum. Eine *Rekursion* $< up, down >$ in Syn besteht aus zwei Knoten $up, down \in N$ mit:

(i) up dominiert $down$
(ii) $cat(up) = A = cat(down)$ für ein $A \in V_N$ $\square$

Die Situation wird durch das folgende Bild verdeutlicht:



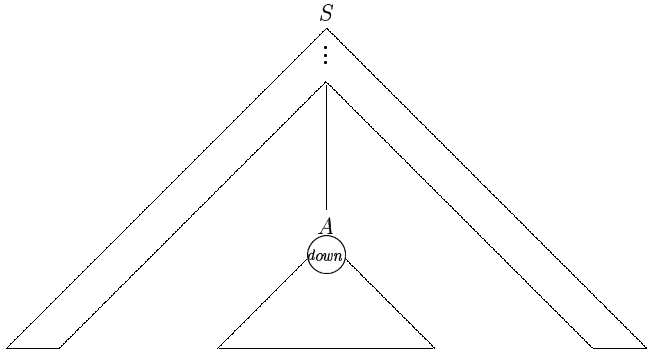
Die von der Rekursion $< up, down >$ betroffenen Knoten des Syntaxbaumes sind in der Abbildung schraffiert eingezeichnet. Definieren wir (für einen Syntaxbaum Syn und einen Knoten $n \in N$):

$$N(n) =_{def} \{n\} \cup \{m \in N \mid n \text{ dominiert } m\}$$

so handelt es sich bei den von der Rekursion betroffenen Knoten offenbar um die folgende Menge:

$$Rec(up, down) =_{def} N(up) \setminus N(down)$$

Um den Begriff der *redundanten Rekursion* einzuführen, ist es zweckmässig, zunächst den Syntaxbaum $Syn[up|down]$ zu beschreiben, den man erhält, wenn man aus einem Syntaxbaum Syn eine in ihm vorkommende Rekursion $< up, down >$ entfernt. Wir können uns $Syn[up|down]$ wie folgt veranschaulichen:



Formal definieren wir:

Definition: Sei $Syn =< N, daughters, cat, constraint >$ ein Syntaxbaum und $< up, down >$ eine Rekursion in Syn. Wir setzen:

$$Syn[up|down] =< N', daughters', cat', constraint' >$$

wobei:

$$N' = N \setminus Rec(up, down)$$

und:

$$daughters' : \quad N' \longrightarrow N'^* \qquad n \mapsto \begin{cases} p\ down\ q & \text{falls} \quad daughters(n) = p\ up\ q \\ & \qquad (p, q \in N^*) \\ daughters(n) & \text{sonst} \end{cases}$$

$$cat' : \quad N' \longrightarrow (V_N \cup V_T)^* \qquad n \mapsto cat(n)$$

$$constraint' : \quad N' \longrightarrow \mathcal{L} \qquad n \mapsto \begin{cases} constraint(n)[x_{up}|x_{down}] & \text{falls} \quad daughters(n) = p\ up\ q \\ & \qquad (p, q \in N^*) \\ constraint(n) & \text{sonst} \end{cases}$$

□

In dieser Definition kommt die Bedingung:

$$daughters(n) = p\ up\ q$$

zweimal vor ($n, up \in N$ und $p, q \in N^*$). Sie bedeutet nichts weiter, als daß $n \in N$ der Mutterknoten von $up \in N$ ist. Für festes $up \in N$ kann diese Bedingung also für höchstens einen Knoten $n \in N$ zutreffen. (Ist $up \in N$ der Wurzelknoten, so trifft sie für keinen Knoten $n \in N$ zu.) Die Funktionen

$$daughters', cat' \text{ und } constraint'$$

sind im wesentlichen die Einschränkungen von

$$daughters, cat \text{ und } constraint$$

auf die neue Knotenmenge $N \setminus Rec(up, down)$. Während dies für cat' wegen der Rekursionsbedingung $cat(up) = cat(down)$ uneingeschränkt gelten darf, müssen wir für den höchstens einen Knoten $n \in N$ mit

$$daughters(n) = p\ up\ q$$

aufpassen, daß $daughters'$ und $constraint'$ diesem Knoten die richtigen Dinge zuweisen:

- anstelle von up muß $down$ Tochterknoten sein
- als Folge hiervon muß sein Constraint anstelle von x_{up} mit x_{down} instantiiert sein

Wir können leicht nachprüfen, daß $Syn[up|down]$ wieder ein Syntaxbaum ist. Wir sind nun in der Lage, *redundante Rekursionen* zu definieren.

Definition: Sei Syn ein Syntaxbaum und $< up, down >$ eine Rekursion in Syn. Dann sagen wir:

$$< up, down > \text{ ist eine } \textit{redundante Rekursion}$$
$$\textbf{gdw.} \quad f\text{–}structure(Syn) = f\text{–}structure(Syn[up|down]) \qquad \square$$

In Kapitel 5 werden wir diese Beziehung eingehender untersuchen. An dieser Stelle reicht es, ein Beispiel zu betrachten.

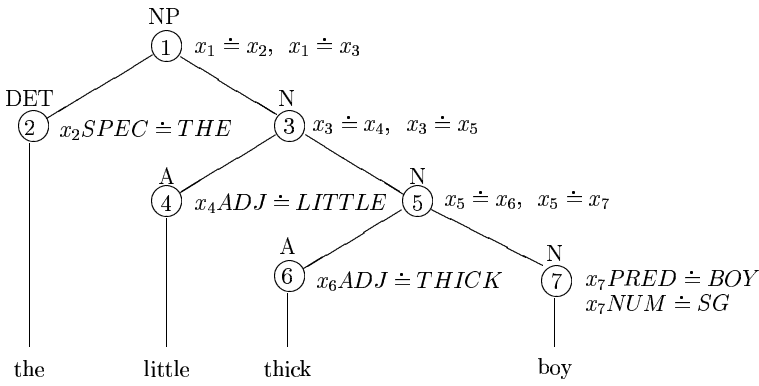
Beispiel: Für die folgende Grammatik (mit Startsymbol NP):

NP	$\longrightarrow$	DET N	$\{x_{n_0} \doteq x_{n_1},\ x_{n_0} \doteq x_{n_2}\}$
N	$\longrightarrow$	A N	$\{x_{n_0} \doteq x_{n_1},\ x_{n_0} \doteq x_{n_2}\}$
DET	$\longrightarrow$	the	$\{x_{n_0} SPEC \doteq THE\}$
A	$\longrightarrow$	little	$\{x_{n_0} ADJ \doteq LITTLE\}$
A	$\longrightarrow$	thick	$\{x_{n_0} ADJ \doteq THICK\}$
N	$\longrightarrow$	boy	$\{x_{n_0} PRED \doteq BOY,\ x_{n_0} NUM \doteq SG\}$

gilt:

$$NP \Longrightarrow^* \text{ the little thick boy}$$

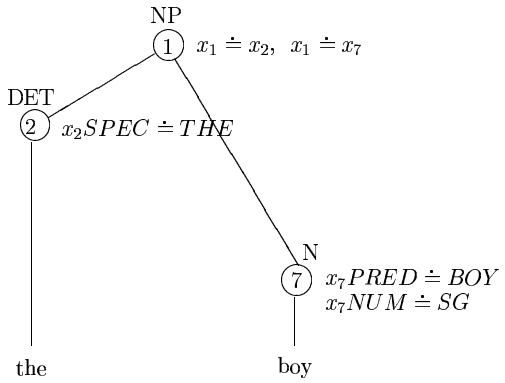
Der Satz besitzt den folgenden Syntaxbaum Syn:



In Syn kommen die Rekursionen:

$$< 3, 7 >$$
$$< 3, 5 >$$
$$< 5, 7 >$$

vor. Keine dieser Rekursionen ist redundant. Wir überprüfen nur die Rekursion $< 3, 7 >$. Der folgende Syntaxbaum stellt $Syn[3|7]$ dar:



Bei der Herstellung dieses Syntaxbaumes ist lediglich darauf zu achten, daß der Constraint des Knotens 1 korrekt instantiiert wird. (Knoten 1 ist in Syn der Mutterknoten von Knoten 3. Die richtige Instantiierung wird daher durch die obligatorische Ersetzung von x_3 durch x_7 erreicht.)

Die Featurestruktur von Syn ist:

$$\begin{bmatrix} \text{SPEC} & \text{THE} \\ \text{ADJ} & \{\text{LITTLE, THICK}\} \\ \text{PRED} & \text{BOY} \\ \text{NUM} & \text{SG} \end{bmatrix}$$

Die Featurestruktur von $Syn[3|7]$ ist:

$$\begin{bmatrix} \text{SPEC} & \text{THE} \\ \text{PRED} & \text{BOY} \\ \text{NUM} & \text{SG} \end{bmatrix}$$

Somit kann $< 3, 7 >$ keine redundante Rekursion in Syn sein $\square$

Bemerkung: Für „vernünftige“ Grammatiken ist es sehr schwer, redundante Rekursionen zu finden. Ein triviales Beispiel ist:

$$\text{X} \longrightarrow \text{X} \qquad \{x_{n_0} \doteq x_{n_1}\}$$

Wird eine solche Produktion angewandt, findet sich im Syntaxbaum die redundante Rekursion $< n_0, n_1 >$ ...

Bevor wir Wedekinds Kernaussage formulieren können, müssen wir noch definieren, was wir unter dem *Informationsgehalt* einer Featurestruktur verstehen.

Definition: Sei F eine Featurestruktur und C ein Constraint mit:

(i) C ist in Normalform
(ii) C enthält keine negativen Literale
(iii) $F = f\text{–}structure(C)$

Dann setzen wir:

$$Info(F) = card(C)$$

und sagen: F hat *Informationsgehalt Info(F)* oder F *enthält Info(F) Informationen* $\square$

Beispiel: Die Featurestruktur:

$$\begin{bmatrix} \text{PRED} & \text{LOVE} \\ \text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{GIRL} \\ \text{SPEC} & \text{THE} \\ \text{NUM} & \text{SG} \end{bmatrix} \\ \text{OBJ} & \begin{bmatrix} \text{PRED} & \text{BOY} \\ \text{SPEC} & \text{A} \\ \text{NUM} & \text{SG} \end{bmatrix} \\ \text{TENSE} & \text{PAST} \end{bmatrix}$$

enthält acht Informationen, denn der Constraint:

$$\begin{aligned} xPRED &\doteq LOVE \\ xSUBJ\ PRED &\doteq GIRL \\ xSUBJ\ SPEC &\doteq THE \\ xSUBJ\ NUM &\doteq SG \\ xOBJ\ PRED &\doteq BOY \\ xOBJ\ SPEC &\doteq A \\ xOBJ\ NUM &\doteq SG \\ xTENSE &\doteq PAST \end{aligned}$$

ist in Normalform, enthält keine negativen Literale, hat die gewünschte Featurestruktur und enthält acht Literale □

Wedekind hat in seiner Arbeit bewiesen, daß der Informationsgehalt einer Featurestruktur wohldefiniert (d.h. unabhängig von der speziellen Wahl des Constraints C) ist. Wir wollen diesen Beweis hier nicht vorstellen, sondern vielmehr Wedekinds entscheidendes Ergebnis präsentieren.

Lemma: Sei $w \in {V_T}^*$ mit Featurestruktur F ableitbar. Ist Syn ein zugehöriger [3] Syntaxbaum ohne redundante Rekursionen, so gilt:

$$|Syn| \leq |V_N| \cdot (1 + 3 \cdot Nodes(F)) \cdot (1 + Info(F))$$

Hierbei bezeichnet $|Syn|$ die Länge des längsten Pfades in Syn und $Nodes(F)$ die Anzahl der Knoten von F □

Aus diesem Lemma können wir die wichtige Folgerung ziehen:

Satz: Das Generierungsproblem ist entscheidbar □

Beweis: Für eine Featurestruktur F definieren wir:

$$\begin{array}{lll} \Sigma_F = \{\ Syn| & (i) & Syn \text{ ist ein Syntaxbaum} \\ & (ii) & c\text{–}string(Syn) \in {V_T}^* \\ & (iii) & F = f\text{–}structure(Syn) \\ & (iv) & |Syn| \leq |V_N| \cdot (1 + 3 \cdot Nodes(F)) \cdot (1 + Info(F)) \quad \} \end{array}$$

Die Ungleichung (iv) garantiert, daß Σ_F eine endliche Menge ist (wobei wir isomorphe Syntaxbäume als gleich ansehen). Wir zeigen nun:

$$\Delta_F \neq \emptyset \quad \textbf{gdw.} \quad \Sigma_F \neq \emptyset$$

„$\Leftarrow$": Ist $Syn \in \Sigma_F$, so ist $c\text{–}string(Syn) \in \Delta_F$.
„$\Rightarrow$": Sei $w \in \Delta_F$. Sei Syn' ein Syntaxbaum mit $w = c\text{–}string(Syn')$ und $F = f\text{–}structure(Syn')$. Entsteht Syn aus Syn' durch sukzessive Entfernung der redundanten Rekursionen, so können wir induktiv leicht zeigen, daß $F = f\text{–}structure(Syn)$. Ausserdem gilt wegen Wedekinds Lemma die Ungleichung (iv). Also ist $Syn \in \Sigma_F$.
Das Generierungsproblem ist also äquivalent dazu, für die *endliche* Menge Σ_F zu entscheiden, ob $\Sigma_F \neq \emptyset$. Das Generierungsproblem ist also entscheidbar □

Aus dem Beweis des Lemmas zieht Wedekind die Folgerung:

[3] d.h. $w = c\text{–}string(Syn)$ und $F = f\text{–}structure(Syn)$

Pumping-Lemma: Sei F eine Featurestruktur und

$$\lambda = max\{k \in I\!N | X \longrightarrow X_1 ... X_k \in Prod\}$$

Dann gilt:

$$\begin{array}{l} \quad w \in \Delta_F \text{ und} \\ \quad |w| > \lambda^{|V_N| \cdot (1+2 \cdot Nodes(F))} \\ \Longrightarrow \\ \quad w = uvzxy \text{ mit } |vx| > 0 \text{ und} \\ \quad uv^i zx^i y \in \Delta_F \end{array}$$ □

Dieses Pumping-Lemma zeigt, daß Δ_F im allgemeinen keine endliche Menge ist. Somit kann es nicht gelingen, einen Generator anzugeben, der alle $w \in \Delta_F$ aufzählt.

Bemerkung: Es ist interessant, daß im Pumping-Lemma nicht der Informationsgehalt der Featurestruktur auftaucht □

Bemerkung: Gemäß Lemma sind alle Syntaxbäume, die keine redundante Rekursion aufweisen und eine vorgegebene Featurestruktur F erzeugen, in der Menge Σ_F zu finden. Um sie aufzufinden, sind also nur die Syntaxbäume zu betrachten, deren längster Pfad eine nur von F und der Grammatik abhängige Länge nicht überschreitet. Für die Praxis hat dies zwei Auswirkungen: zum einen sollte diese Grenze so klein wie theoretisch möglich gedrückt werden (Effizienz); zum anderen sollte bei der Konstruktion von Syntaxbäumen (Z.B. Earley-Algorithmus, etc.) Buch über deren Größe geführt werden □

4.4 Unentscheidbarkeit ambiguitätserhaltender Übersetzung

In [10] zeigen Wedekind und Kaplan (in einem relativ einfachen Beweis), daß es unentscheidbar ist, ob zu zwei oder mehr vorgegebenen Featurestrukturen ein Satz existiert, der diese Featurestrukturen trägt.
In diesem Abschnitt werden wir einen Ansatz zur ambiguitätserhaltenden Übersetzung kennenlernen und zeigen, daß es in diesem Ansatz (aus dem obengenannten Grund) nicht entscheidbar ist, ob ein Satz ambiguitätserhaltend zu übersetzen ist.

Das Phänomen der Ambiguität stellt für die maschinelle Übersetzung natürlicher Sprache eine bedeutende Herausforderung dar. Die Übersetzung eines ambigen Quellsatzes hängt erstens davon ab, welche Lesart des Satzes im Kontext angemessen ist und zweitens, ob ein Zielsatz existiert, der exakt diese Lesart ausdrückt. Dies kann schwierig oder unmöglich sein, wenn die Auflösung der Quellambiguität vom kompletten Verständnis des Textes abhängt, oder wenn mehrere Lesarten kontextuell angemessen sind. Eine attraktive Strategie, die Disambiguierung zu vermeiden, ist es, einen Zielsatz zu produzieren, der exakt dieselben Ambiguitäten, wie der Quellsatz besitzt. Wir betrachten als Beispiel den wohlbekannten ambigen Satz

(1) **John saw the man with the telescope**

welcher in einer unifikationsbasierten Grammatik die folgenden Featurestrukturen tragen könnte:

(2a) *with_the_telescope(see(john, man))*
(2b) *see(john, with_the_telescope(man))*

Das Problem, diesen Satz z.B. ins Deutsche zu übertragen, könnte so behandelt werden, daß sein Parsing-Resultat (2) disambiguiert wird (d.h. eine der beiden Strukturen ausgewählt wird), das Resultat in eine deutsche Featurestruktur konvertiert wird (falls keine Interlingua benutzt wird ...) und dann ein deutscher Satz generiert wird, der diese Featurestruktur (hoffentlich als einzige Featurestruktur) trägt. Unabhängig vom Parsing- und Generierungsproblem ist bei diesem Ansatz das Disambiguierungsproblem zu lösen. Gerade bei diesem Beispiel spielt das Disambiguierungsproblem aber eine Nebenrolle, da es einen deutschen Satz gibt, der exakt dieselbe Ambiguität wie der originale englische Satz aufweist:

(3) **Hans sah den Mann mit dem Fernrohr**

Wir wollen noch bemerken, daß nicht für alle Quellsätze eine ambiguitätserhaltende Übersetzung existiert. Dies wird durch den englischen Satz (4) belegt, wobei Deutsch wiederum als Zielsprache fungiert:

(4) **The duck is ready to eat**

Die zwei Lesarten von (4) sind:

(5a) *ready(duck, eat(someone, duck))*
(5b) *ready(duck, eat(duck, something))*

Diese Lesarten können nur durch zwei verschiedene deutsche Sätze ausgedrückt werden:

(6a) **Die Ente kann jetzt gegessen werden**
(6b) **Die Ente ist zum Fressen bereit** [4]

Wir sehen anhand dieser Beispiele, daß der Disambiguierungsprozess unter Umständen vermieden werden kann, manchmal aber für eine angemessene Übersetzung nötig ist (wenn kein ambiguitätserhaltender Zielsatz existiert). Die Performance des oben skizzierten Übersetzungssystems könnte verbessert werden, wenn wir den (schwierigen und daher teuren) Disambiguierungsprozess nur dann anstoßen, wenn er nötig ist, d.h. wenn kein Zielsatz generiert werden kann, der exakt die Lesarten des Quellsatzes besitzt. Im folgenden werden wir allerdings zeigen, daß wir gerade dies nicht entscheiden können:

Satz: Seien $F_1, ..., F_n \quad (n \geq 2)$ Featurestrukturen einer beliebigen unifikationsbasierten Grammatik. Dann ist es nicht entscheidbar, ob ein String $w \in {V_T}^*$ existiert, sodaß:

$$\begin{array}{c} \Delta(w, F_1) \\ \vdots \\ \Delta(w, F_n) \end{array} \qquad \square$$

Beweis: Wir reduzieren das Problem darauf, für zwei beliebige kontexfreie Grammatiken zu entscheiden, ob ihr Schnitt leer ist. Dieses Problem ist als unentscheidbar bekannt.
Seien $G^1 =< V_N^1, V_T^1, S^1, Prod^1 >$ und $G^2 =< V_N^2, V_T^2, S^2, Prod^2 >$ zwei kontextfreie Grammatiken. O.b.d.A. sei $V_N^1 \cap V_N^2 = \emptyset$. Sei die unifikationsbasierte Grammatik $< V_N, V_T, S, \mathcal{L}, Prod >$ gegeben durch:

$$V_N = V_N^1 \cup V_N^2 \cup \{S\} \qquad \text{wobei } S \notin V_N^1 \cup V_N^2$$

$$V_T = V_T^1 \cup V_T^2$$

$$Prod = Prod^1 \cup Prod^2 \cup \{ \begin{array}{l} S \longrightarrow S^1 \ \{x_{n_0} A \doteq 1\}, \\ S \longrightarrow S^2 \ \{x_{n_0} A \doteq 2\} \ \} \end{array}$$

[4] Im deutschen Slang kann (6b) auch die Lesart (5a) haben. Vermutlich würde dann aber in (6b) nicht der Ausdruck **ist ... bereit** gewählt werden, sondern eher **ist ... fertig**. Im gehobenen Deutsch subkategorisiert das Verb **fressen** ein Subjekt-Argument, welches nichtmenschlich sein muß.

Gemäß Konstruktion gilt offenbar für jedes $w \in {V_T}^*$:

$$\Delta(w, [A\ 1]) \text{ und } \Delta(w, [A\ 2])$$

gdw. $S^1 \Longrightarrow^* w$ und $S^2 \Longrightarrow^* w$

gdw. $w \in L(G^1) \cap L(G^2)$

(Hierbei ist $L(G) = \{w \in {V_T}^* | S \Longrightarrow^* w\}$ die *Sprache* einer Grammatik G.) Somit gilt:

$$\exists w \in {V_T}^* \text{ mit: } \Delta(w, [A\ 1]),\ \ \Delta(w, [A\ 2])$$

gdw. $L(G^1) \cap L(G^2) \neq \emptyset$

Angenommen, es gibt einen Algorithmus, der für jede Grammatik G entscheidet, ob es zu vorgegebenen Featurestrukturen $F_1, ..., F_n$ ein $w \in {V_T}^*$ gibt, sodaß $\Delta(w, F_1), ..., \Delta(w, F_n)$.
Dann gibt es einen Algorithmus, der insbesondere für jede kontextfreie Grammatik G^1 und G^2 entscheidet, ob $L(G^1) \cap L(G^2) \neq \emptyset$. Widerspruch $\Box$

Eine Konsequenz dieses Satzes ist es, daß ambiguitätserhaltende Übersetzung nicht entscheidbar sein kann. Dies schließt allerdings nicht aus, daß für gewisse Sprachpaare und Grammatikfragmente - etwa für PP-Attachment im Englischen und Deutschen - Lösungen gefunden werden können.
Ebenso ist es denkbar, daß Restriktionen für unifikationsbasierte Grammatiken gefunden werden können, sodaß der vorgestellte Beweis für restringierte Grammatiken nicht mehr funktioniert. Im nächsten Kapitel werden wir solche Grammatikrestriktionen kennenlernen.

Kapitel 5

Generator-Restriktionen für unifikationsbasierte Grammatiken

In Kapitel 4 haben wir einen Überblick über Generierung in unifikationsbasierten Grammatiken gewonnen. Wir können festhalten, daß die zwei folgenden Auffassungen von Generierung miteinander konkurrieren:

Generierung ist der Prozess, welcher (alle) Sätze erzeugt, die eine Featurestruktur tragen, die mit einer *vorgegebenen* Featurestruktur

- identisch (Wedekind, [9])
- kompatibel oder teilweise identisch (VanNoord, [5])

ist.

Die zweite Auffassung von Generierung ist für die Praxis sehr angenehm: um den Generierungsvorgang anzustoßen, müssen wir keine (im allgemeinen riesigen) Featurestrukturen vorgeben, sondern können uns auf (kleine) Teil- Featurestrukturen beschränken, welche nur gewisse - für uns interessante - Attribute tragen.
Ein grundlegendes theoretisches Ergebnis von VanNoord besagt aber, daß es keinen Algorithmus geben kann, der für beliebige Grammatiken und beliebige vorgegebene Featurestrukturen entscheiden kann, ob es *wenigstens einen* Satz gibt, dessen Featurestruktur mit der vorgegebenen Featurestruktur kompatibel oder teilweise identisch ist (geschweige denn *alle* solchen Sätze aufzählt). In Kapitel 4 haben wir dieses Ergebnis in dem von uns benutzten Formalismus vorgestellt.
Eine offene Frage ist, ob ein Algorithmus existiert, dem dies wenigstens für bestimmte Grammatikklassen gelingen könnte.

Wenden wir uns der zweiten Auffassung von Generierung zu, so haben wir ein

(teilweise) erfreuliches theoretisches Ergebnis: 1995 konnte Wedekind in [9] einerseits zeigen, daß hier das Generierungsproblem entscheidbar ist, andererseits ein Generator (d.h. ein Algorithmus, welcher *alle* Sätze erzeugt, die eine vorgegebene Featurestruktur tragen) im allgemeinen nicht existieren kann. Wir haben dieses Ergebnis in Kapitel 4 vorgestellt.
Eine offene Frage war bisher, ob wenigstens für (nicht-triviale) Grammatikklassen ein Generator existiert.
Wie wir in der Einleitung und in Kapitel 4 ausgeführt haben, könnte die positive Beantwortung dieser Frage einige wichtige praktische Auswirkungen in der maschinellen Sprachübersetzung haben: die Performance dieser Systeme könnte im Hinblick auf Disambiguierungsprobleme erheblich verbessert werden (Wedekind, Kaplan [10]).

Ziel dieses Kapitels ist es deshalb, für unifikationsbasierte Grammatiken Eigenschaften anzugeben, die die Existenz eines Generators garantieren. Wir legen hierbei die Wedekindsche Auffassung von Generierung zugrunde. Während der Entstehung dieser Arbeit hat sich herausgestellt, daß sich für Grammatiken, die zyklenfreie Featurestrukturen benutzen, relativ problemlos Eigenschaften auffinden lassen, die die Existenz eines Generators garantieren.
Für Grammatiken mit zyklischen Featurestrukturen müssen wir den Terminus des *semantischen Kopfes* einführen. Obwohl semantische Köpfe nicht integraler Bestandteil unifikationsbasierter Grammatiken sind, scheint deren Einführung unproblematisch zu sein. Wir werden dies für den Spezialfall LFG demonstrieren.
Die auf diese Weise erhaltenen Grammatik-Klassen könnten wir nach einer Idee von Shieber [6] - siehe Kapitel 4 - *semantisch monoton* nennen.
Wir wollen darauf hinweisen, daß der Terminus des semantischen Kopfes auch in der Arbeit von VanNoord [5] eine entscheidende Rolle spielt. Allerdings werden semantische Köpfe in seiner Arbeit nicht benutzt, um Grammatik-Eigenschaften zu formulieren, die die Existenz eines Generators garantieren. (Vielmehr benutzt VanNoord semantische Köpfe, um bestimmte Generierungsstrategien (Head- Corner) einzuführen.)

Aus den erwähnten Gründen besteht dieses Kapitel aus zwei Abschnitten, in denen wir unifikationsbasierte Grammatiken mit zyklenfreien bzw. zyklischen Featurestrukturen voneinander gesondert untersuchen.

5.1 Unifikationsbasierte Grammatiken mit zyklenfreien Featurestrukturen

Wedekind's Hauptergebnis in [9] ist, daß es in einer unifikationsbasierten Grammatik nur endlich viele Sätze geben kann, die eine vorgegebene Featurestruktur tragen - vorausgesetzt, daß im Generierungsprozess nur Syntaxbäume ohne redundante Rekursionen vorkommen.
Wedekind erhält dieses Ergebnis, ohne irgendwelche Wohlgeformtheitsbedingungen für die Featurestrukturen zu fordern.
Unter der Wohlgeformtheitsbedingung, daß alle grammatischen Sätze eine zyklenfreie [1] Featurestruktur tragen müssen, könnte die Forderung von Syntaxbäumen ohne *redundante* Rekursionen etwas zu stark sein.
Ein genauerer Blick in Wedekind's Beweis zeigt, daß es sehr wesentlich ist, Syntaxbäume mit (beliebig oft) *pumpbaren* Rekursionen abzublocken. Wir definieren dazu:

Definition: Eine Rekursion $< up, down >$ in einem Syntaxbaum Syn heißt *pumpbar* gdw. es eine redundante Rekursion $< up', down' >$ in einem Syntaxbaum Syn' gibt, sodaß:

$$\begin{array}{ll} (i) & Syn = Syn'[up'|down'] \\ (ii) & down = up' \text{ oder } down' = up \end{array}$$

Syn' heißt dann *gepumpter* Syntaxbaum $\square$ [2]

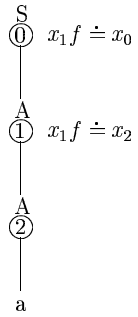
Das folgende Beispiel demonstriert, daß es redundante Rekursionen gibt, welche nicht pumpbar sind.

Beispiel: Für die Grammatik

$$\begin{array}{ll} S \longrightarrow A & \{x_{n_1} f \doteq x_{n_0}\} \\ A \longrightarrow A & \{x_{n_0} f \doteq x_{n_1}\} \\ A \longrightarrow a & \end{array}$$

betrachten wir den folgenden Syntaxbaum Syn mit Rekursion $< 1, 2 >$:

[1] siehe Kapitel 3.1, Wohlgeformtheitsbedingung *cycle-free*
[2] Oft wird noch verlangt, daß die Teilsyntaxbäume $< up, down >$ und $< up', down' >$ isomorph sind.



Die Featurebeschreibung von Syn lautet:

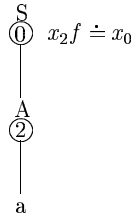
$$x_1 f \doteq x_0$$
$$x_1 f \doteq x_2$$

Der Constraint hat die Normalform:

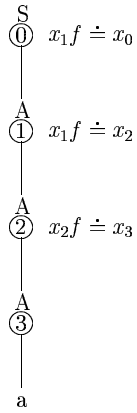
$$x_1 f \doteq x_1 f$$

Wir erhalten also die Featurestruktur:

Der Syntaxbaum $Syn[1|2]$



besitzt offensichtlich dieselbe Featurestruktur. Somit ist die Rekursion $< 1, 2 >$ in Syn redundant.
Wird in Syn die Rekursion $< 1, 2 >$ gepumpt, so erhalten wir den folgenden Syntaxbaum Syn':



Wir wenden auf seine Feature-Beschreibung

$$x_1 f \doteq x_0$$
$$x_1 f \doteq x_2$$
$$x_2 f \doteq x_3$$

das Normalisierungsverfahren an. Elimination von x_0 ergibt:

$$x_1 f \doteq x_1 f$$
$$x_1 f \doteq x_2$$
$$x_2 f \doteq x_3$$

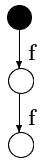
Das erste Literal fällt wegen der Subtermregel weg. Nach Elimination von x_2 erhalten wir:

$$x_1 f \doteq x_1 f$$
$$x_1 f f \doteq x_3$$

Wieder fällt das erste Literal hinaus. Elimination von x_3 ergibt dann die Normalform:

$$x_1 f f \doteq x_1 f f$$

Syn' hat also die folgende Featurestruktur:



Die Bäume Syn' und $Syn = Syn'[2|3]$ haben also nicht dieselbe Featurestruktur,

d.h. die Rekursion $< 2, 3 >$ ist in Syn' nicht redundant. Somit kann die Rekursion $< 1, 2 >$ in Syn nicht pumpbar sein □

Anhand des Beispiels sehen wir, daß jedes Pumpen der Rekursion $< 1, 2 >$ die Featurestruktur (um eine Kante f) verlängert. Im Generierungsprozess werden alle Sätze gesucht, die eine Featurestruktur tragen, welche mit einer vorgegebenen Featurestruktur identisch ist. Die vorgegebene Featurestruktur ist in diesem Abschnitt zyklenfrei, besitzt somit einen (nicht eindeutig bestimmten) längsten Pfad. Ist λ die Länge dieses Pfades, so bedeutet dies für unsere sehr einfache Beispielgrammatik, daß nur Syntaxbäume in Frage kommen, die höchstens λ Rekursionen aufweisen. Die Anzahl aller Syntaxbäume, die die vorgegebene Featurestruktur tragen, muß also endlich sein.
Untersuchen wir, welche Eigenschaft unserer Beispielgrammatik für diesen Sachverhalt verantwortlich ist, so stoßen wir sehr schnell auf:

(GEN1')

> Das Minimalmodell M der Featurebeschreibung
>
> $$\bigcup_{n \in Rec(up,down)} constraint(n)$$
>
> einer jeden Rekursion $< up, down >$ eines beliebigen Syntaxbaumes erfüllt mindestens ein Literal der Form
>
> $$x_{up}\ell \doteq x_{down} \qquad (\ell \in Attr^+)$$

Wenn wir für eine unifikationsbasierte Grammatik nachweisen können, daß sie (GEN1') erfüllt [3], so können wir sehr leicht zeigen, daß nur endlich viele Syntaxbäume dieser Grammatik eine vorgegebene Featurestruktur tragen können. Wir halten dies fest:

Lemma 5.1.1: Sei $< V_N, V_T, S, \mathcal{L}, Prod >$ eine unifikationsbasierte Grammatik mit Wohlgeformtheitsbedingung (*cycle-free*). Es gelte (GEN1'). Sei F eine wohlgeformte Featurestruktur und Syn ein Syntaxbaum mit $f\text{–}structure(Syn) = F$. Dann gilt:

$$|Syn| \leq |V_N| \cdot Nodes(F)$$

Hierbei bezeichnet $|Syn|$ die Länge des längsten Pfades in Syn und $Nodes(F)$ die Anzahl der Knoten von F □

Die im Lemma angegebene Abschätzung ist noch sehr grob. An dieser Stelle

[3] Wir stellen zunächst die Frage zurück, ob (GEN1') eine Grammatik-Eigenschaft ist, d.h. ein Algorithmus existiert, der mit einer beliebigen unifikationsbasierten Grammatiken als Eingabe, entscheiden kann, ob diese (GEN1') erfüllt.

kommt es uns allerdings nur darauf an, zu demonstrieren, daß (GEN1') die richtige Grammatikeigenschaft ist.

Beweis: Sei $n = Nodes(F)$. Angenommen, es gibt einen Syntaxbaum Syn mit $f\text{–}structure(Syn) = F$ und einem Pfad p, der (echt) länger als $|V_N| \cdot n$ ist.
Dann gibt es ein $A \in V_N$, welches wenigstens $(n+1)$-mal auf p vorkommt.
(Denn: kommt jedes Nonterminal höchstens n-mal auf p vor, so ist p - inklusive der einen Kante, die zu dem Terminal führt - von einer Länge kleinergleich $|V_N| \cdot n$. Widerspruch.)
O.B.d.A. können wir also annehmen, daß

$$1, 2, ..., \lambda + 1 \qquad (\text{mit } \lambda \geq n)$$

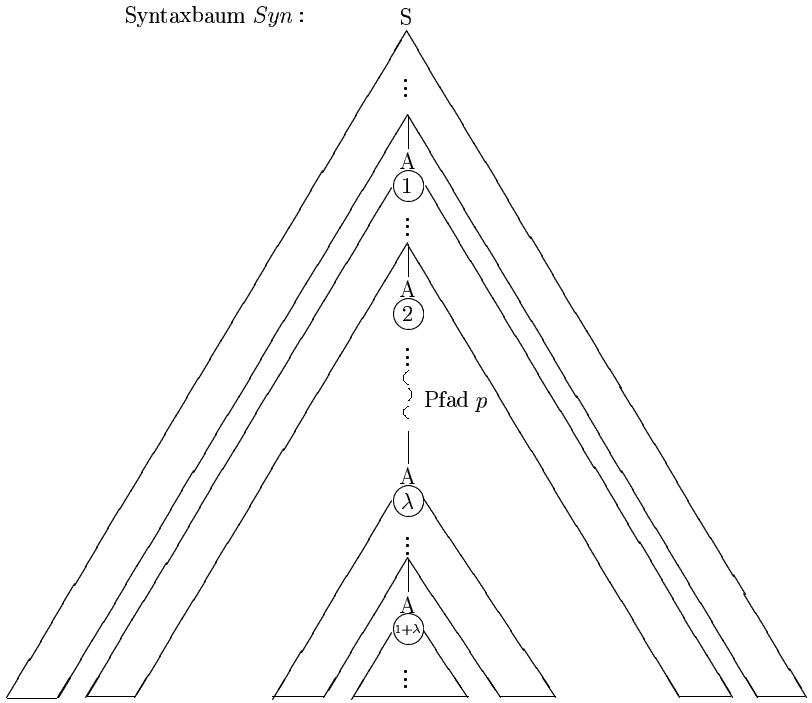
Knoten des Syntaxbaumes Syn sind, sodaß

$$\begin{array}{ll} k \text{ dominiert } k+1 & (k = 1, ..., \lambda) \\ cat(k) = A & (k = 1, ..., \lambda + 1) \end{array}$$

Die folgende Abbildung stellt den Syntaxbaum Syn (für $A \neq S$) dar. Wir haben von den λ nicht-überlappenden Rekursionen

$$< 1, 2 >, ..., < \lambda, \lambda + 1 >$$

die erste und letzte Rekursion eingetragen:



Der wesentliche Beweisgang ist nun einfach:
Sei M das Minimalmodell von $f\!-\!description(Syn)$,d.h. $F = f\!-\!structure(M)$.
Wegen (GEN1') gilt für alle Rekursionen $< k, k+1 >$ $\quad (k = 1, ..., \lambda)$:

$$M \models x_k \ell_k \doteq x_{k+1}$$

Hierbei sind die $\ell_k \in Attr^+$. Insbesondere sind also

$$[\![x_1]\!]^M, [\![x_2]\!]^M, ..., [\![x_{\lambda+1}]\!]^M$$

definiert. Da M wie F nur aus n Knoten besteht, aber $\lambda \geq n$ ist, muß:

$$M \models x_i \doteq x_j$$

für zwei verschiedene Knoten $i, j \in \{1, 2, ..., \lambda + 1\}$. O.B.d.A. sei $i < j$. Dann ist $< i, j >$ natürlich ebenfalls eine Rekursion. Wieder mit (GEN1') gilt:

$$M \models x_i \ell \doteq x_j$$

Somit enthält M einen Zyklus. Widerspruch □

Wir wenden uns nun der Frage zu, ob (GEN1') eine *entscheidbare* Grammatik-Eigenschaft ist. Das Problem besteht offensichtlich darin, daß eine Rekursion $< up, down >$ selbst wieder Rekursionen enthalten kann. Die in (GEN1') betrachteten Rekursionen bilden im allgemeinen also eine unendliche Menge von Teil-Syntaxbäumen der betrachteten unifikationsbasierten Grammatik. Ein Algorithmus, der mit den unifikationsbasierten Grammatiken als Eingabe entscheidet, ob für die jeweilige Grammatik (GEN1') gilt oder nicht gilt, scheint also zunächst fraglich. Viel einfacher wäre der Fall, wenn wir in (GEN1') nur elementare Rekursionen betrachten würden. Wir definieren dazu:

Definition: Sei $< up, down >$ eine Rekursion in einem Syntaxbaum Syn.
$< up, down >$ heißt *elementar* gdw. es keine Rekursion $< up', down' >$ in Syn gibt, sodaß

$$Rec(up', down') \subset Rec(up, down)$$

(Hierbei ist $\subset$ natürlich die echte Teilmengenrelation.) □

Die folgende Grammatik-Eigenschaft:

(GEN1)

> Das Minimalmodell M der Featurebeschreibung
>
> $$\bigcup_{n \in Rec(up,down)} constraint(n)$$
>
> einer jeden elementaren Rekursion $< up, down >$ eines beliebigen Syntaxbaumes erfüllt mindestens ein Literal der Form
>
> $$x_{up}\ell \doteq x_{down} \qquad (\ell \in Attr^+)$$

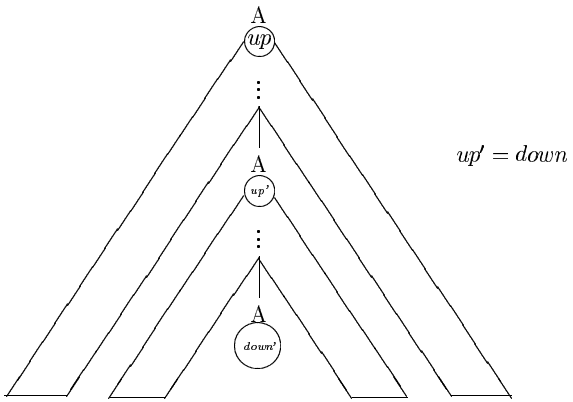
ist sicherlich entscheidbar, da das Minimalmodell der elementaren Rekursion $< up, down >$ vom jeweiligen Syntaxbaum, in dem sie eingebettet ist, unabhängig ist. Somit spielt es keine Rolle, daß die Menge dieser Syntaxbäume im allgemeinen wieder unendlich groß ist. Wichtig ist lediglich, daß das Nonterminal A mit $A = cat(up) = cat(down)$ vom Startsymbol aus erreichbar ist. Dies ist als entscheidbares Problem bekannt. Somit gibt es - in diesem Sinn - nur endlich viele elementare Rekursionen $< up, down >$, die betrachtet werden müssen. Mit dem Normalisierungsverfahren aus Kapitel 2 kann für jede dieser Rekursionen entschieden werden, ob ihr Minimalmodell ein Literal der Form $x_{up}\ell \doteq x_{down}$ mit $\ell \in Attr^+$ erfüllt.

Im folgenden wollen wir zeigen, daß (GEN1') aus der entscheidbaren Grammatik-Eigenschaft (GEN1) folgt:

$$\text{(GEN1)} \implies \text{(GEN1')}$$

Hierzu werden wir eine weitere Wohlgeformtheitsbedingung voraussetzen. (Wir werden sie später an geeigneter Stelle vorstellen.) Da wir Rekursionen induktiv durch *Hintereinanderschaltung* und *Ineinanderschachtelung* von Rekursionen aufbauen können (mit den elementaren Rekursionen als Atomen), wollen wir (GEN1') auch induktiv über den Aufbau von Rekursionen zeigen. Hierfür sei $\Re$ die Menge aller Rekursionen $< up, down >$ deren Minimalmodell ein Literal der Form $x_{up}\ell \doteq x_{down}$ mit $\ell \in Attr^+$ erfüllt. Der Induktionsanfang ist wegen (GEN1) trivial. Im Induktionsschluss müssen wir nachweisen, daß die Hintereinanderschaltung und Ineinanderschachtelung zweier Rekursionen aus $\Re$ wieder eine Rekursion aus $\Re$ ergibt.

Hintereinanderschaltung: Seien also $< up, down >$ und $< up', down' >$ zwei Rekursionen aus $\Re$, etwa mit $down = up'$. Wir erhalten dann:



Ist M das Minimalmodell der Hintereinanderschaltung $< up, down' >$, so folgt offensichtlich aus (GEN1'):

$$M \models x_{up}\ell \doteq x_{up'} \qquad (\ell \in Attr^+)$$
$$M \models x_{up'}\ell' \doteq x_{down'} \qquad (\ell' \in Attr^+)$$

Somit: $M \models x_{up}\ell\ell' \doteq x_{down'}$. Aus $\ell\ell' \in Attr^+$ folgt schliesslich $< up, down' > \in \Re$.

Ineinanderschachtelung: Hierfür nehmen wir an, daß die Grammatik die folgende Wohlgeformtheitsbedingung für Featuregraphen F besitzt:

(*constant/compound-clash-free'*)

Es gibt kein $x \in Var$ und kein $c \in Con$ mit $F \models x \doteq c$

Es ist sehr leicht einzusehen, daß (*constant/compound-clash-free'*) für unifikationsbasierte Grammatiken in etwa gleichwertig zur Wohlgeformtheitsbedingung (*constant/compound-clash-free*) ist. [4] Wie wir im Anhang zeigen werden, ist diese Annahme nötig, um in unifikationsbasierten Grammatiken mit Strings arbeiten zu können.

Unter der Wohlgeformtheitsbedingung (*constant/compound-clash-free'*) können wir für Rekursionen die folgende hilfreiche Beobachtung machen:
Ist M das Minimalmodell einer Rekursion $< up, down >$, so können wir aus

$$M \models x_{up}\ell \doteq x_{down}$$

leicht folgern, daß diese Elimination von x_{down} durch x_{up} sukzessive über die Constraints der Projektionslinie [5] der Syntaxbaumknoten up und $down$ geschieht.
D.h.:
Ist P das Minimalmodell für die Featurebeschreibung

$$\bigcup_{n \in Proj(up,down)} constraint(n)$$

der Projektionslinie $Proj(up, down)$, so gibt es zu jedem $p \in Proj(up, down)$ zwei Attributfolgen $\ell_{up} \in Attr^*$ und $\ell_{down} \in Attr^*$ (mit $\ell_{up}\ell_{down} = \ell$), sodaß

$$P \models x_{up} \quad \ell_{up} \doteq x_p$$
$$P \models x_p \quad \ell_{down} \doteq x_{down}$$

(Die Rückrichtung ist wegen $Proj(up, down) \subseteq Rec(up, down)$ trivial.)

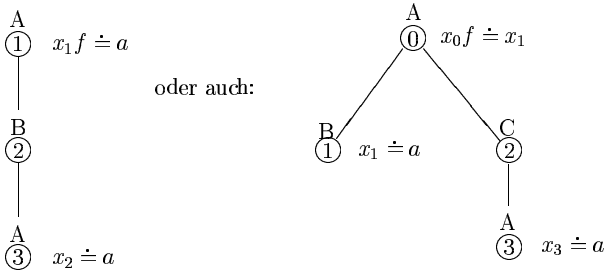
Hinweis: Für unifikationsbasierte Grammatiken, die nicht die Wohlgeformtheitsbedingung (*constant/compound-clash-free'*) fordern, muß dieses Resultat nicht gelten:

[4] siehe Kapitel 3, Schlußbemerkung des 1. Abschnitts

[5] Ist $< N, daughters >$ ein Baum und $m, n \in N$ mit n dominiert m, so heißt:

$$Proj(n, m) = \{n\} \cup \{p \in N | n \text{ dominiert } p \text{ und } p \text{ dominiert } m\}$$

die *Projektionslinie* von n nach m.



Nach diesen Vorbereitungen wollen wir zeigen, daß die Ineinanderschachtelung zweier Rekursionen aus $\Re$ wieder eine Rekursion aus $\Re$ ergibt.
Sei dazu die Rekursion $< up, down >$ *nicht* elementar gewählt. Ist etwa $< up', down' >$ eine Rekursion mit $Rec(up', down') \subset Rec(up, down)$, so ist zu zeigen:

$$\begin{array}{ll} & < up', down' > \in \Re, \;\; < up, down > [up' | down'] \in \Re \\ \Longrightarrow & < up, down > \in \Re \end{array}$$

Das weitere Vorgehen hängt davon ab, ob die Expansion der Rekursion $R =< up, down > [up'|down']$ zu $< up, down >$ vermittels $< up', down' >$ auf der Projektionslinie $Proj(R)$ von R geschieht.
Wenn ja, gilt:

$$(i) \quad Proj(up', down') \subseteq Proj(up, down)$$

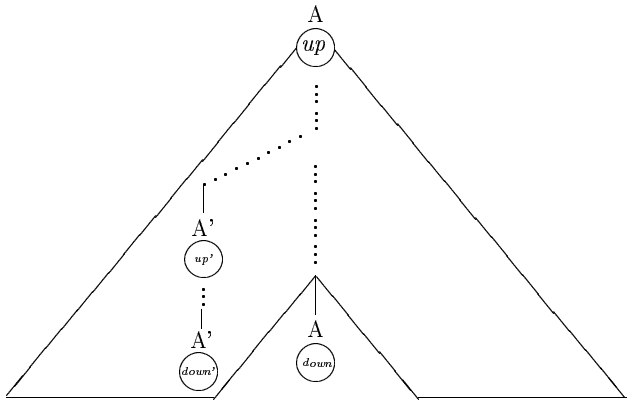
Wenn nein, so:

$$(ii) \quad Proj(up', down') \cap Proj(up, down) = \emptyset$$

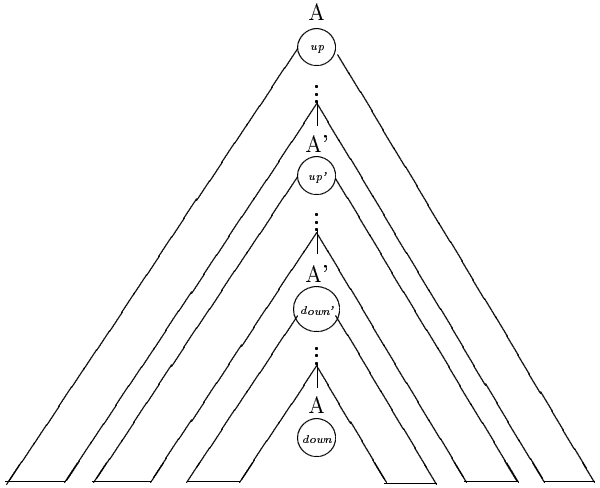
Der zweite Fall ist trivial, da hier:

$$Proj(R) = Proj(up, down)$$

D.h. aus $R \in \Re$ folgt gemäß Vorbemerkung sofort $< up, down > \in \Re$. Die folgende Abbildung verdeutlicht die Situation:



Im zweiten Fall haben wir dagegen folgende Situation vor uns:



Ist M_R das Minimalmodell von $R \in \Re$, so gilt:

$$M_R \models x_{up} \;\; \ell \doteq x_{down} \quad (\ell \in Attr^+)$$

Gemäß Vorbemerkung also (mit $\ell_{up}\ell_{down'} = \ell$):

$$\begin{array}{l} M_R \models x_{up} \;\; \ell_{up} \doteq x_{down'} \\ M_R \models x_{down'} \;\; \ell_{down'} \doteq x_{down} \end{array}$$

Ist M das Minimalmodell von $< up, down >$, so gilt daher:

$$M \models x_{up} \;\; \ell_{up} \doteq x_{up'}$$
$$M \models x_{down'} \;\; \ell_{down'} \doteq x_{down}$$

(Bei der ersten Gleichung war darauf zu achten, daß $x_{down'}$ durch $x_{up'}$ ersetzt wird, da der Constraint am Mutterknoten von up' richtig instantiiert werden muß. Außerdem benutzen wir an dieser Stelle erneut die Vorbemerkung.)

Ist M' das Minimalmodell von $< up', down' > \in \Re$, so gilt:

$$M' \models x_{up'} \;\; \ell' \doteq x_{down'} \quad (\ell' \in Attr^+)$$

Somit auch:

$$M \models x_{up'} \;\; \ell' \doteq x_{down'} \quad (\ell' \in Attr^+)$$

Zusammenfassend:

$$M \models x_{up} \;\; \ell_{up}\ell'\ell_{down'} \doteq x_{down}$$

Da $\ell_{up}\ell'\ell_{down'} \in Attr^+$ ist, gilt $< up, down > \in \Re$ $\quad \square$

Wir sind nun in der Lage, eine verbesserte Version von Lemma 5.1.1 anzugeben. Einerseits haben wir im folgenden Lemma 5.1.2 die entscheidbare Grammatik-Eigenschaft (GEN1) berücksichtigt, andererseits konnte die obere Grenze für $|Syn|$ verkleinert werden.

Lemma 5.1.2: Sei $< V_N, V_T, S, \mathcal{L}, Prod >$ eine unifikationsbasierte Grammatik mit den Wohlgeformtheitsbedingungen (*cycle-free*), (*constant/compound-clash-free'*). Es gelte (GEN1). Sei F eine wohlgeformte Featurestruktur und Syn ein Syntaxbaum mit $f\text{–}structure(Syn) = F$. Dann gilt:

$$|Syn| \leq |V_N| \cdot (|F| + 1)$$

Hierbei bezeichnet $|Syn|$ die Länge eines längsten Pfades in Syn und $|F|$ die Länge eines längsten Pfades in F $\quad \square$

Beweis: Sei $n = |F| + 1$. Angenommen, es gibt einen Syntaxbaum Syn mit $f\text{–}structure(Syn) = F$ und einem Pfad p, der (echt) länger als $|V_N| \cdot n$ ist.
Dann gibt es ein $A \in V_N$, welches wenigstens $(n+1)$-mal auf p vorkommt.
(Denn: kommt jedes Nonterminal höchstens n-mal auf p vor, so ist p - inklusive der einen Kante, die zu dem Terminal führt - von einer Länge kleinergleich $|V_N| \cdot n$. Widerspruch.)
O.B.d.A. können wir also annehmen, daß

$$1, 2, ..., \lambda + 1 \qquad (\text{mit } \lambda \geq n)$$

Knoten des Syntaxbaumes Syn sind, sodaß

$$k \text{ dominiert } k+1 \quad (k = 1, ..., \lambda)$$
$$cat(k) = A \qquad\qquad (k = 1, ..., \lambda + 1)$$

Dann gibt es λ nicht-überlappende Rekursionen

$$< 1, 2 >, ..., < \lambda, \lambda + 1 > \quad (k = 1, ..., \lambda)$$

Sei $k \in \{1, ..., \lambda\}$ beliebig, aber fest gewählt. Sei M_k das Minimalmodell der Rekursion $< k, k+1 >$. Unabhängig davon, ob die Rekursion $< k, k+1 >$ elementar ist (oder eine Rekursion eines Nonterminals $A' \neq A$ in ihr vorkommt), liefert (GEN1) vermittels (GEN1') ein $\ell_k \in Attr^+$, sodaß:

$$M_k \models x_k \;\; \ell_k \doteq x_{k+1}$$

Ist M das Minimalmodell von Syn, so gilt also:

$$M \models x_1 \;\; \ell_1 \;\; \doteq \;\; x_2$$
$$\vdots$$
$$M \models x_\lambda \;\; \ell_\lambda \;\; \doteq \;\; x_{\lambda+1}$$

Zusammenfassend:

$$M \models x_1 \;\; \ell_1 ... \ell_\lambda \doteq x_{\lambda+1}$$

Da F die zyklenfreie Featurestruktur von M ist, gibt es in F also einen Pfad mindestens der Länge λ. D.h.:

$$|F| \geq \lambda \geq n$$

Widerspruch zu $n = |F| + 1$ $\quad \square$

Die entscheidbare Grammatik-Eigenschaft (GEN1) besagt, daß in einer Rekursion die Featurestruktur des *up*-Knotens die Featurestruktur des *down*-Knotens als Teilfeaturestruktur enthalten soll. Dies scheint eine etwas zu starke Eigenschaft zu sein. Sehr viel schwächer wäre die Forderung, daß auch die Featurestruktur des *down*-Knotens die Featurestruktur des *up*-Knotens als Teilfeaturestruktur enthalten könnte. Also:

(GEN2 “)

Das Minimalmodell M der Featurebeschreibung

$$\bigcup_{n \in Rec(up,down)} constraint(n)$$

einer jeden Rekursion $< up, down >$ eines beliebigen Syntaxbaumes erfüllt mindestens ein Literal der Form ($\ell \in Attr^+$):

$$\begin{array}{rl} & x_{up}\ \ell \doteq x_{down} \\ \text{oder} & x_{up}\ \doteq x_{down}\ \ell \end{array}$$

Wir überzeugen uns, daß eine unifikationsbasierte Grammatik mit (GEN2 “) einen Generator besitzt:

Lemma 5.1.3: Sei $< V_N, V_T, S, \mathcal{L}, Prod >$ eine unifikationsbasierte Grammatik mit Wohlgeformtheitsbedingung (*cycle-free*). Es gelte (GEN2 “). Sei F eine wohlgeformte Featurestruktur und Syn ein Syntaxbaum mit $f\text{–}structure(Syn) = F$. Dann gilt:

$$|Syn| \leq |V_N| \cdot Nodes(F)$$

Hierbei bezeichnet $|Syn|$ die Länge des längsten Pfades in Syn und $Nodes(F)$ die Anzahl der Knoten von F $\quad \square$

Beweis: Unter derselben Annahme wie im Beweis zu Lemma 5.1.1, erhalten wir zwei Knoten $i, j \in Nodes(Syn)$ (mit $i \neq j$), sodaß

$$M \models x_i \doteq x_j$$

und

$$M \models x_i \ell \doteq x_j$$

oder

$$M \models x_i \doteq x_j \ell$$

Hierbei ist $\ell \in Attr^+$, d.h. M, bzw. F enthält einen Zyklus. Widerspruch $\square$

Wenden wir uns der Entscheidbarkeit der Grammatik-Eigenschaft (GEN“) zu, so könnten wir wieder auf die Idee kommen, zunächst nur elementare Rekursionen

zu betrachten, d.h.

(GEN2 “)

Das Minimalmodell M der Featurebeschreibung

$$\bigcup_{n \in Rec(up,down)} constraint(n)$$

einer jeden elementaren Rekursion $< up, down >$ eines beliebigen Syntaxbaumes erfüllt mindestens ein Literal der Form ($\ell \in Attr^+$):

$$\begin{array}{rl} & x_{up}\ \ell \doteq x_{down} \\ \text{oder} & x_{up}\ \doteq x_{down}\ \ell \end{array}$$

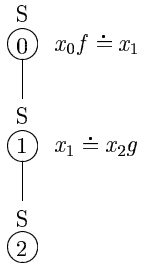
Im allgemeinen gilt jedoch (selbst unter der zusätzlichen Annahme (*constant/compound clash-free*)):

$$\text{(GEN2“)} \quad \not\Longrightarrow \quad \text{(GEN2“)}$$

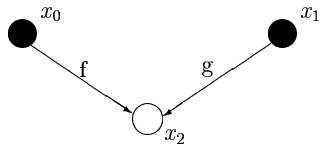
Beispiel: Für

$$\begin{array}{ll} S \longrightarrow S & \{x_{n_0} f \doteq x_{n_1}\} \\ S \longrightarrow S & \{x_{n_1} f \doteq x_{n_0}\} \\ S \longrightarrow a & \end{array}$$

gilt (GEN2“) aber nicht (GEN2“). Wir betrachten dazu die folgende Rekursion $< 0, 2 >$:



$< 0, 2 >$ hat das folgende Minimalmodell M:



Somit gibt es kein $\ell \in Attr^*$ (geschweige denn $\ell \in Attr^+$), sodaß:

$$\begin{array}{ll} & M \models x_{up}\ \ell \doteq x_{down} \\ \text{oder} & M \models x_{up}\ \doteq x_{down}\ \ell \end{array} \qquad \square$$

Eine etwas genauere Untersuchung zeigt, daß wir aus (GEN2“) unter der zusätzlichen Annahme (*constant/compound-clash-free'*) die folgende Grammatik-Eigenschaft folgern können:

(GEN2 ')

Das Minimalmodell M der Featurebeschreibung

$$\bigcup_{n \in Rec(up,down)} constraint(n)$$

einer jeden Rekursion $< up, down >$ eines beliebigen Syntaxbaumes erfüllt (für $k = 0, ..., p-1$):

$$\begin{array}{ll} & x_{n_k}\ \ell_k \doteq x_{n_{k+1}} \\ \text{oder} & x_{n_k}\ \doteq x_{n_{k+1}}\ \ell_k \end{array}$$

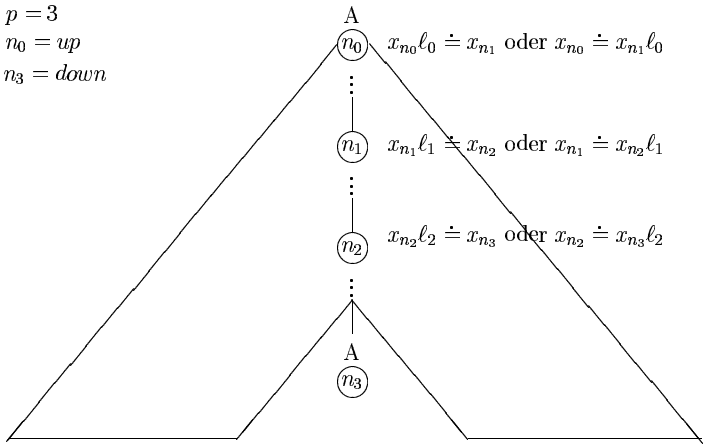
mit $\ell_k \in Attr^+$.
Hierbei ist $p \geq 1$ und $n_0 := up,\ \ n_p := down$, sowie

$$\{n_1, ..., n_{p-1}\} \subseteq Proj(up, down)$$

mit:

$$n_k \text{ dominiert } n_{k+1} \qquad (k = 0, ..., p-1)$$

Das folgende Bild verdeutlicht die Situation:



Offensichtlich erhalten wir (GEN2‘) aber unter der Wohlgeformtheitsbedingung (*constant/compound-clash-free'*) schon aus:

(GEN2)

Das Minimalmodell M der Featurebeschreibung

$$\bigcup_{n \in Rec(up,down)} constraint(n)$$

einer jeden elementaren Rekursion $< up, down >$ eines beliebigen Syntaxbaumes erfüllt (für $k = 0, ..., p-1$):

$$\begin{array}{ll} & x_{n_k}\ \ell_k \doteq x_{n_{k+1}} \\ \text{oder} & x_{n_k}\ \doteq x_{n_{k+1}}\ \ell_k \end{array}$$

mit $\ell_k \in Attr^+$.
Hierbei ist $p \geq 1$ und $n_0 := up,\ \ n_p := down$, sowie

$$\{n_1, ..., n_{p-1}\} \subseteq Proj(up, down)$$

mit:

$$n_k \text{ dominiert } n_{k+1} \qquad (k = 0, ..., p-1)$$

Die Grammatik-Eigenschaft (GEN2) ist sicherlich entscheidbar, da für jede der endlich vielen elementaren Rekursionen $< up, down >$ geprüft werden kann, ob die *endliche* Potenzmenge $\wp(Proj(up, down))$ der Rekursionsprojektionslinie eine Knotenmenge $\{n_1, ..., n_{p-1}\}$ mit den geforderten Eigenschaften enthält.

Wir zeigen nun:

Lemma 5.1.4: Sei $< V_N, V_T, S, \mathcal{L}, Prod >$ eine unifikationsbasierte Grammatik mit den Wohlgeformtheitsbedingungen (*cycle-free*), (*constant/compound-clash-free'*). Es gelte (GEN2). Sei F eine wohlgeformte Featurestruktur und Syn ein Syntaxbaum mit $f\text{–}structure(Syn) = F$. Dann gilt:

$$|Syn| \leq |V_N| \cdot (1 + 2 \cdot (1 + Roots(F)) \cdot |F|)$$

Hierbei bezeichnet $|Syn|$ die Länge eines längsten Pfades in Syn, sowie $Roots(F)$ die Anzahl der Wurzeln von F und $|F|$ die Länge eines längsten Pfades in F $\square$

Beweis: Sei $n = 1 + 2 \cdot (1 + Roots(F)) \cdot |F|$. Angenommen es gibt einen Syntaxbaum Syn mit $f\text{–}structure(Syn) = F$ und einem Pfad p, der (echt) länger als $|V_N| \cdot n$ ist.
Dann gibt es ein $A \in V_N$, welches wenigstens $(n+1)$-mal auf p vorkommt.
(Denn: kommt jedes Nonterminal höchstens n-mal auf p vor, so ist p - inklusive

der einen Kante, die zu dem Terminal führt - von einer Länge kleinergleich $|V_N| \cdot n$. Widerspruch.)
Somit gibt es also $\lambda \geq n$ nicht-überlappende Rekursionen:

$$< up_k, down_k > \quad (k = 1, ..., \lambda)$$

mit

$$down_k = up_{k+1} \quad (k = 1, ..., \lambda - 1)$$

Unabhängig davon, ob diese Rekursionen elementar sind, liefert (GEN2) wegen (GEN2') $\mu \geq \lambda$ nicht-überlappende Teilsyntaxbäume

$$< n_k, n_{k+1} > \quad (k = 1, ..., \mu)$$

mit ($k = 1, ..., \mu$ und $\ell_k \in Attr^+$):

$$\begin{array}{ll} & M \models x_{n_k} \ \ell_k \doteq x_{n_{k+1}} \\ \text{oder} & M \models x_{n_k} \ \doteq x_{n_{k+1}} \ \ell_k \end{array}$$

Hierbei ist M das Minimalmodell des Syntaxbaumes Syn.

Für $k = 1, ..., \mu$ sei

$$\sigma_k = \begin{cases} \downarrow & \text{falls} \quad M \models x_{n_k} \ \ell_k \doteq x_{n_{k+1}} \\ \uparrow & \text{falls} \quad M \models x_{n_k} \doteq x_{n_{k+1}} \ \ell_k \end{cases}$$

Somit ist $\sigma = \sigma_1 ... \sigma_\mu$ ein String der Länge μ über dem Alphabet $\{\downarrow, \uparrow\}$. Z.B.:

$$\sigma = \uparrow\uparrow\uparrow\downarrow\downarrow\uparrow\uparrow\uparrow\uparrow\downarrow \quad (\mu = 10)$$

Offenbar kann in σ jedes Zeichen höchstens $|F|$-mal hintereinander auftauchen. (Denn sonst gäbe es ein $s \in \{1, ..., \mu\}$ und ein $r > |F|$ mit $\sigma_s = ... = \sigma_{s+r-1}$. D.h.:

$$\begin{array}{ll} & M \models x_{n_s} \ \ell_s ... \ell_{s+r-1} \doteq x_{n_{s+r}} \\ \text{oder} & M \models x_{n_s} \doteq x_{n_{s+r}} \ \ell_{s+r-1} ... \ell_s \end{array}$$

In jedem Fall gibt es in M, also in F, einen Pfad der Länge $\geq r$.
D.h. $|F| \geq r$. Widerspruch.)

Als nächstes reduzieren wir σ zu τ, indem wir Teilsequenzen mit gleichen Zeichen zusammenfassen.

$$\begin{array}{lll} \text{Z.B. wird aus} & \sigma = & \uparrow\uparrow\uparrow\downarrow\downarrow\uparrow\uparrow\uparrow\uparrow\downarrow \\ & \tau = & \uparrow\downarrow\uparrow\downarrow \end{array}$$

(In τ erscheint nach einem $\downarrow$ also stets ein $\uparrow$ und umgekehrt.)

Ist ρ die Länge von τ, so erhalten wir (wegen unserer letzten Bemerkung):

$$\mu \leq \rho \cdot |F|$$

Der String $\tau = \tau_1 ... \tau_\rho$ steht nun für ρ nicht-überlappende Teilsyntaxbäume:

$$< k, k+1 > \qquad (k = 1, ..., \rho)$$

mit ($k = 1, ..., \rho, \ q_k \in Attr^+$):

$$\begin{cases} M \models x_k \ q_k \doteq x_{k+1} & \text{falls} \quad \tau_k = \downarrow \\ M \models x_k \doteq x_{k+1} \ q_k & \text{falls} \quad \tau_k = \uparrow \end{cases}$$

Wir beobachten nun, daß wir für jedes Auftreten des Substrings $\uparrow\downarrow$ in τ eine Wurzel in M annehmen müssen. Denn:
In diesem Fall haben wir in Syn (für ein $k \in \{1, ..., \rho - 1\}$) die Teilsyntaxbäume:

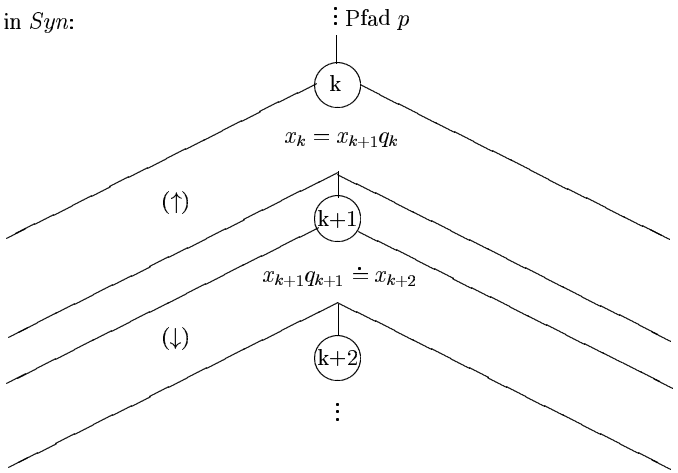
$$\begin{array}{l} < k, k+1 > \\ < k+1, k+2 > \end{array}$$

vorliegen und in M erhalten wir:

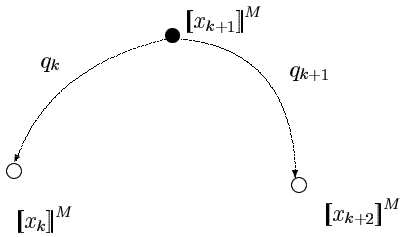
$$\begin{array}{ll} M \models x_k \doteq x_{k+1} \ q_k & (q_k \in Attr^+) \\ M \models x_{k+1} \ q_{k+1} \doteq x_{k+2} & (q_{k+1} \in Attr^+) \end{array}$$

D.h. wir haben folgende Situation vor uns:

in Syn:



in M:



Behauptung A: M hat eine Wurzel $[\![x_\alpha]\!]^M$, wobei α ein Knoten des Teilsyntaxbaums $< k, k+2 >$ ist.

Beweis: Trivialer Fall (i): $[\![x_{k+1}]\!]^M$ ist eine Wurzel von M.
Fall (ii): $[\![x_{k+1}]\!]^M$ ist keine Wurzel von M. Dann gibt es aber eine Wurzel $[\![x_\alpha]\!]^M$ von M und ein $\ell \in Attr^+$, sodaß

$$M \models x_\alpha \ell \doteq x_{k+1}$$

Angenommen: α ist kein Knoten des Teilsyntaxbaumes $< k, k+2 >$. Dann müßte die sukzessive Elimination von x_{k+1} durch x_α wegen der Wohlgeformtheitsbedingung (*constant/compound-clash-free'*) über den Knoten k oder den Knoten $k+2$

erfolgen. D.h. mit $\ell = \ell'\ell''$ gilt entweder:

$$M \models x_\alpha \ell' \doteq x_k$$
$$M \models x_k \ell'' \doteq x_{k+1}$$

oder:

$$M \models x_\alpha \ell' \doteq x_{k+2}$$
$$M \models x_{k+2} \ell'' \doteq x_{k+1}$$

Somit folgt entweder:

$$M \models x_k \ell'' q_k \doteq x_k$$

oder:

$$M \models x_{k+2} \ell'' q_{k+1} \doteq x_{k+2}$$

Da sowohl $q_k \in Attr^+$ als auch $q_{k+1} \in Attr^+$ gilt, ist M in keinem Fall zyklenfrei. Widerspruch □

Behauptung B: Sei $[\![x_\alpha]\!]^M$ eine Wurzel von M, wobei α ein Knoten des Teilsyntaxbaumes $< k, k+2 >$ ist. Sei $[\![x_\beta]\!]^M$ eine Wurzel von M, wobei β ein Knoten des Teilsyntaxbaumes $< k+2, k+4 >$ ist. Dann gilt:

$$[\![x_\alpha]\!]^M \neq [\![x_\beta]\!]^M$$

Beweis: Angenommen:

$$M \models x_\alpha \doteq x_\beta$$

Da die Teilsyntaxbäume $< k, k+2 >$ und $< k+2, k+4 >$ nicht-überlappend sind, muß diese Identifizierung wegen der Wohlgeformtheitsbedingung (*constant/compound-clash-free'*) über den Knoten $k+2$ erfolgen, d.h. es gibt ein $\hat{\ell} \in Attr^*$, sodaß:

$$M \models x_\alpha \doteq x_{k+2}\hat{\ell}$$
$$M \models x_\beta \doteq x_{k+2}\hat{\ell}$$

Im Beweis für Behauptung A hatten wir bemerkt, daß es ein $\ell \in Attr^*$ gibt, sodaß:

$$M \models x_\alpha \ell \doteq x_{k+1}$$

Ferner gilt:

$$M \models x_{k+1} q_{k+1} \doteq x_{k+2}$$

Zusammenfassend:

$$M \models x_\alpha \doteq x_\alpha \ell q_{k+1} \hat{\ell}$$

Da $q_{k+1} \in Attr^+$ enthält M bzw. F einen Zyklus. Widerspruch □

Durch die Behauptungen A und B wird unsere Beobachtung, daß wir für jedes Auftreten von $\uparrow\downarrow$ in τ eine Wurzel in M annehmen müssen, verifiziert. Ist ω die Anzahl der Substrings $\uparrow\downarrow$ in τ, so gilt daher:

$$Roots(M) \geq \omega$$

Eine einfache Rechnung zeigt:

$$\rho \leq 2 \cdot (\omega + 1)$$

Zusammenfassend:

$$n \leq \lambda \leq \mu \leq \rho \cdot |F| \leq 2(\omega + 1) \cdot |F| \leq 2(Roots(F) + 1) \cdot |F|$$

Widerspruch zur Definition von n $\square$

5.2 Unifikationsbasierte Grammatiken mit zyklischen Featurestrukturen

In diesem Abschnitt wollen wir uns mit unifikationsbasierten Grammatiken beschäftigen, die nicht notwendigerweise die Wohlgeformtheitsbedingung (*cycle-free*) verlangen. Für solche unifikationsbasierten Grammatiken ist es wegen des Unifikationsmechanismus sehr schwer, allgemeine Grammatik-Eigenschaften anzugeben, die verhindern, daß in einer Satzanalyse redundante Rekursionen vorkommen. Im Spezialfall LFG ist es allerdings sehr einfach, solche Grammatik-Eigenschaften zu finden, da LFG Prinzipien vorsieht, die es ermöglichen, Terminals, welche mit PRED-Annotationen versehen sind, als *semantische Köpfe* zu identifizieren. Die Idee ist einfach: wir werden verlangen, daß elementare Rekursionen wenigstens einen semantischen Kopf einführen. Rekursionen werden dann immer wenigstens eine semantische Information tragen, die nirgends sonst im Syntaxbaum vorkommt. Somit kann keine Rekursion redundant sein.

Diese Idee läßt sich auch auf beliebige unifikationsbasierte Grammatiken übertragen, sofern sie das Konzept von semantischen Köpfen unterstützen. Wir definieren hierfür:

Definition: Eine unifikationsbasierte Grammatik G unterstützt das Konzept der (lexikalischen) semantischen Köpfe, falls jedem Teilsyntaxbaum Syn eine Menge $\emptyset \subseteq H(Syn) \subseteq Nodes(Syn)$ von *(lexikalischen) semantischen Köpfen* zugeordnet werden kann, sodaß:

($H1$) Ist Syn' ein Teilsyntaxbaum von Syn, so gilt:
$H(Syn') \subseteq H(Syn)$

($H2$) Sind Syn_1 und Syn_2 zwei nicht-überlappende Teilsyntaxbäume von Syn, so gilt:
$H(Syn_1) \cap H(Syn_2) = \emptyset$

($H3$) Es gibt eine Funktion $\Phi : \mathcal{F} \longrightarrow I\!N$ mit:
$|H(Syn)| \leq \Phi(f\text{–}structure(Syn))$ $\square$

Bemerkung: Die Forderung ($H1$) erscheint natürlich. Die (sehr starke) Forderung ($H2$) kann gewaltsam erfüllt werden, indem man fordert, daß ein $n \in H(Syn)$ ein Blatt von Syn sein soll. Wie wir später in LFG sehen werden, sieht die Forderung ($H3$) nur stark aus $\square$

Wir führen nun unsere Grammatik-Eigenschaft ein:

(GEN3) Für jede elementare Rekursion R gilt: $H(R) \neq \emptyset$

Offensichtlich erhalten wir hieraus sofort:

(GEN3') Für jede Rekursion R gilt: $H(R) \neq \emptyset$

Denn: Ist R keine elementare Rekursion, so ist eine elementare Rekursion R_0 Teilsyntaxbaum von R. Mit $(H1)$ erhalten wir: $H(R_0) \subseteq H(R)$. Daher liefert (GEN3): $H(R) \neq \emptyset$.

Lemma 5.2.1: Sei $< V_N, V_T, S, \mathcal{L}, Prod >$ eine unifikationsbasierte Grammatik, welche das Konzept der (lexikalischen) semantischen Köpfe unterstützt. Es gelte (GEN3). Sei F eine Featurestruktur und Syn ein Syntaxbaum mit $f\text{–}structure(Syn) = F$. Dann gilt:

$$|Syn| \leq |V_N| \cdot (1 + \Phi(F))$$

Hierbei bezeichnet $|Syn|$ die Länge des längsten Pfades in Syn □

Beweis: Sei $n = \Phi(F) + 1$. Angenommen es gibt einen Syntaxbaum Syn mit $f\text{–}structure(Syn) = F$ und einem Pfad p, der (echt) länger als $|V_N| \cdot n$ ist.
Dann gibt es ein $A \in V_N$, welches wenigstens $(n+1)$-mal auf p vorkommt.
(Denn: kommt jedes Nonterminal höchstens n-mal auf p vor, so ist p - inklusive der einen Kante, die zu dem Terminal führt - von einer Länge kleinergleich $|V_N| \cdot n$. Widerspruch.)
Somit gibt es also $\lambda \geq n$ nicht-überlappende Rekursionen

$$R_k \quad (k = 1, ...\lambda).$$

Unabhängig davon, ob diese Rekursionen elementar sind, liefert (GEN3'):

$$|H(R_k)| \geq 1 \quad (k = 1, ...\lambda).$$

Mit $(H1)$ und $(H2)$ erhalten wir:

$$H(R_1) \cup ... \cup H(R_k) \subseteq H(Syn)$$
$$H(R_i) \cap H(R_j) = \emptyset \qquad (i \neq j)$$

Wenden wir nun noch $(H3)$ an, so erhalten wir zusammenfassend:

$$\Phi(F) \geq |H(Syn)| \geq |H(R_1)| + ... + |H(R_k)| \geq \lambda \geq n$$

Widerspruch zur Definition von n □

Wir wenden uns nun dem Spezialfall LFG zu und zeigen, daß LFG unser Konzept der lexikalischen semantischen Köpfe unterstützt.

Definition: Sei Syn ein LFG-Teilsyntaxbaum. Für einen Knoten $n \in Nodes(Syn)$ definieren wir:

$$n \in H(Syn)$$
gdw. $constraint(n) \ni \{x_n \text{ PRED } \doteq c\}$ für ein $c \in Con$ □

Wir zeigen nun der Reihe nach $(H1)$, $(H2)$ und $(H3)$.

zu $(H1)$ und $(H2)$: In LFG kommen PRED-Annotationen nur in Produktionen der Form

$$\text{A} \longrightarrow \text{a} \quad (\text{A} \in V_N,\ \text{a} \in V_T)$$

vor, bspw.

N⟶girl ↑NUM $\doteq$ SG
↑PRED $\doteq$ 'GIRL'

V⟶handed ↑TENSE $\doteq$ PAST
↑PRED $\doteq$ 'HAND...'

Somit ist $H(Syn)$ für jeden Teilsyntaxbaum Syn eine Teilmenge seiner Preterminalknoten. $H(.)$ erfüllt trivialerweise $(H1)$ und $(H2)$.

zu $(H3)$: Sei Syn ein Teilsyntaxbaum, M dessen Minimalmodell, sowie $F = f\text{–}structure(M)$. Wir definieren:

$$\Phi(F) = card\{c \in Con \mid \text{es gibt eine Kante } e \text{ in } F, \text{ mit: } label(e) = \text{PRED}, target(e) = [\![c]\!]^F \}$$

D.h. $\Phi(F)$ ist die Anzahl der verschiedenen PRED-Werte in F. (Die PRED-Werte sind verschieden, weil LFG die Wohlgeformtheitsbedingung (*constant-clash-free*) fordert.) Offensichtlich gilt:

$$\Phi(F) = card(C)$$

mit: $C = \{c \in Con \mid \exists x \in Var : M \models x \text{ PRED} \doteq c\}$

O.B.d.A. sei $H(Syn) = \{1, ..., \lambda\}$. Wir müssen zeigen:

$$\lambda \leq card(C)$$

Der Definition von $H(Syn)$ entnehmen wir sofort:

$$M \models x_k\text{PRED} \doteq c_k \text{ für gewisse } c_k \in Con \quad (k = 1, ..., \lambda)$$

D.h.

$$\{c_1, ..., c_\lambda\} \subseteq C.$$

Wenn wir nun noch zeigen können, daß die $c_1, ..., c_\lambda$ paarweise verschieden sind, muß $\lambda \leq card(C)$ sein.

Angenommen: $c_i = c = c_j$ (für zwei Syntaxbaumknoten $i \neq j$).
Dann gilt (und hier wenden wir die Definition von $H(Syn)$ in aller Schärfe an):

$$\begin{array}{ll} x_i\text{PRED} \doteq c & \in constraint(i) \\ x_j\text{PRED} \doteq c & \in constraint(j) \end{array}$$

Das PRED-Instantiierungs-Prinzip (siehe Kaplan u. Bresnan, [2], Seite 225 ff.) in LFG verlangt aber, daß eine PRED-Annotation, etwa:

$$\uparrow\text{PRED='BABY'}$$

an verschiedenen Knoten des Syntaxbaumes mit *verschiedenen* PRED-Werten instantiiert wird, also z.B.

$$\begin{array}{l} x_4\text{PRED='BABY'}_1 \\ x_6\text{PRED='BABY'}_2 \end{array}$$

Die verschiedenen Instanzen des PRED-Wertes sind dabei nicht als identisch anzusehen.
(Kaplan und Bresnan benötigen das PRED-Instantiierungs-Prinzip, damit in LFG Sätze wie

the baby was given a toy by the girl by the girl

ungrammatisch werden.)

Somit ist unsere Annahme falsch. Es folgt $\lambda \leq card(C)$. Zusammenfassend:

$$|H(Syn)| = \lambda \leq card(C) = \Phi(F) \qquad \square$$

Damit haben wir bewiesen:

Lemma 5.2.2: In einer LFG, in der jede elementare Rekursion einen semantischen Kopf (d.h. einen Knoten mit einer Annotation $\uparrow$PRED= c) besitzt, gilt für jeden Syntaxbaum mit $f\text{–}structure(Syn) = F$:

$$|Syn| \leq |V_N| \cdot (1 + Preds(F))$$

Hierbei bezeichnet $|Syn|$ die Länge des längsten Pfades in Syn und $Preds(F)$ die Anzahl der verschiedenen PRED-Werte in F $\square$

Anhang

Strings in unifikationsbasierten Grammatiken

In unifikationsbasierten Grammatiken ist es üblich, Strings oder Listen zu verwenden. (Wir könnten hierbei - um nur ein Beispiel zu nennen - an den Wert des PHON-Attributs oder auch an die SUBCAT-Liste in HPSG denken.)
In Abschnitt 4.1.3 haben wir den Beweis für die Unentscheidbarkeit des Pfad-Parsing-Problems vorgestellt. Die PCP-Grammatiken, die in diesem Beweis eine zentrale Rolle spielten, enthielten Constraints der Form:

$$\begin{array}{ll} t \doteq \sigma & (\sigma \in \Sigma^*, \ t \text{ ein Term}) \\ t \doteq d_1 + d_2 & (t, d_1, d_2 \text{ Terme}) \end{array}$$

Hierbei war Σ eine endliche Menge von Zeichen. Wir wollen nun versuchen, die Benutzung von Strings und der Stringkonkatenation zu rechtfertigen.

Im folgenden werden wir den Terminus „String“ nicht mehr benutzen, sondern nur noch von Listen sprechen.

Definition: Eine *Liste* σ ist eine endliche Folge von Symbolen aus einem Alphabet Σ, d.h.

$$\begin{array}{l} \sigma = < \sigma_1, ..., \sigma_n > \\ n \in \{0, 1, 2, 3, ...\} \\ \sigma_i \in \Sigma \qquad (i = 1, ..., n) \end{array} \qquad \square$$

In Logik-basierten Programmiersprachen, wie PROLOG, besteht der wesentliche Schritt zur Darstellung von Listen darin, die Liste rekursiv in einen *Kopf* und einen *Rest* aufzuteilen, bis schließlich als Rest die leere Liste <> übrigbleibt. Unsere Constraintsprache $\mathcal{L}$ wollen wir auf die gleiche Weise erweitern. Benutzen wir das Attribut *FIRST* zur Repräsentierung des Listenkopfes und das Attribut *REST* zur Repräsentierung des Listenrumpfes, so erhalten wir etwa die folgende

Featurestruktur ($n = 3$):

$$\begin{bmatrix} \text{FIRST} & \sigma_1 \\ \text{REST} & \begin{bmatrix} \text{FIRST} & \sigma_2 \\ \text{REST} & \begin{bmatrix} \text{FIRST} & \sigma_3 \\ \text{REST} & <> \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Diese Repräsentation wird unseren Ansprüchen aber nicht genügen, da wir später in der Lage sein wollen, zwei Listen zu konkatenieren. Ein sehr einfacher Weg führt hierzu über den Begriff der Differenzliste.

Definition: Seien a, b zwei Listen. Für $a = \sigma b$ sei:

$$a - b =_{def} \sigma$$

Wir nennen $a - b$ eine *Differenzliste* □

Offensichtlich kann jede Liste als Differenzliste geschrieben werden:

$$\sigma = \sigma - <>$$

Der Begriff der Differenzliste ist also etwas allgemeiner als der Begriff der Liste. Benutzen wir die Attribute *IN* und *OUT* für die Argumente der Differenzlistenoperation, so könnten wir etwa

$$< \sigma_1\sigma_2\sigma_3 > - <>$$

als

$$\begin{bmatrix} \text{IN} & \begin{bmatrix} \text{FIRST} & \sigma_1 \\ \text{REST} & \begin{bmatrix} \text{FIRST} & \sigma_2 \\ \text{REST} & \begin{bmatrix} \text{FIRST} & \sigma_3 \\ \text{REST} & <> \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{OUT} & <> \end{bmatrix}$$

repräsentieren. Offensichtlich ist dann

$$\begin{bmatrix} \text{IN} & \begin{bmatrix} \text{FIRST} & \sigma_1 \\ \text{REST} & \begin{bmatrix} \text{FIRST} & \sigma_2 \\ \text{REST} & \begin{bmatrix} \text{FIRST} & \sigma_3 \\ \text{REST} & x \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{OUT} & x \end{bmatrix}$$

eine Repräsentation für:

$$< \sigma_1\sigma_2\sigma_3 x > - x \quad .$$

Unabhängig von der Restliste x erhalten wir hierfür aber stets die Liste

$$< \sigma_1\sigma_2\sigma_3 > \quad .$$

Es liegt also sehr nahe, daß wir

$$\begin{bmatrix} \text{IN} & \begin{bmatrix} \text{FIRST} & \sigma_1 \\ \text{REST} & \begin{bmatrix} \text{FIRST} & \sigma_2 \\ \text{REST} & \begin{bmatrix} . & & & & . \\ & . & & . & \\ & & . & & \\ & & \begin{bmatrix} \text{FIRST} & \sigma_n \\ \text{REST} & x \end{bmatrix} & & \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{OUT} & x \end{bmatrix}$$

als Repräsentation für $< \sigma_1 ... \sigma_n >$ ansehen. Wir halten dies formal fest:

Definition: Sei $< \sigma_1 ... \sigma_n >$ eine Liste und t ein Term. Wir schreiben:

$$t \doteq < \sigma_1 ... \sigma_n >$$

als Abkürzung für den Constraint:

$$\begin{array}{lll} t\ IN\ REST^{\,i-1}\ FIRST \doteq \sigma_i & & (1 \leq i \leq n) \\ t\ IN\ REST^{\,n} \doteq t\ OUT & & \\ t\ IN\ REST^{\,j} \not\doteq t\ IN\ REST^{\,i} & & (0 \leq i < j \leq n) \end{array}$$

(Hierbei steht $REST^{\,i}$ für $REST \ldots REST$ (i-mal)) □

Beispiel: Für die leere Liste ($n = 0$) erhalten wir etwa:

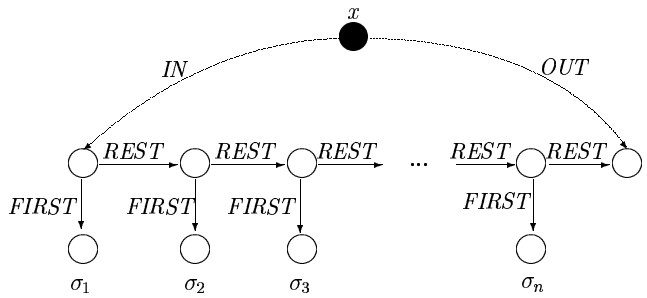
$$t \doteq <>$$

steht als Abkürzung für

$$t\ IN \doteq t\ OUT$$

(keine negativen Literale) □

In der obigen Defintion tauchen negative Literale auf. Wegen ihnen besitzt ein Constraint $x \doteq < \sigma_1 ... \sigma_n >$ stets ein *zyklenfreies* Minimalmodell:



Das folgende Lemma ist recht einfach zu beweisen:

Lemma: Seien σ, τ Listen und t ein Term. Dann gilt:

$\sigma = \tau$
gdw. es gibt einen Featuregraphen F mit:
(i) F ist constant-clash-free
(ii) $F \models t \doteq \sigma,\ t \doteq \tau$ □

Statt eines Beweises betrachten wir:

Beispiel: Für

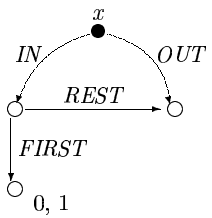
$$x \doteq < 1 >$$
$$x \doteq < 0 >$$

erhalten wir den Constraint:

x IN FIRST $\doteq$ 1
x IN REST $\doteq$ x OUT
x IN REST $\not\doteq$ x IN

x IN FIRST $\doteq$ 0
x IN REST $\doteq$ x OUT
x IN REST $\not\doteq$ x IN

Das Normalisierungsverfahren liefert den folgenden Featuregraphen:



D.h. der Constraint ist zwar erfüllbar, aber nicht wohlgeformt. Für zwei ungleich lange Listen σ, τ wird der Constraint $t \doteq \sigma,\ \ t \doteq \tau$ (wegen der negativen Literale) nicht erfüllbar sein. Wir betrachten:

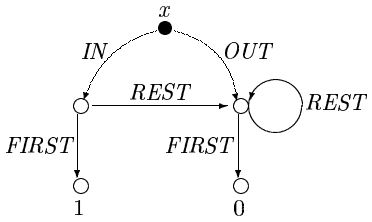
$$x \doteq < 1 >$$
$$x \doteq < 1, 0 >$$

und erhalten den Constraint:

x IN FIRST $\doteq$ 1
x IN REST $\doteq$ x OUT
x IN REST $\not\doteq$ x IN

x IN FIRST $\doteq$ 1
x IN REST FIRST $\doteq$ 0
x IN REST REST $\doteq$ x OUT
x IN REST $\not\doteq$ x IN
x IN REST REST $\not\doteq$ x IN
x IN REST REST $\not\doteq$ x IN REST

Wir führen nicht das Normalisierungsverfahren durch, sondern überzeugen uns, daß der Constraint ohne die negativen Literale erfüllbar wäre:



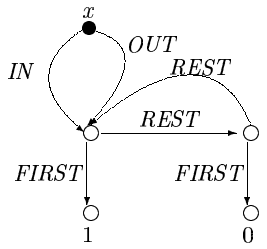
Dieser Featuregraph enthält einen Zyklus, der verhindert, daß der Featuregraph das letzte negative Literal des Constraints erfüllt.
Als letztes Beispiel betrachten wir die leere Liste. Für

$$x \doteq <>$$
$$x \doteq < 1, 0 >$$

erhalten wir den Constraint:

$$\begin{array}{lcl} x \text{ IN} & \doteq & x \text{ OUT} \\ \\ x \text{ IN FIRST} & \doteq & 1 \\ x \text{ IN REST FIRST} & \doteq & 0 \\ x \text{ IN REST REST} & \doteq & x \text{ OUT} \\ x \text{ IN REST} & \not\doteq & x \text{ IN} \\ x \text{ IN REST REST} & \not\doteq & x \text{ IN} \\ x \text{ IN REST REST} & \not\doteq & x \text{ IN REST} \end{array}$$

Ohne die negativen Literale erhalten wir als Minimalmodell:



Auch dieser Featuregraph enthält einen Zyklus. Er verhindert, daß das vorletzte negative Literal durch den Featuregraphen erfüllt wird □

Bemerkung: Die negativen Literale in der Definition für $t \doteq \sigma$ bewirken, daß wir im obigen Lemma für die Featuregraphen (zusätzlich zu der Wohlgeformtheitsbedingung *constant-clash-free* [6]) nicht auch noch die Wohlgeformtheitsbedingung *cycle-free* fordern müssen.
VanNoord gibt in seiner Arbeit [5] einerseits *nicht* die negativen Literale an, andererseits hat seine Grammatik auch *nicht* die Wohlgeformtheitsbedingung *cycle-free*. Dies ist unserer Meinung nach ein Fehler □

Wir wenden uns nun der Konkatenation zweier Listen zu. Sie ist über den Umweg der Differenzlisten relativ einfach zu realisieren, denn es gilt (für geeignete a, b, σ):

$$a - b = (a - \sigma) + (\sigma - b)$$

Über den Ansatz:

$$\begin{array}{lcl} t & \doteq & a - b \\ d_1 & \doteq & a - \sigma \\ d_2 & \doteq & \sigma - b \end{array}$$

[6] genauer benötigen wir eigentlich nur die Eigenschaft Σ-*constant-clash-free*: kein Knoten des Featuregraphen darf mit zwei (oder mehr) Konstanten aus Σ gelabelt sein.

führt dies zu:

Definition: Seien t, d_1, d_2 Terme ($d_1 \neq d_2$). Dann schreiben wir

$$t \doteq d_1 + d_2$$

als Abkürzung für den Constraint:

$$\begin{array}{lcl} t\ IN & \doteq & d_1\ IN \\ d_1\ OUT & \doteq & d_2\ IN \\ d_2\ OUT & \doteq & t\ OUT \end{array} \qquad \square$$

Bemerkung: Die Bedingung $d_1 \neq d_2$ ist sehr wichtig. Wir rechnen dazu ein Beispiel. Für

$$\begin{array}{lcl} x & \doteq & <1,0> \\ z & \doteq & x + x \end{array}$$

erhielten wir (gemäß Definition) unter anderem das Literal:

$$x\ OUT \doteq x\ IN$$

(Auf etwas vage Art und Weise könnten wir hier schon bemerken, daß x also neben der Liste $<1,0>$ auch die leere Liste repräsentieren würde.)
Das Normalisierungsverfahren zeigt, daß der Constraint wegen der Literale

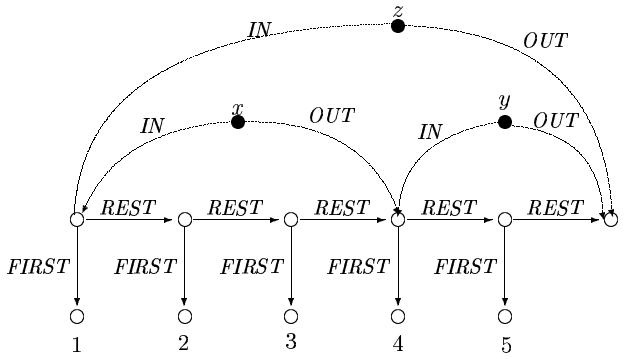
$$\begin{array}{lcl} x\ OUT & \doteq & x\ IN \\ x\ IN\ REST\ REST & \doteq & x\ OUT \\ x\ IN\ REST\ REST & \not\doteq & x\ IN \end{array}$$

nicht erfüllbar ist □

Beispiel: Der Constraint

$$\begin{array}{lcl} x & \doteq & <1,2,3> \\ y & \doteq & <4,5> \\ z & \doteq & x + y \end{array}$$

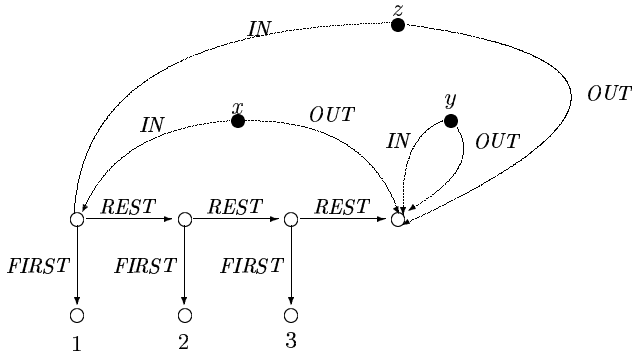
hat das Minimalmodell:



Der Constraint

$$\begin{array}{lcl} x & \doteq & <1,2,3> \\ y & \doteq & <> \\ z & \doteq & x+y \end{array}$$

hat das Minimalmodell:



□

Wir schliessen diesen Abschnitt mit dem folgenden Lemma, welches die formale Begründung für die obige Definition darstellt.

Lemma: Seien σ, τ Listen, t, d_1, d_2 Terme ($d_1 \neq d_2$). Sei F ein Featuregraph (constant-clash-free). Dann gilt:

$$\begin{array}{rc} & F \models d_1 \doteq \sigma,\ \ d_2 \doteq \tau,\ \ t \doteq d_1 + d_2 \\ \Longrightarrow & F \models t \doteq \sigma + \tau \end{array}$$

(Hierbei ist $\sigma + \tau$ die Konkatenation der Listen σ und τ. Die Literale $t \doteq \sigma + \tau$ und $t \doteq d_1 + d_2$ sind unabhängig voneinander definiert ...) □

Literaturverzeichnis

[1] Hopcroft, John E. und Ullman, Jeffrey D. (1979). *Introduction to Automata Theory, Languages and Computation.* Addison Wesley.

[2] Kaplan, R. und Bresnan J. (1982) *Lexical-Functional Grammar: A Formal System for Grammatical Representation.* In Bresnan J., ed., *The Mental Representation of Grammatical Relations.* The MIT Press, Cambridge, Mass.

[3] Johnson, M. (1988) *Attribute-Value Logic and the Theory of Grammar.* Chicago: CSLI Lecture Notes Series, Chicago University Press.

[4] Prestel, Alexander (1986). *Einführung in die mathematische Logik und Modelltheorie.* Vieweg-Verlag, Braunschweig.

[5] VanNoord, Gerardus Johannes Maria (1993). *Reversibility in Natural Language Processing .* Dissertation Reichsuniversität Utrecht, ISBN 90-9005661-0 .

[6] Shieber, Stuart M. (1988). *A Uniform Architecture for Parsing and Generation.* In Proceedings of the 12th International Conference on Computational Linguistics (COLING), Budapest.

[7] Wedekind, Jürgen (1988). *Generation as structure driven derivation.* In Proceedings of the 12th International Conference on Computational Linguistics (COLING), Budapest.

[8] Wedekind, Jürgen (19..). *Some Remarks on the Logic of Unification Grammars.* In C.J. Rupp, M.A. Rosner, R.L. Johnson. *Constraints, Language and Computation.* Academic Press, Harcourt Brace and Company, Publishers.

[9] Wedekind, Jürgen (1995). *Some Remarks on the Decidability of the Generation Problem in LFG- and PATR-Style Unification Grammars.*

[10] J. Wedekind, R. M. Kaplan (1995). *Ambiguity Preserving Generation with LFG- and PATR-Style Grammars.*