Diplomarbeit

Mikroplanungstechniken zur Präsentation mathematischer Beweise

Armin Fiedler

Nach einem Thema von Prof. Dr. Jörg H. Siekmann
FB14, Informatik, Universität des Saarlandes

Hiermit versichere ich an Eides statt, daß ich diese Arbeit selbständig verfaßt habe und keine anderen als die angegebenen Hilfsmittel verwandt habe.

Saarbrücken, 26. März 1996

An dieser Stelle möchte ich mich bei Prof. Siekmann für die Vergabe der Arbeit bedanken.

Xiaorong Huang schulde ich größten Dank für die intensive Betreuung, die wesentlich zum Gelingen dieser Arbeit beigetragen hat. Auch Elisabeth Maier, Michael Kohlhase und Helmut Horacek danke ich herzlich für die zahlreichen Diskussionen.

Meine lieben Kollegen und Kommilitonen sorgten für ein sehr angenehmes Betriebsklima in unserer Arbeitsgruppe. Dafür möchte ich ihnen danken.

Bei meinen Eltern bedanke ich mich dafür, daß sie mir das Studium ermöglichten.

Ganz besonders danke ich meiner lieben Freundin Iris Himbert, die mich immer wieder aufzurichten verstand.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Um eine lange vernachlässigte Anforderung an automatische Beweiser zu erfüllen, nämlich leicht verständliche Beweise zu produzieren, entwickelte Xiaorong Huang das Beweispräsentationssystem *PROVERB*, das Beweise in englischer Sprache ausgibt [Huang, 1994a, Huang, 1994d].[1]

Ausgehend von einem Beweis des natürlichen Schließens, sucht das System zunächst nach einer Repräsentation des Beweises auf einer höheren Abstraktionsebene, der sogenannten Faktenebene. Dabei liegt die Abstraktion darin, daß ein Beweisschritt nicht nur durch eine Inferenzregel aus dem Kalkül des natürlichen Schließens begründet werden kann, sondern auch durch die Anwendung eines *Faktums*, d. h. einer Definition, eines Axioms oder eines bereits bewiesenen Satzes.

Der Beweis auf Faktenebene wird anschließend von dem Textplaner weiterverarbeitet. Hierbei wird zunächst die globale Struktur des Beweistextes festgelegt, bevor dessen linguistische Spezifikation erzeugt wird. Diese dient als Eingabe für den syntaktischen Realisierer TAG-GEN, der am DFKI entwickelt wurde [Kilger und Finkler, 1995].

In der vorliegenden Arbeit wird *PROMI* vorgestellt, ein neu entwickeltes Modul für *PROVERB*, das die Abbildung des Textplans auf die linguistische Spezifikation des Textes vornimmt. Der Text wird dabei in einer *Textstruktur* [Meteer, 1991] repräsentiert, die *PROMI* einerseits auf die Generierung sprachlich korrekt verbalisierbarer Texte eingeschränkt, ihm aber andererseits die volle Ausdrucksfähigkeit der Sprache zur Verfügung stellt. Dazu war eine Integration der Textstruktur mit dem *Upper Model* und der *Hierarchie abstrakter syntaktischer Kategorien* nötig, zwei sprachlich motivierten Konzepthierarchien für die ideationale und die textuelle Metafunktion gemäß der systemisch-funktionalen Linguistik. Zur Satzplanung mußte die Abbildung der Objekte des Anwendungsprogramm über Upper-Model-Objekte in die Textstruktur definiert werden. Die Abbildung geschieht über Upper-Model-Variablen und über mit den Konzepten des Upper Models assoziierten *Realisationsklassen*, die die Verbalisierungsmöglichkeiten der Konzepte repräsentieren.

Um den qualitativen Anforderungen eines menschlichen Lesers zu genügen, steigert *PROMI* die Qualität des Textes mit Hilfe spezieller Operatoren. Es wurden dafür *Aggregationsregeln* zur Manipulation von Upper-Model-Objekten spezifiziert, die zur Vermeidung von Redundanz im Text führen, während durch die Einfügung

[1] Da *PROVERB* Beweise in Englisch verbalisiert, sind auch die Beispiele in dieser Arbeit in Englisch.

von *Schlüsselwörtern* der Sprachfluß verbessert wird. Durch die Erweiterung der Textstruktur zur Planung des *Layouts* ist *PROMI* in der Lage, dem Leser zudem optische Information über die Struktur des Textes zu übermitteln.

Aber ist es überhaupt nötig Verfahren der Sprachgenerierung zur Planung von Beweistexten anzuwenden? Ist es für Mathematiker nicht viel bequemer, wenn ihnen Beweise in einer Formelkette präsentiert werden, die keine von der wesentlichen Information ablenkenden Kommentare mehr enthalten? Daß dies tatsächlich nicht der Fall ist, sondern daß auch Mathematiker natürlichsprachliche Beweise bevorzugen, und daß für deren Planung die Einbringung linguistischen Wissens notwendig ist, wird in diesem Kapitel erläutert.

1.1 Beweisgenerierung und -präsentation

Automatische Beweiser sind Computerprogramme, die vollautomatisch in einem durch einen maschinenorientierten Kalkül aufgespannten Suchraum Beweise für gegebene Sätze finden können. Sie sind in den letzten 25 Jahren zu solcher Stärke weiterentwickelt worden, daß sie mittlerweile auch nichttriviale Sätze beweisen können.

Eine wichtige Anforderung, die an solche automatische Beweiser gestellt wird, ist, daß der Benutzer eines solchen Systems in die Lage versetzt wird, die erzeugten Beweise zu verstehen. Er will nicht nur von der Korrektheit des erzeugten Beweises überzeugt werden, sondern den Beweis auch inhaltlich nachvollziehen können.

Es ist daher notwendig, daß automatische Beweiser die von ihnen gefundenen Beweise dem Benutzer des Systems vermitteln. Da Beweissysteme jedoch mit maschinenorientierten Verfahren wie der Resolution [Robinson, 1965] oder dem Matrixverfahren [Andrews, 1981, Bibel, 1983] arbeiten, wird der Beweis häufig in dem entsprechenden Kalkül ausgegeben.[2] Solche Beweise sind jedoch auch für die meisten Spezialisten nur schwer zu durchschauen, da sie nicht nur schwer verständlich, sondern auch wenig intuitiv sind, wie der Resolutionsbeweis in Abbildung 1.1 verdeutlicht.

Daher wurden Transformationsverfahren entwickelt, die Beweise aus solchen Kalkülen in Beweise des natürlichen Schließens übertragen [Andrews, 1980] [Miller, 1983, Pfenning, 1987, Lingenfelder, 1990, Nesmith und Pfenning, 1990]. Der Kalkül des natürlichen Schließens wurde in den dreißiger Jahren von dem deutschen Logiker Gerhard Gentzen entwickelt, dessen Hauptanliegen darin bestand, die in der Mathematik gebräuchlichen Schlußweisen in diesem Kalkül nachzubilden [Gentzen, 1935]. Der ND-Beweis[3] zu dem Resolutionsbeweis aus Abbildung 1.1, der in Abbildung 1.2 dargestellt ist, ist von dem menschlichen Benutzer in der Tat viel leichter zu verstehen.

In dem Gebiet der Sprachgenerierung wurde untersucht, wie solche ND-Beweise in natürlicher Sprache präsentiert werden können [Chester, 1976, McDonald, 1983] [Edgar und Pelletier, 1993]. Die Ergebnisse waren jedoch nicht von der erwarteten Qualität. Der Hauptgrund dafür liegt darin, daß sich ND-Beweise auf einer zu niedrigen Abstraktionsebene befinden. Analysiert man beispielsweise Beweise aus mathe-

[2]In extremen Fällen wird die Ausgabe gar auf ein schlichtes „ja“ oder „nein“ beschränkt, was dem Benutzer ein Veständnis des Beweises völlig unmöglich macht!

[3]ND steht für *natural deduction*.

Initiale Klauselmenge:
C1 $= \{+(u * u^{-1} = e)\}$ C2 $= \{+(e * w = w)\}$
C3 $= \{-(x \in S), -(y \in S), -(x * y^{-1} = z), +(z \in S)\}$
C4 $= \{+(v \in S)\}$ C5 $= \{-(q^{-1} \in S)\}$
Resolutionsschritte:
C3,4 & C3,1
$\rightarrow$ R1: $\{-(x \in S), -(y \in S), -(x * y^{-1} = z), -(y' \in S), -(z * y'^{-1} = z'),$
$+(z' \in S)\}$
R1,1 & C4,1
$\rightarrow$ R2: $\{-(y \in S), -(v * y^{-1} = z), -(y' \in S), -(z * y'^{-1} = z'), +(z' \in S)\}$
R2,1 & C4,1 $\rightarrow$ R3: $\{-(v * v^{-1} = z), -(y' \in S), -(z * y'^{-1} = z'), +(z' \in S)\}$
R3,2 & C4,1 $\rightarrow$ R4: $\{-(v * v^{-1} = z), -(z * v^{-1} = z'), +(z' \in S)\}$
R4,1 & C1,1 $\rightarrow$ R5: $\{-(e * v^{-1} = z'), +(z' \in S)\}$
R5,1 & C2,1 $\rightarrow$ R6: $\{+(v^{-1} \in S)\}$
R6,1 & C5,1 $\rightarrow$ R7: $\Box$

Abbildung 1.1: Ein Resolutionsbeweis

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall S_1, S_2 .\, S_1 \subset S_2 \Leftrightarrow (\forall x .\, x \in S_1 \Rightarrow x \in S_2)}{U \subset F \Leftrightarrow (\forall x .\, x \in U \Rightarrow x \in F)}\forall E}{U \subset F \Rightarrow (\forall x .\, x \in U \Rightarrow x \in F)}\Leftrightarrow E \qquad U \subset F}{\forall x .\, x \in U \Rightarrow x \in F}\Rightarrow E}{a \in U \Rightarrow a \in F}\forall E \qquad a \in U}{a \in F}\Rightarrow E$$

Abbildung 1.2: Der ND-Beweis zu Abbildung 1.1

matischen Lehrbüchern, stellt man sehr schnell fest, daß einzelne Ableitungsschritte meist nicht durch ND-ähnliche Schlußregeln begründet werden, sondern vielmehr durch die Anwendung mathematischer ***Fakten***, d. h. Definitionen, Axiomen oder bereits bewiesenen Sätzen. In [Huang, 1994e] wurde der Begriff der Anwendung eines Faktums formal gefaßt. Dadurch konnten ***Faktenbeweise*** definiert werden, in denen ähnlich den Beweisen in mathematischen Lehrbüchern die Begründung von Inferenzen durch die Anwendung eines Faktums erlaubt ist. Der Faktenbeweis zu dem ND-Beweis aus Abbildung 1.2 ist in Abbildung 1.3 dargestellt.

$$\dfrac{a \in U, \quad U \subset F}{a \in F}\text{Def-Subset}$$

Abbildung 1.3: Der Faktenbeweis zu Abbildung 1.2

Manche Systeme erstreben die natürlichsprachliche Ausgabe von Beweisen, indem sie die Beweise mit Hilfe von starren Schablonen verbalisieren. Dies erfolgt dadurch, daß eine Klasse von Schablonen alle möglichen Inferenzbegründungen verarbeitet, während die Formeln der Beweisschritte entweder direkt als Formeln ausgegeben werden oder durch eine zweite Klasse von Schablonen rekursiv verbalisiert werden. Die so verbalisierten Beweise entsprechen jedoch immer noch nicht den Erwartungen, wie das folgende Beispiel zeigt:

Beispiel: Betrachten wir einen beliebigen Beweis, der als Hypothese folgende Beweiszeile enthält:

$$Subset(S_1, S) \wedge Subset(S_2, S) \wedge Subset(S_3, S) \qquad \text{Hyp}$$

Eine Schablone der ersten Klasse, die für Hypothesen definiert wurde, erzeugt die folgende Ausgabe:

> “Let $Subset(S_1, S) \wedge Subset(S_2, S) \wedge Subset(S_3, S)$.”

Wird die Formel zusätzlich rekursiv verbalisiert, wird in etwa der folgende Satz ausgegeben:

> “Let S_1 be a subset of S and S_2 be a subset of S and S_3 be a subset of S.”

Ein Mathematiker erwartet jedoch eine Verbalisierung der Form:

> “Let S_1, S_2 and S_3 be subsets of S.” ■

Um eine solche Ausgabe produzieren zu können, ist es unumgänglich, Verfahren zu benutzen, die linguistisches Wissen einbringen. Solche Verfahren wurden im Gebiet der Sprachgenerierung entwickelt.

Ferner ist es notwendig, ND- oder Faktenbeweise, die stets als gerichtete, azyklische Graphen aufgefaßt werden können, zur Verbalisierung zu linearisieren. Methoden aus der Sprachgenerierung, mit deren Hilfe inhaltlich kohärente Texte geplant werden können, sollten hierzu Anwendung finden.

Die Methoden der Sprachgenerierung sind Thema des folgenden Abschnitts.

1.2 Sprachgenerierung und *PROVERB*

Die Generierung natürlicher Sprache erfordert zunächst die Festlegung der zu übertragenden Information. Sodann muß diese Information passend strukturiert werden, um eine kohärente, d. h. inhaltlich passend geordnete Ausgabe zu ermöglichen. Abschließend werden die zu benutzenden syntaktischen und lexikalischen Konstruktionen festgelegt.

Diese Aufgaben werden traditionell auf zwei Komponenten aufgeteilt. Die Auswahl der Information und ihre Strukturierung fällt einem Textplaner zu, der einen entsprechenden Textplan erzeugt. Die linguistische Ausgabe übernimmt eine Realisierungskomponente.

Allerdings sind die meisten Textgenerierungssysteme entweder in der Lage nur wenige Texte zu produzieren, die dann sprachlich eine hohe Qualität besitzen, oder aber sie sind in der Lage, viele Texte zu erzeugen, die jedoch sehr mechanisch wirken und daher den Anforderungen, die an einen Text gemeinhin gestellt werden, nicht genügen. Die Gründe dafür sind in der zweistufigen Architektur dieser Textgenerierungssysteme zu suchen [Hovy, 1992, Meteer, 1992].

Die oben angesprochenen Textgenerierungssysteme entsprechen dieser zweistufigen Arbeitsweise. Die Problematik dieser Zweiteilung besteht darin, daß der Textplaner die Information organisiert, ohne zu prüfen, ob es linguistische Ressourcen gibt, um sie entsprechend auszudrücken. Solche Systeme verfolgen keine generelle Methode, um die *Verbalisierbarkeit (expressibility)* ihrer Äußerungen sicherzustellen. Sie garantieren die sprachliche Korrektheit der Äußerung vielmehr dadurch, daß sie ihre eigene *Ausdrucksfähigkeit (expressiveness)* beschränken, beispielsweise, indem verlangt wird, daß atomare Einheiten der Information Propositionen sein müssen. Eine solche Proposition ist durch die Anwendung von Schemata immer als Satz realisierbar. Dadurch wird jedoch eine unflexible Verbalisierung der Sachverhalte in Kauf genommen. Um die volle Ausdrucksfähigkeit einer Sprache ausschöpfen zu können, ist es aber notwendig, auch den inneren Aufbau der Sätze in die Satzplanung miteinzubeziehen, anstatt sie lediglich in größere Strukturen einzuordnen.

Meteer stellt fest, daß die oben geschilderte Zweiteilung in Textplanung und Realisierung eine Lücke im Verbalisierungsmechanismus zur Folge hat, die sogenannte „generation gap“ zwischen dem von dem Textplaner erzeugten Plan und den linguistischen Ressourcen, auf die die Realisierungskomponente zurückgreift [Meteer, 1991]. Diese Lücke besteht darin, daß der Textplan auf statische Weise direkt in die linguistische Spezifikation überführt wird, ohne daß die interne Struktur der zu realisierenden Sätze auf flexible Art geplant worden wären.

Zusammenfassend läßt sich sagen, daß der Textplaner zwar in der Lage ist einen *kohärenten*, d. h. inhaltlich zusammenhängenden Text zu planen, er garantiert jedoch keinen *kohäsiven*, d. h. sprachlich zusammenhängenden Text. Um dieses Problem zu beheben, ist eine ausgeklügelte Architektur notwendig [Hovy, 1992].

Man verteilt daher heute die Aufgaben eines Textgenerierungssystem auf drei Teilmodule (vgl. Abbildung 1.4), nämlich:

1. den ***Makroplaner*** *(macroplanner, text planner, content determination component)*, der für die Auswahl der zu übermittelnden Information und für die Planung der Diskursstruktur zuständig ist,

2. den ***Mikroplaner*** oder ***Satzplaner*** *(microplanner, sentence planner)*, der für die Planung der syntaktischen Struktur der Sätze und für die Wortwahl verantwortlich ist und

3. den ***linguistischen Realisierer*** *(linguistic realizer, surface generator)*, der schließlich die sprachlich ausformulierten Sätze erzeugt.

Dieser Aufteilung folgend wurde auch *PROVERB* auf eine entsprechenden dreistufigen Architektur ausgelegt. Der Beweis auf Faktenebene wird zunächst von dem *Makroplaner* nach einem Verarbeitungsmodell verarbeitet, das zwei sich ergänzende Techniken, nämlich hierarchische Planung und lokale Organisation, verbindet. Er legt so die globale Struktur des Beweistextes fest und erzeugt eine Folge von ***Beweiskommunikationsaktionen*** (*proof communicative acts*, PCAs), deren jede entweder einem Schritt einer Ableitung entspricht oder aber Hinweise zur Struktur des Beweises gibt [Huang, 1994b].

Der Makroplaner muß dabei u. a. folgende Entscheidungen treffen:

- Wie soll der zu präsentierende Beweis in Teilbeweise gespalten werden?

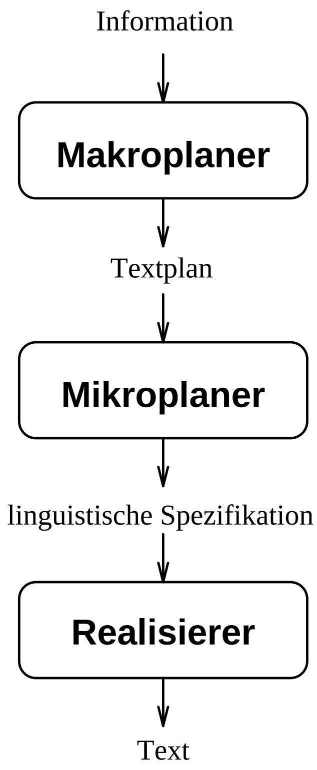


Abbildung 1.4: Die Architektur eines Textgenerators

- In welcher Reihenfolge sollen die Teilbeweise dargeboten werden?

Durch die rekursive Behandlung der Teilbeweise entsteht so eine *attentionale Hierarchie* der Teilbeweise, die die globale Struktur des Beweises widerspiegelt.

Ein Beispiel für eine PCA, die einen Beweisschritt repräsentiert, ist folgende:

Beispiel:

(`Derive Reasons:` $(element(a, F),\ subset(F, G))$
`Method: "Def-Subset"`
`Conclusion:` $element(a, G)$)

`Derive` zeigt an, daß diese PCA die Repräsentation eines Beweisschrittes darstellt. Die Voraussetzungen des Ableitungsschrittes sind durch `Reasons` gekennzeichnet, die Schlußfolgerung durch `Conclusion`. `Method` bezeichnet die Begründung für diese Inferenz. Konkret beschreibt diese PCA, daß aus $a \in F$ und $F \subset G$ nach der Definition der Teilmenge $a \in G$ folgt. ∎

Die folgende PCA vermittelt den Übergang zu einem neuen Teilbeweis:

Beispiel:

(`Begin-Cases Goal:` G
`Assumptions:` $(A,\ B)$)

Durch `Begin-Cases` wird angezeigt, daß der neue Teilbeweis eine Fallunterscheidung beinhaltet. `Goal` bezeichnet die Formel, die durch die Fallunterscheidung bewiesen werden soll, und `Assumptions` deren Fallannahmen. ∎

Der *Mikroplaner* trifft anhand einer Diskurstheorie erst die Entscheidung, ob und wenn ja wie sich das System auf die als Prämissen benutzten Zwischenergebnisse und die als Begründung benutzte Inferenzmethode beziehen soll [Huang, 1994c]. Wie in dieser Arbeit noch detailliert dargestellt wird, erzeugt er sodann eine linguistische Spezifikation des Beweistextes, anhand deren der *syntaktische Realisierer* TAG-GEN [Kilger und Finkler, 1995] den ausformulierten Text erzeugt.

1.3 Anforderungen an die Mikroplanung

Die erste Version von *PROVERB* verfügte nur über einen einfachen Mikroplaner. Die von dem Makroplaner erzeugten PCAs wurden lediglich bezüglich der Form der Referenzierung ihrer Zwischenergebnisse und Inferenzmethoden festgelegt, bevor sie auf statische Art und Weise in eine linguistische Spezifikation überführt wurden. Dadurch wurde im Prinzip jeder Ableitungsschritt in einem Satz verbalisiert. Aufeinanderfolgende PCAs ließen sich in bestimmten Fällen durch einen eingeschränkten Mechanismus zu einer einzelnen PCA zusammenfassen. Das folgende Beispiel gibt einen Eindruck von der Qualität der erzeugten Texte:

Beispiel:
Es handelt sich in diesem Beispiel um den Beweis zu folgendem Satz, der einem deutschsprachigen mathematischen Lehrbuch entnommen wurde [Deussen, 1971]:

> **Satz 1.9** Ist F eine Gruppe, $U \subset F$ eine Untergruppe und sei $1_U \in U$ das Einselement von U, so ist $1 = 1_U$.

Aus dem maschinell generierten ND-Beweis erzeugt *PROVERB* nach seiner Abstraktion auf die Faktenebene folgende Sequenz von PCAs:

1. (Derive, Method: "Hyp",
   Conclusion: $group(F, *)$, $subgroup(U, F, *)$,
   $unit(F, 1, *)$, $unit(U, 1_U)$)
2. (Derive, Method: "Def-Semigroup*Unit",
   Conclusion: $element(1_U, U)$)
3. (Derive, Conclusion: $\exists x.\ element(x, U)$)
4. (Begin-Assume-Choice, Parameters: (x, U),
   Conclusion: $element(u_1, U)$)
5. (Derive, Method: "Def-Semigroup*Unit",
   Conclusion: $u_1 * 1_U = u_1$)
6. (Derive, Reasons: $subgroup(U, F, *)$
   Conclusion: $subset(U, F)$)
7. (Derive, Conclusion: $element(u_1, F)$)
8. (Similar, Reasons: ($element(1_U, U)$),
   Conclusion: $element(1_U, F)$)

9. (Derive, Reasons: ($group(F,*)$),
   Conclusion: $semigroup(F,*)$)
10. (Derive, Reasons: ($u_1 * 1_U = U$) ,
   Conclusion: $solution(u_1, u_1, 1_U, F, *)$)
11. (Derive, Reasons: ($unit(F, 1, *)$),
   Conclusion: $u_1 * 1 = u_1$)
12. (Derive, Reasons: ($unit(F, 1, *)$),
   Conclusion: $element(1, F)$)
13. (Derive, Reasons: ($element(u_1, F)$),
   Conclusion: $solution(u_1, u_1, 1, F, *)$)
14. (Derive, Method: Th-solution",
   Conclusion: $1_U = 1$)
15. (End-Assume-Choice, Parameters: (U),
   Conclusion: $1_U = 1$)

PCA Nr. 1 ist ein Beispiel für die oben erwähnte Zusammenfassung aufeinanderfolgender PCAs. Vier konsekutive PCAs, die die Voraussetzungen des Beweises darstellen, wie durch `Method: "Hyp"` gekennzeichnet ist, wurden zu einer einzigen PCA zusammengefaßt. Die Eingeschränktheit dieses Mechanismus wird an den PCAs Nr. 11 und 12 deutlich, die in ihren Voraussetzungen noch redundante Information enthalten. Auch hier wäre eine Zusammenfassung wünschenswert.

Anschließend generiert *PROVERB* den folgenden Beweistext:

(1) Let F be a group and U be a subgroup of F and 1 be a unit element of F and 1_U be a unit element of U.

(2) According to the definition of unit element $1_U \in U$.

(3) Therefore there is an X, $X \in U$.

(4) Now suppose that u_1 is such an X.

(5) According to the definition of unit element $u_1 * 1_U = u_1$.

(6) Since U is a subgroup of F $U \subset F$.

(7) Therefore $u_1 \in F$.

(8) Similarly $1_U \in F$ since $1_U \in U$.

(9) Since F is a group F is a semigroup.

(10) Since $u_1 * 1_U = u_1$ 1_U is a solution of the equation $u_1 * X = u_1$.

(11) Since 1 is a unit element of F $u_1 * 1 = u_1$.

(12) Since 1 is a unit element of F $1 \in F$.

(13) Since $u_1 \in F$ 1 is a solution of the equation $u_1 * X = u_1$.

(14) Since F is a group $1_U = 1$ by the uniqueness of solution.

(15) This conclusion is independent of the choice of the element u_1. ■

Durch die schablonenhafte Verbalisierung der PCAs werden die Sätze sehr ähnlich verbalisiert. Dadurch wirkt der Text insgesamt mechanisch. Eine flexible Satzplanung würde hier die Textqualität erheblich verbessern. Dabei wäre darauf zu achten, daß sowohl die Wortwahl flexibler gestaltet wird, als auch die Satzkonstruktionen abwechslungsreicher ausgewählt werden.

Ein weiterer Kritikpunkt ist die Verbalisierung jeder PCA durch genau einen Satz. Die daraus resultierende direkte Übertragung der Beweisstruktur in die Struktur des Textes verhindert eine natürlich wirkende Verbalisierung. Durch die gemeinsame Betrachtung der Gesamtheit der PCAs bei der Satzplanung lassen sich flexiblere Texte generieren, in denen die Satzgrenzen nicht notwendigerweise die Beweisstruktur widerspiegeln.

Insgesamt macht das obige Beispiel die mangelnde Kohäsion trotz ausreichender Kohärenz der von der Erstversion von *PROVERB* erzeugten Beweise deutlich.

Allgemein besteht die Aufgabe eines Mikroplaners darin, die zu übermittelnde Information in Sätze einzuordnen und diese linguistisch zu spezifizieren. Man kann generell mehrere Verarbeitungsschritte identifizieren, die notwendig sind, um sprachlich gute Texte zu erzeugen [Hovy, 1992]. Sie lassen sich gemäß der Ebenen, auf denen sie arbeiten, unterscheiden:

Mehrsatzebene *(multi-clause level)* Einen erheblichen Einfluß auf die Qualität des Textes hat die *Aggregation.* Darunter versteht man die Techniken, die die zu äußernde Information so zusammenfaßt, daß keine Redundanz den Sprachfluß stört. Auch die *Einteilung der zu übermittelnden Information in Sätze (sentence scoping)* sowie die Festlegung der *inneren Organisation* der Sätze sind für die Qualität des Textes von großer Bedeutung.

Eine vernünftige Einfügung von *Schlüsselwörtern (cue words, clue words)*, wie beispielsweise „aber“ oder „außerdem“, kann die Kohäsion des Textes erhöhen [Reichman, 1985, Zukerman, 1991]. Sie helfen dem Leser oder Hörer, die konzeptuelle Struktur des Textes erfassen, und erhöhen dadurch die Textqualität.

Satzebene *(clause level)* Für die Kohärenz des Textes spielen neben der Makroplanung *Thematisierung* und *Fokus* eine besondere Rolle.

Das *Thema (theme)* ist ein ausgezeichneter Teil eines Satzes, der dem Leser oder Hörer anzeigt, auf welchen Sachverhalt sich der Satz bezieht. Im Englischen steht das Thema üblicherweise am Satzanfang. Der Mikroplaner sollte die Entwicklung des Themas im Text kontrollieren, um die Einführung eines neuen Diskursgegenstands anzuzeigen und seine thematische Verbindung zu vorherigen Diskursgegenständen liefern zu können.

Der *Fokus* bezeichnet den jeweiligen Diskursgegenstand eines Satzes oder Absatzes. Eine natürliche Verschiebung des Fokus durch den sinnvollen Übergang von einem Diskursgegenstand zum nächsten ist für die Qualität eines Textes sehr wichtig.

Phrasenebene *(phrase level)* und Wortebene *(lexis)* Die Notwendigkeit weiterer Entscheidungen, die der Mikroplaner treffen muß, ist unmittelbar einsichtig. Zu ihnen gehören einerseits die Lexikonauswahl und Referenzierungs- bzw. Pronominalisierungsentscheidungen, andererseits aber auch die Festlegung der Artikel für

Nomina und von Zeit, Modus und Handlungrichtung (d. h. aktiv oder passiv) sowie im Englischen auch Aspekt (d. h. *simple* oder *continuous*) für Verben.

1.4 *PROMI*: der Mikroplaner in *PROVERB*

Nachdem nun die Notwendigkeit eines Mikroplaners für *PROVERB* deutlich geworden ist, und die Anforderungen, die im allgemeinen an Mikroplaner gestellt werden, erläutert wurden, soll nun der Aufbau und die Arbeitsweise von *PROVERB*s Mikroplaner *PROMI* beleuchtet werden, der in dieser Arbeit vorgestellt wird. Seine Architektur ist in Abbildung 1.5 dargestellt.

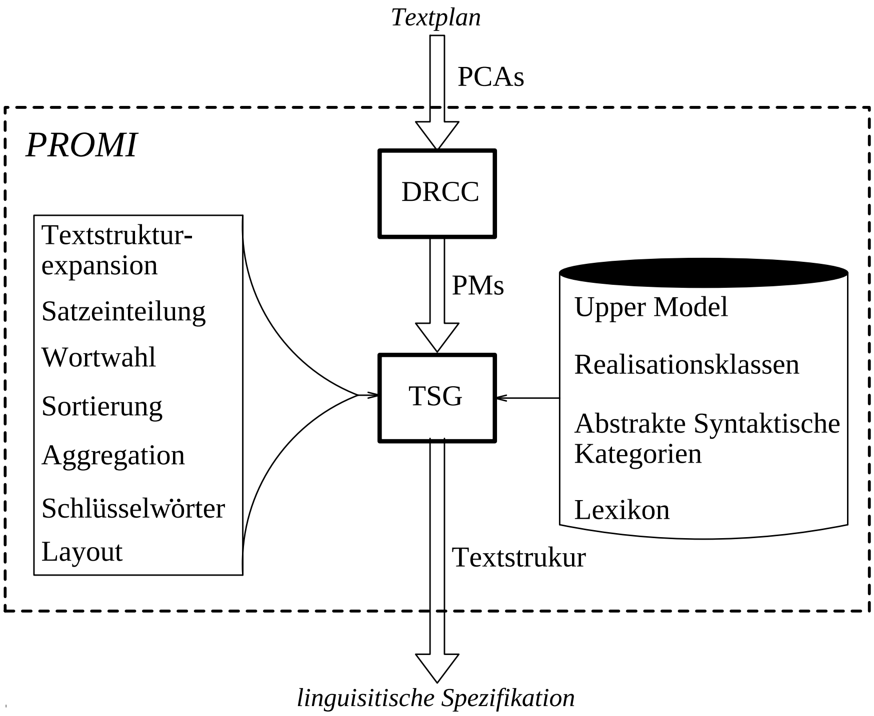


Abbildung 1.5: Die Architektur von *PROMI*

Als erste Komponente von *PROMI* erhält die Referenzierungskomponente DRCC (für *Derivation Reference Choice Component*) die von dem Makroplaner erzeugt Folge von PCAs. Sie trifft anhand einer Diskurstheorie die Entscheidung, ob und wenn ja wie sich das System auf die als Prämissen benutzten Zwischenergebnisse und die als Begründungen benutzten Inferenzmethoden beziehen soll. PCAs, in denen diese Entscheidungen vermerkt sind, heißen ***präverbale Botschaften*** (*preverbal messages*, PMs). Die Funktionsweise dieser Komponente ist in [Huang, 1994c] ausführlich beschrieben und wird in dieser Arbeit nicht weiter behandelt.

Im weiteren Verlauf erzeugt *PROMI*s Hauptmodul, der Textstukturgenerator (TSG) eine ***Textstruktur*** *(text structure)* genannte Repräsentation des Textes, die einerseits den linguistischen Beschränkungen der Zielsprache Rechnung trägt, andererseits aber von syntaktischen Details abstrahiert.

Die Textstruktur wurde von Marie Meteer als adäquate Repräsentation vorgeschlagen, um die in Kapitel 1.2 angeführte Lücke zwischen Textplan und linguistischer Spezifikation zu schließen [Meteer, 1991]. Die Textstruktur ist als Baum organisiert, in dem jeder Knoten ein Textelement oder ***Konstituente*** repräsentiert. Sie enthält Informationen über:

- die Konstituenzstruktur des Textes
- die strukturellen Relationen zwischen den Konstituenten
- die semantischen Kategorien der Konstituenten

Die semantischen Kategorien dienen in erster Linie der Typbeschränkung der Knoten der Textstruktur. Mit ihnen läßt sich definieren, wie verschiedene Textstrukturen kombiniert werden können, so daß dadurch gewährleistet werden kann, daß ausschließlich sprachlich korrekt verbalisierbare Textstrukturen erzeugt werden. Es ist so möglich, dem Textgenerierungssystem die volle Ausdrucksfähigkeit der Sprache zur Verfügung zu stellen.

In [Meteer, 1992] besteht jedoch die Hierarchie semantischer Kategorien lediglich aus 16 Kategorien, die zudem syntaktische und semantische Elemente vermischen, so daß die Ausdrucksfähigkeit nachträglich wieder eingeschränkt wird. Um dieses Problem zu beheben, teilt *PROMI* diese Hierarchie in zwei orthogonale Dimensionen auf, die auf Ansätze der Theorie der systemisch-funktionalen Linguistik zurückgehen (Kapitel 2.1). In *PROMI* repräsentieren diese Dimensionen jeweils die semantischen und syntaktischen Elemente der Typen. Eine solche Aufteilung wurde bereits von Franck Panaget vorgeschlagen [Panaget, 1994a]. Dabei dient das ***Upper Model*** [Bateman *et al.*, 1990] als sprachlich motivierte, domänenunabhängige Konzepthierarchie der Klassifizierung der semantischen Konzepte (Kapitel 2.3.1), während syntaktische Konzepte in einer ***Hierarchie abstrakter syntaktischer Kategorien*** [Panaget, 1994a] taxonomisch erfaßt wurden (Kapitel 2.4).

Der Aufbau der Textstruktur (Kapitel 3) geschieht mittels sogenannter ***Expansionsbäume*** *(resource trees)*, vordefinierten Teilbäumen der Textstruktur, die nach Instantiierung in die vorhandene Textstruktur eingebunden werden. Expansionsbäume werden zu ***Realisationsklassen*** zusammengefaßt, die dadurch die Möglichkeiten definieren, die dem Mikroplaner an einem Auswahlpunkt zur Verfügung stehen.

In *PROMI* dient die Textstruktur der zentralen Repräsentation des Textes (Kapitel 3). Sie wurde allerdings gegenüber dem Original einigen Modifikationen unterzogen:

- Panagets Ansatz folgend wurde die ursprüngliche Hierarchie semantischer Konzepte durch das Upper Model und die Hierarchie abstrakter syntaktischer Kategorien ersetzt, die beide auf die Domäne mathematischer Texte angepaßt wurden (Kapitel 2). Neben den oben genannten Vorteilen ermöglicht diese Auftrennung es insbesondere, Mikroplanungsoperatoren bezüglich einer Dimension zu definieren. So sind auf semantischen Konzepten definierte Aggregationsregeln mächtiger als solche, die auf der syntaktischen Ebene angewandt werden, während Regeln zur Satzeinteilung syntaktischer Natur sind.
- Während Meteer Realisationsklassen zu jedem Objekt des Anwendungsprogramms definiert, sind sie in *PROMI* für die Konzepte des Upper Models

definiert, so daß das System mit einer festen Wissensbasis von Realisationsklassen auskommt. Die Realisationsklassen definieren die Möglichkeiten der strukturellen Umsetzung der Upper-Model-Konzepte. Ihre Expansionsbäume enthalten in ihren inneren Knoten lexikalische Objekte. Die Typbeschränkung geschieht durch die Angabe der erlaubten Upper-Model-Konzepte und abstrakten syntaktischen Kategorien in Wurzel und Blättern. Die Definition einer Konsistenzeigenschaft garantiert die die Realisierbarkeit der durch Einsetzen von Expansionsbäumen erzeugten Textstruktur (Definition 3 in Kapitel 3.4).

- Die Textstruktur wurde dahingehend erweitert, daß auch das Layout des zu generierenden Textes in ihr repräsentiert werden kann (Kapitel 5.5). Dies geschieht einerseits durch spezielle Layout-Knoten, andererseits aber auch durch Layout-Information in den Knoten der Expansionsbäume. Das Layout ermöglicht die optische Übermittelung der globalen Struktur der Texte.

Die Arbeitsweise des TSGs basiert auf einer schrittweisen Verfeinerung der Repräsentation auf verschiedenen Ebenen, die in Abbildung 1.6 dargestellt sind. Der vollständige Algorithmus wird in Kapitel 6 vorgestellt.

$$\begin{array}{c}\text{Anwendungsprogrammobjekte}\\ \Downarrow \\ \text{Upper-Model-Objekte}\\ \Downarrow \\ \text{Textstruktur}\\ \Downarrow \\ \text{Worte}\end{array}$$

Abbildung 1.6: Die Ebenen der Repräsentation in *PROMI*

Die Folge der PMs, die die Referenzierungskomponente dem Textstrukturgenerator übergibt, stellt ein Anwendungsprogrammobjekt dar, das zunächst in ein Upper-Model-Objekt (UMO) übersetzt wird, das die Konstituente des einzigen Knotens der initialen Textstruktur repräsentiert. Dieses Upper-Model-Objekt wird dann rekursiv verarbeitet. Zunächst wird es durch die Anwendung von Aggregationsregeln umstrukturiert (Kapitel 4; siehe auch [Fiedler und Huang, 1995]), bevor es anhand seiner Realisationsklasse die Textstruktur expandiert. Das UMO zerfällt dabei in kleinere Upper-Model-Objekte, die sich in den Blättern der Textstruktur wiederfinden. Diese werden dann rekursiv behandelt, so daß schließlich eine Textstruktur entsteht, die nur noch lexikalische Objekte enthält und nicht weiter expandiert werden kann.

In einer weiteren Traversierung wird diese Textstruktur durch Einfügen von zusätzlichen Knoten, die Schlüsselwörter repräsentieren, und durch eine die Einteilung der Sätze widerspiegelnde Reorganisation umstrukturiert (Kapitel 5.2 und 5.1).

Da die Sprache der mathematischen Beweise weitgehend standardisiert und von eingeschränkter syntaktischer Struktur ist, ist eine gesonderte Behandlung von Thematisierung und Fokus ein untergeordnetes Problem und wird in *PROMI* nicht berücksichtigt. Die Pronominalisierung spielt in mathematischen Texten ebenfalls nur eine geringe Rolle, denn die jeweiligen Objekte werden meist mit einem Bezeichner versehen und über diese referenziert. Daher wird auch Pronominalisierung hier nicht näher untersucht.

Während der Umstrukturierung werden für die lexikalischen Objekte konkrete Worte anhand einer Heuristik ausgewählt (Kapitel 5.3) und die in Kapitel 1.3 angesprochenen Entscheidungen auf der Phrasenebene getroffen (Kapitel 5.4). Die Festlegung von Zeit, Modus, Handlungsrichtung und Aspekt geschieht hier nach im vorhinein festgelegten Entscheidungen, während definite und indefinite Artikel durch einfache Regeln ausgewählt werden.

Nach Abschluß der Traversierung stellt die Textstruktur eine vollständige linguistische Spezifikation des Textes dar, die abschließend von TAG-GEN realisiert wird.

Beispiel:
Das Beispiel aus Kapitel 1.3 wird von der um *PROMI* erweiterten Version von *PROVERB* wie folgt verbalisiert. (Die Zahlen zu Beginn jedes Satzes im Beweis entsprechen den PCAs im oben genannten Beispiel.)

Theorem:

Let F be a group, U be a subgroup of F and 1 and 1_U be unit elements of F and U respectively. Then $1_U = 1$.

Proof:

(1) Let F be a group, U be a subgroup of F and 1 and 1_U be unit elements of F and U respectively.

(2,3) $1_U \in U$ by the definition of unit element which leads to the existence of an x such that $x \in U$. (4) Let u_1 be such an x.

(5) $u_1 * 1_U = u_1$ by the definition of unit element. (6,7) Since U is a subgroup of F $U \subset F$ which implies $u_1 \in F$. (8) Similarly since $1_U \in U$ $1_U \in F$ (9) Since F is a group F is a semigroup. (10) Since $u_1 * 1_U = u_1$ 1_U is a solution of the equation $u_1 * X = u_1$.

(11,12) Since 1 is a unit element of F $u_1 * 1 = u_1$ and $1 \in F$. (13) Since $u_1 \in F$ 1 is a solution of the equation $u_1 * x = u_1$. (14) Since F is a group $1_U = 1$ by the uniqueness of the solution. (15) This conclusion is independent of the choice of the element u_1. ■

Durch eine kleine Erweiterung des Makroplaners von *PROVERB* wird der zu beweisende Satz in den Textplan aufgenommen, so daß auch dessen linguistische Spezifikation von *PROMI* vorgenommen wird. *PROMI* benutzt die attentionale Struktur des Textplans zur Gestaltung des Layouts, indem er Sätze und Beweise als solche kennzeichnet und neue attentionale Einheiten in Absätze gliedert. Das Ende des Beweises markiert er durch „■“. Layoutentscheidungen führten ebenfalls zu der Benutzung mathematischer Symbole.

Die Anwendung von Aggregationsregeln führte zu der Verkettung von PCAs wie in (2,3) und (6,7) oder zu ihrer Zusammenfassung wie in (11,12). In (1) wie auch in der Voraussetzung des Theorems wurde zusätzlich die Formeln $unit(1, F)$ und $unit(1_U, U)$ innerhalb des Satzes aggregiert und mit dem Schlüsselwort “respectively” versehen.

PROMI wurde zunächst auf die Generierung englische Texte ausgelegt. Seine

Architektur ist jedoch prinzipiell geeignet, eine mehrsprachige Ausgabe zu erzeugen, wie in [Huang und Fiedler, 1995] näher dargelegt wurde. Er ist bereits voll implementiert und wird zur Zeit noch getestet. Die Implementation erfolgte in Common Lisp und CLOS mit Hilfe der Programm- und Datenstrukturbibliothek zur Automatischen Deduktion **KEIM** [Nesmith, 1994]. In dieser Arbeit wird lediglich der deklarative Teil der Implementation dargestellt (Anhang A und B).

Zusammenfassung

Kurz zusammengefaßt besteht der wichtigste wissenschaftlicher Beitrag dieser Arbeit in:

- der Integration der Textstruktur mit Ansätzen der systemisch-funktionalen Linguistik in Form des Upper Models und der Hierarchie abstrakter syntaktischer Kategorien
- der Abbildung der Objekte des Applikationsprogramms über Upper-Model-Objekte in die Textstruktur
- der Spezifizierung von Aggregationsregel auf Upper-Model-Objekten zur Vermeidung von Redundanz in mathematischen Texten
- der Erweiterung der Textstruktur zur Planung des Layouts

Kapitel 2

Systemische Ansätze in *PROMI*

Die Theorie der systemisch-funktionalen Linguistik (SFL) bildet die Grundlage für das Upper Model und die Hierarchie der abstrakten syntaktischen Kategorien. Sie stammt in ihren Ursprüngen aus den anthropologisch und soziologisch orientierten Denkschulen, wie sie insbesondere von B. Malinowski und J. R. Firth vertreten wurden, und aus der europäischen funktionalen Linguistik, hier vor allem aus der Prager Schule. Die SFL wurde in den sechziger Jahren vor allem von M. A. K. Halliday in England entwickelt [Halliday, 1994].

Nach [Matthiesen und Bateman, 1990] ist die SFL als holistische Theorie, die Sprache im Kontext betrachtet, besonders gut für die Generierung natürlicher Sprache geeignet. Sie sieht Sprache als funktional an und betrachtet sie als eine Ressource, die es dem Sprecher erlaubt, seine kommunikativen Ziele zu erreichen.

2.1 Konzepte der systemisch-funktionalen Linguistik

Das wichtigste Merkmal der SFL ist die Auswahl zwischen minimalen grammatischen Alternativen. Sie repräsentiert Sprache als Netzwerk miteinander verbundener Knoten, den sogenannten ***Systemen*** *(systems)*, die die Möglichkeiten einer bedeutungsherstellenden Auswahl repräsentieren. In der KI würde man dieses Netzwerk als Entscheidungsbaum charakterisieren. Man nennt diese Sichtweise eine *paradigmatische* Orientierung. Im Gegensatz dazu steht eine *syntagmatische* Orientierung, die die Sprache als eine Menge von Strukturen versteht. Die Unterordung der syntagmatischen Orientierung unter die paradigmatische gewährleistet, daß die Grammatik dazu dienen kann, die Bedeutung auszudrücken [Bateman, 1992].

Die SFL ist also demnach eine Theorie, die die Sprache als Ressource auffaßt, die dazu dient, Bedeutung herzustellen. Dabei repräsentiert jedes System einen Auswahlpunkt, der Vorbedingung, Optionen und Realisation (d. h. die strukturellen Konsequenzen der Auswahl) enthält. Das Netzwerk definiert eine partielle Ordnung von Systemen vom allgemeinen zum spezifischen, wobei die Durchquerung auf mehreren unabhängigen Dimensionen parallel voranschreiten kann. Man beginnt dabei in der allgemeinsten Ebene und schreitet fort bis zur spezifischsten, indem man bei jedem Schritt eine Auswahl trifft.

Man unterscheidet in der SFL allgemein drei Schichten oder ***Strata***:

1. die *Semantik*: die Ebene der Bedeutung

2. die *Lexikogrammatik (lexicogrammar)*: die Wortebene; die Lexikogrammatik besteht aus Grammatik (Syntax und Morphologie) und Lexis (Vokabular)

3. die *Phonologie* bzw. *Orthographie*: die Ebene der Aussprache bzw. Schreibweise

Orthogonal zu den Strata unterteilt man die Funktionalität der Sprache in drei ***Metafunktionen***:

1. die *ideationale* Metafunktion: sie repräsentiert die Bedeutung, also die Semantik

2. die *interpersonale* Metafunktion: sie entspricht der Funktion bzgl. der sozialen symbolischen Interaktion zwischen dem Sprecher und dem Hörer

3. die *textuelle* Metafunktion: sie entspricht der Organisation der Sprache, d. h. der Presentation der Information als Text im Kontext

In *PROMI* wird die ideationale Metafunktion durch die Konzepthierarchie des Upper Models repräsentiert (Kapitel 2.3.1), während die textuelle Metafunktion in vereinfachter Form als Hierarchie abstrakter syntaktischer Kategorien dargestellt wird (Kapitel 2.4). Die interpersonale Metafunktion ist in *PROMI* nur stark simplifiziert enthalten (Kapitel 5.4), da ihre Aufgabe in der eingeschränkten Domäne mathematischer Texte durch im vorhinein feststehende Entscheidungen abgedeckt werden können. Benötigte man die interpersonale Metafunktion in einer mächtigeren Form, so würde man sie in der KI durch ein Benutzermodell darstellen.

In dem folgenden Abschnitt wird eine Implementation der SFL vorgestellt, die die Eignung der SFL zur Sprachgenerierung erfolgreich unter Beweis stellen konnte.

2.2 Penman

Eines der bekanntesten auf der Theorie der systemisch-funktionalen Linguistik basierenden Systeme ist Penman [PENMAN, 1989], ein domänenunabhängiger Textgenerator, der am Information Science Institute der Universität von Südkalifornien in Los Angeles entwickelt wurde. Es ist ein sehr erfolgreiches System, daß bis heute noch in zahlreichen Anwendungen eingesetzt wird, wie z. B. im System KOMET zur Generierung multilingualer deskriptiver Texte [Bateman, 1994] oder in Drafter, einem System zur automatischen Erzeugung von Handbüchern für Softwareprodukte [Paris *et al.*, 1995]. Seine Architektur ist in Abbildung 2.1 dargestellt.

Penman ist einen syntaktischer Realisierer, der als Eingabe einen Textplan erwartet. Den Kern von Penman bildet die Komponente *Nigel*, die, da ein Mikroplaner in dieser Architektur nicht vorgesehen ist, neben der Realisierung auch die Aufgaben der Satzplanung zu einem großen Teil übernimmt. Sie operiert in dem sogenannten *Environment*, in dem sich Wissensbasen, ein Benutzermodell, ein Lexikon und ähnliche Wissensquellen befinden. Nigel selbst besteht wiederum aus zwei Komponenten [Matthiesen, 1990]. Die Hauptkomponente ist eine sogenannte Lexikogrammatik *(lexicogrammar)* — oder kurz: Grammatik — im Sinne der

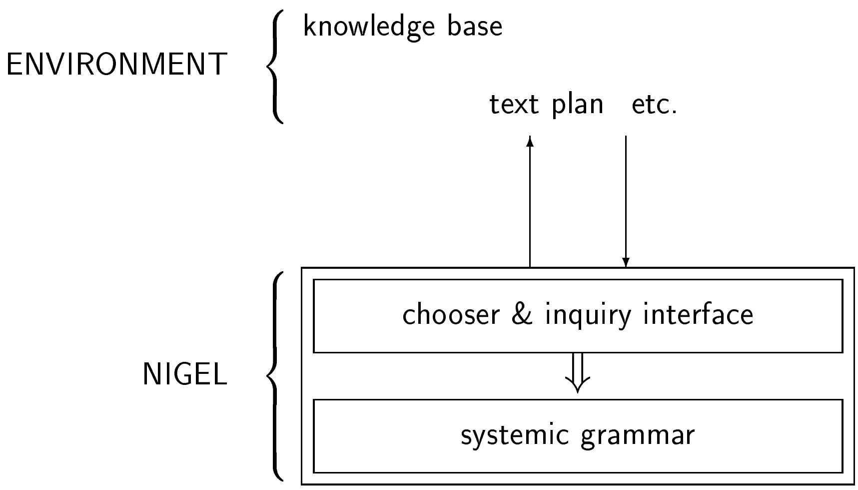


Abbildung 2.1: Die Architektur von Penman (aus [Matthiesen und Bateman, 1991])

systemisch-funktionalen Linguistik. Zur Realisierung wird einem Pfad durch die Grammatik gefolgt wird, wobei an jedem System eine minimale grammatische Entscheidung zu fällen ist. Die Steuerung der Satzplanung geschieht also durch die Grammatik. Die zweite Komponente besteht aus einer semantischen Schnittstelle, die die Grammatik mit dem *Environment* verbindet. Da sie Anfragen an das *Environment* stellt, das das zur Auswahl an den Systemen notwendige Wissen liefert, heißt sie *Chooser and Inquiry Interface*.

Eine Wissensbasis des *Environments* ist das Upper Model. Es handelt sich bei ihm um eine Realisierung der ideationalen Metafunktion. Der nächste Abschnitt stellt das Upper Model in *PROMI* und seine Anwendung in der Mikroplanung vor.

2.3 Das Upper Model

Eine der Wissensbasen, die in Penmans *Environment* enthalten sind, ist das ***Upper Model*** [Bateman *et al.*, 1990]. Es organisiert Bedeutung auf der konzeptuellen Ebene in Abhängigkeit davon, wie sie ausgedrückt wird, d. h. verschiedene Konzepte im Upper Model unterscheiden sich in ihrer sprachlichen Realisierung. Es repräsentiert daher die ideationale Metafunktion.

Die Organisation erfolgt in einer Vererbungshierarchie, in der Nachkommen innerhalb der Hierarchie Eigenschaften von ihren Ahnen erben. Daher ist die linguistische Realisierung eines Konzepts des Upper Models von seiner Position innerhalb der Hierarchie abhängig.

Upper-Model-Konzepte (UMKe) können Argumente haben, die semantischen Rollen entsprechen. Der Wert eines Arguments ist dabei auf bestimmte Konzepttypen eingeschränkt. Beispielsweise hat das Konzept *ascription*[1] zwei Rollen, nämlich `:attribute` und `:attribuend`. Dabei ist erstere auf die Konzepte *quality* oder *object* eingeschränkt, während letztere auf das Konzept *object* festgelegt ist:

[1]UMKe werden als in dieser Arbeit *schräggestellt* geschrieben

```
(def-concept  ascription (intensive)
  (:attribute  quality object)
  (:attribuend  object))
```

Beispielsweise ist in der Upper-Model-Repräsentation von „F is a set“ „F“ der Attribuend und „set“ das Attribut, wobei beide dem UMK *object* angehören. Der unmittelbare Vorfahr von *ascription* ist das UMK *intensive*.

Penmans Upper Model ist also eine eine ausgetüftelte semantische Taxonomie von Objekten und Handlungen bezüglich ihrer Verbalisierung, die vollständig von der Syntax abstrahiert. Es ist in [Bateman *et al.*, 1990] ausführlich beschrieben. Im folgenden wird das Upper Model in *PROMI* und seine Anwendung bei der Satzplanung genauer dargestellt.

2.3.1 *PROMI*s Upper Model

Das Upper Model in *PROMI* ist in Abbildung 2.2 dargestellt. In ihm finden sich einige Abweichungen gegenüber dem Original. Zunächst ist für *PROMI* durch die Einschränkung auf die Domäne mathematischer Texte der Detailreichtum des Originals, wie beispielsweise in den Teilhierarchien von *object* und *quality*, nicht erforderlich.

Ferner wird in Penmans Upper Model *relational-process* in *one-place-relation* und *two-place-relation* unterteilt, die Prozesse repräsentieren, die man als ein- bzw. zweistellige Relationen interpretieren kann. Mehrstellige Relationen lassen sich dann als Verschachtelung zweistelliger Relationen darstellen. Um eine mächtigere Aggregation (vgl. Kapitel 4) zu erlauben, ist es jedoch wünschenswert auch beliebigstellige Relationen zuzulassen, beispielsweise für Konjunktionen und Disjunktionen. Daher wird *relational-process* im *PROMI*s Upper Model in *discrete-place-relation* und *arbitrary-place-relation* unterteilt.

Eine besondere Rolle kommt bei der Planung mathematischer Texte den Konzepten *derive* und *derive-cont* zu.[2] Ihre Definitionen sind identisch:

```
(def-concept  derive (cause-relation)
  (:reason  reason process)
  (:conclusion  process)
  (:method  enablement)
  (:next  derive-cont))

(def-concept  derive-cont (cause-relation)
  (:reason  reason process)
  (:conclusion  process)
  (:method  enablement)
  (:next  derive-cont))
```

[2]Beide Konzepte werden im originalen Upper Model durch das Konzept *reason* umfaßt, das nicht mit dem gleichnamigen Konzept in *PROMI*s Upper Model verwechselt werden sollte.

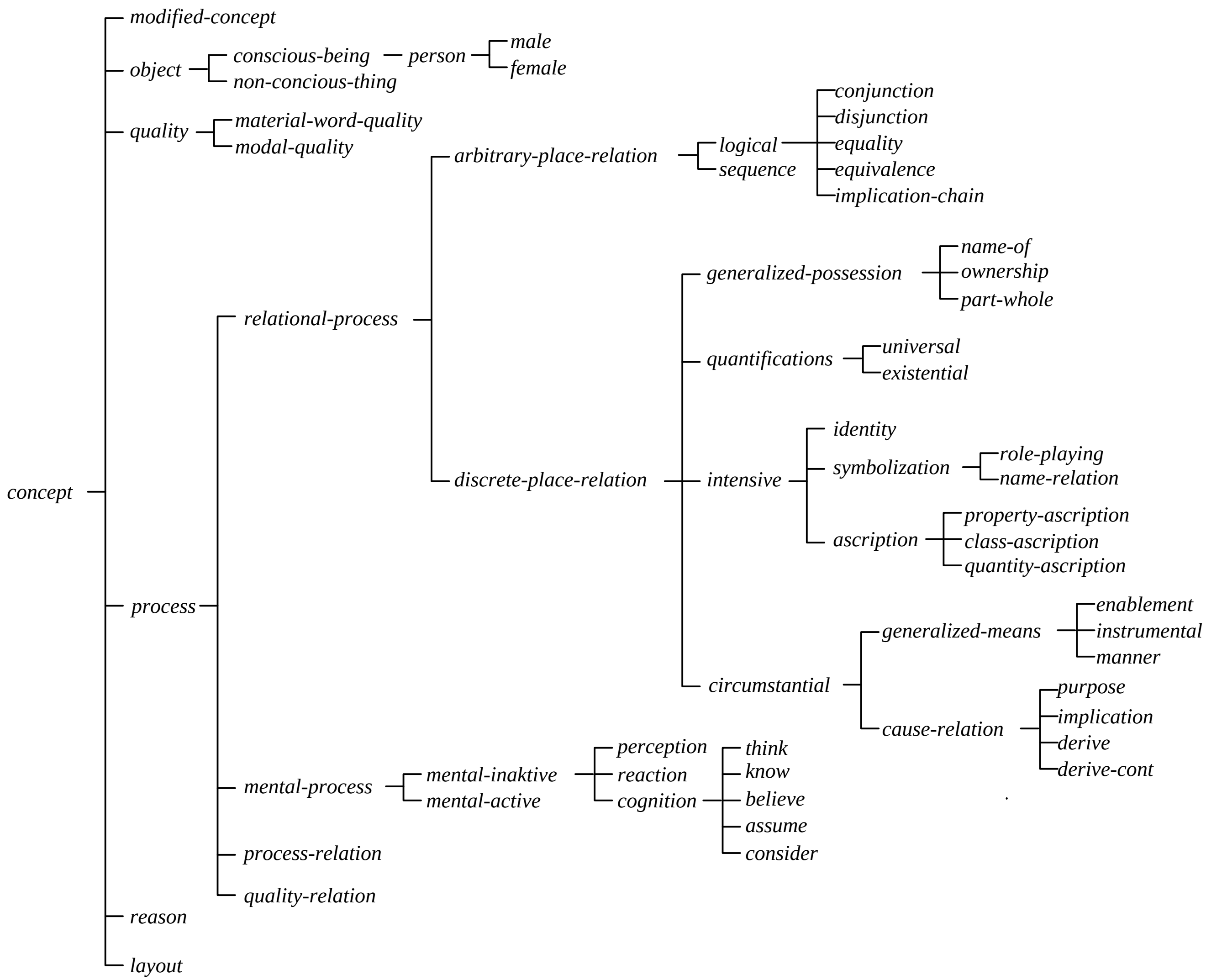


Abbildung 2.2: Das Upper Model

Sie unterscheiden sich jedoch in ihrer Semantik und daher auch in ihrer sprachlichen Realisierung. Während *derive* stets den Anfang einer Ableitungskette bildet, steht *derive-cont* stets innerhalb einer solchen. Dieser Sachverhalt spiegelt sich in der Einschränkung der Rolle `:next`, die den nächsten Ableitungsschritt in einer Kette spezifiziert, auf *derive-cont* wider. In der Upper-Model-Repräsentation des Satzes "Since U is a subgroup of F, we have $U \subset F$, which implies $u_1 \in F$." wird "Since U is a subgroup of F, we have $U \subset F$" durch *derive* repräsentiert, während, "which implies $u_1 \in F$" durch *derive-cont* dargestellt wird. Kapitel 4.2.1 kommt auf diese beiden Konzepte und ihre Unterscheidung zurück.

In dem Upper Model von *PROMI* steht *reason* für mit "since" oder "because" eingeleitete Nebensätze und kennzeichnet die Prämissen einer Ableitungskette, wie in obigem Beispiel "since U is a subgroup of F".

Schließlich wurden aus Gründen der Vereinfachung zum Upper Model von *PROMI* noch die Konzepte *modified-concept*, *process-relation* und *quality-relation* zusätzlich aufgenommen. Das ebenfalls im Original nicht verhandene Konzept *layout* dient der Repräsentation von Layoutdirektiven und erlaubt dadurch die Planung des Layouts durch *PROMI*.

2.3.2 Upper-Model-Objekte und Upper-Model-Variablen

Ein ***Upper-Model-Objekt*** ist eine Instanz eines Konzeptes des Upper Models, dessen Argumente wiederum mit solchen Instanzen besetzt sind. Da die Instantiierung inkrementell erfolgt, sind dabei auch Konzeptvariablen, sogenannte ***Upper-Model-Variablen*** erlaubt.

Definition 1 Eine Variable über Konzepten des Upper Models heißt *Upper-Model-Variable* (UMV).
Notation: UMVn werden *kursiv* geschrieben. ■

Definition 2 Ein *Upper-Model-Objekt* (UMO) wird wie folgt induktiv definiert:

(i) Die Instanz eines nullstelligen Konzeptes des Upper Models ist ein UMO.

(ii) Eine Upper-Model-Variable ist ein UMO.

(iii) Die Instanz eines mehrstelligen Konzeptes oder eine UMV, dessen Argumente durch UMOe, durch Listen von UMOen oder durch lexikale Einträge besetzt sind, ist ein UMO.

UMOe, in denen keine UMVn mehr auftreten, heißen *variablenfrei*.
Notation: UMOe werden *schräggestellt* geschrieben. ■

Beispiel: *conjunction*$\langle($*symm*$\langle$*sigma*$\rangle$, *symm*$\langle$*rho*$\rangle)\rangle$ ist ein UMO ■

Konzepte des Anwendungsprogramms müssen zu ihrer Verbalisierung zunächst auf Upper-Model-Konzepte abgebildet werden. Da ein Anwendungsprogrammkonzept möglicherweise auf mehrere Upper-Model-Konzepte abgebildet werden kann,

kann man jedes Anwendungsprogrammobjekt als eine Variable über Upper-Model-Objekten interpretieren. Daher werden die erlaubten Abbildungen zwischen Anwendungsprogrammkonzepten und Upper-Model-Konzepten durch die Definition der Upper-Model-Variablen spezifiziert. Im Gegensatz zu Anwendungsprogrammobjekten besitzen Upper-Model-Variablen keine feste Stelligkeit. Die Argumente einer Upper-Model-Variablen sind ebenfalls Upper-Model-Objekte (also möglicherweise auch selbst Upper-Model-Variablen). Die Notwendigkeit der Upper-Model-Variablen wird in Kapitel 4 deutlich werden. Dort werden Aggregationsregeln zur Manipulation von Upper-Model-Variablen eingeführt, deren Definition auf Anwendungsprogrammobjekten in dieser Prägnanz und Mächtigkeit nicht erlaubt werden könnte.

2.3.3 Die Abbildung des Textplans in das Upper Model

Zur Satzplanung muß der Textplan in das Upper Model abgebildet werden. Dazu muß zu jedem Konzept des Anwendungsprogramms, d. h. also zu jeder Funktion und zu jedem Prädikat des Beweises, sowie zu jedem logischen Junktoren und Quantoren und zu jeder Klasse von PCAs, genau eine UMV mit ihren Belegungsmöglichkeiten, denen Prioritäten zugeordnet werden können, angegeben werden. Eine Priorität wird als ein Paar (v, n) angegeben, wobei $v \in \{\texttt{verb}, \texttt{sym}\}$ und $n \in \mathbb{N}$. Dabei bedeutet `verb`, daß eine verbalisierte Äußerung bevorzugt werden sollt, während `sym` eine Präferenz zur Realisierung anhand mathematischer Symbole anzeigt. n definiert die Häufigkeit, mit der eine Belegungsmöglichkeit im Vergleich zu konkurrierenden Belegungsmöglichkeiten ausgewählt werden sollten. Die Prioritäten werden von einer Auswahlheuristik zur Entscheidung herangezogen.

Zum Beispiel wird die UMV zu der Funktion `symm`, die für die Symmetrieeigenschaft von Relationen steht, wie folgt spezifiziert:

```
(def-variable  symm
  ((arity 0) (priority (verb . 10))
   (um  object :object (lex 'symm)))
  ((arity 0) (priority (verb . 10))
   (um  quality :quality (lex 'symm)))
  ((arity 1) (priority (verb . 10))
   (um  property-ascription :attribute (umv  symm)
                            :attribuend ?1)))
```

`Arity` bezeichnet dabei die Stelligkeit der Variablen, `?i` steht für das i-te Argument der Variable. Das Schlüsselwort `um` zeigt an, daß es sich bei dem folgenden Ausdruck um ein UMO handelt, während `umv` anzeigt, daß der nachfolgende Ausdruck eine Upper-Model-Variable darstellt, in diesem Falle eine nullstellige, da keine Argumente spezifiziert sind. Das Schlüsselwort `lex` bezeichnet einen Lexikoneintrag. `Priority` kennzeichnet die Priorität. Ist keine Priorität angeben, wird eine voreingestellte Priorität eingesetzt.

Der obige Ausdruck definiert eine Upper-Model-Variable, die im nullstelligen Fall mit den UMOen *object* oder *quality* belegt werden kann. Ersteres würde als "symmetry" verbalisiert, letzteres als "symmetric". Im einstelligen Fall wird die

Variable mit einem UMO des Konzeptes *property-ascription* belegt, das als Argumente die nullstellige UMV *symm* sowie das Argument der Variablen erhält. Dieses UMO entspricht dem oben angeführten nullstelligen Fall und könnte daher als "`?1` is symmetric" oder "`?1` is a symmetry" verbalisiert werden; die Einschränkung des Attribuenden von *property-ascription* auf *quality* garantiert die erste Verbalisierung.

Im folgenden wird nur dann zwischen Upper-Model-Variablen und Upper-Model-Konzepten unterschieden, wenn zu der Upper-Model-Variablen verschiedene Belegungsmöglichkeiten existieren.

Die Definitionen der UMVn für die PCA-Klassen und die logischen Junktoren und Quantoren sowie die formale Definition der Eingabesprache befindet sich im Anhang A. Die UMVn für Funktionen und Prädikate müssen zu jedem mathematischen Gebiet eigens angegeben werden.

Die Abbildung eines Anwendungsprogrammobjektes in ein Upper-Model-Objekt geschieht kanonisch. Es wird zunächst in ein UMO abgebildet, das ausschließlich aus Upper-Model-Variablen besteht. Zu den UMVn werden später sukzessive Upper-Model-Konzepte ausgewählt, durch die sie im Upper-Model-Objekt ersetzt werden. Bei dieser Auswahl ist zu beachten, daß einerseits die Typbeschränkungen der Argumente eingehalten werden. Andererseits soll auch der Grundsatz des ***casting*** eingehalten werden. Dalianis und Hovy bezeichnen mit *casting* das Prinzip, daß innerhalb eines Kontextes semantisch ähnliche Information durch gleiche syntaktische Konstrukte ausgedrückt werden [Dalianis und Hovy, 1993]. Dieses Prinzip wird bei der Abbildung in das Upper Model nur eingeschränkt angewandt. Konkret werden Argumente von Konjunktionen, Disjunktionen, Äquivalenzen und Implikationen in gleiche Upper-Model-Konzepte abgebildet. Ist dies nicht möglich und kommen mehrere Upper-Model-Konzepte in Frage, so bestimmt eine Heuristik H die Auswahl. Diese wird in Kapitel 5.3 vorgestellt.

2.4 Die Hierarchie der abstrakten syntaktischen Kategorien in *PROMI*

Die Hierarchie der abstrakten syntaktischen Kategorien stammt ursprünglich von Panaget [Panaget, 1994a][3] und wurde hier für *PROVERB* angepaßt. Man kann diese Hierarchie als die textuelle Metafunktion der SFL interpretieren, die jedoch mehr Gewicht auf syntaktische Anteile legt. Sie ist ebenfalls als Vererbungshierarchie konzipiert und bildet eine Taxonomie abstrakter syntaktischer Kategorien (ASKn[4]), die dazu dient, die syntaktische Struktur des Textes zu spezifizieren. Die originale Hierarchie der ASK ist in [Panaget, 1994a] ausführlich beschrieben.

Abbildung 2.3 zeigt die Hierarchie der ASK in *PROMI*. Sie ist in Teilen weniger detailliert als das Original, besitzt aber auch einige Kategorien die dort nicht vorkommen. So wurden beispielsweise Informationen über Numerus, Genus und Person aus der Teilhierarchie der Nominalphrasen (np) eliminiert. Dies ist möglich,

[3]Er gab sie später auf zugunsten einer anderen Hierarchie, die er in Anlehnung an die SFL Hierarchie der textuell-semantischen Kategorien nannte [Panaget, 1994b]. Da es sich hier aber anscheinend nur um eine Umbenennung der Kategorien handelt, anstatt um ein anderes Konzept, bleibe ich bei seiner ursprünglichen Version.

[4]ASKn werden in dieser Arbeit ohne Serifen geschrieben.

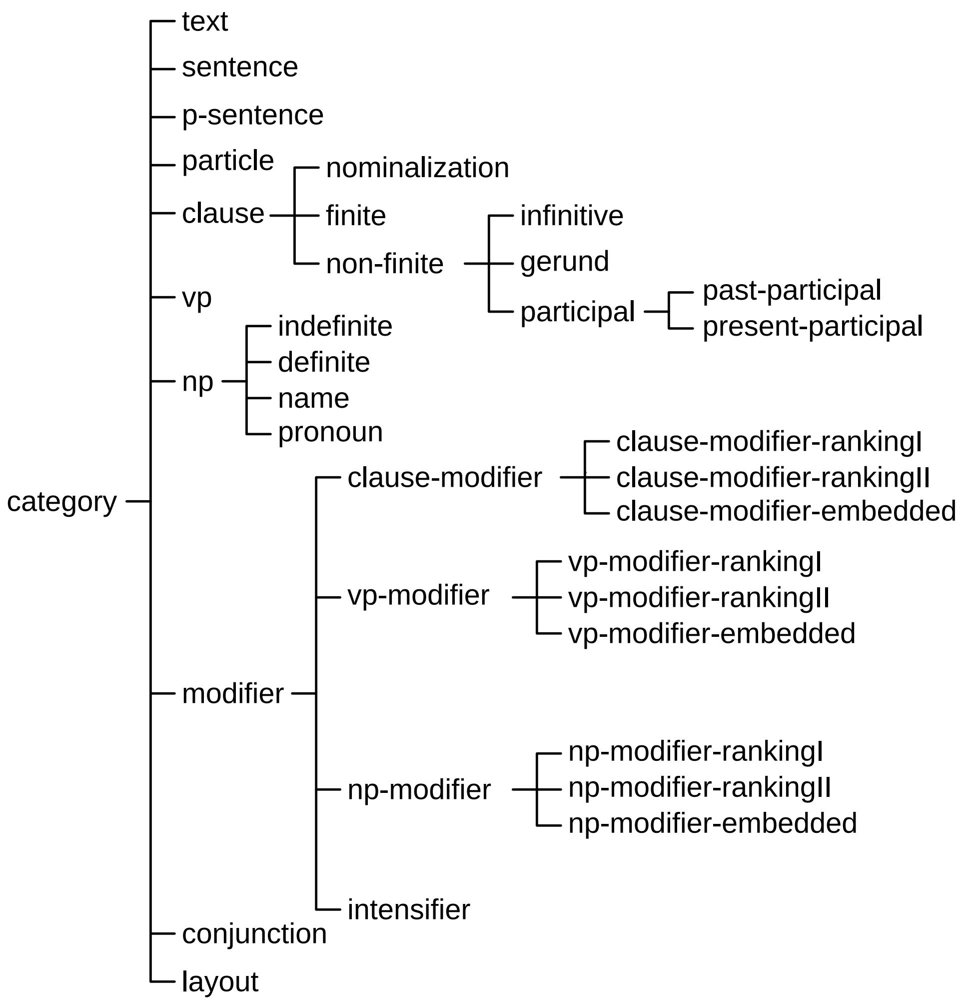


Abbildung 2.3: Die Hierarchie der abstrakten syntaktischen Kategorien

da Genus und Person in den entsprechenden lexikalen Objekten bereits enthalten sind, während der Numerus jeweils aus dem Kontext berechnet wird. Ebenso wurden Informationen über Modus und Handlungsrichtung aus der Teilhierarchie der Verbphrasen (vp) entfernt, die ohnehin eher der interpersonalen Metafunktion zuzurechnen sind. Ihre Berechnung wird in Kapitel 5.4 dargestellt.

Die Kategorien text, sentence und clause stehen für den gesamten Text, einzelne Sätze bzw. Klauseln. , während particle für Partikel, die Sätze In *PROMI* wurde in die Hierarchie die Kategorien p-sentence, conjunction und layout zusätzlich aufgenommen. P-sentence bezeichnet Sätze die durch Partikel (particle) erweitert werden können. Während conjunction für die Konjunktion beliebiger ASKn steht, dient layout der Repräsentation von Layoutdirektiven.

Modifier bezeichnen Kategorien, die andere Kategorien modifizieren. So modifizieren clause-modifier, vp-modifier und np-modifier Sätze, Verbphrasen bzw. Nominalphrasen. Sie zerfallen jeweils in drei Subkategorien rankingI, rankingII und embedded. Die rankingI-Kategorien stehen bei clause-modifier und vp-modifier für Adverbien, bei np-modifier für Adjektive. RankingII-Kategorien stehen stets für Präprositionalphrasen, während embedded bei clause-modifier und vp-modifier für Adverbialsätze steht, bei np-modifier für Relativsätze. Intensifier, schließlich, modifizieren andere modifier. Beispielsweise ist in "the very tall man" "very" ein intensifier, der den np-modifier-rankingI "tall" modifiziert.

Nachdem wir nun mit dem Upper Model und der Hierarchie abstrakter syntaktischer Kategorien vertraut sind, werden wir im nächsten Kapitel sehen, wie diese beiden Taxonomien dazu dienen, *PROMI* auf die Erzeugung verbalisierbarer Textstrukuren einzuschränken.

Kapitel 3

Die Textstruktur

Die ***Textstruktur*** wurde von Marie Meteer vorgeschlagen, um die in Kapitel 1 erläuterte „generation gap“, die Lücke zwischen der Textplanung und der linguistischen Realisierung zu schließen [Meteer, 1991]. Sie implementierte die Textstruktur in dem System Spokesman, das im nächsten Abschnitt kurz vorgestellt wird, bevor die Textstruktur ausführlich besprochen wird.

3.1 Spokesman

In dem von Meteer verfolgten Ansatz ist das Textgenerierungssystem Spokesman unabhängig von dem jeweiligen Anwendungsprogramm [Meteer, 1992]. Es erhält vielmehr von diesem Objekte, die sogenannten ***Anwendungsprogrammobjekte*** (*application program objects*, APOe), die anhand englischer Sätze verbalisiert werden sollen. Spokesman besitzt zwar die bekannte Zweiteilung in Textplaner und Realisierer, der Textplaner beinhaltet aber sowohl einen Makroplaner, als auch einen Mikroplaner. Der Textplaner wählt die auszudrückende Information aus und legt ihre rhetorische Organisation fest, die dann auf linguistische Ressourcen abgebildet wird. Der linguistische Realisierer MUMBLE-86 [Meteer *et al.*, 1987] generiert anschließend aus der erzeugten Repräsentation die Sätze, wobei er syntaktische und morphologische Entscheidungen treffen muß. Daber stellt er sicher, daß die Sätze grammatikalisch korrekt sind.

In Spokesman sind mehrere Repräsentationsebenen realisiert (siehe Abbildung 3.1), die ausgehend von einer abstrakten Repräsentation eine schrittweise Verfeinerung durch konservative Einfügung von Details erlauben, d. h. bereits getroffene Entscheidungen bleiben erhalten. Meteer betont, daß diese Vorgehensweise die sprachliche Realisierbarkeit der geplanten Äußerung sicherstellt, ohne daß ein Backtracking erfolgen muß.

Die zentrale Repräsentation ist durch die sogenannte ***Textstruktur*** *(text structure)* gegeben, die der Satzplanung dient. Die Textstruktur ist als Baum organisiert, der die Relationen zwischen den Textelementen (den sogenannten *Konstituenten*) beschreibt. Jeder Knoten des Baumes entspricht dabei einer Konstituenten und enthält Information über deren semantische Kategorie. Meteer definiert dazu eine einfache Hierarchie semantischer Kategorien, die in [Meteer, 1992] aus 16 Kategorien besteht. Die semantischen Kategorien dienen in erster Linie der Typbeschränkung

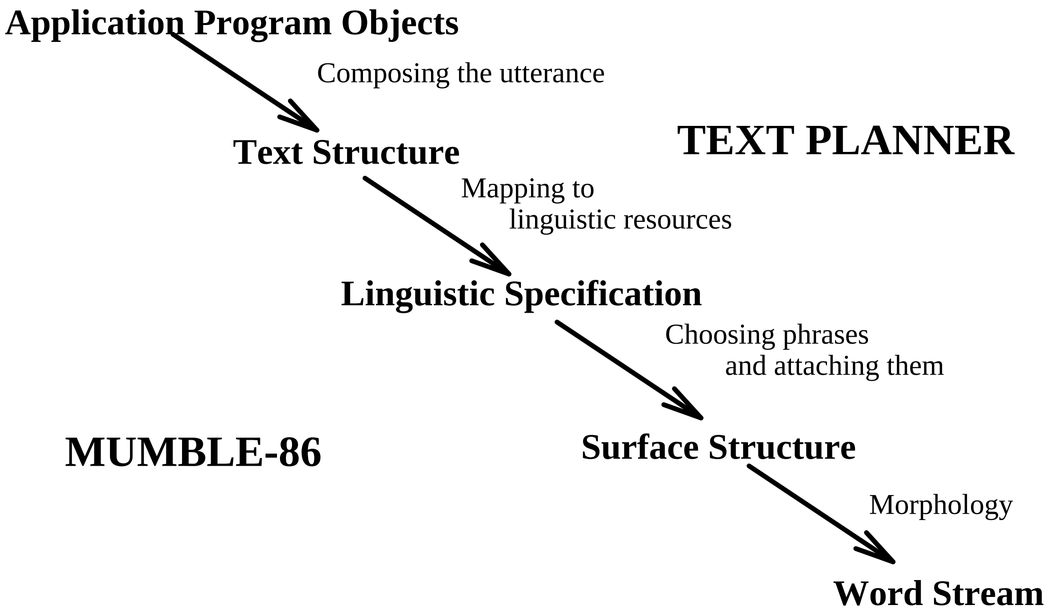


Abbildung 3.1: Die Repräsentationsebenen in Spokesman (aus [Meteer, 1992])

der Knoten der Textstruktur. Mit ihnen läßt sich definieren, wie verschiedene Textstrukturen kombiniert werden können, so daß dadurch gewährleistet werden kann, daß ausschließlich sprachlich korrekt verbalisierbare Textstrukturen erzeugt werden. Der Aufbau der Textstruktur erfolgt mittels sogenannter ***Realisationsklassen***, die den Anwendungsprogrammobjekten zugeordnet sind. Realisationsklassen sind Mengen vordefinierter Teilbäumen der Textstruktur, die nach Instantiierung in die vorhandene Textstruktur eingebunden werden. Sie sind daher als die Möglichkeiten der syntaktischen Realisierung der Anwendungsprogramobjekte zu betrachten.

3.2 Die linguistischen Ressourcen einer Sprache

Der Textplaner muß die ihm zur Verbalisierung der Information zur Verfügung stehenden Ressourcen und ihre Eigenschaften kennen, damit die Verbalisierbarkeit des von ihm konstruierten Textplans gewährleistet ist. Man kann linguistische Ressourcen in *lexikalische* und *grammatikalische Ressourcen* unterteilen. In den lexikalischen Ressourcen sind die Verben, die Nomina und die Adjektive zusammengefaßt. Die grammatikalischen Ressourcen umfassen die übrigen Wörter, die morphologischen Markierungen und die Phrasenstruktur. Nach Talmy determinieren grammatikalische Elemente eines Satzes den größten Teil der Struktur der kognitiven Repräsentation, während die lexikalischen Elemente den größten Teil ihres Inhaltes beisteuern [Talmy, 1987].

Unter den *konkreten linguistischen Ressourcen* versteht Meteer alle syntaktischen Strukturen, Wörter und grammatikalische Merkmale, die dem Sprecher einer Sprache zur Verfügung stehen [Meteer, 1991]. Sie erscheinen direkt im Wortfluß. *Abstrakte linguistische Ressourcen* definiert sie als Gruppierungen von konkreten Ressourcen, wobei manche Parameter gebunden werden, andere aber ungebunden bleiben. In ihnen werden jene konkreten Ressourcen zusammengefaßt, die auch der Sprache zur Verfügung stehen, um sie dann dem Textplaner zugänglich zu machen. Mit diesem Hilfsmittel lassen sich zum Beispiel auch Redewendungen erfassen: man „trifft eine Entscheidung“, aber man „fällt ein Urteil“. So kann man mit Hilfe der abstrakten Ressourcen dem Textplaner die volle Ausdrucksfähigkeit der Sprache zur Verfügung zu stellen. In erster Linie dienen sie aber dazu, den Textplaner auf die Generierung korrekt verbalisierbarer Äußerungen einzuschränken, d. h. die Menge der Abstraktionen sollte genau das enthalten, was in der Sprache ausdrückbar ist.

Meteer unterscheidet zwei Möglichkeiten des Auftretens konkreter Ressourcen in der Sprache: die *lineare* und die *nicht-lineare* Komposition. Mit linearer Komposition meint sie, daß jedes Element etwas kontextunabhängiges und identifizierbares beisteuert. Nicht-linear komponierende Element liefern dagegen in verschiedenen Kontexten verschiedene Dinge. Beispielsweise bedeutet „rot“ immer die Farbe rot, „Auto“ kann sich aber in verschiedenen Kontexten auf verschiedene Dinge beziehen, nämlich auf die verschiedenen konkreten Fahrzeuge.

Durch die Zusammenfassung von mehreren gleichzeitig auftretenden nicht-linearen grammatikalischen Ressourcen zu einer einzelnen abstrakten Ressource zwingt man den Textplaner dazu, nur eine solche Kombinationen auszuwählen, die zu einem in der jeweiligen Sprache ausdrückbarem Ergebnis führt. Beispielweise ist die Kombination “to make an important decision” erlaubt, während “to decide importantly” nicht korrekt ist. Dadurch, daß man dem Textplaner eine vordefinierte Menge von semantischen Kategorien zur Verfügung stellt, anstatt ihn die linguistischen Merkmale, die diese Kategorien definieren, direkt auswählen zu lassen, vermeidet man, daß er eine Kombination auswählt, die sich nicht korrekt realisieren läßt.

Meteer unterscheidet drei Arten der Abstraktion:

1. semantische Kategorien
2. Konstituenz (siehe Abschnitt 3.2.1)
3. Strukturelle Relationen zwischen Konstituenten (siehe Abschnitt 3.2.2)

Wie bereits in Kapitel 1.4 erwähnt, erscheint ihre Hierarchie semantischer Kategorien zu schwach, so daß sie hier durch das Upper Model und die Hierarchie der abstrakten syntaktischen Kategorien ersetzt wurden. Auf die Konstituenz und die strukturellen Relationen zwischen Konstituenten wird im folgenden eingegangen.

3.2.1 Konstituenz

Analysiert man einen Text, so erkennt man, daß er aus Einheiten verschiedener Größe zusammengesetzt ist. Der Text besteht aus Sätzen, die aus Wörtern bestehen, welche wiederum aus Buchstaben zusammengesetzt sind. Man kann also drei Ebenen erkennen. Was weiterhin auffällt, ist, daß die Einheiten auf einer Ebene gleichartig sind und sich niemals überlappen. Man nennt eine solche Verschachtelung ***Konstituenz*** und die Einheiten ***Konstituenten***. Jede Konstituente besteht aus einer oder mehreren kleineren Konstituenten.

Man kann mit dem Begriff der Konstituenz auch die grammatikalische Struktur erfassen, indem man weitere Ebenen einfügt, etwa für Phrasen zwischen der

Satz- und der Wortebene oder für Morpheme zwischen der Wort- und der Buchstabenebene (siehe Abbildung 3.2). Wir erhalten so eine Baumstruktur, die den

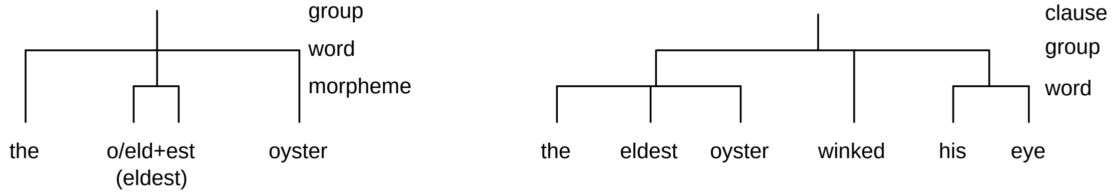


Abbildung 3.2: Zwei Beispiele für Konstituenz (aus [Halliday, 1994])

Text repräsentiert. Durch die Organisation der Textstruktur als Baum können wir die Konstituenz des Textes erfassen. Es genügt für unsere Zwecke, die Wortebene als unterste Ebene der Konstituenz zu betrachten, da die kleinsten Einheiten in der Satzplanung Wörter sind. Erst die linguistische Realisierung muß sich mit den tieferen Ebenen befassen.

3.2.2 Strukturelle Relationen zwischen Konstituenten

[Meteer, 1992] identifiziert zwei grundlegende Relationen zwischen Konstituenten in einem Text. Abbildung 3.3 zeigt diese sogenannten *Basistyp-Teilbäume*.

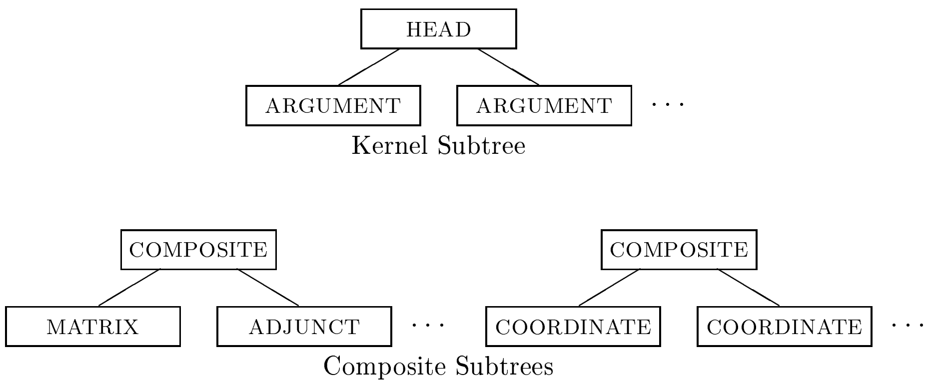


Abbildung 3.3: Basistypen von Textstrukturteilbäumen (aus [Meteer, 1992])

Der ***atomare Teilbaum*** *(kernel tree)* spiegelt die Prädikat-Argument-Struktur wieder, wie sie beispielsweise in einer Verbphrase auftritt, in der das Subjekt und die Objekte in Relation zu dem Prädikat stehen. Das Prädikat bildet den *Kopf (head)* des atomaren Teilbaums und das Subjekt und die Objekte seine *Argumente (arguments)*. Der atomare Teilbaum kann von dem Mikroplaner nur als ganzes in die Textstruktur eingesetzt werden, eine spätere Hinzufügung von weiteren Argumenten ist unzulässig. In diesem Sinne ist er atomar.

Der ***zusammengesetzte Teilbaum*** *(composite tree)* erfaßt die Relation zwischen den Konstituenten einer Ebene. Dabei unterscheidet die SFL *Parataxis* und *Hypotaxis* [Halliday, 1994]. In der Parataxis sind die Konstituenten einer Ebene gleichrangig, wie das in nebengeordnete Sätzen der Fall ist, während in der Hypotaxis ein Element die übrigen dominiert, so wie ein Nebensatz dem Hauptsatz untergeordnet ist. Ein weiteres Beispiel für Parataxis ist die Konjunkton oder Disjunktion in Nominalphrasen („Peter und Maria"). Auch Hypotaxis tritt oft in Nominalphrasen auf, wobei das Nomen die übrigen Konstituenten dominiert („das rote Auto"). Der Differenzierung in Hypotaxis und Parataxis entsprechen zwei Typen zusammengesetzter Teilbäume, nämlich der ***nebenordnende Teilbaum*** mit zwei oder mehr gleichrangigen *Koordinaten (coordinates)* und der ***unterordnende Teilbaum*** mit einer dominanten *Matrix (matrix)*, der null oder mehr *Adjunkte (adjuncts)* untergeordnet sind. Im Gegensatz zum atomaren Teilbaum darf ein zusammengesetzer Teilbaum, dessen Wurzel *Compositum (composite)* genannt wird, inkrementell aufgebaut werden, d. h. der Mikroplaner kann nachträglich weitere Koordinaten oder Adjunkte hinzufügen (vgl. Kapitel 5.2).

3.3 Die Textstruktur in *PROMI*

Marie Meteer gibt keine formale linguistische Definition der Textstruktur an. Die Rechtfertigung dafür ist, daß die zugrundeliegenden psycholinguistischen Konzepte nur grobe Annäherungen sind, eine formale Definition aber eine Erklärung der Konzepte vortäuschen würde. Daher wird an dieser Stelle auch nur die Definition der Textstruktur im System *PROVERB* angegeben.

Die Textstruktur ist in *PROVERB* als Baum organisiert, in dem jeder Knoten eine Konstituente des Textes repräsentiert. Die Relationen zwischen den Konstituenten werden gemäß den atomaren und den zusammengesetzten Teilbäumen definiert. Jeder Knoten in einem Baum verfügt über folgende Informationen:

Objekt: ein UMO oder eine lexikalisches Objekt

strukturelle Relationen: die strukturelle Relation zum Vaterknoten bzw. zu den Kinderknoten (d. h. Argument, Matrix, Adjunkt oder Koordinate, bzw. Kopf oder Compositum)

semantische Rolle: die semantische Rolle in bezug auf den Vater

Metafunktion: das Upper Model Konzept, dem die Konstituente angehört, respektive die abstrakte syntaktische Kategorie der Konstituente

Layout: LaTeX-Befehle, die vor bzw. nach der Verbalisierung des Teilbaums mit diesem Knoten als Wurzel ausgeführt werden müssen

weitere Informationen: weitere Markierungen, die in keiner der bisherigen Klassen angehören (z. B. die Kennzeichnung der Intention bei imperativen Sätzen, oder die Information, ob der Knoten negiert zu verbalisieren ist)

3.4 Konsistente Textstrukturen

Um die syntaktische Realisierbarkeit einer Textstruktur sicherzustellen, müssen bestimmte Typbeschränkungen ihrer Knoten bezüglich der abstrakten syntaktischen

Kategorien erfüllt sein. Diese Typbeschränkungen werden im folgenden dargestellt. Eine Textstruktur, die diese Bedingungen erfüllt, heißt ***konstistent***.

Die erlaubte ASK eines Knotens ist abhängig von der des Elternknotens oder eines ausgezeichneten Geschwisterknoten, nämlich der Matrix in unterordnenden Teilbäumen. Man nennt einen solchen Knoten von seinem Eltern- bzw. Geschwisterknoten *abhängig*, während diese in diesem Kontext *unabhängig* genannt werden. (Sie sind aber wiederum bezüglich ihrer eigenen Eltern- oder Geschwisterknoten abhängig.)

Definition 3 Eine Textstruktur T ist *konsistent*, wenn für alle Basistyp-Teilbäume S von T folgendes gilt:

1. Ist S ein atomarer Teilbaum, so können Wurzel und Blätter beliebige ASKn besitzen.

Wurzel (unabhängig)	Argument (abhängig)
x	y

wobei x und y beliebige ASKn sind.

2. Ist S ein nebenordnender Teilbaum, so sind die Blätter abhängig von der Wurzel. Ist das Objekt der Wurzel von S `nil`, so handelt es sich um die Wurzel der Textstruktur T, die als ASK text hat, wobei die ASKn der Blätter entweder sentence oder layout sind. Andernfalls entspricht das Objekt der zu verwendenden Konjunktion, und die ASKn der Blätter stimmen mit der der Wurzel überein.

Wurzel (unabhängig)	Koordinate (abhängig)
text	sentence, layout
np	np, gerund, layout
gerund	np, gerund, layout
x	x, layout

wobei x eine andere ASK ist, als die oben genannten.

3. Ist S ein unterordnender Teilbaum, so ist die Matrix abhängig von der Wurzel:

Wurzel (unabhängig)	Matrix (abhängig)
sentence	p-sentence, clause, vp
p-sentence	clause, vp
clause	clause, vp
np	np, gerund
gerund	np, gerund
vp-modifier-rankingII	np, gerund
clause-modifier-rankingII	np, gerund
np-modifier-embedded	clause
vp-modifier-embedded	clause
clause-modifier-embedded	clause
x	x

wobei x eine andere ASK ist, als die oben genannten.

Die Adjunkte sind wiederum abhängig von der Matrix:

Matrix (unabhängig)	Adjunkt (abhängig)
p-sentence	particle
clause	clause-modifier, sentence
vp	vp-modifier, sentence
np	name, np-modifier, sentence
gerund	name, np-modifier, sentence
clause-modifier-rankingl	intensifier
vp-modifier-rankingl	intensifier
np-modifier-rankingl	intensifier

■

3.5 Realisationsklassen

Die Konstruktion der Textstruktur geschieht sukzessive mittels sogenannter ***Expansionsbäume*** *(resource trees)*, die eine flexible Möglichkeit bieten, die Textstruktur zu erweitern. Ein Expansionsbaum ist ein vordefinierter, aus den Basistypen zusammengesetzer konsistenter Teilbaum der Textstruktur, der nach Instantiierung an den Blättern der Textstruktur angehängt werden kann und dadurch die Textstruktur an den Blättern expandiert. In jedem Knoten der Textstruktur wird vermerkt, durch welchen Expansionsbaum er erzeugt bzw. expandiert wurde.

Expansionsbäume werden zu allen Konzepten des Upper Models definiert. Sie dienen dazu, deren strukturellen Realisierungen festzulegen und bilden daher die Verbindung zwischen Syntax und Semantik. Man beachte, daß Expansionsbäume selbst konsistent sein müssen, um die Konsistenz der Textstruktur garantieren zu können. Sie müssen daher auch Informationen über die abstrakten syntaktischen Kategorien ihrer Konstituenten enthalten. Da häufig mehrere ASKn möglich sind, wird in jedem Knoten eines Expansionsbaums die Menge der erlaubten ASKn vermerkt. Diese wird im folgenden *ASK-Menge* genannt. Der Konsistenzbegriff wird auf ASK-Mengen so erweitert, daß zu jeder ASK eines unabhängigen Knotens ein konsistente ASK im zugehörigen abhängigen Knoten existiert und umgekehrt.

Da die meisten Upper-Model-Konzepte auf verschiedene Weisen realisierbar sein sollen, gibt es zu ihnen mehrere Expansionsbäume, die mit Prioritäten[1] versehen werden, um eine Auswahl zwischen ihnen treffen zu können. Die Menge der Expansionsbäume eines UMKs wird in seiner ***Realisationsklasse*** zusammengefaßt.

Ein Beispiel für eine Realisationsklasse ist unten für das UMK *implication* angeführt. Die Verbalisierung ist in Kommentaren vor den jeweiligen Expansionsbäumen *(resource trees)* angegeben.

[1]Die Prioritäten sind wie in Kapitel 2.3.3 definiert.

```
(def-realization-class  (implication :cause P :effect Q)
  ; if P then Q
  (resource-tree ((mandatory P Q) (negated no)
                  (priority (verb . 100)))
     (composite :object nil
                :textual (sentence finite)
                :matrix
                (composite :object (lex 'if)
                           :textual (finite)
                           :matrix
                           (leaf :object P
                                 :textual (finite vp)))
                :adjuncts
                ((composite :object (lex 'then)
                            :textual (clause-modifier-embedded)
                            :matrix
                            (leaf :object Q
                                  :textual (finite vp))))))
  ; P implies Q
  (resource-tree ((mandatory Q) (negated no)
                  (priority (verb . 10)))
     (composite :object nil
                :textual (sentence clause)
                :matrix
                (kernel :object (lex 'imply)
                        :textual (finite non-finite vp)
                        :arguments
                        ((leaf :object P
                               :role ':cause
                               :textual (np gerund))
                         (leaf :object Q
                               :role ':effect
                               :textual (np gerund
                                         clause-modifier-embedded
                                         vp-modifier-embedded)))))))
```

```
; P => Q
(resource-tree ((mandatory Q) (negated no)
                (priority (sym .  10)))
   (composite :object nil
              :textual (sentence clause)
              :matrix
              (kernel :object (sym 'imply)
                      :textual (finite non-finite vp)
                      :arguments
                      ((leaf :object P
                             :role ':cause
                             :textual (np))
                       (leaf :object Q
                             :role ':effect
                             :textual (np))))))
(resource-tree ((mandatory P Q) (negated no)
                (priority (sym .  10)))
   (composite :object (sym 'imply)
              :textual (np)
              :coords
              ((leaf :object P :textual (np))
               (leaf :object Q :textual (np))))))
```

Hierbei bedeutet `mandatory`, das die jeweiligen Variablen in dem UMO des Blattes belegt sein müssen, um die Auswahl des jeweiligen Expansionsbaums zu erlauben. In allen drei Fällen darf die Konstituente nicht negiert werden (`(negated no)`). Die Eintragungen `lex` zu den Objekten der Knoten, bedeutet, daß ein Wort ausgewählt wird, während `sym` ein mathematisches Symbol bezeichnet. Die Priorität ist analog zu den Prioritäten für UMVn zu interpretieren (s. Kapitel 2.3.1).

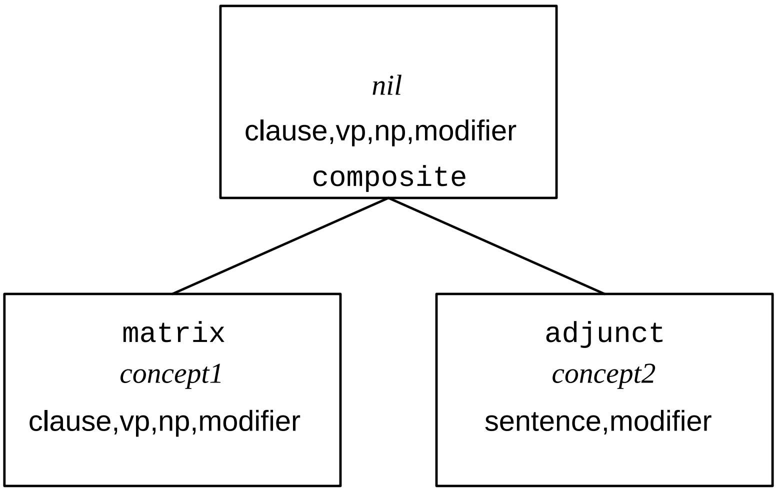
Im Anhang B sind alle Realisationsklassen angegeben und ihre Eingabesprache definiert.

Die Konsistenz der Textstruktur wird durch die konsistente Definition der Realisationsklassen und konsistenzerhaltende Auswahl von Expansionsbäumen gewährleistet. Wie diese Auswahl erfolgt, wird im nächsten Abschnitt besprochen.

3.6 Die Auswahl von Expansionsbäumen

Bei der Auswahl eines Baums aus der Realisationsklasse zur Expansion eines Blattes der Textstruktur, muß einerseits darauf geachtet werden, daß der Expansionsbaum die Konsistenz der Textstruktur bewahrt, d. h. der Schnitt S aus den ASK-Mengen des zu expandierenden Blattes und der Wurzel des Expansionsbaums darf nicht leer sein. Andererseits kann der Expansionsbaum aber auch umgekehrt die ASK-Menge des zu dem expandierten Knoten gehörenden abhängigen und unabhängigen Knoten weiter einschränken, nämlich dann wenn S eine echte Teilmenge der ursprünglichen ASK-Menge des expandierten Knotens ist. Nach Auswahl eines konkreten Expansionsbaum muß diese Information entsprechend propagiert werden.

Beispiel: Betrachten wir folgenden Teilbaum einer Textstruktur:



Wird die Matrix durch einen Expansionsbaum erweitert, dessen Wurzel auf die ASK `np` trägt, so muß die Wurzel des Teilbaums ebenfalls auf `np` eingeschränkt werden, während das Adjunkt auf die ASK-Menge {sentence, np-modifier} einzuschränken ist. ■

Gibt es mehrere mögliche Expansionsbäume wird zunächst versucht, dem Grundsatz des *casting* zu folgen (vgl. Kapitel 2.3.1). An dieser Stelle bedeuted das, daß in Geschwisterknoten des Typs Koordinate gleiche Expansionsbäume ausgewählt werden. Ist das nicht möglich, entscheidet eine Heuristik H (vgl. Kapitel 5.3) anhand der Prioritäten und eines Diskursgedächtnisses, durch welchen Baum die Expansion erfolgt.

3.7 Der Aufbau der Textstruktur

Die zentrale Aufgabe des Mikroplaners in *PROVERB* ist die Erzeugung der Textstruktur. Der unten vorgestellte abstrakte Algorithmus zur Generierung der Textstruktur bildet eine Konkretisierung der Abbildungen zwischen den ersten drei in Kapitel 1.4 angegebenen Ebenen der Repräsentation. Dieser Algorithmus genügt im Prinzip, um die linguistische Spezifikation des Textes zu erzeugen.

1. Initialisierung:

   Die initiale Textstruktur besteht aus einem einzelnen Knoten mit dem UMO als Objekt, auf das die Folge der PMs (also ein APO) abgebildet wurde.

2. Terminierung:

   `if` das Objekt des aktuellen Knotens ist ein lexikalisches Objekt
   `then stop`

3. Abbildung in das Upper Model:

   `if` das Objekt des aktuellen Knotens ist ein UMO, an dessen Wurzel ein UMV steht then wähle unter Berücksichtigung von Typrestriktionen und des Prinzip des *casting* bzw. nach Heuristik H für diese UMV ein passendes UMO aus

4. Expansion:

   wähle unter Berücksichtigung des Prinzips des *casting* bzw. nach Heuristik H einen konsistent einfügbaren Expansionsbaum aus und expandiere den aktuellen Knoten unter Propagierung der Typrestriktionen des Konzeptes an der Wurzel des UMOs

5. Rekursion:

   `for all` Blätter unterhalb des aktuellen Knotens (von links nach rechts)
   `do` das Blatt wird zum neuen aktuellen Knoten; `goto` Schritt 2

Man beachte, daß in Schritt 1 das APO, das den gesamten Text repräsentiert, nämlich die Liste der PMs, in ein UMO abgebildet wird. Wie in Kapitel 2.3.1 bereits angeführt wurde, besteht dieses UMO nur aus UMVn. Ferner beachte man, daß die Expansionsbäume garantieren müssen, daß in den Blättern als Objekte nur UMOe oder lexikalische Objekte (Wörter oder mathematische Symbole) auftreten.

Die Heuristik H wird in Kapitel 5.3 vorgestellt. Sie entscheidet über die Auswahl, indem sie für jedes zur Wahl stehende Objekt anhand der gegebenen Prioritäten und eines Diskursgedächnisses einen Index berechnet. Das Objekt mit dem höchsten Index wird ausgewählt. Wie oben bereits dargelegt, genügt dieser Algorithmus im Prinzip, um die linguistische Spezifikation eines sprachlich korrekten Textes zu generieren. In dem folgenden Kapitel werden Operationen vorgestellt, die die Qualität des Textes zum Teil erheblich verbessern.

Kapitel 4

Sortierung und Aggregation

In den beiden letzten Kapiteln haben wir den grundlegenden Mechanismus zur Satzplanung kennengelernt. Im Gegensatz zu der direkten Abbildung von Anwendungsprogrammobjekte auf einzelne Sätze erlaubt dieser Mechanismus eine flexible Paraphrasierung der zu übermittelnden Information, d. h. sie läßt sich durch verschiedene Verbalisierungsmöglichkeiten ausdrücken. Darüberhinaus verschafft uns der Mechanismus der Satzplanung eine Basis, auf der wir leicht weiter verfeinernde Satzplanungstechniken spezifizieren können.

Folgte man lediglich dem in Kapitel 3.7 vorgestellten Algorithmus, würde die Struktur des Beweises in den linguistischen Strukturen widergespiegelt. Wie wir in Kapitel 1.3 gesehen haben, ist in den einzelnen Beweisschritten aber oft noch redundante Information enthalten. Durch die ***Sortierung*** der Information eines Beweisschrittes und ihrer Umgruppierung durch ***Aggregation*** läßt sich Redundanz in den generierten Äußerungen vermeiden.

Die Sortierung wird in *PROMI* durch eine einfache Heuristik gesteuert, während die Aggregation regelbasiert erfolgt. Sowohl Sortierung als auch Aggregation sollten auf der semantischen Ebene erfolgen, da sie auf der syntaktischen Ebene an Mächtigkeit verlieren. In *PROMI* finden beide Techniken auf der Ebene der Upper-Model-Objekte (oder genauer: auf Upper-Model-Variablen) Anwendung, da auf dieser Ebene noch keine strukturelle Planung des Textes die Anwendung erschwert und daher gleichförmige Elemente behandelbar sind.

Der um diese verfeinernden Techniken erweiterte Mikroplanungsalgorithmus wird in Kapitel 6 dargestellt.

4.1 Sortierung

In *PROMI* wird zur Zeit nur eine einfache Sortierheuristik angewandt, die aber unseren Zwecken genügt. Es werden lediglich die Argumente von UMOen der kommutativen Konzepte *conjunction* und *disjunction* sortiert, da nur bei ihnen gewährleistet ist, daß ihre Reihenfolge keine Bedeutung kodiert.

Die Sortierung der Argumente der genannten UMOe geschieht nach ihrer Stelligkeit, so daß Argumente geringerer Stelligkeit links stehen. Die Idee dabei ist, daß in dem Fall, daß die Argumente logische Prädikate sind, einstellige Prädikate definierende Eigenschaften von Objekten kodieren, während mehrstellige Prädikate die Relationen zu anderen Objekten spezifizieren.

Beispiel: $conjunction\langle(Subset(F,G), Set(F))\rangle$ wird umsortiert zu
$conjunction\langle(Set(F), Subset(F,G))\rangle$ ∎

Es ist sinnvoller, die Information "F is a set" zu verbalisieren, bevor man "F is a subset of G" äußert, als umgekehrt.

4.2 Aggregation

Eine der wichtigsten Mikroplanungstechniken ist die ***Aggregation***. Hierbei wird die zu übermittelnde Information so neu strukturiert, daß Redundanzen vermieden werden. Als Beispiel betrachte man den folgenden Beweis, der zwar alle notwendigen Fakten enthält, aber durch seine Kompliziertheit dem Leser einige Mühen abverlangt:

> "Let F be a set. Let G be a set. Let $F \subset G$. Let $a \in F$. Let $b \in F$. Since $a \in F$ and $F \subset G$, $a \in G$. Since $b \in F$ and $F \subset G$, $b \in G$."

Durch Aggregation der Inhalte kann dieselbe Information auch prägnanter ausgedrückt werden, wodurch sich die Verständlichkeit des Beweises erheblich verbessert:

> "Let F and G be sets, $F \subset G$ and $a, b \in F$. Then $a, b \in G$."

Aggregation reorganisiert den Text durch die Zusammenfassung von Information und eliminiert so inhaltliche Redundanzen, was einen flüssigerer Text zur Folge hat. Aggregationstechniken wurden bereits mehrmals in unterschiedlichen Systemen unter verschiedenen Namen eingesetzt, allerdings ohne daß sie klassifiziert oder tiefgehend untersucht worden wären. So spricht Dale von *discourse level optimization* [Dale, 1992], Kempen von *forward* bzw. *backward conjunction reduction* [Kempen, 1991] und Dalianis von *compacting* [Dalianis, 1993]. Horacek stellt die ausgeklügelste Technik vor, in der er zwischen inhaltsmotivierter Aggregation *(content motivated grouping)*, in der Weltwissen angewandt wird, und strukturell motivierter Aggregation *(structurally motivated grouping)*, die auf der Gleichheit der Strukturen von Aussagen basiert, unterscheidet [Horacek, 1992]. Da in *PROMI* bisher keine Taxonomie mathematischer Objekte enthalten ist, gehören die Aggregationsregeln, die hier vorgestellt werden, ausschließlich der zweiten Kategorie an.

Dalianis und Hovy leiteten aus einer empirischen Studie eine Klassifikation von acht Aggregationsregeln her [Dalianis und Hovy, 1993]. Diese setzt sich zusammen aus:

- zwei sogenannten *grouping*-Regeln, die Subjekte bzw. Prädikate zusammenfassen,
- drei *ordering*-Regeln, die eine Sortierung der Inhalte vornehmen,
- eine *casting*-Regel, die, wie in Kapitel 2.3.1 dargelegt, bewirkt, daß semantisch verwandte Inhalte in syntaktisch ähnlicher Form ausgedrückt werden, sowie

- zwei *parsimony*-Regeln, die auf die Menge der zu übermittelnden Information in einem Satz Einfluß nehmen.

Faßt man Aggregation jedoch als eine Technik zur Redundanzvermeidung auf, sind nur die beiden *grouping*-Regeln als echte Aggregationsregeln zu betrachten, da nur sie redundante Information zusammenfassen. Die übrigen Regeln sind zwar zur Verbesserung der Textqualität geeignet, aber nicht hier zu besprechen, da ihr Zweck in *PROMI* durch die Sortierungsheuristik, durch eine geeignete Auswahl von Upper-Model-Konzepten und Expansionsbäumen sowie durch das Referenzierungsmodul DRCC (vgl. Kapitel 1.4 und [Huang, 1994c]) erfüllt wird. Die Regeln von Dalianis und Hovy wurden auf der syntaktischen Ebene definiert. Ihre *grouping*-Regeln oder *Regeln zur Zusammenfassung* werden an dieser Stelle anhand von Beispielen aus unserer Domäne kurz vorgestellt:

(A1) Zusammenfassung von Aussagen mit gleichem Subjekt *(subject grouping)*, z. B.

“F is a set. F has finite cardinality.” $\curvearrowright$ “F is a finite set and has finite cardinality.”

(A2) Zusammenfassung von Aussagen mit gleichem Prädikat *(predicate grouping)*, z. B.

“F is a set. G is a set.” $\curvearrowright$ “F and G are sets.”

Die meisten der Regeln, die ich hier vorstellen will, gehen auf diese beiden Regeln zurück. Sie können als Anpassungen und Erweiterungen für die Domäne der mathematischen Texte gesehen werden. Darüberhinaus gibt es auch Regeln, für die es bei Dalianis und Hovy kein Pendant gibt. So wurde im Rahmen dieser Arbeit die Klasse der ***Verkettungsregeln*** *(chaining rules)* speziell für Beweistexte entwickelt. Sie erlauben es, Inhalte, die semantisch gesehen in einer Ableitungskette stehen, auch als solche auszudrücken, so daß eine redundante Erwähnung von Zwischenergebnissen vermieden werden kann. Diese Verkettung von Ableitungen ist auch in argumentativen Texten, die nicht auf die mathematische Domäne beschränkt sind, vorteilhaft.

Die hier vorgestellten Aggregationsregeln werden auf Upper-Model-Objekte angewandt, arbeiten also auf der semantischen Ebene. (Die wichtigsten für Aggregation relevanten Konzepte des Upper Models sind in Abbildung 4.1 nochmals dargestellt.) Daher ließen sie sich so definieren, daß sie über die nur auf der syntaktischen Ebene definierten Regeln (A1) und (A2) hinausgehen, obwohl sie in ihrer Idee auf diese zurückgehen. Da die logische Struktur des Beweises unberührt bleiben muß, werden die Aggregationsregeln nur auf Argumente der UMOe der Konzepte *conjunction*, *disjunction* oder *sequence* angewendet. Weiterhin wird nach Anwendung jeder Aggregationsregel geprüft, ob die Argumentliste der *conjunction*, *disjunction* oder *sequence* auf die Länge 1 reduziert wurde. Falls das der Fall ist, wird das UMO durch sein einzelnes Argument ersetzt. Ist die Argumentliste leer, so wird die UMO gelöscht.

Abbildung 4.2 zeigt eine Klassifizierung der hier vorgestellten Aggregationsregeln. In Klammern ist die Anzahl der Regeln in der jeweiligen Klasse angegeben. Die Regeln zur Zusammenfassung werden gemäß ihrer Wirkung unterteilt in solche, die semantische Objekte, d. h. UMOe des Konzepts *object*, oder semantische

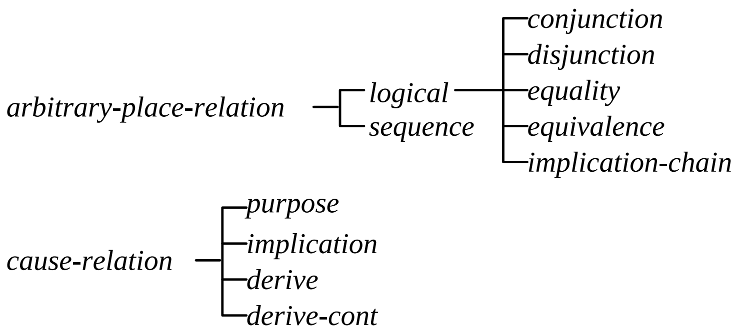


Abbildung 4.1: Einige aggregierbare Konzepte des Upper Models

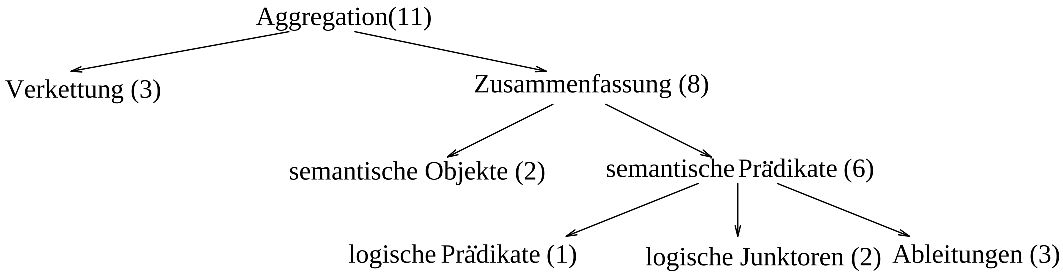


Abbildung 4.2: Die Klassifikation der Aggregationsregeln

Prädikate, d. h. UMOe des Konzepts *process*, zusammenfassen. Die Regeln zur Verkettung werden in Kapitel 4.2.1 vorgestellt. In Kapitel 4.2.2 werden die Regeln zur Zusammenfassung eingeführt.

Zur Beschreibung der Regeln werden folgende Notationen verwandt:

- Eine Regel wird geschrieben als $\frac{T}{T'}$, wobei T, T' UMOe sind. Die Regelanwendung ersetzt T durch T'.

- ϵ bezeichnet das leere UMO.

- Seien A, P UMOe, wobei P das UMO B als Argument hat. Dann bezeichnet $P[A/B]$ das UMO, das durch Ersetzung von B durch A aus P entsteht. Entsprechend bezeichnet $P[\epsilon/B]$ die Löschung des Argumentes B aus P.

- Argumente eines UMOs sind durch $\langle\rangle$ geklammert.

- Objekte der Untertypen von *arbitrary-place-relation* können beliebig viele Argumente besitzen. Diese werden in einer durch () geklammerten Liste zusammengefaßt und als einzelnes Argument betrachtet.

- Vor der formalen Definition jeder Regel erläutert eine Graphik die Funktionsweise der Regel. Grau unterlegt Strukturen repräsentieren dabei redundante Informationen

4.2.1 Verkettung

Eine wichtige Erweiterung zu den Regeln von [Dalianis und Hovy, 1993] bilden drei Regeln, die Redundanzen durch ***Verkettung*** von Ableitungen eliminieren. Die erste Regel behandelt logische Äquivalenzen, während die verbleibenden Regeln auf Ableitungen arbeiten. Dabei ist zwischen ***verketteten*** Ableitungen, in denen die Schlußfolgerung eines Ableitungsschrittes eine Voraussetzung für den nächsten bildet, und ***unabhängigen*** Ableitungen, in denen keine Schlußfolgerung eines Ableitungsschrittes als Voraussetzung eines anderen vorkommt, zu unterscheiden. Nur aus verketteten Ableitungen lassen sich durch Kettenbildung Ableitungsketten erzeugen.

Ableitungsketten

Zunächst müssen wir den Begriff der ***Ableitungskette*** als Upper-Model-Objekt formal fassen.

Definition 4 Eine *Ableitungskette* wird wie folgt induktiv definiert:

(i) Jedes UMO der Form $assume\langle C_1, \epsilon\rangle$ ist eine Ableitungskette der Länge 1.

(ii) Jedes UMO der Form $derive\langle R_1, M_1, C_1, \epsilon\rangle$ ist eine Ableitungskette der Länge 1.

(iii) Jedes UMO der Form $S_1\langle\Phi_1, S_2\langle\Phi_2, \ldots, S_{k-1}\langle\Phi_{k-1}, S_k\langle\Phi_k, \epsilon\rangle\rangle \ldots\rangle\rangle$ ist eine Ableitungskette der Länge k, falls folgendes gilt:

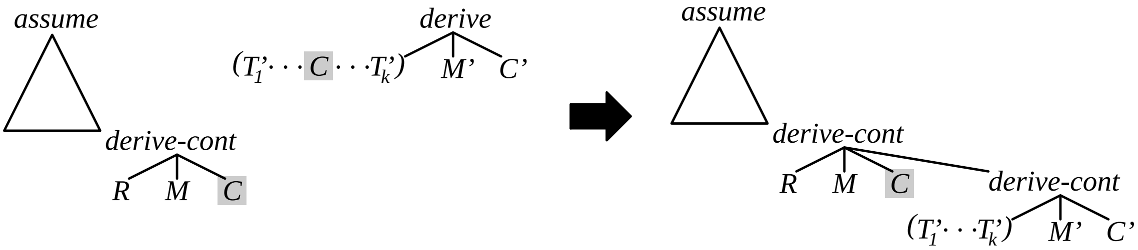
- $S_1 \in \{assume, derive\}$
- $\Phi_1 = \begin{cases} C_1 & \text{falls } S_1 = assume \\ R_1, M_1, C_1 & \text{falls } S_1 = derive \end{cases}$
- und für alle $1 < i \leq k$ gilt $S_i = derive\text{-}cont$ und $\Phi_i = R_i, M_i, C_i$.

Dabei stehen R_i, M_i und C_i für alle $1 \leq i \leq k$ in der semantischen Rolle `reason` (die Vorbedingungen), `method` (die Rechtfertigung) bzw. `conclusion` (die Schlußfolgerung des Ableitungsschrittes) und S_i für alle $1 < i \leq k$ und ϵ in der semantischen Rolle `next` (der nächste Ableitungsschritt).
Notation: Spielt nur die Schlußfolgerung des letzten Ableitungsschrittes einer Ableitungskette eine Rolle, so wird verkürzt $chain\langle C_k, \epsilon\rangle$ geschrieben. ■

Man erinnere sich, daß die beiden Konzepte *derive* und *derive-cont* sich in ihrer Semantik unterscheiden (vgl. Kapitel 2.3.1):

- $derive\langle\epsilon, M, C, \epsilon\rangle$ bedeuted, daß C ohne Voraussetzung mit M gefolgert werden kann: „Es gilt C wegen M."
- $derive\text{-}cont\langle\epsilon, M, C, \epsilon\rangle$ beinhaltet jedoch, daß die Schlußfolgerung des vorhergehenden Schrittes der Ableitungskette die Voraussetzung zur Ableitung von C durch M bildet: „Dann gilt C wegen M."

Bei der Verkettung von Ableitungsschritten, in denen ja die Konklusion eines Schrittes eine Prämisse des nächsten darstellt, wird diese Prämisse nicht nochmals verbalisiert. Vielmehr wird dieser Schritt, um die Verkettung deutlich zu machen, aus dem Konzept *derive* in das Konzept *derive-cont* überführt:



Die formale Definition dieser Regel lautet wie folgt:

Regel 1 *(Ableitungsketten)*
Sei C_1 die Schlußfolgerung einer Ableitungskette (möglicherweise der Länge 1). Seien ferner R_2 die Vorbedingungen, M_2 die Methode und C_2 die Schlußfolgerung eines UMOs *derive*, wobei C_1 in R_2 vorkommt.

$$\frac{\mathit{sequence}\langle(\ldots \mathit{chain}\langle C_1, \epsilon\rangle, \mathit{derive}\langle R_2, M_2, C_2, \epsilon\rangle \ldots)\rangle}{\mathit{sequence}\langle(\ldots \mathit{chain}\langle C_1, \mathit{derive\text{-}cont}\langle R_2[\epsilon/C_1], M_2, C_2, \epsilon\rangle\rangle \ldots)\rangle}$$

■

Diese Regel verlängert die Ableitungskette um einen Schritt, nämlich um das Upper-Model-Objekt der Kategorie *derive-cont*, in der C_1, das soeben verbalisiert wurde, nicht noch einmal als Vorbedingung ausdrücklich erwähnt wird.

Beispiel: Betrachten wir folgende beiden UMOe:

$\mathit{derive}\langle\epsilon, \quad \mathit{DefTrans}, \quad \mathit{Subset}(\sigma, \mathit{transclosure}(\sigma)), \quad \epsilon\rangle$

"$\sigma \subseteq \sigma^*$, by the definition of transitive closure."

$\mathit{derive}\langle(\mathit{el}((x,y),\sigma), \mathit{Subset}(\sigma, \mathit{transclosure}(\sigma))),$
$\quad \mathit{DefSubset}, \quad \mathit{el}((x,y), \mathit{transclosure}(\sigma)), \quad \epsilon\rangle$

"Since $(x,y) \in \sigma$ and $\sigma \subseteq \sigma^*$, $(x,y) \in \sigma^*$ by the definition of subset."

Das erste UMO, eine Ableitungskette der Länge 1, wird mit dem zweiten zusammengefaßt zu:

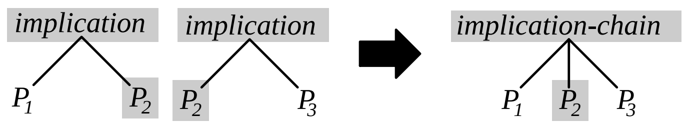
$\mathit{derive}\langle\epsilon, \quad \mathit{DefTrans}, \quad \mathit{Subset}(\sigma, \mathit{transclosure}(\sigma)),$
$\quad \mathit{derive\text{-}cont}\langle(\mathit{el}((x,y),\sigma)), \quad \mathit{DefSubset}, \quad \mathit{el}((x,y), \mathit{transclosure}(\sigma)), \quad \epsilon\rangle\rangle$

"$\sigma \subseteq \sigma^*$ by the definition of transitive closure, thus establishing $(x,y) \in \sigma^*$ by the definition of subset, since $(x,y) \in \sigma$."

■

Logische Implikationen und Äquivalenzen

Da eine Implikation ähnlich einem einzelnen Ableitungsschritt ausdrückt werden kann, kann auch ihre Verkettung sinnvoll sein. Da aber die Prämissen einer Implikation nicht notwendigerweise wahr sein müssen, genügt zu ihrer Verkettung nicht das Vorkommen der Konklusion in den Prämissen der nachfolgenden Implikation, sondern ihre Gleichheit wird verlangt.



Regel 2 *(Implikationen)*
Seien $P_1, \ldots, P_n$ UMOe.

$$\frac{S\langle(\ldots \mathit{implication}\langle P_1, P_2\rangle, \mathit{implication}\langle P_2, P_3\rangle, \ldots, \mathit{implication}\langle P_{n-1}, P_n\rangle \ldots)\rangle}{S\langle(\ldots \mathit{implication\text{-}chain}\langle(P_1, \ldots, P_n)\rangle \ldots)\rangle}$$

wobei $S \in \{\mathit{conjunction}, \mathit{sequence}\}$. ■

Beispiel: In diesem und dem folgenden Beispiel stehe *PL* für “predicate logic”, *PKP* für “Post's correspondence problem” und *HP* für “halting problem”.

$\mathit{sequence}\langle(\mathit{implication}\langle \mathit{decidable}(PL), \mathit{solvable}(PKP)\rangle,$
$\quad\mathit{implication}\langle \mathit{solvable}(PKP), \mathit{solvable}(HP)\rangle$
$\quad\mathit{implication}\langle \mathit{solvable}(HP), \mathit{decidable}(PL)\rangle)\rangle$

“If predicate logic is decidable, then is Post's correspondence problem solvable. If Post's correspondence problem is solvable, then is the halting problem solvable. If the halting problem is solvable, then is predicate logic decidable.”

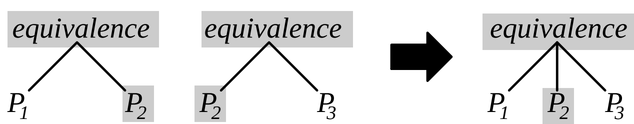
Obige Implikationen werden verkettet zu:

$\mathit{sequence}\langle(\mathit{implication\text{-}chain}\langle(\mathit{decidable}(PL),$
$\quad\mathit{solvable}(PKP),$
$\quad\mathit{solvable}(HP),$
$\quad\mathit{decidable}(PL))\rangle)\rangle$

“Predicate logic is decidable $\Rightarrow$ Post's correspondence problem is solvable
$\Rightarrow$ the halting problem is solvable
$\Rightarrow$ predicate logic is decidable”

■

Ähnlich dem Fall der Implikation ist auch die Verkettung von Äquivalenzen sinnvoll.



Das Upper-Model-Konzept *equivalence* erlaubt eine beliebige Stelligkeit der Äquivalenz, deren Argumente alle untereinander logisch gleichwertig sind. Daher können zwei beliebigstellige Äquivalenzen verkettet werden, wenn das letzte Argument der ersten Äquivalenz mit dem ersten Argument der zweiten identisch ist.

Regel 3 *(Äquivalenzen)*
Seien $P_1, \ldots, P_m$ und $Q_1, \ldots, Q_n$ die Argumente zweier Äquivalenzen mit $P_m = Q_1$

$$\frac{S\langle(\ldots \mathit{equivalence}\langle(P_1, \ldots, P_m)\rangle, \mathit{equivalence}\langle(Q_1, Q_2, \ldots, Q_n)\rangle) \ldots\rangle}{S\langle(\ldots \mathit{equivalence}\langle(P_1, \ldots, P_m, Q_2, \ldots, Q_n)\rangle \ldots)\rangle}$$

wobei $S \in \{\mathit{conjunction}, \mathit{sequence}\}$. ■

Beispiel: Betrachten wir folgende zwei Äquivalenzen:

$$\mathit{sequence}\langle(\mathit{equivalence}\langle(\mathit{decidable}(PL), \mathit{solvable}(PKP))\rangle,\\ \mathit{equivalence}\langle(\mathit{solvable}(PKP), \mathit{solvable}(HP))\rangle)\rangle$$

"Predicate logic is decidable if and only if Post's correspondence problem is solvable. Post's correspondence problem is solvable if and only if the halting problem is solvable."

Diese Äquivalenzen können zusammengefaßt werden zu:

$$\mathit{equivalence}\langle(\mathit{decidable}(PL), \mathit{solvable}(PKP), \mathit{solvable}(HP))\rangle$$

"Predicate logic is decidable if and only if Post's correspondence problem is solvable if and only if the halting problem is solvable."

■

4.2.2 Zusammenfassung

Nachdem wir nun die Regeln zur Verkettung von Ableitungen untersucht haben, gehen wir zu den Regeln zur Zusammenfassung über. Sie lassen sich in Regeln, die ***semantische Objekte*** bzw. ***semantische Prädikate*** aggregieren, unterteilen. Unter semantischen Objekten verstehen wir UMOe, die Objekte der Domäne repräsentieren, während semantische Prädikate Eigenschaften der Objekte und Relationen zwischen Objekten beschreiben.

Regeln, die semantische Objekte behandeln, und Regeln, die semantische Prädikate zusammenfassen, unterscheiden sich in ihrem Mechanismus. Intuitiv gesehen werden durch die erstgenannten Strukturen durch Ineinanderhängen verknüpft (siehe links in Abbildung 4.3), während die letztgenannten gleichartige Strukturen zusammenfassen (rechts in Abbildung 4.3).

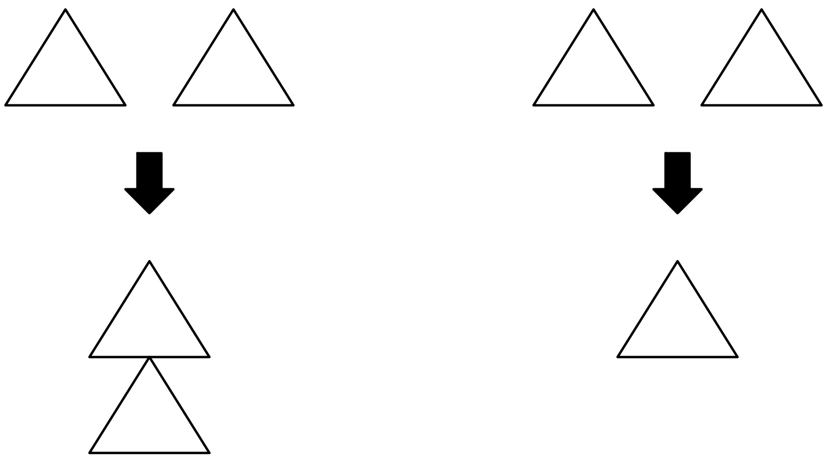
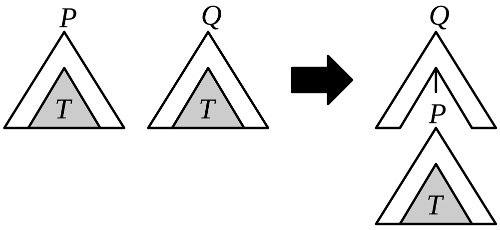


Abbildung 4.3: Aggregationsmechanismen

Semantische Objekte

Semantische Objekte, d. h. UMOe eines Subkonzeptes von *object*, können syntaktisch sowohl als Subjekt als auch als Objekt im Satz realisiert werden. Daher bewirkt eine einzelne Regel, die redundante semantische Objekte eliminiert, sowohl die Zusammenfassung von Sätzen, die in ihren Subjekten übereinstimmen, als auch die Zusammenfassung von Sätzen die in ihren Objekten übereinstimmen. Darüberhinaus faßt diese Regel auch Sätze zusammen, in denen das Subjekt eines Satzes als Objekt in den übrigen auftritt. Sie geht also in ihrer Wirkung beträchtlich über die Regel (A1) von Dalianis und Hovy hinaus, die lediglich Subjekte zusammenfaßt.



Regel 4 *(Semantische Objekte (rechts))*
Sei *con* ein Subkonzept von *object*. Seien ferner P und Q beliebige UMOe mit unterschiedlichen Wurzeln, die beide das UMO T des Konzepts *con* enthalten, wobei die Wurzel von P eine UMV ist, die nach einem UMO des Konzepts *con* abgebildet werden kann.

$$\frac{\mathit{conjunction}\langle(\ldots P, Q \ldots)\rangle}{\mathit{conjunction}\langle(\ldots Q[P/T] \ldots)\rangle}$$

■

Das folgende Beispiel erläutert die Funktion dieser Regel:

Beispiel: Habe F das Konzept *set*, auf das *Set* ebenfalls abgebildet werden kann. Dann wird

$\mathit{conjunction}\langle(\mathit{Set}\langle F\rangle, \mathit{Subset}\langle F, G\rangle)\rangle$

"F is a set. F is a subset of G."

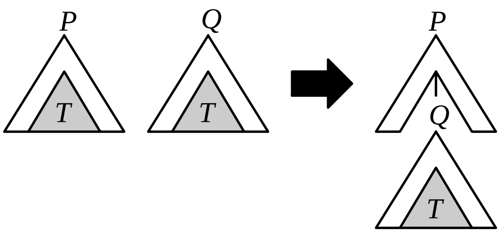
zusammengefaßt zu

$\mathit{Subset}\langle \mathit{Set}\langle F\rangle, G\rangle$

"The set F is a subset of G."

Man beachte, daß die Konjunktion nach der Zusammenfassung der Argumente gelöscht wurde. ■

Eine dazu symmetrische Regel ist ebenfalls sinnvoll.



Regel 5 *(Semantische Objekte (links))*
Sei *con* ein Subkonzept von *object*. Seien ferner P und Q beliebige UMOe mit unterschiedlichen Wurzeln, die beide das UMO T des Konzepts *con* enthalten, wobei die Wurzel von Q eine UMV ist, die nach einem UMO des Konzepts *con* abgebildet

werden kann.

$$\frac{conjunction\langle(\ldots P, Q \ldots)\rangle}{conjunction\langle(\ldots P[Q/T] \ldots)\rangle}$$

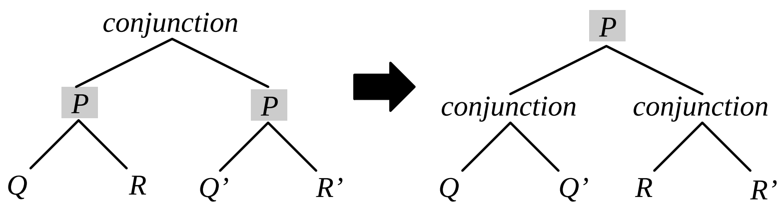
■

Beide Regeln wurden in der Implementation auf solche UMOe eingeschränkt, in denen T direkter Nachkomme von P bzw. Q ist.

Semantische Prädikate

Semantische Prädikate repräsentieren einerseits die semantischen Relationen zwischen semantischen Objekten, andererseits aber auch die rhetorischen Relationen zwischen Konstituenten des Textes. Aus diesem Grunde bewirkt die Zusammenfassung semantischer Prädikate die Aggregation von Sätzen mit gleichen Prädikaten (im syntaktischen Sinne), geht über diese aber hinaus, da sie auch größere Strukturen des Textes behandelt. Die hier vorgestellten Regeln stellen daher in ihrer Gesamtheit eine echte Erweiterung zu den Regeln von Dalianis und Hovy dar. Wie aus Abbildung 4.2 ersichtlich, wird hier unterscheiden zwischen Regeln, die logische Prädikate zusammenfassen, solchen die logische Junktoren zusammenfassen und solchen, die Ableitungen oder Ableitungsschritte zusammenfassen.

Logische Prädikate Die folgende Regel kann nur auf Upper-Model-Objekte angewandt werden, die Prädikate im Sinne der mathematischen Logik repräsentieren. Daher ist insbesondere ihre Anwendung auf UMOe der Konzepte *derive*, *assume* oder *implication*, die für rhetorische Relationen stehen, nicht möglich.



Regel 6 *(Prädikate)*
Sei P ein UMO, das ein logisches Prädikat repräsentiert. Seien $S_{i,j}$ UMOe mit $1 \leq i \leq k$ und $1 \leq j \leq n$.

$$\frac{S\langle(\ldots P\langle S_{1,1}, \ldots, S_{1,n}\rangle, \ldots, P\langle S_{k,1}, \ldots, S_{k,n}\rangle \ldots)\rangle}{S\langle(\ldots P\langle S\langle(S_{1,1}, \ldots, S_{k,1})\rangle, \ldots, S\langle(S_{1,n}, \ldots, S_{k,n})\rangle\rangle \ldots)\rangle}$$

wobei $S \in \{conjunction, disjunction\}$. ■

Beispiel: Mit dieser Regel werden folgende beiden UMOe zusammengefaßt:

$conjunction\langle(Set\langle F\rangle, \quad Set\langle G\rangle)\rangle$

"F is a set. G is a set."

Die Aggregation ergibt:

$Set\langle conjunction\langle(F, G)\rangle\rangle$

"F and G are sets."

Auch hier wurde die einstellige Konjunktion gelöscht.

Die Regel erlaubt auch die Zusammenfassung mehrstelliger Prädikate:

$$\mathit{conjunction}\langle(\mathit{Subset}(F,G), \quad \mathit{Subset}(U,V))\rangle$$

"F is a subset of G. U is a subset of V."

Die Aggregation resultiert in:

$$\mathit{Subset}(\mathit{conjunction}\langle(F,U)\rangle, \mathit{conjunction}\langle(G,V)\rangle)\rangle$$

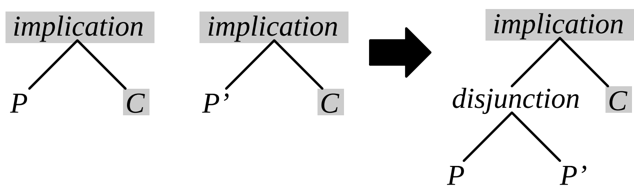
"F and U are subsets of G and V respectively."

■

Man beachte, daß im letzten Beispiel das Schlüsselwort "respectively" die Verständlichkeit sicherstellt. Es wird, wie in Kapitel 5.2 noch dargelegt wird, in einem späteren Stadium eingefügt. Allerdings wird aus diesem Beispiel auch deutlich, daß die Zusammenfassung von drei- oder mehrstelligen Prädikaten problematisch sein kann.

Logische Junktoren Zwei weitere Regeln behandeln logische Implikationen. Die Semantik der Implikation macht es jedoch notwendig, ihre Argumente in Betracht zu ziehen. Die Aggregation von Implikationen ist nur dann sinnvoll, wenn die Implikationen entweder in ihren Prämissen oder in ihren Konklusionen übereinstimmen.

Implikationen, die in ihren Konklusionen übereinstimmen, lassen sich wie folgt zusammenfassen:



Regel 7 *(Implikationen, die in ihren Konklusionen übereinstimmen)*
Seien $P_1, \ldots, P_n$ die Prämissen und C die Konklusion von aufeinanderfolgenden Implikationen.

$$\frac{S\langle(\ldots \mathit{implication}\langle P_1, C\rangle, \ldots, \mathit{implication}\langle P_n, C\rangle \ldots)\rangle}{S\langle(\ldots \mathit{implication}\langle \mathit{disjunction}\langle(P_1, \ldots, P_n)\rangle, C\rangle \ldots)\rangle}$$

wobei $S \in \{\mathit{conjunction}, \mathit{sequence}\}$. ■

Beispiel: Die Implikationen

$$\mathit{conjunction}\langle(\mathit{implication}\langle \mathit{less}(a,b), \mathit{notequal}(a,b)\rangle, \\ \mathit{implication}\langle \mathit{greater}(a,b), \mathit{notequal}(a,b)\rangle)\rangle$$

"If $a < b$ then $a \neq b$. If $a > b$ then $a \neq b$."

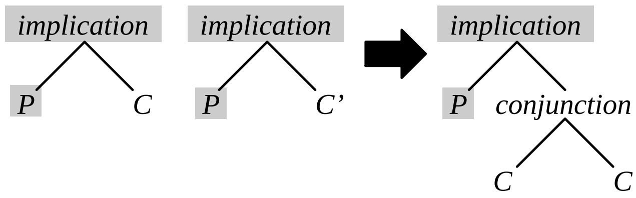
werden durch diese Regel zusammengefaßt zu

$$\mathit{implication}\langle \mathit{disjunction}\langle(\mathit{less}(a,b), \mathit{greater}(a,b))\rangle, \mathit{notequal}(a,b)\rangle$$

"If $a < b$ or $a > b$ then $a \neq b$."

■

Man beachte, daß die Semantik der Implikation die Disjunktion der Prämissen verlangt. Analog ist in der folgenden Regel, die Implikationen behandelt, die in ihren Prämissen übereinstimmen, die Konjunktion der Konklusionen erforderlich.



Regel 8 *(Implikationen, die in ihren Prämissen übereinstimmen)*
Seien P die Prämisse und $C_1, \ldots, C_n$ die Konklusionen von aufeinanderfolgenden Implikationen.

$$\frac{S\langle(\ldots \mathit{implication}\langle P, C_1\rangle, \ldots, \mathit{implication}\langle P, C_n\rangle \ldots)\rangle}{S\langle(\ldots \mathit{implication}\langle P, \mathit{conjunction}\langle(C_1, \ldots, C_n)\rangle\rangle \ldots)\rangle}$$

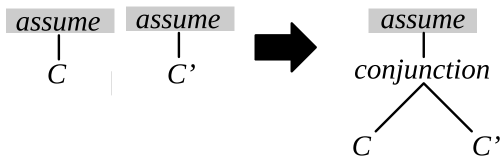
wobei $S \in \{\mathit{conjunction}, \mathit{sequence}\}$. ■

Die Regeln zur Behandlung logischer Junktoren scheinen ihre Anwendung auf der logischen Ebene zu implizieren. In *PROMI* wurde allerdings die Definition aller Aggregationsregeln in einem einheitlichen Rahmen bevorzugt, um ihre Gemeinsamkeiten zu betonen. Ihre Definition innerhalb der Logik ist jedoch nicht erlaubt. Nicht nur daß die Definition der Verkettungsregeln oder der im folgenden noch zu besprechenden Regeln zur Behandlung von Ableitungen dort nicht möglich wäre, die Definition der Regel zur Behandlung logischer Prädikate wäre unzulässig, da diese nicht in einer wohlgeformten Formel resultiert: Die Argumente des Prädikats sind nach Aggregation selbst logische Formeln und keine Terme mehr in Sinne der mathematischen Logik.

Kehren wir nun zu der Definition der Aggregationsregeln auf der sprachlichen Ebene zurück, indem wir im nächsten Abschnitt Ableitungen betrachten.

Ableitungen Als sehr wirkungsvoll haben sich solche Aggregationsregeln erwiesen, die auf Ableitungen operieren, d. h. auf Upper-Model-Objekten der Kategorien *assume* und *derive*, da sie die Struktur des Textes auf einer höheren Ebene betreffen, als die bisher vorgestellten Regeln zur Zusammenfassung. An dieser Stelle dürfen nur unabhängige Ableitungen betrachtet werden, da nur diese ohne Verlust der Korrektheit des Beweises zusammengefaßt werden dürfen.

Die erste Regel aggregiert die Voraussetzungen einer Ableitung. Diese werden durch UMOe der Kategorie *assume* dargestellt.



Regel 9 *(assume)*
Seien C_i die Konklusionen der i-ten *assume*-UMO für $1 \leq i \leq n$.

$$\frac{\mathit{sequence}\langle(\ldots \mathit{assume}\langle C_1, \epsilon\rangle, \ldots, \mathit{assume}\langle C_n, \epsilon\rangle \ldots)\rangle}{\mathit{sequence}\langle(\ldots \mathit{assume}\langle \mathit{conjunction}\langle(C_1, \ldots, C_n)\rangle, \epsilon\rangle \ldots)\rangle}$$

■

Beispiel: Betrachten wir die folgenden beiden Voraussetzungen:

$sequence\langle(assume\langle Set(F), \epsilon\rangle, \quad assume\langle Set(G), \epsilon\rangle)\rangle$

"Let F be a set. Let G be a set."

Sie werden durch die obige Regel zusammengefaßt zu:

$sequence\langle(assume\langle conjunction\langle(Set(F), Set(G))\rangle\rangle)\rangle$.

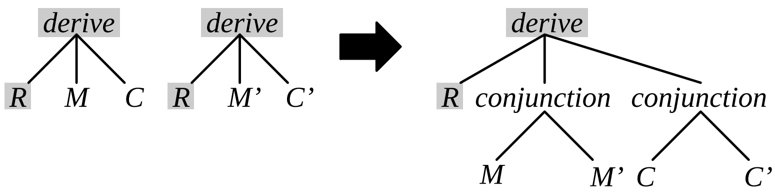
Da *sequence* nun Länge 1 hat, wird es durch sein Argument ersetzt. Die Anwendung der Regel 6 führt zu

$assume\langle Set(conjunction\langle(F, G)\rangle)\rangle$

"Let F and G be sets."

■

Die nächste Regel aggregiert unabhängige Ableitungsschritte, die in ihren Voraussetzungen oder in ihren Methoden übereinstimmen.



Regel 10 *(independent derive)*
Seien R_i die Voraussetzungen, M_i die Methoden und C_i die Konklusionen des i-ten Ableitungsschrittes für $1 \leq i \leq n$, wobei entweder $R_1 = \cdots = R_n$ (kurz: R) oder $M_1 = \cdots = M_n$ (kurz: M) oder beides, jedoch $C_i \notin R_j$ für alle $1 \leq i < j \leq n$.

$$\frac{sequence\langle(\ldots derive\langle R_1, M_1, C_1, \epsilon\rangle, \ldots, derive\langle R_n, M_n, C_n, \epsilon\rangle \ldots)\rangle}{\begin{array}{c}sequence\langle(\ldots derive\langle conjunction\langle(R_1, \ldots, R_n)\rangle, conjunction\langle(M_1, \ldots, M_n)\rangle, \\ conjunction\langle(C_1, \ldots, C_n)\rangle, \epsilon\rangle \ldots)\rangle\end{array}}$$

■

Man beachte, daß die eingangs erwähnte Behandlung null- und einstelliger Konjunktionen gewährleistet, daß die Voraussetzungen bzw. Methoden in R bzw. M zusammenfallen.

Beispiel: Die folgenden beiden Ableitungsschritte werden durch diese Regel aggregiert:

$$\begin{array}{l}sequence\langle(derive\langle conjunction\langle(element\langle a, F\rangle, Subset\langle F, G\rangle)\rangle, \epsilon, element\langle a, G\rangle, \epsilon\rangle, \\ \qquad derive\langle conjunction\langle(element\langle b, F\rangle, Subset\langle F, G\rangle)\rangle, \epsilon, element\langle b, G\rangle, \epsilon\rangle)\rangle\end{array}$$

"Since $a \in F$ and $F \subset G$, $a \in G$. Since $b \in F$ and $F \subset G$, $b \in G$."

Die Zusammenfassung der Ableitungsschritte ergibt:

$$\begin{array}{l}sequence\langle(derive\langle conjunction\langle(element\langle a, F\rangle, element\langle b, F\rangle, Subset\langle F, G\rangle)\rangle, \\ \qquad conjunction\langle(\epsilon)\rangle, \\ \qquad conjunction\langle(element\langle a, G\rangle, element\langle b, G\rangle)\rangle, \\ \qquad \epsilon\rangle)\rangle\end{array}$$

Mehrfache Anwendung der Regel 6 und die Behandlung von null- und einstelligen Konzepten *sequence* und *conjunction* führt zu:

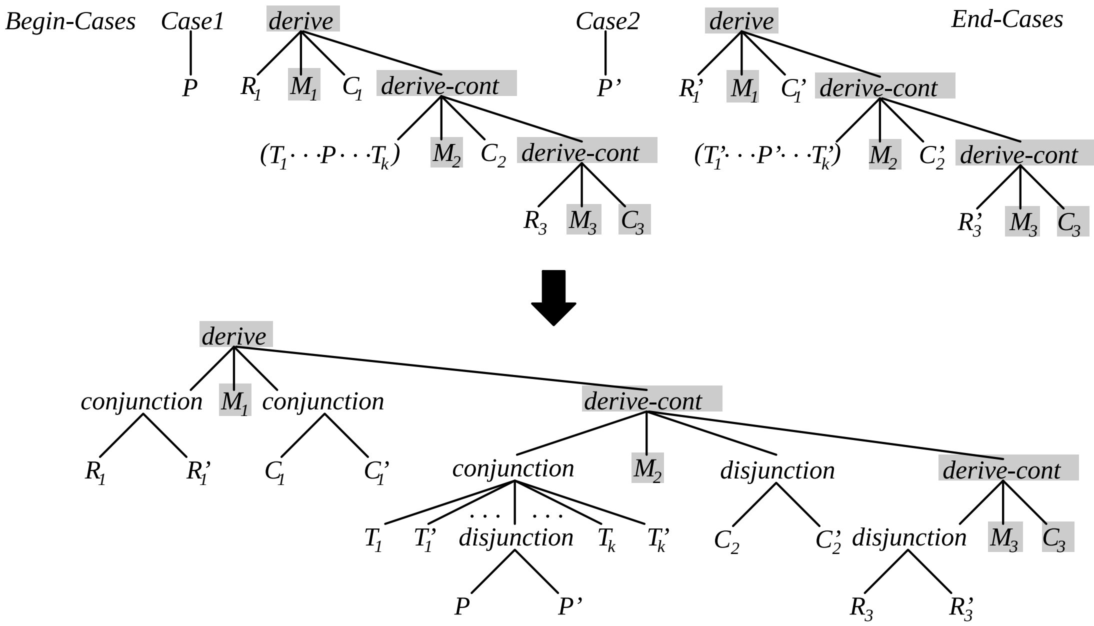
$$\begin{aligned}&\mathit{derive}\langle \mathit{conjunction}\langle(\mathit{element}\langle \mathit{conjunction}\langle(a,b)\rangle, F\rangle, \mathit{Subset}\langle F,G\rangle)\rangle,\\ &\qquad \epsilon,\\ &\qquad \mathit{element}\langle \mathit{conjunction}\langle(a,b)\rangle, G\rangle,\\ &\qquad \epsilon\rangle\end{aligned}$$

"Since $a, b \in F$ and $F \subset G$, $a, b \in G$."

■

Ein beträchtliche Umstrukturierung des Textes erfolgt durch die nächste Regel. Sie faßt Fallunterscheidungen zusammen, deren Fälle ähnlich behandelt werden. Gegenwärtig bedeuted „ähnlich", daß beide Fälle in der Reihenfolge ihrer Methoden übereinstimmen.

Man beachte, daß die Ableitung solange unabhängig von der Fallannahme ist, bis diese benutzt wird. Daher werden die Voraussetzungen und Konklusionen in Konjunktion ausgedrückt, bis die Fallannahmen ins Spiel kommen. Erst dann werden sie disjunktiv verbalisiert.



Regel 11 *(ähnliche Fälle)*
Seien $M_i = M_i'$ für alle $1 \le i \le n$, $C_n = C_n'$ sowie P und P' die Fallannahmen. Ferner seien $P \in R_k$ und $P' \in R_k'$ für ein beliebiges $1 \le k \le n$.

$$\frac{\begin{aligned}&\mathit{sequence}\langle(\ldots \mathit{Begin\text{-}Cases},\\ &\qquad \mathit{Case1}\langle P\rangle, S_1\langle\Phi_1, \ldots S_n\langle\Phi_n, \epsilon\rangle\ldots\rangle,\\ &\qquad \mathit{Case2}\langle P'\rangle, S_1\langle\Phi_1', \ldots S_n\langle\Phi_n', \epsilon\rangle\ldots\rangle,\\ &\qquad \mathit{End\text{-}Cases}\ldots)\rangle\end{aligned}}{\mathit{sequence}\langle(\ldots S_1\langle\Phi_1'', \ldots S_n\langle\Phi_n'', \epsilon\rangle\ldots\rangle\ldots)\rangle}$$

wobei

- $S_1 \in \{\mathit{assume}, \mathit{derive}\}$
- $\Phi_1 = \begin{cases} C_1 & \text{falls } S_1 = \mathit{assume} \\ R_1, M_1, C_1 & \text{falls } S_1 = \mathit{derive}\end{cases}$ und

  $\Phi_1' = \begin{cases} C_1' & \text{falls } S_1 = \mathit{assume} \\ R_1', M_1', C_1' & \text{falls } S_1 = \mathit{derive}\end{cases}$

- für alle $1 < i \leq n$ gilt $S_i = \textit{derive-cont}$, $\Phi_i = R_i, M_i, C_i$ und $\Phi'_i = R'_i, M'_i, C'_i$ und $\Phi''_i = R''_i, M''_i, C''_i$ mit

  – $R''_i = \begin{cases} \textit{conjunction}\langle(R_i, R'_i)\rangle & \text{falls } 1 \leq i < k \\ \textit{conjunction}\langle(\mathrm{R}_i[\epsilon/P], R'_i[\epsilon/P'], \\ \qquad\qquad \textit{disjunction}\langle(P, P')\rangle)\rangle & \text{falls } i = k \\ \textit{disjunction}\langle(R_i, R'_i)\rangle & \text{falls } k < i \leq n \end{cases}$

  – $M''_i = M_i$ für alle $1 \leq i \leq n$

  – $C''_i = \begin{cases} \textit{conjunction}\langle(C_i, C'_i)\rangle & \text{falls } 1 \leq i < k \\ \textit{disjunction}\langle(C_i, C'_i)\rangle & \text{falls } k \leq i < n \\ C_n & \text{falls } i = n \end{cases}$

■

Man beachte, daß, falls $S_1 = \textit{assume}$, R_1, R'_1, M_1 und M'_1 nicht existieren und somit auch nicht R''_1 und M''_1. Das folgende Beispiel veranschaulicht diese recht komplizierte Regel:

Beispiel:

> *Begin-Cases*,
> $\textit{Case1}\langle \textit{el}(a, F_1)\rangle, \textit{derive}\langle(\textit{el}(a, F_1), \textit{Subset}(F_1, G)), \textit{DefSubset}, \textit{el}(a, G), \epsilon\rangle$,
> $\textit{Case2}\langle \textit{el}(a, F_2)\rangle, \textit{derive}\langle(\textit{el}(a, F_2), \textit{Subset}(F_2, G)), \textit{DefSubset}, \textit{el}(a, G), \epsilon\rangle$,
> *End-Cases*
>
> "Let us consider the first case, where $a \in F_1$. Since $a \in F_1$ and $F_1 \subseteq G$, $a \in G$ by the definition of subset. Now, let us consider the second case, where $a \in F_2$. Since $a \in F_2$ and $F_2 \subseteq G$, $a \in G$ by the definition of subset."

Diese Sequenz von UMOen wird zu einem einzelnen UMO zusammengefaßt:

$$\begin{aligned} &\textit{derive}\langle \textit{conjunction}\langle(\textit{Subset}(F_1, G), \textit{Subset}(F_2, G), \\ &\qquad\qquad\qquad \textit{disjunction}\langle(\textit{el}(a, F_1), \textit{el}(a, F_2))\rangle)\rangle, \\ &\qquad \textit{DefSubset}, \textit{el}(a, G), \epsilon\rangle \end{aligned}$$

> "Since $F_1 \subseteq G$ and $F_2 \subseteq G$ and either $a \in F_1$ or $a \in F_2$, $a \in G$ by the definition of subset."

■

4.2.3 Die Steuerung der Aggregation

Betrachtet man die Definitionen der Regeln zur Aggregation semantischer Prädikate, so sieht man leicht, daß die sie voneinander unabhängig sind, da ihre Vorbedingungen disjunkt sind. Daher spielt es keine Rolle, in welcher Reihenfolge sie angewandt werden. Dagegen können die Regeln 4 und 5 zur Aggregation semantischer Objekte beide gleichzeitig anwendbar sein und auch mit der Regel 6 zur Zusammenfassung logischer Prädikate in Konflikt geraten. Daher werden in *PROMI* die Regeln, die semantische Prädikate behandeln, vor denen angewandt, die semantische Objekte aggregieren. In dieser Reihenfolge kann eine effizientere Aggregation erwartet werden, da durch die erstgenannten Regeln mehrere semantische Prädikate gleichzeitig zusammengefaßt werden können, während die letztgenannten stets nur zwei semantische Objekte aggregieren.

Bei den Regeln zur Aggregation semantischer Objekte muß sich für die Anwendung einer von beiden entschieden entschieden werden. Im derzeitigen System wird eine der beiden beliebig ausgewählt. Um eine differenziertere Entscheidung treffen zu können wäre Information über die Stellung der Objekte in einer Taxonomie mathematischer Objekte sowie eine Betrachtung des verfeinerter semantischer und pragmatischer Information wie beispielsweise über den Fokus notwendig (vgl. Kapitel 1.3).

Die Verkettungsregeln werden im Anschluß angewandt, da nach der Zusammenfassung unabhängiger Ableitungsschritte prägnantere Ableitungsketten gebildet werden können. Da Regel 11 zur Aggregation von Fallunterscheidungen die vorausgegangene Anwendung der Kettenregel 1 verlangt — in jedem der beiden Fälle darf nur eine einzige Ableitungskette stehen —, wird diese Regel als letzte angewandt.

Kapitel 5

Die Vervollständigung der linguistischen Spezifikation und Layout

In den bisherigen Kapiteln haben wir gesehen, wie aus der Sequenz von PCAs, die von dem Makroplaner als Textplan generiert wurde, zunächst Upper-Model-Objekte generiert wurden und dann, nachdem die Struktur der UMOe durch Sortierung und Aggregation verbessert wurde, sukzessive durch Einsetzen von Expansionsbäumen die Textstruktur erzeugt wurde. Diese kann allerdings in der Form noch nicht als vollständige linguistische Spezifikation dienen, da einige Entscheidungen noch offenstehen. So sind die ASK-Mengen nicht in allen Knoten einelementig. Auch die konkrete Wortwahl wurde noch nicht getroffen und die Entscheidungen auf Phrasenebene wurden noch nicht gefällt. Ferner ist es noch wünschenswert, die Kohäsion des Textes durch Einfügen von Schlüsselwörtern weiter zu steigern.

Aus diesen Gründen erfolgt eine Traversierung der soeben erzeugten Textstruktur, um die linguistische Spezifikation des Textes zu vervollständigen. Der Durchlauf erfolgt von der Wurzel zu den Blättern und von links nach rechts. Die Beschreibung der genannten Operationen erfolgt in den nächsten Abschnitten, bevor im anschließenden Kapitel der vollständige Mikroplanungsalgorithmus vorgestellt wird.

5.1 Die Auswahl der abstrakten syntaktischen Kategorie

In den Knoten der Expansionsbäume wird in der Regel eine Auswahl an abstrakten syntaktischen Kategorien in Form von ASK-Mengen angeboten. Wie in Kapitel 3.6 bereits dargestellt wurde, kann ein Blatt nur durch einen solchen Expansionsbaum erweitert werden, dessen ASK-Menge der Wurzel mit der ASK-Menge des zu erweiternden Blattes einen nicht-leeren Schnitt bildet. Um die Konsistenz zu erhalten, wird bei Auswahl einer konkreten ASK in einem Knoten die ASK-Menge der von ihm abhängigen Knoten weiter eingeschränkt.

Die Traversierung der Textstruktur von der Wurzel zu den Blättern und von links nach rechts garantiert, daß die ASK eines Knotens vor den ASKn der von ihm abhängigen Knoten ausgewählt wird, so daß Konsistenz stets gewährleistet werden kann.

Da die Konsistenzeigenschaft immer erfüllt ist, kann lokal eine beliebige ASK aus der ASK-Menge ausgewählt werden. Wie im folgenden dargelegt wird, erfahren die Fälle, in denen sentence in der ASK-Menge liegt, eine Sonderbehandlung.

Die Einteilung in Sätze Die Auswahl von sentence entspricht einer Auftrennung in zwei Sätze. Fällt die Wahl an einem Knoten auf sentence, so wird der Unterbaum, der diesen Knoten zur Wurzel hat, unter die Wurzel der Textstruktur transponiert, so daß in der vollständigen linguistischen Spezifikation jeder Unterbaum, der einen Satz repräsentiert, ein Kind der Wurzel ist.

Die Transposition erfolgt so, daß der als erstes transponierte Unterbaum unmittelbar hinter dem Sohn der Wurzel zu liegen kommt, der auf dem Pfad von diesem Knoten zur Wurzel liegt. Später transponierte Unterbäume werden in der Reihenfolge ihrer Transposition dahinter eingesetzt. In Abbildung 5.1 werden die Unterbäume (1) und (2) in dieser Reihenfolge wie durch die Pfeile markiert transponiert.

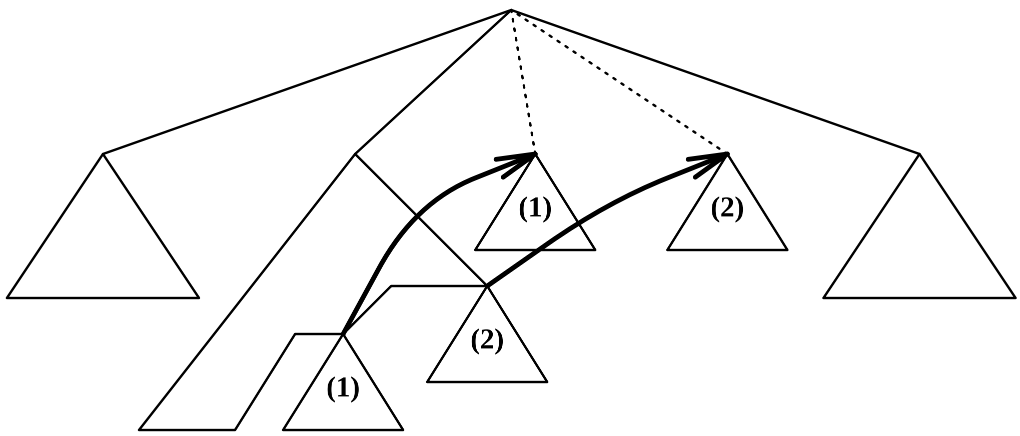
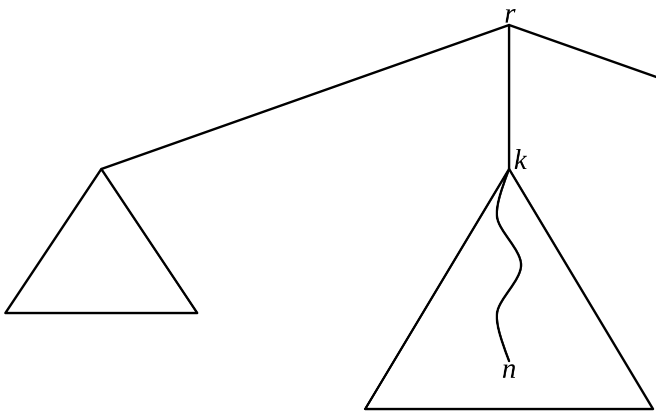


Abbildung 5.1: Die Transposition zweier Unterbäume der Textstruktur

Um die Entscheidung zu treffen, ob sentence ausgewählt werden soll, stehen zur Zeit zwei Heuristiken S_1 und S_2 zur Verfügung, die sich nur in ihren Evaluationsfunktionen s_1 und s_2 unterscheiden. Während S_1 Sätze nach Überschreiten einer bestimmten Länge an der nächstmöglichen Stelle auftrennt, beschränkt S_2 die Schachtelungstiefe der Sätze. Die beiden Heuristiken werden disjunkt behandelt.

Definition der Heuristiken S_1 und S_2 Sei n der aktuelle Knoten der Textstruktur, an dem die Auswahl von sentence zur Entscheidung steht. Sei r die Wurzel der Textstruktur und k das Kind von r auf dem Pfad von r nach n. Wegen der Konsistenzeigenschaft bildet k die Wurzel des Unterbaums, der den aktuellen Satz repräsentiert. Die Situation ist also die folgende:



Sei nun $i \in \{1, 2\}$ und t_i ein vorgegebener Schwellenwert für die Heuristik S_i. Die Heuristik bestimmt anhand ihrer Evaluationsfunktion s_i den Wert $s_i(n)$ des aktuellen Knotens n.

- $s_1(n)$ ist die Anzahl der in dieser Traversierung seit k besuchten Knoten, die Worte als Objekte besitzen, und bildet daher ein Maß für die bisherige Satzlänge.
- $s_2(n)$ ist die Anzahl der Knoten auf dem Pfad von k nach n, die als ASK eine embedded-Kategorie besitzen, und stellt daher, da embedded-Kategorien Nebensätze markieren, ein Maß für die Schachtelungstiefe der Sätze dar.

Ist $s_i(n) > t_i$, so wird als ASK sentence ausgewählt, andernfalls wird eine beliebige andere ASK ausgewählt.

5.2 Die Einfügung von Schlüsselwörtern

In Diskursen zwischen Menschen oder von Menschen verfaßten Texten kommen sehr oft Ausdrücke wie „außerdem“, „aber“ oder „daher“ vor, die die Kohäsion des Textes erhöhen. Reichman nennt solche Ausdrücke ***Schlüsselwörter*** *(clue words)* [Reichman, 1985]. Zukerman stellt diese Ausdrücke in einen größeren Rahmen und nennt sie ***Metakommentare*** *(meta-comments)* [Zukerman, 1991]. Sie definiert Metakommentare als Ausdrücke, die zum Verständnis zwar nicht notwendig sind, es aber doch erleichtern. Daher umfassen Metakommentare auch Phrasen wie „wie bereits erwähnt“ oder „betrachten wir nun“ und sogar ganze Sätze.

Man kann Metakommentare danach unterscheiden, ob sie dem Leser oder Hörer die globale oder die lokale Struktur des Diskurses vermitteln. Das Einfügen globaler Metakommentare fällt in den Aufgabenbereich des Makroplaners. Zu ihnen gehören PCAs, die Beginn oder Ende attentionaler Einheiten markieren, sowie PCAs, die bestimmte Schritte motivieren, wie z. B. “To prove F, we must ...”. Dem Mikroplaner dagegen obliegt das Einfügen von Schlüsselwörtern, die lokal die Kohäsion des Textes erhöhen und so das Verständnis erleichtern.

Der Mikroplaner in *PROVERB* fügt bislang folgende Schlüsselwörter ein:

- Zu Beginn von Ableitungsketten wird “furthermore” oder “moreover” eingesetzt, soweit es sich nicht um den ersten Satz einer attentionalen Einheit handelt.

- In den ersten Satz einer attentionalen Einheit, die auf eine gleichrangige attentionale Einheit folgt (wie beispielsweise die Einleitung des zweiten Falls in einer Fallunterscheidung), wird "now" oder "next" eingefügt.

- Sätze, die Teil einer Ableitungskette darstellen, nicht aber deren Beginn, werden mit "then", "therefore", "thus" oder "hence" eingeleitet.

- In mehrstelligen Prädikaten (in logischem Sinne) mit unterschiedlichen Argumenten, die durch die Aggregationsregel 6 zusammengefaßt wurden, werden die Argumentbeziehungen mit "respectively" präzisiert, wie z. B. in "x and y are elements of X and Y, respectively."

Man beachte, daß Schlüsselwörter stets in konsistenter Weise als Adjunkte in unterordnenden Teilbäumen eingesetzt werden.

5.3 Wortwahl und Auswahlheuristik

Eine Lexikonauswahl im eigentlichen Sinne ist in *PROMI* nicht notwendig, da die Expansionsbäume nur lexikale Synonyme erlauben. Also ist hier nur die Wahl eines Synonyms erforderlich. Zu diesem Zwecke genügt ein sehr einfaches Lexikon, das die Synonyme in einer Liste verwaltet. Den Synonymen können analog zu den UMVn und Expansionsbäumen Prioritäten zugeordnet werden, die zur Auswahlentscheidung herangezogen werden.

Die Auswahl des zu äußernden Synonyms erfolgt durch die Anwendung einer Heuristik, die im folgenden dargestellt wird. Sie dient auch der Auswahl der Upper-Model-Objekte und der Expansionsbäume. Ob deren gemeinsame Behandlung durch eine einzige Heuristik ausreichend ist und ob diese Heuristiken die Erwartungen erfüllt, muß noch untersucht werden.

Definition der Heuristik H *PROMI* vermerkt in einem Diskursgedächtnis alle bereits getätigten Auswahlentscheidungen mit dem ausgewähltem Objekt, sei es ein UMO, ein Expansionsbaum oder ein Synonym. Dieses Diskursgedächtnis wird bei jeder Wahl so mit einer S-förmigen Kurve (einer sogenannten Sigmoidkurve) überlagert, die den einzelnen Einträgen verschiedene Gewichte verleihen, daß länger zurückliegende Einträge höhere Gewichte erhalten.

Die Gewichte der Einträge der zur Wahl stehenden Objekte werden zu n aufsummiert. Der Index eines Objektes bezeichnet den Quotienten $\frac{P}{n+1}$, wobei die P für die gewünschte relative Häufigkeit der Auswahl des jeweiligen Objektes steht, die in der Priorität angegeben wird. Wurde für das Objekt keine Priorität angegeben, wird P auf einen vorgegebenen Wert gesetzt.

Das Objekt mit dem höchsten Index wird ausgewählt. Existieren mehrere Objekte mit höchstem Index, so wird entschieden, welche Objekte unter ihnen die höchsten gewünschte Häufigkeiten besitzen. Sind es mehrere, fällt die Wahl auf ein beliebiges dieser Objekte.

5.4 Entscheidungen auf der Phrasenebene

Sind die endgültigen Worte nun ausgewählt, fehlen nur noch die Entscheidungen auf der Phrasenebene zur vollständigen linguistischen Spezifikation des Textes (vgl. Kapitel 1.3). Die Domäne mathematischer Texte ist eingeschränkt genug, hier sehr einfache Verfahren zu erlauben.

- Die Sätze werden stets aktivisch und im *simple present* geäußert.
- Annahmen werden stets in Imperativsätzen geäußert.
- Artikel werden wie folgt ausgewählt:
  - Namen und Pronomina stehen stets ohne Artikel.
  - Nomina, die in Argumentknoten auftreten tragen indefinite Artikel, sonst definite.

5.5 Layout

Analysiert man Beweise in mathematischen Lehrbüchern, so findet man nur selten ungegliederten Text. Die meisten Beweise zeigen ein optisches Layout: es gibt Absätze, Einrückungen, Zentrierungen und verschiedene Schrifttypen. Das Schriftsatzsystem LaTeX gibt einem bequem die Möglichkeit, das Layout automatisch durch Angabe von Layoutdirektiven im Text zu steuern. Von dieser Möglichkeit macht auch *PROMI* Gebrauch, indem er in die Knoten der Textstruktur Steuerbefehle integriert, die vor bzw. nach dem Unterbaum im jeweiligen Knoten realisiert werden, oder sogar spezielle Layoutknoten in die Textstruktur einfügt. Zur Zeit erfolgen nur die wichtigsten grundlegenden Layoutoperationen.

Der Makroplaner von *PROVERB* strukturiert bereits den gesamten Beweisdiskurs in einer sogenannten attentionalen Hierarchie, in der jede attentionale Einheit einen inhaltlich bedeutungstragenden Teilbeweis darstellt. Es ist sinnvoll solche attentionale Einheiten durch Absätze hervorzuheben.

Der Makroplaner markiert den Beginn und das Ende einer attentionalen Einheit durch spezielle PCAs, die als Metakommentare zu werten sind. Diese werden von dem Mikroplaner über eigene UMOe in Layoutknoten der Textstruktur abgebildet. Ein Layoutknoten, der den Beginn einer attentionalen Einheit markiert, enthält als Steuerbefehl `\newline\newline`, der in LaTeX einen Absatz erzeugt. Layoutknoten, die Beginn und Ende eines Theorems oder eines Beweises markieren, enthalten Befehle, die entsprechende Umgebungen erzeugen.

Eine Beweisstruktur, die häufig durch Layout hervorgehoben wird, ist die Fallunterscheidung, die oft unter Aufzählung der Fälle dargestellt wird. LaTeX bietet als Umgebung zur Aufzählung eine Befehlsfolge an, in der die einzelnen Aufzählungspunkte mit `\item` gekennzeichnet werden und alle Aufzählungspunkte durch `\begin{enumerate}` und `\end{enumerate}`[1] geklammert sind.

[1] In den alternativ angebotenen Umgebungen (`itemize` und `description`) erfolgt die Hervorhebung nicht durch numerische Aufzählung, sondern durch graphische Markierung (in `itemize`) oder durch Voranstellung eines fettgedruckten Textes (in `description`)

Der Mikroplaner veranlaßt auch die mathematische Schreibweise durch Symbole, die durch Klammerung durch die Befehle `\(` und `\)` in mathematischen Schrifttypen gesetzt werden. Diese Klammerung wird bereits in den Zeichenketten für Symbole im Lexikon vorgenommen. Formeln lassen sich durch die Klammerung mit `\[` und `\]` im Text absetzen. Dies geschieht dadurch, daß in den Expansionsbäumen für Formeln diese Klammerung durch Einfügung der Layoutdirektiven im Wurzelknoten erfolgt.

So wie es viele stilistische Variationen für Texte gibt, gibt es auch Variationen im Layout. Noch ist allerdings unklar, wie diese sich gegenseitig beeinflußen. Daher muß das hier vorgestellte Verfahren noch verfeinert werden.

Kapitel 6

Der Mikroplanungsalgorithmus

In diesem Kapitel wird nun der Mikroplanungsalgorithmus vorgestellt und anhand eines kleinen Beispiels erläutert.

Die Aufgabe des Mikroplaners ist es, aus einem Textplan, den der Makroplaner liefert, die linguistische Spezifikation des Textes zu generieren, aus der der syntaktische Realisierer den Oberflächentext erzeugen kann.

Der Makroplaner in *PROVERB* liefert eine Folge von PCAs, die den Textplan darstellt [Huang, 1994b]. Die erste Komponente des *PROMI*, DRCC, trifft die Referenzierungsentscheidungen bezüglich der Zwischenergebnisse des Beweises und seinen Inferenzmethoden, und bildet dadurch die Folge der PCAs auf eine Folge von PMs ab. Der Textstrukturgenerator TSG stellt aus der Folge von PMs eine Textstruktur als linguistische Spezifikation des Textes her, die anschließend auf kanonische Art und Weise in die Eingabe des syntaktischen Realisierers TAG-GEN transformiert wird. TAG-GEN erzeugt dann einen durch Layout-Kommandos angereicherten Text, der von dem Schriftsatzsystem LaTeX zu einem Dokument verarbeitet wird.

6.1 Der Algorithmus des TSG

Eingabe eine Folge vom PMs

Ausgabe eine Textstruktur als vollständige linguistische Spezifikation eines Textes

Teil 1 Der Aufbau der Textstruktur

1. Initialisierung:
   Die initiale Textstruktur besteht aus einem einzelnen Knoten mit dem UMO als Objekt, auf das die Folge der PMs (also ein APO) abgebildet wurde.
2. Terminierung:
   `if` das Objekt des aktuellen Knotens ist ein lexikalisches Objekt
   `then stop`
3. Sortierung und Aggregation:
   wende Sortier- und Aggregationsregeln auf das Objekt des aktuellen Knotens an.

4. Abbildung in das Upper Model:

   `if` das Objekt des aktuellen Knotens ist ein UMO, an dessen Wurzel ein UMV steht

   then wähle unter Berücksichtigung von Typrestriktionen und des Prinzip des *casting* bzw. nach Heuristik H für diese UMV ein passendes UMO aus

5. Expansion:

   wähle unter Berücksichtigung des Prinzips des *casting* bzw. nach Heuristik H einen konsistent einfügbaren Expansionsbaum aus und expandiere den aktuellen Knoten unter Propagierung der Typrestriktionen des Konzeptes an der Wurzel des UMOs

6. Rekursion:

   `for all` Blätter unterhalb des aktuellen Knotens (von links nach rechts)
   `do` das Blatt wird zum neuen aktuellen Knoten; `goto` Schritt 2

Teil 2 Die Vervollständigung der linguistischen Spezifikation

1. Initialisierung:

   mache die Wurzel der Textstruktur zum aktuellen Knoten

2. Auswahl der ASK:

   wähle eine ASK aus der ASK-Menge des aktuellen Knotens aus
   `if` `sentence` gewählt
   `then` transponiere den Teilbaum; `stop`

3. Einfügung von Schlüsselwörtern:

   `if` Einfügung von Schlüsselwörtern möglich
   `then` füge den Kindern des aktuellen Knotens einen weiteren Knoten hinzu, der ein Schlüsselwort repräsentiert.

4. Wortwahl:

   `if` Objekt des aktuellen Knotens $\neq$ `nil`
   `then` wähle nach Heuristik H ein konkretes Wort aus dem Lexikon aus.

5. Entscheidungen der Phrasenebene:

   Treffe die Entscheidungen auf der Phrasenebene.

6. Terminierung:

   `if` der aktuelle Knoten ist ein Blatt
   `then stop`

7. Rekursion:

   `for all` Kindern des aktuellen Knotens (von links nach rechts)
   `do` das Kind wird zum aktuellen Knoten; `goto` Schritt 2

6.2 Ein Laufzeitbeispiel

An dieser Stelle soll der Algorithmus des TSG anhand des Beweises eines kleinen Beispielsatzes erläutert werden. Der Satz besagt, daß zwei Elemente a und b einer Menge F auch in der Obermenge G von F liegen.

Ausgehend von der folgenden PM-Sequenz:

```
(Proof)
(Assume Conclusion: (SET F))
(Assume Conclusion: (SET G))
(Assume Conclusion: (SUBSET F G))
(Assume Conclusion: (ELEMENT A F))
(Assume Conclusion: (ELEMENT B F))
(Derive Reasons: ((ELEMENT A F) (SUBSET F G))
        Conclusion: (ELEMENT A G))
(Derive Reasons: ((ELEMENT B F) (SUBSET F G))
        Conclusion: (ELEMENT B G))
(QED)
```

wird zunächst die initiale Textstruktur erzeugt, deren einziger Knoten folgendes UMO als Objekt besitzt:

$$\begin{aligned}
&\mathit{sequence}\langle(\mathit{proof},\\
&\qquad \mathit{assume}\langle \mathit{set}\langle F\rangle, \epsilon\rangle,\\
&\qquad \mathit{assume}\langle \mathit{set}\langle G\rangle, \epsilon\rangle,\\
&\qquad \mathit{assume}\langle \mathit{subset}\langle F, G\rangle, \epsilon\rangle,\\
&\qquad \mathit{assume}\langle \mathit{element}\langle a, F\rangle, \epsilon\rangle,\\
&\qquad \mathit{assume}\langle \mathit{element}\langle b, F\rangle, \epsilon\rangle,\\
&\qquad \mathit{derive}\langle \mathit{conjunction}\langle(\mathit{element}\langle a, F\rangle, \mathit{subset}\langle F, G\rangle)\rangle, \epsilon, \mathit{element}\langle a, G\rangle, \epsilon\rangle,\\
&\qquad \mathit{derive}\langle \mathit{conjunction}\langle(\mathit{element}\langle b, F\rangle, \mathit{subset}\langle F, G\rangle)\rangle, \epsilon, \mathit{element}\langle b, G\rangle, \epsilon\rangle,\\
&\qquad \mathit{qed})\rangle
\end{aligned}$$

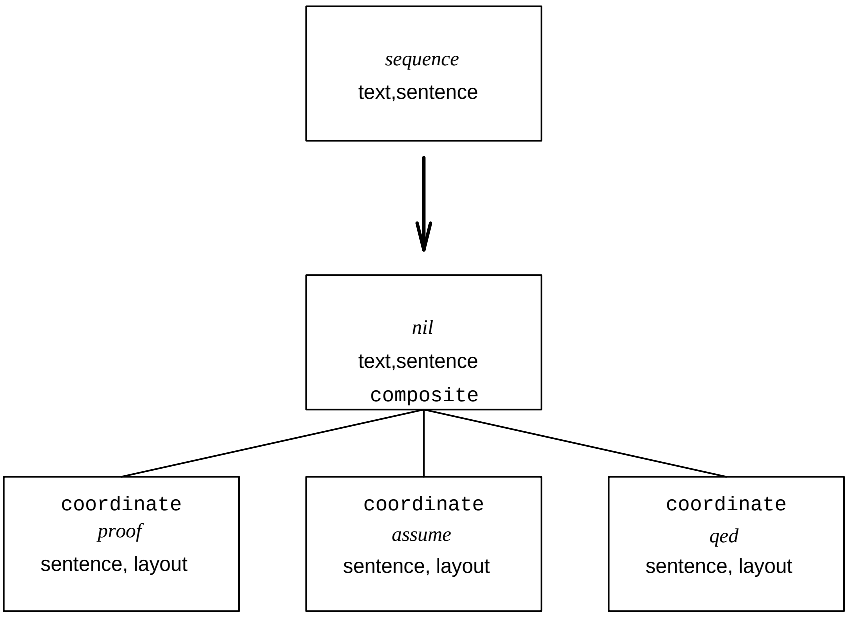
Man beachte, daß das erste Argument von *assume* für die Annahme steht, das zweite für den nächsten Schritt in einer Ableitungskette. Analog steht das erste Argument von *derive* für die Voraussetzungen des Ableitungsschrittes stehen, das zweite Argument für die Inferenzmethode (die hier der Makroplaner nicht zur Verbalisierung zuläßt), das dritte Argument für die Konklusion und das vierte für den nächsten Ableitungsschritt in einer Kette. Anwendungen der Aggregationsregeln 9 und 10 fassen die Annahmen bzw. die voneinander unabhängigen Ableitungsschritte zusammen:

$$\begin{aligned}
&\mathit{sequence}\langle(\mathit{proof},\\
&\qquad \mathit{assume}\langle\, \mathit{conjunction}\langle(\mathit{set}\langle F\rangle, \mathit{set}\langle G\rangle, \mathit{subset}\langle F, G\rangle,\\
&\qquad\qquad\qquad\qquad \mathit{element}\langle a, F\rangle, \mathit{element}\langle b, F\rangle)\rangle,\\
&\qquad\qquad \epsilon\rangle,\\
&\qquad \mathit{derive}\langle\, \mathit{conjunction}\langle(\mathit{element}\langle a, F\rangle, \mathit{element}\langle b, F\rangle, \mathit{subset}\langle F, G\rangle)\rangle,\\
&\qquad\qquad \epsilon,\\
&\qquad\qquad \mathit{conjunction}\langle(\mathit{element}\langle a, G\rangle, \mathit{element}\langle b, G\rangle)\rangle,\\
&\qquad\qquad \epsilon\rangle,\\
&\qquad \mathit{qed})\rangle
\end{aligned}$$

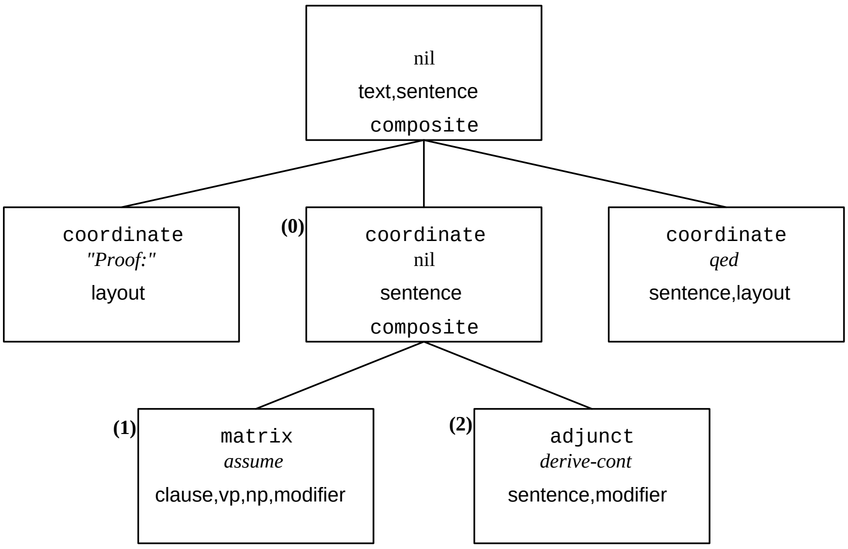
Die Ableitungsschritte werden anschliessend durch die Regel 1 verkettet:

$$
\begin{array}{l}
\textit{sequence}\langle(\textit{proof},\\
\quad\quad \textit{assume}\langle\, \textit{conjunction}\langle(\textit{set}\langle F\rangle, \textit{set}\langle G\rangle, \textit{subset}\langle F, G\rangle,\\
\quad\quad\quad\quad\quad\quad\quad\quad \textit{element}\langle a, F\rangle, \textit{element}\langle b, F\rangle)\rangle,\\
\quad\quad\quad\quad \textit{derive-cont}\langle\epsilon, \epsilon,\\
\quad\quad\quad\quad\quad\quad\quad \textit{conjunction}\langle(\textit{element}\langle a, G\rangle, \textit{element}\langle b, G\rangle)\rangle,\\
\quad\quad\quad\quad\quad\quad\quad \epsilon\rangle\rangle,\\
\quad\quad \textit{qed})\rangle
\end{array}
$$

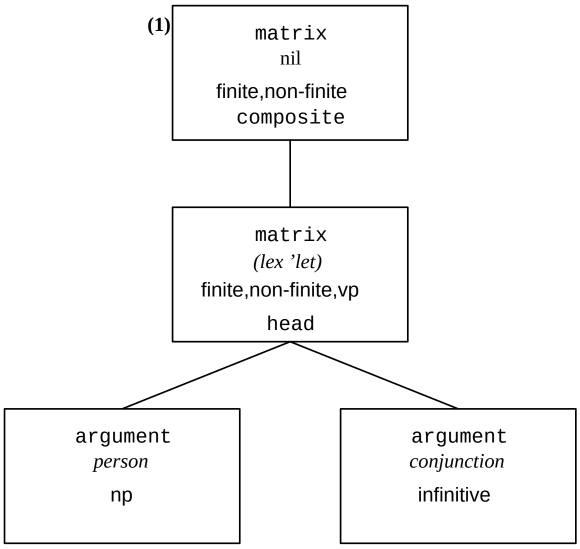
Da dieses UMO das Objekt des initialen Textstrukturknotens ist, wird er nach Abbildung der UMV *sequence* auf das UMK *sequence* durch dessen einzigen Expansionsbaum erweitert:



In den Knoten der Textstruktur dieses Beispiels werden nur die Wurzeln der UMOe dargestellt. Nun werden die Blätter von links nach rechts erweitert. Das erste Blatt expandiert zu einem einzelnen Layout-Knoten, der nicht weiter verfolgt werden kann. Das nächste Blatt hat die Ableitungskette zum Objekt, die nicht aggregiert werden kann. So wird $assume\langle x, y\rangle$ zunächst auf $\textit{modified-concept}\langle assume\langle x\rangle, y\rangle$ abgebildet, zu dem es nur einen Expansionsbaum gibt. Man beachte, daß in dem neuen UMO das UMV *assume* nunmehr einstellig ist. Die Expansion des Blattes liefert:



Der Algorithmus fährt nun im Blatt (1) fort. Da keine Aggregation möglich ist, wird $assume\langle x\rangle$ abgebildet auf $assume\langle we, x\rangle$. Auch zu *assume* existieren zwei Expansionsbäume. Der Expansionsbaum mit dem höchsten errechneten Index expandiert das Blatt (1) expandiert zu:



Man beachte, daß im Knoten (1) die ASK-Menge auf {finite, non-finite} eingeschränkt wurde. Das hat die Einschränkung der ASK-Menge von Blatt (2), dem Adjunkten zu (1), auf {sentence, clause-modifier} zur Folge. Der Algorithmus schreitet nun wiederum zu den Blättern fort. Die Expansion des linken Blattes resultiert in der Ersetzung seines Objektes durch ein entsprechendes lexikales Objekt. Das rechte Blatt trägt als Objekt

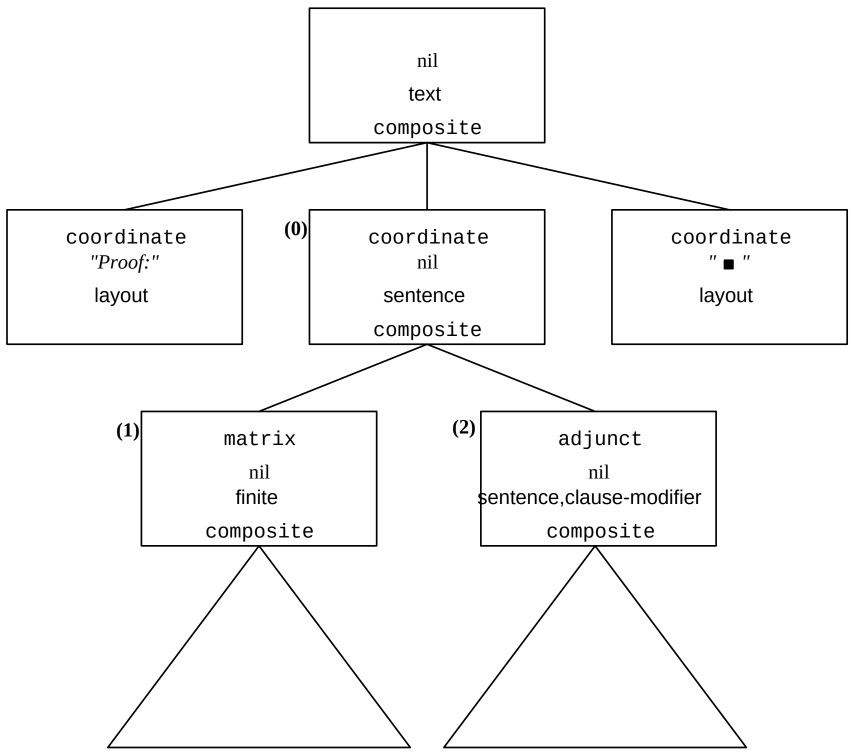
$conjunction\langle(set\langle F\rangle, set\langle G\rangle, subset\langle F, G\rangle, element\langle a, F\rangle, element\langle b, F\rangle)\rangle,$

so daß an dieser Stelle wieder aggregiert werden kann. Zwei Anwendungen der Regel 6 aggregieren die Prädikate *set* und *element* und resultieren daher in:

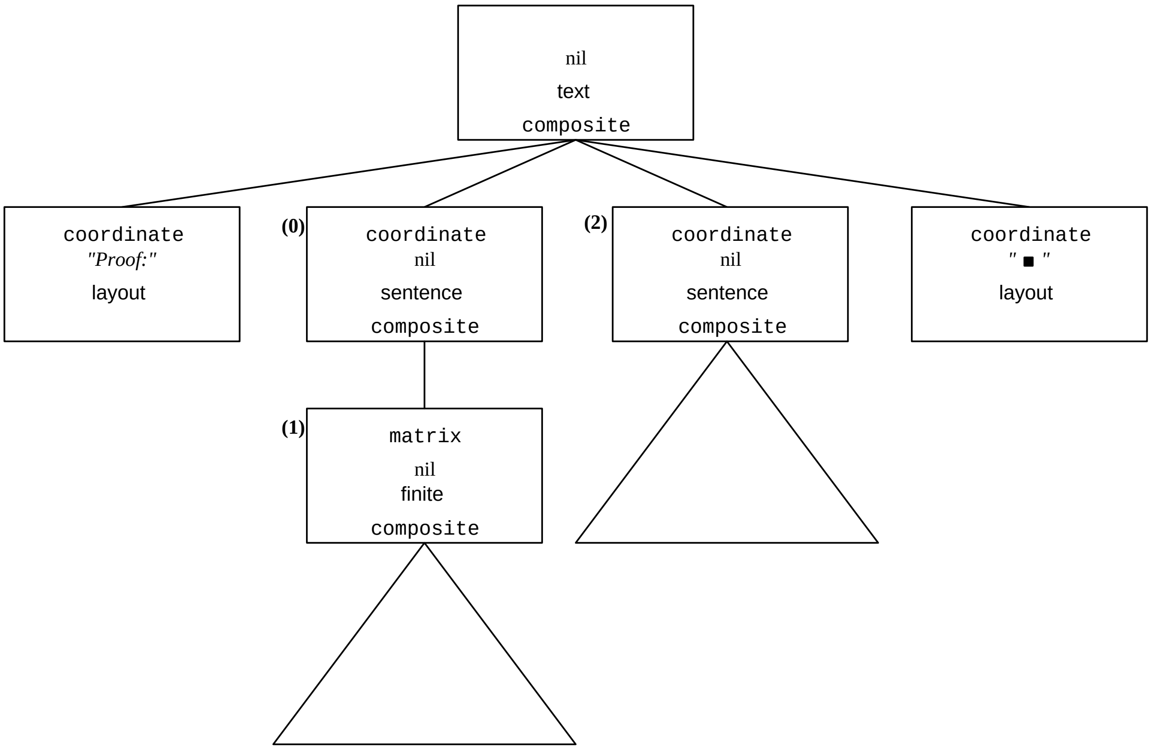
$$\begin{aligned} conjunction\langle(&set\langle conjunction\langle(F, G)\rangle\rangle,\\ &subset\langle F, G\rangle\\ &element\langle conjunction\langle(a, b)\rangle, F\rangle)\rangle \end{aligned}$$

Man beachte, daß bei der Zusammenfassung von *element* als zweites Argument zunächst $conjunction\langle(F,F)\rangle$ erzeugt wird, was aber zu F reduziert wird. Dieses UMO läßt nun keine weitere Aggregation mehr zu. Daher wird auch dieses Blatt nun expandiert.

Nun, da die Arbeitsweise der Erzeugung der Textstruktur klar geworden ist, wollen wir diesen Teil des Algorithmus nicht weiter erklären, sondern zur Vervollständigung der Textstruktur übergehen. Da die Vorgehensweise dieses zweiten Teils des Algorithmus sehr einfach ist, wollen wir nur die Transposition von Teilbäumen und die Einfügung von Schlüsselwörtern näher betrachten. Wir steigen daher erst zu dem Zeitpunkt ein, in dem die Traversierung schon zum Knoten (2) vorgedrungen ist:



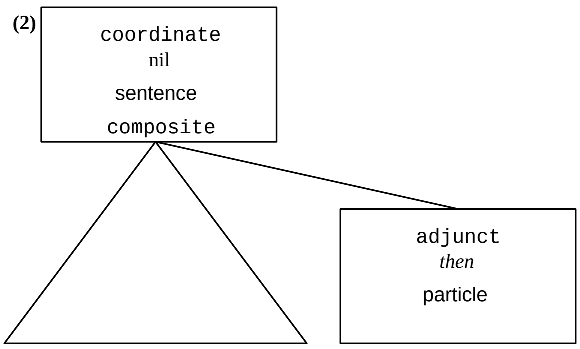
Der Unterbaum mit Wurzel (1) ist also bereits vollständig spezifiziert. Wird nun für den Knoten (2) die ASK sentence ausgewählt, wird der Unterbaum in (2) hinter den aktuellen Satz transponiert:



Man beachte, daß Knoten (2) dadurch von einem Adjunkten zu einer Koordinate wird.

Da somit die Traversierung des Unterbaums des Knotens (0) abgeschlossen ist, fährt der Algorithmus in Knoten (2) fort.

Knoten (2) trägt noch die Information, daß sein ursprüngliches Objekt dem ideationalen Konzept *derive-cont* angehörte. Da es sich hierbei also um einen Satz einer Ableitungskette handelt, der nicht an deren Beginn steht, wird ein Schlüsselwort hinzugefügt. Das geschieht dadurch, daß (2) ein zusätzliches Kind erhält.



Im weiteren Verlauf der Traversierung wird in diesem Kind für das lexikale Objekt *then* anhand der Heuristik H das Wort "therefore" eingesetzt.

So vervollständigt der Algorithmus schließlich die linguistische Spezifikation, aus der der syntaktische Realisierer TAG-GEN folgende (hier bereits durch LaTeX gesetzte) Ausgabe generiert:

Proof:
Let F and G be sets, $F \subset G$ and $a, b \in F$. Therefore $a, b \in G$. ■

Man beachte, daß „*Proof:*“ und „■“ durch LaTeX-Kommandos aus den beiden Layout-Knoten der Textstruktur erzeugt wurden. Der erste Satz des Beweises wird durch den Unterbaum mit der Wurzel (0) in der obigen Textstruktur repräsentiert, während der zweite Satz durch den Unterbaum mit der Wurzel (2) spezifiziert wird.

Betrachten wir noch einmal das Beispiel aus Kapitel refsec:der-mikroplaner-in-proverb:

Theorem:

Let F be a group, U be a subgroup of F and 1 and 1_U be unit elements of F and U respectively. Then $1_U = 1$.

Proof:

(1) Let F be a group, U be a subgroup of F and 1 and 1_U be unit elements of F and U respectively.

(2,3) $1_U \in U$ by the definition of unit element which leads to the existence of an x such that $x \in U$. (4) Let u_1 be such an x.

(5) $u_1 * 1_U = u_1$ by the definition of unit element. (6,7) Since U is a subgroup of F $U \subset F$ which implies $u_1 \in F$. (8) Similarly since $1_U \in U$ $1_U \in F$ (9) Since F is a group F is a semigroup. (10) Since $u_1 * 1_U = u_1$ 1_U is a solution of the equation $u_1 * X = u_1$.

(11,12) Since 1 is a unit element of F $u_1 * 1 = u_1$ and $1 \in F$. (13) Since $u_1 \in F$ 1 is a solution of the equation $u_1 * x = u_1$. (14) Since F is a group $1_U = 1$ by the uniqueness of the solution. (15) This conclusion is independent of the choice of the element u_1. ■

Mit Hilfe der attentionalen Struktur des Textplans gliedert *PROMI* den Beweis in Absätze die jeweils den Beginn einer neuen attentionalen Einheit markieren. Die PCAs, die das Theorem und den Beweis klammern, benutzt er zur Planung des Layouts, indem er sie in LaTeX-Befehle zur Erzeugung der entsprechenden Umgebungen abbildet. Weitere Layoutentscheidungen führten zu der Benutzung mathematischer Symbole bei der Realisierung einiger Formeln.

Durch die Anwendung von Aggregationsregeln verkettete *PROMI* je zwei Ableitungsschritte zu (2,3) und (6,7). (1) entstand aus der Zusammenfassung von vier UMOen der Kategorie *assume*, die die Voraussetzungen des Beweises repräsentierten. Die Aggregation unabhängiger Ableitungsschritte resultierte in (11,12). In (1) wie auch in der Voraussetzung des Theorems wurde zusätzlich die UMOe $unit(1, F)$ und $unit(1_U, U)$ aggregiert. Diese Aggregation wurde später durch das Schlüsselwort “respectively” markiert, um keine Mehrdeutigkeit aufkommen zu lassen.

Kapitel 7

Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war die Entwicklung eines flexiblen Mikroplaners für das Beweispräsentationssystem *PROVERB*, das Beweise im Kalkül des natürlichen Schließens in englischer Sprache verbalisiert.

PROVERB verfolgt hierzu einen mehrstufigen Ansatz, in dem der Beweis zunächst zu einem Beweis der Faktenebene abstrahiert wird, auf der als Beweisschritte Anwendungen von Definitionen, Axiomen oder bereits bewiesenen Sätzen, den sogenannten Fakten, erlaubt sind. Aus dem Faktenbeweis generiert ein Makroplaner eine Folge von PCAs als Textplan, in dem die globale Struktur des Textes festgelegt ist.

Nachdem er die PCAs durch geeignete Referenzentscheidungen in PMs überführt hat, generiert der in dieser Arbeit entwickelte Mikroplaner *PROMI* anhand mehrerer Wissensquellen und Regelsätzen eine Textstruktur, die in ihrer vollständigen Form die linguistische Spezifikation des Textes darstellt. Dies geschieht durch progressive Verfeinerung der PMs als Applikationsprogrammobjekte über Upper-Model-Objekte zur Textstruktur, wobei Techniken zur Sortierung, zur Aggregation und zur Einfügung von Schlüsselwörtern auf die Qualität des Textes verbessert.

Hierzu mußte eine Intergration der Textstruktur mit Ansätzen der systemisch-funktionalen Linguistik unternommen werden. Dabei dienen das Upper Model und die Hierarchie abstrakter syntaktischer Kategorien, die beide auf die Domäne der Beweispräsentation angepaßt wurden, der Typbeschränkung der Knoten der Textstruktur, so daß ihre sprachlich korrekte Verbalisierung stets möglich ist. Ferner wurde eine Klassifikation von elf Aggregationsregel aufgestellt, durch deren Anwendung Redundanz im Text weitgehend vermieden wird. Schließlich wurde die Textstruktur dahingehend erweitert, daß auch die Planung des Layouts des Textes von *PROMI* übernommen werden konnte.

Die linguistische Spezifikation wird von dem syntaktischen Realisierer TAG-GEN als Text verbalisiert, der durch das Schriftsatzsystem LaTeXzu einem Dokument weiterverarbeitet wird.

Um auch Logeleien, die in Alltagssprache formuliert sind, präsentieren zu können, müßste insbesondere das Upper Model umfangreicher gestaltet werden. Ferner wäre eine Erweiterung einiger Regelsätze erforderlich. So spielen in der Alltagssprache Pronominalisierung und Referenzierung eine größere Rolle als in mathematischen Texten. Auch die Behandlung von Artikeln sowie Zeit, Modus und Handlungsrichtung verkompliziert sich. Da die Sprache der mathematischen Beweise weitgehend standardisiert und von eingeschränkter Struktur ist, genügten hier für die Entscheidungen auf der Phrasenebene spezialisierte Regeln, die auf die Domäne mathematischer Texte zugeschnitten sind, um sprachlich gute Texte flexibel zu erzeugen.

Sortierverfahren, die eine Taxonomie der zu sortierenden Objekte in Betracht ziehen, könnten auch für komplizierte mathematische Probleme bereits von Vorteil sein, da taxonomische Information vor anderen Eigenschaften genannt werden sollte. Die Betrachtung von Thematisierung und Fokus würde die Sortierung gemäß der Diskursgegenstände, die augenblicklich im Blickpunkt stehen, erlauben.

Taxonomisches Wissen würde es ferner erlauben, die Beschreibung der Sachverhalte zu verbessern, wie beispielsweise in "Let F and G be sets, where $F \subset G$." Diese Information läßt sich in dem derzeitigen System nur mit "and" verbalisieren.

Die Abstraktion der Layoutdirektiven von LaTeX-Kommandos zu einer allgemeinen Layout-Sprache würde die optische Präsentation des Textes auf verschiedenen Medien erleichtern, so daß einerseits das direkte Layout auf dem Bildschirm möglich wäre, andererseits aber auch andere Schriftsatzsysteme verwendet werden könnten.

Ein Benutzermodell in *PROMI* könnte die Layoutplanung und die Wahl der Verbalisierung auf den jeweiligen Benutzer anpassen.

Anhang A

Upper-Model-Variablen

Upper-Model-Variablen dienen der Abbildung von Objekten in das Upper Model. Im folgenden wird die Eingabesprache zur Definition der UMVn angegeben und eine Bibliothek bereits vordefinierter Variablen aufgeführt. Da diese nur unvollständig sein kann, sollte in der Zukunft eine graphische Eingabe ermöglicht werden.

A.1 Die Definition der Eingabesprache

variable-definition	`::=` `(um~def-variable` *symbol umv deflist**`)`
deflist	`::=` `(`*arity priority definition*`)`
arity	`::=` `(arity` *number*`)`
priority	`::=` `(priority (`*type* . *number*`))`
type	`::=` `sym` | `verb`
definition	`::=` *um-definition* | *umv-definition*
um-definition	`::=` `(um '`*um key-value**`)`
umv-definition	`::=` `(umv '`*umv args**`)`
key-value	`::=` *keyword value*
args	`::=` `:args` *value*
value	`::=` *val* | *lex-definition*
val	`::=` *definition* | *list* | *argument* | *string*
list	`::=` `(list` *val**`)` | `nil`
lex-definition	`::=` `(`*lex-type* `'`*symbol*`)`
lex-type	`::=` `lex` | `sym`
argument	`::=` `?`*number*
keyword	`::=` `:`*symbol*
um	`::=` `um@`*symbol*
umv	`::=` `umv@`*symbol*

A.2 Vordefinierte Upper-Model-Variablen

```
;;; Sequenzen

(um~def-variable sequence umv@concept
  ((arity 1) (priority (verb . 10))
   (um 'um@sequence :domain ?1)))


;;; logische Quantoren und Junktoren

(um~def-variable forall umv@concept
  ((arity 2) (priority (verb . 10))
   (um 'um@universal :variable ?1 :scope ?2)))

(um~def-variable exists umv@concept
  ((arity 2) (priority (verb . 10))
   (um 'um@existential :variable ?1 :scope ?2)))

(um~def-variable conjunction umv@concept
  ((arity 1) (priority (verb . 10))
   (um 'um@conjunction :domain ?1)))

(um~def-variable disjunction umv@concept
  ((arity 1) (priority (verb . 10))
   (um 'um@disjunction :domain ?1)))

(um~def-variable equivalence umv@concept
  ((arity 1) (priority (verb . 10))
   (um 'um@equivalence :domain ?1)))

(um~def-variable implication umv@concept
  ((arity 2) (priority (verb . 10))
   (um 'um@implication :cause ?1 :effect ?2)))

(um~def-variable implication-chain umv@concept
  ((arity 1) (priority (verb . 10))
   (um 'um@implication-chain :domain ?1)))


;;; PCAs

(um~def-variable derivation umv@concept)

(um~def-variable derive umv@derivation
  ((arity 3) (priority (verb . 10))
   (um 'um@derive :reason ?1 :conclusion ?2 :method ?3 :next nil))
  ((arity 4) (priority (verb . 10))
   (um 'um@derive :reason ?1 :conclusion ?2 :method ?3 :next ?4)))
```

```
(um~def-variable derive-cont umv@derive
  ((arity 3) (priority (verb . 10))
   (um 'um@derive-cont :reason ?1 :conclusion ?2 :method ?3
                       :next nil))
  ((arity 4) (priority (verb . 10))
   (um 'um@derive-cont :reason ?1 :conclusion ?2 :method ?3
                       :next ?4)))

(um~def-variable assume-1 umv@derivation
  ((arity 1) (priority (verb . 10))
   (um 'um@assume :senser (um 'um@person :object (lex 'we))
                  :phenomenon ?1
                  :annotations '((intention imperative))))
  ((arity 2) (priority (verb . 10))
   (um 'um@modified-concept :concept (umv 'umv@assume-1
                                          :args (list ?1))
                            :modifier ?2)))

(um~def-variable theorem umv@concept
  ((arity 1) (priority (verb . 10))
   (um 'um@sequence :domain (list (umv 'umv@theo-layout)
                                  (umv 'umv@derive
                                        :args (list nil ?1 nil)))))
  ((arity 2) (priority (verb . 10))
   (um 'um@sequence :domain (list (umv 'umv@theo-layout)
                                  (umv 'umv@assume-1 :args (list ?1))
                                  (umv 'umv@derive-cont
                                       :args (list nil ?2 nil))))))

(um~def-variable case umv@concept
  ((arity 2) (priority (verb . 10))
   (um 'um@enablement
       :domain
       (um 'um@assume
           :sensor (um 'um@person :object (lex 'we))
           :phenomenon
           (um 'um@consider
               :sensor (um 'um@person :object (lex 'we))
               :phenomenon
               (um 'um@modified-concept
                   :concept (um 'um@object :object (lex 'case))
                   :modifier (um 'um@number :quality (lex ?1)))))
       :range (um 'um@assume :phenomenon ?2))))


;;; Methoden

(um~def-variable method umv@concept
  ((arity 1) (priority (verb . 10))
   (um 'um@object :object ?1)))
```

```
;;; Layout

(um~def-variable theo-layout umv@concept
  ((arity 0) (priority (verb . 10))
   (um 'um@layout :before "\\newline\\newline {\\bf Theorem:}"
                  :after "\\newline\\newline")))

(um~def-variable proof umv@concept
  ((arity 0) (priority (verb . 10))
   (um 'um@layout :before "\\newline\\newline {\\em Proof:}"
                  :after "\\newline\\newline")))

(um~def-variable qed umv@concept
  ((arity 0) (priority (verb . 10))
   (um 'um@layout :before "\\newline\\hfill\\box")))

(um~def-variable begin-attention umv@concept
  ((arity 0) (priority (verb . 10))
   (um 'um@layout :before "\\newline\\newline")))

(um~def-variable end-attention umv@concept
  ((arity 0) (priority (verb . 10))
   (um 'um@layout)))
```

Anhang B

Realisationsklassen

Die Realisationsklassen dienen der Abbildung von Upper-Model-Objekten auf die Textstruktur. Zu jedem Konzept der Upper Models sollte eine solche definiert sein, so daß dem Benutzer des Systems die volle Bibliothek zur Verfügung steht. Bedingt durch die Mächtigkeit der Sprache und der Flexibilität der Ausdrucksmöglichkeiten, kann die hier vorgestellte Bibliothek nur eine Annäherung an das Ideal sein.

B.1 Definition der Eingabesprache

```
realization-class ::= (ts~def-realization-class concept resource-tree*)
concept           ::= (um key-variable*)
key-variable      ::= keyword variable
variable          ::= symbol | nil
resource-tree     ::= (resource-tree (declaration*) rt-def)
declaration       ::= mandatory | negation | priority
mandatory         ::= (mandatory symbol*)
negation          ::= (negated neg)
neg               ::= no | yes | symbol
rt-def            ::= (ts-type spec*)
ts-type           ::= kernel | composite | leaf
spec              ::= object | textual | role | child | layout | annotations | negated
object            ::= :object obj
obj               ::= variable | lex-definition | um-definition
textual           ::= :textal tex
tex               ::= '(asc+) | same
asc               ::= asc@symbol
role              ::= :role 'keyword
annotations       ::= :annotations '(symbol+)
negated           ::= :negated neg
layout            ::= layout-type string
layout-type       ::= :layout-before | :layout-after
child             ::= single-child | list-child | map-child
```

single-child ::= `:matrix` *rt-def*
list-child ::= *list-child-type* `(`*rt-def*$^+$`)`
list-child-type ::= `:arguments` | `:adjuncts` | `:coords`
map-child ::= `:coords (map #'(lambda (`*symbol*`)` *rt-def*`)` *symbol*

um-definition, *lex-definition*, *priority* und *keyword* sind wie in Anhang A.1 definiert.

B.2 Die Realisationsklassen in *PROMI*

```
;##              Top Level              ##

(ts~def-realization-class (um@modified-concept :concept C :modifier M)
  (resource-tree ((mandatory C M) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@sentence tsc@clause
                           tsc@vp tsc@np tsc@modifier)
                :matrix
                (leaf :object C
                      :textual '(tsc@clause tsc@vp tsc@np tsc@modifier))
                :adjuncts
                ((leaf :object M
                       :textual '(tsc@sentence
                                  tsc@clause-modifier-rankingI
                                  tsc@clause-modifier-rankingII
                                  tsc@clause-modifier-embedded
                                  tsc@vp-modifier-rankingI
                                  tsc@vp-modifier-rankingII
                                  tsc@vp-modifier-embedded
                                  tsc@np-modifier-rankingI
                                  tsc@np-modifier-rankingII
                                  tsc@np-modifier-embedded
                                  tsc@intensifier))))))

;##               Objects               ##

(ts~def-realization-class (um@object :object O)
  ; O
  (resource-tree ((mandatory O) (negated no) (priority (verb . 10)))
     (leaf :object (lex O)
           :textual '(tsc@np tsc@gerund))))

;##              Processes              ##

;# Relational Processes #

(ts~def-realization-class (um@conjunction :domain D)
  ; (and D)
  (resource-tree ((mandatory D) (negated no) (priority (verb . 10)))
     (composite :object (lex 'and)
                :textual '(tsc@category)
```

```
                :annotations '(conj)
                :coords (map #'(lambda (P)
                                 (leaf :object P
                                       :textual 'same))
                             D))))

(ts~def-realization-class (um@disjunction :domain D)
  ; (or D)
  (resource-tree ((mandatory D) (negated no) (priority (verb .10)))
     (composite :object (lex 'or)
                :textual '(tsc@category)
                :annotations '(conj)
                :coords (map #'(lambda (P)
                                 (leaf :object P
                                       :textual 'same))
                             D))))

(ts~def-realization-class (um@equality :domain D)
  ; (= D)
  (resource-tree ((mandatory D) (negated no) (priority (sym . 10)))
     (composite :object (sym 'equals-to)
                :textual '(tsc@np)
                :annotations '(conj)
                :coords (map #'(lambda (P)
                                 (leaf :object P
                                       :textual 'same))
                             D))))

(ts~def-realization-class (um@equivalence :domain D)
  ; (iff D)
  (resource-tree ((mandatory D) (negated no) (priority (verb . 10)))
     (composite :object (lex 'iff)
                :textual '(tsc@sentence tsc@finite tsc@vp)
                :annotations '(conj)
                :coords (map #'(lambda (P)
                                 (leaf :object P
                                       :textual 'same))
                             D)))
  ; (<=> D)
  (resource-tree ((mandatory D) (negated no) (priority (sym . 10)))
     (composite :object (sym 'iff)
                :textual '(tsc@sentence tsc@finite tsc@vp tsc@np)
                :annotations '(conj)
                :coords (map #'(lambda (P)
                                 (leaf :object P
                                       :textual 'same))
                             D))))

(ts~def-realization-class (um@implication-chain :domain D)
  ; (=> D)
```

```
  (resource-tree ((mandatory D) (negated no) (priority (sym . 10)))
     (composite :object (sym 'imply)
                :textual '(tsc@sentence tsc@finite tsc@vp tsc@np)
                :annotations '(conj)
                :coords (map #'(lambda (P)
                                 (leaf :object P
                                       :textual 'same))
                             D))))

(ts~def-realization-class (um@sequence :domain D)
  ; (Making sequences)
  (resource-tree ((mandatory D) (negated no) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@text)
                :annotations '(conj)
                :coords (map #'(lambda (P)
                                 (leaf :object P
                                       :textual '(tsc@sentence
                                                  tsc@layout)))
                             D)))
  (resource-tree ((mandatory D) (negated no) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@sentence tsc@clause tsc@vp
                           tsc@np tsc@modifier)
                :annotations '(conj)
                :coords (map #'(lambda (P)
                                 (leaf :object P
                                       :textual 'same))
                             D))))

;# Quantifications

(ts~def-realization-class (um@universal :variable V :scope S)
  ; S for all V
  (resource-tree ((mandatory V S) (negated no) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@sentence tsc@finite tsc@np)
                :matrix
                (leaf :object S :textual '(tsc@finite tsc@vp tsc@np))
                :adjuncts
                ((composite :object (lex 'for)
                            :textual '(tsc@clause-modifier-rankingII
                                       tsc@vp-modifier-rankingII
                                       tsc@np-modifier-rankingII)
                            :matrix
                            (composite :object nil
                                       :textual '(tsc@np)
                                       :matrix (leaf :object V
                                                     :textual '(tsc@np))
                                       :adjuncts
```

```
                                        ((leaf :object (lex 'all)
                                               :textual
                                               '(tsc@np-modifier-rankingI
                                                 ))))))))
  ; \forall V S
  (resource-tree ((mandatory V S) (negated no) (priority (sym . 10)))
     (composite :object nil
                :textual '(tsc@sentence tsc@clause tsc@np)
                :matrix
                (kernel :object (sym 'dot)
                        :textual '(tsc@gerund tsc@np)
                        :arguments
                        ((composite :object nil
                                    :textual '(tsc@np)
                                    :matrix
                                    (leaf :object V :textual '(tsc@np))
                                    :adjuncts
                                    ((leaf :object (sym 'forall)
                                           :textual
                                           '(tsc@np-modifier-rankingI))))
                         (leaf :object S :textual '(tsc@np)))))))

(ts~def-realization-class (um@existential :variable V :scope S)
  ; There is a V, such that S
  (resource-tree ((mandatory V S) (negated neg) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@sentence tsc@finite)
                :matrix (kernel :object (lex 'be)
                                :textual '(tsc@vp)
                                :negated neg
                                :arguments ((leaf :object (lex 'there)
                                                  :textual '(tsc@np)
                                                  :role ':agent
                                                  :annotations '(dpron))
                                            (leaf :object V
                                                  :textual '(tsc@np)
                                                  :role ':ident)))
                :adjuncts ((composite :object (lex 'such-that)
                                      :textual '(tsc@vp-modifier-embedded)
                                      :matrix (leaf :object S
                                                    :textual '(tsc@finite
                                                               tsc@vp))))))
  ; S for some V
  (resource-tree ((mandatory V S) (negated no) (priority (verb . 5)))
     (composite :object nil
                :textual '(tsc@sentence tsc@finite tsc@np)
                :matrix (leaf :object S
                              :textual '(tsc@finite tsc@vp tsc@np))
                :adjuncts
                ((composite :object (lex 'for)
```

```
                        :textual '(tsc@clause-modifier-rankingII
                                   tsc@vp-modifier-rankingII
                                   tsc@np-modifier-rankingII)
                        :matrix
                        (composite :object nil
                                   :textual '(tsc@np)
                                   :matrix (leaf :object V
                                                 :textual '(tsc@np))
                                   :adjuncts
                                   ((leaf :object (lex 'some)
                                          :textual
                                          '(tsc@np-modifier-rankingI
                                            ))))))))
; S for no V
(resource-tree ((mandatory V S) (negated yes) (priority (verb . 5)))
   (composite :object nil
              :textual '(tsc@sentence tsc@finite tsc@np)
              :matrix (leaf :object S
                            :textual '(tsc@finite tsc@vp tsc@np))
              :adjuncts
              ((composite :object (lex 'for)
                        :textual '(tsc@clause-modifier-rankingII
                                   tsc@vp-modifier-rankingII
                                   tsc@np-modifier-rankingII)
                        :matrix
                        (composite :object nil
                                   :textual '(tsc@np)
                                   :matrix (leaf :object V
                                                 :textual '(tsc@np))
                                   :adjuncts
                                   ((leaf :object (lex 'no)
                                          :textual
                                          '(tsc@np-modifier-rankingI
                                            ))))))))
; the existence of V such that S
(resource-tree ((mandatory V S) (negated no) (priority (verb . 10)))
   (composite :object nil
              :textual '(tsc@np tsc@gerund)
              :matrix (leaf :object (lex 'exist) :textual '(tsc@np))
              :adjuncts ((composite :object (lex 'of)
                                  :textual '(tsc@np-modifier-rankingII)
                                  :matrix (leaf :object V
                                                :textual '(tsc@np)))
                       (composite :object (lex 'such-that)
                                  :textual '(tsc@np-modifier-embedded)
                                  :matrix (leaf :object S
                                                :textual '(tsc@clause
                                                           tsc@vp))))))
; \exists V S
(resource-tree ((mandatory V S) (negated no) (priority (sym . 10)))
```

```
    (composite :object nil
               :textual '(tsc@sentence tsc@clause)
               :matrix
               (kernel :object (sym 'dot)
                       :textual '(tsc@vp)
                       :arguments
                       ((composite :object nil
                                   :textual '(tsc@np)
                                   :matrix
                                   (leaf :object V :textual '(tsc@np))
                                   :adjuncts
                                   ((leaf :object (sym 'exist)
                                          :textual
                                          '(tsc@np-modifier-rankingI))))
                        (leaf :object S :textual '(tsc@np)))))))

;# Generalized Possession

(ts~def-realization-class (um@name-of :domain D :range R)
  ; the D R
  (resource-tree ((mandatory D R) (negated no) (priority (verb .10)))
    (composite :object nil
               :textual '(tsc@np)
               :matrix (leaf :object D
                             :textual '(tsc@np))
               :adjuncts ((leaf :object R
                                :textual '(tsc@name))))))

(ts~def-realization-class (um@ownership :domain D :range R)
  ; the D of R
  (resource-tree ((mandatory D R) (negated no) (priority (verb . 10)))
    (composite :object nil
               :textual '(tsc@np)
               :matrix (leaf :object D
                             :textual '(tsc@np))
               :adjuncts
               ((composite :object (lex 'of)
                           :textual '(tsc@np-modifier-rankingII)
                           :matrix (leaf :object R
                                         :textual '(tsc@np)))))))

(ts~def-realization-class (um@part-whole :part P :whole W)
  ; P is an element of W
  (resource-tree ((mandatory P W) (negated neg) (priority (verb . 10)))
    (composite :object nil
               :textual '(tsc@sentence tsc@clause)
               :matrix
               (kernel :object (lex 'be)
                       :textual '(tsc@vp tsc@gerund)
                       :negated neg
```

```
                         :arguments
                         ((leaf :object P
                                :textual '(tsc@np tsc@gerund)
                                :role ':agent)
                          (leaf :object (um 'um@ownership
                                            :domain (um 'um@object
                                                        :object 'element)
                                            :range W)
                                :textual '(tsc@np tsc@gerund)
                                :role ':ident)))))
  ;the element P of W
  (resource-tree ((mandatory P W) (negated no) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@np)
                :matrix
                (composite :object nil
                           :textual '(tsc@np)
                           :matrix
                           (leaf :object (lex 'element)
                                 :textual '(tsc@np))
                           :adjuncts
                           ((leaf :object P :textual '(tsc@name))))
                :adjuncts
                ((composite :object (lex 'of)
                            :textual '(tsc@np-modifier-rankingII)
                            :matrix
                            (leaf :object W :textual '(tsc@np)))))))

;# Intensive

(ts~def-realization-class (um@property-ascription :attribute A
                                                  :attribuend B)
  ; B is A
  (resource-tree ((mandatory A B) (negated neg) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@sentence tsc@clause)
                :matrix
                (kernel :object (lex 'be)
                        :textual '(tsc@vp)
                        :negated neg
                        :arguments
                        ((leaf :object B
                               :textual '(tsc@np tsc@gerund)
                               :role ':agent)
                         (leaf :object A
                               :textual '(tsc@np-modifier-rankingI)
                               :role ':ident)))))
  ; the A B
  (resource-tree ((mandatory A B) (negated no) (priority (verb . 10)))
     (composite :object nil
```

```
                 :textual '(tsc@np)
                 :matrix
                 (leaf :object B :textual '(tsc@np tsc@gerund))
                 :adjuncts
                 ((leaf :object A
                        :textual '(tsc@np-modifier-rankingI
                                   tsc@clause-modifier-rankingI))))))

(ts~def-realization-class (um@class-ascription :attribute A :attribuend B)
  ; B is A
  (resource-tree ((mandatory A B) (negated neg) (priority (verb . 10)))
     (composite :object nil
                 :textual '(tsc@sentence tsc@clause)
                 :matrix
                 (kernel :object (lex 'be)
                         :textual '(tsc@finite tsc@non-finite tsc@vp tsc@np)
                         :negated neg
                         :arguments
                         ((leaf :object B
                                :textual '(tsc@np tsc@gerund)
                                :role ':agent)
                          (leaf :object A
                                :textual '(tsc@np tsc@gerund)
                                :role ':ident)))))
  ; the A B
  (resource-tree ((mandatory A B) (negated no) (priority (verb . 10)))
     (composite :object nil
                 :textual '(tsc@np)
                 :matrix (leaf :object A
                               :textual '(tsc@np))
                 :adjuncts ((leaf :object B
                                  :textual '(tsc@name))))))

;# Circumstantial

(ts~def-realization-class (um@implication :cause P :effect Q)
  ; if P then Q
  (resource-tree ((mandatory P Q) (negated no) (priority (verb . 100)))
     (composite :object nil
                 :textual '(tsc@sentence tsc@finite)
                 :matrix
                 (composite :object (lex 'if)
                            :textual '(tsc@finite)
                            :matrix
                            (leaf :object P
                                  :textual '(tsc@finite tsc@vp)))
                 :adjuncts
                 ((composite :object (lex 'then)
                             :textual '(tsc@clause-modifier-embedded)
                             :matrix
```

```
                              (leaf :object Q
                                    :textual '(tsc@finite tsc@vp))))))
  ; P implies Q
  (resource-tree ((mandatory Q) (negated no) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@sentence tsc@clause)
                :matrix
                (kernel :object (lex 'imply)
                        :textual '(tsc@finite tsc@non-finite tsc@vp)
                        :arguments
                        ((leaf :object P
                               :role ':cause
                               :textual '(tsc@np tsc@gerund))
                         (leaf :object Q
                               :role ':effect
                               :textual '(tsc@np tsc@gerund
                                          tsc@clause-modifier-embedded
                                          tsc@vp-modifier-embedded))))))
  ; P => Q
  (resource-tree ((mandatory Q) (negated no) (priority (sym . 10)))
     (composite :object nil
                :textual '(tsc@sentence tsc@clause)
                :matrix (kernel :object (sym 'imply)
                                :textual '(tsc@finite tsc@non-finite
                                           tsc@vp)
                                :arguments ((leaf :object P
                                                  :role ':cause
                                                  :textual '(tsc@np))
                                            (leaf :object Q
                                                  :role ':effect
                                                  :textual '(tsc@np))))))
  (resource-tree ((mandatory P Q) (negated no) (priority (sym . 10)))
     (composite :object (sym 'imply)
                :textual '(tsc@np)
                :coords ((leaf :object P :textual '(tsc@np))
                         (leaf :object Q :textual '(tsc@np))))))

(ts~def-realization-class (um@derive :reason R :conclusion C :method M
                                     :next N)
  ; Since R, C by M, ... N
  (resource-tree ((mandatory C) (negated no) (priority (verb . 100)))
     (composite :object nil
                :textual '(tsc@sentence tsc@finite)
                :matrix (leaf :object C
                              :textual '(tsc@finite))
                :adjuncts
                ((leaf :object (um 'um@reason :reason R)
                       :textual '(tsc@clause-modifier-rankingII))
                 (leaf :object (um 'um@enablement :range M)
                       :textual '(tsc@clause-modifier-rankingII))
```

```
               (leaf :object N
                     :textual '(tsc@sentence
                                tsc@clause-modifier-embedded)))))
(resource-tree ((mandatory C) (negated no) (priority (verb . 20)))
   (composite :object nil
              :textual '(tsc@sentence tsc@finite)
              :matrix
              (composite :object nil
                         :textual '(tsc@finite)
                         :matrix
                         (leaf :object C :textual '(tsc@finite))
                         :adjuncts
                         ((leaf :object N
                                :textual
                                '(tsc@sentence
                                  tsc@clause-modifier-embedded))))
              :adjuncts
              ((leaf :object (um 'um@reason :reason R)
                     :textual '(tsc@finite
                                tsc@clause-modifier-rankingII))
               (leaf :object (um 'um@enablement :range M)
                     :textual '(tsc@clause-modifier-rankingII)))))
; R leads to C by M, ... N
(resource-tree ((mandatory R C) (negated neg) (priority (verb . 20)))
   (composite :object nil
              :textual '(tsc@sentence tsc@finite)
              :matrix
              (composite :object nil
                         :textual '(tsc@finite)
                         :matrix
                         (kernel :object (lex 'derive)
                                 :textual '(tsc@vp)
                                 :negated neg
                                 :arguments
                                 ((leaf :object R
                                        :role ':reason
                                        :textual '(tsc@np tsc@gerund))
                                  (leaf :object C
                                        :role ':conclusion
                                        :textual '(tsc@np tsc@gerund)))))
              :adjuncts
              ((leaf :object (um 'um@enablement :range M)
                     :textual '(tsc@clause-modifier-rankingII))
               (leaf :object N
                     :textual '(tsc@sentence
                                tsc@clause-modifier-embedded)))))
(resource-tree ((mandatory R C) (negated neg) (priority (verb . 10)))
   (composite :object nil
              :textual '(tsc@sentence tsc@finite)
              :matrix
```

```
                (kernel :object (lex 'derive)
                        :textual '(tsc@vp)
                        :negated neg
                        :arguments
                        ((leaf :object R
                               :role ':reason
                               :textual '(tsc@np tsc@gerund))
                         (composite :object nil
                                    :role ':conclusion
                                    :textual '(tsc@np tsc@gerund)
                                    :matrix
                                    (leaf :object C
                                          :role ':conclusion
                                          :textual '(tsc@np tsc@gerund))
                                    :adjuncts
                                    ((leaf :object N
                                           :textual
                                           '(tsc@np-modifier-embedded
                                             tsc@clause-modifier-embedded
                                             )))))))
                :adjuncts
                ((leaf :object (um 'um@enablement :range M)
                       :textual '(tsc@clause-modifier-rankingII)))))

(ts~def-realization-class (um@derive-cont :reason R :conclusion C
                                          :method M :next N)
  ; Then C by M, since R, ... N
  (resource-tree ((mandatory C) (negated no) (priority (verb . 100)))
     (composite :object nil
                :textual '(tsc@sentence)
                :matrix
                (composite :object nil
                           :textual '(tsc@p-sentence)
                           :matrix (leaf :object C
                                         :textual '(tsc@finite))
                           :adjuncts
                           ((leaf :object (um 'um@reason :reason R)
                                  :textual
                                  '(tsc@clause-modifier-embedded))
                            (leaf :object (um 'um@enablement :range M)
                                  :textual
                                  '(tsc@clause-modifier-rankingII))
                            (leaf :object N
                                  :textual
                                  '(tsc@sentence
                                    tsc@clause-modifier-embedded))))
                :adjuncts
                ((leaf :object (lex 'then) :textual '(tsc@particle)))))
  (resource-tree ((mandatory C) (negated no) (priority (verb . 20)))
     (composite :object nil
```

```
              :textual '(tsc@sentence)
              :matrix
              (composite :object nil
                         :textual '(tsc@p-sentence)
                         :matrix
                         (composite :object nil
                                    :textual '(tsc@finite)
                                    :matrix
                                    (leaf :object C
                                          :textual '(tsc@finite tsc@vp))
                                    :adjuncts
                                    ((leaf :object N
                                           :textual
                                           '(tsc@clause-modifier-embedded
                                             tsc@vp-modifier-embedded))))
                         :adjuncts
                         ((leaf :object (um 'um@reason :reason R)
                                :textual '(tsc@clause-modifier-embedded))
                          (leaf :object (um 'um@enablement :range M)
                                :textual
                                '(tsc@clause-modifier-rankingII))))
              :adjuncts
              ((leaf :object (lex 'then) :textual '(tsc@particle)))))
; Then R leads to C by M, ... N
(resource-tree ((mandatory R C) (negated neg) (priority (verb . 20)))
   (composite :object nil
              :textual '(tsc@sentence)
              :matrix
              (composite :object nil
                         :textual '(tsc@p-sentence)
                         :matrix
                         (kernel :object (lex 'derive)
                                 :textual '(tsc@vp)
                                 :negated neg
                                 :arguments
                                 ((leaf :object R
                                        :role ':reason
                                        :textual
                                        '(tsc@np tsc@gerund))
                                  (leaf :object C
                                        :role ':conclusion
                                        :textual
                                        '(tsc@np tsc@gerund))))
                         :adjuncts
                         ((leaf :object (um 'um@enablement :range M)
                                :textual
                                '(tsc@clause-modifier-rankingII))
                           (leaf :object N
                                 :textual
                                 '(tsc@sentence
```

```
                                        tsc@clause-modifier-embedded))))
                   :adjuncts
                   ((leaf :object (lex 'then) :textual '(tsc@particle)))))
(resource-tree ((mandatory R C) (negated neg) (priority (verb . 10)))
    (composite :object nil
               :textual '(tsc@sentence)
               :matrix
               (composite :object nil
                          :textual '(tsc@p-sentence)
                          :matrix
                          (composite :object nil
                                     :textual '(tsc@finite)
                                     :matrix
                                     (kernel
                                      :object (lex 'derive)
                                      :textual '(tsc@vp)
                                      :negated neg
                                      :arguments
                                      ((leaf :object R
                                             :role ':reason
                                             :textual
                                             '(tsc@np tsc@gerund))
                                       (composite
                                        :object nil
                                        :role ':conclusion
                                        :textual '(tsc@np tsc@gerund)
                                        :matrix
                                        (leaf :object C
                                              :role ':conclusion
                                              :textual
                                              '(tsc@np tsc@gerund))
                                        :adjuncts
                                        ((leaf
                                          :object N
                                          :textual
                                          '(tcs@np-modifier-embedded
                                            tcs@clause-modifier-embedded
                                            )))))))
                          :adjuncts
                          ((leaf :object (um 'um@enablement :range M)
                                 :textual
                                 '(tsc@clause-modifier-rankingII))))
               :adjuncts
               ((leaf :object (lex 'then) :textual '(tsc@particle)))))
; leading to C, ... N
(resource-tree ((mandatory C) (negated neg) (priority (verb . 20)))
    (composite :object nil
               :textual '(tsc@clause-modifier-embedded
                          tsc@vp-modifier-embedded)
               :matrix
```

```
                (kernel :object (lex 'derive)
                        :negated neg
                        :textual '(tsc@gerund)
                        :arguments ((leaf :object C
                                          :role ':conclusion
                                          :textual '(tsc@np tsc@gerund))))
                :adjuncts
                ((leaf :object (um 'um@reason :reason R)
                       :textual '(tsc@clause-modifier-rankingII))
                 (leaf :object (um 'um@enablement :range M)
                       :textual '(tsc@clause-modifier-rankingII))
                 (leaf :object N
                       :textual '(tsc@sentence
                                  tsc@clause-modifier-embedded)))))
 (resource-tree ((mandatory C) (negated neg) (priority (verb . 10)))
    (composite :object nil
               :textual '(tsc@clause-modifier-embedded
                          tsc@vp-modifier-embedded)
               :matrix
               (kernel :object (lex 'derive)
                       :negated neg
                       :textual '(tsc@gerund)
                       :arguments
                       ((composite :object nil
                                   :role ':conclusion
                                   :textual '(tsc@np tsc@gerund)
                                   :matrix
                                   (leaf :object C
                                         :role ':conclusion
                                         :textual '(tsc@np tsc@gerund))
                                   :adjuncts
                                   ((leaf :object N
                                          :textual
                                          '(tsc@np-modifier-embedded
                                            tsc@clause-modifier-embedded
                                            ))))))
               :adjuncts
               ((leaf :object (um 'um@reason :reason R)
                      :textual '(tsc@clause-modifier-rankingII))
                (leaf :object (um 'um@enablement :range M)
                      :textual '(tsc@clause-modifier-rankingII)))))
 ; which leads to C, ... N
 (resource-tree ((mandatory C) (negated neg) (priority (verb . 100)))
    (composite :object nil
               :textual '(tsc@sentence tsc@np-modifier-embedded)
               :matrix
               (composite :object nil
                          :textual '(tsc@clause)
                          :matrix
                          (kernel :object (lex 'derive)
```

```
                                        :negated neg
                                        :textual '(tsc@vp)
                                        :arguments
                                        ((leaf :object (lex 'which)
                                               :role ':reason
                                               :textual '(tsc@pronoun)
                                               :annotations '(relpron dpron))
                                         (composite
                                          :object nil
                                          :role ':conclusion
                                          :textual '(tsc@np tsc@gerund)
                                          :matrix
                                          (leaf :object C
                                                :role ':conclusion
                                                :textual '(tsc@np tsc@gerund))
                                          :adjuncts
                                          ((leaf :object N
                                                 :textual
                                                 '(tsc@sentence
                                                   tsc@np-modifier-embedded
                                                   tsc@clause-modifier-embedded
                                                   )))))))
                         :adjuncts ((leaf :object (um 'um@reason :reason R)
                                          :textual '(tsc@clause-modifier-embedded))
                                    (leaf :object (um 'um@enablement :range M)
                                          :textual
                                          '(tsc@clause-modifier-rankingII)))))
          (resource-tree ((mandatory C) (negated neg) (priority (verb . 20)))
             (composite :object nil
                         :textual '(tsc@sentence tsc@np-modifier-embedded)
                         :matrix
                         (composite :object nil
                                    :textual '(tsc@clause)
                                    :matrix
                                    (kernel :object (lex 'derive)
                                            :negated neg
                                            :textual '(tsc@vp)
                                            :arguments
                                            ((leaf :object (lex 'which)
                                                   :role ':reason
                                                   :textual '(tsc@pronoun)
                                                   :annotations '(relpron dpron))
                                             (leaf :object C
                                                   :role ':conclusion
                                                   :textual '(tsc@np tsc@gerund)))))
                         :adjuncts
                         ((leaf :object (um 'um@reason :reason R)
                                :textual '(tsc@clause-modifier-embedded))
                          (leaf :object (um 'um@enablement :range M)
                                :textual '(tsc@clause-modifier-rankingII))
```

```
                 (leaf :object N
                       :textual '(tsc@sentence
                                  tsc@clause-modifier-embedded))))))

(ts~def-realization-class (um@enablement :domain D :range M)
  ; D by M
  (resource-tree ((mandatory D M) (negated no) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@sentence tsc@clause)
                :matrix
                (leaf :object D
                      :textual '(tsc@clause))
                :adjuncts
                ((composite :object (lex 'by)
                            :textual '(tsc@clause-modifier-rankingII)
                            :matrix
                            (leaf :object M
                                  :textual '(tsc@np tsc@gerund)))))))

(ts~def-realization-class (um@enablement :domain nil :range M)
  ; by M
  (resource-tree ((mandatory M) (negated no) (priority (verb . 10)))
     (composite :object (lex 'by)
                :textual '(tsc@clause-modifier-rankingII
                           tsc@vp-modifier-rankingII
                           tsc@np-modifier-rankingII)
                :matrix (leaf :object M
                              :textual '(tsc@np tsc@gerund)))))

(ts~def-realization-class (um@enablement :domain nil :range nil)
  ; (Silence)
  (resource-tree ((negated no) (priority (verb . 10)))
     (leaf :object nil)))

;# Zusaetzlich:

(ts~def-realization-class (um@process-relation :process P :domain D
                                               :range R)
  ; D Ps R
  (resource-tree ((mandatory P D R) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@sentence tsc@clause)
                :matrix
                (kernel :object (lex P)
                        :textual '(tsc@non-finite tsc@vp)
                        :arguments
                        ((leaf :object D
                               :role ':domain
                               :textual '(tsc@np tsc@gerund))
                         (leaf :object R
```

```
                              :role ':range
                              :textual '(tsc@np tsc@gerund))))))
  ; the P of D and R
  (resource-tree ((mandatory P D R) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@np tsc@gerund)
                :matrix
                (leaf :object P :textual '(tsc@np tsc@gerund))
                :adjuncts
                ((composite :object (lex 'of)
                            :textual '(tsc@np-modifier-rankingII
                                       tsc@clause-modifier-rankingII)
                            :matrix
                            (leaf :object (um 'um@conjunction
                                                :domain (list D R))
                                                :textual
                                                '(tsc@np tsc@gerund)))))))

(ts~def-realization-class (um@quality-relation :quality Q :domain D
                                                :range R)
  ; D is Q to R
  (resource-tree ((mandatory Q D R) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@sentence tsc@finite tsc@non-finite)
                :matrix
                (composite :object nil
                           :textual '(tsc@finite tsc@non-finite)
                           :matrix
                           (kernel :object (lex 'be)
                                   :textual '(tsc@finite tsc@non-finite
                                              tsc@vp)
                                   :arguments
                                   ((leaf :object D
                                          :textual '(tsc@np tsc@gerund)
                                          :role ':agent)
                                    (leaf :object Q
                                          :textual
                                          '(tsc@np-modifier-rankingI)
                                          :role ':ident))))
                :adjuncts
                ((composite :object (lex 'to)
                            :textual '(tsc@clause-modifier-rankingII)
                            :matrix (leaf :object R
                                          :textual
                                          '(tsc@np tsc@gerund))))))

  ; The Q of D and R
  (resource-tree ((mandatory Q D R) (negated no) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@np)
```

```
                :matrix
                (leaf :object Q :textual '(tsc@np))
                :adjuncts
                ((composite :object (lex 'of)
                            :textual '(tsc@np-modifier-rankingII)
                            :matrix
                            (composite :object (lex 'and)
                                       :textual '(tsc@np tsc@gerund)
                                       :coords
                                       ((leaf :object D
                                              :textual
                                              '(tsc@np tsc@gerund))
                                        (leaf :object R
                                              :textual
                                              '(tsc@np tsc@gerund)))))))
  ; D is Q
  (resource-tree ((mandatory Q D) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@sentence tsc@finite tsc@non-finite)
                :matrix
                (kernel :object (lex 'be)
                        :textual '(tsc@finite tsc@non-finite tsc@vp)
                        :arguments
                        ((leaf :object D
                               :textual '(tsc@np tsc@gerund)
                               :role ':agent)
                         (leaf :object Q
                               :textual '(tsc@np-modifier-rankingI
                                          tsc@clause-modifier-rankingI)
                               :role ':ident))))))

;# Mental Processes #

(ts~def-realization-class (um@assume :senser S :phenomenon P)
  ; S assumes P
  (resource-tree ((mandatory P) (negated no) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@sentence tsc@finite tsc@non-finite)
                :matrix
                (kernel :object (lex 'assume)
                        :textual '(tsc@finite tsc@non-finite tsc@vp)
                        :arguments
                        ((leaf :object S
                               :role ':senser
                               :textual '(tsc@np))
                         (leaf :object P
                               :role ':phenomenon
                               :textual '(tsc@np tsc@gerund
                                          tsc@clause-modifier-embedded
                                          tsc@vp-modifier-embedded))))))
```

```
  ; S lets P
  (resource-tree ((mandatory S P) (negated neg) (priority (verb . 100)))
     (composite :object nil
                :textual '(tsc@sentence tsc@finite)
                :matrix
                (kernel :object (lex 'let)
                        :textual '(tsc@finite tsc@vp)
                        :arguments
                        ((leaf :object S
                               :role ':actor
                               :textual '(tsc@np))
                         (leaf :object P
                               :role ':action
                               :textual '(tsc@infinitive))))))

(ts~def-realization-class (um@consider :senser S :phenomenon P)
  ; S considers P
  (resource-tree ((mandatory P) (negated no) (priority (verb . 10)))
     (composite :object nil
                :textual '(tsc@sentence tsc@finite tsc@non-finite)
                :matrix
                (kernel :object (lex 'consider)
                        :textual '(tsc@finite tsc@non-finite tsc@vp)
                        :arguments
                        ((leaf :object S
                               :role ':senser
                               :textual '(tsc@np))
                         (leaf :object P
                               :role ':phenomenon
                               :textual '(tsc@np tsc@gerund))))))

;##              Qualities             ##

(ts~def-realization-class (um@quality :quality Q)
  (resource-tree ((mandatory Q) (negated no) (priority (verb . 10)))
     (leaf :object (lex Q)
           :textual '(tsc@modifier))))


(ts~def-realization-class (um@number :quality Q)
  ; Q
  (resource-tree ((mandatory Q) (priority (verb . 10)))
     (leaf :object (lex Q)
           :ideational 'um@number
           :textual '(tsc@np tsc@np-modifier-rankingI))))

;##               Makros               ##

(ts~def-realization-class (um@reason :reason R)
  ; Since R
```

```
  (resource-tree ((mandatory R) (negated no) (priority (verb . 10)))
     (composite :object (lex 'since)
                :textual '(tsc@finite)
                :matrix
                (leaf :object R
                      :textual '(tsc@finite tsc@vp))))
  ; Because of R
  (resource-tree ((mandatory R) (negated no) (priority (verb . 10)))
     (composite :object (lex 'because)
                :textual '(tsc@clause-modifier-rankingII
                           tsc@vp-modifier-rankingII)
                :matrix
                (composite :object (lex 'of)
                           :textual 'same
                           :matrix
                           (leaf :object R
                                 :textual '(tsc@np tsc@gerund))))))
(ts~def-realization-class (um@reason :reason nil)
  ; (Silence)
  (resource-tree ((negated no) (priority (verb . 10)))
     (leaf :object nil)))


(ts~def-realization-class (um@layout :before B :after A)
  (resource-tree ((negated no) (priority (verb . 10)))
     (leaf :object nil
           :textual '(tsc@layout)
           :layout-before B
           :layout-after A)))
```

Literaturverzeichnis

[Andrews, 1980] Peter B. Andrews. Transforming matings into natural deduction proofs. In *Proceedings of the 5th International Conference on Automated Deduction*. Springer Verlag, 1980.

[Andrews, 1981] Peter B. Andrews. Theorem proving via general matings. *Journal of the Association for Computing Machinery*, 28(2):193–214, April 1981.

[Bateman *et al.*, 1990] John A. Bateman, Robert T. Kasper, Johanna D. Moore und Richard A. Whitney. A general organization of knowledge for natural language processing: the Penman upper model. Technischer Bericht, 1990. ISI Penman Note.

[Bateman, 1992] John Bateman. Systemic grammar. In *Encyclopedia of AI*. 1992.

[Bateman, 1994] John Bateman. KPML: The KOMET-Penman (multilingual) development environment. Technischer Bericht, IPSI, GMD, Darmstadt, 1994.

[Bibel, 1983] Wolfgang Bibel. Matings in matrices. *Communications of the ACM*, 26:844–852, 1983.

[Chester, 1976] Daniel Chester. The translation of formal proofs into English. *AI*, 7:178–216, 1976.

[Dale, 1992] Robert Dale. *Generating Referring Expressions*. ACL-MIT Press Series in Natural Language Processing. MIT Press, 1992.

[Dalianis und Hovy, 1993] Hercules Dalianis und Eduard H. Hovy. Aggregation in natural language generation. In *Proceedings of the 4th European Workshop on Natural Language Generation*, 1993.

[Dalianis, 1993] Hercules Dalianis. A natural language generation system for validating specifications for telephone services. In *Proceedings of the 5th Innovative Applications of Artificial Intelligence Conference*, 1993.

[Deussen, 1971] Peter Deussen. *Halbgruppen und Automaten*, Band 99 in *Heidelberger Taschenbücher, Sammlung Informatik*. Springer Verlag, 1971.

[Edgar und Pelletier, 1993] Andrew Edgar und Francis J. Pelletier. Natural language explanation of natural deduction proofs. Vancouver, Kanada, 1993. Centre for Systems Science, Simon Fraser University.

[Fiedler und Huang, 1995] Armin Fiedler und Xiaorong Huang. Aggregation for mathematical proofs. In *Proceedings of the 5th European Workshop on Natural Language Generation*, Leiden, Niederlande, 1995.

[Gentzen, 1935] Gerhard Gentzen. Untersuchungen über das logische Schließen I. *Mathematische Zeitschrift*, 39:176–210, 1935.

[Halliday, 1994] M. A. K. Halliday. *An Introduction to Functional Grammar.* Edward Arnold, 2. Auflage, 1994.

[Horacek, 1992] Helmut Horacek. *An Integrated View of Text Planning.* 1992.

[Hovy, 1992] Eduard H. Hovy. Sentence planning requirements for automated explanation generation. Diamod 23, 1992.

[Huang und Fiedler, 1995] Xiaorong Huang und Armin Fiedler. Generating multilingual proofs. In *Proceedings of the IJCAI-95 Workshop on Multilingual Text Generation*, 1995.

[Huang, 1994a] Xiaorong Huang. *Human Oriented Proof Presentation: A Reconstructive Approach.* Dissertation, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1994.

[Huang, 1994b] Xiaorong Huang. Planning argumentative texts. In *Proceedings of 15th International Conference on Computational Linguistics*, Seiten 329–333, Kyoto, Japan, 1994.

[Huang, 1994c] Xiaorong Huang. Planning reference choices for argumentative texts. [IWN, 1994].

[Huang, 1994d] Xiaorong Huang. Proverb: A system explaining machine-found proofs. In Ashwin Ram und Kurt Eiselt, Herausgeber, *Proceedings of 16th Annual Conference of the Cognitive Science Society*, Seiten 427–432, Atlanta, USA, 1994. Lawrence Erlbaum Associates.

[Huang, 1994e] Xiaorong Huang. Reconstructing proofs at the assertion level. In Alan Bundy, Herausgeber, *Proceedings of the 12th Conference on Automated Deduction*, Nummer814 in LNAI, Seiten 738–752, Nancy, France, 1994.

[IWN, 1994] *Proceedings of the 7th International Workshop on Natural Language Generation*, Kennebunkport, Maine, USA, 1994.

[Kempen, 1991] Gerard Kempen. Conjunction reduction and gapping in clause-level coordination: an inheritance-based approach. *Computational Intelligence*, 7(4):357–360, 1991.

[Kilger und Finkler, 1995] Anne Kilger und Wolfgang Finkler. Incremental generation for real–time applications. Research Report RR-95-11, DFKI, Saarbrücken, July 1995.

[Lingenfelder, 1990] Christoph Lingenfelder. *Transformation and Structuring of Computer Generated Proofs.* Dissertation, Universität Kaiserslautern, Kaiserslautern, Germany, 1990.

[Matthiesen und Bateman, 1990] Christian M. I. M. Matthiesen und John A. Bateman. *Text Generation And Systemic-Functional Linguistics.* Pinter Publishers, London, 1990.

[Matthiesen, 1990] Christian M. I. M. Matthiesen. *Overview of Nigel.* ISI Penman note, 1990.

[McDonald, 1983] David D. McDonald. Natural language generation as a computational problem. 1983.

[Meteer *et al.*, 1987] M. Meteer, D. McDonald, S. Anderson, D. Forster, L. Gay, A. Huettner und P. Sibun. Mumble-86: Design and implementation. Technischer Bericht87-87, University of Massachusetts, 1987.

[Meteer, 1991] Marie W. Meteer. Bridging the generation gap between text planning linguistic realization. *Computational Intelligence*, 7(4), 1991.

[Meteer, 1992] Marie W. Meteer. *Expressibility and the Problem of Efficient Text Planning.* Pinter Publishes, London, 1992.

[Miller, 1983] Dale Miller. *Proofs in Higher-Order Logic.* Dissertation, Carnegie-Mellon University, 1983.

[Nesmith und Pfenning, 1990] Daniel Nesmith und Frank Pfenning. Presenting intuitive deductions via symmetric simplification. In Mark Stickel, Herausgeber, *Proceedings of the 10th Conference on Automated Deduction*, Nummer449 in LNCS, Seiten 336–350, Kaiserslautern, Germany, 1990.

[Nesmith, 1994] Dan Nesmith. **KEIM**-Manual. Version 1.2, 1994. Universität des Saarlandes.

[Panaget, 1994a] Franck Panaget. The micro-planning component of a generation system. Technischer Bericht, IRST, 1994.

[Panaget, 1994b] Franck Panaget. Using a textual representational level component in the context of discourse or dialogue generation. [IWN, 1994].

[Paris *et al.*, 1995] Cécile Paris, Keith Vander Linden, Markus Fischer, Anthony Hartley, Lyn Pemberton, Richard Power und Donia Scott. A support tool for writing multilingual instructions. In *Proceedings of the 14th IJCAI*, Montreal, Canada, 1995. Morgan Kaufman.

[PENMAN, 1989] The PENMAN Project, USC/Information Science Institute, Marina del Rey, California. *The* PENMAN *Documentation and User Guide*, 1989.

[Pfenning, 1987] F. Pfenning. *Proof Transformations in Higher-Order Logic.* Dissertation, Carnegie-Mellon University, Pittsburgh Pa., 1987.

[Reichman, 1985] Rachel Reichman. *Getting Computers to Talk Like You and Me.* MIT Press, 1985.

[Robinson, 1965] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, 1965.

[Talmy, 1987] Leonard Talmy. The relation of grammar to cognition. In B. Rudzka-Ostyn, Herausgeber, *Topics in Cognitive Linguistics*. 1987.

[Zukerman, 1991] Ingrid Zukerman. Using meta-comments to generate fluent text in a technical domain. *Computational Intelligence*, 7:276–295, 1991.