Computerlinguistische Beschreibung der japanischen Verbmorphologie und ihrer semantischen Entsprechungen

Diplomarbeit
bei
FLP/KIT

Aufgabensteller:
Prof. Dr. Bernd Mahr
(TU Berlin)

Betreuer:
Dr. Melanie Siegel
(DFKI Saarbrücken)
Dr. Manfred Stede
(TU Berlin)

Dietrich Bollmann
Matrikelnummer: 154540
Stromstraße 40
10551 Berlin

15. Dezember 1999

Inhaltsverzeichnis

Vorwort

Morphologische Fragestellungen spielen in vielen computerlinguistischen Systemen eine nur untergeordnete Rolle. Der Grund dafür ist jedoch nicht, daß die Morphologie für sprachverarbeitende Systeme keine Wichtigkeit hätte; vielmehr resultiert diese Unterbewertung aus der Komplexität des zu beschreibenden Gegenstandes und der Struktur der in der modernen Linguistik häufig betrachteten indoeuropäischen Sprachen. Der Modellierung von Sprache muß eine Eingrenzung der betrachteten Phänomene vorausgehen. Bei indoeuropäischen Sprachen bietet sich die Ausgrenzung der Morphologie an. Die begrenzte Anzahl ihrer morphologischen Formen legt eine Behandlung durch das Lexikon nahe; an die Stelle systematischer Beschreibungen treten Aufzählungen lexikalischer Einträge.

Die Betrachtung einer anderen Sprachfamilie kann eine Neubestimmung der zu beschreibenden sprachlichen Bereiche notwendig machen. Für das Japanische hat dieses eine höhere Gewichtung der Morphologie zur Folge: Schon die Modellierung eines kleinen Ausschnittes dieser Sprache macht die Einbeziehung morphologischer Regularitäten notwendig. Insbesondere gilt dieses für die Morphologie der japanischen Verben und Adjektive. Damit ist die Motivation für die folgenden Betrachtungen umrissen: Sie widmen sich der Frage, wie sich die Verb- und Adjektivmorphologie im Rahmen einer computerlinguistischen Beschreibung des Japanischen adäquat darstellen läßt.

Das Japanische wird hinsichtlich der Morphologie seiner Verben und Adjektive zu den agglutinierenden Sprachen gezählt. Die Verschleifung der Morpheme erinnert jedoch gleichzeitig an Mechanismen, die charakteristisch für den flektierenden Sprachbau sind. Die Spezifikation adäquater und leicht zu handhabender formaler Mittel zur Beschreibung dieser Charakteristika der japanischer Grammatik bildet zusammen mit der Modellierung eines relevanten Ausschnittes japanischer Verbformen den Schwerpunkt dieser Arbeit.

Die Modifikation der Form eines Verbes hat im Japanischen oft auch eine Veränderung der Valenz zur Folge. Die Modellierung dieses Phänomens erfolgt durch die Integration des Subkategorisierungsrahmens in die lexikalische Beschreibung der Verbstämme und die Spezifikation von valenzmodifizierenden Funktionen im Lexikoneintrag der Verbsuffixe. Eine minimale Syntax, die auf dem so erzeugten Subkategorisierungsrahmen aufsetzt, veranschaulicht die Wechselwirkung von Morphologie und Syntax.

Die morphologische Modifikation von Verben und die Veränderungen, die sich dadurch in ihrer Syntax ergeben, spiegeln sich auch in ihrer semantischen Beschreibung. Diese muß so angelegt sein, daß die morphologisch verursachten Veränderungen der Verbbedeutung und der semantischen Argumentstruktur dargestellt werden können. Diese Aspekte der semantischen Beschreibung lassen sich am besten anhand eines Formalismus veranschaulichen, der von allen Problemen abstrahiert, die durch die Verbmorphologie nicht direkt betroffen sind. Eine an der sprachlichen Oberfläche orientierte Semantik erfüllt diese Bedingung. Die Darstellung der Wechselwirkungen zwischen Morphologie, Syntax und Semantik anhand eines solchen Formalismus bildet daher einen weiteren wesentlichen Aspekt unserer Betrachtungen.

Die formale Ausrichtung dieser Arbeit und die drei Schwerpunkte Morphologie, Syntax und Semantik spiegeln sich in der Gliederung der folgenden Betrachtungen:

Kapitel 1 bietet einen Überblick über Strategie und Methodik formaler Beschreibungen in der Linguistik: Der benutzte Formalismus TFS (Emele 1993) wird anhand von Beispielen beschrieben und mathematisch fundiert.

Kapitel 2 führt in die japanische Verbmorphologie ein. Ausgehend von traditionellen japanischen Beschreibungen wird ein System entwickelt, daß sich zur formalen Modellierung eignet. In Teilen, wie der Subklassifizierung der morphologischen Primitiven und der Anzahl und Struktur der Stammformen, baut dieses Kapitel auf meiner Studienarbeit (Bollmann 1997) auf. Im Gegensatz zu dem dort entwickelten, defaultlogischen Modell, wird jedoch eine Beschreibung angestrebt, die sich — ohne große Einbußen in Eleganz und Ökonomie — mit monotonen Mitteln umsetzen läßt.

Kapitel 3 beschreibt den syntaktisch-semantischen Rahmen der Morphologie. Auf vorhandenen Beschreibungen des Japanischen — wie der Verbmobilgrammatik (Siegel 1996b), der JPSG (Gunji 1987, Gunji 1991, Gunji 1999) und der HPSG (Manning et al. 1999) — aufbauend, wird eine minimale Syntax und Semantik des Japanischen beschrieben. Dargestellt werden nur die Aspekte der japanischen Grammatik, die mit der Morphologie in Wechselwirkung treten und daher zu ihrer Beschreibung vorausgesetzt werden müssen.

In *Kapitel 4* wird die morphologische Beschreibung, die in Kapitel 2 entwickelt wurde, formalisiert und in die syntaktisch-semantische Beschreibung aus Kapitel 3 integriert. Auch wenn Ableitung und Struktur des morphologischen Systems, das in Kapitel 2 vorgestellt wurde, für den menschlichen Betrachter auf den ersten Blick kompliziert erscheinen, zeigen sich hier dessen Vorzüge aus formaler Perspektive: Auf zwei einfachen Typenhierarchien aufbauend lassen sich die Formen japanischer Verben mit nur einem morphologischen Schema einheitlich und elegant erklären.

In *Kapitel 5* wird das morphologische System auf ein repräsentatives Fragment der japanischen Verbmorphologie angewandt. Abgesehen von den speziellen Formen der Höflichkeitssprache (敬語, *keigo*) und einigen altertümlichen Formen werden die meisten Verbformen des modernen Japanischen abgedeckt. Dieses zeigt sich auch in Anhang B, in dem für alle Verbklassen

— von den oben erwähnten Höflichkeitsformen abgesehen — die Ableitung sämtlicher Verbformen gezeigt wird, die in den Verbtafeln des *"The Complete Japanese Verb Guide"* (Ishii et al. 1989) beschrieben werden. Im Gegensatz zu dem *"The Complete Japanese Verb Guide"* erlaubt unser System jedoch ebenso die Ableitung beliebiger Kombinationen dieser Formen.

In *Kapitel 6* wird auf die Vorteile und Probleme eingegangen, die sich aus einem deklarativen Ansatz bei der Operationalisierung der Grammatik ergeben: Aufgrund der Verwendung eines Kalküls zur Interpretation von Constraintbeschreibungen eignet sich TFS zwar gut als experimentelle Umgebung für die Spezifikation von Grammatikausschnitten; für konkrete computerlinguistische Anwendungen hingegen ist es nur begrenzt einsetzbar. Es werden Möglichkeiten zur Bewältigung der auftretenden Schwierigkeiten skizziert und ihre Anwendung anhand der Entwicklungsgeschichte der erarbeiteten japanischen Grammatik illustriert.

Den Schluß dieser Arbeit bildet *Kapitel 7*: Hier werden die Ergebnisse der Arbeit noch einmal zusammengefaßt und mögliche Fortsetzungen angesprochen.

Anhang A beschreibt den Korpus, der in dieser Arbeit verwendet wurde: Zu allen Verbformen werden die benutzten Beispielsätze mit ihren semantischen Entsprechungen aufgelistet.

Anhang B, auf den schon oben hingewiesen wurde, zeigt anhand einer Auswahl repräsentativer Verben, daß die hier beschriebene Morphologie alle Formen abzuleiten gestattet, die in Ishii et al. 1989 aufgelistet werden. Eine Ausnahme bilden die speziellen Höflichkeitsformen, deren Beschreibung mit den vorgestellten Mitteln aber keine Schwierigkeiten verursachen dürfte. Bei der Benennung der Formen wird in Anhang B, der besseren Vergleichbarkeit halber, ausnahmsweise Ishii et al. 1989 gefolgt.

Die Beispielsätze, die in dieser Arbeit verwendet werden, sind zum größten Teil dem Werk *"A Dictionary of Basic Japanese Grammar"* von Seiiche Makino und Michio Tsutsui (Makino und Tsutsui 1986) und der *"Japanische[n] Morphosyntax"* von Jens Rickmeyer (Rickmeyer 1995) entnommen. Da die Sätze auch in diesen Werken meist nur zitiert werden, läßt sich dort nachlesen, aus welchen Korpora, Grammatiken, Aufsätzen etc. diese ursprünglich stammen.

Danksagung

Zuallererst möchte ich mich bei Michelle Brehm, Ulrike Meibohm und Gitta-Miriam Oellinger bedanken. Sodann bei den Mitgliedern der Examensgruppe — Susanne Nowodnick, Ralf Behler, Barbara Kröber, Phillip Bartels, Christiane und Joanna Blümel, Roman Czyborra, Annette Kirsch und Giuseppe Pitronaci — und meinen Freunden in und außerhalb der Informatik: Obwohl — und oft *weil* — fachfremd, haben sie mir durch Anteilnahme, Unterstützung und Interesse über alle auftretenden Schwierigkeiten hinweggeholfen! Ebensolcher Dank gilt meinen Betreuern Melanie Siegel und Manfred Stede. Beide möchte ich für die Geduld und Ausdauer loben, mit der sie meinen immer neuen Vorstellungen und den daraus folgenden Verzögerungen begegnet sind: Für die Zukunft wünsche ich ihnen weniger "aufwendige" Diplomanden. Melanie Siegels Grammatik des Japanischen hat mir große Dienste erwiesen: Sie bildet Basis und Rahmen meines morphologischen Modells. Bei Fragen zur japanischen Sprache und Grammatik standen mir 中村隆治, 押元有治 und 鬼頭直 zur Seite. Besonders dankbar bin ich ihnen auch für die Hilfe bei der Auswahl und Bildung der japanischen Beispielsätze. Marianne Steffens, Wolfram Rauch und Reimund Noll haben mir mit Geduld und Kritik bei der Ausformulierung der Texte geholfen: Vielen Dank! Elizabeth, Sigrid, Leonard und Sebastian gilt mein Dank für das Vorleben der glücklichen Beispielsituationen aus Kapitel 1. Professor Bernd Mahr danke ich für die weite Auslegung des Begriffes *Informatik*: Seine Aufgeschlossenheit und Freude an interdisziplinären Thematiken machte es mir möglich, meine Interessen in dieser Weise zu kombinieren. In diesem Zusammenhang möchte ich auch Dirk Lüdtke danken, dessen Begeisterung für die Verbindung von Informatik und Japanologie manches Motivationsloch stopfte. Meine Eltern haben mich immer wieder ermutigt, im Studium meinen Interessen und Neigungen zu folgen. Ohne ihr Vertrauen und den durch sie gewährleisteten zeitlichen und finanziellen Freiraum hätte ich mich auf einen meiner Interessensschwerpunkte — Japanisch, Linguistik und Informatik — beschränken müssen!

1 Formale Linguistik und Constraintsysteme

In Anlehnung an die Methoden der exakten Wissenschaften fordert Noam Chomsky 1957 in seinem Buch "Syntactic Structures" (Chomsky 1957) die Benutzung *formaler Modelle* für die linguistische Theoriebildung:

> *Precisely constructed models for linguistic structure can play an important role, both negative and positive, in the process of discovery itself. By pushing a precise but inadequate formulation to an unacceptable conclusion, we can often expose the exact source of this inadequacy and, consequently, gain a deeper understanding of the linguistic data. More positively, a formalized theory may automatically provide solutions for many problems other than those for which it was explicitly designed. Obscure and intuition-bound notions can neither lead to absurd conclusions nor provide new and correct ones, and hence they fail to be useful in two important respects.*[1]

An späterer Stelle führt Chomsky weiter aus:

> *A grammar of the language L is essentially a theory of L. Any scientific theory is based on a finite number of observations, and it seeks to relate the observed phenomena and to predict new phenomena by constructing general laws in terms of hypothetical constructs such as (in physics, for example) "mass" and "electron." Similarly, a grammar of English is based on a finite corpus of utterances (observations), and it will contain certain grammatical rules (laws) stated in terms of the particular phonemes, phrases, etc., of English (hypothetical constructs). These rules express structural relations among the sentences of the corpus and the indefinite number of sentences generated by the grammar beyond the corpus (predictions).*[2]

Chomsky begründet damit nicht nur eine neue Schule innerhalb der Linguistik — die sogenannte *Generative Transformations-Grammatik*[3] —, sondern

1 Chomsky 1957, S. 5 (*preface*).

2 Chomsky 1957, S. 49 (*On the goals of linguistic theory, 6.1*).

3 Die Bezeichnung *Generative Transformations-Grammatik* wird als Sammelbegriff für Theorien verwendet, die in der Tradition stehen, die von Noam Chomsky durch das Buch "*Syntactic Structures*" (Chomsky 1957) begründet wurde. Neuere Versionen dieser Theorie sind unter den Namen *Government and Binding* (Chomsky 1981) und *The Minimalist Program* (Chomsky 1995) bekannt.

prägt mit seinen Forderungen die gesamte moderne Linguistik in entscheidender Weise. Notwendige Folge dieser neuen Tendenz ist die Entwicklung einer Vielzahl mathematischer Formalismen, die speziell auf die Darstellung linguistischer Sachverhalte ausgerichtet sind.

Neben der 'Generativen Transformationsgrammatik' ist die 'Head-Driven Phrase Structure Grammar' (HPSG, Pollard und Sag 1987, Pollard und Sag 1994) unter den formalen linguistischen Theorien heute wohl die bekannteste.

Merkmalstrukturen und Constraints — der formale Rahmen der HPSG

Diese Arbeit orientiert sich weitgehend an der HPSG und benutzt daher auch ähnliche formale Mittel:

Merkmalstrukturen dienen der formalen Darstellung sprachlicher Entitäten im Modell: Morphe, Wörter, Phrasen, Sätze etc. werden durch Merkmalstrukturen repräsentiert.

Constraints sind das eigentliche Mittel der linguistischen Modellierung: Die Regularitäten aller linguistischen Beschreibungsebenen — morphologische, lexikalische, syntaktische, semantische etc. — werden durch Constraints formalisiert. Constraints schränken die Menge der kombinatorisch möglichen Merkmalstrukturen ein: Nur Strukturen, die allen Constraints genügen, sind im Sinne des linguistischen Modells korrekt.

Resolution — Mit Hilfe eines *Resolutionsverfahrens* lassen sich aus Sätzen linguistische Strukturbeschreibungen, und aus Strukturbeschreibungen Sätze ableiten. Beide Ableitungen entsprechen einer allgemeineren Form der Berechnung: Aus *unterspezifizierten* Merkmalstrukturen wird die Menge der *vollspezifizierten* Merkmalstrukturen errechnet.[4]

Merkmalstrukturen und Constraints zusammen bilden das linguistische Modell, die Resolution dient ihrer Operationalisierung.

Die mathematischen Grundlagen der HPSG — und damit auch unseres Modells — sind Thema der folgenden Abschnitte: Merkmalstrukturen, Constraints und Resolution werden formal eingeführt. Unsere Darstellung orientiert sich an der formalen Spezifikation des Systems TFS (T*yped* F*eature* S*tructures Representation Formalism*, Emele und Zajac 1990a, Emele und Zajac 1990b, Emele 1991, Emele 1993, Emele 1994, und Zajac 1992), das zur Implementierung verwendet wurde, und an Carpenter 1992, dem "Standardwerk" zur Logik getypter Merkmalstrukturen. Aspekte beider Darstellungen

4 Was unter einer *vollspezifizierten* Merkmalstruktur zu verstehen ist, hängt von dem verwendeten Resolutionsverfahren ab (vgl. Absatz 1.5 (Resolution), S. 46).

wurden — zum Teil in veränderter Form — zu einer möglichst einfachen und verständlichen Darstellung integriert.[5]

1.1 Sprachliche Zeichen als Merkmalstrukturen

Form und Eigenschaften sprachlicher Zeichen, also Morphe, Wörter, Sätze etc., werden in Theorien wie der HPSG durch Merkmalstrukturen dargestellt. Der Einführung und Definition dieser *Merkmalstrukturen* dient der folgende Abschnitt.

Eine informelle Vorstellung von Merkmalstrukturen erleichtert das Verständnis der Definitionen. Es werden deshalb, anhand einiger Beispiele, die wichtigsten Begriffe für Merkmalstrukturen zuerst anschaulich vorgestellt. Anschließend wird die AVM[6]-Notation (Merkmal-Wert-Matritzen-Notation) für Merkmalstrukturen eingeführt. Der größeren Anschaulichkeit halber wird sie an vielen Stellen der Graphennotation vorgezogen.

Für die Spezifikation von Merkmalstrukturen spielen *Typen* und *Typenhierarchien* eine zentrale Rolle. Nach der informellen Einführung von Merkmalstrukturen werden daher die Typen und Typenhierarchien formalisiert.

Die Definition der Merkmalstrukturen als Graphen folgt am Ende des Abschnittes.

---

5 Folge der Vereinfachung sind unter anderem:

1. Merkmalstrukturen werden als gerichtete Graphen modelliert und die AVM-Schreibweise als eine Art Repräsentantensystem für Äquivalenzklassen von Merkmalstrukturen aufgefaßt.
2. Die in der formalen, modelltheoretischen Logik übliche Unterscheidung zwischen *Beschreibungssprache* und *Modelltheoretischer Semantik* wurde aufgegeben. Eine Merkmalstruktur steht — je nach Kontext — entweder für sich selbst, oder für die Menge der Strukturen, die von ihr subsumiert werden.
3. Die Unifikation von Typen — und damit auch von Merkmalstrukturen — wurde nicht-deterministisch definiert: Die Unifikation zweier Typen ergibt eine (evtl. leere) Menge von Typen; die Unifikation von Merkmalstrukturen ergibt eine (evtl. leere) Menge von Merkmalstrukturen.
4. Aus der nicht-deterministischen Definition der Unifikation ergibt sich ein vereinfachtes Resolutionsverfahren.

Die Synthese aus Carpenters und Emeles System wird ebenfalls als TFS bezeichnet. Oft wird auch allgemeiner von *Constraintsystemen* oder *Merkmalslogiken* gesprochen.

6 Das Kürzel *AVM* steht für die Initialen der englischen Bezeichnung *Attribute-Value-Matrix*.

1.1.1 Informelle Einführung

Sprachliche Zeichen lassen sich unter verschiedenen Aspekten beschreiben: sie haben eine *Form* und eine *Bedeutung*; sie sind gekennzeichnet durch Eigenschaften, die ihr *morphologisches*, *syntaktisches* und *semantisches* Verhalten bestimmen; sie haben bestimmte *Funktionen im Text* usw. Dies gilt nicht nur für die Wörter einer Sprache, sondern ebenso für Syntagmen, ganze Sätze oder Texte, sowie für morphologische und phonologische Bestandteile von Wörtern.

Sprachliche Ausdrücke aller linguistischen Beschreibungsebenen müssen folglich durch Mengen von Eigenschaften unterschiedlicher Qualität beschrieben werden. Ihr komplexes Verhalten erfordert das Zusammenspiel verschiedener, weitgehend unabhängiger Theorien, wie Syntax, Semantik und Morphologie.

Sollen die sprachlichen Ausdrücke formal beschrieben werden, liegt es damit nahe, ihre unterschiedlichen Eigenschaften hierarchisch zu ordnen. Die Eigenschaften einer Verbalphrase beispielsweise lassen sich in solche unterteilen, die ihre innere Struktur beschreiben, und solche, die sie als Gesamtheit charakterisieren. Letztere Gruppe kann weiter in die Klasse der semantischen und der syntaktischen Eigenschaften unterteilt werden usw. Dadurch ergibt sich eine hierarchische Struktur, die vieles gemein hat mit den Datenstrukturen moderner objektorientierter Programmiersprachen.

In der Linguistik — und insbesondere im Paradigma der Unifikationsgrammatiken — hat sich der Begriff *Merkmal-Wert-Matrix* (AVM)[7] für diese Art von Datenstrukturen eingebürgert. Eher mathematische Darstellungen bevorzugen den Begriff *Merkmalstruktur*. In diesem Kapitel werden Merkmal-Wert-Matritzen als Notation für Merkmalstrukturen eingeführt. Im linguistischen Teil der Arbeit ist diese Unterscheidung nicht wichtig, so daß beide Begriffe dort synonym verwendet werden.

1.1.2 Aufbau und Nomenklatur von Merkmalstrukturen

Merkmalstrukturen lassen sich nicht nur zur Darstellung und Gliederung von sprachlichen Zeichen (Morphen, Wörtern, Phrasen, Sätzen, Texten etc.) verwenden; auch die Bestandteile von Zeichen (Form, Bedeutung etc.) und die Regularitäten, die diese betreffen (Syntaktische Schemata, Prinzipien etc.), werden durch Merkmalstrukturen dargestellt. Um die Universalität von Merkmalstrukturen besser verstehen zu können, ist es notwendig, ihren Aufbau genauer zu betrachten. Sehen wir uns dazu drei unterschiedlich komplexe Beispiele an:

---

7 engl.: *Attribute-Value-Matrix.*

(1) *Merkmalstruktur für den Satz "Elizabeth glaubt, Sebastian liebt Sigrid":*

$$\begin{bmatrix}
\textit{phrase-rule} & \\
\text{PHON} & \left\langle \boxed{1}\ \textit{Elizabeth} \bullet \boxed{2} \left\langle \boxed{3}\ \textit{glaubt} \bullet \boxed{4} \left\langle \boxed{5}\ \textit{Sebastian} \bullet \boxed{6} \left\langle \boxed{7}\ \textit{liebt} \bullet \boxed{8} \left\langle \textit{Sigrid} \right\rangle \right\rangle \right\rangle \right\rangle \right\rangle \\
\text{CAT} & s \\
\text{CONT}\ \boxed{9} & \begin{bmatrix} \textit{believe} & \\ \textcircled{1}\ \boxed{10} & \textit{elizabeth} \\ \textcircled{2}\ \boxed{11} & \begin{bmatrix} \textit{love} & \\ \textcircled{1}\ \boxed{12} & \textit{sebastian} \\ \textcircled{2}\ \boxed{13} & \textit{sigrid} \end{bmatrix} \end{bmatrix} \\
\text{DTRS} & \left\langle \begin{bmatrix} \textit{lex-entry} & \\ \text{PHON} & \left\langle \boxed{1} \right\rangle \\ \text{CAT} & np \\ \text{CONT} & \boxed{10} \end{bmatrix}, \begin{bmatrix} \textit{phrase-rule} & \\ \text{PHON} & \boxed{2} \\ \text{CAT} & vp \\ \text{CONT} & \boxed{9} \\ \text{DTRS} & \left\langle \begin{bmatrix} \textit{lex-entry} & \\ \text{PHON} & \left\langle \boxed{3} \right\rangle \\ \text{CAT} & sv \\ \text{CONT} & \boxed{9} \end{bmatrix}, \begin{bmatrix} \textit{phrase-rule} & \\ \text{PHON} & \boxed{4} \\ \text{CAT} & s \\ \text{CONT} & \boxed{11} \\ \text{DTRS} & \left\langle \begin{bmatrix} \textit{lex-entry} & \\ \text{PHON} & \left\langle \boxed{5} \right\rangle \\ \text{CAT} & np \\ \text{CONT} & \boxed{12} \end{bmatrix}, \begin{bmatrix} \textit{phrase-rule} & \\ \text{PHON} & \boxed{6} \\ \text{CAT} & vp \\ \text{CONT} & \boxed{11} \\ \text{DTRS} & \left\langle \begin{bmatrix} \textit{lex-entry} & \\ \text{PHON} & \left\langle \boxed{7} \right\rangle \\ \text{CAT} & tv \\ \text{CONT} & \boxed{11} \end{bmatrix}, \begin{bmatrix} \textit{lex-entry} & \\ \text{PHON} & \boxed{8} \\ \text{CAT} & np \\ \text{CONT} & \boxed{13} \end{bmatrix} \right\rangle \end{bmatrix} \right\rangle \end{bmatrix} \right\rangle \end{bmatrix} \right\rangle
\end{bmatrix}$$

Diese Merkmalstruktur beschreibt — gegeben eine einfache formale Grammatik — Form, Syntax und Semantik des Satzes *Elizabeth glaubt, Sebastian liebt Sigrid*. Merkmalstrukturen können aber auch einfacher aussehen:

(2)
$$\begin{bmatrix}
\textit{phrase-rule} & \\
\text{PHON} & \left\langle \boxed{1}\ \textit{Elizabeth} \bullet \boxed{2} \left\langle \textit{schläft} \right\rangle \right\rangle \\
\text{DTRS} & \left\langle \begin{bmatrix} \textit{lex-entry} & \\ \text{PHON} & \left\langle \boxed{1} \right\rangle \end{bmatrix}, \begin{bmatrix} \textit{lex-entry} & \\ \text{PHON} & \boxed{2} \end{bmatrix} \right\rangle
\end{bmatrix}$$

(2) beschreibt Form und Syntax des Satzes *Elizabeth schläft*. Im einfachsten Fall besteht eine Merkmalstruktur nur aus einem Symbol:

(3) *Leonard*

Diese Struktur kann z.B. zur Beschreibung der phonologischen Form des Wortes *Leonard* verwendet werden.

Die wichtigsten Begriffe und Charakteristika von Merkmalstrukturen lassen sich anhand dieser Beispiele veranschaulichen:

Merkmale[8] — In den Merkmalstrukturen (1) und (2) befinden sich rechts neben den sich öffnenden eckigen Klammern Symbole, die in KAPITÄLCHEN gesetzt sind: Diese werden *Merkmale* genannt. Auch mit kleinen Kreisen umrandete Zahlen wie ① und ② sind Merkmale.

In (1) finden sich die Merkmale PHON, CAT, CONT, ①, ② und DTRS. Merkmalstruktur (2) enthält die Merkmale PHON und DTRS.

Typen — In allen drei Merkmalstrukturen befinden sich kursiv geschriebene Symbole, die sogenannten *Typen*:

- Das Zeichen (3) beispielsweise besteht nur aus dem Typ *Leonard*.
- In den Zeichen (1) und (2) befinden sich an der linken oberen Ecke sich öffnender, eckiger Klammern ebenfalls Typen: In (1) beispielsweise *phrase-rule*, *believe*, *love* und *lex-entry*; In (2) befinden sich die Typen *phrase-rule* und *lex-entry*.
- Hinter Merkmalen können ebenfalls Typen stehen: in (1) z.B. unter anderem *s*, *elizabeth*, *sebastian*, *sigrid*, *np*, *vp* usw.

Merkmal-Wert-Paare — Typisch für (1) und (2) sind Paare aus Merkmalen und Typen bzw. Klammerausdrücken. Erstere heißen *Merkmale*, letztere *Merkmalwerte*. Paare aus einem Merkmal und einem Merkmalwert heißen entsprechend *Merkmal-Wert-Paare*.

In (1) beispielsweise finden sich in der ersten Klammerebene folgende Merkmal-Wert-Paare:

(4)

$$
\begin{array}{ll}
\text{Merkmal:} & \text{Merkmalwert:} \\[1ex]
\textsc{phon} & \left\langle \boxed{1}\ \textit{Elizabeth}\ \bullet \boxed{2} \left\langle \boxed{3}\ \textit{glaubt}\ \ \ldots\ldots \right\rangle \right\rangle \\[1ex]
\textsc{cat} & \textit{s} \\[1ex]
\textsc{cont} & \boxed{9} \begin{bmatrix} \textit{believe} & \\ ① & \boxed{10}\ \textit{elizabeth} \\ ② & \boxed{11} \begin{bmatrix} \textit{love} \\ ①\ \boxed{12}\ \textit{sebastian} \\ ②\ \boxed{13}\ \textit{sigrid} \end{bmatrix} \end{bmatrix} \\[1ex]
\textsc{dtrs} & \left\langle \begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \left\langle \boxed{1} \right\rangle \\ \textsc{cat} & \textit{np} \\ \textsc{cont} & \boxed{10} \end{bmatrix}, \ldots\ldots \right\rangle
\end{array}
$$

8 engl.: *attribute* oder *feature*.

Atomare und komplexe Merkmalstrukturen — Die Menge der Merkmalstrukturen zerfällt in zwei Untermengen: die *atomaren* und die *komplexen* Merkmalstrukturen. Im Gegensatz zu den atomaren Merkmalstrukturen, die nur aus einem Typ bestehen,[9] setzen sich die komplexen Merkmalstrukturen aus einem Paar eckiger Klammern zusammen, die durch einen Typ gekennzeichnet werden und eine Menge von Merkmal-Wert-Paaren umschließen.

Beispiel (3) zeigt eine *atomare Merkmalstruktur*; (1) und (2) hingegen repräsentieren komplexe Merkmalstrukturen.

Listen — Neben den eckigen Klammern gibt es in (1) und (2) auch spitze Klammern, die ein oder mehrere durch Kommata getrennte Elemente enthalten. Diese Strukturen heißen *Listen*. Wie wir später sehen werden, sind die spitzen Klammern nur eine Abkürzung für spezielle, rekursive Strukturen aus eckigen Klammern.

Variablen — Zwischen Merkmalen und Werten oder an Stelle der Werte stehen in manchen Fällen rechteckig umrandete Zahlen: die *Variablen*. Steht hinter einer Variablen ein Wert, wird die Variable mit diesem belegt. Merkmale mit gleicher Variable haben (extensional) gleiche Werte.

Durch Merkmalstrukturen lassen sich damit, dank der Variablen, auch Graphen darstellen.

Rekursivität — Listen und Variablen sind nur Kürzel für eckig geklammerte Strukturen. Die *Rekursivität* kann daher als wichtigstes Strukturprinzip der Merkmalstrukturen festgestellt werden:

Merkmalstrukturen sind entweder:

atomar — d.h. sie bestehen nur aus einem Typ (vgl. (3)), oder

komplex — d.h. sie bestehen aus einem eckigen Klammerpaar, einem Typ und einer Menge von Merkmal-Wert-Paaren (vgl. (1) und (2)). Die Werte sind rekursiv nach dem gleichen Prinzip aufgebaut: Sie sind also selbst wiederum atomar oder komplex.

Merkmalstrukturen sind damit *rekursive Datenstrukturen*.

Merkmal-Wert-Matritzen (AVM)

Merkmalstrukturen sind formal gesehen Graphen. Wenn sie mit der Hilfe von Variablen trotzdem als Klammerstrukturen geschrieben werden, gibt es dafür einen einzigen, einfachen Grund: Klammerstrukturen können graphisch leichter dargestellt werden und sind leichter lesbar.

---

9 Atomare Merkmalstrukturen können mit oder ohne eckige Klammern dargestellt werden: [*Leonard*] und *Leonard* stehen also für die gleiche atomare Merkmalstruktur.

Für die geklammerte Form der Darstellung von Merkmalstrukturen soll von nun an der Begriff *Merkmal-Wert-Matrix (AVM)* Verwendung finden. Auch wenn die Begriffe *Merkmalstruktur* und *Merkmal-Wert-Matrix* in dem linguistischen Teil dieser Arbeit synonym gebraucht werden, sind AVMs unserer Definition nach nur eine *Notation* für Merkmalstrukturen. Merkmalstrukturen werden im Abschnitt 1.3 formal als Graphen definiert, da sich Operationen, wie die Unifikation, leichter für Graphen definieren lassen.

Die Beobachtungen des letzten Abschnittes lassen sich in einer einfachen Definition von AVMs zusammenfassen:

Definition 1.1 (Merkmal-Wert-Matritzen (AVM)[10])
Sei Type *eine endliche Menge von Typen,* Feat *eine endliche Menge von Merkmalen und* $\mathsf{Var} = \{\boxed{1}, \boxed{2}, \boxed{3}, \ldots\}$ *eine Menge von Variablen. Die Menge der* Merkmal-Wert-Matritzen (AVM) *über* Type *und* Feat *ist die kleinste Menge* AVM*, die die folgenden Bedingungen erfüllt:*

1. $$\begin{bmatrix} \sigma & \\ f_1 & \phi_1 \\ f_2 & \phi_2 \\ \vdots & \vdots \\ f_n & \phi_n \end{bmatrix} \in \mathsf{AVM} \quad \textit{falls} \quad \begin{array}{l} \sigma \in \mathsf{Type}, \\ n \in \mathbb{N}_0, \\ f_1, f_2, \ldots, f_n \in \mathsf{Feat}, \\ \phi_1, \phi_2, \ldots, \phi_n \in \mathsf{AVM}; \end{array}$$

   Für $n = 0$ *erhalten wir folglich den Spezialfall* $[\sigma]$*, für den wir auch einfach* σ *schreiben.*

2. $v \in \mathsf{AVM}$ *falls* $v \in \mathsf{Var}$

3. $v\phi \in \mathsf{AVM}$ *falls* $v \in \mathsf{Var}$ *und* $\phi \in \mathsf{AVM}$

---

10 Eine andere Notation wird in Carpenter 1992 (Def. 4.1 (Descriptions)) vorgestellt: Die Merkmal-Wert-Paare, die sich in unserer Darstellung in einer Klammerebene befinden, werden als logische Konjunktionen beschrieben, und Merkmale mit gleichen Werten werden als Gleichungen dargestellt:

Die Struktur

$$\begin{bmatrix} a & \\ \mathrm{X} & \boxed{1}\ b \\ \mathrm{Y} & \begin{bmatrix} c & \\ \mathrm{Z} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

beispielsweise wird in seiner Notation durch

$$a \wedge (\textsc{x} : b) \wedge (\textsc{y} : c) \wedge (\textsc{x} \doteq \textsc{y} \cdot \textsc{z})$$

dargestellt.

Atomare Merkmalstrukturen können nach dieser Definition mit oder ohne eckige Klammern geschrieben werden. Die folgenden beiden Strukturen sind also äquivalent:

(5) a. *Leonard*

b. $\begin{bmatrix} \textit{Leonard} \end{bmatrix}$

Nicht allen Elementen von AVM entsprechen auch Merkmalstrukturen:

(6)
$$\begin{bmatrix} a \\ \mathrm{X}\ \boxed{1}\ a \\ \mathrm{X}\ \boxed{1}\ b \end{bmatrix}$$

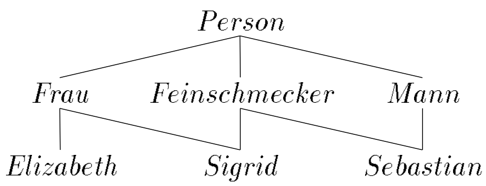
(6) beispielsweise läßt sich nicht sinnvoll interpretieren, da eine Variable nur mit einem Wert belegt werden kann.

1.2 Typenhierarchien

Im letzten Unterabschnitt wurden Merkmalstrukturen informell eingeführt. Wir haben damit eine erste Vorstellung, wie linguistische Informationen mit ihrer Hilfe dargestellt werden können. Da sich das Gesagte jedoch vorwiegend auf die *Form* von Merkmalstrukturen bezog, wurden andere, eher *inhaltliche Aspekte* bisher vernachlässigt. Einer dieser Aspekte bezieht sich auf die Organisation der Typen in Hierarchien, den sogenannten *Typenhierarchien*.

Sehen wir uns dazu ein Beispiel an:

(7)



Charakteristisch für Typenhierarchien sind folgende zwei Eigenschaften:

partielle Ordnung — *Typen* sind partiell geordnet. Sie können daher sowohl "Werte" als auch "Wertemengen"darstellen:

Wenn Carpenters Notation auch schwer zu lesen ist, eignet sie sich doch — dank ihres linearen Charakters — für mathematische Betrachtungen und die Verarbeitung durch den Computer. Carpenter verwendet sie als Syntax im Zusammenhang mit seiner modelltheoretischen Semantik.

- Der Typ *Person* kann die "Werte" *Frau*, *Feinschmecker*, *Mann*, *Elizabeth*, *Sigrid* und *Sebastian* annehmen.
- *Elizabeth*, *Sigrid* und *Sebastian* sind *minimale Typen*[11] und können nicht durch speziellere Typen ersetzt werden.
- *Frau*, *Feinschmecker* und *Mann* hingegen sind spezieller und damit "informativer" als *Person*, können aber trotzdem noch weiter präzisiert werden:
  - *Frau* durch *Elizabeth* oder *Sigrid*,
  - *Feinschmecker* durch *Sigrid* oder *Sebastian*,
  - *Mann* durch *Sebastian*.

größter Typ — Typenhierarchien enthalten einen *größten Typ*[12]: Dieser dominiert alle anderen Typen, ohne selber dominiert zu werden. Er wird durch das Symbol '$\top$' (gelesen: "top") bezeichnet.

In der Hierarchie (7) auf S. 13 beispielsweise ist *Person* ($= \top$) der größte Typ.

Aus den zwei Eigenschaften ergibt sich die Definition von Typenhierarchien:

Definition 1.2 (Typenhierarchie[13])
Eine Typenhierarchie *ist ein 3-Tupel* $\mathcal{H} = \langle \mathsf{Type}, \sqsupseteq, \top \rangle$, *das die folgenden Bedingungen erfüllt:*

1. Type *ist eine endliche Menge von Typen mit* $\top \in \mathsf{Type}$*;*
2. $\langle \mathsf{Type}, \sqsupseteq \rangle$ *ist eine partielle Ordnung;*
3. $\top$ *(gelesen: "top") ist das* größte Element[14] *hinsichtlich der partiellen Ordnung* $\langle \mathsf{Type}, \sqsupseteq \rangle$, *d.h.* $\forall_{\sigma \in \mathsf{Type}} : \top \sqsupseteq \sigma$.

11 *minimale* und *maximale Elemente* einer geordneten Menge M sind folgendermaßen definiert:

$$m = \text{maximales Element}(M) :\Longleftrightarrow m \in M \wedge \forall_{x \in M} \left[m \sqsubseteq x \Rightarrow x = m \right]$$
$$m = \text{minimales Element}(M) :\Longleftrightarrow m \in M \wedge \forall_{x \in M} \left[m \sqsupseteq x \Rightarrow x = m \right]$$

12 *größtes* und *kleinstes Element* einer geordneten Menge M sind folgendermaßen definiert:

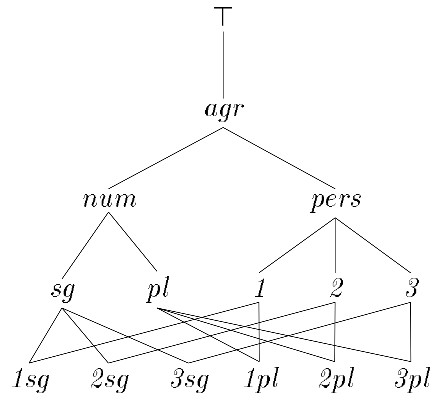
$$g = \text{größtes Element}(M) :\Longleftrightarrow g \in M \wedge \forall_{x \in M} \left[x \sqsubseteq g \right]$$
$$k = \text{kleinstes Element}(M) :\Longleftrightarrow k \in M \wedge \forall_{x \in M} \left[x \sqsupseteq k \right]$$

Subsumtion von Typen

Die partielle Ordnung $\sqsupseteq$ wird bei Typenhierarchien auch als *Subsumtionsordnung* bezeichnet. Für $\sigma \sqsupseteq \tau$ sagt man daher: "*σ subsumiert τ*" bzw. "*τ wird durch σ subsumiert*".

Das folgende Beispiel dient der Modellierung der *Kongruenzmerkmale*:

(8)



In (8) gelten unter anderem die folgenden Subsumtionsbeziehungen:

(9) *a.* $\forall_{\sigma \in \mathsf{Type}} : \top \sqsupseteq \sigma$;

b. $agr \sqsubseteq \top$;

c. $num \sqsubseteq agr$, $num \sqsubseteq \top$;

d. $sg \sqsubseteq num$, $sg \sqsubseteq agr$, $sg \sqsubseteq \top$;

e. $1sg \sqsubseteq sg$, $1sg \sqsubseteq num$, $1sg \sqsubseteq agr$, $1sg \sqsubseteq \top$, $1sg \sqsubseteq 1$, $1sg \sqsubseteq pers$;

Der Einfachheit halber schreiben wir für $\sigma \sqsupseteq \sigma_1, \sigma \sqsupseteq \sigma_2, ..., \sigma \sqsupseteq \sigma_n$ auch $\sigma \sqsupseteq \{\sigma_1, \sigma_2, ..., \sigma_n\}$

13 Carpenter 1992 definiert Typenhierarchien als *finite bounded complete partial order* $\langle \mathsf{Type}, \sqsupseteq \rangle$, vgl. Carpenter 1992, Def. 2.1 (*Inheritance Hierarchie*).

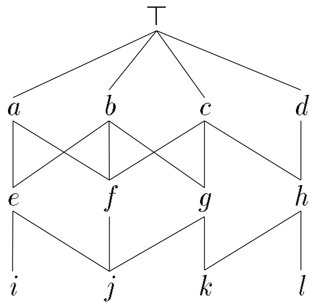
Dank dieser Definition läßt sich die Unifikation als partielle Funktion in die Menge der Typen definieren: Typen haben in seinem System also entweder einen Typ als eindeutiges Unifikat oder sie sind nicht unifizierbar.

14 Carpenter 1992 betrachtet Typenhierarchien als Träger von "Information": der Typ *Person* aus (7) enthält nach dieser Auffassung weniger Information als ein speziellerer Typ wie z.B. *Frau* oder *Sigrid*. Diese Interpretation der Typenhierarchie hat zur Folge, daß der allgemeinste Typ — d.h. der Typ, der von allen Typen am wenigsten Information trägt — nicht, wie in unserem Fall, *oberhalb*, sondern *unterhalb* aller anderen Symbole dargestellt wird. Carpenter verwendet daher für diesen Typ auch nicht das Symbol $\top$, sondern das Symbol $\bot$. Die Bezeichnungen *maximaler* und *minimaler Typ* sind in entsprechender Weise vertauscht. Auch die Subsumtionsordnung $\sqsupseteq$ wird von Carpenter genau spiegelverkehrt definiert.

Unifikation von Typen

Es gibt *unifizierbare*, d.h. "verträgliche", und *nicht unifizierbare*, d.h. "unverträgliche", Typen. *Unifizierbare Typen* besitzen gemeinsame Untertypen, *nicht unifizierbare Typen* haben keine gemeinsamen Untertypen:

(10)



In dieser Typenhierarchie gelten unter anderem die folgenden Beziehungen:

(11) *a.* $a \sqsupseteq \{e, f, i, j\}$

b. $b \sqsupseteq \{e, f, g, i, j, k\}$

c. $c \sqsupseteq \{f, g, h, j, k, l\}$

d. $d \sqsupseteq \{h, k, l\}$

b und c haben gemeinsame Untertypen und sind damit unifizierbar:

(12) $\{e, f, g, i, j, k\} \cap \{f, g, h, j, k, l\} = \{f, g, j, k\}$

a und d hingegen haben keine gemeinsamen Untertypen, sind also auch nicht unifizierbar:

(13) $\{e, f, i, j\} \cap \{h, k, l\} = \emptyset$

Die Schnittmenge der Untertypen enthält zu jedem ihrer Elemente auch alle seine Untertypen. Sie läßt sich daher durch die Menge ihrer maximalen Typen charakterisieren: Die Menge der Maximalen Elemente aus $\{f, g, j, k\}$ relativ zu (10) beispielsweise ist $\{f, g\}$.

Diese Menge — also die Menge der maximalen Typen aus der Menge der gemeinsamen Untertypen — definieren wir als *Unifikation*. Noch einfacher läßt sich die Unifikation mit dem Begriff der *maximalen unteren Schranke*[15] charakterisieren:

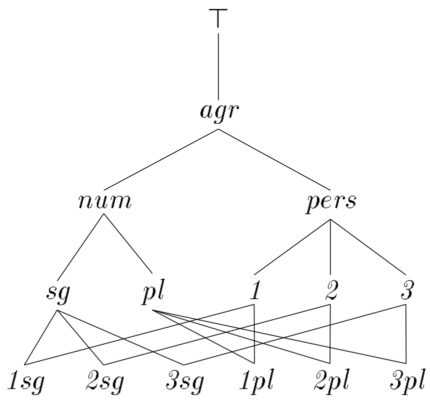
Definition 1.3 (Unifikation von Typen)
Gegeben sei eine Typenhierarchie $\langle \mathsf{Type}, \sqsupseteq, \top \rangle$. Unifikation *heißt die Funktion* $\sqcup : \mathsf{Type} \times \mathsf{Type} \longrightarrow 2^{\mathsf{Type}}$*, die jedem Tupel* $\langle \sigma, \tau \rangle \in \mathsf{Type} \times \mathsf{Type}$ *die Menge seiner maximalen unteren Schranken hinsichtlich der partiellen Ordnung* $\langle \mathsf{Type}, \sqsupseteq \rangle$ *zuweist.*

Für zwei Elemente σ, $\tau \in \mathsf{Type}$ ist $\sigma \sqcup \tau$ folglich nur dann ungleich der leeren Menge, wenn zumindest ein Element $\varphi \in \mathsf{Type}$ existiert, mit $\varphi \sqsubseteq \sigma$ und $\varphi \sqsubseteq \tau$. Für die Typenhierarchie (10) beispielsweise erhalten wir:

(14) *a.* $a \sqcup d = \emptyset$;
b. $b \sqcup c = \{f, g\}$;
c. $b \sqcup f = \{f\}$;
d. $b \sqcup b = \{b\}$;
e. $i \sqcup j = \emptyset$;

Warum diese Definition der Unifikation sinnvoll ist, läßt sich leichter anhand der Hierarchie der *Kongruenzmerkmale* verstehen, die hier noch einmal dargestellt wird:

(15)



15 Sei M eine geordnete Menge und $T \subseteq M$, so sind die Begriffe *untere* und *obere Schranke* folgendermaßen definiert:

$$s = \text{obere Schranke}(T) :\Longleftrightarrow s \in M \wedge \forall_{x \in T}\left[x \sqsubseteq s\right]$$
$$s = \text{untere Schranke}(T) :\Longleftrightarrow s \in M \wedge \forall_{x \in T}\left[x \sqsupseteq s\right]$$

Für diese Hierarchie gilt unter anderem:

(16) *a.* $sg \sqcup pl = \emptyset$;

b. $sg \sqcup sg = \{sg\}$;

c. $sg \sqcup agr = \{sg\}$;

d. $sg \sqcup 1 = \{1sg\}$;

e. $sg \sqcup pers = \{1sg, 2sg, 3sg\}$;

1.3 Merkmalstrukturen

Nachdem Typenhierarchien und die Begriffe Subsumtion und Unifikation von Typen formal eingeführt wurden, soll nun der Begriff *Merkmalstruktur* präzisiert werden.

Am Anfang dieses Kapitels wurden Merkmalstrukturen informell beschrieben und ihre AVM-Schreibweise vorgestellt. Für die formale Behandlung von Merkmalstrukturen ist es jedoch sinnvoll, sie als Graphen zu beschreiben. Zu diesem Zweck wollen wir die Struktur (2) wieder aufgreifen und schrittweise den ihr entsprechenden Graphen ableiten:

(17)
$$\begin{bmatrix} \textit{phrase-rule} & \\ \text{PHON} & \left\langle \boxed{1}\ \textit{Elizabeth} \ \bullet \ \boxed{2}\left\langle \textit{schläft} \right\rangle \right\rangle \\ \text{DTRS} & \left\langle \begin{bmatrix} \textit{lex-entry} \\ \text{PHON}\ \left\langle \boxed{1} \right\rangle \end{bmatrix}, \begin{bmatrix} \textit{lex-entry} \\ \text{PHON}\ \boxed{2} \end{bmatrix} \right\rangle \end{bmatrix}$$

Auflösung der Abkürzungen:

Listen werden in AVMs meistens durch spitze Klammern dargestellt. Diese sind jedoch nur Abkürzungen für rekursive Merkmalstrukturen:

elist steht für die leere Liste:

(18) Abkürzung: abgekürzte Merkmalstruktur:

$\langle\,\rangle$	*elist*

Nicht-leere Listen werden durch Merkmalstrukturen des Types *nelist* dargestellt. HEAD beschreibt das erste Element der Liste, TAIL die Restliste:

(19) Abkürzung: abgekürzte Merkmalstruktur:

$$\left\langle a,\ b,\ c \right\rangle \qquad \begin{bmatrix} \textit{nelist} & \\ \textsc{head} & a \\ \textsc{tail} & \begin{bmatrix} \textit{nelist} & \\ \textsc{head} & b \\ \textsc{tail} & \begin{bmatrix} \textit{nelist} & \\ \textsc{head} & c \\ \textsc{tail} & \textit{elist} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Das Zeichen '•' verbindet die ersten Elemente einer Liste mit der Restliste:

(20) Abkürzung: abgekürzte Merkmalstruktur:

$$\left\langle a,\ b \bullet \textit{list} \right\rangle \qquad \begin{bmatrix} \textit{nelist} & \\ \textsc{head} & a \\ \textsc{tail} & \begin{bmatrix} \textit{nelist} & \\ \textsc{head} & b \\ \textsc{tail} & \textit{list} \end{bmatrix} \end{bmatrix}$$

Ersetzen wir die Listen in (17) durch ihre Merkmalstrukturen und schreiben atomare Merkmalstrukturen in geklammerter Schreibweise (also [*Elizabeth*] statt *Elizabeth*, vgl. Def. 1.1 (Merkmal-Wert-Matritzen)) erhalten wir die folgende Darstellung:

(21)
$$\begin{bmatrix} \textit{phrase-rule} & \\ \textsc{phon} & \begin{bmatrix} \textit{nelist} & \\ \textsc{head} & \boxed{1}\ [\textit{Elizabeth}] \\ \textsc{tail} & \boxed{2} \begin{bmatrix} \textit{nelist} & \\ \textsc{head} & [\textit{schläft}] \\ \textsc{tail} & [\textit{elist}] \end{bmatrix} \end{bmatrix} \\ \textsc{dtrs} & \begin{bmatrix} \textit{nelist} & \\ \textsc{head} & \begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \begin{bmatrix} \textit{nelist} & \\ \textsc{head} & \boxed{1} \\ \textsc{tail} & [\textit{elist}] \end{bmatrix} \end{bmatrix} \\ \textsc{tail} & \begin{bmatrix} \textit{nelist} & \\ \textsc{head} & \begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \boxed{2} \end{bmatrix} \\ \textsc{tail} & [\textit{elist}] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Umsetzung der AVM in einen Graphen:

Merkmalstrukturen wie (21) werden folgendermaßen in Graphen umgesetzt:

1. *Klammern* und *Typen* werden durch Knoten dargestellt, an denen ihr Typ angetragen wird:

   (22) $\begin{bmatrix} \textit{lex-entry} \\ \dots \end{bmatrix} \longrightarrow \boxed{\textit{lex-entry}}$

   Die äußere Klammerebene entspricht dem Wurzelknoten (graphisch durch eine stärkere Umrandung hervorgehoben):

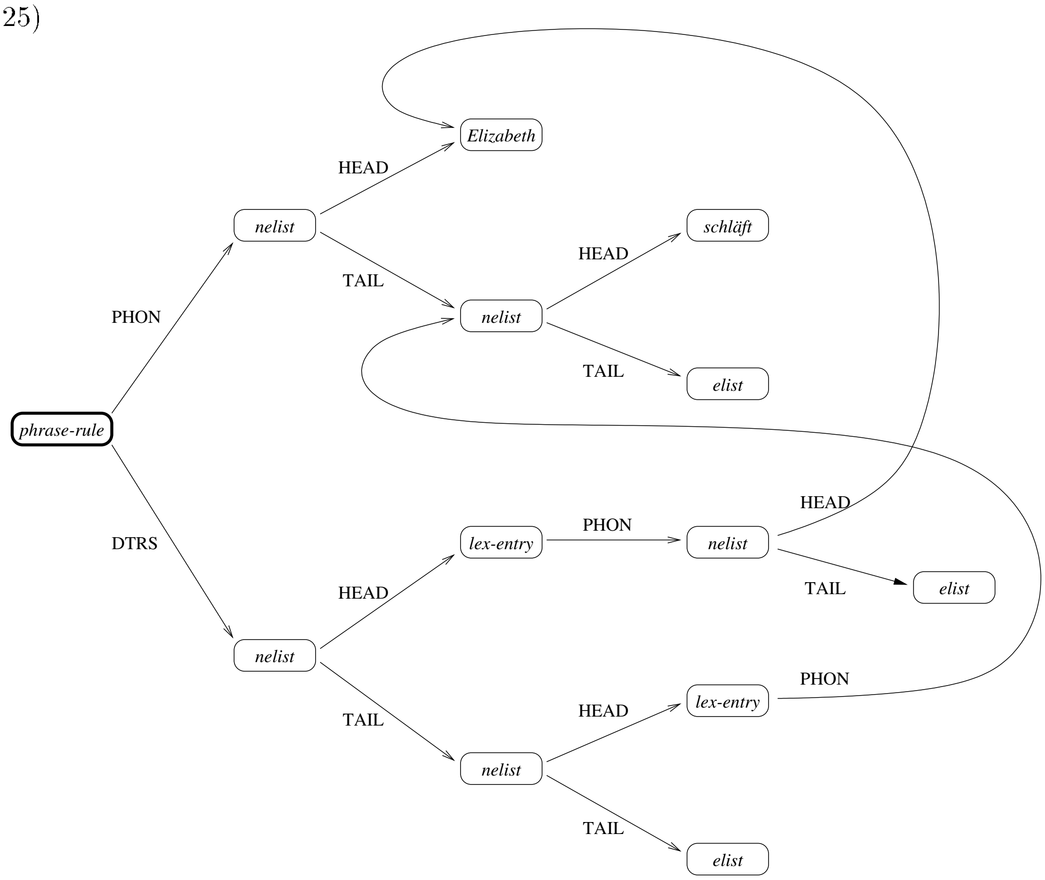
   (23) $\begin{bmatrix} \textit{phrase-rule} \\ \dots \end{bmatrix} \longrightarrow \boxed{\textit{phrase-rule}}$

2. *Merkmal-Wert-Paare* werden durch annotierte Kanten dargestellt:

   (24) $\begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \begin{bmatrix} \textit{nelist} \\ \dots \end{bmatrix} \\ \dots & \end{bmatrix} \longrightarrow \boxed{\textit{lex-entry}} \xrightarrow{\text{PHON}} \boxed{\textit{nelist}}$

Damit erhalten wir aus (21) den folgenden Graphen:

(25)



Formal läßt sich dieser Graph folgendermaßen beschreiben:

(26) 1. Q ist die Menge der Knoten:
$Q = \{q_0, q_1, q_2, \cdots, q_{12}\}$

2. $q_0 \in Q$ ist der Wurzelknoten

3. Die Funktion $\theta : Q \longrightarrow \mathsf{Type}$ ordnet jedem Knoten des Graphen einen Typ zu:

$$\begin{array}{lll} \theta(q_0) = \textit{phrase-rule}, & \theta(q_1) = \textit{nelist}, & \theta(q_2) = \textit{Elizabeth}, \\ \theta(q_3) = \textit{nelist}, & \theta(q_4) = \textit{schläft}, & \theta(q_5) = \textit{elist}, \\ \theta(q_6) = \textit{nelist}, & \theta(q_7) = \textit{lex-entry}, & \theta(q_8) = \textit{nelist}, \\ \theta(q_9) = \textit{elist}, & \theta(q_{10}) = \textit{nelist}, & \theta(q_{11}) = \textit{lex-entry}, \\ \theta(q_{12}) = \textit{elist}. & & \end{array}$$

4. Die partielle Funktion $\delta : \mathsf{Feat} \times Q \longrightarrow Q$ beschreibt die gerichteten Kanten mit den zugeordneten Merkmalen:

$$\begin{array}{ll}
\delta(\textsc{phon}, q_0) = q_1, & \delta(\textsc{dtrs}, q_0) = q_6, \\
\delta(\textsc{head}, q_1) = q_2, & \delta(\textsc{tail}, q_1) = q_3, \\
\delta(\textsc{head}, q_3) = q_4, & \delta(\textsc{tail}, q_3) = q_5, \\
\delta(\textsc{head}, q_6) = q_7, & \delta(\textsc{tail}, q_6) = q_{10}, \\
\delta(\textsc{phon}, q_7) = q_8, & \\
\delta(\textsc{head}, q_8) = q_2, & \delta(\textsc{tail}, q_8) = q_9, \\
\delta(\textsc{head}, q_{10}) = q_{11}, & \delta(\textsc{tail}, q_{10}) = q_{12}, \\
\delta(\textsc{phon}, q_{11}) = q_3. &
\end{array}$$

Eine Merkmalstruktur F ist also ein gerichteter, evt. zyklischer Graph $F = \langle Q, \bar{q}, \theta, \delta \rangle$ mit einem ausgezeichneten Wurzelknoten $q_0 \in Q$.

Ein Charakteristikum von Merkmalstrukturen ist, daß alle Kanten vom Wurzelknoten ausgehend mittels der gerichteten Kanten erreicht werden können. Man sagt: F ist in $\bar{q}$ *verwurzelt.*

Von unserem Beispiel ausgehend können wir nun den Begriff *Merkmalstruktur* definieren:

Definition 1.4 (Merkmalstruktur (MS)[16])
Sei Type *eine endliche Menge von Typen und* Feat *eine endliche Menge von Merkmalen. Ein Tupel* $F = \langle Q, \bar{q}, \theta, \delta \rangle$ *ist eine* Merkmalstruktur (MS)*, gdw. gilt:*

1. *Q ist eine endliche Menge von* Knoten*;*
2. *Es gibt einen ausgezeichneten Knoten $\bar{q} \in Q$, genannt* Wurzelknoten*;*
3. *$\theta : Q \longrightarrow \mathsf{Type}$ ist eine Funktion, die jedem Knoten des Graphen einen Typ zuordnet;*
4. *$\delta : \mathsf{Feat} \times Q \longrightarrow Q$ ist eine partielle Funktion, durch welche die gerichteten Kanten mit den zugeordneten Merkmalen beschrieben werden.*
5. *F ist verwurzelt in $\bar{q}$.*

Mit $\mathsf{MS}^{\mathsf{Type},\mathsf{Feat}}$ *sei die Menge der Merkmalstrukturen über* Type *und* Feat *bezeichnet.*

16 vgl. Carpenter 1992, Def. 3.1 (Feature Structure)

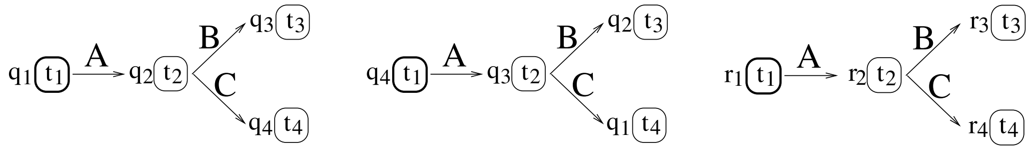
Alphabetische Varianten

Die AVM-Notation von Merkmalstrukturen unterscheidet sich in einem wesentlichen Punkt von der Graphendarstellung: Graphen, die bezüglich der Kanten und Typen isomorph sind, können sich hinsichtlich der Knoten unterscheiden; Die AVM-Notation hingegen abstrahiert von diesem Unterschied.

(27) *a.*

$$\begin{bmatrix} \mathit{t1} & \\ \textsc{a} & \begin{bmatrix} \mathit{t2} & \\ \textsc{b} & \mathit{t3} \\ \textsc{c} & \mathit{t4} \end{bmatrix} \end{bmatrix}$$

b. *i.* *ii.* *iii.*



In linguistischen Abhandlungen findet normalerweise ausschließlich die Matrixschreibweise Verwendung: Aus linguistischer Sicht ist nur die Struktur der Kanten und Typen interessant, nicht die Identität der Knoten.

Um die Abstraktion von den Knoten formal ausdrücken zu können, führen wir den Begriff *alphabetische Variante* ein:

Definition 1.5 (Alphabetische Variante[17])
Zwei Merkmalstrukturen F_1 *und* F_2 *sind* alphabetische Varianten, *wenn sie bezüglich ihrer Kanten und Typen isomorph sind.*

Wir schreiben in diesem Fall: $F_1 \sim F_2$

Die Isomorphie ist eine Äquivalenzbeziehung[18]. Die Menge der Merkmalstrukturen MS zerfällt damit in Äquivalenzklassen (Mengen alphabetischer Varianten) über die sich die Quotientenmenge (die Menge der Äquivalenzklassen) konstruieren läßt. Diese Quotientenmenge ist für uns die eigentlich interessante. Die Matrixschreibweise läßt sich als eine Art Repräsentantensystem[19] bezüglich der alphabetischen Varianten auffassen.

---

17 vgl. Carpenter 1992, Def. 3.6 (Alphabetic Variants): Carpenter definiert Alphabetische Varianten über die Subsumtionsbeziehung: *If F and F' are feature structures such that* $F \sqsubseteq F'$ *and* $F' \sqsubseteq F$*, then we write* $F \sim F'$ *and say that they are alphabetic variants.*

18 *Äquivalenzrelationen* sind Relationen, die reflexiv, symmetrisch und transitiv sind.

19 Auch die Matrixschreibweise ist nicht eindeutig, wie dieses für Repräsentantensysteme gefordert ist. Das folgende Beispiel beweist dieses:

$$\begin{bmatrix} \textsc{a} & \boxed{1}\ a \\ \textsc{b} & \boxed{1} \end{bmatrix} \sim \begin{bmatrix} \textsc{a} & \boxed{1} \\ \textsc{b} & \boxed{1}\ a \end{bmatrix} \sim \begin{bmatrix} \textsc{b} & \boxed{1}\ a \\ \textsc{a} & \boxed{1} \end{bmatrix} \sim \begin{bmatrix} \textsc{b} & \boxed{1} \\ \textsc{a} & \boxed{1}\ a \end{bmatrix}$$

Hilfsdefinitionen

Um die Ausweitung der Begriffe *Subsumtion* und *Unifikation* auf die Menge der Merkmalstrukturen MS im nächsten Kapitel zu vereinfachen, sollen an dieser Stelle einige Begriffe eingeführt werden, die die formale Behandlung von Merkmalstrukturen vereinfachen.

Merkmale verbinden als gerichtete Kanten die Knoten von Merkmalstrukturen. Oft ist es interessant, von einem Knoten ausgehend nacheinander mehreren Kanten zu folgen. Eine solche Folge von Kanten wird als *Pfad* bezeichnet:

Definition 1.6 (Pfad)
Ein Pfad $\pi \in \mathsf{Path}$ *ist eine Folge von Merkmalen, verbunden durch das Zeichen '|':*

1. $m \in \mathsf{Path}$ *falls* $m \in \mathsf{Feat}$*;*
2. $m|\pi \in \mathsf{Path}$ *falls* $m \in \mathsf{Feat}$*,* $\pi \in \mathsf{Path}$*;*
3. Path *ist die kleinste Menge, die 1. und 2. erfüllt.*

Oft wird auch ϵ — der leere Pfad bzw. Pfad der Länge 0 — zu Path gerechnet. Pfade werden als Abkürzungen auch in der AVM-Notation verwendet:

(28) Abkürzung: abgekürzte Merkmalstruktur:

$$\begin{bmatrix} \ldots \\ \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \;\; \textit{value} \\ \ldots \end{bmatrix} \quad \begin{bmatrix} \ldots & \\ \text{SYNSEM} & \begin{bmatrix} \top & \\ \text{LOCAL} & \begin{bmatrix} \top & \\ \text{CAT} & \textit{value} \end{bmatrix} \end{bmatrix} \\ \ldots & \end{bmatrix}$$

Die Gleichung $\delta(\text{M}, q) = q'$ beschreibt den Übergang von q zu q' entlang des Merkmals M (vgl. Def. 1.4 (Merkmalstruktur)):

(29)
$$q \xrightarrow{\text{M}} q'$$

Oft ist es interessant, zu wissen, welcher Knoten q' von q ausgehend entlang eines Pfades $\pi = \text{M}_1|\text{M}_2|...|\text{M}_i$ erreicht wird:

(30)
$$q \xrightarrow{\text{M}_1} q_1 \xrightarrow{\text{M}_2} q_2 \quad \ldots \quad q_{i-1} \xrightarrow{\text{M}_i} q'$$

Zu diesem Zweck weiten wir δ von der Menge der Merkmale Feat auf die Menge der Pfade Path aus:

(31) 1. $\delta(\epsilon, q) = q$

2. $\delta(m|\pi, q) = \delta(\pi, \delta(m, q))$

Merkmalstrukturen sind rekursive Datenstrukturen: Jeder Wert eines Pfades entspricht wiederum einer Merkmalstruktur.

Mit der partiellen Funktion $@ : \mathsf{MS} \times \mathsf{Path} \longrightarrow \mathsf{MS}$ lassen sich diese Unterstrukturen gewinnen:

Definition 1.7 (Wert eines Pfades[20])
Sei $F = \langle Q, \bar{q}, \theta, \delta\rangle$. Wenn $\delta(\pi, \bar{q})$ für F definiert ist, so wird die Unterstruktur $F@\pi = \langle Q', \bar{q}', \theta', \delta'\rangle$ durch die folgenden Bedingungen bestimmt:

1. $\bar{q}' = \delta(\pi, \bar{q})$
2. $Q' = \{\delta(\pi', \bar{q}') \mid \pi' \in \mathsf{Path}\} = \{\delta(\pi|\pi', \bar{q}) \mid \pi' \in \mathsf{Path}\}$
3. $\delta'(f, q) = \delta(f, q) \quad falls \quad q \in Q'$
4. $\theta'(q) = \theta(q) \quad falls \quad q \in Q'$

Subsumtion von Merkmalstrukturen

Genau wie Typen lassen sich Merkmalstrukturen in Subsumtionshierarchien anordnen. Wiederum spielt dabei der "Informationsgehalt" die entscheidende Rolle: In der Typenhierarchie (7) enthält der Typ *Mann* mehr Informationen als der Typ *Person*. Die Extension von *Mann* ist daher kleiner als die von *Person*: Sie entspricht der Menge der "Dinge", die nicht nur *Personen* sind, sondern auch *männlich*. In der Hierarchie (7) spiegelt sich dieser Zusammenhang wieder: Der Typ *Person* subsumiert den Typen *Mann*

Die Anordnung über den Informationsgehalt bzw. die Extension läßt sich in gleicher Weise auf Merkmalstrukturen ausdehnen:

20 vgl. Carpenter 1992, Def. 3.3 (Path Value), S.39.

1. Ausweitung der Typensubsumtion auf Merkmalstrukturen

Da Typen auch Merkmalstrukturen sind, gilt natürlich auch:

(32) $\textit{Person} \sqsupseteq \textit{Mann}.$ bzw. (33) $[\textit{Person}] \sqsupseteq [\textit{Mann}].$

Diese Beziehung setzt sich auch auf komplexe Merkmalstrukturen fort:

(34)
$$\begin{bmatrix} \textit{phrase} \\ \textsc{dtrs} \left\langle \begin{bmatrix} \textit{sign} \\ \textsc{phon} \left\langle \textit{Person} \right\rangle \end{bmatrix} \bullet \top \right\rangle \end{bmatrix} \sqsupseteq \begin{bmatrix} \textit{phrase} \\ \textsc{dtrs} \left\langle \begin{bmatrix} \textit{sign} \\ \textsc{phon} \left\langle \textit{Mann} \right\rangle \end{bmatrix} \bullet \top \right\rangle \end{bmatrix}$$

Aus der Perspektive des Informationsgehaltes, ist das erste Zeichen allgemeiner als das zweite: Die Aussage *"eine Phrase, deren erstes Zeichen eine* Person *beschreibt"* ist weniger informativ als die Aussage *"eine Phrase, deren erstes Zeichen eine* männliche Person (Mann) *beschreibt"*. Vom Standpunkt der Extension aus gesehen, enthält die Menge der Phrasen, die mit einem Zeichen beginnen, das eine *Person* beschreibt, die Menge der Phrasen, die mit einem Zeichen beginnen, das eine *männliche Person* beschreibt. Das erste Zeichen subsumiert also das zweite.

2. Stärker strukturierte Merkmalstrukturen werden von weniger stark strukturierten subsumiert

Stärker strukturierte Merkmalstrukturen enthalten mehr Information als weniger stark strukturierte Strukturen: Ein zusätzliches Merkmal-Wert-Paar beispielsweise fügt einer Merkmalstruktur weitere Information hinzu:

(35)
$$\begin{bmatrix} \textit{phrase} \\ \textsc{dtrs} \left\langle \begin{bmatrix} \textit{sign} \\ \textsc{phon} \left\langle \textit{Mann} \right\rangle \end{bmatrix} \bullet \top \right\rangle \end{bmatrix} \sqsupseteq \begin{bmatrix} \textit{phrase} \\ \textsc{cont} \;\; \textit{love} \\ \textsc{dtrs} \left\langle \begin{bmatrix} \textit{sign} \\ \textsc{phon} \left\langle \textit{Mann} \right\rangle \end{bmatrix} \bullet \top \right\rangle \end{bmatrix}$$

Während die linke Merkmalstruktur in (35) alle Phrasen beschreibt, die mit einer männlichen Person beginnen, beschreibt dir rechte nur solche Phrasen, die zusätzlich semantisch durch die Struktur *love* beschrieben werden.

Auch ein "Zusammenfallen" von Unterstrukturen bedeutet ein Zuwachs von Information:

(36)
$$\left\langle \top, \top \right\rangle \sqsupseteq \left\langle \boxed{1}\, \top, \boxed{1} \right\rangle$$

Natürlich gibt es auch Merkmalstrukturen, die sich nicht subsumieren:

(37) *a.* $\begin{bmatrix} \textit{phrase} \\ \text{PHON} \left\langle \textit{Mann} \right\rangle \end{bmatrix}$ und $\begin{bmatrix} \textit{phrase} \\ \text{CONT} \;\; \textit{love} \end{bmatrix}$

b. $\begin{bmatrix} \textit{Mann} \end{bmatrix}$ und $\begin{bmatrix} \textit{Frau} \end{bmatrix}$

Die Subsumtionshierarchie der Merkmalstrukturen ist damit, wie die Subsumtionshierarchie von Typen (Typenhierarchie), eine partielle Ordnung mit einem größten Element: größtes Element ist die am wenigsten strukturierte Merkmalstruktur mit dem allgemeinsten Type, also wiederum $\top$, bzw. $[\top]$.

Formale Definition der Subsumtion von Merkmalstrukturen

Formal läßt sich die Subsumtionsbeziehung von Merkmalstrukturen über einen *Morphismus* definieren: Alle Knoten der subsumierenden Struktur müssen auf Knoten der subsumierten Struktur abgebildet werden, die gleich oder kleiner (spezieller) getypt sind. Die Kanten der subsumierenden Struktur müssen sich in der subsumierten Struktur wiederfinden.

Definition 1.8 (Subsumtion von Merkmalstrukturen[21])
$F = \langle Q, \bar{q}, \theta, \delta \rangle$ subsumiert *$F' = \langle Q', \bar{q}', \theta', \delta' \rangle$, geschrieben $F \sqsupseteq F'$, genau dann, wenn es eine totale Funktion $h : Q \longrightarrow Q'$, genannt* Morphismus *gibt, mit:*

1. $h(\bar{q}) = \bar{q}'$
2. $\theta(q) \sqsupseteq \theta'(h(q))$ *für alle $q \in Q$*
3. $h(\delta(f, q)) = \delta'(f, h(q))$ *für alle $q \in Q$ und alle Merkmale $f \in$* Feat*,*
   für die $\delta(f, q)$ definiert ist

Unifikation von Merkmalstrukturen

So, wie sich die Subsumtionsbeziehung von Merkmalstrukturen aus der Subsumtionsbeziehung von Typen ableiten läßt, kann auch die Unifikation von Merkmalstrukturen aus der Unifikation von Typen entwickelt werden.

Die Unifikation zweier Typen ist die Menge ihrer *maximalen unteren Schranken* hinsichtlich der Typenhierarchie (vgl. Def. 1.3 (Unifikation von Typen)) Entsprechend ist die Unifikation zweier Merkmalstrukturen definiert als die

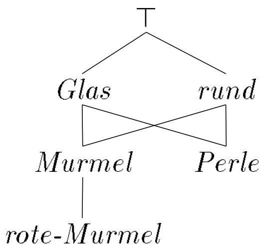
21 vgl. Carpenter 1992, Def. 3.4 (Subsumtion), S. 41.

Menge ihrer *maximalen unteren Schranken* hinsichtlich der Subsumtionshierarchie von Merkmalstrukturen.

Aus der informationstheoretischen Perspektive kann gesagt werden, daß jedes Element des Unifikates zweier Typen bzw. Merkmalstrukturen σ und τ für eine Möglichkeit steht, die Information beider Typen bzw. Merkmalstrukturen zu kombinieren, ohne weitere Information hinzuzufügen. Das Unifikat insgesamt bezeichnet alle Möglichkeiten dieser Kombination.

Betrachten wir zuerst ein Beispiel für die Unifikation von Typen:

(38)



Die “Welt”, die durch diese Typenhierarchie beschrieben wird, besteht aus Dingen aus Glas, runden Dingen, Murmeln, roten Murmeln und (Glas)perlen. Runde Dinge aus Glas können in diesem Universum nur Murmeln oder Perlen sein: Beide stehen hinsichtlich dieses speziellen Universums für eine allgemeinste Möglichkeit, die Informationen *Glas* und *rund* zu vereinen. Der Typ *rote-Murmel* hingegen ist nicht Element des Unifikates, da er zu den Informationen *Glas* und *rund* zusätzlich die Information *rot* hinzufügt.

Die Unifikation der Merkmalstrukturen ergibt sich entsprechend:

1. Ausweitung der Typen-Unifikation auf Merkmalstrukturen

Die Typenunifikation findet sich in der Unifikation von Merkmalstrukturen wieder. Analog zu dem folgenden Beispiel:

(39) $$num \sqcup \mathit{1} = \{\mathit{1sg}, \mathit{1pl}\}$$[22]

ergibt sich:

(40) $$\begin{bmatrix} num \end{bmatrix} \sqcup \begin{bmatrix} \mathit{1} \end{bmatrix} = \left\{ \begin{bmatrix} \mathit{1sg} \end{bmatrix}, \begin{bmatrix} \mathit{1pl} \end{bmatrix} \right\}$$

Übertragen auf ein komplexeres Beispiel:

22 vgl. die Typenhierarchie (8) bzw. (15).

(41)
$$\begin{bmatrix} \textit{phrase} \\ \text{DTRS} \left\langle \begin{bmatrix} \textit{sign} \\ \text{AGR} \ \ \textit{num} \end{bmatrix} \bullet \top \right\rangle \end{bmatrix} \sqcup \begin{bmatrix} \textit{phrase} \\ \text{DTRS} \left\langle \begin{bmatrix} \textit{sign} \\ \text{AGR} \ \ \textit{1} \end{bmatrix} \bullet \top \right\rangle \end{bmatrix}$$
$$= \left\{ \begin{bmatrix} \textit{phrase} \\ \text{DTRS} \left\langle \begin{bmatrix} \textit{sign} \\ \text{AGR} \ \ \textit{1sg} \end{bmatrix} \bullet \top \right\rangle \end{bmatrix}, \begin{bmatrix} \textit{phrase} \\ \text{DTRS} \left\langle \begin{bmatrix} \textit{sign} \\ \text{AGR} \ \ \textit{1pl} \end{bmatrix} \bullet \top \right\rangle \end{bmatrix} \right\}$$

1. Unifikation durch Kombination der "strukturellen Information"

Jedes Pfad-Merkmal-Paar aus einer der zu unifizierenden Merkmalstrukturen findet sich im Unifikat wieder:

(42)
$$\begin{bmatrix} \textit{sign} \\ \text{AGR} & \textit{num} \\ \text{CAT} & \textit{v} \end{bmatrix} \sqcup \begin{bmatrix} \textit{sign} \\ \text{AGR} & \textit{1} \\ \text{CONT} & \textit{sleep} \end{bmatrix} = \left\{ \begin{bmatrix} \textit{sign} \\ \text{AGR} & \textit{1sg} \\ \text{CAT} & \textit{v} \\ \text{CONT} & \textit{sleep} \end{bmatrix}, \begin{bmatrix} \textit{sign} \\ \text{AGR} & \textit{1pf} \\ \text{CAT} & \textit{v} \\ \text{CONT} & \textit{sleep} \end{bmatrix} \right\}$$

Koindizierte Pfade, d.h. Pfade mit identischem Wert, finden sich natürlich auch im Unifikat wieder:

(43)
$$\left\langle \top, \top \right\rangle \sqcup \left\langle \boxed{1} \top, \boxed{1} \right\rangle = \left\{ \left\langle \boxed{1} \top, \boxed{1} \right\rangle \right\}$$

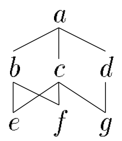
Anschaulich läßt sich die Unifikation von Merkmalstrukturen als *Projektion* der zu unifizierenden Strukturen *aufeinander* beschreiben:

1. Das Unifikat enthält alle Pfade, die in den Ausgangsstrukturen vorkommen;

2. Typen, die vom Wurzelknoten aus gesehen am Ende des gleichen Pfades liegen, und — im Sinne dieses Bildes gesprochen — aufeinander projiziert werden, werden durch ein Element ihrer Typen-Unifikation ersetzt.

Die verschiedenen Fälle, die bei einer Unifikation auftreten können, lassen sich am besten anhand von Beispielen veranschaulichen. Wir setzen die folgende Typenhierarchie voraus:

(44)



Für die Typenunifikation ergeben sich unter anderem folgende Fälle:

(45) a. $a \sqcup a = \{a\}$

b. $a \sqcup b = \{b\}$

c. $b \sqcup d = \emptyset$

d. $c \sqcup d = \{g\}$

e. $b \sqcup c = \{e, f\}$

Mit den Merkmalen $\mathsf{Feat} = \{1, 2\}$ können unter anderem die folgenden Beispiele der Unifikation gebildet werden:

Unifikation ohne Veränderung der Struktur:

(46) a. $$\begin{bmatrix} a & \\ 1 & a \end{bmatrix} \sqcup \begin{bmatrix} a & \\ 1 & b \end{bmatrix} = \left\{ \begin{bmatrix} a & \\ 1 & b \end{bmatrix} \right\}$$

b. $$\begin{bmatrix} a & \\ 1 & b \end{bmatrix} \sqcup \begin{bmatrix} a & \\ 1 & d \end{bmatrix} = \emptyset$$

c. $$\begin{bmatrix} c & \\ 1 & b \end{bmatrix} \sqcup \begin{bmatrix} d & \\ 1 & c \end{bmatrix} = \left\{ \begin{bmatrix} g & \\ 1 & e \end{bmatrix}, \begin{bmatrix} g & \\ 1 & f \end{bmatrix} \right\}$$

d. $$\begin{bmatrix} b & \\ 1 & b \end{bmatrix} \sqcup \begin{bmatrix} c & \\ 1 & c \end{bmatrix} = \left\{ \begin{bmatrix} e & \\ 1 & e \end{bmatrix}, \begin{bmatrix} e & \\ 1 & f \end{bmatrix}, \begin{bmatrix} f & \\ 1 & e \end{bmatrix}, \begin{bmatrix} f & \\ 1 & f \end{bmatrix} \right\}$$

Kombination "struktureller Information":

(47) a. $$\begin{bmatrix} a & \\ 1 & a \end{bmatrix} \sqcup \begin{bmatrix} a & \\ 2 & a \end{bmatrix} = \left\{ \begin{bmatrix} a & \\ 1 & a \\ 2 & a \end{bmatrix} \right\}$$

b. $$\begin{bmatrix} a & \\ 1 & b \\ 2 & e \end{bmatrix} \sqcup \begin{bmatrix} f & \\ 2 & \begin{bmatrix} a & \\ 1 & c \end{bmatrix} \\ 3 & a \end{bmatrix} = \left\{ \begin{bmatrix} f & \\ 1 & b \\ 2 & \begin{bmatrix} e & \\ 1 & c \end{bmatrix} \\ 3 & a \end{bmatrix} \right\}$$

Koindizierung von Merkmalwerten:

(48) a.
$$\begin{bmatrix} b & \\ 1 & c \\ 2 & a \end{bmatrix} \sqcup \begin{bmatrix} c & \\ 1 & \boxed{1}\ d \\ 2 & \boxed{1} \end{bmatrix} = \left\{ \begin{bmatrix} e & \\ 1 & \boxed{1}\ g \\ 2 & \boxed{1} \end{bmatrix}, \begin{bmatrix} f & \\ 1 & \boxed{1}\ g \\ 2 & \boxed{1} \end{bmatrix} \right\}$$

b.
$$\begin{bmatrix} b & \\ 1 & c \\ 2 & b \end{bmatrix} \sqcup \begin{bmatrix} c & \\ 1 & \boxed{1}\ d \\ 2 & \boxed{1} \end{bmatrix} = \emptyset$$

Bei der Unifikation können aus *nicht-zyklischen* Merkmalstrukturen *zyklische*[23] entstehen:

(50)
$$\begin{bmatrix} a & \\ 1 & \boxed{1} \\ 2 & \boxed{1} \end{bmatrix} \sqcup \begin{bmatrix} a & \\ 1 & \begin{bmatrix} a & \\ 1 & \boxed{1} \end{bmatrix} \\ 2 & \boxed{1} \end{bmatrix} = \left\{ \begin{bmatrix} a & \\ 1 & \boxed{1} \begin{bmatrix} a & \\ 1 & \boxed{1} \end{bmatrix} \\ 2 & \boxed{1} \end{bmatrix} \right\}$$

Operationale Beschreibung der Unifikation

Die Unifikation läßt sich auch konstruktiv beschreiben:

Von zwei Strukturen F_1 und F_2 ausgehend wird eine dritte Struktur F_u — Element des Unifikates von F_1 und F_2 — rekursiv mittels der folgenden Schritte konstruiert:

1. Als Typ des Wurzelknotens von F_u wird ein beliebiges Element aus der Unifikation der Typen der Wurzelknoten von F_1 und F_2 ausgewählt;

2. Die Menge der Merkmale, für die F_u am Wurzelknoten definiert ist, wird festgelegt als die Vereinigung der Merkmale, für die F_1 und F_2 am Wurzelknoten definiert sind;

3. Die Werte der Merkmale von F_u am Wurzelknoten ergeben sich durch Auswahl eines Elementes aus dem Unifikat der entsprechenden Merkmalwerte von F_1 und F_2.

Diese drei Schritte werden rekursiv die Strukturen traversierend so lange wiederholt, bis das Ergebnis sich nicht mehr ändert.

23 Beispiel einer einfachen *zyklischen Merkmalstruktur*:

(49)
$$\boxed{1} \begin{bmatrix} a & \\ 1 & \boxed{1} \end{bmatrix}$$

Definition der Unifikation

Bei der formalen Definition der Unifikation lehnen wir uns an Carpenter 1992 an. Im Gegensatz zur Definition in Carpenter 1992 ist die Unifikation von Merkmalstrukturen allerdings nicht als Funktion in die Menge der Merkmalstrukturen definiert, sondern als Funktion in die Potenzmenge der Merkmalstrukturen.

Definition 1.9 (Unifikation[24])
Gegeben seien zwei Merkmalstrukturen $F_1, F_2 \in \mathsf{MS}$. *Mit* $\langle Q_1, \bar{q}_1, \theta_1, \delta_1\rangle \sim F_1$ *und* $\langle Q_2, \bar{q}_2, \theta_2, \delta_2\rangle \sim F_2$ *wählen wir zu diesen alphabetische Varianten, derart, daß* $Q_1 \cap Q_2 = \emptyset$. *Die Äquivalenz-Relation*[25] $\bowtie$ *auf* $Q_1 \cup Q_2$ *definieren wir als die kleinste Relation, die die folgenden Bedingungen erfüllt:*

1. $\bar{q}_1 \bowtie \bar{q}_2$
2. $\delta(f, q_1) \bowtie \delta(f, q_2) \quad \textit{falls} \quad q_1 \bowtie q_2$
   $\textit{und} \quad \delta(f, q_1)\ \textit{und}\ \delta(f, q_2)\ \textit{definiert}$

Die Unifikation $F_1 \sqcup F_2$ *von* F_1 *und* F_2 *ist definiert durch:*

$$F_1 \sqcup F_2 = \{\langle (Q_1 \cup Q_2)/_{\bowtie}, [\bar{q}_1]_{\bowtie}, \theta^{\bowtie}, \delta^{\bowtie}\rangle \mid \theta^{\bowtie}([q]_{\bowtie}) \in \bigsqcup\{(\theta_1 \cup \theta_2)(q') \mid q' \bowtie q\}\}$$

wobei:

$$\delta^{\bowtie}(f, [q]_{\bowtie}) = \begin{cases} [(\delta_1 \cup \delta_2)(f, q)]_{\bowtie} \\ \quad\quad falls \quad (\delta_1 \cup \delta_2)(f, q) \quad definiert \\ undefiniert \\ \quad\quad ansonsten \end{cases}$$

.

24 vgl. Carpenter 1992, Def. 3.11 (Unification)

25 Für eine Äquivalenz-Relation (also eine reflexive, symmetrische und transitive Relation) $\equiv\ \subseteq M \times M$ ist die Äquivalenzklasse von $x \in M$ folgendermaßen definiert:

(51) $[x]_{\equiv} \quad = \quad \{y \in M \mid y \equiv x\}$

Die Quotientenmenge $M/_{\equiv}$ von M modulo $\equiv$ ist die Menge der Äquivalenzklassen, in die M durch $\equiv$ zerfällt:

(52) $M/_{\equiv} \quad = \quad \{[x]_{\equiv} \mid x \in M\}$

1.4 Constraints

Merkmalslogiken sind Formalismen zur Modellierung von Gegenstandsbereichen. Die Objekte eines Gegenstandsbereiches werden im Modell durch Merkmalstrukturen repräsentiert. Ist der Gegenstandsbereich, wie in unserem Falle, ein Ausschnitt der japanischen Sprache, repräsentieren Merkmalstrukturen also Wörter, Phrasen, Sätze, semantische Strukturen etc.

Die Wahl der Merkmale und die Definition der Typenhierarchie bilden den ersten Teil der Modellierung. Durch die Menge der Typen und Merkmale wird, wie in Def. 1.4 (Merkmalstruktur) dargestellt, eine unendliche Menge von Merkmalstrukturen definiert. Die Form der möglichen Merkmalstrukturen über einer Menge von Typen und einer Menge von Merkmalen wird bisher jedoch allein durch die Kombinatorik bestimmt. Nur wenige der kombinatorisch möglichen Strukturen sind jedoch sinnvolle Beschreibungen von Objekten.

Gibt es beispielsweise die Merkmale PHON und CAT und die Typen *Elizabeth*, *Sigrid*, *np* und *lex-entry* lassen sich Strukturen bilden wie:

(53) *a.* $\boxed{1}\begin{bmatrix} \textit{Elizabeth} & \\ \text{PHON} & \boxed{1} \\ \text{CAT} & \boxed{1} \end{bmatrix}$ *b.* $\begin{bmatrix} \textit{Elizabeth} & \\ \text{PHON} & \begin{bmatrix} \textit{np} & \\ \text{CAT} & \begin{bmatrix} \textit{Sigrid} & \\ \text{PHON} & \textit{lex-entry} \end{bmatrix} \end{bmatrix} \end{bmatrix}$

Diese allerdings ergeben wenig Sinn — im Gegensatz zu den folgenden:

(54) *a.* $\begin{bmatrix} \textit{lex-entry} & \\ \text{PHON} & \textit{Elizabeth} \\ \text{CAT} & \textit{np} \end{bmatrix}$ *b.* $\begin{bmatrix} \textit{lex-entry} & \\ \text{PHON} & \textit{Sigrid} \\ \text{CAT} & \textit{np} \end{bmatrix}$

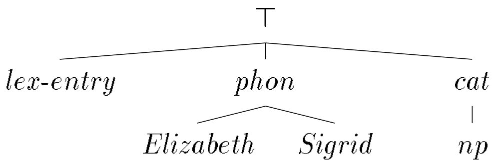
Es ist also notwendig, sinnvolle von nicht sinnvollen Merkmalstrukturen durch ein formales Mittel unterscheiden zu können. Diesem Zweck dienen formale Bedingungen, die sogenannten *Constraints*: Nur solche Strukturen sind sinnvoll oder — um diesen inhaltlichen Begriff durch einen formalen zu ersetzen — *aufgelöst* (*resolved*), die alle Constraints des Systems erfüllen.

In unserem Fall wäre beispielsweise der folgende Constraint denkbar:

(55)
$$\textit{lex-entry} \quad \Rightarrow \quad \begin{bmatrix} \textit{lex-entry} & \\ \text{PHON} & \textit{phon} \\ \text{CAT} & \textit{cat} \end{bmatrix}$$

Ein Zeichen vom Typ *lex-entry* hat zwei Merkmale: PHON und CAT. Das Merkmal PHON kann Subtypen vom Typ *phon* als Werte annehmen; Das Merkmal CAT Subtypen vom Typ *cat*. Eine sinnvolle Typenhierarchie für diese Beschränkung wäre:

(56)



Aus diesem Beispiel läßt sich erahnen, wie der Modellbezug zwischen Gegenstandsbereich und Constraintsystem hergestellt wird: Die Modellierung eines Gegenstandsbereiches durch ein Constraintsystem besteht darin, Typen, Merkmale und Constraints so zu wählen, daß jedes Objekt des Gegenstandsbereiches durch eine aufgelöste Merkmalstruktur beschrieben wird und gleichzeitig jede aufgelöste Struktur einem Objekt des Gegenstandsbereiches entspricht.

Soll — wie in dieser Arbeit — ein Fragment einer natürlichen Sprache durch ein Constraintsystem beschrieben werden, muß also

1. jeder korrekte sprachliche Ausdruck durch eine aufgelöste Merkmalstruktur beschrieben werden,[26] und

2. jeder aufgelösten Merkmalstruktur ein korrekter sprachlicher Ausdruck entsprechen.[26]

Die Rolle der Unifikation für Constraintsysteme läßt sich ebenfalls ableiten: Constraints, die in unserem System ebenfalls die Form von Merkmalstrukturen haben, formulieren *partielle* Anforderungen an Merkmalstrukturen. Diese partiellen Anforderungen oder Informationen werden mit dem Mittel der *Unifikation* kombiniert.

Typen **als "Angriffspunkte" der Constraints**

Anhand einiger einfacher Beispiele wollen wir uns die Funktionsweise von Constraints veranschaulichen:

26 Wir abstrahieren hier von ambigen sprachlichen Ausdrücken, beispielsweise Homophonen, strukturell mehrdeutigen Sätzen etc., und setzen die Vereindeutigung sprachlicher Ausdrücke voraus, z.B. durch Indizierung, geeignete Klammerung etc.

Adäquatheit von Merkmalen — Constraints bestimmen, welche Merkmale in einer Klammerebene adäquat sind: Die Unterstrukturen des Typs *sign* in der Struktur (1) beispielsweise enthalten immer die Merkmale PHON und CAT. Ein Constraint, der dieses sicherstellt, könnte z.B. folgendermaßen aussehen:

(57)
$$sign \quad \Rightarrow \quad \begin{bmatrix} \textit{sign} & \\ \textsc{phon} & \top \\ \textsc{cat} & \top \end{bmatrix}$$

Adäquatheit von Merkmalwerten — Ist ein Merkmal für einen Typ adäquat, so bestimmen Constraints, welche Werte dieses Merkmal annehmen darf. Der Constraint (57) beispielsweise läßt sich leicht in dieser Hinsicht erweitern:

(58)
$$sign \quad \Rightarrow \quad \begin{bmatrix} \textit{sign} & \\ \textsc{phon} & \textit{list} \\ \textsc{cat} & \textit{cat} \end{bmatrix}$$

(58) bestimmt, daß das Merkmal PHON einen Wert des Typs *list* und das Merkmal CAT einen Wert des Typs *cat* annehmen muß.

Abhängigkeiten unter Merkmalen — Zusammenhänge zwischen Merkmalwerten von Zeichen können ebenfalls durch Constraints sichergestellt werden:

(59)
$$phrase \quad \Rightarrow \quad \begin{bmatrix} \textit{phrase} & \\ \textsc{phon} & \boxed{1} \circ \boxed{2} \\ \textsc{dtrs} & \left\langle \begin{bmatrix} \textsc{phon} & \boxed{1} \end{bmatrix}, \begin{bmatrix} \textsc{phon} & \boxed{2} \end{bmatrix} \right\rangle \end{bmatrix}$$

(59) erzwingt, daß der PHON-Wert von Merkmalstrukturen des Typs *phrase* der Konkatenation der PHON-Werte seiner Töchter entspricht.[27]

27 Der Ausdruck "$\boxed{1} \circ \boxed{2}$" beschreibt die Konkatenation der Listen "$\boxed{1}$" und "$\boxed{2}$" und wird ebenfalls durch Constraints realisiert (vgl. Abschnitt "Listen und Listen-Konkatenation", S. 47).

Lexikalische Einträge als Constraints — Auch das Lexikon wird durch Constraints beschrieben. Lexikalische Einträge für

(60) *a.* "Leonard" (np)

b. "schläft" (vp)

c. "trinkt" (vp)

lassen sich z.B. mit den folgenden Constraints beschreiben ($\vee$ steht für die logische Disjunktion):

(61)
$$\textit{lex-entry} \Rightarrow \begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \left\langle \textit{Leonard} \right\rangle \\ \textsc{cat} & \textit{np} \end{bmatrix} \vee \begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \left\langle \textit{schläft} \right\rangle \\ \textsc{cat} & \textit{vp} \end{bmatrix} \vee \begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \left\langle \textit{trinkt} \right\rangle \\ \textsc{cat} & \textit{vp} \end{bmatrix}.$$

Phrasenstrukturregeln als Constraints — Phrasenstrukturregeln beschreiben, wie Wörter und Phrasen zu Phrasen und Sätzen kombiniert werden können. Die Regel

(62)
$$\mathrm{S} \longrightarrow \mathrm{NP}\ \mathrm{VP}$$

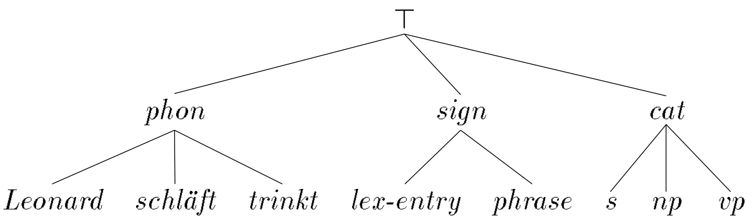
beispielsweise kombiniert eine Nominalphrase mit einer Verbalphrase zu einem Satz. Als Constraint läßt sie sich folgendermaßen schreiben:

(63)
$$\textit{phrase} \Rightarrow \begin{bmatrix} \textit{phrase} & \\ \textsc{cat} & \textit{s} \\ \textsc{dtrs} & \left\langle \begin{bmatrix} \textit{sign} & \\ \textsc{cat} & \textit{np} \end{bmatrix}, \begin{bmatrix} \textit{sign} & \\ \textsc{cat} & \textit{vp} \end{bmatrix} \right\rangle \end{bmatrix}.$$

Eine kleine Grammatik

Wenn wir zu den Constraints (58), (59), (61) und (63) die Typenhierarchie (64) annehmen, erhalten wir eine erste kleine Grammatik:

(64)



Wir wollen uns ansehen, wie das folgende Zeichen erweitert werden muß, damit es alle Constraints erfüllt:

(65)
$$\left[\textit{phrase}\right]$$

Ein Constraint $\sigma \Rightarrow F$ muß auf alle Strukturen angewandt werden, die durch σ subsumiert werden. Da der Typ *phrase* ein Subtyp von *sign* ist, muß der Constraint (58) also auch auf (65) angewandt werden. Durch Unifikation der rechten Seite von (58) mit (65) erhalten wir:

(66)
$$\left[\begin{array}{ll}\multicolumn{2}{l}{\textit{phrase}}\\ \textsc{phon} & \textit{list}\\ \textsc{cat} & \textit{cat}\end{array}\right]$$

Der Constraint (63) muß ebenfalls angewandt werden:

(67)
$$\left[\begin{array}{ll}\multicolumn{2}{l}{\textit{phrase}}\\ \textsc{phon} & \textit{list}\\ \textsc{cat} & \textit{s}\\ \textsc{dtrs} & \left\langle \left[\begin{array}{ll}\multicolumn{2}{l}{\textit{sign}}\\ \textsc{cat} & \textit{np}\end{array}\right], \left[\begin{array}{ll}\multicolumn{2}{l}{\textit{sign}}\\ \textsc{cat} & \textit{vp}\end{array}\right]\right\rangle\end{array}\right]$$

Durch Anwendung von (58) auf die Töchter von (67) ergibt sich:

(68)
$$\left[\begin{array}{ll}\multicolumn{2}{l}{\textit{phrase}}\\ \textsc{phon} & \textit{list}\\ \textsc{cat} & \textit{s}\\ \textsc{dtrs} & \left\langle \left[\begin{array}{ll}\multicolumn{2}{l}{\textit{sign}}\\ \textsc{phon} & \textit{list}\\ \textsc{cat} & \textit{np}\end{array}\right], \left[\begin{array}{ll}\multicolumn{2}{l}{\textit{sign}}\\ \textsc{phon} & \textit{list}\\ \textsc{cat} & \textit{vp}\end{array}\right]\right\rangle\end{array}\right]$$

Zeichen des typs *sign* subsumieren auch Zeichen des Typs *lex-entry* und *phrase*. Da die Töchter von (68) jedoch die Kategorie *np* bzw. *vp* haben müssen, läßt sich (63) nicht anwenden. (61) hingegen kann benutzt werden:

Die erste Tochter unifiziert mit

(69)
$$\left[\begin{array}{ll}\multicolumn{2}{l}{\textit{lex-entry}}\\ \textsc{phon} & \left\langle \textit{Leonard} \right\rangle\\ \textsc{cat} & \textit{np}\end{array}\right]$$

die zweite Tochter wahlweise mit

(70) $$\begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \left\langle \textit{schläft} \right\rangle \\ \textsc{cat} & \textit{vp} \end{bmatrix}$$ oder (71) $$\begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \left\langle \textit{trinkt} \right\rangle \\ \textsc{cat} & \textit{vp} \end{bmatrix}$$

Wir erhalten damit zwei mögliche Fortsetzungen:

(72) a. $$\begin{bmatrix} \textit{phrase} & \\ \textsc{phon} & \textit{list} \\ \textsc{cat} & \textit{s} \\ \textsc{dtrs} & \left\langle \begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \left\langle \textit{Leonard} \right\rangle \\ \textsc{cat} & \textit{np} \end{bmatrix}, \begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \left\langle \textit{schläft} \right\rangle \\ \textsc{cat} & \textit{vp} \end{bmatrix} \right\rangle \end{bmatrix}$$

b. $$\begin{bmatrix} \textit{phrase} & \\ \textsc{phon} & \textit{list} \\ \textsc{cat} & \textit{s} \\ \textsc{dtrs} & \left\langle \begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \left\langle \textit{Leonard} \right\rangle \\ \textsc{cat} & \textit{np} \end{bmatrix}, \begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \left\langle \textit{trinkt} \right\rangle \\ \textsc{cat} & \textit{vp} \end{bmatrix} \right\rangle \end{bmatrix}$$

Durch den Constraint (59) schließlich ergeben sich die folgenden Strukturen:

(73) a. $$\begin{bmatrix} \textit{phrase} & \\ \textsc{phon} & \boxed{1}\left\langle \textit{Leonard} \right\rangle \circ \boxed{2}\left\langle \textit{schläft} \right\rangle \\ \textsc{cat} & \textit{s} \\ \textsc{dtrs} & \left\langle \begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \boxed{1} \\ \textsc{cat} & \textit{np} \end{bmatrix}, \begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \boxed{2} \\ \textsc{cat} & \textit{vp} \end{bmatrix} \right\rangle \end{bmatrix}$$

b. $$\begin{bmatrix} \textit{phrase} & \\ \textsc{phon} & \boxed{1}\left\langle \textit{Leonard} \right\rangle \circ \boxed{2}\left\langle \textit{trinkt} \right\rangle \\ \textsc{cat} & \textit{s} \\ \textsc{dtrs} & \left\langle \begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \boxed{1} \\ \textsc{cat} & \textit{np} \end{bmatrix}, \begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \boxed{2} \\ \textsc{cat} & \textit{vp} \end{bmatrix} \right\rangle \end{bmatrix}$$

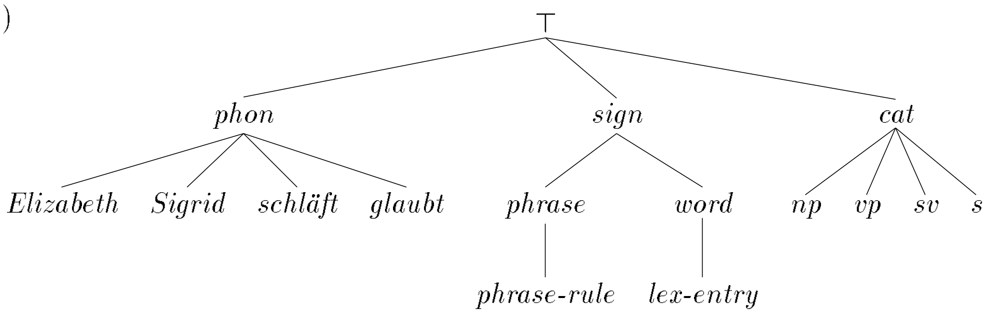
(73*a*) und (73*b*) sind die einzigen Merkmalstrukturen, des Typs *phrase*, die alle Constraints erfüllen. Die Menge dieser beiden Strukturen ist damit die Sprache, die durch die Grammatik aus der Typenhierarchie (64) und den Constraints (58), (59), (61) und (63) hinsichtlich des Types *phrase* beschrieben wird.

Constraintvererbung

Constraints zu einem Typ σ betreffen auch alle Subtypen von σ. Dieser Sachverhalt wird als *Constraintvererbung* bezeichnet.

Zur Veranschaulichung der Constraintvererbung erweitern wir das letzte Beispiel um einige Verben, Eigennamen und Phrasenstrukturregeln. Die folgende Typenhierarchie wird zugrunde gelegt:

(74)



Anstatt der graphischen Darstellung in (74) wird von nun an wahlweise die folgende textuelle Darstellung verwendet:

(75)
$$\begin{array}{lll} \textit{phon} & \Rightarrow & \textit{Elizabeth} \vee \textit{Sigrid} \vee \textit{schläft} \vee \textit{glaubt.} \\ \textit{cat} & \Rightarrow & \textit{np} \vee \textit{vp} \vee \textit{sv} \vee \textit{s.} \\ \textit{sign} & \Rightarrow & \textit{phrase} \vee \textit{word.} \\ \textit{phrase} & \Rightarrow & \textit{phrase-rule.} \\ \textit{word} & \Rightarrow & \textit{lex-entry.} \end{array}$$

Jede Zeile stellt einen Typ zusammen mit seinen direkten Untertypen dar. Typen, die nicht auf der rechten Seite einer anderen Typendefinition vorkommen, sind direkte Subtypen von $\top$. Erscheint ein Typ auf keiner linken Seite einer Typendefinition, handelt es sich um einen atomaren Typ.

Constraints wurden bisher immer in der Form

(76)
$$\sigma \quad \Rightarrow \quad \begin{bmatrix} \sigma \\ \ldots \end{bmatrix} \vee \begin{bmatrix} \sigma \\ \ldots \end{bmatrix} \vee \ \ldots \quad .$$

dargestellt. Da sich die linke Seite jedoch aus dem Typ der äußeren Strukturen auf der rechten Seite ergibt, können wir einfacher schreiben:

(77) $$\begin{bmatrix}\sigma \\ \ldots\end{bmatrix}\begin{bmatrix}\sigma \\ \ldots\end{bmatrix}\ldots$$

Mehrere Constraints des gleichen Typs sind als Alternativen zu verstehen.

Die Constraints für das neue Beispiel können nun folgendermaßen dargestellt werden:

(78) $$\begin{bmatrix}\textit{sign} & \\ \textsc{cat} & \textit{cat}\end{bmatrix}$$

$$\begin{bmatrix}\textit{phrase} & \\ \textsc{dtr1} & \textit{sign} \\ \textsc{dtr2} & \textit{sign}\end{bmatrix}$$

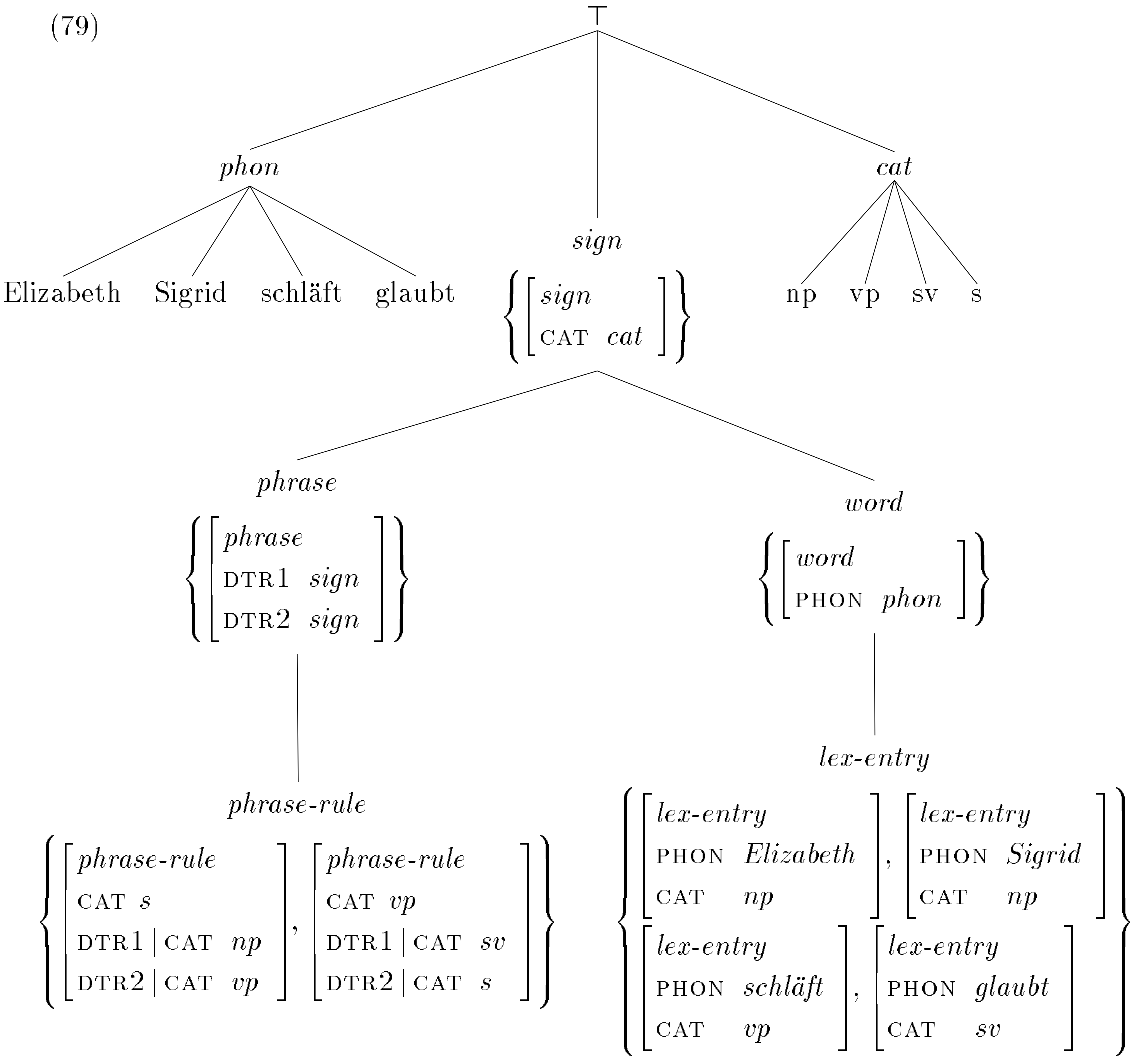
$$\begin{bmatrix}\textit{phrase-rule} & \\ \textsc{cat} & s \\ \textsc{dtr1}\,|\,\textsc{cat} & np \\ \textsc{dtr2}\,|\,\textsc{cat} & vp\end{bmatrix} \quad \begin{bmatrix}\textit{phrase-rule} & \\ \textsc{cat} & vp \\ \textsc{dtr1}\,|\,\textsc{cat} & sv \\ \textsc{dtr2}\,|\,\textsc{cat} & s\end{bmatrix}$$

$$\begin{bmatrix}\textit{word} & \\ \textsc{phon} & \textit{phon}\end{bmatrix}$$

$$\begin{bmatrix}\textit{lex-entry} & \\ \textsc{phon} & \textit{Elizabeth} \\ \textsc{cat} & np\end{bmatrix} \begin{bmatrix}\textit{lex-entry} & \\ \textsc{phon} & \textit{Sigrid} \\ \textsc{cat} & np\end{bmatrix} \begin{bmatrix}\textit{lex-entry} & \\ \textsc{phon} & \textit{schläft} \\ \textsc{cat} & vp\end{bmatrix} \begin{bmatrix}\textit{lex-entry} & \\ \textsc{phon} & \textit{glaubt} \\ \textsc{cat} & sv\end{bmatrix}$$

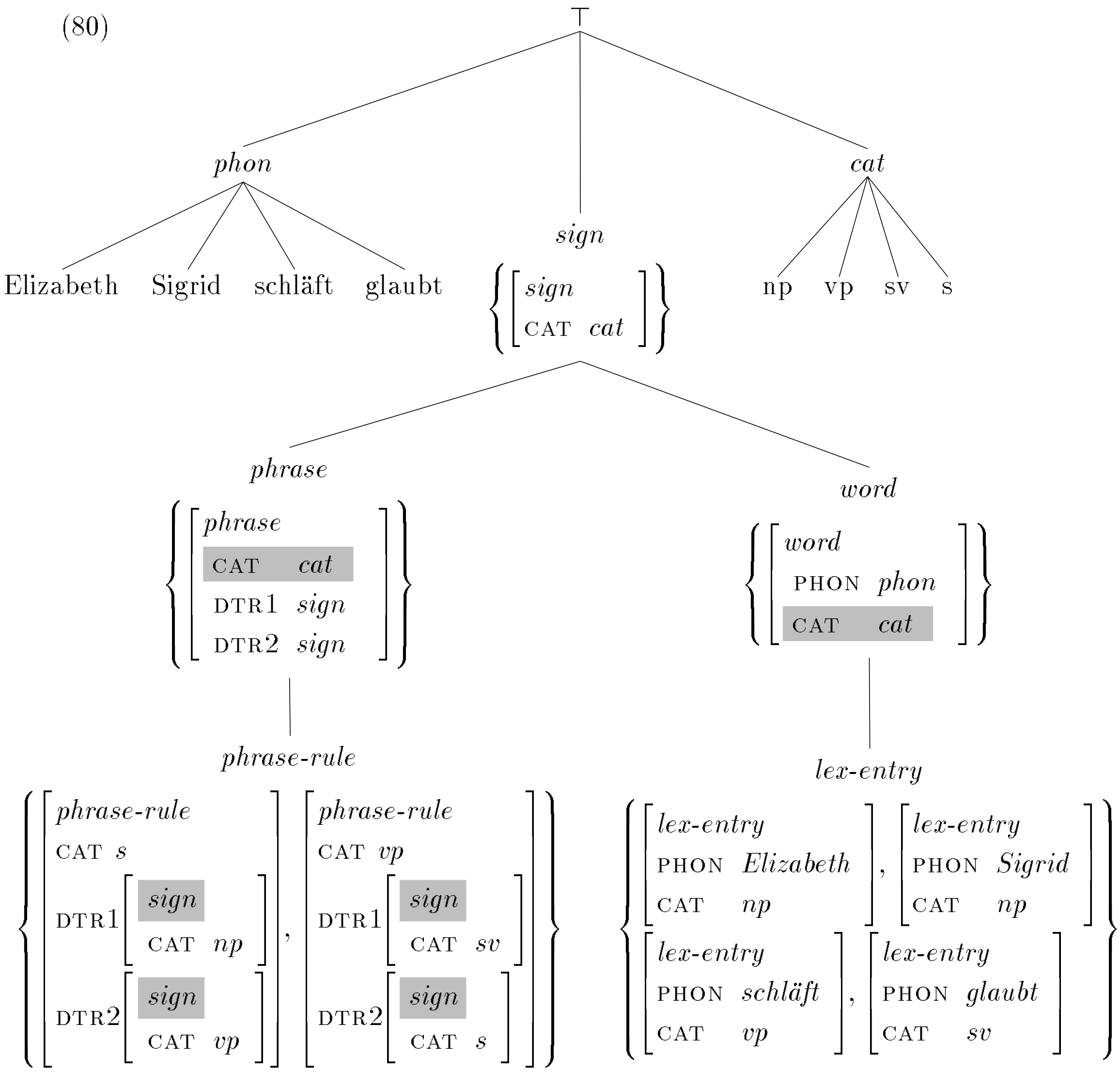
Zur Veranschaulichung der Constraintvererbung werden Typenhierarchie und Constraints in einer gemeinsamen Hierarchie dargestellt. Jedem Typ wird die Menge seiner Constraints zugeordnet. Die Elemente dieser Menge verhalten sich zueinander disjunktiv:

(79)



Die Constraint-Vererbung kann nun graphisch veranschaulicht werden. Die dabei entstehende Hierarchie unterscheidet sich in unserem Fall nur geringfügig von der ursprünglichen Darstellung. Bedingungen, die bei der Vererbung neu hinzukommenden, werden grau unterlegt dargestellt:

(80)



In Hierarchien mit disjunktiven Constraints für nicht-minimale Typen "multiplizieren" sich die Constraints, wie das folgende Beispiel einer "freien Wortstellung" veranschaulicht:

Die ursprüngliche Hierarchie:

(81)

$$
\begin{array}{c}
\textit{phrase} \\
\left\{
\begin{bmatrix} \textit{phrase} \\ \text{PHON } \boxed{1} \circ \boxed{2} \\ \text{DTRS } \left\langle \begin{bmatrix}\text{PHON } \boxed{1}\end{bmatrix}, \begin{bmatrix}\text{PHON } \boxed{2}\end{bmatrix} \right\rangle \end{bmatrix},
\begin{bmatrix} \textit{phrase} \\ \text{PHON } \boxed{1} \circ \boxed{2} \\ \text{DTRS } \left\langle \begin{bmatrix}\text{PHON } \boxed{2}\end{bmatrix}, \begin{bmatrix}\text{PHON } \boxed{1}\end{bmatrix} \right\rangle \end{bmatrix}
\right\} \\
| \\
\textit{phrase-rule} \\
\left\{
\begin{bmatrix} \textit{phrase-rule} \\ \text{CAT } \textit{s} \\ \text{DTRS } \left\langle \begin{bmatrix}\text{CAT } \textit{np}\end{bmatrix}, \begin{bmatrix}\text{CAT } \textit{vp}\end{bmatrix} \right\rangle \end{bmatrix},
\begin{bmatrix} \textit{phrase-rule} \\ \text{CAT } \textit{vp} \\ \text{DTRS } \left\langle \begin{bmatrix}\text{CAT } \textit{sv}\end{bmatrix}, \begin{bmatrix}\text{CAT } \textit{s}\end{bmatrix} \right\rangle \end{bmatrix}
\right\}
\end{array}
$$

Die gleiche Hierarchie mit vererbten Constraints:

(82)

$$
\begin{array}{c}
\textit{phrase} \\
\left\{
\begin{bmatrix} \textit{phrase} \\ \text{PHON } \boxed{1} \circ \boxed{2} \\ \text{DTRS } \left\langle \begin{bmatrix}\text{PHON } \boxed{1}\end{bmatrix}, \begin{bmatrix}\text{PHON } \boxed{2}\end{bmatrix} \right\rangle \end{bmatrix},
\begin{bmatrix} \textit{phrase} \\ \text{PHON } \boxed{1} \circ \boxed{2} \\ \text{DTRS } \left\langle \begin{bmatrix}\text{PHON } \boxed{2}\end{bmatrix}, \begin{bmatrix}\text{PHON } \boxed{1}\end{bmatrix} \right\rangle \end{bmatrix}
\right\} \\
| \\
\textit{phrase-rule} \\
\left\{
\begin{array}{c}
\begin{bmatrix} \textit{phrase-rule} \\ \text{PHON } \boxed{1} \circ \boxed{2} \\ \text{CAT } \textit{s} \\ \text{DTRS } \left\langle \begin{bmatrix}\text{PHON} & \boxed{1} \\ \text{CAT} & \textit{np}\end{bmatrix}, \begin{bmatrix}\text{PHON} & \boxed{2} \\ \text{CAT} & \textit{vp}\end{bmatrix} \right\rangle \end{bmatrix},
\begin{bmatrix} \textit{phrase-rule} \\ \text{PHON } \boxed{1} \circ \boxed{2} \\ \text{CAT } \textit{s} \\ \text{DTRS } \left\langle \begin{bmatrix}\text{PHON} & \boxed{2} \\ \text{CAT} & \textit{np}\end{bmatrix}, \begin{bmatrix}\text{PHON} & \boxed{1} \\ \text{CAT} & \textit{vp}\end{bmatrix} \right\rangle \end{bmatrix} \\
\begin{bmatrix} \textit{phrase-rule} \\ \text{PHON } \boxed{1} \circ \boxed{2} \\ \text{CAT } \textit{vp} \\ \text{DTRS } \left\langle \begin{bmatrix}\text{PHON} & \boxed{1} \\ \text{CAT} & \textit{sv}\end{bmatrix}, \begin{bmatrix}\text{PHON} & \boxed{2} \\ \text{CAT} & \textit{s}\end{bmatrix} \right\rangle \end{bmatrix},
\begin{bmatrix} \textit{phrase-rule} \\ \text{PHON } \boxed{1} \circ \boxed{2} \\ \text{CAT } \textit{vp} \\ \text{DTRS } \left\langle \begin{bmatrix}\text{PHON} & \boxed{2} \\ \text{CAT} & \textit{sv}\end{bmatrix}, \begin{bmatrix}\text{PHON} & \boxed{1} \\ \text{CAT} & \textit{s}\end{bmatrix} \right\rangle \end{bmatrix}
\end{array}
\right\}
\end{array}
$$

Formale Beschreibung von Constraints und Constraintvererbung

Formal kann das Verhältnis von Typen und zugehörigen Constraints durch eine Funktion Cons beschrieben werden:

Definition 1.10 (Constraint-Funktion)
Die Constraint-Funktion Cons *ist eine Funktion, die jedem Typ* $\sigma \in \mathsf{Type}$ *eine Menge von Constraints* $\mathsf{Cons}(\sigma) \subseteq \mathsf{MS}$ *zuordnet:*

$$\mathsf{Cons} : \mathsf{Type} \longrightarrow 2^{\mathsf{MS}}$$

Die Elemente aus $\mathsf{Cons}(\sigma)$ für ein $\sigma \in \mathsf{Type}$ sind als alternative Constraints zu interpretieren. Die Menge $\mathsf{Cons}(\sigma)$ ließe sich also auch als Disjunktion ihrer Elemente darstellen. Eine leere Constraintmenge wird immer erfüllt.

Um die Definition der Constraintvererbung zu vereinfachen, definieren wir die Funktion $\mathsf{Cons}^+ : \mathsf{Type} \longrightarrow 2^{\mathsf{MS}}$ folgendermaßen:

$$\forall_{\sigma \in \mathsf{Type}} : \mathsf{Cons}^+(\sigma) = \begin{cases} \{\top\} & \textit{falls} \quad \mathsf{Cons}(\sigma) = \emptyset, \\ \mathsf{Cons}(\sigma) & \textit{ansonsten} \end{cases}$$

Mit *Super* sei die Funktion $\mathit{Super} : \mathsf{Type} \longrightarrow 2^{\mathsf{Type}}$ bezeichnet, die einen Typ $\sigma \in \mathsf{Type}$ auf die Menge seiner unmittelbaren Supertypen abbildet:

$$\forall_{\sigma \in \mathsf{Type}} : \mathit{Super}(\sigma) = \{\tau \in \mathsf{Type} \mid \tau \sqsupset \sigma \wedge \neg\exists_{\upsilon \in \mathsf{Type}} : \tau \sqsupset \upsilon \wedge \upsilon \sqsupset \sigma\}$$

Definition 1.11 (Constraintvererbung[28])
$\mathsf{Cons}^* : \mathsf{Type} \longrightarrow 2^{\mathsf{MS}}$ *bildet jeden Typ auf die Menge seiner* ererbten Constraints *ab. Für jeden Typ* $\sigma \in \mathsf{Type}$ *mit* $\mathit{Super}(\sigma) = \{\tau_1, ..., \tau_k\}$ *ist* $\mathsf{Cons}^*(\sigma)$ *definiert durch:*

$$\mathsf{Cons}^*(\sigma) = \bigcup \{ \bigsqcup_{0 \leq i \leq k} \upsilon_i \mid \begin{array}{l} \upsilon_0 \in \mathsf{Cons}^+(\sigma), \\ \upsilon_i \in \mathsf{Cons}^*(\tau_i) \quad \textit{für} \quad 1 \leq i \leq k\} \end{array}$$

Für $\top$ folgt aus dieser Definition wegen $\mathit{Super}(\top) = \emptyset$:

$$\mathsf{Cons}^*(\top) = \bigcup \{ \bigsqcup_{0 \leq i \leq 0} \upsilon_i \mid \upsilon_0 \in \mathsf{Cons}^+(\top)\} = \mathsf{Cons}^+(\top)$$

28 Vgl. Carpenter 1992, Def. 15.2 (Inherited Constraint), S. 229.

Aufgrund der hier benutzten Definition von Typenhierarchien und den daraus folgenden Konsequenzen für die Unifikation als einer Funktion auf *Mengen* von Merkmalstrukturen, sieht die hier vorgestellte Definition der Constraintvererbung auf den ersten Blick schwieriger aus, als die von Carpenter vorgeschlagene. Wird sie jedoch anhand von Beispielen durchgespielt, zeigt sich, daß sie unkompliziert ist und direkt in einen einfachen Algorithmus umgesetzt werden kann.

Definition von Constraintsystemen

Die Mengen Type und Feat, eine Typenhierarchie $\mathcal{H}$ über Type und eine Constraint-Funktion Cons ergeben zusammen ein Constrainsystem. Da Grammatiken in dieser Arbeit als Constraintsysteme formalisiert werden, besitzen wir damit alle Bausteine unseres formalen Modells.

Definition 1.12 (Constraintsystem[29])
Ein Constraintsystem *ist ein 4-Tupel* $\langle \mathsf{Type}, \mathsf{Feat}, \mathcal{H}, \mathsf{Cons} \rangle$*, mit:*

1. Type *ist eine endliche Menge von Typen;*
2. Feat *ist eine endliche Menge von Merkmalen;*
3. $\mathcal{H} = \langle \mathsf{Type}, \sqsupseteq, \top \rangle$ *ist eine Typenhierarchie über* Type*;*
4. Cons *ist eine Constraint-Funktion.*

Lösung eines Constraintsystems

Am Anfang dieses Abschnitts (S. 37) wurde anhand eines einfachen Beispiels gezeigt, wie Typenhierarchie und Constraints zusammenwirken, um eine Menge aufgelöster Merkmalstrukturen zu definieren: Von der Struktur [*phrase*] ausgehend wurden die Strukturen (73*a*) und (73*b*) abgeleitet.

Neben der Forderung, daß alle Constraints erfüllt werden müssen, gibt es eine weitere Anforderung an die Lösung eines Constraintsystems: Alle Typen einer Lösung müssen minimal sein. Diese Forderung läßt sich in der Syntax von Typendeklarationen wiederfinden:

(83) $\sigma \;\Rightarrow\; \tau_1 \vee \tau_2 \vee \ldots \vee \tau_n.$

läßt sich als Constraint des Types σ lesen:

> "Wenn ein Zeichen vom Typ σ ist, muß es entweder vom Typ τ_1, vom Typ τ_2, ... oder vom Typ τ_n sein."

Definition 1.13 (Lösung eines Constraintsystems[30])
Eine Merkmalstruktur $F \in \mathsf{MS}^{\mathsf{Type},\mathsf{Feat}}$ *ist eine* Lösung *eines Constraintsystems* $C = \langle \mathsf{Type}, \mathsf{Feat}, \mathcal{H}, \mathsf{Cons} \rangle$*, wenn:*

1. *Alle Typen in* F *minimal sind;*

29 vgl. Carpenter 1992, Def. 15.1 (Constraint System), S. 228.

2. $F@\pi \sqsubseteq \sigma$ *für ein* $\sigma \in \mathsf{Cons}^*(\theta(\delta(\pi, \bar{q})))$ *für alle Pfade, für die* $F@\pi$ *definiert ist;*

3. es gibt keine Lösung F'*, die 1 und 2 erfüllt, mit* $F' \sqsupset F$.

Die Menge aller Lösungen von C bezeichnen wir mit $\mathsf{L}(C)$.

Die letzte Bedingung stellt sicher, daß eine Lösung keine Information enthält, die nicht durch die ersten beiden Bedingungen erzwungen wurde.

1.5 Resolution

Wir wissen nun, was Constraintsysteme sind und wie durch ein solches System eine Menge von Merkmalstrukturen definiert wird. Um die mathematische Spezifikation von Constraintsystemen und deren Lösungsmenge für die Praxis nutzbar zu machen, muß ein syntaktisches Verfahren gefunden werden, das zu einer Anfrage in Form einer unterspezifizierten Merkmalstruktur die Menge der Lösungen aufzählt. Ein solches Verfahren wird *Resolutionsverfahren* genannt.

Wie ein Resolutionsverfahren aussehen kann, läßt sich aus der Definition der Lösungsmenge leicht ersehen:

Von einer beliebigen unterspezifizierten Merkmalstruktur F ausgehend, wird rekursiv, einer fairen Strategie (beispielsweise Breitensuche) folgend, einer der folgenden zwei Schritte angewandt:

1. ein allgemeinerer Typ wird durch einen spezielleren ersetzt;
2. eine Teilstruktur von F des Typs σ wird mit einem Element von $\mathsf{Cons}^*(\sigma)$ unifiziert;

Diese beiden Schritte werden solange wiederholt, bis sich eine Lösung ergibt.[31]

Formal läßt sich dieses Resolutionsverfahren folgendermaßen definieren:

Definition 1.14 (π-Resolution[32])
Als π*-Resolution bezeichnen wir die kleinste Relation* $\stackrel{\pi}{\Longrightarrow} \subseteq \mathsf{MS} \times \mathsf{MS}$*, die die folgende Bedingung erfüllt:*

30 vgl. Carpenter 1992, Def. 15.3 (Resolved Feature Structure), S. 229.

31 Enthält die Ausgangsstruktur schon "überflüssige" Information, oder werden auf eine Substruktur vom Typ σ mehrere Elemente von $\mathsf{Cons}^*(\sigma)$ nacheinander angewandt, kann eine Lösung mehr Information als nötig enthalten. Auch in diesem Fall wird sie jedoch durch eine Lösung subsumiert.

$$F \xRightarrow{\pi} F' \qquad \textit{falls} \qquad \begin{array}{ll} \textit{1.} & \pi \in \mathsf{Path} \ \textit{und} \ \delta(\pi, \bar{q}) \ \textit{in} \ F \ \textit{definiert}; \\ \textit{2.} & \sigma = \theta(\delta(\pi, \bar{q})); \\ \textit{3.} & \phi \in \mathsf{Cons}^*(\sigma) \quad \textit{oder} \quad \phi \sqsubset \sigma; \\ \textit{4.} & F' \in F \sqcup [\pi \ \phi]. \end{array}$$

Sei π also ein beliebiger Pfad in F und σ der Typ der Unterstruktur, die sich in F am Ende des Pfades π befindet, so entsteht $F \sqcup [\pi \ \phi]$ entweder, indem σ durch einen direkten Untertyp ersetzt wird, oder indem $F@\pi$ durch ein Element der Unifikation von $F@\pi$ mit einem Element von $\mathsf{Cons}^*(\sigma)$ ersetzt wird.

Als *Resolution* bezeichnen wir die wiederholte Anwendung der π-*Resolution* bis eine vollspezifizierte Merkmalstruktur (Lösung) entsteht:

Definition 1.15 (Resolution[33])
Als Resolution *bezeichnen wir die wiederholte Anwendung der π-Resolution auf eine Merkmalstruktur (Anfrage) nach einer fairen Strategie (z.B. Breitensuche), bis alle Typen der entstandenen Struktur minimal sind und mit einem Element ihres geerbten Constraints unifiziert wurden.*

1.6 Beispiele zur Anwendung von Constraintsystemen

In den letzten Abschnitten wurden die wesentlichen Punkte des Systems besprochen, das in dieser Arbeit zur Modellierung der japanischen Verbmodifikation verwendet wird. Das bisher Gesagte soll nun anhand einiger Beispiele, in denen die später verwendeten Techniken vorgestellt werden, veranschaulicht werden:

Berechnungen von Funktionen — Die Realisierung von Berechnungen durch Constraintsysteme wird anhand der *Listenkonkatenation* dargestellt.

Semantik — Die Modellierung einer einfachen *Semantik* für natürliche Sprachen und ihr Zusammenspiel mit der Syntax wird an einem weiteren kleinen Beispiel vorgestellt.

1.6.1 Listen und Listen-Konkatenation

Constraintsysteme lassen sich zur Berechnung von Funktionen benutzen. Berechnungsvorschriften ("Programme") für Constraintsysteme haben große

32 vgl. Carpenter 1992, Def. 15.5 (π-Resolution), S. 231.
33 vgl. Carpenter 1992, Def. 15.8 (Resolution), S. 232.

Ähnlichkeit mit logischen Programmen, beispielsweise in *Pure Prolog.*[34] Um eine Vorstellung zu vermitteln, wie Berechnungsvorschriften für Constraintsysteme ausgedrückt werden können, wird in diesem Abschnitt die Konkatenation von Listen beschrieben.

Listen als rekursive Merkmalstrukturen

Listen sind rekursive Merkmalstrukturen[35]. Sie genügen der folgenden Spezifikation:

(84) **Typenhierarchie:**

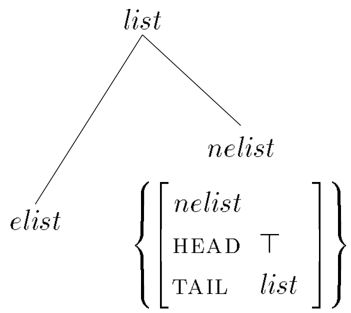
$\textit{list} \;\Rightarrow\; \textit{elist} \vee \textit{nelist}.$

Constraints:

$$\begin{bmatrix} \textit{nelist} & \\ \textsc{head} & \top \\ \textsc{tail} & \textit{list} \end{bmatrix}$$

Anschaulicher ist die graphische Darstellung:

(85)



Der leichteren Lesbarkeit halber, wird diese explizite Form durch spitze Klammern abgekürzt:

34 vgl. Sterling und Shapiro 1986 und Lloyd 1984.

35 vgl. Abschnitt "Auflösung der Abkürzungen", S. 18.

(86) Abkürzung: abgekürzte Merkmalstruktur:

$\langle\rangle$	*elist*
$\left\langle a,\ b,\ c \right\rangle$	$\begin{bmatrix} \textit{nelist} & \\ \text{HEAD} & a \\ \text{TAIL} & \begin{bmatrix} \textit{nelist} & \\ \text{HEAD} & b \\ \text{TAIL} & \begin{bmatrix} \textit{nelist} & \\ \text{HEAD} & c \\ \text{TAIL} & \textit{elist} \end{bmatrix} \end{bmatrix} \end{bmatrix}$
$\left\langle a,\ b \bullet \textit{list} \right\rangle$	$\begin{bmatrix} \textit{nelist} & \\ \text{HEAD} & a \\ \text{TAIL} & \begin{bmatrix} \textit{nelist} & \\ \text{HEAD} & b \\ \text{TAIL} & \textit{list} \end{bmatrix} \end{bmatrix}$

Rekursive Berechnung der Listen-Konkatenation

Rekursion läßt sich in Constraintsystemen durch rekursive Constraints darstellen. Bei der Konkatenation von Listen $\circ : \textit{Listen} \times \textit{Listen} \longrightarrow \textit{Listen}$ lassen sich zwei Fälle unterscheiden:

1. das erste Argument ist die *leere Liste* (*Rekursionsverankerung*):
   Die Konkatenation einer leeren Liste $\langle\rangle$ mit einer anderen Liste l ist l:

   $$\langle\rangle \circ l = l$$

2. das erste Argument ist eine *nicht-leere Liste* (*Rekursionsschritt*):
   Die Konkatenation einer nicht-leeren Liste $\langle e \bullet r\rangle$ aus einem ersten Element e und einer Restliste r mit einer Liste l entspricht einer Liste mit e als erstem Element und der Konkatenation von r mit l als Restliste:

   $$\langle e \bullet r\rangle \circ l = \langle e \bullet (r \circ l)\rangle$$

Mit Hilfe der Merkmale LEFT, RIGHT und RESULT lassen sich diese beiden Fälle folgendermaßen beschreiben:

(87) a. *Rekursionsverankerung*:

$$\begin{bmatrix} \textit{append} & \\ \text{LEFT} & \langle\rangle \\ \text{RIGHT} & \boxed{1} \\ \text{RESULT} & \boxed{1} \end{bmatrix}$$

b. *Rekursionsschritt*:

$$\begin{bmatrix} \textit{append} & \\ \text{LEFT} & \left\langle \boxed{1} \bullet \boxed{2} \right\rangle \\ \text{RIGHT} & \boxed{3} \\ \text{RESULT} & \left\langle \boxed{1} \bullet \boxed{4} \right\rangle \end{bmatrix} \quad \textit{gdw.} \quad \begin{bmatrix} \textit{append} & \\ \text{LEFT} & \boxed{2} \\ \text{RIGHT} & \boxed{3} \\ \text{RESULT} & \boxed{4} \end{bmatrix}$$

Auf eine Struktur

(88) $$\begin{bmatrix} \textit{append} & \\ \text{LEFT} & \textit{Liste 1} \\ \text{RIGHT} & \textit{Liste 2} \\ \text{RESULT} & ? \end{bmatrix}$$

muß also im Falle einer leeren ersten Liste der Constraint 1 und im Falle einer nicht-leeren ersten Liste der Constraint 2 angewandt werden. Werden die beiden Bedingungen (87*a*) und (87*b*) als alternative Constraints zum Typ *append* aufgefaßt, wird genau dieses erreicht: [LEFT ⟨ ⟩] läßt sich nur mit einer Struktur mit leerer Liste als Wert von LEFT unifizieren, und [LEFT $\langle \boxed{1} \bullet \boxed{2} \rangle$] kann nur mit einer Struktur mit nicht-leerer Liste als erstem Argument unifiziert werden.

Die Bedingung im Rekursionsschritt, hier durch "*gdw.*" ("*genau dann wenn*") ausgedrückt, läßt sich mit einem zusätzlichen Merkmal CONDITION mit Werten, die wiederum vom Typ *append* sind, formulieren. Insgesamt erhalten wir damit:

(89) **Constraints für den Typ *append*:**

$$\begin{bmatrix} \textit{append} & \\ \text{LEFT} & \langle\,\rangle \\ \text{RIGHT} & \boxed{1} \\ \text{RESULT} & \boxed{1} \end{bmatrix} \quad \begin{bmatrix} \textit{append} & \\ \text{LEFT} & \left\langle \boxed{1} \bullet \boxed{2} \right\rangle \\ \text{RIGHT} & \boxed{3} \\ \text{RESULT} & \left\langle \boxed{1} \bullet \boxed{4} \right\rangle \\ \text{CONDITION} & \begin{bmatrix} \textit{append} & \\ \text{LEFT} & \boxed{2} \\ \text{RIGHT} & \boxed{3} \\ \text{RESULT} & \boxed{4} \end{bmatrix} \end{bmatrix}$$

Das Merkmal CONDITION erzwingt eine rekursive Berechnung der Konkatenation: Eine Struktur des Typs *append* mit nicht leerer Liste als Wert des Merkmals LEFT muß eines der beiden Constraints erfüllen. Da es mit dem ersten Constraint wegen des nicht leeren ersten Elementes im Widerspruch steht, muß es dem zweiten Constraint genügen. Durch die Unifikation der Variablen $\boxed{2}$ mit der Restliste von LEFT erscheint diese als erstes Argument der Struktur des Merkmals CONDITION. Das zweite Argument dieser Struktur ist mit dem zweiten Argument der äußeren Struktur identisch. Da die CONDITION-Struktur wiederum com Typ *append* ist, wiederholt sich der Berechnungsschritt mit der neuen, um ein Element kürzeren Argumentliste rekursiv, bis die Rekursionsverankerung angewandt werden kann. Über die koindizierten RESULT-Merkmale ergibt sich das Ergebnis der Listen-Konkatenation.

Im Falle der Ausgangsstruktur

(90)
$$\begin{bmatrix} \textit{append} & \\ \text{LEFT} & \left\langle a,\ b \right\rangle \\ \text{RIGHT} & \left\langle c,\ d \right\rangle \\ \text{RESULT} & ? \end{bmatrix}$$

würde sich beispielsweise die folgende Ergebnisstruktur ergeben (der Übersichtlichkeit halber werden die Werte der Variablen wiederholt):

(91)
$$\begin{bmatrix} \textit{append} & \\ \text{LEFT} & \left\langle \boxed{1}\ a \bullet \boxed{2} \left\langle \boxed{3}\ b \bullet \boxed{4} \langle\,\rangle \right\rangle \right\rangle \\ \text{RIGHT} & \boxed{5} \left\langle c,d \right\rangle \\ \text{RESULT} & \left\langle \boxed{1}\ a \bullet \boxed{6} \left\langle \boxed{3}\ b \bullet \boxed{5} \left\langle c,d \right\rangle \right\rangle \right\rangle \\ \text{CONDITION} & \begin{bmatrix} \textit{append} & \\ \text{LEFT} & \boxed{2} \left\langle \boxed{3}\ b \bullet \boxed{4} \langle\,\rangle \right\rangle \\ \text{RIGHT} & \boxed{5} \left\langle c,d \right\rangle \\ \text{RESULT} & \boxed{6} \left\langle \boxed{3}\ b \bullet \boxed{5} \left\langle c,d \right\rangle \right\rangle \\ \text{CONDITION} & \begin{bmatrix} \textit{append} & \\ \text{LEFT} & \boxed{4} \langle\,\rangle \\ \text{RIGHT} & \boxed{5} \left\langle c,d \right\rangle \\ \text{RESULT} & \boxed{5} \left\langle c,d \right\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Das Merkmal CONDITION wird nur zur Berechnung der funktionalen Abhängigkeit von Merkmalen benötigt. Die Verwendung einer Funktion als Abkürzung ist deshalb übersichtlicher:

(92) Abkürzung: abgekürzte Merkmalstruktur:

$$\begin{bmatrix} \text{LIST1} & \boxed{1} \\ \text{LIST2} & \boxed{2} \\ \text{LIST} & \boxed{1} \circ \boxed{2} \end{bmatrix} \qquad \begin{bmatrix} \text{LIST1} & \boxed{1} \\ \text{LIST2} & \boxed{2} \\ \text{LIST} & \boxed{3} \\ \text{CONDITION} & \begin{bmatrix} \textit{append} & \\ \text{LEFT} & \boxed{1} \\ \text{RIGHT} & \boxed{2} \\ \text{RESULT} & \boxed{3} \end{bmatrix} \end{bmatrix}$$

1.6.2 Syntax und Semantik

Der letzte Abschnitt dieses Kapitels ist dem Zusammenspiel von Syntax und Semantik gewidmet. Es soll beschrieben werden, wie diese beiden Komponenten der Grammatik mit ihren wechselseitigen Abhängigkeiten in einem Constraintsystem dargestellt werden können.

Aufbau der Semantik

Nicht die adäquate Modellierung semantischer Phänomene soll uns hier beschäftigen, sondern der Mechanismus der *Komposition* semantischer Beschreibungen entlang der syntaktischen Struktur. Aus diesem Grund wird eine schematisierte, kompositionale Semantik verwendet, die — soweit möglich — von allen semantischen Details abstrahiert, die zur Veranschaulichung des kompositionalen Prinzips nicht erforderlich sind.

Eigennamen werden als Atome dargestellt:

(93) *Semantik von Eigennamen:*
elizabeth, *sigrid*, *leonard*, *sebastian*

Adjektive und Verben durch Prädikate:

(94) *Semantik von Verben:*

$$\begin{bmatrix} \textit{sleep} & \\ 1 & \textit{cont} \end{bmatrix} \begin{bmatrix} \textit{love} & \\ 1 & \textit{cont} \\ 2 & \textit{cont} \end{bmatrix} \begin{bmatrix} \textit{believe} & \\ 1 & \textit{cont} \\ 2 & \textit{cont} \end{bmatrix}$$

Die Semantik von Phrasen und Sätzen setzt sich kompositional aus der Semantik ihrer Bestandteile zusammen.

Der Eigenname *Leonard* und das Verb *schlafen* haben beispielsweise die folgende Semantik:

(95) *a.* *Semantik von "Leonard":*
leonard

b. *Semantik von "schlafen":*

$$\begin{bmatrix} \textit{sleep} & \\ ① & \textit{cont} \end{bmatrix}$$

Durch Komposition ergibt sich daraus die Semantik des Satzes *Leonard schläft*:

(96) *Semantik von "Leonard schläft":*

$$\begin{bmatrix} \textit{sleep} & \\ ① & \textit{leonard} \end{bmatrix}$$

Die Semantik transitiver Verben ergibt sich analog:

(97) *Semantik von "Leonard liebt Sigrid":*

$$\begin{bmatrix} \textit{love} & \\ ① & \textit{leonard} \\ ② & \textit{sigrid} \end{bmatrix}$$

Das Verb *glauben* nimmt einen Satz als Argument. Die Semantik dieses Satzes erscheint als Wert des zweiten Argumentes in der semantischen Struktur des Matrixverbes:

(98) *Semantik von "Sigrid glaubt, Elizabeth liebt Leonard":*

$$\begin{bmatrix} \textit{believe} & \\ ① & \textit{sigrid} \\ ② & \begin{bmatrix} \textit{love} & \\ ① & \textit{elizabeth} \\ ② & \textit{leonard} \end{bmatrix} \end{bmatrix}$$

Da die Semantik in Form von Merkmalstrukturen dargestellt werden kann, läßt sie sich als Wert des neu eingeführten Merkmals CONT in die lexikalischen Einträge und die Phrasenstrukturregeln integrieren.

Den lexikalischen Einträgen wird der Type zugeordnet, der ihre Semantik kodiert:

(99) *a.*

$$\begin{bmatrix} \textit{lex-entry} & \\ \textsc{phon} & \left\langle \textit{Leonard} \right\rangle \\ \textsc{cat} & \textit{np} \\ \textsc{cont} & \textit{leonard} \end{bmatrix}$$

b. $\begin{bmatrix} \textit{lex-entry} & \\ \text{PHON} & \left\langle \textit{schläft} \right\rangle \\ \text{CAT} & \textit{vp} \\ \text{CONT} & \textit{sleep} \end{bmatrix}$

In den Phrasenstrukturregeln wird angegeben, wie sich die Semantik der Tochterzeichen zu der Semantik des ganzen Zeichens zusammensetzen läßt:

Im Falle der Regel

(100) $$\text{S} \longrightarrow \text{NP VP}$$

für den Satz *Leonard schläft* beispielsweise wird die Semantik des Verbes *schläft* auf die Semantik der Nominalphrase *Leonard* angewandt, und schließlich als Semantik des ganzen Zeichens übernommen:

(101) $$\begin{bmatrix} \textit{phrase-rule} & \\ \text{CAT} & \textit{s} \\ \text{CONT} & \boxed{1} \\ \text{DTRS} & \left\langle \begin{bmatrix} \text{CAT} & \textit{np} \\ \text{CONT} & \boxed{2} \end{bmatrix}, \begin{bmatrix} \text{CAT} & \textit{vp} \\ \text{CONT} & \boxed{1} \begin{bmatrix} 1 & \boxed{2} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

Eine einfache Grammatik, die beispielsweise die Sätze

(102) a. *Elizabeth schläft*

$\begin{bmatrix} \textit{sleep} & \\ 1 & \textit{elizabeth} \end{bmatrix}$

b. *Sebastian liebt Sigrid*

$\begin{bmatrix} \textit{love} & \\ 1 & \textit{sebastian} \\ 2 & \textit{sigrid} \end{bmatrix}$

c. *Elizabeth glaubt, Sebastian liebt Sigrid*

$\begin{bmatrix} \textit{believe} & \\ 1 & \textit{elizabeth} \\ 2 & \begin{bmatrix} \textit{love} & \\ 1 & \textit{sebastian} \\ 2 & \textit{sigrid} \end{bmatrix} \end{bmatrix}$

beschreibt, läßt sich damit folgendermaßen als Constraintsystem modellieren:

(103) **Typenhierarchie:**

phon $\Rightarrow$ *Elizabeth* $\vee$ *Sigrid* $\vee$ *Leonard* $\vee$ *Sebastian* $\vee$ *liebt* $\vee$ *schläft* $\vee$ *glaubt.*
cont $\Rightarrow$ *elizabeth* $\vee$ *sigrid* $\vee$ *leonard* $\vee$ *sebastian* $\vee$ *love* $\vee$ *sleep* $\vee$ *believe.*
cat $\Rightarrow$ *np* $\vee$ *vp* $\vee$ *tv* $\vee$ *sv* $\vee$ *s.*
sign $\Rightarrow$ *phrase* $\vee$ *word.*
phrase $\Rightarrow$ *phrase-rule.*
word $\Rightarrow$ *lex-entry.*

Constraints:

$$\begin{bmatrix} \textit{sleep} \\ ① \ \textit{cont} \end{bmatrix} \quad \begin{bmatrix} \textit{love} \\ ① \ \textit{cont} \\ ② \ \textit{cont} \end{bmatrix} \quad \begin{bmatrix} \textit{believe} \\ ① \ \textit{cont} \\ ② \ \textit{cont} \end{bmatrix}$$

$$\begin{bmatrix} \textit{sign} \\ \text{PHON } \textit{list} \\ \text{CAT } \textit{cat} \\ \text{CONT } \textit{cont} \end{bmatrix}$$

$$\begin{bmatrix} \textit{phrase} \\ \text{PHON } \boxed{1} \circ \boxed{2} \\ \text{DTRS } \left\langle \begin{bmatrix} \textit{sign} \\ \text{PHON } \boxed{1} \end{bmatrix}, \begin{bmatrix} \textit{sign} \\ \text{PHON } \boxed{2} \end{bmatrix} \right\rangle \end{bmatrix}$$

$$\begin{bmatrix} \textit{phrase-rule} \\ \text{CAT } s \\ \text{CONT } \boxed{1} \\ \text{DTRS } \left\langle \begin{bmatrix} \text{CAT } np \\ \text{CONT } \boxed{2} \end{bmatrix}, \begin{bmatrix} \text{CAT } vp \\ \text{CONT } \boxed{1} \left[① \ \boxed{2} \right] \end{bmatrix} \right\rangle \end{bmatrix} \quad \begin{bmatrix} \textit{phrase-rule} \\ \text{CAT } vp \\ \text{CONT } \boxed{1} \\ \text{DTRS } \left\langle \begin{bmatrix} \text{CAT } tv \\ \text{CONT } \boxed{1} \left[② \ \boxed{2} \right] \end{bmatrix}, \begin{bmatrix} \text{CAT } np \\ \text{CONT } \boxed{2} \end{bmatrix} \right\rangle \end{bmatrix}$$

$$\begin{bmatrix} \textit{phrase-rule} \\ \text{CAT } vp \\ \text{CONT } \boxed{1} \\ \text{DTRS } \left\langle \begin{bmatrix} \text{CAT } sv \\ \text{CONT } \boxed{1} \left[② \ \boxed{2} \right] \end{bmatrix}, \begin{bmatrix} \text{CAT } s \\ \text{CONT } \boxed{2} \end{bmatrix} \right\rangle \end{bmatrix}$$

$$\begin{bmatrix} \textit{lex-entry} \\ \text{PHON } \langle \textit{Elizabeth} \rangle \\ \text{CAT } np \\ \text{CONT } \textit{elizabeth} \end{bmatrix} \quad \begin{bmatrix} \textit{lex-entry} \\ \text{PHON } \langle \textit{Sigrid} \rangle \\ \text{CAT } np \\ \text{CONT } \textit{sigrid} \end{bmatrix} \quad \begin{bmatrix} \textit{lex-entry} \\ \text{PHON } \langle \textit{Leonard} \rangle \\ \text{CAT } np \\ \text{CONT } \textit{leonard} \end{bmatrix} \quad \begin{bmatrix} \textit{lex-entry} \\ \text{PHON } \langle \textit{Sebastian} \rangle \\ \text{CAT } np \\ \text{CONT } \textit{sebastian} \end{bmatrix}$$

$$\begin{bmatrix} \textit{lex-entry} \\ \text{PHON } \langle \textit{liebt} \rangle \\ \text{CAT } tv \\ \text{CONT } \textit{love} \end{bmatrix} \quad \begin{bmatrix} \textit{lex-entry} \\ \text{PHON } \langle \textit{schläft} \rangle \\ \text{CAT } vp \\ \text{CONT } \textit{sleep} \end{bmatrix} \quad \begin{bmatrix} \textit{lex-entry} \\ \text{PHON } \langle \textit{glaubt} \rangle \\ \text{CAT } sv \\ \text{CONT } \textit{believe} \end{bmatrix}$$

Gegeben eine Grammatik kann das Resolutionsverfahren für die folgenden Berechnungen benutzt werden:

Untersuchung der Korrektheit — Für ein gegebenes Zeichen wird ermittelt, ob es im Sinne der Grammatik korrekt ist;

Parsen — Zu einem Satz kann die Menge möglicher linguistischer Strukturen — und damit auch möglicher semantischer Interpretationen — berechnet werden;

Generierung — Zu einer gegebenen semantischen Struktur kann die Menge der möglichen, sprachlichen Realisierungen ermittelt werden;

Vervollständigung — Zu einem unterspezifizierten Zeichen können alle korrekten Zeichen aufgezählt werden, die durch dieses subsumiert werden;

Aufzählung — Alle Zeichen, die durch die Grammatik beschrieben werden, können aufgezählt werden.

Für die drei Sätze (102*a*), (102*b*) und (102*c*) erhalten wir mit dieser Grammatik die folgenden Strukturen:

(104) *a.* *Elizabeth schläft*

$$\begin{bmatrix} \multicolumn{2}{l}{\textit{phrase-rule}} \\ \textsc{phon} & \left\langle \boxed{1}\ \textit{Elizabeth} \bullet \boxed{2} \left\langle \textit{schläft} \right\rangle \right\rangle \\ \textsc{cat} & s \\ \textsc{cont} & \boxed{3} \begin{bmatrix} \textit{sleep} \\ 1\ \boxed{4}\ \textit{elizabeth} \end{bmatrix} \\ \textsc{dtrs} & \left\langle \begin{bmatrix} \multicolumn{2}{l}{\textit{lex-entry}} \\ \textsc{phon} & \left\langle \boxed{1} \right\rangle \\ \textsc{cat} & np \\ \textsc{cont} & \boxed{4} \end{bmatrix}, \begin{bmatrix} \multicolumn{2}{l}{\textit{lex-entry}} \\ \textsc{phon} & \boxed{2} \\ \textsc{cat} & vp \\ \textsc{cont} & \boxed{3} \end{bmatrix} \right\rangle \end{bmatrix}$$

b. *Sebastian liebt Sigrid*

$$\left[\begin{array}{l}
\textit{phrase-rule}\\
\text{PHON}\ \left\langle \boxed{1}\ \textit{Sebastian} \bullet \boxed{2} \left\langle \boxed{3}\ \textit{liebt} \bullet \boxed{4} \left\langle \textit{Sigrid}\right\rangle\right\rangle\right\rangle\\
\text{CAT}\ \ s\\
\text{CONT}\ \boxed{5}\left[\begin{array}{l}\textit{love}\\ \textcircled{1}\ \boxed{6}\ \textit{sebastian}\\ \textcircled{2}\ \boxed{7}\ \textit{sigrid}\end{array}\right]\\
\text{DTRS}\ \left\langle \left[\begin{array}{l}\textit{lex-entry}\\ \text{PHON}\ \langle\boxed{1}\rangle\\ \text{CAT}\ \ np\\ \text{CONT}\ \boxed{6}\end{array}\right], \left[\begin{array}{l}\textit{phrase-rule}\\ \text{PHON}\ \boxed{2}\\ \text{CAT}\ \ vp\\ \text{CONT}\ \boxed{5}\\ \text{DTRS}\ \left\langle \left[\begin{array}{l}\textit{lex-entry}\\ \text{PHON}\ \langle\boxed{3}\rangle\\ \text{CAT}\ \ tv\\ \text{CONT}\ \boxed{5}\end{array}\right], \left[\begin{array}{l}\textit{lex-entry}\\ \text{PHON}\ \boxed{4}\\ \text{CAT}\ \ np\\ \text{CONT}\ \boxed{7}\end{array}\right]\right\rangle\end{array}\right]\right\rangle
\end{array}\right]$$

c. *Elizabeth glaubt, Sebastian liebt Sigrid*

$$\left[\begin{array}{l}
\textit{phrase-rule}\\
\text{PHON}\left\langle \boxed{1}\ \textit{Elizabeth} \bullet \boxed{2}\left\langle \boxed{3}\ \textit{glaubt} \bullet \boxed{4}\left\langle \boxed{5}\ \textit{Sebastian} \bullet \boxed{6}\left\langle \boxed{7}\ \textit{liebt} \bullet \boxed{8}\left\langle \textit{Sigrid}\right\rangle\right\rangle\right\rangle\right\rangle\right\rangle\\
\text{CAT}\ \ s\\
\text{CONT}\ \boxed{9}\left[\begin{array}{l}\textit{believe}\\ \textcircled{1}\ \boxed{10}\ \textit{elizabeth}\\ \textcircled{2}\ \boxed{11}\left[\begin{array}{l}\textit{love}\\ \textcircled{1}\ \boxed{12}\ \textit{sebastian}\\ \textcircled{2}\ \boxed{13}\ \textit{sigrid}\end{array}\right]\end{array}\right]\\
\text{DTRS}\left\langle \left[\begin{array}{l}\textit{lex-entry}\\ \text{PHON}\langle\boxed{1}\rangle\\ \text{CAT}\ \ np\\ \text{CONT}\ \boxed{10}\end{array}\right], \left[\begin{array}{l}\textit{phrase-rule}\\ \text{PHON}\ \boxed{2}\\ \text{CAT}\ \ vp\\ \text{CONT}\ \boxed{9}\\ \text{DTRS}\left\langle \left[\begin{array}{l}\textit{lex-entry}\\ \text{PHON}\langle\boxed{3}\rangle\\ \text{CAT}\ \ sv\\ \text{CONT}\ \boxed{9}\end{array}\right], \left[\begin{array}{l}\textit{phrase-rule}\\ \text{PHON}\ \boxed{4}\\ \text{CAT}\ \ s\\ \text{CONT}\ \boxed{11}\\ \text{DTRS}\left\langle \left[\begin{array}{l}\textit{lex-entry}\\ \text{PHON}\langle\boxed{5}\rangle\\ \text{CAT}\ \ np\\ \text{CONT}\ \boxed{12}\end{array}\right], \left[\begin{array}{l}\textit{phrase-rule}\\ \text{PHON}\ \boxed{6}\\ \text{CAT}\ \ vp\\ \text{CONT}\ \boxed{11}\\ \text{DTRS}\left\langle \left[\begin{array}{l}\textit{lex-entry}\\ \text{PHON}\langle\boxed{7}\rangle\\ \text{CAT}\ \ tv\\ \text{CONT}\ \boxed{11}\end{array}\right], \left[\begin{array}{l}\textit{lex-entry}\\ \text{PHON}\ \boxed{8}\\ \text{CAT}\ \ np\\ \text{CONT}\ \boxed{13}\end{array}\right]\right\rangle\end{array}\right]\right\rangle\end{array}\right]\right\rangle\end{array}\right]\right\rangle
\end{array}\right]$$

Vergleichen wir (104*c*) mit (1), und (104*a*) mit (2), so zeigt sich, daß wir am Anfang dieses Kapitels von den Zeichen ausgegangen sind, die wir nun durch unsere Grammatik erklären können.[36]

1.7 Defaults und Defaultvererbung

Monotone und nicht-monotone Logiken

Der Formalismus, der in den letzten Abschnitten vorgestellt wurde, kann als eine *monotone Logik* aufgefaßt werden. Monotone Logiken sind — einfach ausgedrückt — dadurch gekennzeichnet, daß Folgerungen, die aus einer Menge von Aussagen abgeleitet werden können, auch dann ihre Gültigkeit bewahren, wenn zu dieser Menge weitere Aussagen hinzugefügt werden. Diese Eigenschaft hat aus mathematischer und algorithmischer Sicht viele Vorteile; aus einer Perspektive, die sich an Eleganz und Ökonomie der Modelle orientiert, ist sie jedoch oft Ursache sperriger, unökonomischer Beschreibungen.

Logiken, in denen Folgerungen durch die Hinzunahme zusätzlicher Aussagen revidiert werden können, werden als *nicht-monotone Logiken* bezeichnet.

Defaultlogiken

Unter den nicht-monotonen Logiken sind aus Sicht der Linguistik insbesondere *Defaultlogiken* interessant: Sie gestatten die Formulierung von "Annahmen", die solange gültig bleiben, wie sie nicht durch andere, höher bewertete Annahmen überschrieben werden:

(105) *a.* *Alle Vögel können fliegen.*

b. *Pinguine sind Vögel, die nicht fliegen können.*

(105*a*) beispielsweise behält seine Gültigkeit für alle Vögel, solange keine weitere Aussage ihr widerspricht. In dem Moment, in dem (105*b*) hinzugenommen wird, gilt (105*a*) nur noch eingeschränkt für Vögel, die keine Pinguine sind: (105*b*) überschreibt (105*a*) für die Untermenge der Pinguine.

Defaulthierarchien

In der Linguistik gestatten Defaults eine elegante Modellierung von Ausnahmen: Das Perfekt japanischer Verben läßt sich beispielsweise durch Verknüpfung eines Stammes mit einem Perfekt-Suffix beschreiben. Das Perfekt-Suffix hat für konsonantische Verben die die Form *ita*. Nur ein konsonantisches Verb weicht davon ab: das Verb 行く (*iku*; gehen, fahren), das das Perfekt mit dem Suffix *tta* bildet.

36 Das Zeichen (2) wurde zur einfacheren Beschreibung vereinfacht dargestellt.

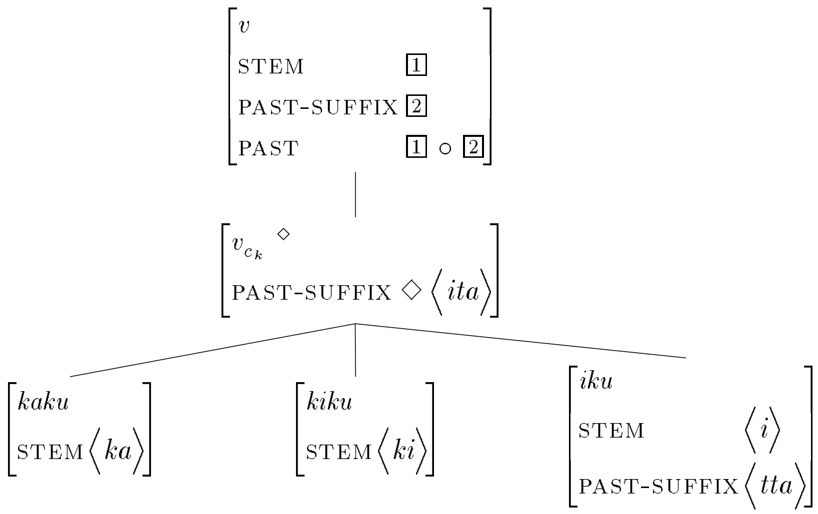
(106) *a.* 書く *kaku* ⟶ 書いた *kaita* *(schreiben)*

b. 聞く *kiku* ⟶ 聞いた *kiita* *(hören)*

c. 行く *iku* ⟶ 行った *itta* *(gehen)*

Zur Darstellung solcher Ausnahmen verwenden wir Hierarchien mit einem abweichenden Vererbungmechanismus, die wir als *Defaulthierarchien* bezeichnen. Durch eine Defaulhierarchie läßt sich die Abweichung des Perfektes von *iku* folgendermaßen beschreiben:[37]

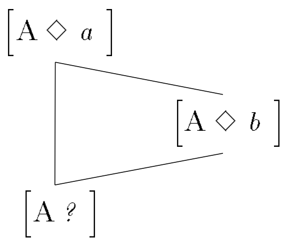
(107)



Die Raute ◇ dient der Hervorhebung von Defaults: Werte, die durch eine Raute markiert werden, lassen sich durch entsprechende Werte kleinerer Typen überschreiben. Damit Constraints, die Defaults enthalten, leichter erkannt werden können, markieren wir ihren äußeren Typ durch eine hochgestellte Raute.

Unterschiedliche Defaultwerte können kollidieren:

(108)



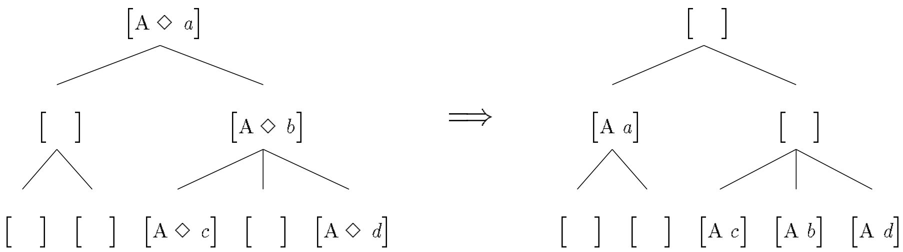
37 Die hier dargestellte Modellierung des Perfekts von Verben dient nur der Veranschaulichung der Defaultvererbung; sie entspricht nicht dem Vorgehen in unserem Modell der japanischen Verbmorphologie.

Solche Fälle können in unterschiedlicher Weise aufgelöst werden. In dem hier vorgestellten Modell kommen Kollisionen nicht vor. Wir können uns jedoch vorstellen, daß aufeinandertreffende Defaultwerte genauso behandelt werden, wie normale Merkmalwerte: durch Unifikation.

Von Defaulthierarchien zu monotonen Hierarchien

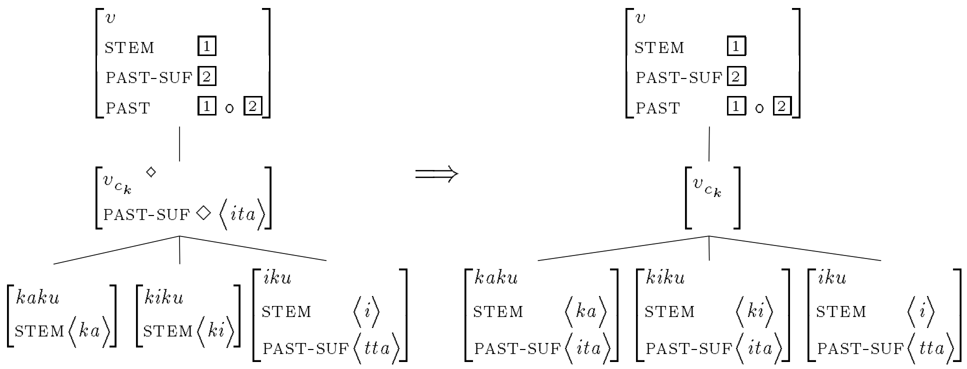
Defaulthierarchien lassen sich als eine Art "Metasprache" für monotone Hierarchien auffassen: Durch das "Herunterpropagieren" der Defaultwerte lassen sich alle Überschreibungen auflösen, so daß — wie das folgende Beispiel veranschaulicht — eine monotone Hierarchie entsteht:

(109) *Ableitung einer monotonen Hierarchie aus einer Defaulthierarchie:*



Für die Hierarchie (107), der Beschreibung des Perfekts k-konsonantischer Verben, würde sich in analoger Weise die folgende Hierarchie ergeben:

(110) *Ableitung der monotonen Hierarchie für die Perfektform:*



Defaulthierarchien, wie sie in dieser Arbeit verwendet werden, dienen also nur der verkürzten Notation monotoner Hierarchien. Für Beschreibungen, die mit Hilfe von Defaulthierarchien spezifiziert werden, läßt sich daher der Kalkül benutzen, der für monotone Beschreibungen im letzten Kapitel entwickelt wurde. Defaultlogiken, die sich nicht auf monotone Logiken reduzieren lassen, werden beispielsweise von Carpenter 1993 und Lascarides und Copestake 1999 behandelt.