gefördert vom
Bundesministerium
für Forschung
und Technologie

Verbmobil
Verbundvorhaben

# Erläuterungen zur Umsetzung einer HPSG im Basisformalismus STUF III

Stefan Geißler, Tibor Kiss

IBM Informationssysteme GmbH

Vm Report 19
August 1994

August 1994

Stefan Geißler, Tibor Kiss

IBM Informationssysteme GmbH
Institut für Logik und Linguistik
Vangerowstr. 18
69115 Heidelberg

Tel.: (06221) 59 - 4482/4483
Fax: (06221) 59 - 3200
e-mail: {sgeissler;kiss}@vnet.ibm.com

**Gehört zum Antragsabschnitt:** 6

Das diesem Bericht zugrundeliegende Forschungsvorhaben wurde mit Mitteln des Bundesministers für Forschung und Technologie unter dem Förderkennzeichen 01 IV 101 G gefördert. Die Verantwortung für den Inhalt dieser Arbeit liegt bei dem Autor.

**Abstract:**
In the following paper, we describe some aspects of an implementation of an HPSG-style grammar in the Verbmobil Basic Formalism STUF III. The presentation includes basic design considerations as well as fundamental properties of the current STUF III grammar, including the type system, type modules, constraints, and rule instances. The purpose of the following paper is to give the reader some basic guidelines regarding the representation of linguistic information in STUF III. Hence, theoretical analyses or reflections on the grammar of German are omitted from the current paper.

**Zusammenfassung:**
Der vorliegende Aufsatz beschreibt einige Grundzüge der Implementation einer HPSG im Verbmobil-Basisformalismus STUF III. Wir beginnen mit Betrachtungen zur Implementation der Grammatik und stellen einzelne Module der implementierten Grammatik vor. Hierbei diskutieren wir das Typsystem, das Konzept des Typmodules, die Behandlung von Sorten und ihre Beziehung zur Verarbeitung durch Regelinstanzen. Im vorliegenden Papier wird auf theoretische Untersuchungen zur Grammatik des Deutschen verzichtet; im Vordergrund steht vielmehr zu vermitteln, wie die einzelnen Konstrukte von STUF III bei der Grammatikimplementierung eingesetzt werden können.

## 0. **Vorüberlegungen**[1]

Im folgenden werden wir einige Aspekte der Implementation einer HPSG (Pollard/Sag 1994) für das Deutsche im Verbmobil-Basisformalismus STUF III (vgl. Momma/Opalka/Raasch 1994) beschrieben. Im Vordergrund steht hierbei weniger eine vollständige Dokumentation der Grammatik als vielmehr eine Erläuterung der verwendeten syntaktischen Konstrukte des Basisformalismus STUF III. Anderen Benutzern soll es so erleichtert werden, mit der Grammatik zu arbeiten bzw. den Formalismus zu verwenden, um eigene Grammatiken zu implementieren.

Wir beginnen unserer Überlegung mit der Erörterung einiger wesentlicher Eigenschaften der HPSG. Eine konstituierende Eigenschaft der HPSG ist die Deklarativität der Grammatik. Dies bedeutet, daß syntaktisches und semantisches Wissen unabhängig von einem konkreten Verarbeitungsmodell dargestellt werden kann. Ein Verarbeitungsmodell kann hierbei ein gewöhnlicher Parser für standardisiertes, geschriebenes Deutsch sein; genau so gut ist jedoch auch ein spezieller Verarbeitungsmechanismus für spontansprachliche Konstruktionen vorstellbar, der dennoch auf die deklarativ formulierte Grammatik zurückgreift. Dies ist gerade für Verbmobil relevant, da Reparaturen und ähnliche Phänomene nicht innerhalb des linguistischen Wissens, sondern durch geeignete Verarbeitungsmodule erfaßt werden sollen. Als Beschreibung sprachlicher Kompetenz macht die Basisgrammatik keine Aussagen zu ‚ungrammatischen' Äußerungen, wie etwa Reparaturen (1). Diese betrachten wir als Performanzphänomene, die prozedural zu beschreiben sind.

(1) *Extrakt aus mhs2-mjg1-tsponti1.trans:*

```
mjg1_1_02:
   <[jo] /h#/ ja ich /eh/ {seos}> von mir aus [k"onn(en)]
   [wer (wir)] [des (das)] machen {seos} am [beschden (besten)]
   w"ar' der /h#/ Monat Juli *pause* /h#/ <un' *pause* wann
   {seos}> welche Wochentage w"urden Ihnen am ehesten liegen
   #klick# {seos}
```

Wir plädieren somit für eine strenge Trennung der linguistischen Repräsentationen (Grammatik) von der linguistischen Verarbeitung (Parser). Diese Aufteilung ist schon deswegen sinnvoll, weil so sichergestellt werden kann, daß linguistisch motivierte Änderungen in der Grammatik durchgeführt werden können, ohne daß das Verarbeitungsmodell verändert werden muß. Auf der anderen Seite sollen weniger effiziente Methoden der Verarbeitung durch effizientere Methoden ersetzbar sein, ohne daß hierfür eine Recodierung der Grammatik erforderlich wäre. Die Grammatik wurde um einen einfachen Bottom-Up-Par-

[1] Wir danken Annette Opalka und Ingo Schröder für ihre Bereitschaft, ältere Versionen dieses Dokuments zu lesen und mit uns zu diskutieren.

ser erweitert, der allerdings auch die Verarbeitung von Lücken unterstützt. Der Parser wurde als rudimentäre Kontrollstruktur direkt in STUF III kodiert, da eine deklarative Grammatik ohne jede Kontrollstruktur nicht effizient verarbeitet werden kann. Dieser Parser dient allein der Steigerung der Entwicklungseffizienz und besitzt darüber hinaus keinerlei konzeptuellen Status. Zukünftige Versionen der Grammatik werden einen Parser verwenden, der vom Teilprojekt 15.7 (Univ. Erlangen) zur Zeit entwickelt wird. Dieser Parser wird über die externe Schnittstelle mit STUF III kommunizieren. Durch die strikte Trennung von Grammatik und Parser besteht daher die Möglichkeit, die Grammatik auch nach Austausch des Parsing-Moduls zu verwenden.

Die implementierte Grammatik teilt mit anderen HPSGen die folgenden Eigenschaften:

- Sie enthält wenige, sehr schematisierte (abstrakte) Regelschemata.
- Die Aufteilung der linguistischen Information führt zu einer starken Lexikalisierung.[2]
- Strukturelle und nicht regelspezifische Information wird durch Prinzipien repräsentiert.
- Hierarchische Information und Wortstellungsbeschränkungen werden modular in der Form von ID-Schemata und LP-Regeln formuliert.

Eine wesentliche Eigenschaft der HPSG konnte bislang nicht umgesetzt werden: In der vorliegende Verbmobil-HPSG werden nur syntaktische Regelmäßigkeiten beschrieben, da eine Integration mit anderen Wissensquellen, insbesondere der an der Univ. Saarbrücken vorgenommenen Semantikkonstruktion (vgl. Bos et al. 1994) noch nicht erfolgt ist.

Einige abschließende Bemerkungen zur Terminologie: Wenn wir im folgenden von einer HPSG sprechen, so meinen wir immer eine implementierte Grammatik, die wenigstens den o.g. Kriterien entspricht, wobei in diesem Kontext die Implementationssprache irrelevant ist. Sprechen wir von einer STUF-Grammatik, so meinen wir eine *in* STUF implementierte Grammatik. Da STUF III als Formalismus keineswegs deckungsgleich mit den formalen Grundlagen der HPSG ist, können in STUF beliebige Grammatiken (allgemeiner: Programme) implementiert werden. Wir haben eine Grammatik in STUF III implementiert, die den Grundideen der HPSG verpflichtet ist; die implementierte Grammatik ist eine STUF-Grammatik, aber eine STUF-Grammatik muß keine HPSG sein.

---

[2] Wie allgemein in der HPSG üblich, liegt der Grad der Lexikalisierung deutlich unter dem einer Kategorialgrammatik, in der nur wenige genuin syntaktische Operationen verwendet werden und die Formulierung von Prinzipien je nach Format der Kategorialgrammatik relativ schwierig ist. Vgl. hierzu Kiss (1991).

## 1. Typen in STUF III

Innerhalb der HPSG besitzt der Begriff des *Typs* bereits seit Pollard/Sag (1987) eine Bedeutung, die vom Typenkonzept in STUF III deutlich abweicht. Das folgende Zitat aus Pollard/Sag (1994:17) verdeutlicht, daß die ontologischen Objekte, über die die Theorie spricht, getypt sind und daß diesen Typen in der Theorie jeweils *Sorten* entsprechen, die die *Typen* modellieren.

> „[T]he feature structures employed in HPSG are *sorted.* This means simply that each node is labelled with a *sort symbol* that tells which type of object the structure is modelling; i.e. there is one sort symbol for each basic type (ontological category) of construct."

In STUF III denotieren Typen Mengen von Objekten, sie können also im Sinne der HPSG-Sorten verstanden werden, wenn wir davon ausgehen, daß eine HPSG-Sorte die Menge der ontologischen Objekte eines Typs beschreibt. Um Verwirrungen zu vermeiden, werden wir im folgenden nur noch die STUF III-Terminologie verwenden.

Wird in einer STUF-Grammatik ein Typ definiert, so geschieht dies durch Angabe der hierarchischen Beziehung dieses Typs zu anderen Typen und durch Spezifikation der Eigenschaften von Objekten dieses Typs.

Alle Typen sind Untertypen des vordefinierten allgemeinsten Typs *top.* Für einen beliebigen Typ $\alpha$ gilt hierbei zum einen, daß *top* als direkter Supertyp von $\alpha$ angesetzt wird, wenn kein anderer Supertyp für $\alpha$ definiert ist. Zwei Subtypen $\alpha$ und $\beta$ von $\gamma$ sind solange disjunkt, wie nicht gefordert wird, daß es Subtypen gibt, die sowohl vom Typ $\alpha$ als auch vom Typ $\beta$ sind. Aus der Hierarchie in (2) darf darüber hinaus nicht abgeleitet werden, daß *sign* nur die Subtypen *word* und *phrase* besitzt. Eine solche Partition ist außerhalb einer Typmoduls *(type module,* siehe unten) nicht möglich.

(2) *einfache Typhierarchie*

```
word < sign.
phrase < sign.
```

Die interne Struktur eines Typs wird durch Benennung der durch diesen Typen eingeführten Merkmale und deren Werte charakterisiert. Hier wird also definiert, welche Merkmale für diesen Typ angemessen sind. Die folgende Definition des Typs *sign* kann daher auch in Anlehnung an Carpenter (1991) als *Appropriateness Condition* interpretiert werden.

(3) *Appropriateness Condition*

```
sign::
   PHON: list,
   SYN: syn,
   SEM: sem.
```

Diese Definition sagt aus, daß der Typ *sign* die Merkmale PHON, SYN und SEM besitzt und daß der Wert des Merkmals PHON vom Typ *list* sein muß. In STUF III wird nicht die Annahme vertreten, daß ein Typ vollständig spezifiziert sein muß. Der Typ in (2) kann einen Subtypen besitzen, der zusätzlich zu PHON, SYN und SEM weitere Merkmale besäße. Man spricht hier von einer *open-world-assumption* im Gegensatz zu einer *closed-world-assumption,* in der nur die Objekte existieren, die explizit definiert worden sind. Merkmale werden in der Grammatik von Typen durch die Schreibung unterschieden. Merkmale werden immer in Versalien dargestellt, Typen immer durch vollständige Kleinschreibung.[3]

Da die Definition in (2) festlegt, daß *word* und *phrase* Subtypen von *sign* sind und *sign* die Merkmale in (3) besitzt, folgt, daß auch *word* und *phrase* diese Merkmale besitzt. STUF III unterstützt somit die Vererbung von Information, die durch *Appropriateness Conditions* festgelegt werden kann. Hieraus darf nicht abgeleitet werden, daß STUF III im allgemeinen Fall die Vererbung von Informationen unterstützt. Dies werden wir weiter unten diskutieren.

Im Gegensatz zu anderen Formalismen gestattet STUF III die Verwendung von Polymerkmalen nicht. Dies bedeutet, daß jeder Pfad genau bei einem Typ eingeführt werden muß. Bezogen auf das Beispiel in (3) wäre es also nicht möglich, einen weiteren Typ zu definieren, der ebenfalls das Merkmal SYN einführt. Durch das Verbot von Polymerkmalen kann innerhalb von STUF III aus dem Vorhandensein eines Pfads automatisch auf den geringsten Typ des betreffenden Objekts geschlossen werden (beispielsweise *sign* für das Merkmal SYN). Darüber hinaus wird so sichergestellt, daß ein Wert eines Merkmals keine Merkmale einführt, die bereits anderweitig definiert sind.[4]

---

[3] Diese Konvention geht auf eine syntaktische Beschränkung in STUF III 1.1 zurück, die für die Kodierung von Grammatiken irrelevant ist, die sog. *Eindeutigkeitsforderung* für Symbole. Diese besagt, daß ein gegebener Name nur einmal verwendet werden darf, d.h. ein Name darf nicht gleichzeitig für Merkmale und Typen verwendet werden. Anstelle der hier verwendeten Konvention könnten auch alle Typen an der zweiten Stelle mit einem Großbuchstaben oder einer anderen eindeutigen Kennzeichnung versehen werden.

[4] In der vorliegenden Version von STUF III (1.1) wird diese Beschränkung aus unserer Perspektive zu restriktiv gehandhabt, da so auch die Spezialisierung eines Typs verboten wird. Eine Spezialisierung kann wie folgt beschrieben werden: Nehmen wir an, daß ein Typ $\alpha$ das Merkmal A mit dem Wert $\beta$ besitzt. Nehmen wir weiterhin an, daß $\alpha$ einen Subtyp $\alpha$' besitzt, der sich von $\alpha$ nur dadurch unterscheidet, daß der Wert des Merkmals A nicht $\beta$ sondern $\beta$' ist, wobei gilt, daß $\beta' < \beta$. Die entsprechende Typdeklaration in (i) ist bislang unzulässig.

Innerhalb eines *Typmoduls* besteht die Möglichkeit, Partitionen aufzuspannen. Ein Beispiel für ein *type module* ist in (4) gegeben. Hier wird gesagt, daß nur zwei verschiedene Subtypen des Typs *case* existieren.

(4)
```
type module case.
   case = struc_case ; lex_case.
   struc_case = nom ; acc.
   lex_case = gen ; dat.
   nom || acc || gen || dat.
end.
```

Die Repräsentation in (4) orientiert sich an der in der theoretischen Linguistik üblichen Unterscheidung zwischen strukturellen und lexikalischen Kasus (vgl. Haider 1985). In einem *type module* können beliebige aussagenlogische Ausdrücke verwendet werden – neben der Konjunktion und der Disjunktion ist auch die Negation von Typen zulässig. Es muß allerdings beachtet werden, daß in einem *type module* nicht auf benutzerdefinierte Typen zugegriffen wird, die außerhalb des Moduls definiert wurden. So wird in (4) ausgesagt, daß die Typen *nom* und *acc* disjunkt sind, während *struc_case* und *lex_case* nicht disjunkt sind. Es wäre also denkbar innerhalb des *type modules* einen Kasus zu definieren, der sowohl strukturell als auch lexikalisch ist (beispielsweise den Akkusativ – vgl. Kiss 1995).

Innerhalb der vorliegenden Grammatik stellt das *type module head* insbesondere in bezug auf die Integration semantischer Eigenschaften eine Übergangslösung dar. Eine Reihe der dort kodierten Unterscheidungen von Typen, würde bei der angestrebten parallelen Repräsentation von syntaktischer und semantischer Information auf syntaktische und semantische Module verteilt. Ein Beispiel für eine Unterscheidung, die semantisches Wissen nachmodelliert ist die Trennung in *abstract_ref* und *substantive_ref*. Durch den Typ *abstract_ref* werden alle Kategorien beschrieben, die als Modifikatoren agieren.[5]

---

(i)
```
α  :: A: β.
α' :: A: β'.
```
Durch die Blockade der Spezialisierung wird die Repräsentation von Typdeklarationen gemäß Pollard/Sag (1994) stark beschränkt. Es erscheint daher sinnvoll, die Syntax von STUF III dahingehend zu ändern, daß ein Merkmal bei mehr als einem Typ eingeführt werden darf, wenn für den Typ, der das Merkmal einführt und für den Wert, den das Merkmal besitzt, jeweils Subsumtion konstatiert werden kann, wie oben dargestellt.

[5] Anhand dieses knappen Beispiels wird auch deutlich, daß die parallele Repräsentation von syntaktischer und semantischer Information und die daraus resultierende parallele Verarbeitung einer rein sequentiellen Verarbeitung gegenüber vorzuziehen ist, da eine reine syntaktische Analyse entweder – wie oben beschrieben – gezwungen ist, semantische Eigenschaften nachzuzeichnen, oder aber die syntaktische Verarbeitung eine Vielzahl von Analysen liefert, die anschließend durch semantische Filterung beschränkt werden müssen.

Obwohl das Typ-Konzept in STUF die Repräsentation komplexer Typverbände erlaubt, unterliegen diesem Konzept einige Beschränkungen, die es nicht ermöglichen, Grammatiken nur durch Typen zu implementieren. Die wohl relevanteste Beschränkung betrifft die Verwendung von Pfadgleichungen: Innerhalb einer Typdefinition können keine Pfadgleichungen angegeben werden. Dies bedeutet, daß in einem Typ α, von dem wir nicht nur wissen, daß er die Merkmale A und B einführt, sondern auch, daß diese Merkmale immer übereinstimmen müssen, nicht in der in (5) gegebenen Form implementieren können.

(5) Unzulässige Integration einer Pfadgleichung in eine Typdefinition

```
α::
   A: #1 & β,
   B: #1.
```

Die Definition in (5) würde besagen, daß A und B Werte des Typs β besitzt und daß diese Werte immer übereinstimmen. Dies kann in STUF III so nicht ausgedrückt werden.

## 2. Sorten und Constraints in STUF III

Das in STUF III verwendete Konzept der Sorte hat in einer ‚theoretischen' HPSG kein direktes Analogon. Eine hilfreiche Analogie zur Bedeutung einer Sorte in STUF ist es, die Sorte als Prolog-Prädikat zu interpretieren. Wenn man diese Sichtweise annimmt, kann man etwa die Verwendung von relationalen Abhängigkeiten wie *append* etwa im *Subcategorization Principle* (Pollard/Sag 1987) als Instanzen eines Sortenkonzepts betrachten. Der Begriff *Sorte* kann auch gleichgesetzt werden mit dem Terminus *Constraint.* Durch Sorten werden diejenigen Eigenschaften von Objekten repräsentiert, die über reine Hierarchiebeziehungen bzw. Definitionen der internen Struktur in der Form von *Appropriateness Conditions* hinausgehen.

Da Sorten wie Prolog-Prädikate interpretiert werden können, erlauben sie die Darstellung komplexer Zusammenhänge zwischen Merkmalen. So kann man etwa die *append*-Relation zwischen zwei Listenwerten oder andere rekursive Beziehungen in der Form einer Sorte repräsentieren. Allerdings müssen auch triviale Fälle von Abhängigkeiten zwischen Merkmalen durch Sorten kodiert werden. Dies betrifft etwa die Repräsentation von Pfadgleichungen. So kann die Sorte in (6) interpretiert werden als Implementation des *Head Feature Principle,* das besagt, daß die HEAD-Merkmale einer Phrase mit den HEAD-Merkmalen der Kopftochter übereinstimmen.[6]

---

[6] Als Notationskonvention werden alle Sorten der vorliegenden STUF-Grammatik mit dem Suffix ‚_s' versehen. So kann die Beziehung zwischen einem beliebigen Typ α und der assoziierten Sorte α_s leicht kenntlich gemacht werden. Soll in STUF III eine HPSG umgesetzt werden, müssen Typen und Sorten als

(6) *Head Feature Principle* als STUF-Sorte:

```
hfp_s =>
 SYN: LOC: HEAD: #Head &
 DTRS: H_DTR: SYN: LOC: HEAD: #Head.
```

Durch die Sorte *hfp_s* wird nur die Koreferenz zwischen dem HEAD-Wert der Kopftochter und dem HEAD-Wert einer *phrase* etabliert. Hieraus folgt jedoch nicht, daß nun durch diese Sorte alle Typen subsumiert werden, die durch die o.g. Merkmalstruktur subsumiert werden. Diese Interpretation der Prinzipien als allgemeine Beschränkungen über Phrasen kann so in STUF nicht umgesetzt werden, weil *Sorten* keinerlei Vererbungseigenschaften besitzen. Statt dessen besteht die Notwendigkeit, Sorten wie (4) explizit mit Typen zu verbinden und die Vererbung jeweils nachzukodieren. Dies kann auf zwei verschiedene Weisen erfolgen.

Eine Sorte σ' übernimmt dann die Beschränkungen einer anderen Sorte σ, wenn in der Definition von σ' ein Aufruf von σ enthalten ist. Um etwa auszudrücken, daß das *Head Feature Principle* in der o.g. Form auf alle Phrasen angewandt werden muß, nehmen wir in (7) zunächst die Definition einer trivialen Sorte *sign_s* an. Es ist die einzige Eigenschaft dieser Sorte, vom Typ *sign* zu sein. Anschließend definieren wir die neue Sorte *phrase_s,* die dann explizit als Subsorte von *sign_s* und *hfp_s* gekennzeichnet ist und somit auch die durch das *hfp_s* ausgedrückte Eigenschaft der Strukturteilung erbt.

(7)
```
sign_s => sign.
phrase_s => sign_s & hfp_s.
```

Problematisch ist bei dieser Vorgehensweise vor allem, daß bei einem Aufruf der Beschränkung *sign_s* nichts weiter passiert, als daß die Typisierung überprüft wird. Tatsächlich würde man bei einem Aufruf von *sign_s* als Ergebnis alle vollständig ausspezifizierten Objekte erwarten, die dem Typ *sign* genügen. Diese Interpretation eines Constraints findet sich in Pollard/Sag (1994).

Alternativ kann in einer Supersorte σ der Aufruf einer Subsorte σ' enthalten sein. Dies ist in (8) dargestellt. Hier enthält die Sorte *sign_s* die zusätzliche Beschränkung *phrase_s,* wobei implizit angenommen wird, daß *phrase_s* eine Subsorte von *sign_s* ist.

---

unterschiedliche Beschreibungsmittel *derselben* Objekte betrachtet werden. Eine Assoziierung von Typen und Sorten ist daher für den Grammatikautor unumgänglich.

(8) *Top-Down-Kodierung der Vererbung von Constraints:*

```
sign_s =>
   hfp_s &
   subcatp_s &
   …
   phrase_s.

phrase_s => id1 & lp1_s.
phrase_s => id2 & lp2_s.
```

Diese aus linguistischer Perspektive zunächst kontraintuitive Kodierung der Vererbungsverhältnisse erlauben bei einem Aufruf der Sorte *sign_s* auch die vollständige Ableitung eines Zeichens unter Berücksichtigung der Prinzipien, da STUF III nun versucht, die im Rumpf von *sign_s* aufgeführten Sorten abzuleiten. Das Denotat einer Supersorte wird somit explizit als Vereinigung der Denotate aller ihrer Subsorten definiert. Wenn wir also beispielsweise davon ausgehen, daß die Prinzipien nicht für Objekte des Typs *word* gelten sollen, weil dieser kein DTRS-Attribut einführt, können wir dies wie in (9) definieren.

(9) *Definition von Supersorten:*

```
sign_s =>
   word ;
   ( phrase &
     hfp_s &
     phrase_s ).
```

Nun können wir mit dem Aufruf der Sorte *sign_s* Phrasen und Wörter ableiten.

Die Definition rekursiver parametrisierter Sorten unterscheidet sich von von der Definition einfacher Sorten nur dadurch, daß der Sorte Parameter zugeordnet werden. Als Beispiel für eine rekursive Sorte ist in (10) die Definition der Listenkonkatenationsoperation *append_s* angegeben.

(10) *Rekursive Sorte:*

```
append_s([], #SecondList) =>
   #SecondList.
append_s([#First | #Rest], #SecondList) =>
   [#First | append_s(#Rest, #SecondList)].
```

Eine spezielle parametrisierte Sorte ist die – benutzerdefinierte – Sorte *true_s.* Diese erlaubt die Kapselung einer Berechnung ohne aktuelle Instantiierung der berechneten Werte. Die Definition von *true_s* ist in (11) gegeben.

(11) `true_s( #Something ) => #SomethingElse.`

Innerhalb des Parameters von *true_s* wird eine Berechnung durchgeführt, deren Ergebnis nur dann nach ‚außen' weitergegeben wird, wenn die Referenz über eine Variable im Körper von *true_s* erfolgt. Dies kann anhand der Beschränkungen in (12) erläutert werden.[7]

(12) a. Weitergabe des Parameters außerhalb von *true_s*

```
(a : #X & [1]) &
(b : #Y & [2]) &
(c : #Z) &
true_s(append_s(#X, #Y) & #Z)
```

liefert: $\begin{bmatrix} a & [1] \\ b & [2] \\ c & [1, 2] \end{bmatrix}$

b. Verkapselte Auswertung innerhalb von *true_s*

```
(a : #X & [1]) &
(b : #Y & [2]) &
(c : #Z) &
true_s(append_s(#X, #Y))
```

liefert: $\begin{bmatrix} a & [1] \\ b & [2] \\ c & \#Z \end{bmatrix}$

Anstatt den Typ eines Objekts in eine parametrische oder nicht-parametrische Sorte zu integrieren, können auch Typrestriktionen verwendet werden, die festlegen, welcher Typ durch eine Sorte beschränkt wird. Beispiele für Typrestriktionen sind in (13) gegeben.

(13) a. Typrestriktion bei nicht-parametrischer Sorte:

```
@sign_s -> sign.
```

b. Typrestriktion bei parametrischer Sorte:

```
@append_s(list, list) -> list.
```

## 3. Macros und das Lexikon

Im Gegensatz zu Typen und Sorten, die im Inferenzprozeß eingesetzt werden, spielen Macros in der Inferenzmaschine keine Rolle – sie dienen lediglich der einfacheren Repräsentation. Die Macros werden zur Compile-Zeit expandiert; jedes Vorkommen eines Macros im Code wird durch den entsprechenden Code im Körper des Macros ersetzt. Aus

---

[7] Es sollte nicht unerwähnt bleiben, daß die Verwendung der Sorte *true_s* im Beispiel (12a) überflüssig ist, da als Wert des Merkmals *c* auch *append_s(#X, #Y)* festgelegt werden könnte. (12a) dient daher nur der Illustration.

diesem Grund dürfen Macros im Gegensatz zu Sorten nicht rekursiv definiert werden. Ein Beispiel für Macrodefinitionen ist in (14) gegeben.[8]

(14) Macrodefinitionen:

```
subcat( *SubcatList ) :=
   SYN: LOC: SUBCAT: *SubcatList.
sat_cat := subcat([]).
```

Gegeben die Macrodefinitionen in (14) expandiert der Aufruf `sat_cat` zu der Struktur `(SYN: LOC: SUBCAT: [])`. Macros werden besonders im Lexikon verwendet, um dort statische Information zu repräsentieren. Ein Beispiel für die Verwendung von Macros ist in (15) gegeben. Hier enthält die Definition des Lexems *guter* die Macrodefinitionen *adj* und *sat_cat.* Die Definition der Macros *adj* ist in (16) gegeben.

(15) *guter:*

```
word_s(guter) =>
   lexical_token &
   adj(guter, adj_er) &
   sat_cat.
```

(16)

```
adj(*Phon, *Ending) :=
   PHON : [*Phon] &
   head( adj &
         MOD_DIR : right &
         REG_DIR : left &
         AGR : #Agr &
         MOD: SYN: LOC: HEAD ( nominal & AGR : #Agr )) &
   *Ending.
```

Neben statischer Information können Lexikoneinträge auch dynamische Beschränkungen enthalten, so etwa die *Argumentanziehung* bei kohärent konstruierenden Verben (vgl. Hinrichs/Nakazawa 1990, Kiss 1992). So erhält das Wort *kann* die Repräsentation in (17), in der die SUBCAT-Liste durch die Konkatenation der SUBCAT-Liste des regierenden Verbs mit der SUBCAT-Liste des regierten Verbs determiniert wird.

---

8 Macroparameter werden in STUF III von Sortenparametern durch Präfigierung unterschieden. Die Verwendung des Asterisk (*) ist in den Macrodefinitionen somit nicht optional.

(17) *kann:*

```
word_s(kann) =>
   lexical_token &
   verb(kann, fin_verb, sg, ¬second, #Anchor) &
   subcat([#Subj |
           append_s(#Comps, [LEX : yes &
                             subcat(#Comps) &
                             subj([#Subj])])]).
```

## 4. Zur Repräsentation von ID-Schemata

### 4.1 ID-Schemata

HPSG-Regelschemata enthalten keine kategorienspezifischen Informationen und keinerlei Informationen über die lineare Abfolge der Töchter. In der vorliegenden Grammatik liegen drei Schemata vor, die den grundlegenden Kombinationsmodi der *Komplementation,* der *Adjunktion* und der *Lückenfüllung* entsprechen. Bei der Erweiterung der Grammatik werden weitere Regelschemata hinzukommen, die ggfs. spezifischer sind als die vorgestellten, wobei auch kategorialen Information nicht ausgeschlossen werden kann.

Das Komplementationsschema ist in (18) dargestellt.

(18) *Komplementationsschema:*

```
phrase_s => id1 & lp1_s.
id1 :=
   DTRS : ( head_comp_struc &
            C-DTR : [sign] &
            H-DTR : SYN : LOC : SUBCAT : nelist ).
```

Die erste Bedingung in (18) besagt, daß eine *Phrase* aus der Unifikation des Schemas *id1* mit der LP-Beschränkung *lp1_s* gebildet wird. Das Schema selbst ist momentan in der Form eines Macros kodiert. Dieses Macro beschreibt eine *Phrase,* die eine C-DTR-Liste besitzt, die genau ein Element enthält. Die H-DTR dieser Phrase besitzt eine nicht-leere SUBCAT-Liste.

Informationen über die lineare Abfolge der Töchter werden momentan durch die Merkmale MOD-DIR und REG-DIR erfaßt. Diese Merkmale legen fest, in welchem Verhältnis ein Funktor zu seinem Argument realisiert werden kann. Vereinfachte Linearisierungsbeschränkungen für Köpfe, die nach links regieren (A, V) sind in (19) aufgeführt.[9]

---

[9] Tatsächlich ist head_final in der implementierten Fassung allgemeiner und erfaßt so Köpfe in allen Regelschemata, während die hier vorliegende Form nur Köpfe in Kopf-Komplement-Strukturen erfaßt.

(19) *Linearisierungsbeschränkungen:*

```
lp1_s =>
   DTRS: H-DTR:
     ( SYN: LOC: HEAD: REG-DIR: left ) &
   head_final(head_comp_struc).

head_final(*Dtrs) :=
   DTRS: ( *Dtrs & head_comp_struc &
                   ( H-DTR: PHON: #Phon2 &
                     C-DTR: [PHON: #Phon1] )) &
   PHON: append_s(#Phon1, #Phon2).
```

### 4.2 Regelinstanzen

Die Schnittstelle zum Parser bilden nicht die Regelschemata, sondern Regelinstanzen, die Informationen der Grammatik redundant bündeln und an den Parser weitergeben. Instanzen sollten in zukünftigen Versionen der Grammatik durch automatische Compilation gewonnen werden. Ein Beispiel für eine Regelinstanz ist in (20) gegeben. Diese Instanz beschreibt Phrasen, deren Kopf links vom Komplement liegt.

(20) *Regelinstanz:*

```
rule_s( #Sign1, #Sign2 ) =>
   id1 & lp1_s &
   PHON: append_s( extract_phon_s( #Sign1 ),
                   extract_phon_s( #Sign2 ) ) &
   DTRS: ( H-DTR: #Sign1 &
           C-DTR: [#Sign2] ).
```

Das Prädikat *extract_phon_s* extrahiert den PHON-Wert eines Zeichens. Hierbei handelt es sich um Listen, die durch *append_s* in der o.g. Reihenfolge verknüpft werden.

## 5. Sprachliche Zeichen in der Grammatik

In der Grammatik können wir zwischen drei verschiedenen Strukturen unterscheiden, die abgeleitet werden können. Zum einen sind dies *Wörter,* d.h. Zeichen, die nicht als Ergebnis der Kombination syntaktischer Zeichen lizensiert werden, sondern dadurch, daß sie entweder Elemente des Kernlexikons sind oder durch eine lexikalische Regel abgeleitet werden können.

*Phrasen* – die zweite Gruppe sprachlicher Zeichen – bilden das getypte Ergebnis der Kombination einfacherer Zeichen zu komplexeren. *Phrasen* werden durch die Beschränkungen eines der ID-Schemata, eines entsprechenden LP-Schemas und aller Prinzipien lizensiert. Die Grammatik erlaubt die Verarbeitung von Phrasen beliebiger Komplexität, sie ist also keineswegs auf die Abarbeitung von Sätzen beschränkt. Der Benutzer kann mittels Parametrisierung der Verarbeitung entscheiden, ob nur Sätze oder alternativ alle saturierten oder auch beliebige unsaturierte Phrasen abgeleitet werden sollen. Die jeweilige Einstellung beeinflußt hierbei natürlich das Verarbeitungsverhalten und vor allem – die Verarbeitungsdauer.

Neben *Wörtern* und *Phrasen* erlaubt die Grammatik auch die Ableitung sog. *Sequenzen.* Eine Sequenz besteht aus einer beliebige Abfolge wohlgeformter Zeichen, so wie etwa im Beispiel (21), das nicht als Satz, sondern als Sequenz dreier syntaktisch unverbundener Zeichen analysiert wird, wie die Segmentierung andeutet.

(21) Oh | ja | dann treffen wir uns in der Eingangshalle.

Abgesehen davon, daß die Elemente einer Sequenz jeweils wohlgeformt sein müssen, werden keine weiteren Beschränkungen für den Aufbau einer Sequenz formuliert. Die Verwendung von Sequenzen in der Grammatik muß als vorläufiges Hilfsmittel betrachtet werden, das zugunsten eines elaborierten Verarbeitungsmoduls eliminiert werden sollte. Sequenzen als linguistische Einheiten zu betrachten, führt beispielsweise zu systematisch sinnlosen Ambiguitäten bei Verbzweitsätzen, da diese einerseits als Sequenz aus einem Satz, andererseits aber auch als Sequenz aus saturierter Phrase und Vorfeldellipse analysiert werden können, wenn die vorliegende prosodische Information keine weiteren Informationen liefert.

## 6. **Anmerkungen zur Verarbeitung**

Da in der vorliegenden Form nicht auf ein elaboriertes Parsingkonzept zurückgegriffen wird, sondern die Grammatik einen speziellen Parser in STUF enthält, ist sie nicht gänzlich unabhängig von der Verarbeitung. So wird beispielsweise in der vorliegenden Grammatik nicht spezifiziert, daß die Töchter eines Zeichens ebenfalls wohlgeformte Zeichen sein müssen. Dies wirkt sich bei unterschiedlichen Verarbeitungsmodi jeweils unterschiedlich aus:

So werden die terminalen Knoten bei einer *Bottom-Up-Verarbeitung* jeweils durch das Lexikon lizensiert. Somit ist sichergestellt, daß alle Töchter in einem Zeichen selber Zeichen sind. Wird statt der *Bottom-Up* eine *Top-Down-Verarbeitung* gewählt, ist es allerdings notwendig, eine entsprechende Beschränkung in die Grammatik einzuführen. Diese muß

besagen, daß die Töchter eines Zeichens jeweils auch Zeichen sind. Verwendet man ein Verfahren, das dem *Left-Corner-Parsing* (vgl. Alshawi 1992) entlehnt ist, so muß lokal an jedem Teilbaum überprüft werden, ob das untersuchte Zeichen wohlgeformte Töchter besitzt.

Die vorliegende Grammatik verwendet eine *Bottom-Up-Verarbeitung,* eine entsprechende Beschränkung muß also nicht implementiert werden.

## 7. **Literatur**


Alshawi, H. (Hrsg) (1992): *The Core Language Engine.* Cambridge/London: The MIT Press.

Bos, J./E. Mastenbroek/S. Millies/S. McGlashan/M. Pinkal (1994): *The Verbmobil Semantic Formalism. Version 1.3.* Verbmobil Report 6. Saarbrücken: Universität des Saarlandes.

Carpenter, B. (1991): *The Logic of Typed Feature Structures.* Cambridge: Cambridge University Press.

Haider, H. (1985): *The Case of German.* In: Toman, J. (Hrsg.): Studies in German Grammar. Dordrecht: Foris, S. 65-101.

Hinrichs, E./T. Nakazawa (1990): *Subcategorization and VP Structure.* In: Hughes, S. (Hrsg): Proceedings of the Third Symposium on Germanic Linguistics. Amsterdam: J. Benjamins.

Kiss, T. (1991): *The Grammars of LILOG.* In: Herzog, O./C.-R. Rollinger: Textunderstanding in LILOG. Berlin: Springer Verlag.

Kiss, T. (1992): *Variable Subkategorisierung. Eine Theorie unpersönlicher Einbettungen im Deutschen.* Linguistische Berichte 140, S. 256-293.

Kiss, T. (1995): *Infinite Komplementation.* Neue Studien zum deutschen Verbum infinitum. Tübingen: Max Niemeyer Verlag.

Momma, S./A. Opalka/I. Raasch (1994): *STUF III User Manual.* Institut für Logik und Linguistik, IBM Deutschland Informationssysteme, Heidelberg.

Pollard, C./I.A. Sag (1987): *Information-Based Syntax and Semantics. Vol. I: Fundamentals.* Chicago: University of Chicago Press.

Pollard, C./I.A. Sag (1994): *Head-Driven Phrase Structure Grammar.* Chicago: University of Chicago Press.