A Web-based Course in Grammar Formalisms and Parsing

Frank Richter
Seminar für Sprachwissenschaft
Abteilung Computerlinguistik
Eberhard-Karls-Universität Tübingen

August 11, 2006

Contents

Summary

Abstract

This electronic course book is an introduction to the development of grammars, based on a constraint-based grammar formalism, Head-driven Phrase Structure Grammar (HPSG). Special emphasis is placed on the combining of several topics in the field of HPSG, which are usually treated separately. These topics are the practical development of grammar fragments, linguistic theory formation, modern programming methods, mathematical foundations of constraint-based grammar frameworks and the implementation of linguistic theories with the methods of computational linguistics. An integrated study of these seemingly heterogeneous issues will provide new perspectives and will lead to a better understanding of how they are related. The mathematical foundations of constraint based grammar formalisms are presented in the format of interactive teaching software. This course is an instance of the paradigm of web based training. This means that students of this course are expected to work through the lessons on their own, using the educational software provided by the course. They should regularly work through exercises on their computers, which will help them to check their progress. Communication with lecturers and among students takes place on an electronic learning platform. However, the course materials can also be used in a traditional classroom setting or for self-study.

This course uses the interactive web based instructive software MorphMoulder *(MoMo) to teach the formal foundations of HPSG, and the grammar development environment TRALE for grammar implementation. These software tools were created, or significantly extended, within the framework of the MiLCA-Consortium located at the Seminar für Sprachwissenschaft in Tübingen, in collaboration with Professor Detmar Meurers from Ohio State University and Professor Gerald Penn from the University of Toronto.*

In this seminar we will take a closer look at the constraint-based grammar framework of HPSG from at least three significantly different perspectives. At the beginning of the course, we will study the mathematical structure of HPSG grammars. What is an HPSG grammar from a formal point of view, and what is the meaning of an HPSG grammar? Then our task will be to write grammars of natural languages in the HPSG formalism, and to do that, we will have to review a few basic linguistic notions which will be treated

in our grammars. We will learn how to use our HPSG specifications of natural languages as input to a computational system, in order to make queries to the system about the natural language which we have specified. We will see how we can do this on one particular implementation platform, and we will learn to recognize and pay attention to the procedural aspects and problems which occur when a powerful declarative language of the kind employed by HPSG is used in a general computational environment.

In the mathematical section of this course we will identify the most important components of a constraint-based grammar by investigating a prominent example, the HPSG grammar of English specified by Carl Pollard and Ivan Sag in the appendix of *Head-Driven Phrase Structure Grammar* [Pollard and Sag, 1994]. We will clarify the notion of *Sort Hierarchy*, and explain in which sense Pollard and Sag's *Partitions* and *Feature Declarations* provide the essential syntactic vocabulary for formulating their *Principles* of grammar. We will also define a certain kind of *Feature Structures*, which will serve as the modeling domain for grammars. We will briefly discuss the relationship between our modeling domain of feature structures and empirical (observable, measurable) phenomena, and we will mention alternative ways of defining a model theory for HPSG grammars and the motivation behind existing alternative proposals.

Among the notions which we will clarify in this first section of the course are: *sort hierarchy*, *attribute*, *multiple inheritance*, *feature structure*, *sort-resolvedness*, *total well-typedness*, *grammar*, *constraint satisfaction* and *feature structure admission*. All of them will be visualized, and some of them can be explored interactively in the *Morph Moulder* (MoMo), which is a teaching software for the formal foundations of constraint-based HPSG. The screen shot below shows a feature structure in MoMo.

With a precise understanding of HPSG grammars, we can then turn our attention to linguistic theories formulated in HPSG. Assuming some familiarity with the most important notions of current linguistic theorizing in syntax, we will briefly review how they are expressed. Step by step we will increase the number of phenomena covered in our grammars. The phenomena which will be covered include a theory of syntactic selection, phrase structure, agreement, government, raising constructions, thematic roles, word order, and unbounded dependency constructions. Another important topic will be the issue of lexical generalizations, and we will look at mechanisms such as lexical rules, which play an important role in the discussion in the syntactic literature.

Our introduction to linguistic phenomena and their analysis will go hand in hand with their implementation in TRALE. As Detmar Meurers recently put it in the description of a course he taught at The Ohio State University, Columbus:

> From the linguistic perspective, the development of grammar fragments can be an important means of obtaining feedback on the empirical consequences of a linguistic theory and the compatibility of the various theories which are integrated in the grammar fragment. I would argue that one can go one step further by stating that comprehensive grammar fragments integrating the state-of-the-art of syntactic theorizing are essential for reestablishing the credibility of generative syntax as a science with a measurable criterion for progress.

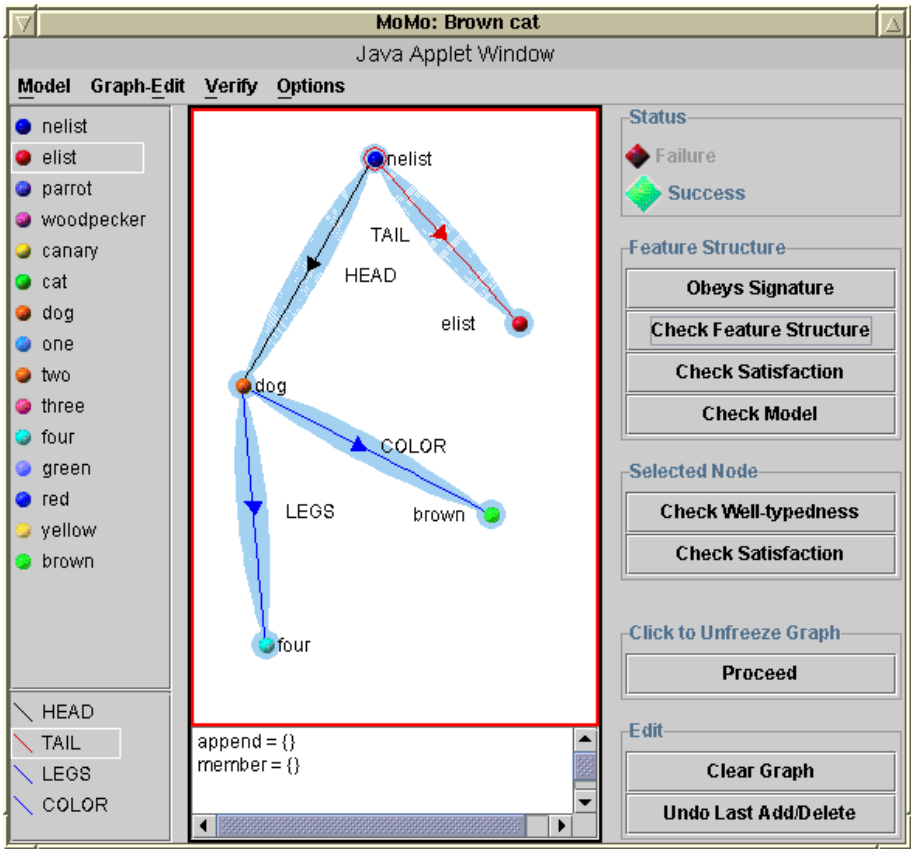


Figure 1: A screen shot of MoMo

We will first familiarize ourselves with the technical necessities and tools which the user of an implementation platform for grammars will be working with. We will also discuss the properties that an implementation platform of HPSG should have in order to be maximally useful in the sense expressed by Meurers, and we will see which fundamental limits the mathematical results on the properties of our HPSG formalism impose on computing with HPSG grammars. In developing fragments of English we will then gain practical experience with grammar implementation, and we will learn basic grammar implementation skills. The seminar will finish open-ended with an overview of a large fragment of English exemplifying the techniques of large-scale grammar development and providing the course participants with a wealth of material for studying grammar implementation on their own after the end of the course.

Structure of the Course Material Chapter 1 provides an overview of HPSG formalisms and HPSG implementation platforms, and briefly reviews the most important assumptions about grammars of human languages in the HPSG framework. The first chapter should also give readers an idea about the level of background knowledge needed

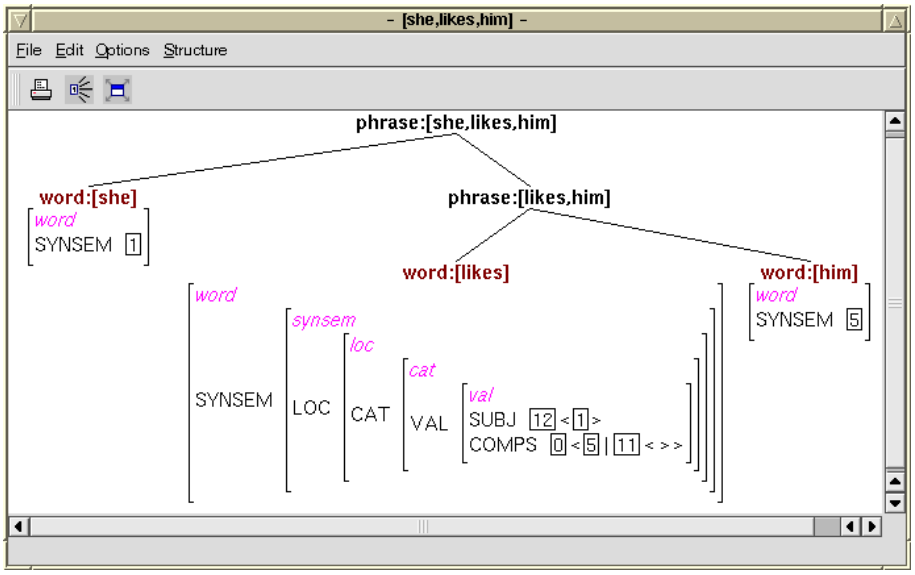


Figure 2: A screen shot of TRALE output

to work successfully through the course.

Chapter 2 is an introduction to the formal foundations of constraint-based grammar formalisms. A succession of three grammar formalisms leads from a simple feature logic to a very expressive class of formal languages, which is adequate for formalizing complete HPSG grammars. Chapter 3 investigates the relationship between grammars specified in logical languages and the needs and necessities of computational grammar implementation. The readers will learn how to use HPSG grammars as specifications of grammar implementations. Successively larger and more comprehensive grammars will explain how HPSG grammars can be used for computation.

Chapter 4 contains a symbolization of the grammar of English presented in [Pollard and Sag, 1994], in the last formalism from the sequence of formalisms in Chapter 2, which illustrates how its abstract mathematical concepts are applied in practice. This chapter offers a complete and precise specification of the grammar from the appendix of Pollard and Sag's book. This chapter might also serve as an illustration for grammar specification, as well as providing an initial overview of HPSG grammars and their formal structure. It will be useful to consult this chapter occasionally while systematically working through the materials in the textbook.

Chapter 5 and Chapter 6 provide important resources for the course. Chapter 5 contains links to the user's manuals of MoMo and TRALE, and gives online users access to the software packages of MoMo and TRALE for downloading and installing them on their computers. Together with these resources, a link to the MERGE grammar for TRALE is available. Chapter 6 contains all teaching grammars which belong to the chapter on grammar implementation, Chapter 3. The last grammar available in this chapter is once again the MERGE, which is the largest and final grammar of this course.

Document Formats The present document exists in two electronic formats; firstly, it is integrated in the ILIAS learning platform on the MiLCA server at the Seminar für Sprachwissenschaft in Tübingen as an electronic textbook (*ILIAS edition*). The ILIAS edition takes advantage of various functions provided by the ILIAS environment. In ILIAS the present course materials are seamlessly connected to an online edition of MoMo, to prepared exercise files in the file format of MoMo, to grammars and documentation resources of TRALE, and to the downloadable source code of TRALE. The TRALE resources include a large grammar of English, the MERGE. For using MoMo and TRALE, the reader should consult the respective online manuals or download them in appropriate file formats for printing. For information on how to use the functions of ILIAS, we refer the reader to the online system manual.

This textbook is also available in PDF format, which is a better choice if you prefer to read the texts offline (*PDF edition*). Of course, using the textbook offline also means not having direct access to all the interactive features of the materials. The PDF edition is, however, also linked to the same MoMo and TRALE resources as the ILIAS edition. If you read the PDF edition on a computer connected to the Internet, you are able to access the same documents by clicking on links in the text as you can access through the ILIAS edition. The only feature not available in the PDF edition besides the functionalities of the electronic learning environment themselves, is an interactive flash animation of the signature of Pollard and Sag's 1994 grammar of English, which is included in Chapter 4.1 of the ILIAS edition. An alphabetical list of the glossary entries of the ILIAS edition is contained in the PDF edition as an additional chapter. The glossary entries of each section can be accessed in the PDF edition (when used online) by clicking on the *glossary* link at the end of each section. Working with the PDF edition online is thus a workable approximation to using the ILIAS edition.

Acknowledgments Many people have contributed to the creation of this course; many of them have made such significant contributions that it is fully justified to say that without them, it would either not have happened, or the result would have been very, very different.

Beata Trawiński solved the intricate problems which had to be overcome in order to integrate the present text with all its mathematical symbols and clickable pictures on the ILIAS platform; she set up procedures for all the necessary data transformations from and into various different file formats for 'learning objects,' quite often making up for the lack of adequate tools by giving additional time and effort. Without Beata's immense contribution there would be no ILIAS edition, and without her support in teaching the first three classes with the present materials, the course could never have been developed as far as it has been. She always managed to keep the big picture in view, steering me towards the most urgent tasks when I got lost in the overwhelming plethora of everyday problems. Her ideas and suggestions permeate this textbook. Last but not least Beata translated the TRALE grammars of Section 3.1 from original teaching grammars in the ConTroll system by Detmar Meurers and Erhard Hinrichs.

Ekaterina Ovchinnikova designed and wrote MoMo, adding functionalities to it beyond

my wildest dreams. For those who already know MoMo, nothing else needs to be said. Whenever there was another crazy idea about what else MoMo could do, it did not take long until it came to exist. Katja also gave emergency support for using MoMo in class, took over the tedious and time-consuming task of maintaining the ILIAS edition after Beata had left, and supported almost every aspect of creating these course materials with creative ideas and technical solutions. Finally, Katja helped planning the conversion of the ILIAS edition of the textbook to the PDF edition, wrote the necessary xml and html code to link the PDF edition to the glossary files she had created and set it up for the complete integration with all the online resources of the course.

Obviously, there could not be a TRALE system and a MERGE grammar without the people who wrote them. W. Detmar Meurers, Gerald Penn and their students and colleagues in Columbus and Toronto did everything they could to make it happen. Needless to say, they gave permanent feedback on every aspect of this project, and their collaboration shaped everything from the original project proposal to the final result. Holger Wunsch wrote the graphical user interface of TRALE, GRISU, and made many very useful comments on the course materials. Gianina Iordăchioaia, as a teaching assistant, helped to teach the first truly web based course with these materials. Ashley Brown and Levente Barczy came for a summer internship from Stanford to Tübingen and did a fantastic job in supporting Katja's MoMo development. They implemented the first version of MoMo's note pad, and with their great enthusiasm and expertise in Java and software development brought in many new ideas and suggestions.

Lothar Lemnitzer, coordinator and driving force behind the MiLCA consortium, provided advice whenever it was needed; Erhard Hinrichs gave significant additional financial support. Manfred Sailer co-authored first versions of the fragment grammars of Section 3.2 for a class on grammar writing in HPSG, which we taught together. Adam Przepiórkowski carefully read early versions of the textbook, tested MoMo and TRALE, and gave extensive and very detailed comments which led to many improvements.

Carmella Payne and Guthrun Love were resourceful and very conscious proofreaders, who not only took care of the grammar, but also forced me in many places to reformulate and rethink unclear argumentations. I am very grateful to all who supported this project.

GLOSSARY

Chapter 1

Introduction

Abstract

In the introductory section of our seminar we want to get an overview over HPSG in general, before we go into the mathematical details of it. To gain a broader perspective, we will first look at the history of HPSG and mention the grammar frameworks that influenced it. We will also learn to distinguish between different kinds of HPSG. Then we will investigate the HPSG grammar of English that Pollard and Sag present in [Pollard and Sag, 1994]. The main purpose of our investigation will be to find out what kinds of phenomena are treated in their fragment of English, and what kind of structure their grammar has. Most importantly, we will discuss in which form the principles of grammar are stated and what kind of formal status they have.

1.1 Historical Overview

Abstract

In the historical overview we are interested in getting a general perspective on when and how HPSG developed. We will also say something about how the history of the HPSG framework is related to a few grammar implementation platforms for computing with HPSG grammars.

Before we begin the enterprise of formalizing the grammar framework of HPSG, it is necessary to survey the field in order to determine whether this is a feasible enterprise in the first place. The main question is: Is there such a thing as one single coherent HPSG framework that was created sometime in the early days of HPSG, remained unchanged until the present days, is employed by all linguists working in HPSG, and targeted by programmers who create implementation platforms for HPSG-style grammars?

As the rhetorical nature of the question already reveals, the situation is, of course, not quite as simple as in such an idealistic picture. To begin with, at least two different HPSG formalisms have been informally presented by Carl Pollard and Ivan Sag themselves:

One was presented in their first book of 1987, Information-based Syntax and Semantics [Pollard and Sag, 1987]; and the other one in their second book of 1994, Head-Driven Phrase Structure Grammar [Pollard and Sag, 1994]. The question of what linguists working in HPSG use, often tacitly and implicitly, is an entirely different matter again (and quite often it turns out to be hard to answer). Finally, the practical constraints on implementation platforms and particular reasons for creating them as well as their purpose might even lead to computational systems which, formally speaking, have very little to do with any of the two HPSG formalisms informally described by Pollard and Sag, although their appearance on the surface might be close enough to let the user implement grammars *in the style of* HPSG, whatever that may turn out to mean when investigated with mathematical rigor.

Under these circumstances, how can we formalize the linguistic framework of HPSG at all? Which HPSG is the right HPSG that we should consider? Which version of the HPSG framework is relevant? A short overview of the history of HPSG indicates what the answers to these questions are.

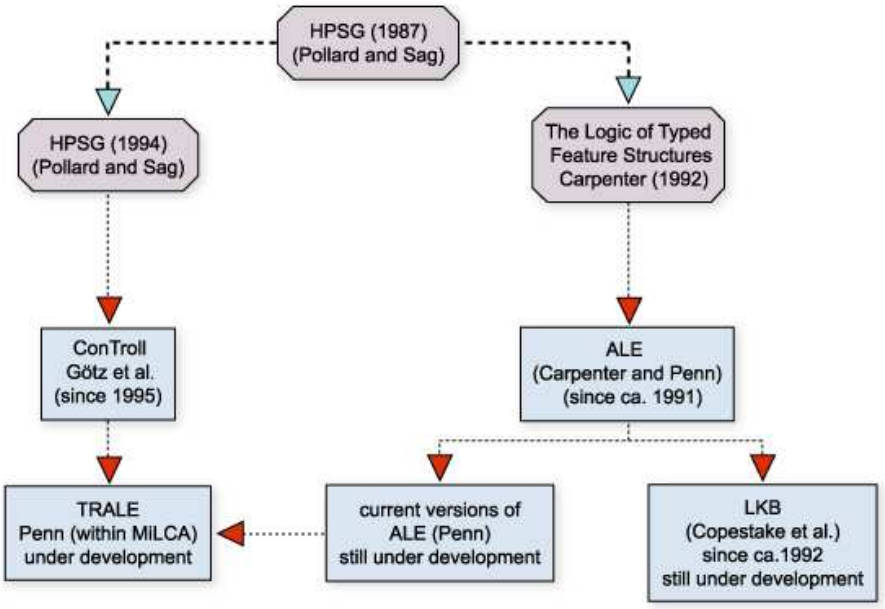
HPSG began in the mid 1980s as an eclectic linguistic framework that was inspired by several other frameworks. Among its spiritual ancestors we find the then current versions of the Chomskian paradigm, Government and Binding theory, often referred to as the GB framework [Chomsky, 1981]. Since many linguists were working in that framework at the time, it was important for a new framework to show that it could at least handle the data that were discussed in the GB literature. Some of the ideas of the combinatorial system of HPSG were inspired by Categorial Grammar; and the ideas about using feature structures besides something akin to phrase structure were inspired by Lexical Functional Grammar (LFG, [Bresnan, 1982]) and Generalized Phrase Structure Grammar (GPSG, [Gazdar et al., 1985]). GPSG in particular also contributed a significant number of analytical ideas, with GPSG's slash percolation mechanism for *unbounded dependencies* being only the most prominent example. GPSG's relatively strong mathematical rigor was certainly another target for the creators of HPSG.

The first incarnation of HPSG was published in [Pollard and Sag, 1987]. Henceforth we will refer to the framework presented in that book as HPSG 87. HPSG 87 is a typical instance of a unification-based or information-based grammar formalism. The basic underlying intuition is that linguists specify pieces of partial information about language in their grammars. All the pieces of partial information about a language that are specified in a grammar are then combined by algebraic operations such as unification to obtain all available information about the language of a mature speaker of that language.

In their second HPSG book [Pollard and Sag, 1994], published in 1994, Pollard and Sag outline a radically different architecture of grammar: HPSG 94, as we will call it in our overview, can be called an object-based grammar formalism, or, as Carl Pollard prefers to call it, a constraint-based grammar formalism. HPSG 94 is not concerned with the specification of pieces of partial information. Instead, it envisions an architecture in which linguists use a logical language in order to specify language as a collection of total objects. HPSG 94 can thus be said to proceed model theoretically.

The following figure shows how the development of formalisms for HPSG branches out

after HPSG 87. Besides HPSG 94 with its new, model theoretic foundations, there is a second tradition that pursues and refines the formalism of 1987:



It is necessary to know about the radical change of the underlying formalism between 1987 and 1994 to be able to appreciate the nature of current implementation platforms for HPSG. As the figure above shows, in the second branch of developments after 1987 (besides Pollard and Sag's switch from unification-based to constraint-based HPSG), there is a book that Bob Carpenter published in 1992 under the title *The Logic of Typed Feature Structures* [Carpenter, 1992]. In this book Carpenter presents a class of logics that can be viewed as a refinement of the unification-based formalism of [Pollard and Sag, 1987]. In other words, Carpenter's logics support a unification-based variant of HPSG. This is particularly important, since Carpenter's book laid the formal foundations to the creation of a number of implementation platforms for HPSG. In the figure above we mention only two very influential implementation platforms, ALE and LKB. ALE was initially co-developed by Bob Carpenter and Gerald Penn, but fairly soon Gerald Penn became the main developer of ALE, which he still is today. LKB was originally conceived by Ann Copestake and then further developed with many collaborators. Because ALE and LKB follow Carpenter's theoretical ideas, they adopt the partial-information paradigm of HPSG 87. With them, this brand of HPSG extends to the present and still exerts significant influence, although theoretical HPSG has moved away from the unification framework.

There is another tradition of implementation platforms for HPSG grammars which took HPSG 94 as its point of reference. The first of those that we want to mention here is the ConTroll system, whose chief developer was Thilo Götz, who worked in a team with many collaborators. ConTroll was an experimental system. Its development ended roughly in 1997, with upgrades to new versions of software and operating systems still being done. The idea of ConTroll was to try to get as close to HPSG 94 as possible but not to use any

computational means not inherent in that formalism. Its successor, the TRALE system, inherited the idea to be as faithful to HPSG 94 as possible and to combine that with all means of efficient processing that are available under those premises. With Gerald Penn being the chief developer of TRALE, TRALE inherited the core of the ALE system, but with the underlying logic specialized to the case where it becomes a logic in the tradition of HPSG 94. Later in the course, when we implement grammars in TRALE, we will learn many more details about the system, and we will investigate how considerations of efficient parsing lead to the introduction of various techniques that are not inherent in the HPSG 94 formalism per se, but maintain a transparent relationship between the specification of a grammar in the formalism of HPSG 94 and its implementation in TRALE. However, for reasons of tractability, the formal language of TRALE does not contain all the syntactic and semantic constructs of HPSG 94. When we specify HPSG 94 and then compare it with TRALE, we will see exactly where the differences are, and we will discuss their consequences for grammar implementation.

In our brief overview, we saw that there is more than one formalism for HPSG, and more than one formalism is used in ongoing work on HPSG. In this course, we follow the newer, constraint-based version of HPSG rather than Pollard and Sag's first proposal of 1987 of unification-based formal foundations. With that decision, we adopt the version of HPSG that is nowadays most prevalent in theoretical linguistic work. At the same time, we must keep in mind that there are current implementation platforms for HPSG grammars that follow the old paradigm.

Recommended Reading

In the introduction of [Pollard and Sag, 1994, p. 1–6] you find more information on which other grammar frameworks directly influenced HPSG 94.

GLOSSARY

1.2 The Structure of HPSG Grammars

Abstract

We will give a brief summary of the architecture of grammar that Pollard and Sag proposed in their HPSG book in 1994. We will introduce the most important keywords and concepts that we will formalize in Section 2, Grammar Formalisms, of our course.

[Pollard and Sag, 1994] does not present a formalism of HPSG. The reason they give for this lack of explicitness is that in their book they want to keep the technical tools in the background, and instead focus on the linguistic theories that can be expressed within their framework. However, they give a rather precise outline of the formalism they envision and they refer to the relevant underlying logical literature at the time of writing. If we consider the following three sources we will obtain sufficient information to decide whether a given

formalism is a good candidate for a mathematical rendering of Pollard and Sag's ideas: Citations in Pollard and Sag's book from various articles and other books; Pollard and Sag's detailed description of the formalism; and the appendix of their book, whose concise statement of their grammar of English is particularly helpful. In this section we want to introduce the most relevant parts of Pollard and Sag's terminology and concepts. This gives us an initial idea about what we will have to formalize later in the first part of our course.

Pollard and Sag distinguish two levels of their grammatical architecture. First of all, there is a level of *linguistic description*. This is the level at which linguists state their observations and generalizations about human language(s). Traditionally these observations would be phrased in natural language itself, being a collection of statements about the syntax, phonology, morphology or semantics of a particular language or of language in general. A familiar example of a grammar of that kind is the Duden grammar of German. In addition, Pollard and Sag introduce a second level, a level of meaning of grammars. Since they have a formal language in mind in which they want to state their observations, they are in a position to envision a mathematically exact interpretation of their formalized statements. As the meaning of their statements they envision collections of so-called *feature structures*.

What do the collections of feature structures correspond to when we compare an HPSG grammar of English to the Duden grammar of German? It is clear that the interpreting structures of an HPSG grammar are supposed to give us the denotation or meaning of the grammar. But the informal meaning of the Duden grammar is certainly the German language. In other words, the the purpose of the Duden grammar is to give us a description of German. The problem with the Duden grammar is of course that it only gives us informal, general rules, written in natural language. It is therefore often hard or impossible to pinpoint the exact predictions of that theory of German. In HPSG, on the other hand, a precise definition will be given of formal description languages and of the meaning of our principles and observations (written in those description languages) in terms of collections of feature structures. We thus obtain precise predictions of our theories of language, and we can falsify our predictions by looking at the feature structure models that are admitted by them. For this idea of the architecture of grammar to work we have to assume that the feature structure models are mathematical models of the language that we want to describe. The feature structures are in some sense mathematical idealizations of expressions of natural language. For example, the feature structures in the denotation of Pollard and Sag's grammar are mathematical idealizations of corresponding expressions in the English language.

Summarizing what we have said so far, HPSG distinguishes two levels of linguistic theories, the level of description and the level of denotation. HPSG grammars consist of descriptions of (or generalizations about) natural languages. The natural languages are modeled in terms of mathematical structures called feature structures, which are viewed as idealizations of linguistic expressions.

A closer look at the structure of HPSG grammars reveals that there are a number of notions which we have to explain to understand what a grammar is. The appendix of

Pollard and Sag consists of two main parts (there are two more, which we will ignore for the moment – we will come back to them in the next section): The first, A.1, is called *The Sort Hierarchy* and comprises two subsections, *Partitions* and *Feature Declarations*. The second part, A.2, is called *The Principles*, and contains statements about the English language. Those two parts, essentially, make up their *grammar* of English. The Sections *The Sort Hierarchy* and *The Principles* of the present textbook quote (and elaborate) these two parts of Pollard and Sag's appendix. We can observe that what is introduced in A.1 are the symbols or syntactic means that the statements in part A.2 use to express generalizations about English: The *sort* symbols of *Partitions* and the *attribute* symbols of *Feature Declarations* reoccur throughout the statements in the *Principles* section. One of our first tasks will be to determine what the sort hierarchy and the feature declarations are from a mathematical point of view, and how the relevant mathematical constructs enter into the theory of English.

This still leaves the theory of the modeling domain, the *feature structures*, to be discussed. We are given some hints on what these feature structures look like in the introductory notes of the appendix (p. 396) and in various sections of this book. From these remarks we will have to reconstruct the relationship of feature structures to the sort hierarchy and the feature declarations. The reconstruction of that relationship will finally lead us to a definition of the meaning of formally defined well-formed descriptions of the description language(s) of HPSG in the domain of *totally well-typed* and *sort-resolved* feature structures. For this to make sense, we will of course also have to explain what it means for a feature structure to be *totally well-typed* and *sort-resolved*.

To summarize, our main task of the formal part of our class will be to provide a mathematical reconstruction of the notion of grammar of HPSG. We will have to define what partitions and feature declarations are, and how we can use these to write descriptions. This will allow us to fully formalize the principles of grammar that Pollard and Sag state in natural language in the appendix of their book. In Section 4 of our course materials, you can already get an impression about what the principles will look like in some formal language (we will, however, use a formal language that is slightly different from the one that you see in Section 4, and we will explain why and what the relationship between the two is). With these definitions in hand, we will know precisely what an HPSG grammar is. In order to understand what it means in the eyes of Pollard and Sag, we will also have to formalize the modeling domain of HPSG grammars. For that purpose, we have to give a definition of totally well-typed, sort-resolved feature structures, and it is necessary to define an interpretation function that assigns meaning to expressions of our description language in the domain of feature structures.

Before we move on to that, however, we will first have a closer look at the principles of the grammar of English of Pollard and Sag. Our goal is to get a feeling for what kind of linguistic phenomena they describe in their grammar, and how they phrase their observations of English.

Recommended Reading

Along with this section, you should read pp. 6–14, *The Nature of Linguistic Theory*, and take a look at the Appendix, pp. 395–403, of [Pollard and Sag, 1994].

GLOSSARY

1.3 The Grammar of English of Pollard and Sag 1994

Abstract

We will give an overview over the principles and the empirical coverage of the grammar of Pollard and Sag 1994. Rather than trying to go into details, we want to give the reader an intuitive sense of what their grammar is like.

In Section 1.2, *The Structure of HPSG Grammars*, we focused on HPSG as a formalism for linguistic theories. By contrast, in the current section we are interested in HPSG as a formal linguistic theory. As our example we take Pollard and Sag's grammar of English. It is a concrete example of how linguistic theorizing can be formulated within the confines of the formal assumptions outlined in the last section.

HPSG as a linguistic theory has been attributed a number of characteristic properties. Usually it is perceived as a member of the family of linguistic frameworks that are called *generative* grammars. In this context, *generative* does not mean a grammar that describes a language as the set of strings that can be derived by applying phrase structure rules to a given start symbol, but a grammar which is formalized to a high degree and thus makes exact predictions about the collection of objects that are considered to belong to a language. In contrast to the original theories of generative transformational grammar of the 1950s and 1960s, HPSG does not have phrase structure rules and transformations that are applied to phrase markers that are constructed by the successive application of phrase structure rules to a start symbol. Instead, it is *declarative*, *non-derivational* and *constraint-based*. Sets of constraints which hold simultaneously determine the collections of admissible linguistic structures without defining an order of the derivation or generation of signs. In its choice of tools, HPSG is decidedly *eclectic*. It uses insights of various other linguistic frameworks and analyses and reconstructs them in its own specific way. We have mentioned GB theory, LFG, and Categorial Grammar already in Section 1.1, *Historical Overview*.

HPSG is often called a *lexicalist* framework. This means that HPSG stresses the importance of lexical descriptions and generalizations over the lexicon of natural languages. It has developed various techniques of stating generalizations over the words of a language, and it postulates a rich, fine-grained structure of words, which allows elaborate descriptions of the properties of lexical signs. Much of the explicit development of these techniques followed only after the publication of [Pollard and Sag, 1994], and surprisingly, the book discusses little about how to integrate the lexicon with its theory of English. What is evident is an elaborate structure of words, which tended to increase in complexity in subsequent work.

Pollard and Sag use that structure in particular for their largely lexical analysis of English case assignment, in their analysis of subject verb agreement in Chapter 2, and in their analysis of raising and control verbs in Chapter 3.

The concept of *signs*, which are divided into *words* and *phrases*, is central to many HPSG analyses of linguistic phenomena. Words form the major building blocks of phrases, which are signs that recursively contain additional words or phrases as their components. Signs are central in HPSG, since they are considered to be those elements of a language which are empirically observable and whose linguistic properties can be described by the various sub-disciplines of linguistics. This brings us to another very important aspect: HPSG was designed as a grammar framework that can integrate theories of different aspects of linguistic expressions. In that sense, it is designed as a framework for the *comprehensive* description of language. By now, HPSG theories are based in the various subjects of the phonology, morphology, syntax, semantics, pragmatics, and even certain discourse phenomena of linguistic expressions. However, first and foremost HPSG has been applied in the area of syntax, which is the aspect of English that Pollard and Sag describe in their book, although they comment briefly on the areas of semantics and pragmatics. Although syntax was initially the main area of application of HPSG, the original architecture of signs already clearly indicates the interfaces to a broader theory of language. This is illustrated in Figure 1.1 with a description of the English personal pronoun *she*.

Linguistic structure concerning different aspects of signs is distributed under certain attributes. Clearly the representation of the phonology of the pronoun under the attribute PHON as an orthographic string is a gross oversimplification. The immediate justification for this simplistic initial approach is that the given structure can and should be replaced by a much more elaborate representation that can take phonological theories into account. A theory of that kind was later developed in [Höhle, 1999] (and elsewhere). In much the same sense the grammar of English of Pollard and Sag leaves much of pragmatics, all of morphology and considerable parts of semantics unspecified, the idea being that these aspects of language can be incrementally added to the syntactic structures that they do specify.

The attribute SYNSEM unites syntactic and semantic aspects of signs, which are in turn divided in local and non-local properties, as distinguished by two more attributes. For Pollard and Sag, non-local properties of signs are those that are typically associated with theories of *unbounded dependency constructions*, illustrated by the following sentences:

(1) a. Mary Peter adores *t*.

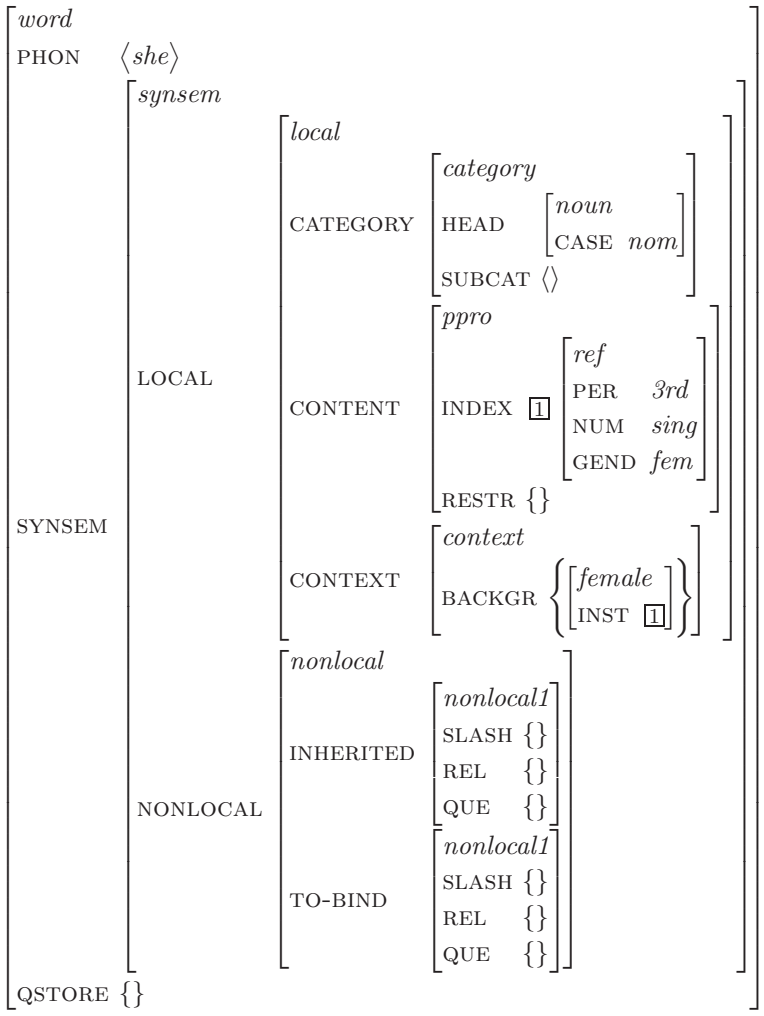
b. I wonder who Mary saw *t* last night.

c. This is the actor who Mary wanted to see *t* all her life.

d. Mary is easy to please *t*.

The sentences (1a)–(1d) are of course instances of phrases, formed from other phrases, which are ultimately composed of words.

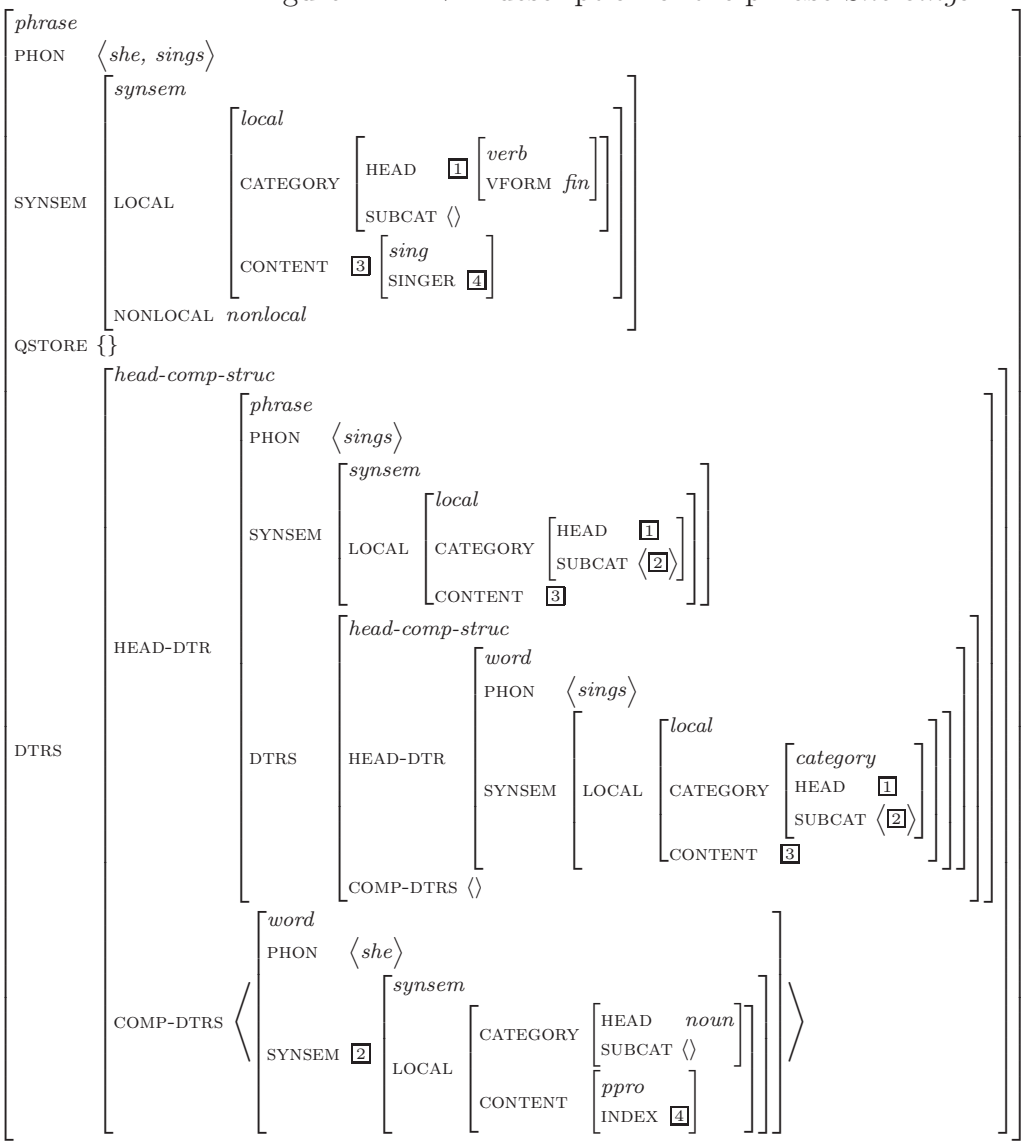
Figure 1.1: AVM description of the pronoun *she*



For a more explicit illustration, consider the description of the English phrase *She sings* in Figure 1.2.

We can see that the phrase has an attribute DTRS (short for DAUGHTERS). Under that attribute, we find two more attributes HEAD-DTR (for the syntactic head of the phrase) and COMP-DTRS (for the syntactic complement daughters of the phrase). In our example, a projection of the verb *sings* is the syntactic head of the construction, and *she* is its only complement daughter. As a consequence, the list of complement daughters is of length one. Note also that the constituent structure as encoded by the two syntactic daughters of the phrase does not by itself determine the word order of the sentence. The only place in the description that tells us about the relative order of *sings* and *she* in the overall phrase is the order in which the phonologies of the two words are stated on the PHON list of the overall phrase.

Figure 1.2: AVM description of the phrase *She sings*



Let us briefly return to our first description, the description of the word *she*. The local properties of signs, which are mentioned under the attribute LOCAL, comprise contextual aspects (under the attribute CONTEXT), their semantic properties (under the attribute CONTENT) and properties such as their syntactic category (under the attribute CATEGORY). The pronoun *she* considered here is, of course, a *noun*, and nouns bear case, which is *nominative* in this instance. If you take a closer look at the description of the phrase *She sings*, you may discover that the syntactic category of *verbs* has a different attribute, called VFORM, whose value is *finite* for the verb form of *sings*.

Another important feature of HPSG that can be seen in the description above is the idea of structural identities, depicted in the form of *tags*, which are boxed numbers. For

example, we learn from the description that the single element on the SUBCAT list of *sings* is identical to the SYNSEM value of the complement daughter of the overall phrase, *she*. In the grammar, this identity is enforced by the interaction of the SUBCATEGORIZATION PRINCIPLE and the way syntactic structures are formed. The SUBCATEGORIZATION PRINCIPLE says essentially that the SYNSEM values of complement daughters must be identical with corresponding elements of the SUBCAT list of the syntactic head, and the SUBCAT list of the mother phrase is shortened relative to the one of the head daughter by exactly those elements which are realized as syntactic complement daughters of the phrase. In our case that means that the SUBCAT list of the overall phrase is empty.

Moreover, every phrase is licensed by an ID SCHEMA: If a phrase is a headed phrase, it must be described by one out of six ID SCHEMATA. Given the available ID SCHEMATA of Pollard and Sag's grammar, we can determine that *sings* first has to project to a phrase and can then combine with its subject to form the phrase *She sings*.

Pollard and Sag's grammar also provides an analysis of relative clauses, it introduces a simple theory of semantic structures and quantifier scope, and it develops a complex BINDING THEORY. BINDING THEORY is concerned with the distribution of certain pronouns, reflexive and non-reflexive, as illustrated by the following sentences:

(2) a. She$_i$ likes herself$_i$.

b. She$_i$ likes her$_k$. (with i is not equal to k)

BINDING THEORY predicts that in (2a), *she* and *herself* must refer to the same entity in the world. They are said to be coindexed, as indicated by the subscript i. By contrast *she* and *her* in sentence (2b) must not be coindexed, and, under *normal* circumstances, they cannot refer to the same entity in the world. Formally the principles of the BINDING THEORY belong to the most complex principles that Pollard and Sag formulate in their grammar.

We have given a very short overview of the phenomena that are treated in Pollard and Sag's grammar of English. Before we look at their principles of grammar in more detail, we will turn to the formalization of HPSG. With an increasingly expressive formal language in hand, we will come back to some of the principles of their grammar and investigate what it is exactly that they say.

Exercises

The following exercises are meant to encourage you to go and investigate the form and structure of Pollard and Sag's grammar of English. You should not feel obliged to study Pollard and Sag's theory of unbounded dependency constructions in depth. The aim of the exercises is to get an overview and an intuitive understanding of how their HPSG grammar works, taking an important module of any formal syntactic theory as a representative example.

Exercise 1 *Which constituents of the sentences (3)–(6) are involved in the unbounded dependency construction of each sentence?*

(3) Mary Peter adores t.

(4) I wonder who Mary saw t last night.

(5) This is the actor who Mary wanted to see t all her life.

(6) Mary is easy to please t.

Exercise 2 *Which principles and components of Pollard and Sag's grammar are involved in licensing the unbounded dependency aspects of the sentences (3)–(6)? Name them, and briefly characterize the function they have in the analysis of the sentences (keywords will suffice).*

GLOSSARY

Chapter 2

Grammar Formalisms

Abstract

In this section we will investigate the formal foundations of constraint-based HPSG. Starting with a simple feature logic formalism, we will develop a series of increasingly complex formalisms for formalizing constraint-based grammars. Our most important tool for understanding these formalisms will be MoMo, which we will use to study the syntax of our languages, the well-formedness of feature structures, the notion of constraint satisfaction and the modeling relation between (sets of) descriptions and (sets of) feature structures.

2.1 Introduction

Abstract

This section gives a brief overview of the course section on grammar formalisms.

The course section on grammar formalisms proceeds in four steps. First we will work out a basic syntax for the description of natural languages, we will motivate its design, and we will discuss the basic ingredients of a syntax for HPSG. MoMo will help us to get an informal understanding of the underlying issues before we give the first formal definitions.

With a basic syntax in hand, we will then enter into a sequence of three increasingly more powerful formalisms for writing grammars. The first of them takes the syntax of our introductory discussion and augments it with a semantics of standard feature logics. Observations about missing concepts relative to standard HPSG principles then motivate extensions of the syntax and semantics. To make these extensions as digestible as possible, they are broken down into two parts: In the first stage, our extension is mainly an extension of syntax, with the interpreting structures staying the same. In the second stage, the syntactic extension is very simple, but there is a non-trivial extension concerning the interpreting structures.

Altogether we will see three increasingly expressive grammar formalisms of the family of constraint-based grammars. Whereas the simplest one is close to standard languages

of feature logics, the more complex ones are specifically geared towards formalizing HPSG grammars in the style of [Pollard and Sag, 1994].

GLOSSARY

2.1.1 An Initial Syntax

ABSTRACT

In this section, we discuss basic design issues of the syntax of the formal languages that we will want to use for notating the principles of grammars. We introduce an initial, simple syntax by giving examples.

In constraint-based grammar we need a formal language that is suitable to talk about a domain of objects. A statement in the formal description language is a statement about what the objects in the domain look like. Formalisms that provide that kind of formal languages are sometimes called *logics of description*. The idea is that expressions of these logics do not denote True or False when interpreted over a domain of objects as they do in First Order Predicate Logic. Instead expressions are thought of as denoting sets of objects. At first glance, the difference seems very subtle. Technically speaking, it has some important ramifications, which we can put aside for now.

At the moment it is sufficient to keep in mind that we want to formulate generalizations about some empirical domain that denote sets of objects. Our ultimate sets of objects will be natural language expressions. For the sake of simplicity and concreteness, we may pick any other appropriate empirical (or ontological) domain to illustrate our enterprise. In the beginning we will work with a small zoo: We will talk about pets like dogs and cats, and about various birds such as parrots, canaries, and woodpeckers. They will be distinguished by color and their number of legs, and we will order them on lists.

What we need to be able to say with our formal language can be extracted from the principles of grammar of the HPSG book and from the numerous descriptions of linguistic objects that we find throughout the HPSG literature. From the principles of grammar that we have seen we know that we need to be able to say that objects are of some sort. We want to be able to say that an object is a sign, a phrase, or a word. In addition, we want to say that objects have certain attributes, which have certain objects as values. And again we might want to say that those attribute values are of a certain sort and might have certain attributes.

The *antecedent* of the HEAD FEATURE PRINCIPLE in Section 4.2, *The Principles*, provides an example for these observations:

(7) $\begin{bmatrix} \textit{phrase} \\ \text{DTRS} \;\; \textit{headed-struc} \end{bmatrix}$

The intended interpretation of this AVM is that it denotes the set of all objects such that they are of sort *phrase*, they have an attribute DTRS (which is just an abbreviation for

the full name of the attribute, DAUGHTERS), and the value of that attribute is an object of sort *headed-struc* (short for *headed-structure*).

Even this small example is already a bit more complex than what we have made explicit so far, not to mention the complete structure of the entire HEAD FEATURE PRINCIPLE. We haven't really said in which way we want to notate complex statements where we mention several properties of an object such as it being a phrase *and* having a DTRS attribute with a certain value. Of course, the AVM in the example above presents one possible solution by arranging the two statements in a matrix between square brackets. It is easy to see how we can extend such a matrix to make any finite number of statements by inserting as many statements as we need. A concrete example of how that works is provided by our description of the phrase *She sings* in Figure 1.2 of Section 1.3, *The Grammar of English of Pollard and Sag 1994*, where we also see that matrices can be embedded into each other. However, the HEAD FEATURE PRINCIPLE shows that there are other logical connectives like *implication* in our target description language which are not as easily integrated in a *graphical* matrix notation like the logical connective *AND*. In the common notation of AVM matrices, the symbol for logical implication, $\rightarrow$, is placed between two AVM matrices in a way that is reminiscent of the infix notation for implication in First Order Logic.

A second look at logical connectives in the principles of grammar of Section 4.2, *The Principles*, shows that our initial observation about implication generalizes to other logical connectives. We also find *disjunction* (symbolized as $\vee$) and *equivalence* ($\leftrightarrow$) as well as negation. A simple example for disjunction is the consequent of the ID PRINCIPLE, which says that for an object with attribute DTRS and DTRS values of sort *headed-struc*, it must also be the case that the object is described by one of six ID SCHEMATA. A simple example of negation can be found in the HEAD SUBJECT SCHEMA, which says, among other things, that either the value of SYNSEM LOC CAT HEAD INV is an object of sort *minus* or the value of SYNSEM LOC CAT HEAD is not of sort *verb*, i.e., the description that it is a verb is not true of the object.

Obviously, the AVM notation of Section *The Principles* is one possible notation for the originally informally stated principles of grammar of Pollard and Sag. Due to its two-dimensional representation of conjunction in matrices the AVM notation is fairly readable, and it is close to a notation that linguists are used to. However, the readability of the notation comes at a price. Typesetting AVMs using a standard computer keyboard is non-trivial and requires a relatively high number of conventions. For very much the same reasons, a recursive definition of a language of AVM matrices needs a relatively high number of base cases and a fairly complex structure in order to achieve the kind of flexible notation that linguists use. Of course, if we take that route, we will have to mirror this complexity in the number of clauses with which we define the interpretation function of the expressions of the description language. The resulting formal language would be considerably more complex than appropriate for an introductory study of a formalism for constraint-based grammar.

On the other end of the scale we could design formal languages that are mathematically elegant, have a completely non-redundant definition of base cases and a straightforward recursive structure, only use standard symbols from computer keyboards and have a linear

notation that is easy to typeset. However, this mathematical elegance also comes at a price. In this case, the price is readability. Whereas the resulting syntactic structure of the language is very simple and, in that sense, easy to study, languages of that kind turn out to be very hard to read for the human user once the descriptions become as big as in our example of the English phrase *She sings* in Figure 1.2.

For an introduction to the topic, it seems to be reasonable to take a middle road that takes into account that our ultimate goal is the implementation of grammars in a particular computational system, namely in TRALE. In short, our syntax should be a syntax to talk about objects in terms of their sorts and the attributes that they might bear; for readability and familiarity the syntax should resemble the AVM notation of the linguistic literature as much as possible while avoiding the typesetting problems of AVM languages and an unnecessarily complex and highly redundant recursive structure; it should lead us towards the description language of TRALE; and it should only use symbols from standard computer keyboards. In particular, while the symbols $\wedge$, $\vee$, $\rightarrow$ and $\neg$ are graphically very satisfying representations of logical connectives, they should be replaced by symbols taken from standard keyboards. The similarity of the layout of our formal languages to the common AVM matrices of linguistics should allow us to get a firm grasp of how those two notations are related and to be able to switch between the two whenever it is convenient for us. That means that for our specifications we will use our own syntax, but for linguistic examples from the literature we may choose to refer to AVMs.

Let us introduce our initial syntax by way of simple examples. In the following list, you see pairs of expressions in a language of AVM matrices and of our syntax. We will refer to our syntax as MoMo syntax.

AVM expressions	MoMo expressions
sort	`sort`
[*sort*]	`(sort)`
[ATTRIBUTE *sort*]	`attribute:sort`
[*sort1* ATTRIBUTE *sort2*]	`(sort1, attribute:sort2)`
[*sort1* ATTRIBUTE1 *sort2* ATTRIBUTE2 [ATTRIBUTE3 *sort3*]]	`(sort1, attribute1:sort2,` `attribute2:attribute3:sort3)`
sort1 $\vee$ [ATTRIBUTE *sort2*]	`(sort1; attribute:sort2)`
[ATTRIBUTE $\neg$ *sort*]	`attribute:~sort`
sort1 $\rightarrow$ [ATTRIBUTE1 [*sort2* ATTRIBUTE2 *sort3*]]	`sort1 *>` `(attribute1:(sort2,attribute2:sort3))`
sort1 $\leftrightarrow$ [ATTRIBUTE1 [*sort2* ATTRIBUTE2 *sort3*]]	`sort1 <*>` `(attribute1:(sort2,attribute2:sort3))`
[ATTRIBUTE *sort1*] $\rightarrow$ ($\neg$ *sort1* $\wedge$ $\neg$ *sort2*)	`attribute:sort1 *> (~sort1,~sort2)`

As can be inferred from these examples, we represent conjunction as ‘,’, disjunction as ‘;’, implication as ‘*>’, bi-implication as ‘<*>’, and negation as ‘∼’. We use colon, ‘:’, to separate attributes (and sorts) in sequences of attributes (and sorts). In summation, we now have a syntax using a set of attribute symbols and a set of sort symbols, which have to be declared by the user, and a small number of logical connectives, colon, and brackets to state our generalizations about some domain of objects. For each pair of a set of sorts and attribute symbols, we get a different language, but they all use the same logical symbols.

From the above discussion of the syntax that we want to use to formalize HPSG, we learn an important lesson about logical languages in general. Each formal language has alternatives for a given formalism: For one and the same formalism, we can design different kinds of logical languages. Design decisions are made on the basis of theoretical goals and of practical requirements, and are strongly influenced by the purpose that one has in mind for the formal language. For our feature logic formalism of HPSG, we might want to use a different notation when we intend to prove new theorems, when we want to use it in a linguistics paper, and when we specify a grammar for implementation in TRALE. As a consequence, we should not be confused by differences in *look and feel* of the syntax in which statements of a theory are made. Ultimately what counts is the semantics that we assign to the syntax. This is probably a good point to remember that there are many different ways to notate expressions of First Order Predicate Logic, although we might nowadays be used to one particular standard notation.

In our course we will use two notations. For illustrating examples, we will occasionally use an AVM notation, because it is the most readable notation, and we will stick to conventions that will always allow us to translate the examples into MoMo syntax in our minds. When the emphasis is on rigorous specifications, we use MoMo and TRALE, and for online communication we use MoMo syntax.

Before we conclude this section, we should make a few remarks about terminology. Readers that are familiar with implementation platforms for HPSG from the unification-based tradition, and have seen papers influenced by that tradition, might have noticed that we consistently use the word *sort* where they use the term *type*. This might be particularly obvious in the phrase *sort hierarchy* vs. *type hierarchy*. The reason for the difference lies in the different foundational sources of the two terminologies: Unification-based HPSG started out with the term *type* and kept it. Constraint-based HPSG makes a distinction between *sorts*, which replace the original *types*, and the term *type*, which is now used for a different philosophical concept that has nothing to do with sorts and sort hierarchies. We will briefly discuss the use of the term *type* in constraint-based HPSG when we introduce a semantics for our formal languages. An even simpler remark concerns the attributes. Occasionally, they are called *features* in the literature. The two expressions are simply used as synonyms.

In the next section we will formalize our initial MoMo syntax, and we will add Pollard and Sag’s *partitions* and *feature declarations* to the picture. This will put us in a position to use MoMo to explore the properties of our formal languages and their intended interpretation in a domain of feature structures.

GLOSSARY

2.1.2 Signatures: Partitions and Feature Declarations

Abstract

We will make our informal notion of the initial syntax of description languages explicit by giving an inductive definition. The definition leads to a discussion of the sets of sorts and attributes that our languages depend on, and we will discuss how Pollard and Sag's partitions *and* feature declarations *introduce additional structure to our formalism.*

In the previous section we have motivated the design of our initial description language, and we introduced it by looking at some examples. While introducing a notation by example may actually be enough to work reasonably well with it, it does not suffice for our purpose of defining formalisms for constraint-based grammar. To be emphatically precise about our syntax, we need an inductive definition of our languages.

The following definition makes the languages that we have introduced by example fully explicit:

Definition 1 *For each set* $\mathcal{G}$ *of sorts, for each set* $\mathcal{A}$ *of attributes, for each* $\Sigma = \langle \mathcal{G}, \mathcal{A} \rangle$, $\mathcal{D}^\Sigma$ *is the smallest set such that*

for each $\sigma \in \mathcal{G}$,

$\sigma \in \mathcal{D}^\Sigma$,

for each $\alpha \in \mathcal{A}$, *for each* $\delta \in \mathcal{D}^\Sigma$,

$\alpha : \delta \ \in \mathcal{D}^\Sigma$,

for each $\delta \in \mathcal{D}^\Sigma$, $\sim \delta \ \in \mathcal{D}^\Sigma$,

for each $\delta_1 \in \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathcal{D}^\Sigma$, $(\delta_1, \delta_2) \in \mathcal{D}^\Sigma$,

for each $\delta_1 \in \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathcal{D}^\Sigma$, $(\delta_1; \delta_2) \in \mathcal{D}^\Sigma$,

for each $\delta_1 \in \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathcal{D}^\Sigma$, $(\delta_1 \texttt{*>} \delta_2) \in \mathcal{D}^\Sigma$, *and*

for each $\delta_1 \in \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathcal{D}^\Sigma$, $(\delta_1 \texttt{<*>} \delta_2) \in \mathcal{D}^\Sigma$.

As usual, we use parentheses liberally and omit them where no confusion can arise. In the case of missing parentheses, we assume the following operator precedence (in decreasing order): '`:`', '$\sim$', '`,`', '`;`', '`*>`', '`<*>`'. That means that the colon, '`:`', binds more tightly than the tilde, '$\sim$', (for negation), and so on. These conventions reflect the conventions for First Order Predicate Logic.

In the above definition we can plainly see that we do not introduce just one formal language, but an entire collection of formal languages: Each formal language depends on the choice of a set of sort symbols, $\mathcal{G}$, and a set of attribute symbols, $\mathcal{A}$, and for each chosen pair we get a different description language. In logic, the set of symbols from

which a language is constructed is called an *alphabet*. We call the tuples $\Sigma = \langle \mathcal{G}, \mathcal{A} \rangle$ the *signatures* of the languages. The tuples consist of sets of sort symbols and sets of attribute symbols, which are the non-logical symbols of the respective description language. When we compare our terminology to the situation in First Order Logic as introduced in textbooks (e.g., [Ebbinghaus et al., 1992]), our signature plus the logical symbols and brackets of the above definition (i.e. the set of symbols $\{(,), :, \sim, ,, ;, *>, <*>\}$) corresponds to the alphabet of a first order language. The symbols in our signature correspond to the n-ary relation symbols, n-ary function symbols, and the set of constants of a first order language, i.e., to its non-logical symbols.

HPSG chooses more complex signatures than our initial signatures, Σ. The additional structure on the sort symbols and attribute symbols comes from the *partitions* and the *feature declarations*. The idea of partitions is to impose some order or hierarchy on the sort symbols. This hierarchy of the sorts will be used in a certain way when we interpret expressions of the description language. Similarly the feature declarations introduce a relationship between sorts and attributes by declaring attributes appropriate to certain sorts and demanding certain sorts as values for particular attributes. Again, these specifications will become relevant when we interpret our signatures.

This is best explained by a simple and intuitive example. In Figure 2.1, we see a complete signature specification. Appealing to a number of notational conventions, the figure actually specifies a set of sorts, a set of attributes, a sort hierarchy (Pollard and Sag's *partition*) and appropriateness conditions (Pollard and Sag's *feature declarations*). When we talk about them, we will stick to the common conventions of notating sort symbols in *italics* and attribute symbols in CAPITALS. In our signature declaration of the figure below, however, we have used simple keyboard typesetting. Of course, our reason for this is that we will very soon specify signatures for software programs, and we will then need notational conventions that are easy to follow using a computer keyboard.

At the top of our sort hierarchy we have the sort *bot*, which subsumes all other sorts. It immediately subsumes the sorts *list*, *animal*, *number*, and *color*, as indicated by indentation. In the same way, indentation indicates that *list* immediately subsumes *nelist* and *elist*, *animal* immediately subsumes *bird* and *pet*, and so on. Sorts that do not subsume any other sorts (except for themselves) are called *maximally specific* sorts or *species*. In our signature *nelist*, *elist*, *parrot*, *woodpecker*, *canary*, *cat*, *dog*, *one*, *two*, *three*, *four*, *green*, *red*, *yellow* and *brown* are species.

The set of attributes is specified by notating each attribute behind the highest sort in the sort hierarchy for which we want to consider it appropriate. For example, *animal* is the highest sort in the hierarchy to which LEGS and COLOR are considered appropriate. By convention, they are then considered appropriate to each subsort of *animal*, and the specification will not be repeated there. By this so-called *attribute inheritance*, COLOR is also appropriate to *bird*, *parrot*, *woodpecker*, *canary*, *pet*, *cat*, and *dog*. A colon is positioned between an attribute and the sort which is considered appropriate for the attribute at the sort which is positioned at the beginning of the line. For example, the sort *color* is appropriate for COLOR at *animal*. These values are also inherited downwards in the sort hierarchy, unless some more specific sort is specified further down. For example, the value

```
type_hierarchy
bot
  list
     nelist head:animal tail:list
     elist
  animal legs:number color:color
     bird legs:two
       parrot
       woodpecker
       canary
     pet legs:four
       cat
       dog
  number
     one
     two
     three
     four
  color
     green
     red
     yellow
     brown
.
```

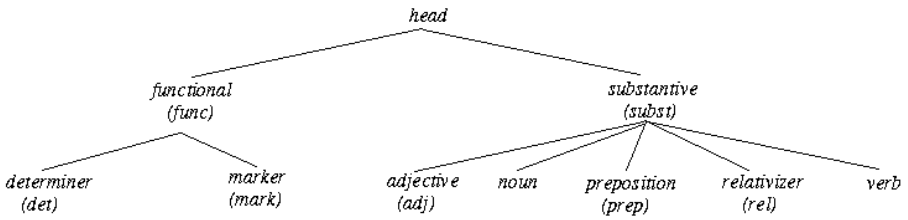
Figure 2.1: A signature specification

of LEGS at *animal* is specified as *number*. Further down the hierarchy, however, the LEGS value becomes *two* (a species subsumed by *number*) at *bird*, and it becomes *four* (another species subsumed by *number*) at *pet*. By our convention of attribute inheritance, this means that the LEGS value at *dog* is *four*, and *two* for *woodpecker*.

Figure 2.1 shows one (particularly compact) way of notating HPSG signatures comprising partitions and feature declarations. As with the syntax of the description languages, many alternatives are conceivable, the most prominent example being the notation in which the signature of the HPSG book is presented and repeated in Section 4.1, *The Sort Hierarchy*, of our course material (which is already a slight variation of the presentation in the book!). Another particularly nice mode of presentation is a graph notation that depicts the sort hierarchy as a taxonomic tree. A part of such a specification can be found in the HPSG book, [Pollard and Sag, 1994], on page 391, which shows *headed-struc* and its immediate subsorts.[1] In the following figure we illustrate that notation with a taxonomic tree showing

[1]When looking at that example you should keep in mind that in the last chapter, Pollard and Sag discuss possible revisions of their grammar that differ from the grammar of the appendix; the piece of sort

the subsorts of *head* as given by the partitions of Section 4.1, *The Sort Hierarchy*:



In graphically even more elaborate presentations of signatures, a sort hierarchy graph can be decorated with appropriateness conditions. The resulting depiction is very reader friendly and at least as compact as our computer friendly version above. In comparison to our version, its major drawback is that we need software that is much more elaborate than a simple editor to create such decorated graphs.

Another option for the notation of signatures will become available as soon as we give a mathematical definition. We may then follow the definition and write down a particular signature by using some mathematical notation for the sets of sorts and attributes, for a *partial order* on sorts, and for a *partial function* that formalizes the appropriateness requirements. However, for reasons of readability it is advisable to prefer one of the available graphical notations.

With a first definition of an initial syntax and a first understanding of signatures, we are now ready to enter into the sequence of formalisms of constraint-based grammar that we promised in the introduction. In the next section, we will formalize our first notion of an HPSG signature and revise the definition of our initial syntax by referring to the more complex signatures that we have just introduced. With those definitions we will immediately be able to define our first notion of a grammar. This will put us in a position to write our first, fully defined constraint-based grammar. The only difference to an HPSG grammar like Pollard and Sag's grammar of English will be that we do not have all syntactic means yet that are necessary to express that grammar completely. For example, we have not included tags and relational expressions like `append` in our syntax.

Before we can extend the syntax accordingly, we will have to face another inevitable question: What do our grammars mean? To give an answer to that question, we will have to consider the various proposals that have been advanced for formalizations of constraint-based HPSG grammars so far, and we will have to make a choice between them. This choice will lead us to an informal discussion of feature structures and interpretations of signatures in terms of feature structures.

Exercises

Exercise 3 *We presuppose the signature presented earlier in this section. Which ones of the following expressions are syntactically well-formed, and which ones are not? For those which are ill-formed, indicate what is wrong with them (keywords suffice).*

hierarchy on page 391 thus differs from the corresponding part of the sort hierarchy in the appendix

1. `nelist,head:(cat;three).`
2. `elist;(nelist,tail):elist.`
3. `~yellow;brown.`
4. `(~color):black.`
5. `woodpecker;legs.`
6. `(list,head:bird:legs:number,color:number);elist.`
7. `nelist,head:(pet:cat);dog.`
8. `head:(parrot,legs:two,color:(yellow;green,~brown)).`
9. `head:bird;pet *> color:color,legs:number.`
10. `color:red;legs:three <*> ~(bird),green,two.`
11. `head,tail *> nelist.`
12. `(head:green,color:one);(head:cat,color:cat).`
13. `animal:pet:(cat;dog).`
14. `~elist <*> nelist:(first,rest).`
15. `nelist,head:legs:(two;four),color:(green;red;yellow;brown),`
    `tail:nelist,head:canary,legs:two,color:yellow,tail:elist.`

Note that in MoMo notation, each description ends with a full stop.

We suggest that you first solve this exercise with paper and pencil, and then check with MoMo whether your answers are correct. For this purpose, we have prepared an mmp file, syntax-exercise.mmp, which you may download.

The file already contains the signature and description cards with all descriptions of the exercise. However, be warned that the syntax checker of MoMo will only tell you about the first syntax error it finds in any given expression. When you analyze ill-formed descriptions, please consider all *mistakes that you can detect in them.*

Exercise 4 *The signature of Pollard and Sag's grammar of English is fairly big. In this exercise, we want to translate a small part of it into the notation for signatures that we have introduced in this section.*

Assume that there is a sort object which subsumes all other sorts of the signature that we want to design. Then take Pollard and Sag's sort head as immediate subsort of object and complete the sort hierarchy under head as it is given by Pollard and Sag. Add all appropriate attributes to the sorts under head as well as the sorts that are in turn appropriate for them.

Hint: Attributes will introduce attribute values that may not be in the sort hierarchy of your signature yet, since they may not be subsorts of head. Introduce them in your sort hierarchy as additional immediate subsorts of object. However, you do not have to complete the sort hierarchy under those additional sorts.

To illustrate this exercise, we have prepared the file nominalobjects.mmp, in which we have done the task we are asking you to do for the sort head for the sort nom-obj.

Use MoMo to verify that you have created a well-formed signature. You can check that by typing your signature into the signature window and then pressing the Check Syntax *button above that window. Don't forget to precede the type hierarchy by the line* type_hierarchy *and to finish it with a full stop in the last line.*

Sketch of the solution:

```
type_hierarchy
object
   head
      subsort-of-head1 attribute1:new-sort1
         moresorts1 ...
      subsort-of-head2 attribute2:new-sort2
         moresorts2 ...
   new-sort1
   new-sort2
.
```

Exercise 5 *This exercise is quite challenging, since we have not yet talked about the meaning of descriptions. However, on the basis of the descriptions of the grammar of English that we already saw and of intuitions that we get from the form of the descriptions, we certainly have some idea about what they mean. Based on those intuitions, try to translate the following statements of natural language into descriptions under the signature introduced in Section 2.1.2.*

1. *Parrots are green, or red, or yellow.*
2. *Lists are brown.*
3. *If something is a pet, then it has four legs and it is not green.*
4. *If something is a list, then it is an empty and non-empty list.*
5. *Something is a list if and only if it has a head and a tail.*
6. *No animals have one leg and are green.*
7. *The first element of a list is not brown, and the third one has two legs.*
8. *The second element of a list does not have six legs.*

9. *Everything.*

10. *Nothing.*

GLOSSARY

2.2 Initial Grammars

Abstract

In this section, we will formalize the syntactic concepts that we have been using informally, and we add a semantics. Our goal is to make mathematically precise what a grammar is, and what a grammar means. To achieve that, we need to define signatures, the set of descriptions of a signature, a certain kind of feature structures, and the meaning of descriptions and sets of descriptions in a domain of feature structures.

2.2.1 The Syntax of Initial Grammars

Abstract

This section contains a definition of initial signatures, defines the description languages with respect to initial signatures and explains what a grammar is.

We begin with defining the essential components of grammars that we extracted from the form of the principles of Pollard and Sag's grammar of English. As we saw before, the *Partitions* and the *Feature Declarations* of Pollard and Sag can be interpreted as belonging to what is more traditionally called the *Signature* of a logical language. When we now define signatures for HPSG grammars, we will not include everything immediately that is needed to formalize a typical HPSG grammar entirely. Initially, we omit everything that has to do with relations in the description language. Our first formalism for constraint-based grammars will allow us to explore constraint-based grammars and their meaning using MoMo and get a fuller understanding of how they work.Then we will augment our technical machinery in two phases of fattening up our mathematical foundations of HPSG.

To distinguish the smaller signatures that we start with terminologically from later signatures, we call them *initial* signatures. The definition of initial signatures does not really contain anything new. All of its components were already contained in our previous characterization of signatures, and when reading the definition the first time, it might be useful to try to guess what each line corresponds to in our example signature of birds and pets.

Definition 2 Σ *is an* ***initial signature*** *iff*

Σ *is a quintuple* $\langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F} \rangle$,
$\langle \mathcal{G}, \sqsubseteq \rangle$ *is a partial order,*
$$\mathcal{S} = \left\{ \sigma \in \mathcal{G} \;\middle|\; \begin{array}{l} \text{for each } \sigma' \in \mathcal{G}, \\ \text{if } \sigma' \sqsubseteq \sigma \text{ then } \sigma = \sigma' \end{array} \right\},$$
$\mathcal{A}$ *is a set,*
$\mathcal{F}$ *is a partial function from the Cartesian product of* $\mathcal{G}$ *and* $\mathcal{A}$ *to* $\mathcal{G}$,
for each $\sigma_1 \in \mathcal{G}$, *for each* $\sigma_2 \in \mathcal{G}$ *and for each* $\alpha \in \mathcal{A}$,

if $\mathcal{F}(\langle \sigma_1, \alpha \rangle)$ *is defined and* $\sigma_2 \sqsubseteq \sigma_1$
then $\mathcal{F}(\langle \sigma_2, \alpha \rangle)$ *is defined and* $\mathcal{F}(\langle \sigma_2, \alpha \rangle) \sqsubseteq \mathcal{F}(\langle \sigma_1, \alpha \rangle)$.

We call each element of $\mathcal{G}$ a *sort*, $\langle \mathcal{G}, \sqsubseteq \rangle$ the *sort hierarchy*, each element of $\mathcal{S}$ a *species*, each element of $\mathcal{A}$ an *attribute*, and $\mathcal{F}$ the *appropriateness function*. If for two sorts σ_1 and σ_2, $\sigma_2 \sqsubseteq \sigma_1$, then we say that σ_2 is *at least as specific as* σ_1. Alternatively, we say that σ_1 *subsumes* σ_2, or σ_2 is a *subsort* of σ_1. Note that subsumption is a reflexive relation (by the definition of which properties the relation in a *partial order* must have). In other words, each sort is assumed to subsume itself. As we said before, species are sorts that subsume no other sort besides themselves. $\mathcal{S}$ is of course a subset of $\mathcal{G}$, and it is always possible to recover $\mathcal{S}$ from $\mathcal{G}$ by looking at the sort hierarchy. In fact, this is exactly how $\mathcal{S}$ is defined in line 4: $\mathcal{S}$ contains exactly all of those elements of the set of sorts, $\mathcal{G}$, that are only subsumed by themselves and no other sort. When we introduce a symbol for the set of species in the signature, we do it simply for later convenience: We will often refer to the set of species, and having a name for that set in the signature makes it easier for us to do it.

The condition on $\mathcal{F}$ (lines 7–9) enforces attribute inheritance: If an attribute α is appropriate to some sort σ_1, then it is also appropriate to all subsorts of σ_1, and the value of $\mathcal{F}$ at the subsorts of σ_1 and α is at least as specific as $\mathcal{F}(\langle \sigma_1, \alpha \rangle)$. This is the attribute inheritance that we have observed in our example signature of Section 2.1.2, *Signatures: Partitions and Feature Declarations*, where birds and pets (and their subsorts) inherited the attributes LEGS and COLOR from their supersort *animal*; and the attribute values at these sorts were at least as specific as at their respective supersorts. Note that the definition of an initial signature does not enforce the presence of a top or bottom sort in the sort hierarchy, and leaves the decision of whether to include or omit them to the grammar writer.

Computational systems for the implementation of HPSG grammars are usually less general and impose stricter conditions on the form of the sort hierarchy. Sometimes the sort hierarchy necessarily contains some pre-defined sorts together with some appropriate attributes and attribute values. They also might enforce additional restrictions on where you are allowed to introduce attributes as appropriate to a sort in the hierarchy.

Let us illustrate the definition of initial signatures by stating the small example signature of lists, pets, and birds of Section 2.1.2, *Signatures: Partitions and Feature Declarations*, in its mathematical notation:

$$\mathcal{G} = \left\{ \begin{array}{l} bot, list, nelist, elist, animal, bird, parrot, woodpecker, canary, pet, \\ cat, dog, number, one, two, three, four, color, green, red, yellow, brown \end{array} \right\}$$

$$\sqsubseteq = \{(\sigma,\sigma) \mid \sigma \in \mathcal{G}\} \cup \{(\sigma, bot) \mid \sigma \in \mathcal{G}\} \cup \left\{ \begin{array}{l} (nelist, list), (elist, list), (bird, animal), (parrot, animal), \\ (woodpecker, animal), (canary, animal), (pet, animal), \\ (cat, animal), (dog, animal), (parrot, bird), \\ (woodpecker, bird), (canary, bird), (cat, pet), (dog, pet), \\ (one, number), (two, number), (three, number), (four, number), \\ (green, color), (red, color), (yellow, color), (brown, color) \end{array} \right\}$$

$$\mathcal{S} = \left\{ \begin{array}{l} nelist, elist, parrot, woodpecker, canary, cat, dog, \\ one, two, three, four, green, red, yellow, brown \end{array} \right\}$$

$$\mathcal{A} = \{\text{HEAD}, \text{TAIL}, \text{LEGS}, \text{COLOR}\}$$

$$\mathcal{F} = \left\{ \begin{array}{l} ((nelist, \text{HEAD}), animal), ((nelist, \text{TAIL}), list), ((animal, \text{LEGS}), number), \\ ((bird, \text{LEGS}), two), ((parrot, \text{LEGS}), two), ((woodpecker, \text{LEGS}), two), \\ ((canary, \text{LEGS}), two), ((pet, \text{LEGS}), four), ((cat, \text{LEGS}), four), \\ ((dog, \text{LEGS}), four), ((animal, \text{COLOR}), color), ((bird, \text{COLOR}), color), \\ ((parrot, \text{COLOR}), color), ((woodpecker, \text{COLOR}), color), \\ ((canary, \text{COLOR}), color), ((pet, \text{COLOR}), color), ((cat, \text{COLOR}), color), \\ ((dog, \text{COLOR}), color) \end{array} \right\}$$

Based on the definition of initial signatures, we can now define the notion of descriptions relative to a given initial signature:

Definition 3 *For each initial signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F} \rangle$, $\mathcal{D}^\Sigma$ *is the smallest set such that*

for each $\sigma \in \mathcal{G}$,

$\sigma \in \mathcal{D}^\Sigma$,

for each $\alpha \in \mathcal{A}$, *for each* $\delta \in \mathcal{D}^\Sigma$,

$\alpha : \delta \ \in \mathcal{D}^\Sigma$,

for each $\delta \in \mathcal{D}^\Sigma$, $\sim \delta \ \in \mathcal{D}^\Sigma$,

for each $\delta_1 \in \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathcal{D}^\Sigma$, $(\delta_1, \delta_2) \in \mathcal{D}^\Sigma$,

for each $\delta_1 \in \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathcal{D}^\Sigma$, $(\delta_1; \delta_2) \in \mathcal{D}^\Sigma$,

for each $\delta_1 \in \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathcal{D}^\Sigma$, $(\delta_1 *> \delta_2) \in \mathcal{D}^\Sigma$, *and*

for each $\delta_1 \in \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathcal{D}^\Sigma$, $(\delta_1 <*> \delta_2) \in \mathcal{D}^\Sigma$.

We call each element of $\mathcal{D}^{\Sigma}$ an *initial* Σ *description*. Although our new definition of the syntax of our formal language now refers to initial signatures instead of just to a set of sorts and a set of attributes, nothing has really changed in our previous definition. The reason for having changed very little is that our descriptions do not depend on the sort hierarchy or on the appropriateness function, which are the new elements in the signatures. Obviously, our syntax of initial Σ descriptions is not sufficient yet to notate all principles of a typical HPSG grammar of a natural language. For example, there is no syntax to write down the boxed integers called *tags*, and there are no relation symbols. Once we have explored the meaning of grammars in the given fragment of a syntax for HPSG, we will extend it by plugging in missing elements of a complete syntax and semantics. For the moment, we will build our definition of *initial grammars* on initial signatures and descriptions. In the definition below of an initial grammar we use the insight that the principles of an HPSG grammar are a set of descriptions. We use the Greek letter Θ to refer to a set of initial Σ descriptions:

Definition 4 Γ *is an* ***initial grammar*** *iff*

Γ *is a pair* $\langle \Sigma, \theta \rangle$,

Σ *is an initial signature, and*

$\theta \subseteq \mathcal{D}^{\Sigma}$.

An initial grammar is simply an initial signature, Σ, together with a set of initial Σ descriptions. To write a grammar, we have to define a signature, and we have to state a set of descriptions. Pollard and Sag did that in their grammar of English by notating the signature in the form of Partitions (which comprised the declaration of a set of sorts and the sort hierarchy) and Feature Declarations (comprising the declaration of the set of attributes and the appropriateness function), and stating the set of descriptions as a set of Principles of Grammar in natural language. To turn the Principles into a formal theory, we have to rephrase them in an appropriate formal language based on Pollard and Sag's signature.

Exercises

Exercise 6 *In the present section, we showed how a MoMo signature can be rewritten in a standard set notation of mathematics: We saw the MoMo signature with lists and animals of Section 2.1.2,* Signatures: Partitions and Feature Declarations*, in a notation that follows Definition 2 of initial signatures.*

In Exercise 4 of Section 2.1.2, Signatures: Partitions and Feature Declarations*, we asked you to reconstruct a small part of Pollard and Sag's signature of English in MoMo notation. The relevant part of the signature was the sort hierarchy under head plus that part of the overall appropriateness function that concerns head and its subsorts. Take the signature that you constructed in MoMo (where you had to use MoMo's notation for signatures) and write it down in the notation of our definition of initial signatures (as given at the beginning of this section).*

Exercise 7 *In this exercise, you will write your first grammar:*

Take the MoMo version of the signature of the previous exercise and write three arbitrary, well-formed descriptions for that signature in MoMo. Your descriptions should consist of at least five symbols (a sort name and an attribute name count as one symbol only). Please upload the resulting mmp-file to ILIAS.

Exercise 8 *We said that we did not want to introduce a potentially complicated definition of a syntax of AVM descriptions as Pollard and Sag use in their HPSG book. Nevertheless, we will often appeal to a correspondence between the syntax of descriptions that we use in MoMo and the AVM notation that is standard in linguistics. With a bit of practice, you will in fact find that this correspondence is fairly obvious. This exercise is meant to help you get some of the necessary practice.*

In the file momo-avms.mmp you find MoMo descriptions that correspond to the first three AVM descriptions below. Complete this file with corresponding MoMo descriptions for the last three AVMs and upload the completed file to ILIAS.

a.
$$\begin{bmatrix} \textit{cat} \\ \text{COLOR} \;\; \textit{yellow} \vee \textit{brown} \end{bmatrix}$$

b.
$$\begin{bmatrix} \textit{dog} \\ \text{LEGS} \;\; \neg \; \textit{three} \end{bmatrix}$$

c.
$$\begin{bmatrix} \textit{list} & & \\ \text{HEAD} & \begin{bmatrix} \textit{canary} \\ \text{COLOR} \;\; \textit{yellow} \end{bmatrix} & \\ \text{TAIL} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{woodpecker} \\ \text{COLOR} \;\; \textit{brown} \end{bmatrix} \\ \text{TAIL} & \textit{elist} \end{bmatrix} & \end{bmatrix}$$

d.
$$\begin{bmatrix} \textit{parrot} \\ \text{COLOR} \;\; \textit{green} \vee \textit{red} \vee \textit{yellow} \vee \textit{brown} \end{bmatrix}$$

e. $\begin{bmatrix} \textit{canary} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{COLOR} \;\; \textit{yellow} \end{bmatrix}$

f.
$$\begin{bmatrix} \textit{nelist} \\ \text{TAIL TAIL} \;\; \textit{elist} \end{bmatrix} \leftrightarrow \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{canary} \\ \text{COLOR} \;\; \textit{yellow} \end{bmatrix} \\ \text{TAIL} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{parrot} \\ \text{COLOR} \;\; \textit{green} \end{bmatrix} \\ \text{TAIL} & \textit{elist} \end{bmatrix} \end{bmatrix}$$

GLOSSARY

2.2.2 Meaning

Abstract

We discuss three ways of defining the meaning of grammars that have been proposed in the development of constraint-based HPSG. We will see that a choice between them must be based on a decision between different views on what a scientific theory does, and is thus ultimately based on one's preferred philosophy of science. For the purposes of our course, we will not take a position but make a pragmatic choice.

When we define the meaning of grammars, we first have to decide what the objects in the denotation of grammars are supposed to stand for. In the evolution of the HPSG formalisms following the tradition of Pollard and Sag's HPSG book, different answers were given to this fundamental question.

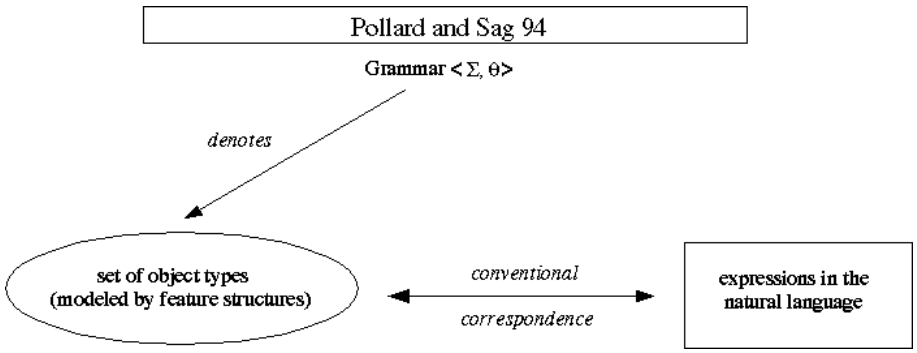
Pollard and Sag themselves envisioned a formalism in which grammars are about the object types of a language. A grammar admits a collection of object types, which are constructed as a collection of a specific kind of feature structures. [King, 1999] presented a formalism in which a grammar delineates a class of what King calls *exhaustive models*. The exhaustive models of a grammar are mutually indiscernible, and one of them contains exactly all possible tokens of the natural language that the grammar is about. [Pollard, 1999] presented a formalization of HPSG in which the *strong generative capacity* of a grammar is defined as an isomorphism class of structures that are mathematical idealizations of natural language tokens. Neither the formalization of King nor the formalization of Pollard uses feature structures.[2]

The crucial difference between the three approaches is their views about how the objects in the denotation of a grammar (or a scientific theory in general) are related to the empirical domain of observable and measurable data. Grossly oversimplifying, one might say that Pollard and Sag believe that knowledge of language is knowledge about the object types of a language, object types are real objects, and a theory of grammar should include mathematical entities that function as models of object types. King denies that, at the current stage of linguistic research, object types are a scientifically useful concept, and claims that a grammar should talk directly about the domain of observable data, which are language tokens. Pollard abandons Pollard and Sag's claim about the ontological reality of object types but maintains the view that grammars should be about mathematical models of language that do not contain structurally isomorphic members. They contain exactly one structurally isomorphic member for each utterance token that will be judged grammatical.

Let us consider an example to illustrate the consequences of the different views on the meaning of grammars. We look at the sentence *There are four CDs on my desk.* According to Pollard and Sag, there is a unique object type of that sentence in the real world, and this object type can be mathematically constructed as some kind of feature

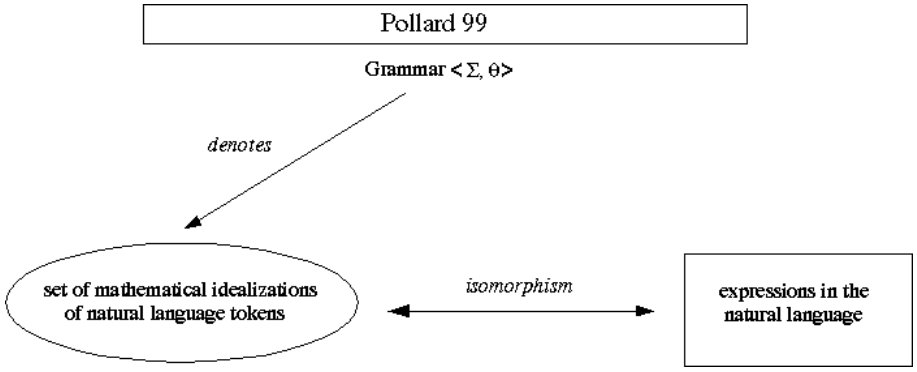
[2]Pollard still calls the structures in the strong generative capacity of grammars feature structures. However, his feature structures are construed differently from any kind of classical feature structures, and some of their important properties differ from those of traditional feature structures.

structure. Of course, when we observe occurrences of that sentence in the real world (for example, if Beata asks me at 7.45pm on November 18th, 2002, how many CDs I have on my desk, and I answer *There are four CDs on my desk*), we do not directly observe the feature structure that is the mathematical structure of that object type. We observe an acoustic phenomenon or an utterance token of the sentence, which we assume to stand in a conventional correspondence to the abstract feature structure representing that sentence in the denotation of our grammar of English. The conventional correspondence is not part of our theory of English. We assume that a linguist recognizes that correspondence and that linguists agree on the correspondences. If they do not, data become immune to empirical falsification. The following figure illustrates Pollard and Sag's architecture:



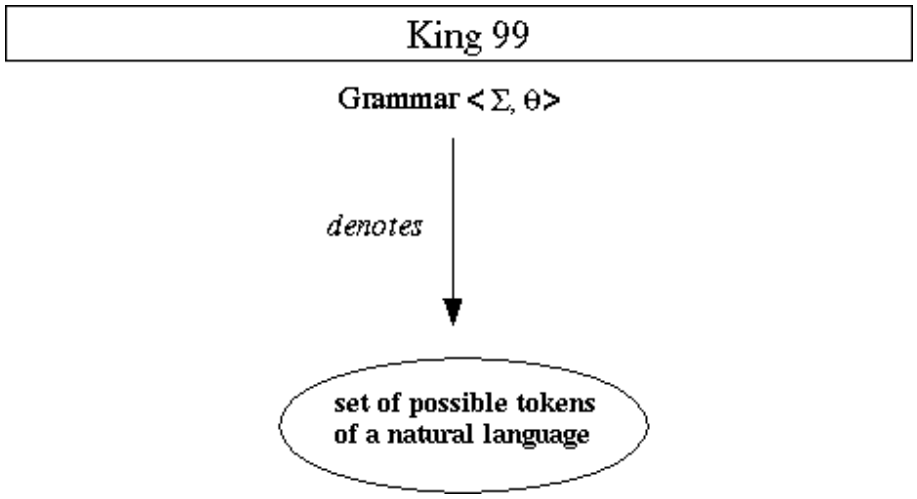
In this picture, we see how Pollard and Sag use the term *type* in constraint-based HPSG: It is reserved for object types of natural language expressions and is now understood in opposition to the notion of *utterance tokens*, concrete utterances of sentences by a specific person at a specific coordinate in space and time. Entire feature structures may be object types, whereas the most specific *sorts* of the sort hierarchy provide the labels of the nodes in feature structures, which is something entirely different from types. The connection between sorts and feature structures will be explained in detail when we define the architecture of feature structures.

[Pollard, 1999] tightens the relationship between the denotation of a grammar and the empirical domain of observable, measurable data. This is done by replacing the conventional correspondence between the domain of the grammar and the domain of empirical phenomena by a relationship of isomorphism. The entities in the denotation of a grammar become isomorphic mathematical idealizations of concrete utterance tokens. Technically, this cannot be done with the kind of feature structures that are standardly used to formalize Pollard and Sag's architecture. Therefore Pollard replaces them with different mathematical structures that function as models of grammars. Pollard's view can be visualized as follows:



Pollard maintains that there is a mathematical modeling domain that intervenes between grammars and the empirical domain. Moreover, a sentence like *There are four CDs on my desk* receives a unique mathematical representation in the denotation of the grammar.

[King, 1999] challenges the view that types are an interesting and justifiable concept in the scientific description of language, and he replaces models that contain types with models that contain utterance tokens. In King's opinion, grammars directly denote collections of possible utterance tokens:



According to King, the grammar of English denotes as many tokens of the sentence *There are four CDs on my desk* as there were, are, and will be, utterances of this sentence in the real world, including me uttering that sentence to Beata at 7.45pm on November 18th, 2002 in the Seminar für Sprachwissenschaft in Tübingen. According to this view, there are no intervening mathematical structures between the grammar and the domain of observable linguistic data. The tight connection between the two is supposed to close any possible escape hatches that anyone might want to use in order to avoid the falsification of their theory of English by observable data.

We will not delve deeper into the comparison of the respective advantages and the problems of these three explanations of the meaning of constraint-based grammars. A choice

between the three views about the meaning of grammars is first of all a choice between different views of the nature of scientific theories. This choice has nothing to do with technical matters; it is about philosophical arguments. As for the underlying mathematics, [Richter, 2004] shows how all three views can be integrated within one unifying formalism of HPSG, and how the three different modeling domains are mathematically related. In effect, it is possible to switch arbitrarily from any one of the models suggested above to the others. From a mathematical point of view, feature structure models are just a very specialized kind of model; the model theory of King is more general and follows standard ideas of logical models. Pollard's models are defined on the basis of King's denotation functions, combining them with a variation of the techniques usually employed in defining feature structure models.

When we use feature structures in our formalization of the meaning of grammars, we do not intend to express a preference as to which idea about the nature of scientific theories is correct. Our decision is purely pragmatic in nature: Computational theories of constraint-based grammars have traditionally relied on feature structure models of grammars and on processing methods for feature structures. In the context of grammar implementations, feature structure models are no more than an efficient tool for computational purposes. On a theoretical level, we know that feature structure models are mathematically closely related to a precise version of each of the other views about the meaning of grammars. Thus, we know that feature structure models of grammars can always be reinterpreted in light of the other two model-theoretic interpretations of grammars.

GLOSSARY

2.2.2.1 Concrete Feature Structures

Abstract

We will introduce concrete feature structures *as interpreting structures of constraint-based grammars and explain their most important properties. In particular we will focus on the connection between signatures and feature structures.*

The original idea of feature structures as some kind of graph came to linguistics from theoretical computer science. In the theory of finite state automata, it is common to think of the states of a machine as nodes in a graph, postulate a distinguished start state and picture the transitions between the states of an abstract machine as arcs from one state to another. Two alphabets are often associated with these automata, one for the arcs (modeling input strings) and one for the states (modeling output strings). Additionally, there can be result states in automata, and the architecture of the automata can vary from the one we have just sketched in many ways, depending on what the purpose of the automata is.

From early on linguists have found these graph representations useful for modeling the interpreting structures of so-called feature-value grammars, and HPSG inherited that tradition. The start state of the automata becomes the *root node* of feature structures,

where the root node is defined as a node from which all other nodes in the feature structure can be reached by following a finite sequence of arcs. The nodes of feature structures are of course derived from the states of the automata. The input alphabet of automata turns into the set of attributes, which now label the arcs. Elements of the species subset of the set of sorts label the nodes of the feature structure. We say that the nodes in the feature structure have certain sorts (the sorts that label them). Similarly, we also say that an entire feature structure is of some sort σ, meaning that the root node of the feature structure is labeled by σ.

Feature structures of the kind envisioned here consist of four defining constructs. First, a set of nodes. Second, a distinguished root node. Third, a transition function whose task it is to tell us which nodes can be reached from which other nodes by following an arc labeled by a particular attribute. And finally, a labeling function that assigns each node a (maximally specific) sort label.

Pollard and Sag's requirement that not just any sort may be used as a label for the nodes of feature structures is to be understood as an ontological assumption about linguistic objects. The fact that feature structures must be *sort-resolved*, which means that all nodes must be labeled by species, formalizes the idea that linguistic objects themselves are entities of maximally specific sorts whose components (as given by the nodes that we can reach by following the arcs from the root node) are again of maximally specific sorts. To draw an example from Pollard and Sag's grammar of English, remember that *word* and *phrase* are immediate subsorts of *sign* and have no proper subsorts. With these signature specifications, we can talk about *signs* in the description language (because *sign* is a sort, and any sort can occur in descriptions), but in the real world we only find feature structures of the maximally specific sorts *word* or *phrase*. There is no such thing as a feature structure of sort *sign*, because Pollard and Sag believe that there is no linguistic entity of sort *sign*.

Sort-resolvedness is by no means a necessary property of feature structures. For example, sort-resolvedness is not a property of the feature structures of HPSG 87. In general we observe that the feature structures of HPSG 87 have very different mathematical properties from the feature structures of HPSG 94, and it is very important to keep those two formalisms cleanly separate when working with HPSG.

Sort-resolvedness is the first point where we can see that feature structures depend on the structure of signatures even more than descriptions. Not only do we need to know the set of sorts and the set of attributes (to label nodes and arcs) as we did for descriptions, we also need to know the sort hierarchy in order to determine which sorts are species. The appropriateness function (the Feature Declarations) is equally crucial as the sort hierarchy for the feature structures of HPSG, due to a completeness assumption that Pollard and Sag make regarding linguistic objects. This assumption says that all or none of the objects of a certain sort may have certain properties, where properties are represented by attributes (or, more precisely, arcs labeled by attributes). The properties of the entities of each sort are declared by the appropriateness function. A node of some species σ has an outgoing arc labeled by some attribute α if and only if the attribute is appropriate to the species; and the node that we reach by that arc must be of a species that is either the sort σ' appropriate for α at σ, or some subsort of σ' (in case σ' is not a species). Again, we have

to stress that the reason for this completeness assumption about feature structures is an assumption about the empirical domain of linguistic objects. It is not given by the use of feature structures *per se*. Feature structures that are complete in the way discussed here are called *totally well-typed*. *Well-typedness* means that every node has only outgoing arcs labeled by attributes appropriate to the species of the node. *Total* well-typedness means that it has arcs for exactly *all* of those attributes that are appropriate to its species.

Since feature structures are defined relative to a given signature Σ (a property which they share with descriptions), strictly speaking we always have to refer to them as Σ feature structures in order to say which underlying signature we have in mind. In practice this turns out to be very cumbersome. For that reason, we will usually omit the reference to a signature wherever possible without causing any confusion.

The properties of feature structures, sort-resolvedness and total well-typedness, that the linguist imposes by the sort hierarchy and the appropriateness conditions of the signature, express ontological assumptions about natural language. By declaring a signature, a linguist declares what kind of linguistic entities she expects to exist in the world, and what attributes they all have. If the kind of ontology envisioned here is not the kind of ontology that one has in mind for linguistic entities, then constraint-based HPSG does not provide the right framework to express one's theories of language.

There is one final consideration about feature structures that we need to discuss. Why do we call them *concrete* feature structures in this section's heading? The reason is trivial, but it will have mathematical consequences: The feature structures derived from automata theory that we have talked about so far are concrete objects. They consist of nodes and arcs, and we can even build real models of them out of wood and wire. In MoMo we draw *paintings* or *pictures* of them that are also quite concrete. However, this concreteness has drawbacks. Nothing stops us from having two concrete feature structures of the same shape. To evoke our signature of birds and pets, we can easily have two distinct concrete feature structures of a green parrot with two legs. This is clearly not what Pollard and Sag had in mind for feature structure representations of object types, which where supposed to be *unique* representations of linguistic expressions. To construct a more linguistic example than parrots, with concrete feature structures, we may construct two structurally isomorphic but distinct concrete feature structure representations of the sentence *There are four CDs on my desk*. How should we precede?

The answer is quite simple. When we get to the formalization of feature structures, we will formalize them in such a way that two structurally isomorphic feature structures will necessarily be identical. Instead of using the concrete, automata theoretic graph construction we have worked with in this section, we will use a construction involving equivalence classes, which will result in so-called *abstract* feature structures. Abstract feature structures will have exactly those properties that Pollard and Sag had in mind. For our illustrations however (and for our work with MoMo), concrete feature structures are better suited, simply because of their tangible concreteness. We will thus go on using them, but keep in mind that our mathematical reformulation will abstract away the very concreteness that we like about these objects.

It is time to get some experience with the effects of signatures on the possible shapes of feature structures, and the best way to acquire it is to construct some feature structures and see what they are like. The file birdsandpets2221.mmp gets us started with a number of examples that we now discuss in turn.

The file birdsandpets2221.mmp contains the familiar signature of birds, pets and lists, and six interpretations with one feature structure of that signature in each. Two of the feature structures are well-formed, but four are not. The reason for why it is not a well-formed feature structure is different in each case.

The feature structure in *well-formed1* depicts a cat of color brown with four legs. The node labeled *cat* is marked as the root node. According to the signature, LEGS and COLOR are the appropriate attributes to *cat*, with the values restricted to *four* and a maximally specific subsort of *color*, respectively. Since our feature structure respects these restrictions, we say it obeys the signature. Moreover, since it has a root node from which all other nodes can be reached by following a sequence of arcs, we say that this configuration of objects (consisting of labeled nodes and labeled arcs) is a well-formed feature structure.

The configuration of objects in *well-formed2* is particularly trivial. It consists of a node labeled *red*, which is a maximally specific *atomic* sort of the signature. Sorts are called atomic if no attributes are appropriate to them. Since there are no outgoing arcs on red, this trivial configuration of objects obeys the signature. Since our only node is marked as the root node of the configuration, we have a well-formed feature structure.

The configuration of objects in *ill-formed1*, which fails to be a well-formed feature structure, is much bigger than the ones that we have seen so far. The root node is of sort *nelist*, and the first element of the list—which we reach from the root node by following the arc labeled HEAD—is a feature structure representation of a yellow canary. The second element of our list, which we could also call the first element of the tail of the list, is a (representation of a) yellow cat. So far, we can easily check that everything is well: The *canary* node has exactly those outgoing arcs that are required by appropriateness in the signature, and so does the *cat* node. The nodes that these arcs point to have labels that also agree with appropriateness. Since they are all atomic, the lack of any outgoing arcs from any of them is correct. Moreover, the present configuration of objects is a well-formed feature structure in the sense that each node in the configuration may be reached from the distinguished root node by following some sequence of arcs. However, there is a problem with completeness. The second *nelist* node does not have an arc labeled TAIL besides the HEAD arc that leads to the yellow cat. But according to the signature, it should have one to be well-formed.

MoMo tells us about the problem when we press the key *Obeys Signature*. The communication window displays a message that names the node at which MoMo detected the missing arc. MoMo displays the names that it automatically assigns to each node next to them on the canvas if you set the display options accordingly as explained in the MoMo manual. After a well-formedness check with respect to the requirements of the signature, MoMo colors the part of the feature structure (starting from the root node) that it found to be well-formed.

The configuration of objects in *ill-formed2* fails to be a feature structure for a different reason. According to the connectedness condition on the nodes and reachability from the root node, it is fine. In contrast to the previous example, this configuration also contains all correct arcs: The *canary* node has a COLOR and a LEGS arc. The problem is that the label of the target node of the LEGS arc, *three*, violates the signature, which says that the value of LEGS at canary must be *two*. Again, MoMo tells us exactly what is wrong when we press the button *Obeys Signature*.

The structure in *ill-formed3* is well-formed with respect to the signature. This time, however, there is a problem with the basic properties that feature structures possess in general, since there is no distinguished root node.

The configuration in *ill-formed4* suffers from a similar flaw. There is a root node, but it is not the case that all other nodes in the structure that are connected to it can be reached from it by following some sequence of arcs. Therefore, the structure cannot be a well-formed feature structure, although it fulfills the requirements of the signature.

The following exercises give you an opportunity to get some hands-on experience in constructing concrete feature structures. However, a small number of exercises are only a start, and it would be very useful to go beyond those exercises and experiment a bit with MoMo. It is fairly easy to construct a signature of any empirical domain that you can imagine and construct feature structures for it, and doing so remains the best way to study our feature structure formalism.

Exercises

We use our familiar signature with lists, birds and pets, which we have prepared for this exercise in the file birdsandpets-exs2221.mmp.

Exercise 9 *Create an interpretation called* two parrots. *Draw* two *green parrots in MoMo (on one canvas). Use the functions of MoMo to make sure that you created two well-formed feature structures.*

Exercise 10 *Create a second interpretation (while keeping the old one) called* list of length 2. *Draw a list with two elements, the first being a yellow canary and the second being a brown dog.*

Recall that nonempty lists are represented as nelist nodes, with an outgoing arc labeled HEAD *that leads to the first element of that nonempty list and a second outgoing arc labeled* TAIL *that leads to the remainder of that list. The remainder might either be an elist node (the list ends here) or another nelist node (the list continues).*

Exercise 11 *Create three graphs that are not well-formed feature structures, either because they violate the signature or other conditions on what constitutes a feature structure. Indicate briefly why each one of them is ill-formed. Please use a new interpretation window for each ill-formed graph and assign meaningful names to the interpretation windows.*

When you finish the exercises, please save your results as an mmp-file with the name `exercises-sec-2221.mmp` and upload the file to your group in ILIAS.

GLOSSARY

2.2.2.2 Satisfaction

ABSTRACT

We introduce an intuitive notion of constraint satisfaction.

What does it mean for a feature structure to satisfy a description? Clearly we want to say that for the satisfaction relation to hold between a feature structure and a description, the feature structure should in some intuitive sense be described by the description, or the description should be true of the feature structure.

Working once more with the signature of Section 2.1.2 *Signatures: Partitions and Feature Declarations*, it seems to make sense to say that the description `yellow` describes feature structures with root label *yellow*. Since there are no attributes appropriate to yellow, we know in addition that there will not be any arcs leaving that root node. When we add negation and say `~yellow`, it then makes sense to say that all feature structures that do not satisfy `yellow` should satisfy `~yellow`. This would of course be all well-formed feature structures except the one that consists only of a root node labeled *yellow*. Considering the fact that the feature structures representing lists of animals can be of arbitrary size due to the length of lists, it follows immediately that `~yellow` is satisfied by infinitely many feature structures of different form.

Let's consider the description `pet`, which, intuitively, should be satisfied by all feature structures that represent pets. Which ones are they? Since feature structures are of a maximally specific sort, and the maximally specific subsorts of *pet* are *cat* and *dog*, it must be the feature structures with root nodes labeled *cat* or *dog*. These feature structures have outgoing arcs according to the signature: one arc is labeled LEGS and leads to a node labeled *four* (due to the appropriateness function). The other one is labeled COLOR and leads to a node labeled by one of the four speciate colors. Since the color species and *four* are atomic, i.e., there are no attributes appropriate to them, this exhausts the cases that we have to consider.

What about `(dog,legs:three)`? Is this a well-formed description at all? It certainly is according to our definition of the initial syntax in Section 2.2.1, *The Syntax of Initial Grammars*. First of all, according to line 8 of the definition, it is a description if `dog` is a description δ_1 and `legs:three` is a description δ_2. But `dog` is a description according to lines 3 and 4, because `dog` is a sort. `legs:three` is a description if `legs` is an attribute (according to lines 5 and 6), which it is, and if `three` is a description (still according to lines 5 and 6). But `three` is a description, because it is a sort (lines 3 and 4). Therefore, `(dog,legs:three)` is a description.

Why would anyone want to claim that something is wrong with `(dog,legs:three)`? This claim would probably have to do with the signature, more precisely with the appropriateness conditions. In our signature we say that LEGS is appropriate to *dog*, and the

value of LEGS at *dog* is *four*. However, the appropriateness conditions of the signature do not have any effect on well-formed descriptions. If we look at the definition of our initial syntax, we see that descriptions are built from attributes, sorts, and logical symbols without any reference to the sort hierarchy or the appropriateness conditions.

This is of course different for feature structures. We introduced well-formed feature structures with respect to the sort hierarchy (nodes are only labeled by those sort symbols that do not have any proper subsorts) and the appropriateness conditions (arcs have to respect appropriateness with respect to the node where they originate and the node they lead to). Therefore there cannot be a feature structure with a node labeled *dog* and an outgoing arc labeled LEGS that leads to a node labeled *three*: it would violate the appropriateness function of our signature. What we can conclude from all of this is that although `(dog,legs:three)` is a well-formed description, there cannot be any feature structure that satisfies it, because there is no well-formed feature structure that represents a dog with three legs. In that sense, we are in a similar situation to the first order expression $p \wedge \neg p$, which will always denote *False*, because a sentence cannot simultaneously hold together with its negation.

With a bit of classical logic as background, it is now easy to infer how the satisfaction function for more complex descriptions is defined. $(\delta_1\,;\delta_2)$ is satisfied by all feature structures that satisfy δ_1 or δ_2. $(\delta_1$`*>`$\delta_2)$ is satisfied by all feature structures such that if they satisfy δ_1 then they also satisfy δ_2, which is classically equivalent to saying that $(\delta_1$`*>`$\delta_2)$ is satisfied by all feature structures such that they satisfy `~`δ_1 or they satisfy δ_2. It is particularly important to keep that in mind, since all our principles of grammar will be formulated as implications (you may want to go and check the principles of Pollard and Sag, and you will find that they are all formulated as implications as well). We can leave it to the reader to figure out which feature structures satisfy the description $(\delta_1$`<*>`$\delta_2)$.

Exercises

Once more we take the signature that we first introduced in Section 2.1.2, *Signatures: Partitions and Feature Declarations* with lists, birds, and pets. We also refer to the well-formed descriptions of Exercise 3 of Section 2.1.2, *Signatures: Partitions and Feature Declarations*. For your convenience, you may want to download the file satisfaction-exercise.mmp, which already contains the signature and the relevant descriptions.

Use MoMo to create a file `exercises-2222.mmp` containing the solutions to the following exercises.

Exercise 12 *Create interpretations with feature structures that satisfy the description in (1) and in (8).*

Exercise 13 *Is the description in (12) satisfiable? If it is, draw a feature structure satisfying it. If not, give a short explanation why it is not.*

Exercise 14 *This exercise is a bit tricky but lots of fun if you like syntactic puzzles. It may take some time to solve it.*

The description in (15) is well-formed but not satisfiable. It is possible to turn the description into a satisfiable description by modifying it in such a way that all non-logical symbols (attributes and sorts) are kept exactly in the order in which they appear in the description, none is added or removed, and other than that only the bracketing is changed and commas (`,`) may be turned into colons (`:`) or vice versa*. Do that and draw a feature structure satisfying your description.*

GLOSSARY

2.2.2.3 Admission

ABSTRACT

We will discuss why constraint satisfaction is not the right notion of meaning for interpreting generalizations over the shape of feature structures. To capture the intended meaning of descriptions that are generalizations over domains of objects, we will introduce a second notion of the meaning of descriptions. We will also say what it means for a constraint to license *a feature structure.*

It might seem strange that we return to the question of the meaning of descriptions again after having introduced satisfaction functions. Doesn't satisfaction tell us everything we need to know about the denotation of descriptions? In order to see that it does not, consider the following situation that is typical for the type of generalizations we want to state in grammars:

Assume that we enumerate the animals that a person owns on a list. Since we want to keep that enumeration as economical as possible, we do not want to describe every single animal in full detail when we can generalize over common properties of all members of some classes of animals. For example, if all dogs happen to be brown, all canaries are yellow, and all parrots are green, we would like to state these facts only once without having to repeat them in the descriptions of every dog, canary, and parrot. As it turns out, we cannot do that using the satisfaction function as our notion of the meaning of descriptions!

Why is it not possible? Let us focus on the generalization that all canaries are yellow. An obvious try to make that statement seems to be to say `(canary,color:yellow)`. However, this description is true of a feature structure exactly if its root node is of sort *canary*, and its COLOR value is *yellow*. Clearly, we cannot make the intended statement about all canaries on a feature structure representing lists, which is a feature structure of sort *nelist*, that way.

What about an implicational statement, then? After all, we have already noted that the principles of linguistic grammars are of that form. Unfortunately, that doesn't help us much either. Consider `(canary *> color:yellow)`. This description is satisfied by all feature structures that are either not of sort *canary*, or their COLOR value is *yellow*. Because of the antecedent, it thus holds of any list, no matter what elements are on that list. For example, a list with red canaries will do just fine.

While the problem with our first attempt, `(canary,color:yellow)`, had to do with the form of the description itself, the problem that we observe with the implicational statement is a problem with the use of constraint satisfaction as the interpretation of descriptions: Under constraint satisfaction the descriptions are always interpreted with respect to the root node of feature structures. What we need to do to get the right interpretation for our generalization is a notion of the meaning of descriptions that allows us to impose restrictions on entire feature structures, comprising the root node *and* all nodes inside of the feature structure. With our generalizations, we want to say something about feature structures representing canaries and about representations of canaries inside a bigger overall feature structure, for example about canaries on lists.

We observe the very same situation in linguistic examples. Consider the HEAD FEATURE PRINCIPLE, which says that in headed structures, the HEAD values of mother and head daughter must be identical. It is evident that this constraint should not just hold for the topmost phrase in a sentence, but for each headed phrase that might occur embedded more deeply in phrase structure. The situation is thus exactly parallel to the situation we found ourselves in with our statement that every canary on the list must be yellow.

The solution to this problem is to come up with a stronger notion of denotation, and this is what feature structure admission is designed for. As we have just seen in the examples of yellow canaries on lists and the HEAD FEATURE PRINCIPLE, we want a denotation function that in a sense generalizes constraint satisfaction: For *admission*, it is not simply the root node that is relevant, but all nodes in the feature structure. We want to say that a feature structure is admitted by a constraint if and only if its root node and *all* the nodes that occur in the feature structure satisfy the constraint. Note that the definition of admission is directly based on satisfaction. We could characterize admission as some kind of super-satisfaction that generalizes satisfaction from the root node to all nodes of feature structures. From this perspective, it is also immediately obvious why admission is a more powerful notion of denotation.

The notion of admission solves our problem with yellow canaries. The root node of a list satisfies the constraint `(canary *> color:yellow)`, because it is not a canary. Assume the HEAD of the list is a canary. Then the entire list can only be admitted by the constraint if that canary is yellow. If the HEAD value is not a canary, on the other hand, then it satisfies the constraint by satisfying the negation of the antecedent of the constraint (since $A \rightarrow B$ is equivalent to $\neg A \vee B$). We can pursue this reasoning through the entire feature structure: Whenever we see a node that is not of sort *canary*, it satisfies the negation of the antecedent of the constraint. Whenever we see a node that is of sort *canary*, the consequent must hold for the node to satisfy the constraint: The canary must be yellow. In short, the feature structure is only admitted by the constraint if all its component canaries are yellow, which is the effect that we wanted.

Feature structure *admission* is not the only terminology for the function that we have just described. Alternatively, it is often said that a feature structure is *licensed* by a constraint if it is admitted by the constraint. Another way of expressing the same fact is to say that the feature structure *models* the constraint. Generalizing this way of speaking, saying that a set of feature structures *models* a (set of) constraint(s) means that every

feature structure in the set is admitted by each constraint in the constraint set: Each node in each feature structure satisfies each constraint. That is precisely the effect that we want to achieve with sets of grammatical principles, which—together with a signature—constitute a grammar.

We are now ready to define simple abstract feature structures, constraint satisfaction relative to simple abstract feature structures and simple abstract feature structure admission in the next course section. However, before we proceed we should look at a few examples of satisfaction and admission.

Examples for Satisfaction and Admission

The file satis-admis2223 contains our familiar initial signature, generating a description language in which we can talk about lists of animals, birds and pets. It also contains a number of interpretations and three initial descriptions. In what follows we discuss which interpretations satisfy or model which description in these examples.

The description card *canaries 1* contains a description which says:

> If something is a canary then its color is yellow.

The description in *canaries 2* says:

> It is a canary, and its color is yellow.

The feature structure in the interpretation *canary interpretation 1* satisfies the constraint *canaries 1*. It is a feature structure with a root node of sort *four* and no attribute arcs. It must not have arcs, because no attributes are appropriate to *four*. Thus, it is a well-formed feature structure. It satisfies the initial description in *canaries 1*, because it is not a canary at all, thus satisfying the negation of the antecedent. Since the feature structure consists of one node, satisfaction also implies that it models the constraint.

The feature structure in the interpretation *canary interpretation 1* does not satisfy the initial description on the description card *canaries 2*, because it is not a yellow canary. Since it does not satisfy that description, it cannot model it either.

The feature structure in *canary interpretation 2* is a well-formed feature structure which obeys all requirements of the signature: Canaries must have two legs and some color. Moreover, it satisfies the initial description in *canaries 1*, because it satisfies the antecedent of the implication **and** it satisfies its consequent: The canary is yellow. It also models that initial description: All *canary* nodes in the feature structure have a COLOR arc pointing to a node labeled *yellow*.

The feature structure in *canary interpretation 2* satisfies the description on the description card *canaries 2*: It represents a yellow canary. However, it does not model that initial description. The reason is that this yellow canary contains components that are not yellow canaries themselves, namely the nodes labeled *yellow* and *two*. Under admission, the conjunctive description requires by its first conjunct that all nodes be labeled *canary* (which is impossible, by virtue of the appropriateness function of the signature!).

canary interpretation 3 contains again a well-formed feature structure, a list with a red canary on it. And it also satisfies the initial description in *canaries 1*! Why? After all, it is a *red* canary that is on the list. The reason is that the feature structure as a whole is of sort *nelist*, thus satisfying the negation of the antecedent of the description. That is all that is needed for satisfaction. However, the simple feature structure in *canary interpretation 3* is not a model of the initial description in *canaries 1*. Upon pressing the button *Check Modeling*, a black circle appears around the canary node. This black circle indicates that the node of sort *canary* does not satisfy the description. Remember that for admission (which is another expression for modeling and for licensing), **all** nodes of a feature structure must satisfy the constraint(s). In this case, there is exactly one node that does not satisfy the description, namely the node of sort *canary*, because it has a COLOR arc with a node of sort *red* as value. Thus the *canary* node satisfies the antecedent of the constraint but not the consequent, and thus fails to satisfy the constraint as a whole.

The feature structure in *canary interpretation 3* neither satisfies nor models the initial description of the card *canaries 2*. It does not satisfy it, because as a list it is not a canary, as the first conjunct of the description requires. And since the feature structure fails to satisfy the description, it does not model it either: With the root node, we have already found one node that does not satisfy it, and for modeling all nodes would have to satisfy it.

The initial description in the description card *list* says:

> It is a list whose first element is a dog.

The simple feature structure in *list interpretation 1* satisfies this initial description. It is a well-formed feature structure with all required arcs and the required arc values. It represents a list with one element, the element being a brown dog (an arbitrary color is required by the signature) with four legs (as required by the signature). Finally, the *nelist* node has the two required arcs with appropriate values.

The feature structure in *list interpretation 1* does not model the description. Why? Upon pressing the button *Check modeling*, the black circles that appear around some nodes on the canvas indicate which nodes do not satisfy the description. It is all nodes except for the root node. They cannot satisfy the description, because they are not of sort *nelist*.

The crucial difference to the description on the description card *canaries 1* is that the present description is not an implication. With implications, we use admission to express restrictions on all those nodes in feature structures that satisfy the antecedent. With a conjunctive description like the present one, admission requires that all nodes in the feature structure satisfy the entire conjunctive description. Since the present description describes a non-empty list (by virtue of its first conjunct), no feature structure at all can possibly be admitted by it under the given signature. The reason is that our signature requires that a list contain animals, and animals are not lists. But for a feature structure to be admitted by the present description, it would have to consist exclusively of nodes of sort *nelist*, due to its first conjunct.

list interpretation 2 contains another feature structure satisfying but not modeling the description. With the *Check Modeling* function, MoMo is prompted to show which nodes do not satisfy the initial description of the description card *list*. The root node satisfies it, since all the description requires is that it be a list, and its first element be a dog. That does not say anything about the length of the list, and it does not say anything about the other elements on it, if there are any. The only restriction about them comes from the signature, which requires that they all be animals. The root node is the only node that satisfies the initial description of *list*, since no other node in the entire feature structure is a node of sort *nelist* with a HEAD arc pointing to a node labeled *dog*.

Exercises

Again we take the signature introduced in Section 2.1.2, *Signatures: Partitions and Feature Declarations*, and the well-formed descriptions of Exercise 3 of the same section.

To answer Exercise 15, please create a file `admission.mmp` and upload it to your group in ILIAS.

Exercise 15 *Are there any well-formed descriptions that admit a feature structure? If so, name at least one of them and give an example of a feature structure admitted by it.*

Exercise 16 *From the exercises of the last section we already know that description (8) is satisfiable, and we have drawn one of the feature structures that satisfy it. Give a short reasoning why there are no models of that description. (This can be done in three or four sentences!)*

Hint: You may want to play with the modeling function of MoMo to get some feedback on your ideas when you think about this question.

GLOSSARY

2.2.2.4 Formalization

ABSTRACT

We will define totally well-typed and sort-resolved abstract feature structures, show how they correspond to the concrete feature structures of MoMo, and use them as the modeling domain of initial grammars. For that purpose we will first define a satisfaction function relating initial descriptions to sets of abstract feature structures. Building on the satisfaction function, we will then define an admission function that relates sets of initial descriptions to sets of abstract feature structures.

Our first understanding of the meaning of descriptions and grammars in MoMo was based on an informal notion of totally well-typed and sort-resolved feature structures that can be extracted from Pollard and Sag's book. Following linguistic tradition, we described feature structures based on finite state automata. Due to their tangible nature, this kind of feature

structure is sometimes called *concrete* feature structure. To be able to define the meaning of initial grammars with respect to feature structures, it is necessary to go beyond our first informal characterization and to give an exact definition of feature structures. This is what we will do in this section.

Guided by the remarks on feature structures and on the explanation of the meaning of grammars in [Pollard and Sag, 1994] (see Section 2.2.2, *Meaning*), we define them as totally-well typed, sort-resolved and *abstract* feature structures. We call our new feature structures *abstract* because they are built in such a way that two structurally isomorphic feature structures are necessarily identical, which captures one of the intuitions underlying Pollard and Sag's concept of feature structure models: The members of the set of feature structures admitted by a grammar are mathematical representations of the object types of a language. On the other hand, the feature structures pictured in MoMo correspond to so-called *concrete* feature structures: Two structurally identical concrete feature structures may well be different entities.[3]

What is the idea behind abstract feature structures? The construction that we use (and the name for these feature structures) was first introduced to linguistics by [Moshier, 1988]. It starts from the insight that if we want to force two feature structures of the same shape to be always identical, then they should not have individual nodes. As soon as there are individual nodes we can always choose other nodes as the nodes of the next feature structure of the same shape, and we will get two feature structures that look the same. We can easily imagine this situation as the one that we have in MoMo when we draw feature structures. Nothing can stop us from drawing two non-identical feature structures of the same shape on the canvas, since we can always choose to place more nodes and arcs from a potentially infinite resource on the canvas.

The alternative to individual nodes is to use equivalence classes of appropriate mathematical entities to represent what we thought of as nodes before. In order to appeal to our intuitions and to our practical experience with concrete feature structures in MoMo, let us call these equivalence classes *abstract nodes*, in analogy to the "abstract feature structures" that we define using abstract nodes.

Before we give this further thought, let us remind ourselves what equivalence classes are. First of all, they build on the notion of equivalence relations. Equivalence relations are relations with three important properties: They are reflexive, transitive, and symmetric. We can formulate that as follows:

Definition 5 *η is an equivalence relation iff*

η is an ordered pair, $\langle S, \circ \rangle$,
S is a set,

[3] Concrete feature structures are thus closer to the concept of utterance tokens and to King's idea of the meaning of grammars in terms of collections of possible utterance tokens. However, there are some philosophically important differences, since concrete feature structures are defined as mathematical objects, and King envisions natural languages in the denotation of grammars. Natural languages are presumably not mathematical objects.

$\circ$ *is a binary relation on* S,
for each $x \in S$, $x \circ x$ *(*$\circ$ *is reflexive),*
for each $x \in S$, *for each* $y \in S$, *for each* $z \in S$,
  if $x \circ y$ *and* $y \circ z$ *then* $x \circ z$ *(*$\circ$ *is transitive),*
for each $x \in S$, *for each* $y \in S$, *if* $x \circ y$ *then* $y \circ x$ *(*$\circ$ *is symmetric).*

We can then define an equivalence class of an equivalence relation $\eta = \langle S, \circ \rangle$ as a non-empty subset S' of S such that for each $x \in S'$ and each $y \in S'$, $x \circ y$, but for no element z of $S \backslash S'$ and any $x \in S'$, $z \circ x$. Equivalence classes of equivalence relations have some properties that are very interesting for our purpose. Given an equivalence relation, $\eta = \langle S, \circ \rangle$, the union of all its equivalence classes equals the carrier set of η, S. On the other hand, any equivalence class of η can be uniquely identified by one of its elements. Assume that S' is an equivalence class of η. Then the last property of equivalence classes allows us to pick any element of S', say x, and write $|x|_{\langle S,\circ \rangle}$ for the equivalence class of η to which x belongs. That means, for any $x \in S'$, we get $S' = |x|_{\langle S,\circ \rangle}$.

The key idea of abstract feature structures is to take the sequences of attributes that label the arcs that we need to traverse in concrete feature structures from the root node to a given node as the representations of the abstract nodes. To simplify our terminology, let us call a (possibly empty) sequence of attributes a *path*. An abstract node ν in an abstract feature structure is then an equivalence class of paths that we could travel along on from the root node to ν in its concrete counterpart. The property of unique identifiability of equivalence classes discussed above will now give us a way to identify an abstract node of an abstract feature structure by any of the paths that lead to its concrete counterpart.

It is time for a few simple examples. To avoid getting too used to a single signature, we will take the signature in Figure 2.2 for the examples below.

Note that in contrast to the signature that we used before, *animal* and its subsorts do not have any attributes appropriate to them in the signature of Figure 2.2. The new signature is designed to talk about what people like best, it distinguishes between men and women, and it may be used for talking about two kinds of cars and their owners and drivers. You find the concrete feature structures of our examples in the mmp-file animalscars.mmp, and we recommend looking at those feature structures while working through the examples.

Take the concrete feature structure consisting of a single node labeled *penguin*. How could we represent the single node of this concrete feature structure in an abstract feature structure? There is exactly one possibility to travel from the root node of this concrete feature structure to its root node, namely not to travel at all. We could call this trivial case the empty traversal from one node to another, and we write ε for this special case. Our new representation of the single node of the concrete feature structure is then the abstract node where ε is in the equivalence relation with itself. Call the equivalence relation ϱ, and its carrier set (which we will characterize later) β. Then we may take $|\varepsilon|_{\langle \beta,\varrho \rangle}$ to be the representation of the abstract node.

Let's look at a less trivial case and consider a concrete feature structure representing the fact that a dog is a man's best friend. The feature structure has a root node labeled *man*, a second node labeled *dog* and an arc labeled LIKES-BEST from the first node to the second.

```
type_hierarchy
top
   car owner:person driver:person
      vw
      bmw
   person likes-best:top
      man
      woman
   animal
      pet
         dog
         cat
      bird
         canary
         parrot
         penguin
.
```

Figure 2.2: A signature with people, cars and animals

As before, the abstract root node will be represented as $|\varepsilon|_{\langle\beta,\varrho\rangle}$, since the empty path is the only possibility to get from the root node to the root node. But for the second node, the situation is also quite simple: Following the LIKES-BEST arc from the root node is the only way to get to it. Therefore, we represent it by the equivalence class $|\text{LIKES-BEST}|_{\langle\beta,\varrho\rangle}$

Let's make the feature structure even more complex. Take a VW whose owner is a woman and whose driver is a man. The man and the woman are each other's best friends. The reason that this case is more tricky than the previous one is not just that a feature structure representing the depicted situation has three nodes. It also introduces a cyclic sequence of arcs, meaning that for some nodes there is a sequence of arcs that allows us to always return to a node that we started from. The root node is of sort *vw*, and it has two outgoing arcs. The one labeled OWNER leads to a node of sort *woman*, and the one labeled DRIVER leads to a node labeled *man*. From the node of sort *woman* and from the node of sort *man* there is a LIKES-BEST arc leading to the other node. Starting from the root node along the OWNER arc, we can thus return to the *woman* node by traversing LIKES-BEST arcs any even number of times. Similarly, starting from the root node along the DRIVER arc, we return to the *man* node by following LIKES-BEST arcs any even number of times. Following LIKES-BEST arcs an odd number of times from the *man* node or the *woman* node will always get us to the other node.

Representing the abstract nodes in the abstract feature structure is now a bit more interesting than before. For the root node, we keep $|\varepsilon|_{\langle\beta,\varrho\rangle}$. The representation of the *woman* node can be given by any path that leads to its concrete counterpart as before, and $|\text{OWNER}|_{\langle\beta,\varrho\rangle}$ indicates the relevant equivalence class. Note, however, that this equivalence

class is now an infinite set, since there is an infinite number of ways to get from the root node to the *woman* node. Of course, the same reasoning applies to the *man* node, whose equivalence class can be given by $|\textsc{driver}|_{\langle\beta,\varrho\rangle}$.

The so-termed infinite feature structures are another interesting case. A feature structure is called infinite if it has infinitely many nodes. Imagine the situation where every woman likes some other woman best without ever running into a cycle. We get an infinite concrete feature structure, because we have infinitely many nodes labeled *woman*, and we traverse from each node to another one by following an arc labeled LIKES-BEST. Since the set of nodes is infinite, we cannot picture this feature structure in MoMo, because we can never finish drawing it (moreover our computing resources are still finite, preventing us from obtaining enough computer memory to store an electronic representation of an infinitely large picture with infinitely many, physically distinct nodes). However, for each concrete node we can still name its abstract representation: It is the equivalence class of $\langle\beta, \varrho\rangle$ named by a sequence of LIKES-BEST attributes that is as long as the distance of the respective node from the root node measured in the number of traversals of LIKES-BEST arcs.

Looking at the representations of the abstract nodes in the previous examples one may already see what the carrier set of the equivalence relations for the representations of abstract nodes will be. Clearly, for any given feature structure, we do not need the entire range of possible paths that we can obtain from the signature. We only need those that actually occur in the respective feature structures as sequences of arcs that we can traverse starting from the root node. In our first example, this was only the empty path, which we represented as ε. In the second example, the path LIKES-BEST was added. The third example was the first to comprise an infinite set of paths, due to its cyclicity: All paths (with the exception of the empty path) either started with the DRIVER or the OWNER attribute, followed by any finite sequence of LIKES-BEST attributes.

With the examples of what we want to do in mind, we can now define a first version of abstract feature structures. To do this, we must define the carrier sets and the equivalence relations that are supposed to represent our abstract nodes; we must build in a labeling function for the abstract nodes in order to assign exactly one species to each abstract node; and we must make sure that our abstract feature structures are totally well-typed as required by our target formalism. For convenience, we introduce a special notation for the set of paths generated by a given signature. If S is a set (like our sets of attributes), we will write S^* for the set of sequences of symbols from S. For a set of attributes, $\mathcal{A}$, the notation $\mathcal{A}^*$ thus stands for the set of paths that we can construct from the attributes, including the empty path.

DEFINITION 6 introduces our abstract feature structures as *simple* abstract feature structures. They are called *simple*, because they will ultimately have to be augmented along with the syntax of the description language in order to achieve a complete formalization of a feature structure-based interpretation of HPSG.

Definition 6 *For each initial signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F} \rangle$, $\mathbb{A}$ *is a* ***simple abstract feature structure under*** Σ *iff*

$\mathbb{A}$ *is a triple* $\langle \beta, \varrho, \lambda \rangle$,
$\beta \subseteq \mathcal{A}^*$,
$\varepsilon \in \beta$,
for each $\pi \in \mathcal{A}^*$, *for each* $\alpha \in \mathcal{A}$,
    if $\pi\alpha \in \beta$ *then* $\pi \in \beta$,
ϱ *is an equivalence relation over* β,
for each $\pi_1 \in \mathcal{A}^*$, *for each* $\pi_2 \in \mathcal{A}^*$, *for each* $\alpha \in \mathcal{A}$,
    if $\pi_1\alpha \in \beta$ *and* $\langle \pi_1, \pi_2 \rangle \in \varrho$ *then* $\langle \pi_1\alpha, \pi_2\alpha \rangle \in \varrho$,
λ *is a total function from* β *to* $\mathcal{S}$, *and*
for each $\pi_1 \in \mathcal{A}^*$, *for each* $\pi_2 \in \mathcal{A}^*$,
    if $\langle \pi_1, \pi_2 \rangle \in \varrho$ *then* $\lambda(\pi_1) = \lambda(\pi_2)$.
for each $\pi \in \mathcal{A}^*$, *for each* $\alpha \in \mathcal{A}$,
    if $\pi\alpha \in \beta$ *then* $\mathcal{F}(\langle \lambda(\pi), \alpha \rangle)$ *is defined and* $\lambda(\pi\alpha) \sqsubseteq \mathcal{F}(\langle \lambda(\pi), \alpha \rangle)$,
for each $\pi \in \mathcal{A}^*$, *for each* $\alpha \in \mathcal{A}$,
    if $\pi \in \beta$ *and* $\mathcal{F}(\langle \lambda(\pi), \alpha \rangle)$ *is defined then* $\pi\alpha \in \beta$,

If Σ is an initial signature, we write $\mathbb{AFS}^{\Sigma}$ for the *set of simple abstract feature structures under* Σ. If $\mathbb{A} = \langle \beta, \varrho, \lambda \rangle$, we call β the *basis set in* $\mathbb{A}$, ϱ the *re-entrancy relation in* $\mathbb{A}$, and λ the *label function in* $\mathbb{A}$. The basis set in $\mathbb{A}$ is a prefix closed set of paths, the re-entrancy relation in $\mathbb{A}$ is a right invariant equivalence relation on the basis set in $\mathbb{A}$, and the label function in $\mathbb{A}$ is a total function from the basis set in $\mathbb{A}$ to the set of species, $\mathcal{S}$, which respects the re-entrancy relation in $\mathbb{A}$. In other words, if two paths in β are re-entrant then they are assigned the same species.

For each simple abstract feature structure $\langle \beta, \varrho, \lambda \rangle$, $\langle \beta, \varrho \rangle$ is the equivalence relation that we use to represent the abstract nodes. As discussed for our examples, the carrier set is a subset of the set of paths generated by the signature (line 4), always including the empty path (line 5). The set of paths in the basis set is prefixed closed, because if we can get to a node by some path, then we get to some other node by any shorter path that we get by chopping away the last attribute of the path (lines 6 and 7). In line 8 we require that ϱ really be an equivalence relation. Lines 9 and 10 enforce another property of ϱ that we see in concrete feature structures: If two paths lead to the same node, then traversing the same

arc after traversing those two paths necessarily leads to the same node again. Otherwise our representation of simple abstract feature structures would not match our intuitions. Lines 11–13 introduce the sort labeling of our abstract nodes by assigning each path a species. Naturally, paths that lead to the same abstract node, i.e., paths that stand in the re-entrancy relation must be assigned the same species (lines 12–13). Finally, we have to make sure that the sort labeling obeys the appropriateness conditions of the signature (lines 14 and 15), and our simple abstract feature structures are totally well-typed (lines 16 and 17).

We can now repeat our previous examples by giving the full mathematical representations of the simple abstract feature structures that correspond to the concrete feature structures that we saw in MoMo, and the one that we were not able to draw in MoMo because it was infinite:

1. $\mathbb{A}_0 = \langle \beta_0, \varrho_0, \lambda_0 \rangle$, where
   $\beta_0 = \{\varepsilon\}$,
   $\varrho_0 = \{\langle \varepsilon, \varepsilon \rangle\}$, and
   $\lambda_0 = \{\langle \varepsilon, \mathit{penguin} \rangle\}$,

2. $\mathbb{A}_1 = \langle \beta_1, \varrho_1, \lambda_1 \rangle$, where
   $\beta_1 = \{\varepsilon, \text{LIKES-BEST}\}$,
   $\varrho_1 = \{\langle \varepsilon, \varepsilon \rangle, \langle \text{LIKES-BEST}, \text{LIKES-BEST} \rangle\}$, and
   $\lambda_1 = \{\langle \varepsilon, \mathit{man} \rangle, \langle \text{LIKES-BEST}, \mathit{dog} \rangle\}$.

3. First we state the infinite sets that represent the non-root nodes. Let
   $$n_2 = \left\{ \text{OWNER } \underbrace{\text{LIKES-BEST} \ldots \text{LIKES-BEST}}_{2*n \text{ times}} \,\middle|\, n \in \mathbb{N} \right\} \cup \left\{ \text{DRIVER } \underbrace{\text{LIKES-BEST} \ldots \text{LIKES-BEST}}_{n \text{ times}} \,\middle|\, \begin{array}{l} n \text{ is an odd} \\ \text{natural number} \end{array} \right\} \text{ and}$$
   $$n_3 = \left\{ \text{DRIVER } \underbrace{\text{LIKES-BEST} \ldots \text{LIKES-BEST}}_{2*n \text{ times}} \,\middle|\, n \in \mathbb{N} \right\} \cup \left\{ \text{OWNER } \underbrace{\text{LIKES-BEST} \ldots \text{LIKES-BEST}}_{n \text{ times}} \,\middle|\, \begin{array}{l} n \text{ is an odd} \\ \text{natural number} \end{array} \right\}.$$
   With the sets n_2 and n_3, we define $\mathbb{A}_2$ as follows:
   $\mathbb{A}_2 = \langle \beta_2, \varrho_2, \lambda_2 \rangle$, where
   $$\beta_2 = \{\varepsilon\} \cup \left\{ \text{OWNER } \underbrace{\text{LIKES-BEST} \ldots \text{LIKES-BEST}}_{n \text{ times}} \,\middle|\, n \in \mathbb{N} \right\} \cup \left\{ \text{DRIVER } \underbrace{\text{LIKES-BEST} \ldots \text{LIKES-BEST}}_{n \text{ times}} \,\middle|\, n \in \mathbb{N} \right\},$$
   $$\varrho_2 = \{\langle \varepsilon, \varepsilon \rangle\} \cup \left\{ \langle \pi_1, \pi_2 \rangle \,\middle|\, \pi_1 \in n_3, \text{ and } \pi_2 \in n_3 \right\} \cup \left\{ \langle \pi_1, \pi_2 \rangle \,\middle|\, \pi_1 \in n_2, \text{ and } \pi_2 \in n_2 \right\}, \text{ and}$$

$$\lambda_2 = \{\langle \varepsilon, vw \rangle\} \cup \Big\{ \langle \pi, woman \rangle \,\Big|\, \pi \in n_2 \Big\} \cup \Big\{ \langle \pi, man \rangle \,\Big|\, \pi \in n_3 \Big\}.$$

4. $\mathbb{A}_3 = \langle \beta_3, \varrho_3, \lambda_3 \rangle$, where

$$\beta_3 = \Big\{ \underbrace{\text{LIKES-BEST} \ldots \text{LIKES-BEST}}_{n \ \ times} \,\Big|\, n \in \mathbb{N} \Big\},$$

$$\varrho_3 = \Big\{ \langle \pi, \pi \rangle \,\Big|\, \pi \in \beta_3 \Big\}, \text{ and}$$

$$\lambda_3 = \Big\{ \langle \pi, woman \rangle \,\Big|\, \pi \in \beta_3 \Big\}.$$

Whereas the nodes of concrete feature structures are real objects that are accessible from the root node by following sequences of arcs labeled by attributes, the attribute paths obtained from the labels of the sequences of arcs take over the role of representing the formerly concrete nodes in abstract feature structures. The basis set in $\mathbb{A}$ specifies the shape of $\mathbb{A}$, because it specifies which paths belong to $\mathbb{A}$. The re-entrancy relation in $\mathbb{A}$ specifies the set of abstract nodes of $\mathbb{A}$ as the set of equivalence classes of the equivalence relation $\langle \beta, \varrho \rangle$. Each equivalence class represents an abstract node. Since the label function in $\mathbb{A}$ assigns each path in β that belongs to the same equivalence class in $\langle \beta, \varrho \rangle$ the same sort, it effectively assigns each abstract node a sort label.

Now we know our modeling domain of simple abstract feature structures. All we need to do to complete our formalization of the meaning of initial grammars is to define a constraint satisfaction function and an admission function from initial descriptions to sets of simple abstract feature structures. Before we can state the *simple abstract feature structure satisfaction function*, we need to introduce the auxiliary notion of reducts of simple abstract feature structures. The reason for that is that with statements like `owner:man` we say something about non-root nodes. About the root node, we say that it has an outgoing OWNER arc. About a particular non-root node, namely the one we reach following the OWNER arc, we say that it is of sort *man*. Metaphorically speaking, reducts of simple abstract feature structures allow us to traverse a sequence of arcs and treat the node that we reach that way as a new root node. We can then see whether that root node satisfies a given constraint. For example, taking the second simple abstract feature structure from above (the one representing the fact that a man's best friend is a dog), the LIKES-BEST reduct of the feature structure is the simple abstract feature structure with the abstract root node of sort *dog* (and no outgoing arcs).

Definition 7 *For each initial signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F} \rangle$*, for each* $\mathbb{A} = \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^{\Sigma}$*, for each* $\pi \in \mathcal{A}^*$*,*

$$\beta/\pi = \{\pi' \in \mathcal{A}^* \mid \pi\pi' \in \beta\},$$

$$\varrho/\pi = \{\langle \pi_1, \pi_2 \rangle \in \mathcal{A}^* \times \mathcal{A}^* \mid \langle \pi\pi_1, \pi\pi_2 \rangle \in \varrho\},$$

$$\lambda/\pi = \{\langle \pi', \sigma \rangle \in \mathcal{A}^* \times \mathcal{S} \mid \langle \pi\pi', \sigma \rangle \in \lambda\}, \text{ and}$$

$$\mathbb{A}/\pi = \langle \beta/\pi, \varrho/\pi, \lambda/\pi \rangle.$$

We observe that if a Σ path π is in the basis set of a simple abstract feature structure under Σ, $\mathbb{A}$, then the π reduct of $\mathbb{A}$ is also a simple abstract feature structure under Σ. This is important enough to keep it in mind as a proposition:

Proposition 1 *For each initial signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F} \rangle$, *for each* $\mathbb{A} = \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^{\Sigma}$, *for each* $\pi \in \mathcal{A}^*$,

if $\pi \in \beta$ *then* $\mathbb{A}/\pi \in \mathbb{AFS}^{\Sigma}$.

After all these lengthy preliminaries, we finally get to a first rigorous definition of constraint satisfaction. The simple abstract feature structure satisfaction function in Σ, Δ_{Σ}, is defined inductively over all initial Σ descriptions:

Definition 8 *For each initial signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F} \rangle$, Δ_{Σ} *is the total function from* $\mathcal{D}^{\Sigma}$ *to* $Pow(\mathbb{AFS}^{\Sigma})$ *such that*

for each $\sigma \in \mathcal{G}$,

$$\Delta_{\Sigma}(\sigma) = \left\{ \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^{\Sigma} \;\middle|\; \begin{array}{l} \textit{for some } \sigma' \in \mathcal{S}, \\ \quad \langle \varepsilon, \sigma' \rangle \in \lambda, \textit{ and} \\ \quad \sigma' \sqsubseteq \sigma \end{array} \right\},$$

for each $\alpha \in \mathcal{A}$, *for each* $\delta \in \mathcal{D}^{\Sigma}$,

$$\Delta_{\Sigma}(\alpha : \delta) = \left\{ \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^{\Sigma} \;\middle|\; \begin{array}{l} \alpha \in \beta, \textit{ and} \\ \langle \beta, \varrho, \lambda \rangle / \alpha \in \Delta_{\Sigma}(\delta) \end{array} \right\},$$

for each $\delta \in \mathcal{D}^{\Sigma}$, $\Delta_{\Sigma}(\sim \delta) = \mathbb{AFS}^{\Sigma} \backslash \Delta_{\Sigma}(\delta)$,

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$, $\Delta_{\Sigma}((\delta_1, \delta_2)) = \Delta_{\Sigma}(\delta_1) \cap \Delta_{\Sigma}(\delta_2)$,

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$, $\Delta_{\Sigma}((\delta_1; \delta_2)) = \Delta_{\Sigma}(\delta_1) \cup \Delta_{\Sigma}(\delta_2)$,

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$, $\Delta_{\Sigma}((\delta_1 \mathbin{*>} \delta_2)) = \left(\mathbb{AFS}^{\Sigma} \backslash \Delta_{\Sigma}(\delta_1)\right) \cup \Delta_{\Sigma}(\delta_2)$,
and

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$,
$\Delta_{\Sigma}((\delta_1 \mathbin{<*>} \delta_2)) = \left(\left(\mathbb{AFS}^{\Sigma} \backslash \Delta_{\Sigma}(\delta_1)\right) \cap \left(\mathbb{AFS}^{\Sigma} \backslash \Delta_{\Sigma}(\delta_2)\right)\right) \cup (\Delta_{\Sigma}(\delta_1) \cap \Delta_{\Sigma}(\delta_2))$.

We call Δ_{Σ} the *simple abstract feature satisfaction function under* Σ, and say $\mathbb{A}$ satisfies δ in Σ if and only if $\mathbb{A} \in \Delta_{\Sigma}(\delta)$. The power set of a set S, which we write as $Pow(S)$, is the set of all subsets of S.[4]

Although the definition may look complicated at first, it does not introduce anything that we have not thoroughly discussed yet. It is really only a way to make our intuitions

[4]The power set of $\{1, 2\}$ is thus the set $\{\{\}, \{1\}, \{2\}, \{1, 2\}\}$. Note that the empty set is contained in the power set of any set.

about constraint satisfaction mathematically precise. The first clause says that for any sort σ, σ is satisfied by any simple abstract feature structure whose abstract root node is labeled by a maximal subsort of σ (or σ itself, in case σ is a species). The second clause, $\alpha : \delta$, is the reason we needed reducts of simple abstract feature structures: $\alpha : \delta$, where α is an attribute and δ any initial description, is satisfied by each simple abstract feature structure whose abstract root node has an α arc and whose α reduct satisfies the initial description δ.[5]

The interpretation of the logical symbols of our languages follows standard classical logic: The negation of δ is satisfied by all simple abstract feature structures that do not satisfy δ (set complement); the conjunction of two initial descriptions is satisfied by all simple abstract feature structures that satisfy both of them; and the disjunction of two initial descriptions is the set union of the simple abstract feature structures that satisfy them; and $\delta_1 *> \delta_2$ is the set complement of the simple abstract feature structures that satisfy δ_1 merged with the set of simple abstract feature structures that satisfy δ_2. The denotation of equivalence may look difficult, but it can be derived from the denotation of the other logical symbols in the usual way.

With *simple abstract feature structure admission* we want to make precise what it means for a set of simple abstract feature structures to be admitted by a set of initial descriptions. We can characterize our intentions as follows: The set of simple abstract feature structures admitted by a set of initial descriptions, θ, should be the set of simple abstract feature structures such that each one of them satisfies each initial description contained in θ; and each possible reduct of each one of them also satisfies each element of θ. The second condition can be rephrased as: each abstract node within each of the simple abstract feature structures also satisfies each initial description in θ. DEFINITION 9 expresses this idea:

Definition 9 *For each initial signature* Σ,

M_Σ *is the total function from* $Pow(\mathcal{D}^\Sigma)$ *to* $Pow(\mathbb{AFS}^\Sigma)$ *such that for each* $\theta \subseteq \mathcal{D}^\Sigma$,

$$M_\Sigma(\theta) = \left\{ \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^\Sigma \;\middle|\; \begin{array}{l} \textit{for each } \pi \in \beta, \textit{ and} \\ \textit{for each } \delta \in \theta, \\ \quad \langle \beta, \varrho, \lambda \rangle / \pi \in \Delta_\Sigma(\delta) \end{array} \right\}.$$

For each initial signature Σ, we call M_Σ the *simple abstract feature structure admission function in* Σ. A simple abstract feature structure under Σ, $\mathbb{A}$, is in the set of simple abstract feature structures under Σ admitted by a Σ theory θ exactly if $\mathbb{A}$ and every possible π reduct of $\mathbb{A}$ satisfy every description in θ.

Given an initial grammar, $\langle \Sigma, \theta \rangle$, we can regard the set of feature structures admitted by θ, $M_\Sigma(\theta)$ as the intended meaning of $\langle \Sigma, \theta \rangle$. Pollard and Sag say that the members

[5] This is where PROPOSITION 1 is important, because it guarantees the existence of the α reduct of the relevant simple abstract feature structures.

of $M_\Sigma(\theta)$ are mathematical representations of the object types of the language that the grammar is about.

Abstract feature structures are clearly harder to understand at first than concrete feature structures. If you are not quite sure about them, it is very helpful and perfectly legitimate to picture them in your mind as concrete feature structures and work with intuitions about them. The intuitions about their correspondence will rarely fail.

Examples

To acquire better intuition about simple abstract feature structures and their reducts, we will provide a few examples. The underlying signature is the signature of lists and animals (birds and pets) of Section 2.1.2. We start with $\mathbb{A}_0$, a list of length one, whose single element is a brown dog. $\mathbb{A}_1$ is the TAIL reduct of $\mathbb{A}_0$, which we can write as $\mathbb{A}_1 = \mathbb{A}_0/\textsc{tail}$. It is a single abstract root node of sort *elist*. $\mathbb{A}_2$ is the HEAD reduct of $\mathbb{A}_0$, which we can write as $\mathbb{A}_2 = \mathbb{A}_0/\textsc{head}$. It is a simple abstract feature structure representing a brown dog.

$\mathbb{A}_3$ is a rather odd case: It is a list whose HEAD value is a brown dog and whose TAIL value is the list itself. In other words, it is a cyclic list. If you are not careful about your grammars, you can get them rather easily in the set of abstract feature structures admitted by the grammar. The TAIL reduct of $\mathbb{A}_3$ equals $\mathbb{A}_3$, whereas its HEAD reduct equals the HEAD reduct of $\mathbb{A}_0$.

1. $\mathbb{A}_0 = \langle \beta_0, \varrho_0, \lambda_0 \rangle$, where
   $\beta_0 = \{\varepsilon, \textsc{tail}, \textsc{head}, \textsc{head color}, \textsc{head legs}\}$,
   $\varrho_0 = \left\{ \begin{array}{l} \langle \varepsilon, \varepsilon \rangle, \langle \textsc{tail}, \textsc{tail} \rangle, \langle \textsc{head}, \textsc{head} \rangle, \langle \textsc{head color}, \textsc{head color} \rangle, \\ \langle \textsc{head legs}, \textsc{head legs} \rangle \end{array} \right\}$, and
   $\lambda_0 = \left\{ \begin{array}{l} \langle \varepsilon, \mathit{nelist} \rangle, \langle \textsc{tail}, \mathit{elist} \rangle, \langle \textsc{head}, \mathit{dog} \rangle, \langle \textsc{head color}, \mathit{brown} \rangle, \\ \langle \textsc{head legs}, \mathit{four} \rangle \end{array} \right\}$.
2. $\mathbb{A}_0/\textsc{tail} = \mathbb{A}_1 = \langle \beta_1, \varrho_1, \lambda_1 \rangle$, where
   $\beta_1 = \{\varepsilon\}$,
   $\varrho_1 = \{\langle \varepsilon, \varepsilon \rangle\}$, and
   $\lambda_1 = \{\langle \varepsilon, \mathit{elist} \rangle\}$.
3. $\mathbb{A}_0/\textsc{head} = \mathbb{A}_2 = \langle \beta_2, \varrho_2, \lambda_2 \rangle$, where
   $\beta_2 = \{\varepsilon, \textsc{color}, \textsc{legs}\}$,
   $\varrho_2 = \{\langle \varepsilon, \varepsilon \rangle, \langle \textsc{color}, \textsc{color} \rangle, \langle \textsc{legs}, \textsc{legs} \rangle\}$, and
   $\lambda_2 = \{\langle \varepsilon, \mathit{dog} \rangle, \langle \textsc{color}, \mathit{brown} \rangle, \langle \textsc{legs}, \mathit{four} \rangle\}$.
4. $\mathbb{A}_3 = \langle \beta_3, \varrho_3, \lambda_3 \rangle$, where
   $\beta_3 = \{\textsc{tail}\}^* \cup \left\{ \pi\textsc{head} \,\middle|\, \pi \in \{\textsc{tail}\}^* \right\} \cup \left\{ \pi\textsc{head color} \,\middle|\, \pi \in \{\textsc{tail}\}^* \right\} \cup$
   $\left\{ \pi\textsc{head legs} \,\middle|\, \pi \in \{\textsc{tail}\}^* \right\}$,

$$\varrho_3 = \Big\{ \langle \pi_1, \pi_2 \rangle \,\Big|\, \pi_1 \in \{\textsc{tail}\}^*, \pi_2 \in \{\textsc{tail}\}^* \Big\} \cup \Big\{ \langle \pi_1\textsc{head}, \pi_2\textsc{head} \rangle \,\Big|\, \pi_1 \in \{\textsc{tail}\}^*, \pi_2 \in \{\textsc{tail}\}^* \Big\} \cup \Big\{ \langle \pi_1\textsc{head legs}, \pi_2\textsc{head legs} \rangle \,\Big|\, \pi_1 \in \{\textsc{tail}\}^*, \pi_2 \in \{\textsc{tail}\}^* \Big\} \cup \Big\{ \langle \pi_1\textsc{head color}, \pi_2\textsc{head color} \rangle \,\Big|\, \pi_1 \in \{\textsc{tail}\}^*, \pi_2 \in \{\textsc{tail}\}^* \Big\}$$

$$\lambda_3 = \Big\{ \langle \pi, \mathit{nelist} \rangle \,\Big|\, \pi \in \{\textsc{tail}\}^* \Big\} \cup \Big\{ \langle \pi\textsc{head}, \mathit{dog} \rangle \,\Big|\, \pi \in \{\textsc{tail}\}^* \Big\} \cup \Big\{ \langle \pi\textsc{head color}, \mathit{brown} \rangle \,\Big|\, \pi \in \{\textsc{tail}\}^* \Big\} \cup \Big\{ \langle \pi\textsc{head legs}, \mathit{four} \rangle \,\Big|\, \pi \in \{\textsc{tail}\}^* \Big\}.$$

5. $\mathbb{A}_3/\textsc{tail} = \mathbb{A}_3$
6. $\mathbb{A}_3/\textsc{head} = \mathbb{A}_2 = \mathbb{A}_0/\textsc{head}$

Exercises

Exercise 17 *Submit an mmp file in which you have saved graphs of concrete feature structure counterparts of the six abstract feature structures in the preceding examples.*

Exercise 18 *Try to write up the following simple abstract feature structures, where* $\mathbb{A}_1$, $\mathbb{A}_2$ *and* $\mathbb{A}_3$ *are the examples from pages 55–56, by stating their basis set, their re-entrancy relation and their label function:*

1. $\mathbb{A}_1/\textsc{likes-best}$ *(the* LIKES-BEST *reduct of* $\mathbb{A}_1$*),*
2. $\mathbb{A}_2/\textsc{owner}$ *(the* OWNER *reduct of* $\mathbb{A}_2$*),*
3. $\mathbb{A}_3/\textsc{likes-best}$ *(the* LIKES-BEST *reduct of* $\mathbb{A}_3$*.)*

Hint and remark: It might be useful to start with drawing the corresponding concrete feature structures in MoMo and then to think about their abstract counterparts. If you find these exercises difficult, it would be good if you tried to formulate questions and post them to the seminar forum. Remember that the forum can be the on-line version of classroom discussions!

GLOSSARY

2.3 Complex Grammars and their Meaning

Abstract

We will extend the languages of our initial formalism of a basic feature logic. The new syntax is more complex and closer to the special requirements of HPSG. To interpret the more complex descriptions, we will still be able to use simple abstract feature structures. Once more this demonstrates that the structure of a description language is fairly independent of the structure of the domain of objects in which it is interpreted.

With *initial descriptions*, the notion of *initial grammars* and the interpretation of sets of initial descriptions in a domain of *simple abstract feature structures*, we have faithfully reconstructed the architecture of grammar that [Pollard and Sag, 1994] envisions for linguistic theories. Moreover, we have done it in such a way that we will be able to use the syntax of our descriptions and of the specifications of signatures directly in a computational environment. However, as soon as we try to formalize actual HPSG grammars with our current means, we immediately discover serious shortcomings: Although our general architecture is adequate, the expressiveness of our formal languages is not up to the task. We simply cannot express the grammatical principles of real grammars yet.

For that reason we now turn to an extension of the description languages. The extension will introduce variables and a special form of universal and existential quantification. The extension of the basic formal languages will make it possible to express the identity (or inequality) of substructures in principles like the Head Feature Principle, the Spec Principle and the Control Theory of [Pollard and Sag, 1994], which were beyond the expressive power of our initial formalism. With the additions to the formalism of this section we will, for the first time, be able to formulate real principles of a grammar of English in a mathematically completely explicit way.

To see what is at stake, we start by briefly reviewing the aforementioned principles as they are stated in Pollard and Sag's book, followed by their formulation in an AVM notation. Since we are interested in their formal properties rather than in their empirical predictions at the moment, we will focus on their logical structure and ignore their linguistic significance. You should not be worried if it is not entirely clear to you what these principles do in the theory of English. The original formulations in natural language will give us an idea about the intended meaning of the principles in a domain of abstract feature structures, no matter what exactly they are supposed to correspond to in the empirical domain of the English language. The re-formulation of the principles in some kind of AVM syntax appeals to our prior knowledge of linguistic notation and, at the same time, demonstrates which constructs are missing in our current syntax. The AVM syntax will also be quite helpful for explaining the structure of our own extended syntax.

The Head Feature Principle says that, in a headed phrase, the value of synsem | local | category | head and daughters | head-daughter | synsem | local | category | head are token-identical [Pollard and Sag, 1994, p. 399]. The Head Feature Principle is often notated as depicted in (8).

(8) *The* HEAD FEATURE PRINCIPLE *in AVM notation*

$$\begin{bmatrix} \textit{phrase} \\ \text{DTRS } \textit{headed-struc} \end{bmatrix} \rightarrow \begin{bmatrix} \text{SYNSEM LOC CAT HEAD } \boxed{1} \\ \text{DTRS HEAD-DTR SYNSEM LOC CAT HEAD } \boxed{1} \end{bmatrix}$$

The boxed integer, 1, conventionally called a *tag*, of (8) is a new construct. Tags do not have any counterpart in our initial descriptions. In our initial syntax we do not have the means yet to say that the values of two paths must be identical, which is the intended interpretation of the use of 1. Nor can we say that two path values are necessarily distinct. For example, when we say[6] `head:parrot,tail:(head:parrot,tail:elist)`, we describe a list whose first and second element are parrots. Nothing forces them to be the same parrot, but nothing prevents them from being the same parrot either. As you can see for yourself, using MoMo and our signature with lists of various animals, feature structures representing both situations (identity and non-identity of the parrots) satisfy the description. In the mmp-file possible-identities23.mmp, you find the relevant example. From the observed properties and the intended interpretation of *tags* we conclude preliminarily that they behave like variables in logical languages.

At this point, it is still undecided whether or not we should introduce explicit quantification over variables. Since classical logical theories usually do not have free variables, we might at first assume that we have to bind a variable like the tag 1 by some quantificational device. On the other hand, it might be possible to leave quantification implicit if it is used consistently in one and the same way in all grammatical principles. The remaining two examples for the use of tags show us that this is not the case and that we really need explicit quantification in a description language that expresses these principles directly.

The SPEC PRINCIPLE says that, in a headed phrase whose non-head daughter (either the MARKER-DAUGHTER or COMPLEMENT-DAUGHTERS | FIRST) has a SYNSEM | LOCAL | CATEGORY | HEAD value of sort *functional*, the SPEC value of that value must be token-identical with the phrase's DAUGHTERS | HEAD-DAUGHTER | SYNSEM value [Pollard and Sag, 1994, p. 400]:

(9) *The* SPEC PRINCIPLE *in AVM notation*

$$\forall \boxed{1}\, \forall \boxed{2}$$
$$\left(\left(\begin{bmatrix} \text{DTRS } \left[\left[\text{MARKER-DTR } \boxed{1} \right] \vee \left[\text{COMP-DTRS } \langle \boxed{1} \mid \textit{list} \rangle \right] \right] \\ \wedge \boxed{1} \left[\text{SS LOC CAT HEAD } \begin{bmatrix} \textit{functional} \\ \text{SPEC } \boxed{2} \end{bmatrix} \right] \end{bmatrix} \right) \rightarrow \left[\text{DTRS HEAD-DTR SS } \boxed{2} \right] \right)$$

As indicated in (9), it makes sense to assume that we see some kind of universal quantification going on in the SPEC PRINCIPLE: For all entities 1 such that they are either a marker daughter or the first complement daughter, and for all SPEC values 2 of such an entity 1 (which also has a HEAD value of sort *functional*), we require that the SPEC

[6] Assuming the signature of lists and animals first introduced in Section 2.1.2

value [2] of [1] be the SYNSEM value of the head daughter of the headed phrase that we are talking about. The quantificational situation is thus parallel to a first order formulation of a sentence like *Every woman works*, which is usually rendered as $\forall$`x (woman′(x)` $\rightarrow$ `work′(x))`. In contrast to the SPEC PRINCIPLE, the variable binding of the HEAD FEATURE PRINCIPLE has an existential flavor: The identity statement there says something like for each headed phrase, there exists an object in the denotation of the sort *head* such that it is the value of the HEAD attribute of the phrase and of its head daughter.

In the CONTROL THEORY we can observe both kinds of quantification in a single principle. The CONTROL THEORY says that, if the SOA-ARG value of a *control-qfpsoa* is token-identical with the CONTENT value of a *local* entity whose CATEGORY | SUBCAT value is a list of length one, then the member of that list is (1) reflexive, and (2) coindexed with the INFLUENCED (respectively, COMMITTOR, EXPERIENCER) value of the *control-qfpsoa* if the latter is of sort *influence* (respectively, *commitment*, *orientation*) [Pollard and Sag, 1994, p. 401]:

(10) *The* CONTROL THEORY *in AVM notation*

$$\forall\boxed{1}\ \forall\boxed{2}\ \forall\boxed{3}\left(\begin{array}{l}\exists\boxed{4}\left(\boxed{1}\begin{bmatrix}\textit{influence}\\ \text{SOA-ARG}\ \boxed{4}\end{bmatrix}\wedge\boxed{2}\begin{bmatrix}\textit{local}\\ \text{CAT SUBCAT}\ \langle\boxed{3}\rangle\\ \text{CONT}\ \boxed{4}\end{bmatrix}\right)\rightarrow\\ \exists\boxed{5}\left(\boxed{3}\left[\text{LOC CONT}\ \begin{bmatrix}\textit{reflexive}\\ \text{INDEX}\ \boxed{5}\end{bmatrix}\right]\wedge\boxed{1}\left[\text{INFLUENCED}\ \boxed{5}\right]\right)\end{array}\right)\wedge$$

$$\forall\boxed{1}\ \forall\boxed{2}\ \forall\boxed{3}\left(\begin{array}{l}\exists\boxed{4}\left(\boxed{1}\begin{bmatrix}\textit{commitment}\\ \text{SOA-ARG}\ \boxed{4}\end{bmatrix}\wedge\boxed{2}\begin{bmatrix}\textit{local}\\ \text{CAT SUBCAT}\ \langle\boxed{3}\rangle\\ \text{CONT}\ \boxed{4}\end{bmatrix}\right)\rightarrow\\ \exists\boxed{5}\left(\boxed{3}\left[\text{LOC CONT}\ \begin{bmatrix}\textit{reflexive}\\ \text{INDEX}\ \boxed{5}\end{bmatrix}\right]\wedge\boxed{1}\left[\text{COMMITTOR}\ \boxed{5}\right]\right)\end{array}\right)\wedge$$

$$\forall\boxed{1}\ \forall\boxed{2}\ \forall\boxed{3}\left(\begin{array}{l}\exists\boxed{4}\left(\boxed{1}\begin{bmatrix}\textit{orientation}\\ \text{SOA-ARG}\ \boxed{4}\end{bmatrix}\wedge\boxed{2}\begin{bmatrix}\textit{local}\\ \text{CAT SUBCAT}\ \langle\boxed{3}\rangle\\ \text{CONT}\ \boxed{4}\end{bmatrix}\right)\rightarrow\\ \exists\boxed{5}\left(\boxed{3}\left[\text{LOC CONT}\ \begin{bmatrix}\textit{reflexive}\\ \text{INDEX}\ \boxed{5}\end{bmatrix}\right]\wedge\boxed{1}\left[\text{EXPERIENCER}\ \boxed{5}\right]\right)\end{array}\right)$$

The syntax of the AVM notation of the principles in (8)–(10) reveals what we need to integrate into our formal language: We need to introduce a way to notate existential and universal quantification (of some kind), and we need variables. But at which places in our syntax do we need them?

A closer look at the SPEC PRINCIPLE and the CONTROL THEORY gives us crucial hints. From the HEAD FEATURE PRINCIPLE we know that variables (tags) may occur in descriptions at the same place where sort symbols occur. Quantifier symbols, however, seem to take scope over AVM matrices, they do not occur inside them: Both universal quantifiers in the SPEC PRINCIPLE quantify over the entire expression. Similarly, in the CONTROL THEORY the universal quantifiers scope over entire conjuncts of the expression (which

consists of three implicational conjuncts), with an additional existential quantification over the antecedent and over the consequent of each of the three implications. In the SPEC PRINCIPLE we can also see that variables may precede AVM matrices: The tag [1] precedes the AVM matrix that is the second conjunct of the complex antecedent of the implication.

We can summarize our observations as follows. Our initial syntax apparently provided a language that can best be understood as corresponding to simple AVM matrices. We can think of an initial description as corresponding to a single (potentially complex) AVM matrix as shown in (11):

(11) a. $\begin{bmatrix} \textit{parrot} & \\ \textsc{legs} & \textit{number} \\ \textsc{color} & \textit{red} \end{bmatrix}$ corresponds to `parrot,legs:number,color:red`

b. $\begin{bmatrix} \textit{list} & & \\ & \begin{bmatrix} \textit{parrot} & \\ \textsc{legs} & \textit{number} \\ \textsc{color} & \textit{red} \end{bmatrix} & \\ \textsc{head} & & \\ \textsc{tail} & \textit{nelist} & \end{bmatrix}$ corresponds to

`list,head:(parrot,legs:number,color:red),tail:nelist`

The parallel syntactic structure becomes even clearer when we notate the initial descriptions with the line-breaks of the AVM matrices. This is also legitimate in MoMo and makes descriptions much more readable: You can see the descriptions in that format in the MoMo file box-syntax23.mmp that we have prepared for inspection.

Call the initial descriptions that we have been working with *boxes*, in analogy to the matrix notation of AVMs. Inside these boxes, at the places where sort symbols could appear, and in front of these boxes, we should be able to write variables, analogous to the matrices $[\textsc{dtrs}\ \textsc{head-dtr}\ \textsc{ss}\ \boxed{2}]$ and $\boxed{1}\begin{bmatrix} \textsc{ss}\ \textsc{loc}\ \textsc{cat}\ \textsc{head} & \begin{bmatrix} \textit{functional} \\ \textsc{spec}\ \boxed{2} \end{bmatrix} \end{bmatrix}$ in the SPEC PRINCIPLE in (9). Entire boxes of that form should then be combinable by standard logical connectives (conjunction, disjunction, implication, equivalence) in order to form bigger expressions. And finally, we would like to write quantifiers in front of boxes or in front of complex expressions formed by combining several boxes by logical symbols.

This picture suggests two levels of syntax. On the lower level, we have the initial descriptions we have already been using, augmented by the option of placing variables inside them. Conjunction inside this box syntax is similar to starting new lines in AVMs (see (11a), where each line-break in the AVM is translated into a "," in the description). Besides conjunction, disjunction is occasionally used in AVM matrices ($[\textsc{color}\ \textit{red} \vee \textit{yellow}]$), whereas implication and equivalence rarely occur there—we only admit them for reasons of generality. In the file new-syntax-examples23.mmp, you will find examples of our initial syntax augmented with variables. The example *var value* illustrates the case where a variable occurs inside of a box. Note that variables in MoMo start with an upper case letter, just like in the Prolog programming language.

Then there will be a second syntactic layer of formulae, where we can combine boxes syntactically to form bigger expressions. On this outer layer, we allow the placement of variables in front of boxes, and we allow symbols for existential and universal quantification. Those formulae can be combined with each other using conjunction, disjunction, implication, and equivalence. The implication symbols of our three principles all belong to this outer layer of the syntax of descriptions (after all, they do not occur inside of boxes).

The example *var prefixed* in new-syntax-examples23.mmp shows a variable preceding a box and, as already familiar from the previous example, a variable inside a box. *quantification* shows the use of a quantifier. MoMo notates the universal quantifier as `V` and the existential quantifier as `^`. The details of our new syntax will of course be presented in detail in the next section.

There is one more syntactic construct in the AVM syntax that we should point out before we define our new syntax. The AVM notation uses a special notation for lists, $\langle \delta_1, \ldots, \delta_n \rangle$, which may be combined with variables. What is meant with this notation should be fairly transparent in light of the signature for lists that we have been using so far. Assume that $\delta_{avm1}, \delta_{avm2}$, and δ_{avm3} are AVMs, and δ_1, δ_2, and δ_3 are corresponding descriptions of our syntax. Then the AVM notation $\langle \delta_{avm1}, \delta_{avm2}, \delta_{avm3} \rangle$ can be translated to `(head:`δ_1`, tail:(head:`δ_2`, tail:(head:`δ_3`, tail:elist))`.

GLOSSARY

2.3.1 The Syntax of Complex Grammars

ABSTRACT

We will add variables and quantification to the syntax of initial descriptions and get a language of initial formulae. Along the way, we will need to introduce a few new concepts known from first order predicate logic. The syntax that we obtain is considerably more complex than our initial syntax. Analogies with the AVM syntax of the linguistic literature will help us to understand its structure.

From our observations about the AVM notation we already know what kind of description languages we would like to have at our disposal. Fortunately, we do not have to change anything about the initial signatures that we have been working with: Our syntactic extension of the initial syntax of descriptions is still based on initial signatures, since they provide almost all the symbols that we need. The only additional symbols that we use now come from a countably infinite set of variables, $\mathcal{VAR}$, augmented by the reserved symbol \$. For the sake of simplicity, we will write $\mathcal{VAR}^{\$}$ when we refer to the set $\mathcal{VAR} \cup \{\$\}$. We use capital Latin letters like X, Y, and Z for the members of $\mathcal{VAR}$. \$ is a reserved symbol, which is used as a special kind of variable. The basic intuition is that for satisfaction, \$ is a reserved variable that always refers to the root node. We will have to say more about it when we define the semantics for our extended syntax.

Based on the definition of initial signatures, we define initial formulae relative to a given initial signature. The first part of the definition is concerned with the syntax of boxes, and then comes the syntax of formulae that are built from boxes:

Definition 10 *For each initial signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F} \rangle$, $\mathbb{BOX}^{\Sigma}$ *and* $\mathcal{D}^{\Sigma}$ *are the smallest sets such that*

for each $X \in \mathcal{VAR}^{\$}$,

$X \in \mathbb{BOX}^{\Sigma}$,

for each $\sigma \in \mathcal{G}$,

$\sigma \in \mathbb{BOX}^{\Sigma}$,

for each $\alpha \in \mathcal{A}$, *for each* $\delta \in \mathbb{BOX}^{\Sigma}$,

$\alpha : \delta \ \in \mathbb{BOX}^{\Sigma}$,

for each $\delta \in \mathbb{BOX}^{\Sigma}$, $\sim \delta \ \in \mathbb{BOX}^{\Sigma}$,

for each $\delta_1 \in \mathbb{BOX}^{\Sigma}$, *for each* $\delta_2 \in \mathbb{BOX}^{\Sigma}$, $(\delta_1, \delta_2) \in \mathbb{BOX}^{\Sigma}$,

for each $\delta_1 \in \mathbb{BOX}^{\Sigma}$, *for each* $\delta_2 \in \mathbb{BOX}^{\Sigma}$, $(\delta_1; \delta_2) \in \mathbb{BOX}^{\Sigma}$,

for each $\delta_1 \in \mathbb{BOX}^{\Sigma}$, *for each* $\delta_2 \in \mathbb{BOX}^{\Sigma}$, $(\delta_1 *> \delta_2) \in \mathbb{BOX}^{\Sigma}$,

for each $\delta_1 \in \mathbb{BOX}^{\Sigma}$, *for each* $\delta_2 \in \mathbb{BOX}^{\Sigma}$, $(\delta_1 <*> \delta_2) \in \mathbb{BOX}^{\Sigma}$,

for each $\delta \in \mathbb{BOX}^{\Sigma}$,

$\delta \in \mathcal{D}^{\Sigma}$,

for each $X \in \mathcal{VAR}^{\$}$, *for each* $\delta \in \mathbb{BOX}^{\Sigma}$,

$X : \delta \in \mathcal{D}^{\Sigma}$,

for each $X \in \mathcal{VAR}^{\$}$, *for each* $Y \in \mathcal{VAR}^{\$}$,

$X = Y \in \mathcal{D}^{\Sigma}$,

for each $X \in \mathcal{VAR}$, *for each* $\delta \in \mathcal{D}^{\Sigma}$,

$\exists X \, \delta \in \mathcal{D}^{\Sigma}$,

for each $X \in \mathcal{VAR}$, *for each* $\delta \in \mathcal{D}^{\Sigma}$,

$\forall X \, \delta \in \mathcal{D}^{\Sigma}$,

24 *for each* $\delta \in \mathcal{D}^\Sigma$, $\sim \delta \ \in \mathcal{D}^\Sigma$,
25 *for each* $\delta_1 \in \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathcal{D}^\Sigma$, $(\delta_1, \delta_2) \in \mathcal{D}^\Sigma$,
26 *for each* $\delta_1 \in \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathcal{D}^\Sigma$, $(\delta_1; \delta_2) \in \mathcal{D}^\Sigma$,
27 *for each* $\delta_1 \in \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathcal{D}^\Sigma$, $(\delta_1 \texttt{*>} \delta_2) \in \mathcal{D}^\Sigma$, *and*
28 *for each* $\delta_1 \in \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathcal{D}^\Sigma$, $(\delta_1 \texttt{<*>} \delta_2) \in \mathcal{D}^\Sigma$.

We call each element of $\mathbb{BOX}^\Sigma$ a Σ *box*, and each element of $\mathcal{D}^\Sigma$ an *initial* Σ *formula*. Initial Σ formulae of the form $X = Y$, where X and Y are variables, are called equations. An expression of the form $X : \delta$, with X as a variable and δ as a Σ box, is called a tagged Σ box. If a variable occurs to the right of the colon, ':', in a Σ box or Σ formula of the form $\alpha : \delta$ or $X : \delta$, we say that it occurs in a Σ box. The only kind of syntactic expression missing in initial Σ formula compared to what we will finally need is relational expressions. Note that the lines 3–13 of DEFINITION 10 are almost identical to the definition of initial descriptions. The only addition there is, is the addition of variables in lines 3 and 4, which will allow us to write boxes of the form `color:X` (analogous to [COLOR 1]). In the lines 14–28 we then get the outer syntactic layer with equations, quantification, and, again, the logical connectives.

Initial Σ formulae are clearly more complex than the initial Σ descriptions of our first syntax. This is due to the fact that they are built from other possibly complex expressions, Σ boxes. Roughly speaking, Σ boxes correspond to the familiar AVM matrices, the expressions between angled brackets in the principles (8)–(10). More complex initial Σ formulae can be built out of them by prefixing them with variables (tags), combining expressions with the logical connectives, by quantifying over complex expressions, and by forming equations between variables. The special nature of the reserved symbol, \$, is emphasized by its syntactic distribution. In contrast to normal variables, it may not be quantified over. We cannot write $\forall\$$ or $\exists\$$ to quantify over a formula.

A few examples may be found in examples-231.mmp. *box 1* illustrates the fact that the special variable, \$, like any ordinary variable, is a box. In *box 2*, the simple sort symbol *list*, is a box as we knew it before. Similarly, the formulae in *box 3* and *box 4*, describing a canary and a parrot, respectively, are boxes of the type that we are already familiar with, except that a variable is used in the place of a sort symbol describing the value of LEGS in each of them. Note that, according to our definition, each of these boxes may alternatively be viewed as a formula: The definition says that each complete box is also a formula (lines 14 and 15).

With *form 1* and *form 2* we proceed from boxes to formulae: We prefix the boxes from *box 3* and *box 4*, respectively, with a variable. In *form 3*, finally, the resulting formulae are combined in a conjunction, together with an additional conjunct, a negated equation formula. All variables in this conjunctive formula are bound by existential quantifiers.

Our last example, *form 4*, illustrates a syntactic property of initial Σ formulae which tends to be surprising when first encountered: If a feature value which we refer to by a variable inside a box receives any further description, that description is added conjunctively. In *form 4* the color value of the parrot is referred to by the variable `Y`, and then

further described as being *yellow*. The sort *yellow* is the second conjunct of the box `(Y, yellow)`. The color of the canary, which is described as the second element on the list, is identified with the COLOR value of the parrot. Note that we cannot write `(Y:yellow)` for describing the COLOR value of the parrot, because `(Y:yellow)` would be a formula, and a formula cannot occur inside a box. A hypothetical box, such as "`color:(Y:yellow)`" would, therefore, be syntactically ill-formed. The reader may try this out using MoMo.

Our new notion of grammars will not use arbitrary initial formulae but only a restricted subset of them. The relevant subset is the set of those initial formulae that do not contain any free occurrences of variables. An instance of a variable occurs free in an initial formula if and only if it is not in the scope of any quantifier binding the variable in the initial formula. We formalize this idea using the function FV:

Definition 11 *For each initial signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F} \rangle$, FV *is the total function from* $\mathbb{BOX}^\Sigma \cup \mathcal{D}^\Sigma$ *to* $Pow\,(\mathcal{VAR})$ *such that*

$FV(\$) = \emptyset$,
for each $X \in \mathcal{VAR}$, $FV(X) = \{X\}$,
for each $\sigma \in \mathcal{G}$, $FV(\sigma) = \emptyset$,
for each $\alpha \in \mathcal{A}$, *for each* $\delta \in \mathbb{BOX}^\Sigma$, $FV(\alpha : \delta) = FV(\delta)$,
for each $X \in \mathcal{VAR}^\$$, *for each* $\delta \in \mathbb{BOX}^\Sigma$, $FV(X : \delta) = FV(X) \cup FV(\delta)$,
for each $X \in \mathcal{VAR}^\$$, *for each* $Y \in \mathcal{VAR}^\$$, $FV(X = Y) = FV(X) \cup FV(Y)$,
for each $X \in \mathcal{VAR}$, *for each* $\delta \in \mathcal{D}^\Sigma$, $FV(\exists X\, \delta) = FV(\delta) \backslash \{X\}$,
for each $X \in \mathcal{VAR}$, *for each* $\delta \in \mathcal{D}^\Sigma$, $FV(\forall X\, \delta) = FV(\delta) \backslash \{X\}$,
for each $\delta \in \mathbb{BOX}^\Sigma \cup \mathcal{D}^\Sigma$, $FV(\sim \delta) = FV(\delta)$,
for each $\delta_1 \in \mathbb{BOX}^\Sigma \cup \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathbb{BOX}^\Sigma \cup \mathcal{D}^\Sigma$, $FV((\delta_1, \delta_2)) = FV(\delta_1) \cup FV(\delta_2)$,
for each $\delta_1 \in \mathbb{BOX}^\Sigma \cup \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathbb{BOX}^\Sigma \cup \mathcal{D}^\Sigma$, $FV((\delta_1; \delta_2)) = FV(\delta_1) \cup FV(\delta_2)$,
for each $\delta_1 \in \mathbb{BOX}^\Sigma \cup \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathbb{BOX}^\Sigma \cup \mathcal{D}^\Sigma$, $FV((\delta_1 \texttt{*>} \delta_2)) = FV(\delta_1) \cup FV(\delta_2)$,
for each $\delta_1 \in \mathbb{BOX}^\Sigma \cup \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathbb{BOX}^\Sigma \cup \mathcal{D}^\Sigma$, $FV((\delta_1 \texttt{<*>} \delta_2)) = FV(\delta_1) \cup FV(\delta_2)$.

The function FV yields for each Σ box and for each initial Σ formula the set of variables that occur free in them. For example, in the formulae in (11), no variables occur free (because there are no variables at all). In the formula `color:X`, however, `X` is free, thus $FV($`color:X`$)=\{$`X`$\}$; and in `Y:color:X` both `X` and `Y` are free: $FV($`Y:color:X`$)=\{$`Y, X`$\}$. As soon as we quantify over one of the variables that occur free in a formula, it is no longer in the set of free variables of the overall expression: $FV(\exists$`Y Y:color:X`$)=\{$`X`$\}$. If we quantify over all variables in the expression the set of variables that occur free in it becomes empty: $FV(\forall$`X` $\exists$`Y Y:color:X`$)=\{\}$.

For grammars we are interested in Σ *descriptions*, the set of initial Σ formulae that do not contain any unbound variables:

Definition 12 *For each initial signature* Σ,

$$\mathcal{D}_0^\Sigma = \{\delta \in \mathcal{D}^\Sigma \mid FV(\delta) = \emptyset\}.$$

In other words, in grammatical principles we want to bind all occurrences of variables by some quantifier.

For each initial signature Σ, we call each member of $\mathcal{D}_0^\Sigma$ a Σ *description*. Our new notion of a grammar is based on initial signatures and on Σ descriptions. In DEFINITION 13, we use the insight that the principles of an HPSG grammar are a set of descriptions. The Greek letter θ now refers to a set of Σ descriptions:

Definition 13 Γ *is a* ***complex grammar*** *iff*

Γ *is a pair* $\langle \Sigma, \theta \rangle$,

Σ *is an initial signature, and*

$\theta \subseteq \mathcal{D}_0^\Sigma$.

A complex grammar is an initial signature, Σ, together with a set of Σ descriptions. Complex grammars are closer to real HPSG grammars than our initial grammars were. Complex grammars provide enough to formalize a significant number of the principles of Pollard and Sag's grammar of English, including our three examples for the use of variables and quantification, the HEAD FEATURE PRINCIPLE, the SPEC PRINCIPLE, and the CONTROL THEORY.

At this point you might be wondering about the HEAD FEATURE PRINCIPLE in (8). How can we claim that this is a description in the theory of Pollard and Sag's grammar of English? Isn't there a free variable, $\boxed{1}$? But with a free variable, it is a formula and not a description, and it should not be allowed in the theory!

That observation is completely correct, and we cannot leave the HEAD FEATURE PRINCIPLE as it stands. Either we have to add an existential quantification or we have to say something more. For the time being, we assume that we add an existential quantification. Later in our discussion (Section 2.3.3), we will make our lives a bit easier by introducing convenient conventions for our notation of grammatical principles. These conventions will make it possible to leave existential quantification implicit in some cases. At the moment, we are not quite that far yet: We first have to understand the meaning of our unabbreviated notation, which is what we will do next.

Exercises

Exercise 19

(a) We presuppose the signature of lists and animals first given in Section 2.1.2. How many simple abstract feature structures satisfy the following description?

`pet, color:~green.`

Create a MoMo file non-green-pets231.mmp with an interpretation window that contains (MoMo counterparts of) all these feature structures. Using the link just given you may download an mmp file that already contains the necessary signature.

(b) How many simple abstract feature structures satisfy the following description?

`pet *> color:~green.`

Add three examples to your mmp file (in a second interpretation window).

Exercise 20 *Translate the* Spec Principle, *(9), and the* Control Theory, *(10), into our syntax of descriptions (under Pollard and Sag's signature).*

Recall that MoMo does not use the symbols ∃ and ∀ as quantifier symbols, since they do not belong to the standard symbols of keyboards. ∃ is written as `^`, *and ∀ is written as* `V` *(capital V). ∃X thus becomes* `^X` *and ∀X is written as* `VX`. *Following the conventions of the Prolog computer language, variables are written with an initial capital letter and may consist of strings of letters. Some examples for variables in MoMo are:* `H`, `Z`, `W`, `Head`, `Color`.

Please test the well-formedness of your translations in MoMo on the basis of an adequately large fragment of the signature of [Pollard and Sag, 1994] in MoMo. For the purpose of this exercise, we have prepared an mmp-file, signature-fragment231.mmp that already contains the necessary fragment of Pollard and Sag's signature for downloading. You may notice that Pollard and Sag use sorts of the shape list(synsem) (so-called "parametric sorts"). For simplicity, we leave out the specification in brackets for now. Instead of "list(synsem)", we write "list". We will discuss this topic later in the course, in Section 2.5.2, Parametric Sorts.

Please upload the mmp files with your solutions to your forum in ILIAS, and either put an extra text file there with the additional answers or send an ILIAS message with them.

GLOSSARY

2.3.2 The Meaning of Complex Grammars

Abstract

> *We will define what it means for a simple abstract feature structure to satisfy a complex formula, and say what it means for a set of simple abstract feature structures to be admitted by a set of complex descriptions. The definition proceeds in parallel to the corresponding definitions of satisfaction and admission relative to simple descriptions in Section 2.2.2.4. This will allow us to focus on those constructs that are new.*

Fortunately, we do not have to modify our interpreting structures for grammars when we now proceed from interpreting initial descriptions and initial grammars to interpreting complex descriptions and complex grammars. Our syntactic inventory has become richer, and the expressiveness of the languages of our formalism will increase with the semantic extensions, but the abstract feature structures that we are using as interpreting structures do not change at all.

Although we do not change our definition of abstract feature structures, we still need to explain how we want to interpret our extended languages over the old and familiar domain of simple abstract feature structures. In particular, we must give meaning to the new pieces of syntax, variables, equations, and quantifiers. For the other constructs, on the other hand, we will simply keep the old semantics. In the beginning, this continuity might be slightly obscured by the fact that with the distinction between boxes and formulae, our entire syntax has become more complex. To avoid getting unnecessarily distracted or confused by the new "look and feel" of the old constructs, the reader should always keep in mind that the only substantial changes concern the addition of variables. In fact, if we subtract those boxes and formulae from our syntax that concern variables, having both boxes and formulae in the syntax becomes entirely redundant and we may as well return to our initial syntax without any loss in the set of well-formed expressions.

From what we have said about our syntactic extensions, it follows that when we start to think about interpreting our new syntax we have to start by thinking about how we want to interpret variables. Before we can define a new satisfaction function from Σ descriptions to sets of simple abstract feature structures under Σ, we must understand what variables should do for us. What do we mean by the boxed integers in the HEAD FEATURE PRINCIPLE or in the CONTROL THEORY, or by the corresponding variables in our MoMo and TRALE syntax?

To approach our problem gently, it might be useful to recall how variables are interpreted in a classical logic without the specialized interpreting domain of abstract feature structures. Standard logical languages employ variable assignment functions that assign entities in the structures that serve as their interpretations to variables. In our case, the entities that serve as interpreting structures are abstract feature structures. However, it does not seem right to let variables refer to only entire feature structures! As we have seen in our examples from a grammar of English, variables typically occur at the end of paths, like in the HEAD FEATURE PRINCIPLE. Since we know that path values are nodes *inside of* feature structures, an adequate choice for the interpretation of variables are the nodes of feature structures: Informally, variables point to nodes in feature structures.

This is very easy to see when we think about where variables occur in typical examples, and what they are supposed to do. If the same variable occurs at the end of two different paths, it is supposed to signal that these two paths lead to the same node in the feature structures that satisfy the description. If two different variables, `X` and `Y`, occur at the end of two paths, and they are inequated ($\sim$ `X=Y`), then this is supposed to signal that, in feature structures which satisfy the description, the two paths lead to different nodes of the feature structure. Since we use *abstract* feature structures, whose nodes are represented by equivalence classes of an equivalence relation over sequences of attributes, it makes sense to assign sequences of attributes to variables. A variable then denotes the abstract node represented by an equivalence class of paths. To refer to an equivalence class of paths, it suffices to assign any path that is a member of the equivalence class to the variable. Naturally, the assignment functions that do this are defined relative to initial signatures: Since we associate variables with paths, we have to refer to the set of attributes in a signature.

Definition 14 *For each initial signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F} \rangle$,

I_Σ *is the set of total functions from* $\mathcal{VAR}^{\$}$ *to* $\mathcal{A}^*$ *such that for each* $\iota \in \mathrm{I}_\Sigma$, $\iota(\$) = \varepsilon$.

For each initial signature Σ, we call each member of I_Σ a Σ *insertion*. The empty path, ε, is always inserted for the special symbol, \$.

With Definition 14, each variable is assigned exactly one sequence of attributes. By definition, \$ is a special variable with a fixed value. It will always refer to the root node of abstract feature structures. In a sense it helps us to "find" the root node when we need it.

For quantification over variables we need to adopt another technique that is familiar from predicate logic. With quantification, we find ourselves in a situation where we occasionally want to have the value of a particular variable X range over all paths π, or we want to say that there exists some path π that we can insert for a particular variable X such that a description is satisfied. In the new notation whose meaning we are going to define next, we can employ the syntax $\iota[\frac{\pi}{X}]$ to express this:

Definition 15 *For each initial signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F} \rangle$, *for each* $\iota \in \mathrm{I}_\Sigma$, *for each* $\pi \in \mathcal{A}^*$, *for each* $X \in \mathcal{VAR}$, *for each* $Y \in \mathcal{VAR}$,

$$\iota[\tfrac{\pi}{X}](Y) = \begin{cases} \pi & \textit{if } \mathtt{X}{=}\mathtt{Y} \\ \iota(Y) & \textit{otherwise.} \end{cases}$$

This definition is saying that $\iota[\frac{\pi}{X}]$ is exactly the same function as the function ι except that the value of the former at X is π (whereas we do not know anything about the value of ι at X—its value at X is simply not relevant for us). We will see in a minute how this notation is used for the definition of the satisfaction function of quantificational formulae.

Before we can define a satisfaction function for Σ boxes and Σ formulae, we need to consider one more peculiarity of the new syntax. In the definition of our extended syntax, we saw that variables may occur in both layers of our syntax: They occur in boxes, which correspond to AVM matrices; and they occur on the syntactic level of formulae in front of boxes, behind quantifier symbols, and in equations. The reason for a certain complication in the following interpretation of variables that occur in boxes is that, informally speaking, we want to interpret all variables on the formula level, even those that occur inside of boxes. Since we cannot know the position of the box from inside, we need to get out of the box before we can locate the position of the variables in the box in the overall formula. In effect, we look from the formula level into the boxes and collect the sequences of attributes which are syntactically followed by variables.

This outside perspective on the location of variables in boxes will be realized by keeping track of the attribute path that we traverse during the stepwise interpretation of boxes and by also storing (parts of) the feature structure of the formula level in order to check later whether variables inside boxes refer to some node of this feature structure. A simple example, in which we employ the AVM notation, illustrates this basic idea nicely.

The consequent of the Head Feature Principle, repeated in (12), is a box in our syntax.

(12) *The consequent of the* HEAD FEATURE PRINCIPLE *in AVM notation*

$$\begin{bmatrix} \text{SYNSEM LOC CAT HEAD} & \boxed{1} \\ \text{DTRS HEAD-DTR SYNSEM LOC CAT HEAD} & \boxed{1} \end{bmatrix}$$

Two paths lead to occurrences of the variable $\boxed{1}$ in (12): SYNSEM LOC CAT HEAD and DTRS HEAD-DTR SYNSEM LOC CAT HEAD. Assuming that $\boxed{1}$ is bound existentially and that the box corresponding to the AVM in (12) is not part of a larger box, an abstract feature structure, $\langle \beta, \varrho, \lambda \rangle$, that satisfies (12) must include the set

$$\left\{ \begin{array}{l} \langle \text{SYNSEM LOC CAT HEAD}, \iota'(\boxed{1}) \rangle\,, \\ \langle \text{DTRS HEAD-DTR SYNSEM LOC CAT HEAD}, \iota'(\boxed{1}) \rangle \end{array} \right\}$$

in its re-entrancy relation, ϱ. Here, $\iota'(\boxed{1})$ is, of course, a path that can be inserted for the variable $\boxed{1}$ by virtue of how we will handle existential quantification.

In slightly more general terms, we may say that the purpose of keeping track of the paths inside of boxes and of the re-entrancy relation of the formula level in the satisfaction function for boxes is to tell us, for each Σ formula of the form $\alpha : \delta$ or $X : \delta$, which variables occur in the Σ box δ and which paths lead to them in the description δ. Informally speaking, our satisfaction functions will then require that, for a simple abstract feature structure to satisfy an expression, each path ending in a variable leads to the same abstract node to which the corresponding variable refers. In other words, for a variable `X` occurring in a box δ, the insertion function has to insert a path that is in the same equivalence class of ϱ as the path that syntactically leads to the variable `X` in δ in order for δ to be satisfiable by a simple abstract feature structure $\langle \beta, \varrho, \lambda \rangle$.

The definition of the semantics for Σ formulae follows the syntactic structure of our complex expressions. It uses two separate satisfaction functions, Ξ^{ι}_{Σ} and Δ^{ι}_{Σ}, for interpreting the two levels of our syntax, boxes and formulae, respectively. For each initial signature Σ and each Σ insertion $\iota \in \mathrm{I}_{\Sigma}$, the complex abstract feature structure satisfaction function under ι in Σ, Δ^{ι}_{Σ}, is defined inductively over all initial Σ formulae, and the auxiliary function Ξ^{ι}_{Σ} is defined inductively over the structure of Σ boxes. This is to say that the second feature structure satisfaction function under ι in Σ for Σ boxes, Ξ^{ι}_{Σ}, gives us the feature structure denotation of Σ boxes under ι. Since the task of interpreting expressions is divided between Δ^{ι}_{Σ} for the formula level and Ξ^{ι}_{Σ} for the box level, the interpretation of a formula starts with Δ^{ι}_{Σ} and continues with Ξ^{ι}_{Σ} as soon as we enter the box level of our syntax by breaking down the formula into small enough subparts. Ξ^{ι}_{Σ} stores the re-entrancy relation of the formula level (ϱ_0) and keeps track of the path which is traversed in the stepwise interpretation of the box (π). The only times when the information kept in ϱ_0 and accumulated in π becomes immediately relevant is when a variable is encountered in a box and is interpreted (lines 4 and 5 of DEFINITION 16).

Following the structure of the definition of the syntax, the definition of abstract feature structure satisfaction proceeds the other way round, bottom-up, from the smallest pieces of syntax (boxes) to the bigger expressions (formulae). Therefore, the clauses of DEFINITION 16 define Ξ^{ι}_{Σ} before we get to Δ^{ι}_{Σ}:

Definition 16 *For each initial signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F} \rangle$, *for each* Σ *insertion* $\iota \in \mathrm{I}_\Sigma$, Ξ^ι_Σ *is the total function from the threefold Cartesian product of* $Pow(\mathcal{A}^* \times \mathcal{A}^*)$, $\mathcal{A}^*$ *and* $\mathbb{BOX}^\Sigma$ *to* $Pow(\mathbb{AFS}^\Sigma)$, *and* Δ^ι_Σ *is the total function from* $\mathcal{D}^\Sigma$ *to* $Pow(\mathbb{AFS}^\Sigma)$ *such that*

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$, *for each* $\pi \in \mathcal{A}^*$, *for each* $X \in \mathcal{VAR}^\$$,

$$\Xi^\iota_\Sigma(\varrho_0, \pi, X) = \left\{ \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^\Sigma \,\middle|\, \langle \iota(X), \pi \rangle \in \varrho_0 \right\},$$

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$, *for each* $\pi \in \mathcal{A}^*$, *for each* $\sigma \in \mathcal{G}$,

$$\Xi^\iota_\Sigma(\varrho_0, \pi, \sigma) = \left\{ \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^\Sigma \,\middle|\, \begin{array}{l} \textit{for some } \sigma' \in \mathcal{S}, \\ \quad \langle \varepsilon, \sigma' \rangle \in \lambda, \textit{ and} \\ \quad \sigma' \sqsubseteq \sigma \end{array} \right\},$$

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$, *for each* $\pi \in \mathcal{A}^*$, *for each* $\alpha \in \mathcal{A}$, *for each* $\delta \in \mathbb{BOX}^\Sigma$,

$$\Xi^\iota_\Sigma(\varrho_0, \pi, \alpha : \delta) = \left\{ \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^\Sigma \,\middle|\, \begin{array}{l} \alpha \in \beta, \textit{ and} \\ \langle \beta, \varrho, \lambda \rangle / \alpha \in \Xi^\iota_\Sigma(\varrho_0, \pi\alpha, \delta) \end{array} \right\},$$

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$, *for each* $\pi \in \mathcal{A}^*$, *for each* $\delta \in \mathbb{BOX}^\Sigma$,

$$\Xi^\iota_\Sigma(\varrho_0, \pi, \sim \delta) = \mathbb{AFS}^\Sigma \backslash \Xi^\iota_\Sigma(\varrho_0, \pi, \delta),$$

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$, *for each* $\pi \in \mathcal{A}^*$, *for each* $\delta_1 \in \mathbb{BOX}^\Sigma$, *for each* $\delta_2 \in \mathbb{BOX}^\Sigma$,

$$\Xi^\iota_\Sigma(\varrho_0, \pi, (\delta_1, \delta_2)) = \Xi^\iota_\Sigma(\varrho_0, \pi, \delta_1) \cap \Xi^\iota_\Sigma(\varrho_0, \pi, \delta_2),$$

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$, *for each* $\pi \in \mathcal{A}^*$, *for each* $\delta_1 \in \mathbb{BOX}^\Sigma$, *for each* $\delta_2 \in \mathbb{BOX}^\Sigma$,

$$\Xi^\iota_\Sigma(\varrho_0, \pi, (\delta_1; \delta_2)) = \Xi^\iota_\Sigma(\varrho_0, \pi, \delta_1) \cup \Xi^\iota_\Sigma(\varrho_0, \pi, \delta_2),$$

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$, *for each* $\pi \in \mathcal{A}^*$, *for each* $\delta_1 \in \mathbb{BOX}^\Sigma$, *for each* $\delta_2 \in \mathbb{BOX}^\Sigma$,

$$\Xi^\iota_\Sigma(\varrho_0, \pi, (\delta_1 \mathbin{*>} \delta_2)) = \left(\mathbb{AFS}^\Sigma \backslash \Xi^\iota_\Sigma(\varrho_0, \pi, \delta_1) \right) \cup \Xi^\iota_\Sigma(\varrho_0, \pi, \delta_2),$$

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$, *for each* $\pi \in \mathcal{A}^*$, *for each* $\delta_1 \in \mathbb{BOX}^\Sigma$, *for each* $\delta_2 \in \mathbb{BOX}^\Sigma$,

$$\begin{array}{l} \Xi^\iota_\Sigma(\varrho_0, \pi, (\delta_1 \mathbin{<*>} \delta_2)) = \\ \left(\left(\mathbb{AFS}^\Sigma \backslash \Xi^\iota_\Sigma(\varrho_0, \pi, \delta_1) \right) \cap \left(\mathbb{AFS}^\Sigma \backslash \Xi^\iota_\Sigma(\varrho_0, \pi, \delta_2) \right) \right) \cup \\ (\Xi^\iota_\Sigma(\varrho_0, \pi, \delta_1) \cap \Xi^\iota_\Sigma(\varrho_0, \pi, \delta_2)), \textit{ and} \end{array}$$

for each $X \in \mathcal{VAR}^\$$,

$$\Delta^{\iota}_{\Sigma}(X) = \Big\{ \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^{\Sigma} \,\Big|\, \iota(X) \in \beta \Big\},$$

for each $\sigma \in \mathcal{G}$,

$$\Delta^{\iota}_{\Sigma}(\sigma) = \left\{ \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^{\Sigma} \,\middle|\, \begin{array}{l} \textit{for some } \sigma' \in \mathcal{S}, \\ \quad \langle \varepsilon, \sigma' \rangle \in \lambda, \textit{ and} \\ \quad \sigma' \sqsubseteq \sigma \end{array} \right\},$$

for each $\alpha \in \mathcal{A}$, *for each* $\delta \in \mathbb{BOX}^{\Sigma}$,

$$\Delta^{\iota}_{\Sigma}(\alpha : \delta) = \left\{ \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^{\Sigma} \,\middle|\, \begin{array}{l} \alpha \in \beta, \\ \langle \beta, \varrho, \lambda \rangle / \alpha \in \Xi^{\iota}_{\Sigma}(\varrho, \alpha, \delta) \end{array} \right\},$$

for each $X \in \mathcal{VAR}^{\$}$, *for each* $\delta \in \mathbb{BOX}^{\Sigma}$,

$$\Delta^{\iota}_{\Sigma}(X : \delta) = \left\{ \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^{\Sigma} \,\middle|\, \begin{array}{l} \iota(X) \in \beta, \textit{ and} \\ \langle \beta, \varrho, \lambda \rangle / \iota(X) \in \Xi^{\iota}_{\Sigma}\left(\varrho, \iota(X), \delta\right) \end{array} \right\},$$

for each $X \in \mathcal{VAR}^{\$}$, *for each* $Y \in \mathcal{VAR}^{\$}$,

$$\Delta^{\iota}_{\Sigma}(X = Y) = \Big\{ \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^{\Sigma} \,\Big|\, \langle \iota(X), \iota(Y) \rangle \in \varrho \Big\},$$

for each $X \in \mathcal{VAR}$, *for each* $\delta \in \mathcal{D}^{\Sigma}$,

$$\Delta^{\iota}_{\Sigma}(\exists X\, \delta) = \left\{ \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^{\Sigma} \,\middle|\, \begin{array}{l} \textit{for some } \pi \in \beta, \\ \quad \langle \beta, \varrho, \lambda \rangle \in \Delta^{\iota[\frac{\pi}{X}]}_{\Sigma}(\delta) \end{array} \right\},$$

for each $X \in \mathcal{VAR}$, *for each* $\delta \in \mathcal{D}^{\Sigma}$,

$$\Delta^{\iota}_{\Sigma}(\forall X\, \delta) = \left\{ \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^{\Sigma} \,\middle|\, \begin{array}{l} \textit{for each } \pi \in \beta, \\ \quad \langle \beta, \varrho, \lambda \rangle \in \Delta^{\iota[\frac{\pi}{X}]}_{\Sigma}(\delta) \end{array} \right\},$$

for each $\delta \in \mathcal{D}^{\Sigma}$, $\Delta^{\iota}_{\Sigma}(\sim \delta) = \mathbb{AFS}^{\Sigma} \backslash \Delta^{\iota}_{\Sigma}(\delta)$,

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$, $\Delta^{\iota}_{\Sigma}((\delta_1, \delta_2)) = \Delta^{\iota}_{\Sigma}(\delta_1) \cap \Delta^{\iota}_{\Sigma}(\delta_2)$,

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$, $\Delta^{\iota}_{\Sigma}((\delta_1; \delta_2)) = \Delta^{\iota}_{\Sigma}(\delta_1) \cup \Delta^{\iota}_{\Sigma}(\delta_2)$,

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$, $\Delta^{\iota}_{\Sigma}((\delta_1 \mathbin{*>} \delta_2)) = \Big(\mathbb{AFS}^{\Sigma} \backslash \Delta^{\iota}_{\Sigma}(\delta_1)\Big) \cup \Delta^{\iota}_{\Sigma}(\delta_2)$, *and*

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$,
$\Delta^{\iota}_{\Sigma}((\delta_1 \mathbin{<*>} \delta_2)) = \Big(\Big(\mathbb{AFS}^{\Sigma} \backslash \Delta^{\iota}_{\Sigma}(\delta_1)\Big) \cap \Big(\mathbb{AFS}^{\Sigma} \backslash \Delta^{\iota}_{\Sigma}(\delta_2)\Big)\Big) \cup (\Delta^{\iota}_{\Sigma}(\delta_1) \cap \Delta^{\iota}_{\Sigma}(\delta_2))$.

We call Δ^{ι}_{Σ} the *simple abstract feature structure satisfaction function under ι in* Σ, and say $\mathbb{A}$ satisfies δ under ι in Σ if and only if $\mathbb{A} \in \Delta^{\iota}_{\Sigma}(\delta)$.

The definition of Ξ^{ι}_{Σ} is almost identical to our previous simple abstract feature structure satisfaction functions of initial grammars. This fact is, however, slightly obscured by the two additional arguments of Ξ^{ι}_{Σ}, symbolized as ϱ_0 and π, which serve only as an aid in the interpretation of variables inside of boxes, as already explained above. If you are initially confused by the complexity of the notation, it might be useful to entirely ignore these two additional arguments of Ξ^{ι}_{Σ} at first and to concentrate on what is identical to the former simple abstract feature structure satisfaction functions.

Except for the very first clause concerning variables in lines 4 and 5, no variables occur, and the Σ insertion does nothing. A Σ box consisting of a sort symbol (lines 6–7) is interpreted just as before, and ϱ_0 and the traversed path π are irrelevant. A Σ box of the form $\alpha : \delta$, with α an attribute and δ another Σ box, is satisfied by a simple abstract feature structure on whose root node α is defined and whose α reduct satisfies δ (lines 8–10). In the recursion of this case, ϱ_0 is passed on and the concatenation of π and α is the new path relevant for the interpretation of variables that might be embedded in δ. For the cases of Σ boxes involving negation and all two place logical connectives (lines 11–26), ϱ_0 and π are simply passed on to the recursive case, since in none of these cases we traverse any attribute. The one clause that is different concerns the interpretation of variables, `X`, in lines 4–5. In this case, the simple abstract feature structure satisfies the Σ box if and only if the Σ insertion assigns a path π' to `X` such that the inserted path π' and the traversed path π are in the re-entrancy relation ϱ_0 that we stored at the point we went from the formula level to the box level of the satisfaction function. The reason for this indirect way of handling variables inside of Σ boxes is that the restrictions that come with them can be evaluated only relative to the formula level of a description.

The interpretation of variables at the formula level achieves a parallel effect much more directly. We can see this immediately when we look at the definition of Δ^{ι}_{Σ}. For each variable `X`, $\Delta^{\iota}_{\Sigma}(X)$ is the set of simple abstract feature structures such that ι inserts a path for `X` that is in the basis set of the feature structure (lines 27–28). Since the elements of the basis set are all paths that characterize an abstract node of the feature structure (by virtue of characterizing an equivalence class of the re-entrancy relation), we can take this to mean that `X` denotes the set of simple abstract feature structures in which it refers to an abstract node.

Why that makes sense is best explained by looking at the denotation of quantificational expressions. Understanding quantification is essential for understanding variable denotations. The reason is the crucial condition that we imposed on the formulae that may occur in grammars when we extended our syntax to comprise variables: We demanded that complex grammars consist of sets of *descriptions*, where a description is a formula *without free variables*. This means that in complex grammars, each variable in each member of the theory of the grammar, θ, is bound by a quantifier.

With that in mind, let us look at the value of the satisfaction function for expressions with existential quantification: For each Σ formula δ, $\Delta^{\iota}_{\Sigma}(\exists X\, \delta)$ is the set of simple abstract feature structures such that we can find some path π to insert for all occurrences of `X` in the

scope of the existential quantifier such that the simple abstract feature structure satisfies δ. But as we have already seen, this means that we can find an abstract node in the denotation of X in the simple abstract feature structure, such that X referring to that node is consistent with the other restrictions that δ imposes.

To see more clearly what is going on, suppose we quantify existentially over the variable 1 of the consequent of the HEAD FEATURE PRINCIPLE in (12). A feature structure that satisfies this description must appear as follows: There must be a node X in it that we can reach from the root node by following a sequence of arcs labeled SYNSEM LOC CAT HEAD; and we reach the very same node X by alternatively following a sequence of arcs labeled DTRS HEAD-DTR SYNSEM LOC CAT HEAD from the root node.[7] In non-technical terms that means that the consequent of the HEAD FEATURE PRINCIPLE with existential quantification over its single variable is satisfied by all feature structures in which SYNSEM LOC CAT HEAD and DTRS HEAD-DTR SYNSEM LOC CAT HEAD are structure-shared—which is exactly what linguists intend to express with that notation.

What about universal quantification, then? According to our definition, $\forall X\ \delta$ is satisfied by all simple abstract feature structures, $\mathbb{A}$, such that for each denotation that we can choose for X in $\mathbb{A}$, i.e., for X referring to any abstract node in $\mathbb{A}$, $\mathbb{A}$ also consistently satisfies δ.

To see how this interpretation of universal quantification can be used productively, consider the SPEC PRINCIPLE of Section 2.3, *Complex Grammars and their Meaning*. As every principle of grammar, the SPEC PRINCIPLE is an implicational description. An implicational description is true of all feature structures that fail to satisfy the antecedent, and of all those feature structures that satisfy both the antecedent and the consequent.

Let us first consider the case in which feature structures fail to satisfy the antecedent. Feature structures fail to satisfy the antecedent under universal quantification over the variables 1 and 2 in the following circumstances: They have neither a marker daughter nor a list of complement daughters with at least one complement daughter on the list (first line of the SPEC PRINCIPLE); or it is not the case that 1 refers to the marker daughter or the first complement daughter; or 1 either refers to a marker daughter or the first complement daughter, but is not a sign whose SYNSEM LOC CAT HEAD value is of sort *functional*; or 1 refers to a marker daughter or the first complement daughter, and its SYNSEM LOC CAT HEAD value is of sort *functional*, but the value of the SPEC arc of the *functional* node in the feature structure is not referred to by 2.

The second case to consider is the case where feature structures satisfy the antecedent. As we have just seen, a simple abstract feature structure $\mathbb{A}$ may satisfy the antecedent if 1 refers either to a functional marker daughter or to the first complement daughter (which is then also functional), and the SYNSEM LOC CAT HEAD SPEC value of the daughter that 1 designates is referred to by 2. For $\mathbb{A}$ to satisfy the entire principle, it must also satisfy the consequent, which is in the scope of the universal quantifiers as well. This means that the SYNSEM value of the head daughter of the phrase is identical to the SYNSEM LOC CAT HEAD SPEC value of 1. But according to the antecedent, 1 is a functional marker daughter or the

[7] Here we have switched to the terminology of concrete feature structures, since its metaphors seem to be easier to picture than the equivalence classes of paths of abstract feature structures.

(functional) first complement daughter. We have thus used the combination of universal quantification and implication to keep the referents of $\boxed{1}$ and $\boxed{2}$ constant for those situations in which we want to express a restriction on the satisfiers of the antecedent.

The definition of Δ^{ι}_{Σ} for $\alpha : \delta$ shows how the interpretation of box-internal variables is initialized when we go from the interpretation of formulae to the interpretation of boxes: The case $\Delta^{\iota}_{\Sigma}(\alpha : \delta)$ is one of the cases where this happens. As is clear from its definition, δ is then interpreted under Ξ^{ι}_{Σ}, i.e., we arrive at the box level. Unsurprisingly, we demand that α be a path defined on the abstract root node, and that the α reduct of the simple abstract feature structure satisfy δ. The interesting thing is what the first two arguments of Ξ^{ι}_{Σ} contain. This is the re-entrancy relation, ϱ, whose satisfaction conditions we are checking, and the attribute α, which we traverse to check whether the α reduct of the simple abstract feature structure satisfies δ. Whenever we go a level deeper in δ, the recursion in Ξ^{ι}_{Σ} records the traversed attribute by appending it to the path built up in the preceding steps. In this way, it collects paths that lead to occurrences variables, if there are any. Then it requires for each pair, $\langle \pi, \mathtt{X} \rangle$, of a path, π, leading to a variable, `X`, and the variable `X` at its end that π be in the equivalence relation with the insertion for the variable. Informally, the variable must denote the node to which the path leads. Intuitively, this is clearly how we understand variables in boxes.

The satisfaction function for formulae of the form $X : \delta$ is a similar case: `X` takes over the role of α of the previously discussed case. Now a simple abstract feature structure that satisfies the formula must have the path π inserted for `X` in its basis set, and we require that the π reduct of the abstract feature structure also satisfy δ. Simultaneously we have to make sure that the variables that occur in δ denote the same abstract node in the abstract feature structure as the paths that lead to them. These paths must now be reconstructed from the insertion for `X` plus the paths in δ that lead to each variable `Y`. Again, we use the recursive structure of Ξ^{ι}_{Σ} to record the paths in δ that lead to occurrences of variables.

The interpretation of equations, our final new syntactic construct, is quite straightforward. It is obvious that with `X=Y` we want to say that `X` and `Y` refer to the same abstract node in those abstract feature structures that satisfy the equation. For example, an alternative formulation of the consequent of the Head Feature Principle using an equation could be rendered as:

(13) *The consequent of the* Head Feature Principle *in AVM notation with an equation*

$$\exists \boxed{1} \exists \boxed{2} \left(\begin{bmatrix} \text{SYNSEM LOC CAT HEAD } \boxed{1} \\ \text{DTRS HEAD-DTR SYNSEM LOC CAT HEAD } \boxed{2} \end{bmatrix} \wedge \boxed{1} = \boxed{2} \right)$$

The idea behind equations is captured by saying that an abstract feature structure satisfies the equation `X=Y` iff the paths inserted for `X` and `Y` belong to the same equivalence class in the feature structure: They are alternative ways of getting to the same abstract node.

Concluding our discussion of the denotation of complex formulae, we should stress that quantification is always quantification over the substructure of the (abstract) root node to

which the special symbol, $, refers. It follows that existential and universal quantification in HPSG does not behave like existential and universal quantification in first order logic, where quantification ranges over the entire universe of entities. In terms of the feature structure models of HPSG grammars that we are considering here, quantification is always quantification within feature structures, never across different, structurally unrelated feature structures. An existential statement is thus not a statement about the existence of a feature structure in the domain of feature structures. It is rather a statement about the existence of a node *in* a feature structure. Similarly, a universal statement or a statement that uses universal quantifiers is not a statement about all feature structures, at least not by way of being a statement with universal quantifiers.[8] Universal quantification is quantification over all nodes in a feature structure.

Simple abstract feature structure admission of sets of complex descriptions is parallel to our previous definition with initial descriptions. The only difference is that we are now working relative to complex descriptions, which forces us to mention Σ insertions. However, Σ insertions are irrelevant for expressions without free occurrences of variables, since their meaning is independent of the Σ insertion we choose. As a consequence of this observation, our final definition of the admission function does not contain anything that we haven't seen before:

Definition 17 *For each initial signature* Σ*,*

M_Σ *is the total function from* $Pow(\mathcal{D}_0^\Sigma)$ *to* $Pow(\mathbb{AFS}^\Sigma)$ *such that for each* $\theta \subseteq \mathcal{D}_0^\Sigma$*,*

$$M_\Sigma(\theta) = \left\{ \langle \beta, \varrho, \lambda \rangle \in \mathbb{AFS}^\Sigma \;\middle|\; \begin{array}{l} \textit{for each } \pi \in \beta, \\ \textit{for each } \iota \in \mathrm{I}_\Sigma, \textit{ and} \\ \textit{for each } \delta \in \theta, \\ \quad \langle \beta, \varrho, \lambda \rangle / \pi \in \Delta_\Sigma^\iota(\delta) \end{array} \right\}.$$

For each initial signature Σ, I call M_Σ the *simple abstract feature structure admission function in* Σ. A simple abstract feature structure under Σ, $\mathbb{A}$, is in the set of simple abstract feature structures under Σ admitted by a Σ theory θ if and only if $\mathbb{A}$ and every possible π reduct of $\mathbb{A}$ satisfy every description in θ. This is precisely the definition that we already had before with our simpler description language.

Just as before, given a complex grammar, $\langle \Sigma, \theta \rangle$, we can regard the set of feature structures admitted by θ, $M_\Sigma(\theta)$, as the intended meaning of $\langle \Sigma, \theta \rangle$. Pollard and Sag say that the members of $M_\Sigma(\theta)$ are mathematical representations of the object types of the language that the grammar is about.

[8] Any statement in a grammar is universal in the sense that it makes a statement about all nodes of all feature structures admitted by the grammar. But that is an independent property of all descriptions in a grammar and has nothing to do with (universal) quantification.

Examples

It is again time to consider a few examples, and to practice our understanding of the meaning of variables and quantification in our description languages. The file synsem-232.mmp uses our signature of lists, pets and birds. It contains a few description cards and two interpretations with a feature structure.

With the description cards *and* and *colon* we are going to investigate the difference in our description languages between variables that occur in front of boxes and variables that occur embedded somewhere within boxes. The difference is far from obvious when we think of the standard AVM notation for HPSG descriptions.

Consider the following two AVM matrices, which are instances of each case:

(14) a. $\boxed{1}\,[\mathit{parrot}\,]$

b.
$$\begin{bmatrix} \mathit{nelist} \\ \textsc{head} \;\; \boxed{1}\,[\mathit{parrot}] \\ \textsc{tail} \quad \mathit{list} \end{bmatrix}$$

At first it might seem as if in our own syntax, the translation of (14a) will occur somewhere embedded into the translation of (14b), preserving the intended meaning of the formula. However, upon closer inspection, this will turn out not to be the case. As an investigation of the semantics of our formulae shows, variables that occur *in front of* complete AVM matrices of the AVM notation are translated to variables followed by colon (':') in MoMo syntax: From a formal point of view, these are variables that precede Σ boxes (which is the construct of MoMo syntax that emulates AVM matrices). Variables that occur *inside of* AVM matrices occur with the same syntactic distribution as sort symbols in MoMo syntax.

This means that (14a), $\boxed{1}\,[\mathit{parrot}]$ is translated to `X:parrot`, (occurrence of a variable in front of a Σ box). Example (14b), on the other hand, is translated as `nelist, head:(X,parrot), tail:list` (occurrence of a variable inside a Σ box).

The difference in meaning that comes with the notational distinction between variables in front of Σ boxes and variables inside Σ boxes is best understood by considering an example like the one provided in our mmp file. Note that since MoMo works with descriptions, we have bound the variables existentially.

In the interpretation *list with one parrot*, you find a feature structure representing a singleton list with one red parrot. This feature structure does not satisfy the description `^X(X,parrot)` on the card *and*. Why not? Under satisfaction, `^X(X,parrot)` means: There is some `X`, and the root node of the feature structure is of sort *parrot*. The first part (there is some `X`) is trivially true of any feature structure. The second restriction (the feature structure is of sort *parrot*), however, is relevant: The feature structure we are looking at is of sort *nelist*, which cannot satisfy `parrot`. In AVM notation, we would have to express our description as $\exists\boxed{1}\,\left(\boxed{1} \wedge [\mathit{parrot}]\right)$ (in contrast to (14a)).

Now consider `^X(X:parrot)` on the description card *colon*. Under satisfaction, this means the following: There is a node `X` in the feature structure, and `X` is of sort *parrot*.

Of course, our singleton list does contain a parrot and thus satisfies `^X(X:parrot)`. The corresponding AVM description is (14a).

Let us make our example even more complex, and consider a description with a variable both in front and inside a box: `^X^Y(X:(parrot, legs:(Y,two)))`, as contained on the description card *two quantifiers*. What does that mean under satisfaction? It means: There is a parrot, and it has the LEGS value *two*. Variable `Y` doesn't even do anything interesting here, because we already know (from the signature) that there exists a node that we reach following the LEGS arc from the *parrot* node. However, we see again that the combination of a variable with some further box *inside* a bigger box (like `Y` in our case) usually comes with a conjunctive interpretation and thus syntactically with a comma. By the way, our feature structure of course satisfies the description, as you may want to try out yourself in MoMo.

The AVM description corresponding to (14b) is on the description card *ten-b* (where we have again added explicit existential quantification over the variable). If we replace the comma for conjunction between the variable `X` and `parrot` in that description by a colon, we get an ill-formed expression (as witnessed by *ill-formed*): An expression of the form `X:parrot` is a formula, and a formula must not occur inside a box!

Let us close with a few remarks on differences between satisfaction and admission with variables and quantification, and on the use of the reserved variable, $.

As we have already seen, the feature structure in *list with one parrot* satisfies the description on the card *colon*, because there is a node of sort *parrot* in the feature structure. However, the feature structure is not a model of that description. To be admitted by the description, every node of the feature structure would have to satisfy the description. As you can check, there are three nodes in the feature structure that do not satisfy the description. The reason is that, if we consider those three nodes as root nodes, they do not contain any node of sort *parrot*.

The description on the card *dollar* illustrates universal quantification and the symbol $. The description says that for each node of sort *nelist*, its TAIL value is the root node. The feature structure in *list with one parrot* does not model this description: Clearly, the TAIL value of the root node is not the root node itself. The feature structure in *cyclic list with one parrot*, however, models the description: The TAIL arc of each *nelist* node points to the root node. Closer inspection shows that we may simplify the description *dollar* as given in *dollar without V* to obtain an equivalent description (under admission) that is formulated without variable `X` and universal quantification.

GLOSSARY

2.3.3 A Notational Convention

ABSTRACT

We introduce a notational convention that allows us to leave existential quantifiers implicit in some cases in principles of grammar.

Complex grammars only allow descriptions as principles. This means that we must quantify explicitly over all variables (tags) in all principles of grammar. In the linguistically oriented HPSG literature, this is not typically done. The difference between common practice and formal explicitness becomes clear when we look at the HEAD FEATURE PRINCIPLE in our current notation. It translates to the following description:

(15) *The* HEAD FEATURE PRINCIPLE *in our syntax*

$$\exists X\ ((\textit{phrase}, \textsc{dtrs}{:}\textit{headed-struc})\text{*>}\ (\textsc{synsem:loc:cat:head}{:}X, \textsc{dtrs:head-dtr:synsem:loc:cat:head}{:}X))$$

To be able to be a bit lazy with simple cases of existential quantification like the HEAD FEATURE PRINCIPLE, we can introduce a notational convention: We will assume that whenever a variable in an initial Σ formula that is meant to be a principle of grammar is not explicitly bound by a quantifier, it is bound by an implicitly intended existential closure over the entire initial Σ formula. Note that this leads to wide scope existential quantification over the variables in question. This is exactly the convention that MoMo follows if you omit (or forget!) quantifiers: MoMo always obeys the convention of existential closure with wide scope of the existential quantifiers over the entire formula.

Convention 1 *If in a principle of grammar, a variable is not explicitly bound by a quantifier, we assume that it is captured by an implicit existential closure with wide scope over the entire initial Σ formula.*

With CONVENTION 1, we can simplify the notation of the HEAD FEATURE PRINCIPLE:

(16) *The* HEAD FEATURE PRINCIPLE *again*

$$(\textit{phrase}, \textsc{dtrs}{:}\textit{headed-struc})\text{*>}\ (\textsc{synsem:loc:cat:head}{:}X, \textsc{dtrs:head-dtr:synsem:loc:cat:head}{:}X))$$

As a principle of grammar, (16) is a notational abbreviation of its fully explicit rendering in (15).

In complex principles of grammar, it is necessary to be careful when we use this notational convention. In particular for beginners it is a good idea to write down all quantifiers explicitly if there is an interaction between several quantifiers in a principle. For example, we cannot appeal to CONVENTION 1 to simplify the notation of the CONTROL THEORY

of Section 2.3, because all of its existential quantifiers are in the scope of some universal quantifiers with wider scope. Omitting the existential quantifiers would lead to an interpretation with existential quantifiers that take scope over the universal quantifiers. But then the meaning of the principle changes entirely, and it no longer means what it is supposed to mean according to Pollard and Sag's formulation in natural language.

Let us close with a second, general observation about notational conventions. The definition of the syntax of boxes and initial formulae succinctly prescribes how the expressions of our logical languages must be bracketed in order to be well-formed. However, being pedantic about the bracketing typically results in unreadable notation for larger descriptions. We will therefore follow general conventions and typically relax our notation in examples. We omit brackets where this cannot lead to confusion. MoMo follows the very same common bracketing conventions. Conversely, it may sometimes enhance readability to insert brackets in places where they are not strictly permitted syntactically. One such place involves expressions after quantifiers. For example, the brackets after the existential quantifier in `^X(X:parrot)` are not in accordance with the syntax, since the tagged box `X:parrot` is not a complex box. Nevertheless, we occasionally employ this notation for clarity, and this is accepted by MoMo as well.

Exercises

Exercise 21 *Let Σ be our familiar initial signature of Section 2.1.2 with lists, birds, and pets. Write a complex grammar using that signature such that for feature structure models of your grammar the following conditions hold:*

1. *Each list contains at least one parrot.*
2. *There are at least two distinct canaries on each list.*
3. *Every list is finite.*

By writing the descriptions that express these statements on a single description card, we obtain a grammar consisting of our signature and of the set of these three principles in MoMo. Feature structures in the graph window for which the modeling test succeeds are feature structure models of this grammar. Upload the file with your grammar to ILIAS.

Consider the statements (1) and (2) in your grammar in turn. For each of the two descriptions, characterize the simple abstract feature structures that satisfy it and the ones that model it.

What do your previous observations tell us about the meaning of the entire grammar?

Exercise 22 *Let Σ again be our familiar initial signature of lists, birds, and pets. Take a simple abstract feature structure under Σ representing a singleton list with a green parrot on it. quant-exercise-233.mmp is a prepared file.*

Consider the following Σ descriptions and decide for each one whether our simple abstract feature structure under Σ satisfies it or is admitted by it. In the descriptions, `V`

stands for the universal quantifier, and ^ for the existential quantifier (MoMo notation for quantifiers).

1. *(a)* `^X(X,color:green).`
   (b) `^X(color:green).`
2. *(a)* `VX((X:parrot)*>(X:color:green)).`
   (b) `^X((X:parrot)*>(X:color:green)).`
3. *(a)* `VX((X:parrot)*>(color:green))`
   (b) `VX((X:parrot)*>(^Y(Y:color:yellow))).`

2.4 Grammars and their Meaning

Abstract

Here, we will extend our formalism for constraint-based grammars one more time. By adding relational expressions and their interpretation in a domain of relational abstract feature structures, we arrive at a formalism that is adequate for the complete formalization of constraint-based HPSG grammars.

With a mathematical characterization of complex grammars and their models in a domain of simple abstract feature structures we have already achieved a significant part of our goals: We started with a language of a rather simple feature logic and augmented it in such a way that we are able to formalize some fairly complex principles of Pollard and Sag's grammar of English. However, going through the informal presentation of that grammar in the appendix to [Pollard and Sag, 1994], we can easily observe that we are still missing our target by one important type of expression: Whenever lists (or sets) are involved in grammatical principles (which is rather often the case), the authors quite often observe some interdependency of shape between two or more list values at syntactic mothers and daughters, and they express it in terms of their *mutual relationship*. Another typical statement of a very similar kind occurs when they sometimes say that an entity of a certain kind must be a *member* of some list (or set). From these statements it is clear that we need some general means that enables us to formulate arbitrary *relations* between entities in linguistic signs.

The good news is that in at least one respect, we are almost there. Equipped with a syntax and semantics that already comprise variables and quantification, the final *syntactic* extension of our language is trivial. The introduction of a new distinction between boxes and formulae, and of constructs like path insertions and functions to determine free variables in formulae which made the syntactic step from initial grammars to complex grammars a rather difficult affair has no counterpart when we move from complex grammars to full grammars (which we will simply call *grammars*).

The other side of the coin is the interpretation of the new expressions. Extending our feature structure semantics in a way appropriate for the intended usage of relational expressions in grammatical principles of the HPSG literature is not as simple as the extension from a variable-free fragment of the formalism to a fragment with variables and quantification, where we were able to maintain the modeling domain of simple abstract feature structures. This time we need to add more structure to our modeling domain, and we need abstract feature structures with a richer internal structure. To obtain an adequate semantics, we will need to proceed from simple abstract feature structures under a given signature to *relational* abstract feature structures under a given signature. Once we have done that, we will focus on how to write and interpret principles of grammar that restrict the meaning of relation symbols like `member` and `append` to their intended interpretation in feature structure models of grammars, and on how the meaning of relational expressions like these two is then used in principles of grammar.

A few introductory examples of prominent grammatical principles with relations of the HPSG grammar of English of Pollard and Sag will remind us of the importance of relational expressions and will give us a fairly good idea of what relations are meant to do in the description of natural languages. The Subcategorization Principle ensures the correct valency of projections of lexical functors when (some of) their arguments are realized as syntactic daughters in phrasal structures: In a headed phrase, the list value of DAUGHTERS | HEAD-DAUGHTER | SYNSEM | LOCAL | CATEGORY | SUBCAT is the concatenation of the list value of SYNSEM | LOCAL | CATEGORY | SUBCAT with the list consisting of the SYNSEM values (in order) of the elements of the list value of DAUGHTERS | COMPLEMENT-DAUGHTERS [Pollard and Sag, 1994, p. 399].

(17) *The* Subcategorization Principle *in AVM notation*

$$\begin{bmatrix} \textit{phrase} \\ \text{DTRS} \;\; \textit{headed-struc} \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} \textit{phrase} \\ \text{SS LOC CAT SUBCAT } \boxed{1} \\ \text{DTRS} \begin{bmatrix} \textit{headed-struc} \\ \text{HEAD-DTR} \begin{bmatrix} \textit{sign} \\ \text{SS LOC CAT SUBCAT } \boxed{3} \end{bmatrix} \\ \text{COMP-DTRS } \boxed{2} \end{bmatrix} \end{bmatrix}$$

$$\wedge \texttt{sign-ss-append}(\boxed{1}, \boxed{2}, \boxed{3})$$

The relation `sign-ss-append` is supposed to ensure the correct relationship between the SUBCAT list of a headed phrase (a list of *synsems*), the list of complement daughters (a list of *signs*), and the SUBCAT list of the head daughter (again a list of *synsems*) as verbosely described by Pollard and Sag. The relation `sign-ss-append` is clearly a close relative of the relation `append`, which is familiar from logic programming. Essentially, `append` says about the relationship between three lists X, Y, and Z that Z starts with exactly the elements of X (ordered as on X), immediately followed by list Y as its final

part. The slight variation on the `append` theme of `sign-ss-append` hints at the flexibility that grammarians expect to be given by the logical formalism for their use of relations: They want to be able to define their own, non-standard relations when they need them. Note also that the AVM notation in (17) appeals to the existential closure convention that we introduced for complex grammars in Section 2.3.3. We will observe appeals to this convention throughout many examples of principles without pointing them out any more.

The ID Principle is in many respects HPSG's counterpart to the phrase structure rules of classical phrase structure grammars and to the phrase structural base of various versions of transformational grammar. The purpose of the ID Principle is to restrict the structural makeup of headed phrases in the same way as phrase structure rules dictate the possible nature of syntactic daughters. According to the ID Principle every headed phrase must satisfy exactly one of the ID schemata [Pollard and Sag, 1994, p. 399 and p. 402–403]. Pollard and Sag assume that there are six ID schemata for English. Schema 6, which we want to investigate more closely, is the Head-Filler Schema. The authors characterize head filler phrases in the following way: The daughters value is an object of sort *head-filler-struc* whose head-daughter | synsem | local | category value satisfies the description [head *verb*[vform *finite*], subcat ⟨⟩], whose head-daughter | synsem | nonlocal | inherited | slash value contains an element token-identical to the filler-daughter | synsem | local value, and whose head-daughter | synsem | nonlocal | to-bind | slash value contains only that element [Pollard and Sag, 1994, p. 403]. In (18), we see a schematic representation of the ID Principle, where the AVM formulae that express the six ID schemata are represented by six place holders in a disjunction in the consequent of the principle. The AVM formula for Schema 6, the Head-Filler Schema, is spelled out in AVM notation.

(18) *The* ID Principle

$$\left[\textsc{dtrs}\ \textit{headed-struc}\right] \rightarrow$$

(Schema1 ∨ Schema2 ∨ Schema3 ∨ Schema4 ∨ Schema5 ∨ Schema6)

where Schema6 stands for

$$\left[\textsc{dtrs}\ \left[\begin{array}{l}\textit{head-filler-struc}\\ \textsc{filler-dtr ss loc}\ \boxed{1}\\ \textsc{head-dtr ss}\ \left[\begin{array}{l}\textsc{loc cat}\ \left[\begin{array}{l}\textsc{head vform}\ \textit{finite}\\ \textsc{subcat}\ \langle\rangle\end{array}\right]\\ \textsc{nonloc}\ \left[\begin{array}{l}\textsc{inherited slash}\ \boxed{2}\\ \textsc{to-bind slash}\ \{\boxed{1}\}\end{array}\right]\end{array}\right]\end{array}\right]\right]$$

$\wedge$ `member`($\boxed{1}$, $\boxed{2}$)

The property of the Head-Filler Schema that is of interest to us is the use of the `member` relation, expressing the membership of the filler's local value in the inherited slash set of the head daughter. And again, just like in the case of the relation `sign-ss-append` for the Subcategorization Principle, it is fairly straightforward to

read and to understand the intended meaning of the relational expression at the end of the formula without knowing anything of the mathematical background necessary to spell it out explicitly. Especially later, in Section 2.4.2, when we get to technical explanations that might seem difficult at first, we should not lose sight of our original, informal understanding that might then guide our understanding of its mathematical reconstruction.

The most complex example of a principle that relies on relations in the theory of English of Pollard and Sag is their BINDING THEORY. Its relative complexity derives from the rich interaction of quantification, negation, and relations that is a peculiar property of the BINDING THEORY. The upshot of it is summarized in the binding principles A–C:

(19) *The* BINDING THEORY *of [Pollard and Sag, 1994, p. 401] informally stated*

Principle A. A locally o-commanded anaphor must be locally o-bound.
Principle B. A personal pronoun must be locally o-free.
Principle C. A nonpronoun must be o-free.

Without going into any details, we can observe that the BINDING THEORY is expressed in terms of relations (o-command, (local) o-binding, (local) o-freedom) that must or must not hold for different kinds of pronouns in syntactic structures. Ultimately it is formulated in terms of relationships between *synsem* entities on SUBCAT lists. Bearing in mind that (local) o-freedom is described by Pollard and Sag as holding between two elements if and only if (local) o-binding does not hold, their binding principles can be rendered as AVM descriptions in the following way:

(20) *The* BINDING THEORY *in AVM notation*

Principle A:

$$\forall X \left(\exists Y \, \texttt{loc-o-command}\Big(Y, {}^{X}[\textsc{loc cont } \textit{ana}]\Big) \rightarrow \exists Z \, \texttt{loc-o-bind}(Z, X)\right)$$

Principle B:

$$\forall X \left({}^{X}[\textsc{loc cont } \textit{ppro}] \rightarrow \neg\exists Y \, \texttt{loc-o-bind}(Y, X)\right)$$

Principle C:

$$\forall X \left({}^{X}[\textsc{loc cont } \textit{npro}] \rightarrow \neg\exists Y \, \texttt{o-bind}(Y, X)\right)$$

Concerning the notation of variables in (20), we have adopted a mixture of notation of the syntax that we have defined and of the informal AVM syntax. If you prefer, you may imagine that the capital letters for variables are replaced by the usual boxed integers.

Independent of the linguistic significance of the binding principles, we can observe the full range of interaction between quantification, negation and relational expressions in them. Moreover, the relations of the BINDING THEORY are idiosyncratic, linguistically motivated relations that, unlike `member` and `append`, do not at all belong to the standard

inventory of logical languages or of programming languages in computer science. Again we are led to conclude that linguists require great freedom in using relational expressions with idiosyncratic meanings that they want to be able to define themselves in their grammars. Their relations are supposed to express linguistic generalizations about languages, and as such their formulation is subject to linguistic research and should not be limited by the formal tools used for that research. It does not seem to be sufficient to provide an inventory of standard relations with their corresponding meaning as part of the logical formalism itself, since it might not provide what a linguist needs for a particular descriptive purpose. The formalism has to provide the means to define the meaning of relation symbols as part of linguistic grammars to fit the linguistic needs of grammarians.

The language and model theory that we will define in the following two sections corresponds almost completely to a feature structure based version of the RSRL (Relational Speciate Re-entrant Language) formalism for HPSG. For sake of simplicity, it omits *chains*, a more flexible interpretation of variables which allows for expressing certain grammatical principles which employ relations with quasi-list arguments more directly than the restricted version of RSRL presented here.

GLOSSARY

2.4.1 Syntax

ABSTRACT

We will add relational expressions to the syntax of formulae. For this simple and straightforward extension, we will also need to extend our notion of signatures, and indicate what the set of free variables of an expression of the extended languages is.

In order to work with relational expressions, we need to extend the initial signatures that we have been working with up to now. Just as was required for attribute symbols, relation symbols must be declared in signatures. In contrast to attributes, it is not sufficient to just declare the names of the relation symbols. We also need to say how many arguments each relation has. For example, the `member` relation usually has two arguments (the entity given as first argument is a member of the set given in the second argument),[9] whereas the standard `append` relation of grammars has three arguments (the concatenation of the two lists given as the first and second argument is the list in the third argument). In our definition of *signatures* in DEFINITION 18, the set of relation symbols is symbolized as $\mathcal{R}$, and the arity function, $\mathcal{AR}$, assigns each relation symbol a positive natural number as the arity of the respective relation.

[9] Occasionally variants of this occur, in particular in the HPSG literature. The `member` relation is also used to indicate membership of elements on lists.

Definition 18 Σ *is a* ***signature*** *iff*

Σ *is a septuple* $\langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{R}, \mathcal{AR} \rangle$,

$\langle \mathcal{G}, \sqsubseteq \rangle$ *is a partial order,*

$$\mathcal{S} = \left\{ \sigma \in \mathcal{G} \;\middle|\; \begin{array}{l} \text{for each } \sigma' \in \mathcal{G}, \\ \text{if } \sigma' \sqsubseteq \sigma \text{ then } \sigma = \sigma' \end{array} \right\},$$

$\mathcal{A}$ *is a set,*

$\mathcal{F}$ *is a partial function from the Cartesian product of* $\mathcal{G}$ *and* $\mathcal{A}$ *to* $\mathcal{G}$,

for each $\sigma_1 \in \mathcal{G}$, *for each* $\sigma_2 \in \mathcal{G}$ *and for each* $\alpha \in \mathcal{A}$,

> *if* $\mathcal{F}(\langle \sigma_1, \alpha \rangle)$ *is defined and* $\sigma_2 \sqsubseteq \sigma_1$
> *then* $\mathcal{F}(\langle \sigma_2, \alpha \rangle)$ *is defined and* $\mathcal{F}(\langle \sigma_2, \alpha \rangle) \sqsubseteq \mathcal{F}(\langle \sigma_1, \alpha \rangle)$,

$\mathcal{R}$ *is a finite set, and*

$\mathcal{AR}$ *is a total function from* $\mathcal{R}$ *to* $\mathbb{N}^+$.

For MoMo signatures, we need new notational conventions to declare relations and their arity. Figure 2.3 illustrates how we do this: The declaration of the sort hierarchy and the appropriateness conditions is followed by the keyword `relations` in a new line, which is in turn followed by the declaration of the relation symbols. The arity of the relations is indicated by notating it behind the relation symbols, separated from them by a slash.

For variety, the signature of Figure 2.3 is new and contains some modifications of declarations that we saw before. One of the purposes of these variations is that we should learn to pay close attention to the details of a signature declaration. Within the bounds of the definition of what signatures are, the symbols we introduce and the appropriateness conditions we specify are completely arbitrary, and we should not be tempted to think that anything is fixed.

In this spirit, note the new way in which we have specified lists in the new signature. Instead of using the features HEAD and TAIL on *nelist*, we used FIRST and REST. FIRST and REST are actually the attribute names that Pollard and Sag employ in their grammar of English for the purpose of encoding lists. In the extensive example of a grammar over a non-linguistic domain in Section 2.4.3, we will take the signature of Figure 2.3 to state a few insightful observations about the relationship between owners and drivers of certain types of cars (or a truck), and to say something about the relationship between the owner, the driver, the passengers in the front seats, the passengers in the back seats, and all the passengers of cars.

Based on the definition of signatures, we define boxes and formulae relative to a given signature. The only difference to the previous definition is the addition of relational expressions to Σ formulae. Σ boxes do not change at all. The two new lines are marked by an asterisk (*) on the margin.

```
type_hierarchy
top
   person likes-best:top
      man
      woman
   car owner:person driver:person passengers:interior
      bmw
      mercedes
      vw
      magirus
   interior front-seats:list back-seats:list total:list
   list
      elist
      nelist first:person rest:list
relations
append/3
member/2
no-repetitions/1
.
```

Figure 2.3: A signature with relations in MoMo notation

Definition 19 *For each signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{R}, \mathcal{AR} \rangle$, $\mathbb{BOX}^{\Sigma}$ *and* $\mathcal{D}^{\Sigma}$ *are the smallest sets such that*

for each $X \in \mathcal{VAR}^{\$}$,

$X \in \mathbb{BOX}^{\Sigma}$,

for each $\sigma \in \mathcal{G}$,

$\sigma \in \mathbb{BOX}^{\Sigma}$,

for each $\alpha \in \mathcal{A}$, *for each* $\delta \in \mathbb{BOX}^{\Sigma}$,

$\alpha : \delta \ \in \mathbb{BOX}^{\Sigma}$,

for each $\delta \in \mathbb{BOX}^{\Sigma}$, $\sim \delta \ \in \mathbb{BOX}^{\Sigma}$,

for each $\delta_1 \in \mathbb{BOX}^{\Sigma}$, *for each* $\delta_2 \in \mathbb{BOX}^{\Sigma}$, $(\delta_1, \delta_2) \in \mathbb{BOX}^{\Sigma}$,

for each $\delta_1 \in \mathbb{BOX}^{\Sigma}$, *for each* $\delta_2 \in \mathbb{BOX}^{\Sigma}$, $(\delta_1; \delta_2) \in \mathbb{BOX}^{\Sigma}$,

for each $\delta_1 \in \mathbb{BOX}^{\Sigma}$, *for each* $\delta_2 \in \mathbb{BOX}^{\Sigma}$, $(\delta_1 *> \delta_2) \in \mathbb{BOX}^{\Sigma}$,

for each $\delta_1 \in \mathbb{BOX}^{\Sigma}$, *for each* $\delta_2 \in \mathbb{BOX}^{\Sigma}$, $(\delta_1 <*> \delta_2) \in \mathbb{BOX}^{\Sigma}$,

for each $\delta \in \mathbb{BOX}^{\Sigma}$,

$\delta \in \mathcal{D}^{\Sigma}$,

for each $X \in \mathcal{VAR}^{\$}$, *for each* $\delta \in \mathbb{BOX}^{\Sigma}$,

$X : \delta \in \mathcal{D}^{\Sigma}$,

for each $X \in \mathcal{VAR}^{\$}$, *for each* $Y \in \mathcal{VAR}^{\$}$,

$X = Y \in \mathcal{D}^{\Sigma}$,

* *for each* $\rho \in \mathcal{R}$, *for each* $X_1 \in \mathcal{VAR}$, ..., *for each* $X_{\mathcal{AR}(\rho)} \in \mathcal{VAR}$,

$\rho\left(X_1, \ldots, X_{\mathcal{AR}(\rho)}\right) \in \mathcal{D}^{\Sigma}$,

for each $X \in \mathcal{VAR}$, *for each* $\delta \in \mathcal{D}^{\Sigma}$,

$\exists X\, \delta \in \mathcal{D}^{\Sigma}$,

for each $X \in \mathcal{VAR}$, *for each* $\delta \in \mathcal{D}^{\Sigma}$,

$\forall X\, \delta \in \mathcal{D}^{\Sigma}$,

for each $\delta \in \mathcal{D}^{\Sigma}$, $\sim \delta \ \in \mathcal{D}^{\Sigma}$,

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$, $(\delta_1, \delta_2) \in \mathcal{D}^{\Sigma}$,

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$, $(\delta_1; \delta_2) \in \mathcal{D}^{\Sigma}$,

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$, $(\delta_1 \texttt{*>} \delta_2) \in \mathcal{D}^{\Sigma}$, *and*

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$, $(\delta_1 \texttt{<*>} \delta_2) \in \mathcal{D}^{\Sigma}$.

We call each element of $\mathbb{BOX}^{\Sigma}$ a Σ *box*, and each element of $\mathcal{D}^{\Sigma}$ a Σ *formula*. The new Σ formulae of the form $\rho\left(X_1, \ldots, X_{\mathcal{AR}(\rho)}\right)$ consist of a relation symbol together with as many variables as the arity of the relation is within round brackets. We call them relational Σ formulae. Here is an example of their usage:

(21) *The* HEAD FILLER SCHEMA *in MoMo notation*[10]

```
^Local ^Set (
dtrs:(head-filler-struc,
      filler-dtr:synsem:local:Local,
      head-dtr:synsem:(local:cat:(head:vform:finite,
                                  subcat:elist),
                       nonlocal:(inherited:slash:Set,
                                 to-bind:slash:element:Local,
                                              rest-set:eset))),
member(Local,Set) )
```

Remember that the HEAD FILLER SCHEMA is not a principle of grammar; it is only a disjunct in the consequent of the ID PRINCIPLE. Since, according to the way we introduced it, appealing to the existential closure convention is strictly speaking only permitted in complete principles of grammar, we made the existential quantification over the two variables `Local` and `Set` explicit in (21).

Just as with complex grammars, our final notion of grammars will not use arbitrary formulae but only the subset of formulae that do not contain any unbound occurrences of variables. We formalize this idea using the function FV. Definition 20 differs from its preliminary version for complex grammars only in the additional base case for relational Σ formulae.

Definition 20 *For each signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{R}, \mathcal{AR} \rangle$, FV *is the total function from* $\mathbb{BOX}^{\Sigma} \cup \mathcal{D}^{\Sigma}$ *to* $Pow\,(\mathcal{VAR})$ *such that*

$FV(\$) = \emptyset$,
for each $X \in \mathcal{VAR}$, $FV(X) = \{X\}$,
for each $\sigma \in \mathcal{G}$, $FV(\sigma) = \emptyset$,
for each $\alpha \in \mathcal{A}$, *for each* $\delta \in \mathbb{BOX}^{\Sigma}$, $FV(\alpha : \delta) = FV(\delta)$,
for each $X \in \mathcal{VAR}^{\$}$, *for each* $\delta \in \mathbb{BOX}^{\Sigma}$, $FV(X : \delta) = FV(X) \cup FV(\delta)$,
for each $X \in \mathcal{VAR}^{\$}$, *for each* $Y \in \mathcal{VAR}^{\$}$, $FV(X = Y) = FV(X) \cup FV(Y)$,
for each $\rho \in \mathcal{R}$, *for each* $X_1 \in \mathcal{VAR}$, ..., *for each* $X_{\mathcal{AR}(\rho)} \in \mathcal{VAR}$,
$FV\left(\rho\left(X_1, \ldots, X_{\mathcal{AR}(\rho)}\right)\right) = FV\left(X_1\right) \cup \ldots \cup FV\left(X_{\mathcal{AR}(\rho)}\right)$,
for each $X \in \mathcal{VAR}$, *for each* $\delta \in \mathcal{D}^{\Sigma}$, $FV(\exists X\, \delta) = FV(\delta) \backslash \{X\}$,
for each $X \in \mathcal{VAR}$, *for each* $\delta \in \mathcal{D}^{\Sigma}$, $FV(\forall X\, \delta) = FV(\delta) \backslash \{X\}$,
for each $\delta \in \mathbb{BOX}^{\Sigma} \cup \mathcal{D}^{\Sigma}$, $FV(\sim \delta) = FV(\delta)$,
for each $\delta_1 \in \mathbb{BOX}^{\Sigma} \cup \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathbb{BOX}^{\Sigma} \cup \mathcal{D}^{\Sigma}$, $FV((\delta_1, \delta_2)) = FV(\delta_1) \cup FV(\delta_2)$,
for each $\delta_1 \in \mathbb{BOX}^{\Sigma} \cup \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathbb{BOX}^{\Sigma} \cup \mathcal{D}^{\Sigma}$, $FV((\delta_1; \delta_2)) = FV(\delta_1) \cup FV(\delta_2)$,

[10] For the purposes of this example, we have assumed a set-signature where a supersort *set* has two immediate, maximally specific subsorts, *eset* (without any appropriate attribute) and *neset*. ELEMENT and REST-SET are appropriate to *neset*, with appropriate values *object* (the top sort of the signature of [Pollard and Sag, 1994]) and *set*, respectively.

16 *for each* $\delta_1 \in \mathbb{BOX}^\Sigma \cup \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathbb{BOX}^\Sigma \cup \mathcal{D}^\Sigma$, $FV((\delta_1 * > \delta_2)) = FV(\delta_1) \cup FV(\delta_2)$,
17 *for each* $\delta_1 \in \mathbb{BOX}^\Sigma \cup \mathcal{D}^\Sigma$, *for each* $\delta_2 \in \mathbb{BOX}^\Sigma \cup \mathcal{D}^\Sigma$, $FV((\delta_1 < * > \delta_2)) = FV(\delta_1) \cup FV(\delta_2)$.

The function FV yields for each Σ box and for each Σ formula the set of variables that occur free in them. In relational expressions, the variables in the argument slots behind the relation symbol occur free. It follows that in grammatical principles they always have to be bound by some quantifiers. Sometimes we can leave it to our notational convention of existential closure to bind the variables. Most notably, the fact that a principle with relations is stated without any kind of explicit quantification somewhere in the linguistically oriented (and, for the most part, quite informal) HPSG literature is by no stretch of the imagination sufficient to believe that existential closure is the right interpretation!

As we have said repeatedly, for grammars we are interested in Σ *descriptions*, the set of Σ formulae that do not contain any unbound variables:

1 **Definition 21** *For each signature* Σ,

2 $$\mathcal{D}_0^\Sigma = \{\delta \in \mathcal{D}^\Sigma \mid FV(\delta) = \emptyset\}.$$

For each signature Σ, we call each member of $\mathcal{D}_0^\Sigma$ a Σ *description*. Our final notion of a grammar is based on signatures and on Σ descriptions:

1 **Definition 22** Γ *is a* ***grammar*** *iff*

2 Γ *is a pair* $\langle \Sigma, \theta \rangle$,
3 Σ *is a signature, and*
4 $\theta \subseteq \mathcal{D}_0^\Sigma$.

Just as before, a grammar is a signature, Σ, together with a set of Σ descriptions. With this definition of our final notion of constraint-based grammars, we can now turn to the task of assigning our relational expressions an adequate meaning in a domain of abstract feature structures.

Exercises

Exercise 23 *Translate the* Subcategorization Principle *from the AVM notation as given in (17) to MoMo syntax. For this and the following exercise you may want to use the mmp file two-principles-ex241.mmp, which already contains the necessary fragment of the signature of Pollard and Sag to write the* Subcategorization Principle *and the principle of the next exercise in MoMo.*

Exercise 24 *Translate* Principle A *of the* Binding Theory *from the AVM notation as given in (20) to MoMo syntax. Be careful about the kinds of expressions that are allowed in the argument slots of relational expressions in our syntax!*

Exercise 25 *In this exercise we want to think again about the meaning of grammars, i.e., the meaning that we ascribe to tuples consisting of a signature and a set of descriptions.*

We are working with the signature that we have favored so far, the signature of Section 2.1.2 with lists, birds and pets (birdsandpets-ex-241.mmp). First we will formulate a set of descriptions over that signature. Together they form our grammar of lists of birds and pets. We have the following principles:

1. *Woodpeckers are red.*
2. *Canaries are yellow.*
3. *Parrots are green.*
4. *Pets are brown.*
5. *Non-empty lists contain one or two elements.*
6. *The first element of each non-empty list is a canary.*

(a) *Write this grammar down in MoMo notation. (you may do this immediately in MoMo by either creating description cards separately for each principle of grammar, or one card that contains all of them. If you do this exercise directly in MoMo, please indicate separately what* precisely *your grammar is.)*

(b) *State the (complete!) set of feature structures admitted by the grammar. (To keep this readable, use pictures of concrete feature structures that correspond to the abstract feature structures admitted by the grammar according to our definitions. The easiest way to create and submit your solution is to use MoMo. With MoMo you can of course also check whether each of your feature structures is really admitted by your grammar. To check whether your feature structures model your grammar, all the descriptions of the grammar have to be on one single description card of MoMo.)*

GLOSSARY

2.4.2 Meaning

Abstract

We will extend our simple abstract feature structures from triples $\langle \beta, \varrho, \lambda \rangle$ to relational abstract feature structures, which are quadruples $\langle \beta, \varrho, \lambda, \xi \rangle$. The new component, ξ, of our feature structures is the relational extension of feature structures. We can interpret a fully formalized version of Pollard and Sag's grammar of English in a domain of relational abstract feature structures by defining a satisfaction function and an admission function for the descriptions and grammars of the previous section, parallel to our former definitions of the semantics of constraint-based grammars.

The real work of defining a relational extension of a feature logic with an interpretation in a domain of feature structures lies in finding an appropriate extension of the notion of feature structures. Standard definitions of feature structures simply do not foresee a representation of arbitrary relations. But this is exactly what we need in order to be able to determine what it means for a feature structure to satisfy a description that contains a relational formula.

In this situation it is a good idea to be very precise about what we want to achieve. In the linguistic examples we saw that relations in HPSG are supposed to express the idea that certain relationships hold between the components of a linguistic structure. In terms of our feature structure semantics of HPSG this means that certain nodes of a feature structure stand in some relationship. Moreover, the nodes are in the relationship if and only if certain properties hold of a certain number of nodes in a feature structure. These relationships between nodes with certain properties are the kind of situation that is best captured in terms of the mathematical concept of relations. Relations are sets of tuples of objects: For example, a three-place relation like `append` is a set of triples. The first element of each triple is in the first argument of the relation, the second element of each triple in the second, and the third element of each triple in the third.

As we have seen, linguists might come up with arbitrary relations. We cannot predict in advance which ones linguists might find useful, since they will want to be able to define for themselves what kinds of nodes in a feature structure stand in these relations. We may view the problem of providing the means to express arbitrary relations between tuples of nodes as another instance of what we always see in our constraint-based grammars: The users of the formalism want to make well-formedness statements about a domain of objects such that, somewhat loosely speaking, the sum of all feature structures that satisfy the given requirements in all their components constitute the intended model of the statements over a signature. The same holds now for relations. The grammar writers tell us with statements (usually called *principles*) in their grammars which nodes in each feature structure are required in a given relation in order for the relation symbol to mean what the users intend. All feature structures that are admitted by the set of these statements fulfill the relevant requirements and exhibit the right behavior of the relations. At the end of the day, a relation symbol like `member` means the right thing according to the grammar writer in all feature structure models of her grammar since feature structure models are licensed by the set of all principles of grammar, including those "principles" that tell us which nodes are supposed to stand in a membership relation. We will see how this works below in an example, after we clarify our technical representation of relations in abstract feature structures.

How can we represent relations in abstract feature structures? Well, we have said that an n-ary relation is a set of n-tuples of objects. For us, an n-ary relation is a set of n-tuples of abstract nodes in abstract feature structures. We know that we might have an arbitrary number of relation symbols in a grammar. Under these circumstances it will work for us if we collect the n-tuples of nodes for our relations and attach a label to each n-tuple in order to signal which n-ary relation is meant by it. This label distinguishes between different relations in case we have several relations of the same arity in our grammar, like `member`

and `loc-o-command` in the grammar of English of Pollard and Sag. As a name tag of each tuple we can of course use the respective relation symbol. A relation in a feature structure is then the set of all n-tuples of nodes with the respective name tag attached to them. As is always the case in our abstract feature structures, the abstract nodes in relation tuples will be represented by the attribute paths that lead to them from the abstract root node.

DEFINITION 23 extends simple abstract feature structures (triples consisting of a basis set, a re-entrancy relation, and a label function) to quadruples. We call the extensions *relational abstract feature structures*. They comprise all components of simple abstract feature structures, defined in exactly the same way as before; plus a relation extension, ξ, whose elements are tuples consisting of a relation symbol (the name tag) together with n paths, one in each argument slot, where the number of arguments is given by the arity of the respective relation as stated in the signature by means of the arity function $\mathcal{AR}$. As developed above, the idea behind this representation is that the paths indicate the abstract nodes that stand in the relation. For each relation symbol ρ of the signature, each relational abstract feature structure under that signature may contain tuples in ξ in which the (abstract) nodes that are in ρ are represented as the paths π that lead to them. These ideas result in the form $\langle \rho, \pi_1, \ldots, \pi_{\mathcal{AR}(\rho)} \rangle$ of these tuples, in which the symbol ρ indicates to which relation the following $\mathcal{AR}(\rho)$-tuple of paths belongs, and each $\mathcal{AR}(\rho)$-tuple of paths indicates which abstract nodes are a tuple of arguments in the relation ρ. Lines 3 through 17 of DEFINITION 23 repeat our old definition of simple abstract feature structures. Lines 18 through 26 are new and concern the relational extension:

1 **Definition 23** *For each signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{R}, \mathcal{AR} \rangle$, $\mathbb{A}$ *is a* ***relational abstract feature structure under*** Σ *iff*

3 $\mathbb{A}$ *is a quadruple* $\langle \beta, \varrho, \lambda, \xi \rangle$,
4 $\beta \subseteq \mathcal{A}^*$,
5 $\varepsilon \in \beta$,
6 *for each* $\pi \in \mathcal{A}^*$, *for each* $\alpha \in \mathcal{A}$,
7 *if* $\pi\alpha \in \beta$ *then* $\pi \in \beta$,
8 ϱ *is an equivalence relation over* β,
9 *for each* $\pi_1 \in \mathcal{A}^*$, *for each* $\pi_2 \in \mathcal{A}^*$, *for each* $\alpha \in \mathcal{A}$,
10 *if* $\pi_1\alpha \in \beta$ *and* $\langle \pi_1, \pi_2 \rangle \in \varrho$ *then* $\langle \pi_1\alpha, \pi_2\alpha \rangle \in \varrho$,
11 λ *is a total function from* β *to* $\mathcal{S}$, *and*
12 *for each* $\pi_1 \in \mathcal{A}^*$, *for each* $\pi_2 \in \mathcal{A}^*$,
13 *if* $\langle \pi_1, \pi_2 \rangle \in \varrho$ *then* $\lambda(\pi_1) = \lambda(\pi_2)$.

14 *for each* $\pi \in \mathcal{A}^*$, *for each* $\alpha \in \mathcal{A}$,

15 *if* $\pi\alpha \in \beta$ *then* $\mathcal{F}(\langle\lambda(\pi), \alpha\rangle)$ *is defined and* $\lambda(\pi\alpha) \sqsubseteq \mathcal{F}(\langle\lambda(\pi), \alpha\rangle)$,

16 *for each* $\pi \in \mathcal{A}^*$, *for each* $\alpha \in \mathcal{A}$,

17 *if* $\pi \in \beta$ *and* $\mathcal{F}(\langle\lambda(\pi), \alpha\rangle)$ *is defined then* $\pi\alpha \in \beta$,

18 $\xi \subseteq \mathcal{R} \times \beta^*$,

19 *for each* $\rho \in \mathcal{R}$, *for each* $\pi_1 \in \beta$,..., *for each* $\pi_n \in \beta$,

20 *if* $\langle\rho, \pi_1, \ldots, \pi_n\rangle \in \xi$ *then* $n = \mathcal{AR}(\rho)$, *and*

21 *for each* $\rho \in \mathcal{R}$, *for each* $\pi_1 \in \beta$, ..., *for each* $\pi_n \in \beta$, *for each* $\pi'_1 \in \beta$, ..., *for each*
22 $\pi'_n \in \beta$,

23 *if* $\langle\rho, \pi_1, \ldots, \pi_n\rangle \in \xi$, *and*
24 *for each* $i \in \{1, \ldots, n\}$,
25 $\pi_i \in \beta$ *and* $\langle\pi_i, \pi'_i\rangle \in \varrho$,
26 *then* $\langle\rho, \pi'_1, \ldots, \pi'_n\rangle \in \xi$.

The last condition (in the last 6 lines of DEFINITION 23 (lines 21–26)) on the tuples in ξ is saying that if a path π occurs in the nth argument position of a relation then all other paths π' that lead to the same abstract node must also occur in that position, all other argument slots being equal. Of course, this has to happen for all argument slots, which means that in the end, every argument occurs with all possible paths for each abstract node, and in all possible combinations with all other paths representing the abstract nodes in the other argument positions. Another way of expressing the same idea is to say that if one member of an equivalence class in ϱ occurs in an argument position of a relation ρ, then all other members of that equivalence class must also occur with the same combination of the other arguments of the relation ρ, since it is the entire equivalence class in ϱ that represents the abstract node.[11]

Let us look at what this means in more detail in a very simple MoMo example. Assume that we have a signature that contains sorts and appropriateness conditions for the description of lists. Also assume that we have a sort *element* for atomic objects that we want to use as elements on lists, and the relation symbol `member`. We can see in Figure 2.4 what this looks like in MoMo.

Now assume we have a feature structure representing a list with three elements (you can look at the following example in MoMo by downloading and opening the file

[11] This observation suggests that we might have formalized the relational extension of abstract feature structures just as well by using the equivalence classes of ϱ as arguments of each relation. In a sense, we would thus have built the conditions of the lines 21–26 directly into a reformulation of the lines 19–20.

```
type_hierarchy
top
   list
      elist
      nelist first:element rest:list
   element
relations
member/2
.
```

Figure 2.4: A simple signature for lists in MoMo notation

membership-princ242. Choose the interpretation *Example for list membership*. You must activate the menu option "Show Node Numbers" in the interpretation window of *Example for list membership* in order to see the node numbers that we will keep referring to).[12] A list feature structure with three elements has three *nelist* nodes, which we will call A0, A1 and A2. A0 is the root node of the feature structure, A1 is the REST value of A0, and A2 is the REST value of A1. From A2 a REST arc leads to an *elist* node signaling the end of the list. Each *nelist* node has an outgoing FIRST arc leading to an *element* node. We call the FIRST value of A0 A4, the FIRST value of A1 is A5, and the FIRST value of A2 is A6.

Given this basic scaffold of our relational feature structure, what should be in the representation of `member` so that it corresponds to our intuitive understanding of a list membership relation?

The description fixing the intended meaning of `member` would probably say something like: 'A node is on a list if it is the FIRST value of a node that we can reach following a (possibly empty) sequence of REST arcs.' Which nodes are then the members of the list A0? Well, it should be A4 (its direct FIRST value), A5 (because it is the FIRST value of A1, which we can reach from A0 following one REST arc), and A6, because it is the FIRST value of A2, which we can reach from A0 following two REST arcs. But that is not all, since there are more *nelist* nodes in our feature structure! We also have to list the members of the *nelist* nodes A1 and of A2 in order to get an intuitively correct representation of the membership relations in the overall feature structure. Thus, we also put the tuples $\langle A5, A1\rangle$, $\langle A6, A1\rangle$, and $\langle A6, A2\rangle$ in our representation of member. This being completed, we have constructed a feature structure representation of a list with three elements and an intuitively correct representation of the list-membership relation in it. Whereas the tuples in the relation are represented by paths in abstract feature structures, the concrete feature structure representation of them is accomplished by referring to the concrete nodes and indicating which of them are in the `member` relation.

Note that it was completely left to us to decide which nodes we would put into the

[12] Once more we are introducing a concept for abstract feature structures using the more gentle concrete feature structures. By now it should be obvious to the reader how to translate the terminology of concrete feature structures into the terminology of corresponding abstract feature structures.

tuples of `member`: Any pair of nodes may occur there according to the signature and the definition of relational abstract feature structures. We decided to put certain nodes into the pairs and leave others out because of our understanding of what the member relation should be. Later we will have to write a MEMBERSHIP PRINCIPLE as part of our theory of grammar. Relational abstract feature structures representing lists with three elements that are licensed by that principle will then have to look just like the one that we have just investigated: Only if the nodes that we put into the `member` relation are in the relation and no others, will the feature structure be admitted by the MEMBERSHIP PRINCIPLE. However, before we present and discuss a MEMBERSHIP PRINCIPLE for lists, we have to say how relational abstract feature structure satisfaction works.

In the definition of the satisfaction functions, we must refer to Σ insertions. Fortunately, upon inspection, we find that we do not have to change anything substantial in their definition. It is sufficient to minimally modify the old definition by changing the former reference to *initial signatures* to a reference to *signatures*. Since the new components of signatures, $\mathcal{R}$ and $\mathcal{AR}$, are referred to nowhere else in the definitions, the definitions do not change in any significant way, and we do not repeat them here.

What we have to modify, though, is the definition of π reducts of feature structures, since we need to say what the π reduct of the representations of relations in the relational extension of feature structures is. To get the π reduct of ξ, we take each tuple in it and remove the initial string π from each path in the paths representing the nodes that stand in the relation. Tuples that contain an argument with a path that does not start with π get thrown out and are not in the π reduct of ξ. This is entirely parallel to what happens in π reducts of the re-entrancy relation, ϱ, except that the re-entrancy relation is always a pair, whereas a relation is a tuple with an arbitrary positive finite number of arguments.

Definition 24 *For each signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{R}, \mathcal{AR} \rangle$*, for each* $\mathbb{A} = \langle \beta, \varrho, \lambda, \xi \rangle \in \mathbb{AFS}^{\Sigma}$*, for each* $\pi \in \mathcal{A}^*$*,*

$$\beta/\pi = \{\pi' \in \mathcal{A}^* \mid \pi\pi' \in \beta\},$$
$$\varrho/\pi = \{\langle \pi_1, \pi_2 \rangle \in \mathcal{A}^* \times \mathcal{A}^* \mid \langle \pi\pi_1, \pi\pi_2 \rangle \in \varrho\},$$
$$\lambda/\pi = \{\langle \pi', \sigma \rangle \in \mathcal{A}^* \times \mathcal{S} \mid \langle \pi\pi', \sigma \rangle \in \lambda\},$$
$$\xi/\pi = \{\langle \rho, \pi_1, \ldots, \pi_{\mathcal{AR}(\rho)} \rangle \in \mathcal{R} \times (\beta/\pi)^* \} \mid \langle \rho, \pi\pi_1, \ldots, \pi\pi_{\mathcal{AR}(\rho)} \in \xi \rangle\}, \textit{ and}$$
$$\mathbb{A}/\pi = \langle \beta/\pi, \varrho/\pi, \lambda/\pi, \xi/\pi \rangle.$$

As before, we can show that if a relational abstract feature structure under some signature Σ, $\mathbb{A}$, contains path π as an element of its basis set then the π reduct of $\mathbb{A}$ is also a relational abstract feature structure under Σ.

We can now immediately proceed to the final definition of the satisfaction functions. For each signature Σ and each Σ insertion $\iota \in \mathrm{I}_{\Sigma}$, the relational abstract feature structure satisfaction function under ι in Σ, Δ^{ι}_{Σ}, is defined inductively over all Σ formulae. As in the previous version, the definition of the semantics for Σ formulae follows the syntactic structure of our expressions and uses a second feature structure satisfaction function under

ι in Σ for Σ boxes, Ξ^{ι}_{Σ}, which gives us the relational feature structure denotation of Σ boxes under ι. Since the structure of Σ boxes has not changed at all, Ξ^{ι}_{Σ} does not change either. The only difference to the previous satisfaction functions is the additional clause for relational Σ formulae in lines 27 and 28.

Definition 25 *For each signature* $\Sigma = \langle \mathcal{G}, \sqsubseteq, \mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{R}, \mathcal{AR} \rangle$*, for each* Σ *insertion* $\iota \in \mathrm{I}_{\Sigma}$*,* Ξ^{ι}_{Σ} *is the total function from the threefold Cartesian product of* $Pow(\mathcal{A}^* \times \mathcal{A}^*)$*,* $\mathcal{A}^*$ *and* $\mathbb{BOX}^{\Sigma}$ *to* $Pow(\mathbb{AFS}^{\Sigma})$ *and* Δ^{ι}_{Σ} *is the total function from* $\mathcal{D}^{\Sigma}$ *to* $Pow(\mathbb{AFS}^{\Sigma})$ *such that*

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$*, for each* $\pi \in \mathcal{A}^*$*, for each* $X \in \mathcal{VAR}^{\$}$*,*

$$\Xi^{\iota}_{\Sigma}(\varrho_0, \pi, X) = \Big\{ \langle \beta, \varrho, \lambda, \xi \rangle \in \mathbb{AFS}^{\Sigma} \;\Big|\; \langle \iota(X), \pi \rangle \in \varrho_0 \Big\},$$

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$*, for each* $\pi \in \mathcal{A}^*$*, for each* $\sigma \in \mathcal{G}$*,*

$$\Xi^{\iota}_{\Sigma}(\varrho_0, \pi, \sigma) = \left\{ \langle \beta, \varrho, \lambda, \xi \rangle \in \mathbb{AFS}^{\Sigma} \;\middle|\; \begin{array}{l} \text{for some } \sigma' \in \mathcal{S}, \\ \quad \langle \varepsilon, \sigma' \rangle \in \lambda, \text{ and} \\ \quad \sigma' \sqsubseteq \sigma \end{array} \right\},$$

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$*, for each* $\pi \in \mathcal{A}^*$*, for each* $\alpha \in \mathcal{A}$*, for each* $\delta \in \mathbb{BOX}^{\Sigma}$*,*

$$\Xi^{\iota}_{\Sigma}(\varrho_0, \pi, \alpha : \delta) = \left\{ \langle \beta, \varrho, \lambda, \xi \rangle \in \mathbb{AFS}^{\Sigma} \;\middle|\; \begin{array}{l} \alpha \in \beta, \text{ and} \\ \langle \beta, \varrho, \lambda, \xi \rangle / \alpha \in \Xi^{\iota}_{\Sigma}(\varrho_0, \pi\alpha, \delta) \end{array} \right\},$$

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$*, for each* $\pi \in \mathcal{A}^*$*, for each* $\delta \in \mathbb{BOX}^{\Sigma}$*,*

$$\Xi^{\iota}_{\Sigma}(\varrho_0, \pi, \sim \delta) = \mathbb{AFS}^{\Sigma} \backslash \Xi^{\iota}_{\Sigma}(\varrho_0, \pi, \delta),$$

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$*, for each* $\pi \in \mathcal{A}^*$*, for each* $\delta_1 \in \mathbb{BOX}^{\Sigma}$*, for each* $\delta_2 \in \mathbb{BOX}^{\Sigma}$*,*

$$\Xi^{\iota}_{\Sigma}(\varrho_0, \pi, (\delta_1, \delta_2)) = \Xi^{\iota}_{\Sigma}(\varrho_0, \pi, \delta_1) \cap \Xi^{\iota}_{\Sigma}(\varrho_0, \pi, \delta_2),$$

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$*, for each* $\pi \in \mathcal{A}^*$*, for each* $\delta_1 \in \mathbb{BOX}^{\Sigma}$*, for each* $\delta_2 \in \mathbb{BOX}^{\Sigma}$*,*

$$\Xi^{\iota}_{\Sigma}(\varrho_0, \pi, (\delta_1; \delta_2)) = \Xi^{\iota}_{\Sigma}(\varrho_0, \pi, \delta_1) \cup \Xi^{\iota}_{\Sigma}(\varrho_0, \pi, \delta_2),$$

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$*, for each* $\pi \in \mathcal{A}^*$*, for each* $\delta_1 \in \mathbb{BOX}^{\Sigma}$*, for each* $\delta_2 \in \mathbb{BOX}^{\Sigma}$*,*

$$\Xi^{\iota}_{\Sigma}(\varrho_0, \pi, (\delta_1 *\!\!> \delta_2)) = \Big(\mathbb{AFS}^{\Sigma} \backslash \Xi^{\iota}_{\Sigma}(\varrho_0, \pi, \delta_1)\Big) \cup \Xi^{\iota}_{\Sigma}(\varrho_0, \pi, \delta_2),$$

for each $\varrho_0 \in Pow(\mathcal{A}^* \times \mathcal{A}^*)$, *for each* $\pi \in \mathcal{A}^*$, *for each* $\delta_1 \in \mathbb{BOX}^\Sigma$, *for each* $\delta_2 \in \mathbb{BOX}^\Sigma$,

$$\Xi^\iota_\Sigma(\varrho_0, \pi, (\delta_1 \texttt{<*>} \delta_2)) = \\ \Big(\Big(\mathbb{AFS}^\Sigma \backslash \Xi^\iota_\Sigma(\varrho_0, \pi, \delta_1)\Big) \cap \Big(\mathbb{AFS}^\Sigma \backslash \Xi^\iota_\Sigma(\varrho_0, \pi, \delta_2)\Big)\Big) \cup \\ (\Xi^\iota_\Sigma(\varrho_0, \pi, \delta_1) \cap \Xi^\iota_\Sigma(\varrho_0, \pi, \delta_2)),\ \textit{and}$$

for each $X \in \mathcal{VAR}^\$$,

$$\Delta^\iota_\Sigma(X) = \Big\{ \langle \beta, \varrho, \lambda, \xi \rangle \in \mathbb{AFS}^\Sigma \,\Big|\, \iota(X) \in \beta \Big\},$$

for each $\sigma \in \mathcal{G}$,

$$\Delta^\iota_\Sigma(\sigma) = \left\{ \langle \beta, \varrho, \lambda, \xi \rangle \in \mathbb{AFS}^\Sigma \,\middle|\, \begin{array}{l} \textit{for some } \sigma' \in \mathcal{S}, \\ \quad \langle \varepsilon, \sigma' \rangle \in \lambda, \textit{ and} \\ \quad \sigma' \sqsubseteq \sigma \end{array} \right\},$$

for each $\alpha \in \mathcal{A}$, *for each* $\delta \in \mathbb{BOX}^\Sigma$,

$$\Delta^\iota_\Sigma(\alpha : \delta) = \left\{ \langle \beta, \varrho, \lambda, \xi \rangle \in \mathbb{AFS}^\Sigma \,\middle|\, \begin{array}{l} \alpha \in \beta, \\ \langle \beta, \varrho, \lambda, \xi \rangle / \alpha \in \Xi^\iota_\Sigma(\varrho, \alpha, \delta) \end{array} \right\},$$

for each $X \in \mathcal{VAR}^\$$, *for each* $\delta \in \mathbb{BOX}^\Sigma$,

$$\Delta^\iota_\Sigma(X : \delta) = \left\{ \langle \beta, \varrho, \lambda, \xi \rangle \in \mathbb{AFS}^\Sigma \,\middle|\, \begin{array}{l} \iota(X) \in \beta, \textit{ and} \\ \langle \beta, \varrho, \lambda, \xi \rangle / \iota(X) \in \Xi^\iota_\Sigma(\varrho, \iota(X), \delta) \end{array} \right\},$$

for each $X \in \mathcal{VAR}^\$$, *for each* $Y \in \mathcal{VAR}^\$$,

$$\Delta^\iota_\Sigma(X = Y) = \Big\{ \langle \beta, \varrho, \lambda, \xi \rangle \in \mathbb{AFS}^\Sigma \,\Big|\, \langle \iota(X), \iota(Y) \rangle \in \varrho \Big\},$$

for each $\rho \in \mathcal{R}$, *for each* $X_1 \in \mathcal{VAR}$, ..., *for each* $X_{\mathcal{AR}(\rho)} \in \mathcal{VAR}$,

$$\Delta^\iota_\Sigma(\rho(X_1, \ldots, X_{\mathcal{AR}(\rho)})) = \Big\{ \langle \beta, \varrho, \lambda, \xi \rangle \in \mathbb{AFS}^\Sigma \,\Big|\, \langle \rho, \iota(X_1), \ldots, \iota(X_{\mathcal{AR}(\rho)}) \rangle \in \xi \Big\},$$

for each $X \in \mathcal{VAR}$, *for each* $\delta \in \mathcal{D}^\Sigma$,

$$\Delta^\iota_\Sigma(\exists X\, \delta) = \left\{ \langle \beta, \varrho, \lambda, \xi \rangle \in \mathbb{AFS}^\Sigma \,\middle|\, \begin{array}{l} \textit{for some } \pi \in \beta, \\ \quad \langle \beta, \varrho, \lambda, \xi \rangle \in \Delta^{\iota[\frac{\pi}{X}]}_\Sigma(\delta) \end{array} \right\},$$

for each $X \in \mathcal{VAR}$, *for each* $\delta \in \mathcal{D}^\Sigma$,

$$\Delta^\iota_\Sigma(\forall X\, \delta) = \left\{ \langle \beta, \varrho, \lambda, \xi \rangle \in \mathbb{AFS}^\Sigma \,\middle|\, \begin{array}{l} \textit{for each } \pi \in \beta, \\ \quad \langle \beta, \varrho, \lambda, \xi \rangle \in \Delta^{\iota[\frac{\pi}{X}]}_\Sigma(\delta) \end{array} \right\},$$

for each $\delta \in \mathcal{D}^{\Sigma}$, $\Delta^{\iota}_{\Sigma}(\sim \delta) = \mathbb{AFS}^{\Sigma} \backslash \Delta^{\iota}_{\Sigma}(\delta)$,

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$, $\Delta^{\iota}_{\Sigma}((\delta_1, \delta_2)) = \Delta^{\iota}_{\Sigma}(\delta_1) \cap \Delta^{\iota}_{\Sigma}(\delta_2)$,

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$, $\Delta^{\iota}_{\Sigma}((\delta_1; \delta_2)) = \Delta^{\iota}_{\Sigma}(\delta_1) \cup \Delta^{\iota}_{\Sigma}(\delta_2)$,

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$, $\Delta^{\iota}_{\Sigma}((\delta_1 \texttt{*>} \delta_2)) = \left(\mathbb{AFS}^{\Sigma} \backslash \Delta^{\iota}_{\Sigma}(\delta_1)\right) \cup \Delta^{\iota}_{\Sigma}(\delta_2)$, *and*

for each $\delta_1 \in \mathcal{D}^{\Sigma}$, *for each* $\delta_2 \in \mathcal{D}^{\Sigma}$,
$\Delta^{\iota}_{\Sigma}((\delta_1 \texttt{<*>} \delta_2)) = \left(\left(\mathbb{AFS}^{\Sigma} \backslash \Delta^{\iota}_{\Sigma}(\delta_1)\right) \cap \left(\mathbb{AFS}^{\Sigma} \backslash \Delta^{\iota}_{\Sigma}(\delta_2)\right)\right) \cup (\Delta^{\iota}_{\Sigma}(\delta_1) \cap \Delta^{\iota}_{\Sigma}(\delta_2))$.

We call Δ^{ι}_{Σ} the *relational abstract feature structure satisfaction function under* ι *in* Σ, and say $\mathbb{A}$ satisfies δ under ι in Σ if and only if $\mathbb{A} \in \Delta^{\iota}_{\Sigma}(\delta)$. Intuitively speaking, a relational feature structure satisfies a relational expression if and only if the paths inserted for the variables in the relational expression lead to abstract nodes that stand in the relation according to the representation of the relation given by the elements of ξ.

Before we take a look at how the satisfaction function works for relational expressions in grammars, let us first define the corresponding admission function. It does not change at all compared to our previous version in Section 2.3 for complex grammars:

Definition 26 *For each signature* Σ,

M_{Σ} *is the total function from* $Pow(\mathcal{D}^{\Sigma}_{0})$ *to* $Pow(\mathbb{AFS}^{\Sigma})$ *such that for each* $\theta \subseteq \mathcal{D}^{\Sigma}_{0}$,

$$M_{\Sigma}(\theta) = \left\{ \langle \beta, \varrho, \lambda, \xi \rangle \in \mathbb{AFS}^{\Sigma} \;\middle|\; \begin{array}{l} \textit{for each } \pi \in \beta, \\ \textit{for each } \iota \in \mathrm{I}_{\Sigma}, \textit{ and} \\ \textit{for each } \delta \in \theta, \\ \quad \langle \beta, \varrho, \lambda, \xi \rangle / \pi \in \Delta^{\iota}_{\Sigma}(\delta) \end{array} \right\}.$$

For each signature Σ, we call M_{Σ} the *relational abstract feature structure admission function in* Σ. A simple abstract feature structure under Σ, $\mathbb{A}$, is in the set of relational abstract feature structures under Σ admitted by a Σ theory θ exactly if $\mathbb{A}$ and every possible π reduct of $\mathbb{A}$ satisfy every description in θ. Except for the extended notion of abstract feature structures, this is the definition that we already had before with simpler description languages. Given a grammar, $\langle \Sigma, \theta \rangle$, we regard the set of relational abstract feature structures admitted by θ, $M_{\Sigma}(\theta)$, as the intended meaning of $\langle \Sigma, \theta \rangle$. Pollard and Sag say that the members of $M_{\Sigma}(\theta)$ are mathematical representations of the object types of the language that the grammar is about.

How do we use the semantics of relational expressions to write principles that give relation symbols the meanings we want them to have in feature structure models of our grammars? It is clear that once we provide an answer to this question and are able to force relation symbols to have the meanings we want them to have, it is no longer difficult to use the relation symbols appropriately in the Subcategorization Principle and in the Head-Filler Schema, or even in principles as complex as the Binding Theory. Indeed, we have already seen how Pollard and Sag do that in their grammar of English.

```
VX VY(member(Y,X) <*> X:first:Y;
                      ^Z (X:rest:Z,
                          member(Y,Z)))
```

Figure 2.5: A MEMBERSHIP PRINCIPLE for lists

In Figure 2.5 we see an example of the relevant new type of grammatical principle, the MEMBERSHIP PRINCIPLE, which we state relative to the signature of Figure 2.4.

What does the MEMBERSHIP PRINCIPLE mean? Its effect on feature structure models of a grammar that contains it can be characterized as follows. Under the feature structure admission function, it says that a feature structure $\mathbb{A}$ is admitted by it exactly if the following holds: For each pair `Y` and `X` of (abstract) nodes in $\mathbb{A}$, they are in the `member` relation if and only if `Y` is the FIRST value of `X` (i.e., `Y` is the first element on list `X`), or there is a `Z` which happens to be the REST value of `X`, and `Y` is in the `member` relation with `Z` (i.e., we find `Y` on the rest of the list).[13] But that means that if the MEMBERSHIP PRINCIPLE is satisfied for each node in a relational abstract feature structure, all of those nodes that we want, and, mind you, only those nodes, are, indeed, in the relation. In other words, the relational abstract feature structure admission function will take care of enforcing the right meaning of the `member` symbol. We can see this by activating the description card *membership principle* in the MoMo file membership-princ242.mmp and pressing the button *Check modeling* in the interpretation window *Example for list membership*, which contains a feature structure representation of a list with a correct relational extension representing the member relation: Our feature structure is indeed admitted by the MEMBERSHIP PRINCIPLE.

The situation changes immediately and the feature structure becomes ill-formed under admission when we take out tuples of nodes or add other tuples to the relational feature structure. Then the MEMBERSHIP PRINCIPLE is no longer valid, because there are either pairs missing in the relation or there are pairs in it that do not belong there according to this principle. You can easily check this effect yourself by doing model checking in MoMo relative to the description *membership principle* in membership-princ242.mmp with the relational feature structures in *missing pairs* and *too many pairs*.

From our discussion we conclude that the full grammar of English of Pollard and Sag must in fact contain more principles of grammar than they actually state in their appendix. For each relation that they use in their grammar, we must add a principle that fixes its intended meaning in models of the grammar. Before we can do that, we also have to augment their signature with declarations of the necessary relation symbols and the arity of the relations, just as we have done it in our own definitions and examples. In this respect, the signature in the appendix to [Pollard and Sag, 1994] is incomplete.

Our syntax and semantics gives us a complete formalization of HPSG as it was presented

[13] This description also applies to nodes that are both the first member of a list and re-occur at some later point on the list.

informally in [Pollard and Sag, 1994]. The way we defined our formal language and its interpretation in a domain of relational abstract feature structures in Section 2.4 allows a fairly direct formalization of the entire grammar of English of [Pollard and Sag, 1994]. As things stand at the moment, there is only a very small number of open questions or minor complications left. These open issues concern the description of sets, so-called parametric sorts (or parametric types, according to the usual terminology of computer science), certain relations between lists that require an extension of what counts as list-values in our abstract feature structures (for directly expressing these relations as they are stated in the linguistic literature), and certain conventional notational devices that one will want to be able to use.

Some of these remaining issues will be addressed in the next section, such as parametric sorts and special notational conventions for the description of lists. Others, such as *chains* for non-standard uses of list-like structures in the arguments of relations, and an explanation of how to describe set values, fall outside the scope of the present course. Therefore, we must refer the interested reader to the relevant literature outside of this course to find information on satisfactory solutions to these remaining problems.

Exercises

Exercise 26 *Write down the relational abstract feature structure whose concrete counterpart we discussed in the text below Figure 2.4 (*Example for list membership *in the MoMo file membership-princ242.mmp) in the set notation of the definition of relational abstract feature structures above, i.e., state its sets β, ϱ, λ, and ξ.*

Exercise 27 *Let us call the relational abstract feature structure of the previous exercise $\mathbb{A}_1$. State the* REST *reduct of $\mathbb{A}_1$, $\mathbb{A}_1$/*REST *in the set notation of the definition.*

Exercise 28 *The relational feature structure in the interpretation window* too many pairs *of membership-princ242.mmp is not admitted by the* MEMBERSHIP PRINCIPLE. *MoMo tells us with black circles around the relevant nodes which nodes do not satisfy the description. Give a very short explanation for each of these nodes saying why it does not satisfy the* MEMBERSHIP PRINCIPLE.

Exercise 29 *Take the signature of the file append-exercise242.mmp. It contains the relation symbol* ***append****, which is assigned arity 3 by $\mathcal{AR}$.*

Write an APPEND PRINCIPLE *over that signature in MoMo notation that says that in every feature structure model of a grammar containing that principle, all the following triples of nodes are in the* ***append*** *relation: The third element in the triple represents a list that starts with the elements of the list represented in the first argument (if this list has any elements) and continues as the list represented in the second argument. If the list represented in the first argument slot of the tuple has no elements, then the list represented in the third argument slot is identical with the list represented in the second argument slot. If you like, you are welcome to submit your solution in an mmp file, but you are not required*

to do so.

GLOSSARY

2.4.3 An Extended Example for Grammars

ABSTRACT

To illustrate our final formalism for constraint-based grammars, we will discuss an example from a grammar of a non-linguistic domain.

In Exercise 25 of Section 2.4.1 we investigated a *complex grammar* by taking a signature, formalizing a set of six principles as descriptions and then writing up the set of feature structures licensed by the resulting grammar.

Now we want to look at another example, but this time with a different signature. The crucial difference is that the present signature includes relations. The grammar that we will investigate contains the full range of syntactic constructs that occur in the grammar of English of Pollard and Sag. Figure 2.6 shows the signature that we assume:

```
type_hierarchy
top
   person likes-best:top
      man
      woman
   car owner:person driver:person passengers:interior
      bmw
      mercedes
      vw
      magirus
   interior front-seats:list back-seats:list total:list
   list
      elist
      nelist first:person rest:list
relations
append/3
member/2
no-repetitions/1
.
```

Figure 2.6: A signature for a theory of people and cars

We can now formulate a theory regarding our small domain of cars, people, people who own and drive cars, and people in cars. First we state this theory in plain natural language.

Then we think about a possible formalization of our statements on the basis of the signature in Figure 2.6. The complete example is available in the file cars-and-people243.mmp.

(22) a. The driver always sits in the front seat.

b. At most two people sit in the front seats.

c. Nobody sits in two different seats at the same time.

d. At most three people sit in the back seats.

e. The total number of people in a car equals the number of people in the front seats plus the number of people in the back seats.

f. A mercedes owner is always the driver.

g. A truck owner is never the truck driver.

h. Everybody likes a person or some car best.

i. A male BMW driver likes his car best.

Let us first state descriptions that we may use to formalize the statements in (22). We will discuss them afterwards.

(23) a. `VD(driver:D *> ^L(passengers:front-seats:L,member(D,L)))`

b. `car *> passengers:front-seats:(rest:elist;rest:rest:elist)`

c. `car *> passengers:total:X,no-repetitions(X)`

d. `car *> passengers:back-seats:(elist;rest:elist;`
`rest:rest:elist;rest:rest:rest:elist)`

e. `car *> passengers:front-seats:X,passengers:back-seats:Y,`
`passengers:total:Z,append(X,Y,Z)`

f. `mercedes *> (owner:X,driver:X)`

g. `magirus *> ∼ ^X(owner:X,driver:X)`

h. `person *> likes-best:(person;car)`

i. `(bmw,driver:man) *> $:driver:likes-best:X,X=$`

Ad (23a): The formalization says that the driver is one of the persons on the list of people in front seats. Using the `member` relation, we do not make any assumptions as to the position of the driver on that list. An alternative formalization without a relation might use a different principle which says that the driver is always the first person on the front seats list.

Ad (23b): This is a trivial way of saying that the front seats list is either of length one or of length two. Why don't we allow lists of length zero, since the natural language formulation only says *at most two*? The answer is that we are a little loose here, and assume, justified by the previous principle, that there is always at least one person in the front seats, namely the driver.

Ad (23c): Note the implicit wide scope existential quantification over the variable `X`. The relation `no-repetitions` is supposed to hold for all lists on which no element occurs twice. Why do we say this about the list value of TOTAL only, but not about the values of FRONT-SEATS and BACK-SEATS? Again we are relying on the effects of a different principle: Principle (23e) says that the total of all people is the sum of people in the front and back seats. As we will see below, we achieve this by appending the list of people in back seats to the list of people in front seats to get the list of all people in the car. That nobody appears twice on the TOTAL list will thus be sufficient to ensure that nobody sits in more than one seat.

This kind of economy in formulating generalizations according to observations about natural languages is found quite often in grammars.

Ad (23d): The formalization of the restriction of the number of people on the back seats corresponds to the formalization of the restriction of the number of people in the front seats, except that this time we allow zero passengers.

Ad (23e): We use the `append` relation to say that the list of all people in a car is composed of the list of the people in the front and back seats.

Ad (23f): Note again the appeal to the existential closure convention for binding off the variable `X`.

Ad (23g): In this case we cannot appeal to the existential closure convention for the intended reading: We want to say that *there exists no* `X`, and in this statement the negation takes scope over the existential quantification. Therefore we have to make the existential quantifier in the scope of the negation operator, $\sim$, explicit.

Ad (23h): In the signature there is no sort that allows us to generalize exactly over cars and people. For that reason we need the disjunction `person;car` in the consequent.

Ad (23i): Note the use of the reserved symbol $. Remember that we said that this is a special variable whose denotation we defined as referring to the root node. Thus the consequent of this principle says that `X`, which is the DRIVER LIKES-BEST value of the described feature structure, is identical to the root node. In the relevant case, when the antecedent is true, $ refers to the *bmw* in question.

Have we finished formalizing the grammar expressed by the signature of Figure 2.6 and the informal principles in (22a)–(22i)? We have, after all, expressed all principles by descriptions of the description language generated by the signature. But that isn't all! What about the relations and their meaning? To enforce the intended meaning of `append`,

member, and no-repetitions, we have to make explicit what we mean by these relational symbols in models of our grammar. Therefore, to complete our formalization we have to write three principles which determine the correct meaning for our relation symbols in models of the grammar.

```
VXVY(member(X,Y) <*> (Y:first:X;
                      ^Z(Y:rest:Z,member(X,Z))))

VX(no-repetitions(X) <*> (X:elist;
                          ^Y^Z(X:(first:Y,rest:Z),
                               ~member(Y,Z),no-repetitions(Z))))

VXVYVZ(append(X,Y,Z)  <*> (X:elist,Y:list,Z:list,Y=Z;
                           ^U^W^A(X:first:U,Z:first:U,
                                  X:rest:W,Z:rest:A,append(W,Y,A))))
```

Having added these descriptions, which we may call the MEMBER PRINCIPLE, the APPEND PRINCIPLE and the NO REPETITIONS PRINCIPLE, to the principles of grammar, we have completed the formulation of the set of descriptions, θ, of our grammar.

In the file cars-and-people243.mmp you may examine one of the feature structures which are licensed by our theory. The feature structure in the interpretation window included in cars-and-people243.mmp shows a feature structure representing a BMW with a male driver, who is also the owner of the car. There are two persons in the back seats, a man and a woman who like each other best.[14]

The reason for omitting the APPEND PRINCIPLE from the description card which otherwise contains the complete theory is that computing the denotation of append as stated above is computationally very expensive, and we want to restrict our example to a theory which even relatively slow computers can evaluate fairly quickly.[15] The principles for the relations member and no-repetitions are a lot less costly and are still suitable even for slow machines. You may note, however, that we still entered those triples into the append relation that are necessary for making the feature structure a model of the APPEND PRINCIPLE. Even if we omitted the APPEND PRINCIPLE, we would have to put at least the triple consisting of the node representing the FRONT-SEATS value, the node representing the BACK-SEATS value, and the node representing the TOTAL value into the append relation. It is necessary to do this in order to meet the requirements of the principle which states that in feature structures admitted by the grammar, the TOTAL list must be in the append relation with the FRONT-SEATS list and the BACK-SEATS list. Without the aforementioned triple in append our feature structure would not be admitted, not even by the theory without the APPEND PRINCIPLE.

[14]Depending on what kind of computer you are using, expect to have to wait some time for MoMo to answer a model checking query about the theory provided in the mmp file.

[15]This means that it is perfectly all right to work with the APPEND PRINCIPLE in MoMo if you wish. With feature structures of the size of our example it may take a few minutes on a fast machine with enough memory to return an answer to a model checking query.

All list nodes are in the `no-repetitions` relation, since none of the lists contain any element twice. The driver of the car occurs on two lists, the FRONT-SEATS list (in which he is the only element) and on the TOTAL list (in which he is the first element). The `member` relation comprises all pairs of nodes with the following properties: The first element of the pair is a FIRST value of an *nelist* node, and the second is an *nelist* node from which we can reach the respective FIRST value by following only REST arcs (including the possibility of following no REST arc at all), and one single FIRST arc.

It would be a good exercise for the reader to take the given feature structure and modify it in such a way that the modified feature structure represents a significantly different situation, but is still licensed by the grammar. You will quickly develop your own strategies for checking whether any modification you make results in a well-formed structure, and you will improve your intuitions about the meaning of grammars. An even better but more demanding task would be to build a new feature structure from scratch and to turn it into a well-formed feature structure licensed by the theory.

Constructing a well-formed feature structure which is licensed by a fairly small and uncomplicated looking theory, such as our grammar, is a surprisingly time-consuming and demanding task. Constructing a model of the APPEND PRINCIPLE alone, which is a standard relation in HPSG grammars, can easily become daunting if a feature structure comprises several lists with shared elements. This observation gives us a good impression of the complexity of the task faced by a grammar writer in the constraint-based framework of HPSG. After all, it is exactly the set of all relational abstract feature structures licensed by their grammar that linguists try to characterize. That means that linguists need to have a clear idea of the feature structures that they want to characterize, and they need to understand which feature structures are licensed by their grammar (and where the difference is and how it might be eliminated). Trying to construct a single feature structure licensed by our grammar is thus only a very small part of the real challenge!

GLOSSARY

2.5 Wrapping up: Summary and Open Issues

ABSTRACT

In this section we will review what we have achieved in our formalization of constraint-based grammars in general, and of the particular version of a constraint-based grammar which is presented in [Pollard and Sag, 1994]. We will comment on a few open issues which we mentioned at the end of the previous section.

In the course of constructing a rigorous formalization of the formalism that is implicitly underlying the HPSG framework of [Pollard and Sag, 1994], we saw three consecutive grammar formalisms that fall within the framework of constraint-based grammar. On the syntactic side we started with *initial grammars* and augmented them with variables and

a particular form of quantification to get *complex grammars*. We then ended up with *grammars* which are a relational extension of complex grammars.

All three types of constraint-based grammar had exactly the same structure: They consisted of a signature which provided the non-logical symbols for the constraint-languages generated by each type of grammar, and a set of constraints which we called *descriptions*. In linguistic applications of these constraint-based grammars, the elements of the set of descriptions are conventionally called the *principles of grammar*.

For the interpretation of constraint-based grammars, we initially considered three options that concerned assumptions about the interpretation of formalized scientific theories in general. According to the theory of Paul King, constraint-based grammars should denote collections of possible utterance tokens (directly observable and measurable empirical phenomena). According to the theory of Carl Pollard, constraint-based grammars should denote collections of mathematical idealizations of natural language expressions, where each mathematical entity in the denotation of a grammar is isomorphic to the hypothesized structure of a natural language expression. According to the theory of Carl Pollard and Ivan Sag in their HPSG book of 1994, constraint-based grammars should denote collections of feature structures that stand in an intuitive correspondence relationship to observable and measurable occurrences of natural language tokens in the real world. Each of these three views of the meaning of constraint-based grammars requires its own model theory, and the mathematical structures that are used to implement them look quite different. Each one of these views can be fully formalized as a theory of the meaning of grammars on top of our three notions of constraint-based grammars.

In our formalization we opted for the theory of Carl Pollard and Ivan Sag. We stressed that our choice was purely pragmatic in nature and should not at all be understood as the result of an evaluation of the three opinions about the nature of formal scientific theories. A justifiable and rational evaluation would be a rather difficult and very complex task and would have to draw its arguments from philosophy of science. The underlying reasons for our pragmatic choice, on the other hand, were threefold: A feature structure semantics of constraint-based grammars was the first to be formulated and is still the mainstream tradition in constraint-based grammars in general and in the HPSG community in particular. Second, feature structure models are fairly simple and can easily be employed in mathematical investigations of all model theories of constraint-based grammars of the HPSG-type. It is thus quite useful to know how feature structures work as a modeling domain. Finally, feature structures have been employed prolifically and successfully in computational treatments of HPSG, and are the structures closest to the way in which computational theories think of the semantics of constraint-based grammars. In a course in which we want to address the computational implementation of HPSG grammars, it only makes sense to start with the kind of declarative semantics that uses mechanisms and structures which are the most widespread on the computational side.

In spelling out a feature structure semantics for our three types of constraint-based grammars we saw the semantic counterparts to the two components of constraint-based grammars; signatures and sets of descriptions. The signature declares an ontology of objects by saying which structure the entities in the denotation of a grammar may have.

The collection of well-formed feature structures is determined by the syntactic material provided by signatures. In our signatures we are given a certain set of attribute symbols, sort symbols, and relation symbols, which we can use to characterize the empirical domain of language; and the appropriateness conditions determine what kinds of structures are possible by prescribing which sort of entity may have what kind of attribute arcs, and what the possible values at the other end of these attribute arcs are. Similarly, the arity of the relations in the denotation of the relation symbols is fixed by the signatures.

The task of the set of descriptions is then to narrow down the possible structures delineated by the signature: The constraint set discards those possible well-formed feature structures that are deemed not to correspond to any well-formed linguistic entity in the empirical domain. The effect of the constraint set of zooming in on the set of well-formed structures can be seen as one of the special characteristic properties of *constraint-based* grammars, and is one of its distinguishing properties compared to phrase-structure rule based grammars. Grammars that are built with phrase structure rules *generate* phrase markers.

The difference between the two paradigms can best be understood by considering the effect induced by adding a statement to the grammar. Adding a description to the set of descriptions of a constraint-based grammar will exclude expressions from the set of entities in the denotation of a grammar, in the case that the new description is not redundant in relation to already existing ones. Adding a new phrase structure rule to a phrase structure grammar, on the other hand, may only add expressions to the set of sentences that a phrase structure grammar generates, thus potentially increasing the empirical coverage of the grammar.

Linguists have two main tasks in the scenario of constraint-based grammars: Firstly, they have to design a signature which adequately represents the domain of possible linguistic expressions. No description will later be able to increase the coverage of the grammar if the signature fails to allow some possible linguistic structures. Secondly, they have to formulate the correct generalizations about their empirical observations in order to predict exactly those structures as grammatical structures which correspond to well-formed language tokens in the empirical domain.

HPSG 94 is a particularly pure incarnation of a constraint-based grammar. Whereas HPSG does not have any notion of phrase structure rules, related grammar frameworks such as Generalized Phrase Structure Grammar (GPSG) and Lexical Functional Grammar (LFG) combine a feature structure component with phrase structure rules. The resulting grammars comprise a layer of structure-generating phrase structure rules which produce phrase markers or phrase structure trees, such as those familiar from more traditional linguistic theories. In addition these trees are then decorated by, or related to, feature structures whose main task it is to enrich the information encoded in the linguistic structures. Moreover, constraint languages over feature structures and decorated trees have the same function as the constraint languages of HPSG: Linguists use them to determine which structures in the domain of well-formed decorated trees are licensed by their theory of grammar. Some introductory textbooks to HPSG-style grammatical theory, such as [Sag and Wasow, 1999], use the appeal to a combination of the more familiar notion

of phrase structure rules and phrase structure trees and a feature structure mechanism with an accompanying constraint language, in an attempt to achieve a gentler introduction for novices to the architecture of HPSG grammars and to the concept of constraint satisfaction. In the section on grammar implementation we will have further opportunity to investigate the relationship between pure, theoretical HPSG grammars and mixtures of HPSG grammars with concepts of phrase structure grammars, although for an entirely different reason.

Our mathematical perspective on constraint-based HPSG grammars provided a number of important insights into some of the central terminology which is frequently used in technical talk about HPSG grammars. For example *attribute inheritance* is nothing but a condition on the appropriateness functions of signatures. If we declare an attribute appropriate to a sort in the partial order of sorts, we are forced to declare it appropriate to all subsorts of that sort. *Multiple inheritance* is nothing but a consequence of this definition of appropriateness. In informal discussions of the topic, *multiple inheritance* refers to "inheritance of information" from different supersorts σ_1 and σ_2 of some sort σ_0, where σ_1 and σ_2 do not stand in a mutual subsort relationship. Thus, according to our definitions, σ_0 will "inherit" all appropriateness conditions of both σ_1 and σ_2. Moreover, if there is an implicational description δ in the grammar whose antecedent is a description of feature structures in the denotation of sort σ_1 or σ_2, the possible shape of feature structures of sort σ_0 will be directly restricted by it in the sense that they have to satisfy the consequent of δ. In this sense it is sometimes loosely said that σ_0 inherits the restrictions on σ_1 and σ_2.

We learned a particularly important lesson about the syntax of our constraint-languages: There are usually several alternative and perfectly reasonable syntactic ways of defining the same class of formal languages. With being "the same" we mean that they can characterize exactly the same classes of objects, and each formulation of a theory in one syntax can be transformed in a formulation of the same theory in the other syntax. Thus we should not be deceived by different appearances: It does not matter at all whether a grammar is written in a hardly readable mathematical notation or in an implementation language such as those of MoMo or TRALE, or in the linguistic, reader-friendly AVM notation (although it might make the grammar more or less fun to read). What does matter a great deal is the underlying meaning of the constraint language, and the theory of what constitutes a model of a given grammar. These are the essential questions which we have to ask when we see a constraint-based grammar, and the answers to these questions dictate how informative and useful a grammar can be for us.

GLOSSARY

2.5.1 The Lexicon

ABSTRACT

This section will explain briefly how a lexicon can be expressed in HPSG.

We saw that the descriptions in an HPSG grammar are implicational descriptions. This form is practically dictated by the way the meaning of grammars is defined on the basis of a relational abstract feature structure admission function: Every (abstract) node of every relational abstract feature structure denoted by the grammar must be described by each element contained in the theory θ of the grammar. For non-atomic nodes, i.e., for objects with outgoing arcs which lead to nodes of some other sort, the recursive nature of feature structure admission dictates a complex, non-conjunctive structure of the descriptions in θ. If an element of θ were simply a conjunctive description (of objects of a non-atomic sort), the set of feature structures admitted by the grammar would immediately be empty. For instance, remember that we cannot assume that the ID Schemata are descriptions in θ. If the HEAD FILLER SCHEMA were a principle of grammar in Pollard and Sag's theory of English, the grammar could only license phrases, because only phrases satisfy the HEAD FILLER SCHEMA. However, phrases necessarily contain *synsem* nodes as components (according to the signature). Thus not even phrases can be licensed by the HEAD FILLER SCHEMA, because their *synsem* components do not satisfy it. For that reason, the HEAD FILLER SCHEMA is only a disjunct in the consequent of an implication (the ID-PRINCIPLE) whose purpose it is to restrict the possible shapes of headed phrases.

But what about the lexicon? How do we get words in our grammars? Pollard and Sag say surprisingly little about the lexicon, and it is not mentioned in the appendix of their book, although the text contains many sketches of so-called *lexical entries*. Lexical entries are descriptions of words, i.e., descriptions that are satisfied by feature structures of sort *word*. However, we cannot put lexical entries in the theory of a grammar, for the same reason we cannot put the HEAD FILLER SCHEMA there. Words contain *synsem* nodes, and *synsem* nodes do not satisfy descriptions of words. A lexical entry in our theory of grammar would thus entail that only the empty set of relational abstract feature structures is admitted by the grammar. This is not what we want.

The solution to this puzzle is rather simple. As with the ID SCHEMATA and the ID PRINCIPLE we have to make lexical entries part of an implicational principle, conventionally referred to as the WORD PRINCIPLE. A very simple version of a WORD PRINCIPLE can be pictured as in (24):

(24) *The logical structure of the* WORD PRINCIPLE

$$word \rightarrow (LE_1 \vee LE_2 \vee \ldots \vee LE_n)$$

The meta-variables LE_1 to LE_n stand for the actual lexical entries in the grammar, which are descriptions of the words that the grammar predicts. Given the WORD PRINCIPLE, every feature structure of sort *word* admitted by the grammar must be described by at

least one lexical entry. The set of descriptions LE_1 to LE_n is the lexicon. Note that in this formulation the lexicon is necessarily finite, since our formal languages do not allow infinitary disjunctions. Nevertheless, depending on the formulation of the lexical entries and on the structure of words, there might still be an infinite number of relational abstract feature structures of sort *word* that are described by the lexical entries.

Finally there are techniques of integrating mechanisms for so-called *lexical rules* with the word principle. According to one possible and prominent interpretation, lexical rules say that if a word of a given shape is in the denotation of the grammar, then there is another word of a related but slightly different shape which is also in the denotation of the grammar. As soon as we extend the WORD PRINCIPLE by lexical rules of this sort, an infinite number of substantively different words can be characterized as being in the denotation of the grammar.

GLOSSARY

2.5.2 Parametric Sorts

ABSTRACT

> *So far we have used lists as lists of entities of any sort. Many HPSG grammars distinguish lists of different kinds of entities and use a special notation such as list(synsem) or list(phrase) to denote respectively lists of synsem entities or lists of phrases. In this section we will explain how this notation can be read and interpreted.*

We have treated the symbols in sort hierarchies as atomic symbols. While this agrees with most of the sort symbols in HPSG grammars, there are some exceptions. Some sort names are not meant to be atomic. For lists and for sets it is common practice to appeal to so-called *parametric sorts*. The idea of parametric polymorphism comes from logic programming. Pollard and Sag explain their use of parametric sorts as follows:

> We employ limited parametric polymorphism for lists and sets as follows: where σ is a meta-variable over (nonparametric) sorts, we partition *list*(σ) into *elist* and *nelist*(σ) and declare *nelist*(σ)[FIRST σ, REST *list*(σ)]; likewise we partition *set*(σ) into *eset* and *neset*(σ). [Pollard and Sag, 1994, p. 396, fn. 2]

We will avoid discussing sets in our seminar—informally we can simply assume that sets can be encoded as special kinds of lists with some special conditions imposed on them.[16]

Pollard and Sag's specification of parametric lists can be pictured as follows, where σ is a meta-variable over nonparametric sorts. For variety we will not state this signature fragment in MoMo notation, but in a closely related notation for signatures which is often found in the literature.

[16] The formalism that we have introduced can handle finite sets, which seems to cover all the sets that are actually used in HPSG grammars.

(25) *list(σ)*

elist		
nelist(σ)	FIRST	*σ*
	REST	*list(σ)*

With the specification in (25), Pollard and Sag intend to state the sort hierarchy and the appropriateness function for all lists whose members are of exactly one nonparametric sort.[17] For example, (25) comprises the specification in the signature of their grammar of lists of *synsem* entities. (26) expresses that specification without a meta-variable:

(26) *list(synsem)*

elist		
nelist(synsem)	FIRST	*synsem*
	REST	*list(synsem)*

Given the fact that the set of nonparametric sorts in Pollard and Sag's grammar is finite, (25) induces a finite number of specifications of possible lists, one specific example being (26). [Pollard and Sag, 1994] uses parametric lists for specifying in the signature that lists which are appropriate for certain attributes may only contain entities of a given sort. For example, the entities on SUBCAT lists must be of sort *synsem* by virtue of the appropriateness specification for the sort *category*:

(27)

category	HEAD	*head*
	SUBCAT	*list(synsem)*
	MARKING	*marking*

Thus the purpose of the limited use of parametric sorts in HPSG is clear. The question remains of how we can express parametric sorts in our formalism.

One way would be to extend the formalism by a new mathematical construct. However, there is a much simpler solution. We could take the formal tools that we already have, i.e. relations and principles of grammar, and reformulate what parametric sorts are supposed to achieve in these terms.

Consider the specification of SUBCAT lists as lists of *synsem* objects as given in (27). We can replace that appropriateness declaration by substituting our familiar sort *list* for *list(synsem)*, and using our standard appropriateness declaration for lists: The sorts *nelist* and *elist* are (the only) immediate subsorts of *list* and are species. The most general sort of the sort hierarchy, *object*, is appropriate to FIRST at *nelist*, and *list* is appropriate to REST at *nelist*. *elist* is an atomic sort.

In the second step, we need to restrict the permitted values for SUBCAT lists to *synsem* objects.

For that purpose, we introduce a relation symbol (with arity one) `list-of-synsems` to our grammar and define its meaning in the following principle:

[17] Note that this does not imply that the members of a list must be of the same species, because it is not required that *σ* be a maximally specific sort. In fact, since *σ* may be the sort *object*, the members of some lists can, in principle, be of any species.

```
VX (list-of-synsems(X) <*> (X:elist;
                            ^Y(X:(first:synsem,
                                  rest:Y),
                               list-of-synsems(Y))))
```

As you can easily check, all and only the *nelist* nodes that represent lists of *synsem* objects (and no other objects) are in the `list-of-synsems` relation in feature structures modelling grammars which contain our new principle.

But now we can exploit the meaning that we have given to the relation symbol `list-of-synsems` in models of our grammar in order to say that SUBCAT lists only contain *synsem* objects:

```
category *> ^X (subcat:X, list-of-synsems(X))
```

In words: For each node of sort *category* in our grammar, the value of its outgoing SUBCAT arc is in the `list-of-synsems` relation. It is a representation of a list that only contains *synsem* objects.

We conclude that in order to formalize the effects of parametric sort specifications, we do not have to make our HPSG formalism more complicated. We can take parametric sort specifications of the form *list(σ)*, exemplified in (27), the appropriateness specifications for the sort *category*, with *list(synsem)*, simply as a shorthand notation for: (1), the assumption that the grammar contains a generic list signature which licenses lists of arbitrary objects by making the top sort the appropriate list value; (2), a suitable relation symbol (`list-of-`σ) of arity one; (3), a principle which appropriately fixes the meaning of the new relation symbol in models of the grammar: it denotes lists of σ objects; and (4), a second principle of grammar which fixes the list values of the relevant attributes in the desired way, just as we saw it for SUBCAT lists.

GLOSSARY

2.5.3 Notation for Lists

ABSTRACT

To make expressions which describe lists of our formal languages simpler, we will introduce a convenient and widely used notation.

One final issue that we should at least briefly touch on concerns the notation for lists. We have seen two fragments of signatures that are typically used for the specification of lists in HPSG: One comprising the supersort *list* with speciate subsorts *elist* and *nelist* and attributes HEAD and TAIL appropriate for *nelist* where *list* is appropriate to TAIL, whereas the value of HEAD may vary according to what one might want to have on the lists. In an alternative formulation, attributes FIRST and REST were used instead of HEAD and TAIL. However, it is obvious that the particular names of the symbols are completely irrelevant as long as the geometry of the partial order and the relevant relationships in

the appropriateness conditions are preserved. Henceforth we call a signature with the appropriate feature geometry a *list signature*.

Using the symbols provided by a list signature, we can describe feature structures representing lists in exactly the same way in which we describe any other feature structure with any other non-logical symbols of the signature. However, a more perspicuous notation would be very welcome and is quite often used. Instead of describing a singleton list containing one green parrot as `(head:(parrot,color:green),tail:elist)`, people prefer to write `[(parrot,color:green)]`, which is much more readable. The advantages of this notation become even more striking when the lists get longer. To appreciate this, just compare

```
(head:(parrot,color:green),
 tail:(head:(canary,color:yellow),
       tail:elist))
```

with an expression as transparent as

`[(parrot,color:green),(canary,color:yellow)]`.

What can we do about this? Do we have to extend our syntax? In one sense yes, in another no. The relationship between the two notations is so perspicuous that we do not really have to bother about integrating an additional kind of notation into the syntax of our formalism. In fact this would make the definition of our description languages unduly complicated. This kind of situation is usually handled by introducing a syntactic convention on top of the formalism itself.[18] We simply state our signature of lists as we have always done and explain how we want to use a few additional notational devices to make our lives easier. For example we might include the HEAD-and-TAIL variant of list signatures in our signature, and then we would say that, by convention, we may sometimes write `[]` for `elist` and enumerate the elements of lists using square brackets: `[`δ_1`, ...,` δ_n`]` stands for the expression in (28):

(28) $\texttt{head:}\delta_1$,
$\ldots$,
$\texttt{tail}_1\texttt{:}\ldots\texttt{:tail}_{n-1}\texttt{:head:}\delta_n$,
$\texttt{tail}_1\texttt{:}\ldots\texttt{:tail}_n\texttt{:elist}$

A second abbreviatory notation which is often used together with the square bracket notation for lists is the vertical slash, |. The vertical slash separates a description of the first n elements on a list from a description of the tail of the list after the nth element. The following sketch pictures the use of this notation schematically:

[18] Sometimes additional alternative notations which do not change anything substantial in a formal language are called *syntactic sugar*.

(29) $\texttt{head:}\delta_1$,
$\ldots$,
$\texttt{tail}_1\texttt{:}\ldots\texttt{:tail}_{n-1}\texttt{:head:}\delta_n$,
$\texttt{tail}_1\texttt{:}\ldots\texttt{:tail}_n\texttt{:}\mathcal{BOX}$
may be written as
$[\delta_1, \ldots, \delta_n|\ \mathcal{BOX}]$

In (29), $\mathcal{BOX}$ stands for an arbitrary box, which will normally be a box that describes lists. This notation is particularly useful when we want to refer to the tail of a list with some variable, as in the following example:

(30) $[(\texttt{cat}), (\texttt{dog})|\ \boxed{1}]$

The expression in (30) is satisfied by feature structures representing a list whose first element is a cat and whose second element is a dog. We use the variable $\boxed{1}$ to refer to the tail of the list. This variable may be used again elsewhere in a larger description, of which our description may be a part. This would allow us to say that the tail of the list stands in a relation with other lists in feature structures which satisfy or model our description.

If we wanted to be really exact, we could describe these new notational conventions in an appropriate standard notation of mathematics. However, as long as it is plain enough what we want to abbreviate, and as long as we are consistent with our use of notation, an explanation such as the one given here is sufficient.

Exercises

Exercise 30 *We will take a tiny fragment of Pollard and Sag's signature specification, but change it slightly: We assume that the value of* SLASH *is a list of local objects rather than a set of local objects:*

```
type_hierarchy
object
  local
  nonlocal1 slash:list(local)
  list(sigma)
    nelist first:sigma rest:list(sigma)
    elist
.
```

Interpret the parametric list in this signature specification the way we did in Section 2.5.2, and create a MoMo file `parametric-sorts.mmp` *containing an appropriate standard signature and the relevant principles which ensure that, in models of the grammar, the* SLASH *lists may only contain lists of local objects.*

Exercise 31 *Take our familiar signature of lists, birds, and pets, birds-and-pets-exs253.mmp.*

In the Options *menu of MoMo you find a menu item* Top sort for lists. *Enter the top sort of lists in our signature. For your work with that signature, MoMo now gives you the option of using interchangeably either the usual syntactic notation for lists or the notation with square brackets,* `[, ]`.

Draw a list feature structure containing three animals. Then write two descriptions on separate description cards such that the feature structure satisfies them. Both descriptions should at least describe the color of each animal. The first description should use our normal syntax for the description of lists, the second one the new syntactic sugar with square brackets.

GLOSSARY

Chapter 3

Grammar Implementation

Abstract

In this section of the course, we will learn how to implement HPSG grammars in the TRALE system.

We know what an HPSG grammar is and what it means: An HPSG grammar is a signature (declaring a sort hierarchy, a set of attributes, appropriateness conditions for attributes, and a set of relation symbols with their arity) together with a set of descriptions. It can be interpreted in a domain of relational abstract feature structures. We say that a grammar denotes the set of relational abstract feature structures which are admitted by it.

In the present course module we will investigate how HPSG grammars of the kind that we have just briefly characterized can be implemented on computers. Our instance of an implementation platform for HPSG grammars will be the TRALE system. In Section 1.1, *Historical Overview*, we already placed the TRALE grammar development environment (GDE) in the context of the evolution of HPSG formalisms and implementation platforms. Recall that TRALE is a platform in the tradition of the HPSG formalism of Pollard and Sag 1994, as opposed to systems in the unification grammar paradigm of HPSG which Pollard and Sag presented in their book of 1987. Standing in the tradition of constraint-based HPSG, TRALE is a GDE closely related to the formal architecture of HPSG which is stepwise introduced and formalized as a constraint-based grammar formalism in the course section on *Grammar Formalisms*.

In this chapter we will see that the issues surrounding grammar implementation are much more complex than one might think at first, considering the fact that we have a complete mathematical account of constraint-based HPSG. With a rigorously defined formalism in hand one might quite mistakenly assume that one can take any HPSG grammar specified in a syntax for signatures and descriptions such as the one that we introduced together with MoMo, feed it into a computational system and perform the intended computations with that grammar. Unfortunately the situation is not that simple.

First of all we have to clarify what we mean by the *intended computations* which we vaguely hinted at in the previous paragraph. What is it that we want to compute on the basis of a grammar? What kind of questions do we want to ask the system, or, more

technically speaking, what are our *queries* to the system? After formulating precisely what it is that we want to do, we will then have to consider the computational properties of the computations that we want to perform based on our grammars and our queries. With regard to this question we will quickly discover that the computational costs of working with our theoretical, declarative grammar specifications of HPSG are very high. Even in cases where it might be possible in principle to perform successful computations on the basis of these grammar specifications, the computations can easily be very inefficient.

In order to achieve satisfactory results regarding computational efficiency, the specification of grammars in systems like TRALE differs to some extent from the declarative specifications that we are used to from theoretical linguistics. Some of the descriptive means of the full HPSG formalism of Pollard and Sag 1994 are not directly available in the computational environment, because they would generally be too costly. On the other hand, the system offers alternative constructs that a computational linguist can use in order to increase the efficiency of the intended computations. As a result, not only do we have to understand theoretical HPSG grammars and their meaning very well, we also have to know how our theoretical grammars can be expressed in a different way, taking advantage of the additional computationally motivated devices of TRALE. In a sense we have to understand how we can reformulate a grammar expressed in one formal language in a different—but closely related—formal language. If we are careful and understand what we are doing, we can be certain that our computational grammar means the same thing as the original, purely linguistically motivated specification.

The tensions between the properties of a very expressive description language which are desirable for linguistic purposes of constraint-based grammar formalisms and issues of computational efficiency will be a leading theme of our discussion of grammar implementation, and it will largely determine the structure of the first half of the present course section.

Besides getting acquainted with the user interface of TRALE and learning how to interact with the system, we will initially address the question of why we want to implement HPSG grammars in the first place and what we want to do with a computational system for our grammars. This will inevitably lead us to the computational problems that come with grammar implementation.

With some basic grasp of the nature of the general problem, we will then see what TRALE offers to make our grammars computationally efficient. The first TRALE grammars that we will work with look quite different from HPSG grammars, because we will first learn what the differences are between TRALE and our constraint-based grammar formalism. This directs our initial focus towards those parts of the grammar specification language of TRALE which are different from the description languages that we have worked with so far. We will explain what the new constructs of TRALE mean in terms of our HPSG formalism, and we will learn how to write TRALE grammars that use these additional devices but nevertheless look more and more like the theoretical HPSG grammars that we are used to. With sufficient understanding of how to do this we will then develop a fragment of English inspired by Pollard and Sag's grammar of English. We will start with a very simple, small fragment and extend it slowly in order to cover more and more

linguistic phenomena. The linguistic issues and phenomena covered by our grammar will include lexical entries, phrase structure, syntactic selection, agreement, case government, thematic roles, raising constructions, word order, and unbounded dependency constructions. At each step of extending our specification, we will take the initial, constraint-based specification and reformulate it in the language of TRALE, thus testing the consistency and linguistic predictions of our specification in the computational environment.

Exercises

Exercise 32 *Download the TRALE User's Manual, available in pdf and ps format from the TRALE documentation pages.*[1] *As you can see on these pages, you may also use an online version of the manual. Browse the manual to get an overview of the system.*

Exercise 33 *Download the TRALE grammar consisting of the files* `signature`[2] *and* `theory.pl`[3]*, which you also find in Section 6.6,* Core Fragment*, of Chapter 6,* Grammars*, and put them in a new directory on your computer account. Start TRALE with the graphical interface GRiSU from that directory, and compile the grammar. If you do not know how to do that, look it up in the TRALE manual.*

Take a look at the specifications in the file `theory.pl`*. You do not have to understand exactly how this grammar works to be able to easily guess which words are available in this grammar and which sentences are licensed by it. Take a few words to form and parse at least five different sentences (*it rains *might be a good start). Save two of your parse results and upload them to your personal forum in ILIAS, giving them intuitive names which make them easy to recognize as your solutions to this exercise. Use this opportunity to play with GRiSU and get somewhat familiar with its functions.*

GLOSSARY

3.1 Computing with HPSG Grammars

Abstract

We will explain our motivation for using HPSG grammars in a computational environment and we will investigate the computational properties of our grammars with respect to the tasks which we would like to perform with them. This will give rise to a few observations about the nature of implementation platforms for HPSG-style grammars.

Why would one want to compute with HPSG grammars—or with grammars in general, for that matter—, and what would one want to compute with them?

[1] `http://www.ale.cs.toronto.edu/docs/`
[2] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Core-Fragment/signature`
[3] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Core-Fragment/theory.pl`

At the very beginning, and on a very abstract level of our discussion of this question, we may distinguish two distinct applications which one might have in mind for a computational system for grammars. We might want to use this kind of computational system for commercial purposes or we might want to use it to enhance our scientific understanding of the nature of language, of particular natural languages, and of language processing. Ultimately we will be be dealing with scientific aspects of grammar implementation. However, let us start with examples of products which might gain in quality by the implementation of these kinds of grammars. We should bear in mind, though, that for our purposes we will be taking a very simplistic view and ignoring many aspects and possible alternative solutions to the development of products, as long as they are practically feasible and can perform the envisioned tasks satisfactorily.

For commercial purposes, it is of course central to investigate the kinds of application that one might envisage. A couple of well known examples are automatic email sorting systems, or systems in call centers which either immediately answer customers' questions or direct their questions to the appropriate employees who can answer them personally. Other typical applications are conceivable for car navigation systems where natural language is a very suitable input and output modality, since drivers typically do not have a free hand for typing questions or selecting menu items, and they have to focus their attention on the road rather than on a screen on which relevant information might be displayed. Future Internet search engines on the world wide web might be able to employ some natural language processing component which is able to parse and analyze the contents of documents and to respond to queries that are asked in natural language.

All of these applications could at least to some degree use a grammar of (an appropriate fragment of) the relevant natural language(s), which would solve some of the problems involved. Let us assume that the products we are thinking about comprise an HPSG grammar. What would that grammar be used for? If it is a good grammar, the grammar would describe all and only grammatical expressions of the language:[4] In its denotation we would find relational abstract feature structures representing the phonological (or orthographic) form of expressions of the language together with their syntactic structures and structures which represent their meaning, probably in a formal language of a higher-order logic (such as Richard Montague's *Intensional Logic* or Daniel Gallin's Two-sorted Type Theory [Gallin, 1975]) with a model-theoretic interpretation. On this level the grammar would establish a connection between phonology (orthography) and meaning, which would apparently be necessary for actually carrying out the tasks implied by the applications outlined above. To interpret the input strings of natural language for these devices, it is necessary that their language processing component maps the input strings to some form of meaning representation. Conversely, to respond to any kind of request to the system in natural language, it is necessary that the system maps some kind of internal logical representation of what it needs to communicate to a phonological (or orthographic) representation in the target natural language. In either case, our HPSG grammar would be able to establish the connection between an external representation which is communicated and

[4] And, possibly, components thereof, depending on how the signature of the grammar is set up.

an internal, non-observable meaning representation.

But let us not be too hasty: First we must ask what exactly is the task here, and how can our grammar help to accomplish it? Let us assume that we have the phonological string belonging to an expression and we are interested in its meaning; how can the grammar help us to compute it? Obviously, although we have the grammar (the signature and a set of descriptions), we do not yet have either to hand or stored in the computer the (usually infinite!) set of relational abstract feature structures which the grammar admits. On the contrary, we know that it is a fairly difficult task to construct even a small feature structure admitted by a given grammar, even if the grammar is only moderately complex, far less complex than we may expect any grammar of a natural language to be.

One conceivable solution to the problem of constructing the necessary computational system for grammars would be to design a component for it which tries to automatically construct feature structures in the denotation of a grammar, and then to find one whose phonological representation satisfies the input description. If we were to find this kind of feature structure, we would also find as one of its components the desired meaning representation. However, this is not how systems such as TRALE would work. For one thing we would have to construct and somehow represent a usually infinite set of feature structures and find the relevant one(s) in the set (or somehow decide that they are not in the set). An approach that works with representations of the meaning of the grammar in the form of feature structure models seems to be unnecessarily complicated, and in the face of the immense complexity involved in this task we prefer to stick to minimal solutions to accomplish our task. What if we were to transform the problem into one which we could solve by only manipulating syntactic objects, i.e. entirely based on descriptions, without ever having to deal directly with the meaning of descriptions? With this approach it is interesting to note that our grammars are usually finite, they involve finite descriptions, and the output which we are ultimately interested in is also finite.

Following this idea, our original problem can be reformulated as follows: We have two entities, a grammar and an input description of a phonological string (our *query*). What we want to know is whether there is anything admitted by the grammar such that its phonological string satisfies our query, and we want to know what this looks like. For the latter purpose we will be satisfied with obtaining a set of descriptions of the relevant entities. From these descriptions we can then extract the part which concerns the meaning representation of the expression and pass it on to a component of our product which is able to process meaning representations.

What we have just described is the *parsing problem* of constraint-based grammars: We stated what the queries are which we want to process (descriptions of phonological strings), and we stated that we want to obtain sets of fairly precise descriptions of those objects in the denotation of the given grammar whose phonological strings satisfy the query. The output descriptions may thus be read as analyses of the structure of the queried objects.

Interestingly enough the problem of *language generation*, which is also part of the products mentioned above, has the same formal shape in our constraint-based setting. The only difference consists in the kind of query which we start with. In the case of parsing, we query (descriptions of) phonological strings. In the case of generation, we

query (descriptions of) meaning representations, and we want to solve a parallel question: Is there anything admitted by the grammar whose meaning representation satisfies the query, and what is a precise enough description of its overall structure, especially of its phonological string?

The following two examples presuppose the existence of an appropriate grammar and sketch two queries to a system about that grammar. The first one is a query to parse the sentence *It is snowing in Tübingen*, and the second one is a query to generate a sentence with a meaning representation saying that Jon runs. For reasons of simplicity, we have used an orthographic representation for the phonology and made up an intuitively sensible description of a CONTENT value in the two examples.

(31) a. $\begin{bmatrix} \textit{sign} \\ \textsc{phonology} \ \langle \textit{it, is, snowing, in, tübingen} \rangle \end{bmatrix}$

b. $\begin{bmatrix} \textit{sign} \\ \textsc{synsem local content} \begin{bmatrix} \textsc{relation} & \textit{run} \\ \textsc{argument} & \textit{jon} \end{bmatrix} \end{bmatrix}$

At this point we have made the computational problem precise enough in order to investigate its mathematical properties. Is it actually possible to compute what we want to compute on the basis of the HPSG formalism? The answer is straightforward but not very comforting: Bjørn Aldag proved [Aldag, 1997] that it is impossible to design an algorithm so that for an arbitrary query and an arbitrary grammar, the algorithm determines whether there is any feature structure admitted by the grammar which satisfies the query.

What does this result mean? One of the immediate and most prominent consequences is that we cannot just write any grammar in the HPSG formalism and expect that there is a computational system that can help us to effectively parse and generate sentences relative to the grammar we wrote. Closer investigation reveals that it is even difficult to determine an interesting subset of grammars for which this could always be done. For the time being we have to acknowledge that HPSG uses a highly expressive description language in a very powerful formalism which is very flexible and amenable to many different linguistic analyses of empirical phenomena, but, for the very same reason, difficult to handle computationally.

Through considerations about the architecture of possible commercial products integrating HPSG grammars we have come a long way towards a genuinely scientific insight: There cannot be a general solution for computing with arbitrary HPSG grammars. However, there can very well be and there are task-specific solutions which are tailor made to suit the types of grammars which linguists actually write. The idea is to produce a general mechanism for computing with these grammars, with task and structure specific computational constructs which make processing with them possible and at the same time much more efficient than general algorithms for computing with our description languages could ever be.

Now when we start to work with TRALE, we will immediately see the effects of these conditions. TRALE is a system which tries to combine the potential for efficient and fast computing with large-scale grammars with an architecture which is as close as currently

possible to theoretical HPSG grammars. This should also make TRALE a suitable system for scientific purposes. The scientific benefit of this kind of system consists in providing a good testing ground for the complete formalization of linguistic theories. In the TRALE environment we can test large grammars for their internal logical and technical consistency, and we can detect consequences and predictions of grammars more easily than with paper and pencil, where mistakes can occasionally go undetected for some time. Moreover, having a system which is already close to HPSG stimulates the further development towards closing the gap between linguistic theories and grammar implementation.

In this context one might argue that much of the scientific value of an implementation platform depends on how close it gets to the HPSG formalism while still being useful in a practical sense. For example, it would not be useful at all to be able to specify arbitrary HPSG grammars directly, if the processing of any non-trivial query would then never terminate. Obviously it would not really matter how "purely HPSG-like" a system of this nature would be: It would simply not be of much use for anything.

On the other hand, however, we postulate that one of the major criteria for a grammar development environment to be productive is that it supports an encoding of linguistic theories which is transparent with respect to its relationship to a pure, declarative specification in the HPSG formalism. The inevitable recoding of such a specification which occurs during the implementation of HPSG grammars in TRALE should be done in a systematic way, and it should be precisely documented. This is the only way to ensure that a computational implementation really reflects the original grammar, and that the behavior of the system reliably tells us about the properties of the grammar.

In the next sections of the course we will explore very small grammars in TRALE. They introduce features of the implementation platform which we do not know from HPSG grammars, since they are not part of their formalism. We will focus on getting familiar with these features and on explaining how they are systematically related to pure HPSG grammars. In the end we want to be able to tell precisely how to translate these new mechanisms into grammatical principles of an HPSG grammar specification, and how we can make the processing of certain HPSG principles more efficient by employing special mechanisms of TRALE for their encoding.

GLOSSARY

3.1.1 A Minute Grammar

Abstract

Our first small TRALE grammar will introduce phrase structure rules and a number of syntactic conventions of implemented grammars.

We can regard an implemented grammar in TRALE as an HPSG grammar with a number of additional special devices. This means that TRALE grammars consist of the same two components which we are already familiar with, signatures and theories.

TRALE signatures are a subset of HPSG signatures: Every TRALE signature is also an HPSG signature, but because of the restrictions on possible TRALE signatures, some of

our more general HPSG signatures are not permitted in TRALE. However, the differences that exist due to the restrictions which TRALE imposes on the partial order of sorts, on appropriateness specifications, and on relations (which do not belong to TRALE signatures at all) are initially of very little practical significance for us.

TRALE theories, as opposed to the signatures, are the starting point in which we will first encounter new devices, which are immediately visible in the grammar specification files. Thus we may simply investigate increasingly complex grammar specifications in TRALE and explain the meaning and purpose of new notations and of new mechanisms as they occur.

Before we jump into our first TRALE grammar, we want to classify very briefly the new devices that we are about to see. This will give us a little bit of guidance for estimating their relative importance when we first encounter them.

First of all a theory file in TRALE contains a number of technical specifications which have to do with importing necessary files, declaring file names, loading and declaring operators, and assigning names to syntactic entities. For example we have to say what the name of the signature file is, and where it is located in the file system. Obviously such specifications are necessary for a computer program to work, but they are fairly uninteresting for grammar implementation itself and they have nothing to do with the question of how HPSG grammars are related to their implementation in TRALE. We have to get acquainted with the necessary specifications, but need not discuss them.

Secondly TRALE provides a number of new notations which make it easier and more convenient to write grammars. These notations are essentially abbreviations which are in principle optional, and can always be explained in terms of complete, traditional HPSG descriptions. A very simple example of this is the list notation with square brackets, which can also be used in the MoMo syntax of descriptions as syntactic sugar (see Section 2.5.3, *Notation for Lists*). The macros and macro hierarchies of TRALE are a similar but much more sophisticated device of notational convenience. The specific notation for lexical entries which we will use in our first grammar is a notational variant of the WORD PRINCIPLE, although in this case an alternative notation to standard HPSG conventions is fitting, not only for the convenience of the grammar writer but also for the demands of the computational system.[5]

The most significant and most problematic differences reside in additional, divergent mathematical concepts in TRALE, concepts which do not belong to the HPSG formalism. Most prominent among these are phrase structure rules and relational attachments to descriptions. Since they do not belong to HPSG, we have to investigate which tasks these mechanisms perform in the grammar implementation and how exactly they are related to HPSG grammars. Clearly they play a crucial role in making our implemented grammars work in a computational environment. But to recognize our original HPSG grammars in their implementations, and to be sure that the TRALE grammars behave according to the original specifications, we have to identify precisely what the new constructs mean in terms of HPSG grammars.

[5] We will say more about this below.

With this in mind, let us now turn to our first example of a TRALE grammar (signature[6], theory.pl[7]). As mentioned above, the signatures of TRALE look exactly like the signatures of MoMo. Thus we are able to read them directly, as they are stated in the grammar files (Figure 3.1).

Notice that sorts for phonetic (or rather orthographic) strings are not declared in the signature: In the signature of an explicit grammar which uses the simplistic orthographic representation of phonology from Pollard and Sag we need sorts for all words of a grammar, which would have to be subsorts of *phonstring*. The fact that this declaration is not necessary in TRALE is an initial simplification provided by the implementation platform. This is an immediate consequence of treating the phonology of signs as a distinguished attribute value. This special treatment is ultimately caused by considerations of computational efficiency: The phonology of signs is an essential data structure of the parsing system.

In the specification of the appropriateness conditions for SUBCAT lists, we may observe that lists are not parametric (as in Pollard and Sag's specification *list(synsem)*). Ignoring this issue altogether in our little toy grammar, however, will not affect the correctness of the intended denotation of this particular grammar. We will see later that we can simulate parametric lists in TRALE in the same way as we can in our HPSG grammar specifications.

Finally there is an attribute in the signature which serves only the technical purposes of our current version of TRALE, the DTRS attribute. This is simply a given and necessary attribute in the current version of TRALE. It may not be used in grammatical principles, since this would interfere with what DTRS does system-internally.

Let us now look at our first implemented theory (theory.pl) of a grammar (Figure 3.2).

The first seven lines contain technical declarations about the treatment of reserved symbols (for declaring phrase structure rules (##) and lexical entries (∼∼>)) in the system and about loading a particular file, `tree_extensions`, for parsing. The file `tree_extensions` and details of its purpose do not concern us here. The file `signature` is specified as the file which contains the signature for the present theory file.

The technical declarations are followed by a richly decorated phrase structure rule (PS rule). The decorations consist in descriptions (of the kind we know) of the mother node and the two daughter nodes. The effect of a phrase structure rule corresponds to an ID-Schema within an ID-Principle, and a principle about constituent order of HPSG: The order in which the daughters are described in the PS rule determines the linear order of their phonologies in strings. Moreover, in the present formulation, the phrase structure rule incorporates the familiar SUBCATEGORIZATION PRINCIPLE, the HEAD FEATURE PRINCIPLE, and a simple SEMANTICS PRINCIPLE.

The format of the phrase structure rule is fairly transparent. It begins with a name declaration to the left of the reserved symbol, ##, followed by the phrase structure rule itself. The rule starts with a description of the mother node. This may contain variables which can also occur in the descriptions of the daughter nodes. Descriptions of the daugh-

[6] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik1/signature`
[7] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik1/theory.pl`

```
type_hierarchy
bot
   list
     ne_list hd:bot tl:list
     e_list
   sign phon:ne_list synsem:synsem
      phrase dtr1:sign dtr2:sign dtrs:list
      word
   synsem category:cat content:cont context:conx
   cat head:head subcat:list
   head case:case vform:vform
   vform
      fin
      bse
   case
      nom
      acc
   cont relation:rel_name arg1:index arg2:index index:index
   conx backgr:list
   index person:person number:number gender:gender
   person
      first
      second
      third
   number
      sing
      plur
   gender
      masc
      fem
   rel_name
      walk
      female
.
```

Figure 3.1: The signature of our first TRALE grammar

```
% Multifile declarations.
:- multifile '##'/2.
:- multifile '~~>'/2.

% load phonology and tree output
:- [trale_home(tree_extensions)].

% specify signature file
signature(signature).

% phrase structure rule
subject_head_rule ##
(phrase, synsem:(category:(head:H,
                           subcat:[]),
                 content:Content),
 dtr1:Subj, dtr2:Head)
   ===>
cat> (Subj, synsem:Synsem),
cat> (Head, synsem:(category:(head:H,
                                    subcat:[Synsem]),
                        content:Content)).

% lexical entries
she ~~> (synsem:(category: (head:case:nom,
                            subcat:e_list),
                 content: (index: (X, (person:third,
                                       number:sing,
                                       gender:fem))),
                 context: (backgr:[(relation:female,arg1:X)]))).

walks ~~> (synsem: (category: (head:vform:fin,
                               subcat:[(category:head:case:nom,
                                        content:index: (X,(person:third,
                                                           number:sing
                                                          ))))]),
                   content:(relation:walk,
                            arg1:X))).
```

Figure 3.2: The theory of our first TRALE grammar

ter nodes of the phrase structure rule follow the reserved symbol `cat>`. There may be as many daughters in a phrase structure rule as one likes. In the present example, the phrase structure rule has two daughters. The order in which the daughters are mentioned determines the linear order of their phonologies on the mother node, exactly as in standard phrase structure rules with atomic symbols: `S` → `NP VP` is probably the most famous example among linguists. The variables `Subj` and `Head` (which are given suggestive names) determine that the value of DTR1 is identical with the first daughter, and the value of DTR2 is identical with the second daughter in the phrase structure rule.

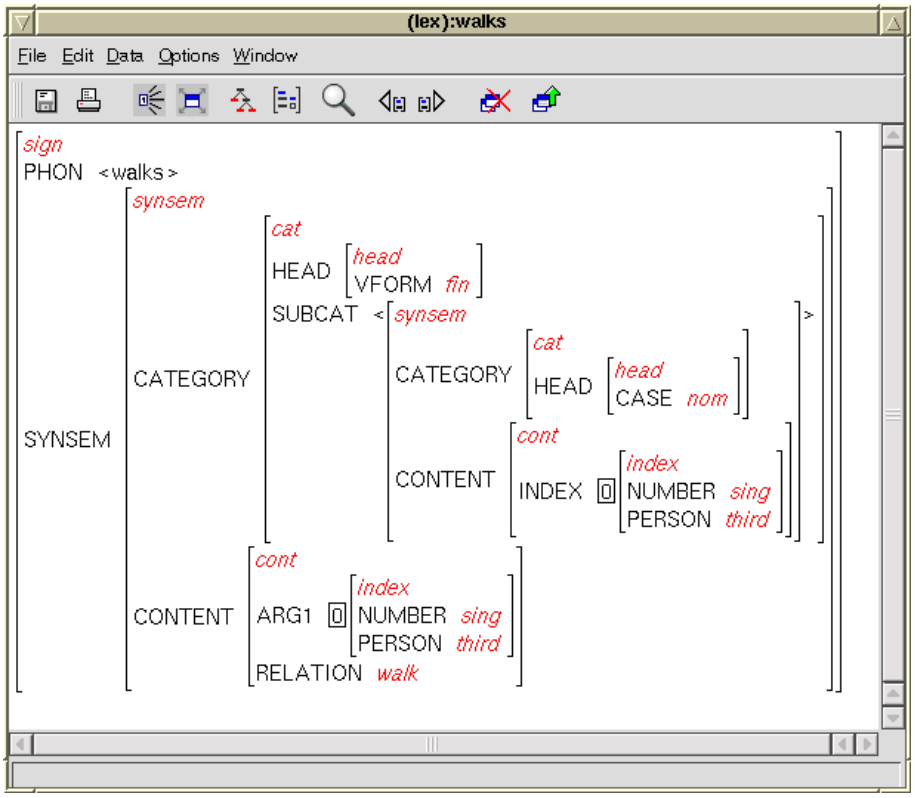
The effect of the HEAD FEATURE PRINCIPLE is achieved by using the variable `H`. The description of the SUBCAT values of the mother and the head daughter, and the sharing of the single element on the SUBCAT list of the second daughter and the SYNSEM value of the first daughter by virtue of the variable `Synsem`, implement the effect of the SUBCATEGORIZATION PRINCIPLE. A SEMANTICS PRINCIPLE is integrated in the phrase structure rule by identifying the CONTENT values of the head daughter and the mother by the variable `Content`. When we parse a sentence we will see that the `subject_head_rule` licenses a sentence which looks as if it were licensed by an HPSG grammar, although our grammar does not contain a single explicit implicational principle, integrating all but one of them in the annotation of the phrase structure rule.

The exception to the integration of all principles in the PS rule is the WORD PRINCIPLE, which is normally an implication with a disjunction of lexical entries in its consequent. In TRALE lexical entries are stated in a reserved syntax. Instead of being introduced as disjuncts in the consequent of a single WORD PRINCIPLE, they are stated in a syntax which separates them syntactically from each other. To the left of the reserved symbol `~~>` is the phonology of the word. To the right of it is the rest of its description. Initially this is just a practical convention: Its logical interpretation by the system is identical with the interpretation of a traditional WORD PRINCIPLE.

There are more reasons for the special notation for lexical entries: A WORD PRINCIPLE would quickly become very hard to read due to its size, and it would become extremely difficult to find typos in this huge principle. Secondly, there are limitations to the size of terms which Prolog can handle. Splitting up the WORD PRINCIPLE into disjoint syntactic entities helps to avoid this problem.

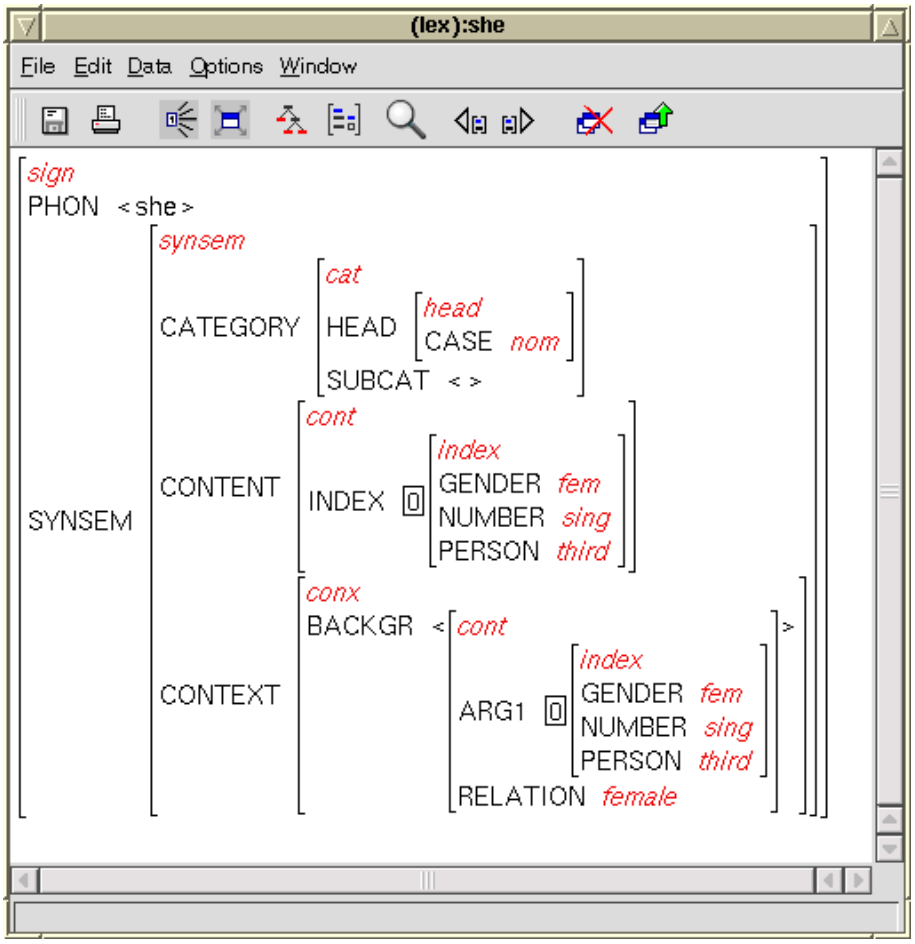
Finally, let us observe that TRALE provides the notation for lists with square brackets. This allows us to avoid the rather cumbersome and error-provoking notation which is based on an explicit use of the attributes HD and TL, as the list attributes are called in the present signature.

After starting TRALE with the option `-g` and compiling the grammar, we can investigate our first grammar using the graphical interface GRiSU. Typing `lex walks.<Return>` returns a description of the word *walks*. This description displays the information contained in the system about the shape of words with the phonology *walks*, which are in the denotation of the grammar.

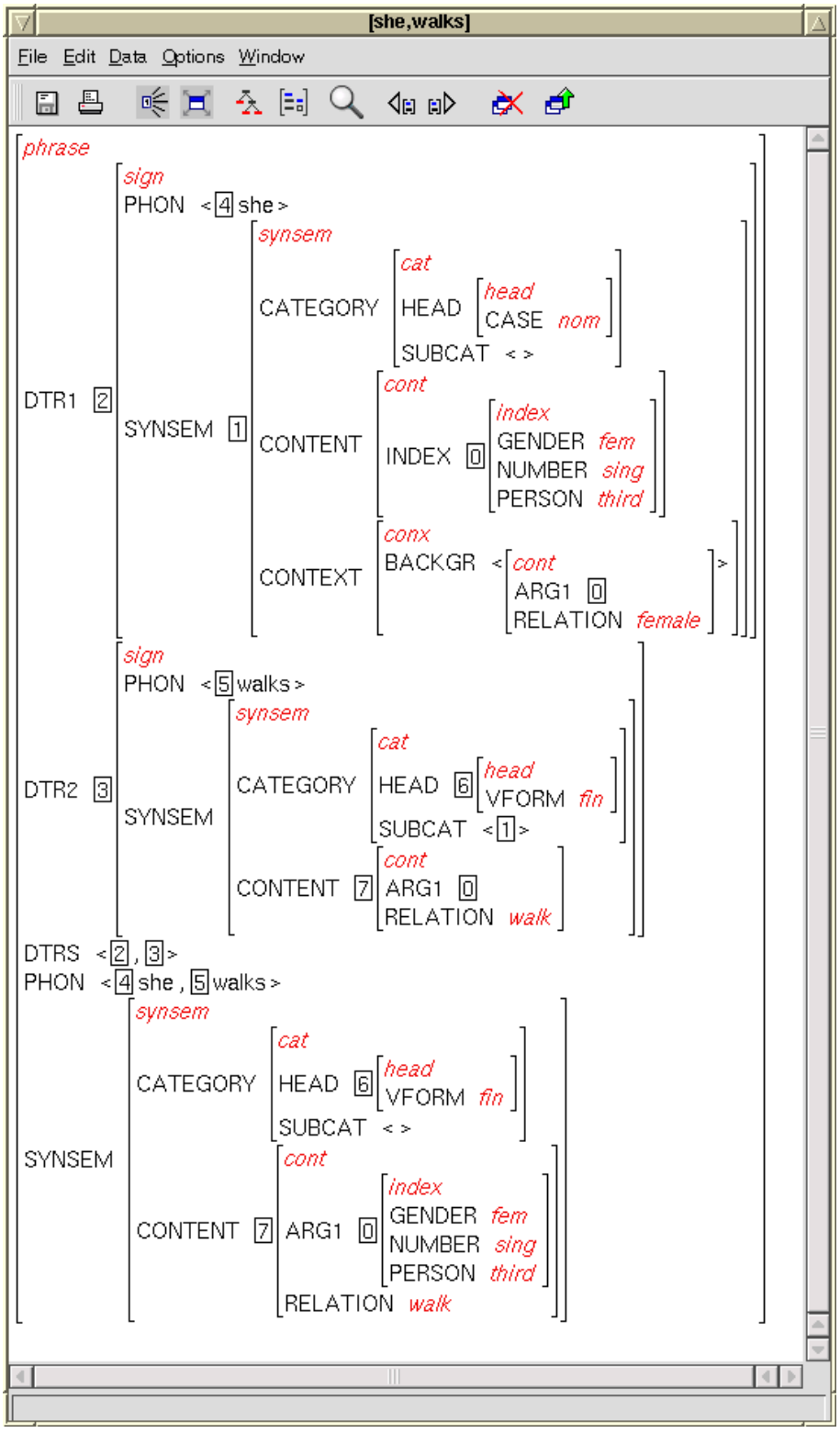


Obviously `lex` is a system predicate which takes the phonology of single lexical items as its argument. It is interpreted as a query for the elements of sort *word* with the given phonology in the denotation of the grammar.

Similarly we may examine the system internal description representation of the word *she* by typing `lex she.<Return>`:

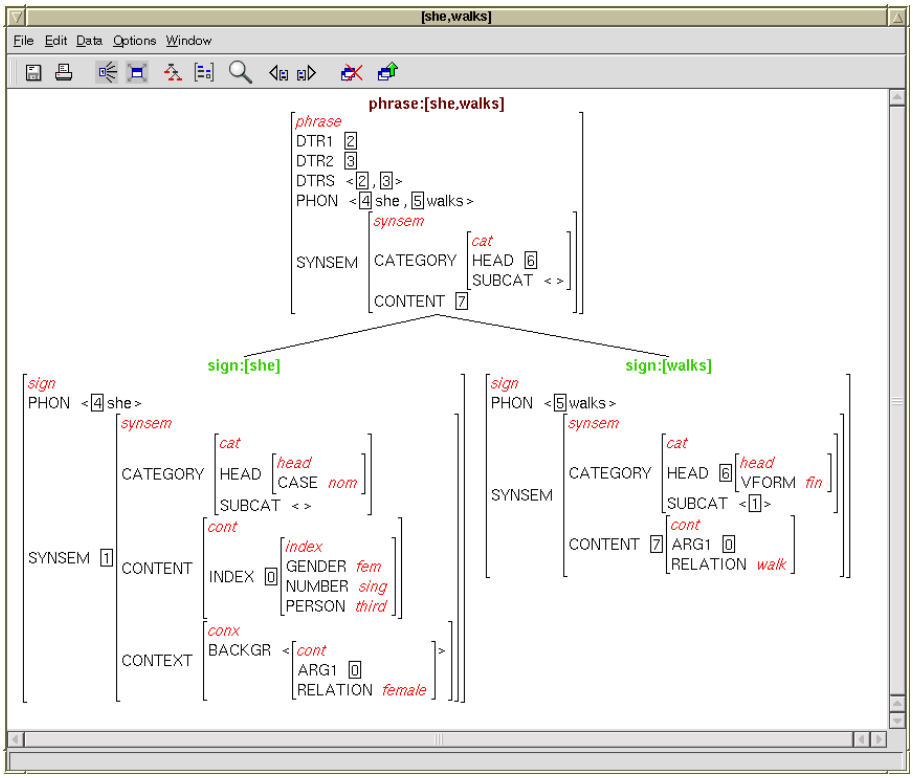


Our small grammar predicts exactly one sentence. The parsing predicate of TRALE is `rec`, which is followed by a list with the phonology of the expression that we want to test. The query `rec[she,walks].<Return>` is answered with a description that can be viewed in two major formats, either as a pure AVM description or in a phrase structure-like format. Here we provide both formats in turn, with the AVM format first.



Note that the tree representation format of GRiSU is nothing but a display function of the graphical interface of the grammar, which is supposed to help the user to examine

the grammar. There is no logical significance to the two different modes of graphical representation. In the tree representation two of the attributes, DTR1 and DTR2, are simply represented in a different mode, as edges of a tree. This has no theoretical significance.



In the graphical tree representation of the system's output shown above, the values of DTR1 and DTR2 are actually displayed redundantly. In the description of the mother node we find the two attributes followed by tags. We can expand these tags by clicking on them to reveal more complete descriptions of the two daughters. At the same time, descriptions of these two daughters are displayed under the tree branches which represent the two attributes DTR1 and DTR2 as edges in a tree. GRiSU offers the option of *hiding* arbitrary attributes to avoid such redundancies in the output (see the TRALE manual for how the user can specify this). This will be a very useful option when we investigate the parsing output of bigger sentences with huge descriptions.

In the next section we will extend our small grammar and learn more techniques of working with TRALE.

Exercises

Exercise 34 *Summarize the* parsing problem *of HPSG in your own words.*

Exercise 35 *Take a close look at the small grammar and ask about* **anything** *that remains unclear to you.*

GLOSSARY

3.1.2 The Second TRALE Grammar

ABSTRACT

The second TRALE grammar (signature[8], theory.pl[9]) will introduce a slightly modified signature and extend the syntactic combinatorics of the grammar.

The signature of our second small fragment of English differs slightly from the signature of the first fragment of the previous section.

In the first grammar we did not distinguish between different types of head values of signs. Nodes of sort *head* were the only possible HEAD values. Since CASE and VFORM were appropriate to *head*, the verb *walks* as well as the pronoun *she* had a verb form and case. Since a case value was not specified for *walks* and a verb form value was not specified for *she* in the grammar, and no principles of grammar imposed any restrictions on possible HEAD values, answers to lexical queries with the predicate `lex` did not mention the unspecified attributes and attribute values. This meant in effect that these values, being both possible but unspecified, were permitted in models of the grammar for each of the words. The verb *walks* occurred in models of the grammar with either CASE *nom* or CASE *acc* (but never without a CASE arc), and the noun *she* occurred with either VFORM *fin* or VFORM *bse* (but never without a VFORM arc).

The present grammar introduces subsorts of *head*, *noun* and *verb*. The attribute CASE is appropriate to *noun* and the attribute VFORM is appropriate to *verb*. Again, the output descriptions to a lexical query about *walks* will not mention case. However, this time this has a different significance: Now the word *walks* can no longer have case at all, because the signature says so. Similarly, *she* has no verb form due to the new signature.

A second important change in the signature concerns the possible values of the feature CONTENT. In the signature of the first grammar values of CONTENT were always of sort *cont*, since *cont* did not have any subsorts. Four attributes were appropriate to *cont*: RELATION, ARG1, ARG2, and INDEX. This is the kind of signature which Pollard and Sag use implicitly and in many informal examples in the first few chapters of the 1994 book, before they reconsider their semantic representations and suggest a revision. The motivation behind the revision is obvious. According to the first signature every sign has a CONTENT value with four outgoing arcs, one for a RELATION, two for the first and the second argument of the relation, and one for an index. If we look at what the first grammar does with the attributes we see that this is not what is actually intended. The word *walks* expresses the semantic relation *walk*, and the walk relation has exactly one argument; the subject of walking. There is no second argument and there is no index. Since no information is provided about a second argument or an index, all that the query `lex walks.` returns is a description which includes the relation and the first argument. This is not correct although it might at first appear so, since we know that, given the signature, the output description means that *walks* has an arbitrary second logical argument and an arbitrary index; in fact any second argument and any index permitted by the signature. The reader

[8] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik2/signature`
[9] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik2/theory.pl`

```
type_hierarchy
bot
   list
     ne_list hd:bot tl:list
     e_list
   sign phon:ne_list synsem:synsem
      phrase dtr1:sign dtr2:sign dtrs:list
      word
   synsem category:cat content:cont context:conx
   cat head:head subcat:list
   head
      noun case:case
      verb vform:vform
   vform
      fin
      bse
   case
      nom
      acc
   cont
      relations arg1:index
          unary_rel
             walk_rel
             female_rel
          love_rel arg2:index
      nom_obj index:index
   conx backgr:list
   index person:person number:number gender:gender
   person
      first
      second
      third
   number
      sing
      plur
   gender
      masc
      fem
.
```

Figure 3.3: The signature of our second TRALE grammar

might want to go back and check the description in the last section, or even better, use TRALE to reproduce the result.[10]

A similar observation holds for the pronoun *she* in the first grammar. The pronoun *she* has a semantic index, but it does not specify a relation nor any arguments. The description which comes up in answer to `lex she.` relative to the first grammar reflects this idea, since nothing is specified in the grammar about a relation and arguments. However, given the signature, the answer to our query implies that *she* does have two arguments and a RELATION value. *Any value* which is permitted by the signature is allowed. This is of course not what we want.

What is interesting about this example is that we cannot detect the problem by just looking at the output of lexical queries or at parse results. Because the "unwanted" meanings are not mentioned anywhere in the grammar except for the signature, the system does not make any inferences about the relevant parts of the structures in the denotation of the grammar and does not say anything about them in the descriptions it returns. The only way to spot the problem is to consider the specifications in the signature and the *meaning* of the descriptions returned by the system. Since our system works entirely on a syntactic level, meaning is constructed entirely by the user, and it is therefore the user's responsibility to be aware of the meaning of the grammar and to make sure that it is sensible. The important lesson here is that just because the output of a grammar looks right, it does not automatically follow that the grammar itself is flawless. A computational system such as TRALE will detect some problems in our grammars for us, but by no means all of them. Or to put it even more clearly, computing cannot ever replace thinking.

Our new signature fixes the problems with the CONTENT values by introducing subsorts of *cont* which have the right kinds of attributes appropriate to them. The new signature distinguishes *relations* from nominal objects (of sort *nom_obj*). Relations have an argument, nominal objects do not. Nominal objects have an attribute INDEX instead. As we can see in the signature, there are unary relations in the grammar. They have exactly one argument. The love relation, however, requires two arguments. At the subsort *love_rel* of *relations*, a second argument, ARG2, is introduced for this purpose. As a consequence of these changes not only will the output descriptions to lexical queries look sensible, they will also have a reasonable meaning: The verb *love* has exactly two semantic arguments, *walk* has exactly one argument, and the meaning of *she* is represented by an index and nothing else.

Let us now look at the extension of the theory of the grammar (theory.pl). Our theory file now looks as follows:

```
% Multifile declarations.
:- multifile '##'/2.
:- multifile '~~>'/2.
```

[10] If you are not sure about the meaning of $\left[\text{CASE } nom\right]$ relative to the signature of Section 3.1.1, you might want to use MoMo to examine it. You may simply load the file `signature` as your signature in MoMo and check satisfaction of the descriptions which you are interested in! You will, of course, have to draw feature structures.

```
% load phonology and tree output
:- [trale_home(tree_extensions)].

% specify signature file
signature(signature).

% lexical entries
she ~~> (synsem:(category: (head:case:nom,
                            subcat:e_list),
                 content: (index: (X, (person:third,
                                       number:sing,
                                       gender:fem))),
                 context: (backgr:[(female_rel, arg1:X)]))).

her ~~> (synsem:(category: (head:case:acc,
                            subcat:e_list),
                 content: (index: (X, (person:third,
                                       number:sing,
                                       gender:fem))),
                 context: (backgr:[(female_rel, arg1:X)]))).

walks ~~> (synsem: (category: (head:vform:fin,
                               subcat:[(category:head:case:nom,
                                        content:index: (X,(person:third,
                                                           number:sing
                                                          )))]),
                    content:(walk_rel,
                             arg1:X))).

loves ~~>  (synsem: (category: (head:vform:fin,
                               subcat:[(category:head:case:acc,
                                        content:index: Y),
                                       (category:head:case:nom,
                                        content:index: (X,
                                                        person:third,
                                                        number:sing))]),
                 content: (love_rel,arg1:X,arg2:Y))).

i ~~> (synsem:(category: (head:case:nom,
                          subcat:e_list),
                 content: (index:(person:first,
                                  number:sing)))).
```

```
walk ~~> (synsem: (category: (head:vform:fin,
                              subcat:[(category:head:case:nom,
                                       content:index: (X,(person:first,
                                                          number:sing
                                                         ))),
                  content:(walk_rel,
                           arg1:X))).


% phrase structure rules
subject_head_rule ##
(phrase, synsem:(category:(head:H,
                           subcat:[]),
                 content:Content),
 dtr1:Subj, dtr2:Head)
   ===>
cat> (Subj, synsem:Synsem),
cat> (Head, synsem:(category:(head:H,
                              subcat:[Synsem]),
                    content:Content)).

head_complement_rule ##
(phrase, synsem:(category:(head:H,
                           subcat:(ne_list, Rest)),
                 content:Content),
 dtr1:Comp, dtr2:Head)
   ===>
cat> (Head, synsem:(category:(head:H,
                              subcat:[First|Rest]),
                    content:Content)),
cat> (Comp, synsem:First).
```

The first seven lines contain the same technical declarations about the treatment of reserved symbols (for declaring phrase structure rules and lexical entries) in the system and about loading a particular file for parsing, as we saw before. Also we still use a signature file called `signature`.

In the part of the file which specifies the grammar itself, the lexical entries now precede the phrase structure rules. Obviously the order of our specifications does not matter here. The only difference with the lexical entries is the addition of four new ones. There is a second personal pronoun in the nominative case: *I*, a personal pronoun in the accusative case: *her*, and there is a new, transitive verb form: *loves*. Finally there are now two forms of the verb *walk* in the lexicon. We have the form *walks* and the form *walk*. Both of them are specified as finite verb forms, but they have different subcategorization requirements.

The form *walks* requires a third person singular subject by the specification of the single element on its SUBCAT list, whereas *walk* requires a first person singular subject. This lexical requirement can be understood as a simple, lexical theory of agreement. The co-indexations between the respective INDEX values under the CONTENT attribute of the subcategorized subject and the semantic argument under the CONTENT value of the verb itself guarantees sensible albeit very simple semantic representations.

The theory of case assignment is equally simple at this point. The finite verb forms require their subjects (the last elements on their SUBCAT lists!) to be CASE *nom*. The transitive verb *loves* requires a second argument (the first element of its SUBCAT list) with the accusative case.

Note that the lexical specification of *she* had not changed compared to the first grammar, although the part of the signature which structures possible *cont* entities has.

Ultimately we would like to formulate a linguistic theory which says something about the apparent systematic relationship between *walk* and *walks*, and which does not posit the two words as seemingly completely independent entities. It is easy to see that it would be immediately beneficial for the size of our grammar implementation to have such a theory. At the moment we have to write two lexical entries for (almost) every finite verb in order to distinguish third person singular forms from others. A general solution would be to reduce the necessary specifications to a combination of a base lexical entry (or to an appropriate morphological unit) and a general theory. In order to do this we need to develop a theory, applicable to all words of a given class, about the forms in which the words may occur in utterances. We will see a concrete version of one possible theory which does this in Section 3.2.2, where we will discuss *Lexical Rules* in TRALE.

Instead of having one phrase structure rule, we now have two. The `subject_head_rule` has not changed. The second phrase structure rule is constructed for the syntactic realization of the objects of transitive verbs such as *loves*. Just as with the first phrase structure rule, the `head_complement_rule` incorporates a number of principles of grammar in its specification. Firstly it takes over the task of an ID SCHEMA in an ID PRINCIPLE. Secondly it specifies part of a principle for word order (since it says that the phonology of the head (in this case DTR2) precedes the phonology of the non-head, the value of DTR1). It also performs a part of the job of the SUBCATEGORIZATION PRINCIPLE, since it dictates that the first element on the SUBCAT list of the head daughter is the SYNSEM value of the non-head daughter, and the rest of the SUBCAT list of the head daughter is passed on to the mother via the variable `Rest`. Finally it guarantees that phrases of this particular type look as if they were licensed by the HEAD FEATURE PRINCIPLE and the SEMANTICS PRINCIPLE because the variables `H` and `Content` enforce the identity of the HEAD and CONTENT values of the mother and the head daughter, which is what the HEAD FEATURE PRINCIPLE and the SEMANTICS PRINCIPLE would do.

It is important to notice, however, that we still do not have any real principles of grammar. There are no implicational descriptions in this new theory file either, and the fact that the grammar specifies a small fragment of English as if we had a HEAD FEATURE PRINCIPLE, a SEMANTICS PRINCIPLE, etc., is a mere side effect of the way in which we specified the phrase structure rules. To observe the relevance of this point, consider adding

a second head complement rule just like the one in our second TRALE grammar, but leaving out the use of the variable H. The second head complement rule would then look like this:

```
head_complement_rule_2 ##
(phrase, synsem:(category:subcat:(ne_list, Rest),
                 content:Content),
 dtr1:Comp, dtr2:Head)
   ===>
cat> (Head, synsem:(category:subcat:[First|Rest],
                    content:Content)),
cat> (Comp, synsem:First).
```

Then what would happen? Suddenly our grammar would license phrases which no longer obey the HEAD FEATURE PRINCIPLE! In an HPSG grammar an effect of this nature could never occur by *adding* a principle to the theory. Once the theory contains a restriction such as the HEAD FEATURE PRINCIPLE, that restriction is always present. Adding another principle will add another restriction, but it will never remove the effects of a previously existing one.

It is thus clear that when implementing HPSG grammars it is dangerous to integrate general principles into specific phrase structure rules. The leading idea should be to keep the general principles as implicational descriptions. If we had a separate HEAD FEATURE PRINCIPLE in our grammar, it would automatically restrict new types of phrases which were licensed by new phrase structure rules as expected, and we would not have to worry about integrating all the necessary effects of principles into the formulation of phrase structure rules.

We will now pursue this crucial idea of factoring out general principles of grammar by investigating how we may extract the general principles which are implicitly present from our two phrase structure rules. This will also give us a better idea about which elements of HPSG grammars can be taken over by phrase structure rules to make them more efficient, and what should not be taken over by phrase structure rules if we want to ensure that our grammar exhibits a structure which is close to HPSG grammars.

How could we factor out a SEMANTICS PRINCIPLE from the two phrase structure rules? The easiest part of the exercise is to remove the elements which concern semantic representations. We simply delete the parts of the descriptions in the rules which mention CONTENT values:

```
% modified phrase structure rules
new_subject_head_rule ##
(phrase, synsem:(category:(head:H,
                           subcat:[])),
 dtr1:Subj, dtr2:Head)
   ===>
cat> (Subj, synsem:Synsem),
cat> (Head, synsem:category:(head:H,
                             subcat:[Synsem])).

new_head_complement_rule ##
(phrase, synsem:category:(head:H,
                           subcat:(ne_list, Rest)),
 dtr1:Comp, dtr2:Head)
   ===>
cat> (Head, synsem:category:(head:H,
                             subcat:[First|Rest])),
cat> (Comp, synsem:First).
```

If we parse with this newly modified grammar we will see that the CONTENT values of phrases are now arbitrary, because they are no longer restricted in any way. In what way do we want to restrict them? In our tiny fragment of English it is very simple: The CONTENT value of phrases should always be the CONTENT value of the head daughter, which always happens to be the semantic head of our phrases. This is after all exactly the restriction which was originally built into the formulation of the phrase structure rules. As an independent, general principle of grammar it looks of course like this:

```
phrase *> (synsem:content:Content,
           dtr2:synsem:content:Content).
```

The reason for choosing to share the CONTENT value of DTR2 with the CONTENT value of the syntactic mother node is as follows: in formulating the phrase structure rules we ensured that the value of DTR2 is always the head daughter of the phrases, irrespective of the relative linear order of its phonology to the phonology of the DTR1 value. For the purposes of the SEMANTICS PRINCIPLE which we have just formulated, we may then generalize over DTR2 values.

After adding the SEMANTICS PRINCIPLE to the new version of the file `theory.pl` which already contains the new versions of the phrase structure rules, we have an implementation of our second fragment of English which is equivalent to the version that we started with. The crucial difference is that we have now begun to modify it in such a way that it looks more like a true HPSG grammar! This is the path which we will pursue further in the following sections.

Exercises

Exercise 36 *Our second grammar (signature[11], theory[12]) licenses the sentences* I walk *and* She walks *while correctly excluding* *I walks *and* She walk. *Add two new pronouns to the grammar,* you *and* they*, and modify the lexical entry of* walk *in such a way that the grammar licenses* You walk *and* They walk *with a reasonable representation for agreement and semantics, while excluding ungrammatical sentences which may result from the syntactic combination of the given words.*

Exercise 37 *In the text we have shown how to factor out the* Semantics Principle *from the original formulation of our two phrase structure rules. Take the two new versions of the phrase structure rules and factor out a* Head Feature Principle*. Your new grammar will then consist of two even shorter phrase structure rules, a* Head Feature Principle*, a* Semantics Principle*, and the lexical entries.*

Use TRALE to develop and test your modifications!

GLOSSARY

3.1.3 The Third TRALE Grammar

Abstract

The third TRALE grammar is essentially a lexical extension of the previous grammar. We will observe more properties of TRALE grammars, talk about how the syntactic form of logically equivalent grammar specifications can lead to different answers to queries, and specify a logical re-interpretation of the phrase structure rules in the HPSG formalism.

In studying the properties of a third grammar which will extend the lexicon of the grammar that we investigated in the previous section, we will discuss three topics. First we will discuss restrictions on appropriateness specifications in TRALE signatures. Our second topic is the behavior of TRALE's output. We will see how equivalent grammar specifications can lead to syntactically different results. In contrast to a purely model-theoretic approach, where only meaning matters, we will observe that in grammar implementations the syntactic form of the input specification might also matter for the user of the grammar. To conclude our output-related considerations we will introduce a useful option for manipulating what is displayed by the graphical user interface, GRiSU.

The last point on our agenda is the most complex and also the most important one: We want to understand what the phrase structure rules of TRALE mean in terms of the logical grammar specification of HPSG. For this purpose, we will reformulate the phrase structure rules as a principle of an HPSG grammar. Using this principle we will then be able to continue the extraction of conceptually independent principles of grammar from the

[11] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik2/signature
[12] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik2/theory.pl

phrase structure rules which we already started in the previous section, having extracted the SEMANTICS PRINCIPLE.

Transforming the phrase structure rules initially into a principle of grammar of the kind that we are already familiar with makes it easier to see what it means to factor out general grammatical principles from our first formulation of phrase structure rules. This is because we already have experience with logical reformulations of principles. The factoring out of conceptually independent principles is nothing but a logically equivalent reformulation of one principle of grammar as several principles of grammar. The meaning of implicational descriptions is already very familiar to us, and we understand how an implicational description can be reformulated as one or more implicational descriptions of different syntactic forms which have the same meaning.

Our aim then is to factor out all independent principles of grammar from our logical specification of the phrase structure rules as a single grammatical principle. After we complete our reformulation we will translate the resulting HPSG grammar back into a TRALE grammar. Our main goal in this enterprise is to understand how we can use the construct of phrase structure rules for efficient parsing of HPSG grammars without giving up more than what is absolutely necessary of the formal structure of logically pure HPSG grammars.

TRALE Signatures The signature of our third small fragment of English (Fig. 3.4, signature[13]) is only a minimal extension of the old signature. Extra sorts and attributes under *cont* are necessary for the new lexical entries of *give*, and a modification in the lexical entry of first person personal pronouns. A third CASE value, *dat*, will serve to illustrate the computational behavior of TRALE under certain conditions.

The addition of the maximally specific sort *dat* is not significant. It is just one more subsort of *case*, its linguistic purpose is obvious, and we do not need to discuss it any further. What happens under *cont* is more theoretically interesting.

Firstly, there is one more sort, *speaker_rel*, under *unary_rel*. The sort *unary_rel* subsumes all the sorts which denote nodes with exactly one (*index* valued) outgoing arc, ARG1. The sort *speaker_rel* is used in *context* values of first person personal pronouns in order to indicate the anchoring of the pronoun's reference to the speaker in a given type of situation.

The second extension of the sort hierarchy under *cont* reveals important properties of TRALE signatures. The extension in question is necessary for modeling predicates with two or three arguments. In the signature of the second grammar *relations* had a second immediate subsort besides *unary_rel*, namely *love_rel*, which was the only sort in the *cont* hierarchy with two argument attributes, ARG1 and ARG2. In the new signature we also have one sort, *give_rel*, with three argument attributes, ARG1, ARG2, and ARG3, because 'giving' is semantically a relation between three existing entities in the world. One might wonder why we then introduce two new sorts instead of one by giving *love_rel* and *give_rel* an additional immediate supersort, *more_arg_rel*. As long as we have very few semantic relations, the following sort hierarchy with appropriateness conditions seems to be more

[13] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik3/signature`

```
type_hierarchy
bot
   list
     ne_list hd:bot tl:list
     e_list
   sign phon:ne_list synsem:synsem
      phrase dtr1:sign dtr2:sign dtrs:list
      word
   synsem category:cat content:cont context:conx
   cat head:head subcat:list
   head
      noun case:case
      verb vform:vform
   vform
      fin
      bse
   case
      nom
      acc
      dat
   cont
      relations arg1:index
        unary_rel
           walk_rel
           female_rel
           speaker_rel
        more_arg_rel arg2:index
           love_rel
           give_rel arg3:index
      nom_obj index:index
   conx backgr:list
   index person:person number:number gender:gender
   person
      first
      third
   number
      sing
      plur
   gender
      masc
      fem
.
```

Figure 3.4: The signature of our third TRALE grammar

economical:

```
cont
   relations arg1:index
      unary_rel
         walk_rel
         ...
      love_rel arg2:index
      give_rel arg2:index arg3:index
   ...
```

The reason we cannot do this in TRALE (although it would be perfectly legitimate for the signatures of HPSG) is that there is a condition on how we are allowed to introduce attributes in the sort hierarchy, the so-called *Feature Introduction Condition*. The Feature Introduction Condition says that for each attribute which is appropriate to a sort there must be a unique sort σ in the sort hierarchy, such that (a), the attribute is appropriate to σ, and, (b), σ subsumes all other sorts to which the attribute is appropriate. In the simplified hierarchy which we have just sketched this condition is violated by ARG2, because there is no unique common supersort of *love_rel* and *give_rel* to which ARG2 is appropriate.

The signature of the third grammar does not have this problem because it adds the sort *more_arg_rel*, and declares ARG2 appropriate to it. The sort *more_arg_rel* is then the unique highest sort in the overall sort hierarchy to which ARG2 is appropriate, and both *love_rel* and *give_rel* "inherit" the attribute from *more_arg_rel*. The Feature Introduction Condition is a restriction to the form of legitimate signatures in TRALE which we have to bear in mind when we implement grammars. It is motivated by the inferencing mechanisms which TRALE uses for computing with grammars.

Let us now examine the extension of the grammar's theory (theory.pl[14]). The changes only concern the addition of new lexical entries. Our theory file now looks as follows:

```
% Multifile declarations.
:- multifile '##'/2.
:- multifile '~~>'/2.

% load phonology and tree output
:- [trale_home(tree_extensions)].

% specify signature file
signature(signature).
```

[14] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik3/theory.pl

```
% lexical entries
i ~~>     synsem:(category:(head:case:nom,
                            subcat:e_list),
                  content:(index: (X,(person:first,
                                      number:sing))),
                  context:(backgr:[(speaker_rel,arg1:X)])).

me ~~>    synsem:(category:(head:case:(acc;dat),
                             subcat:e_list),
                   content:(index: (X,(person:first,
                                       number:sing))),
                   context:(backgr:[(speaker_rel,arg1:X)])).

she ~~>   synsem:(category:(head:case:nom,
                            subcat:e_list),
                  content:(index: (X,(person:third,
                                      number:sing,
                                      gender:fem))),
                  context:(backgr:[(female_rel,arg1:X)])).


her ~~>   synsem:(category:(head:case:(acc;dat),
                             subcat:e_list),
                   content:(index: (X,(person:third,
                                       number:sing,
                                       gender:fem))),
                   context:(backgr:[(female_rel,arg1:X)])).

milk ~~>  synsem:(category:(head:noun,
                            subcat:e_list),
                  content:(index:(person:third,
                                  number:sing))).

walk ~~>  synsem:(category:(head:vform:fin,
                             subcat:[(category:(head:case:nom),
                                      content:index:(X,
                                                     person:first,
                                                     number:sing)
                                                     )]),
                   content:(walk_rel,
                            arg1:X)).

walks ~~>  synsem:(category:(head:vform:fin,
```

```
                              subcat:[(category:(head:case:nom),
                                       content:index:(X,
                                                      person:third,
                                                      number:sing)
                                                      )]),
                     content:(walk_rel,
                              arg1:X)).

love ~~> synsem:(category:(head:vform:fin,
                              subcat:[(category:(head:case:acc),
                                       content:index: Y),
                                      (category:(head:case:nom),
                                       content:index: (X,
                                                       person:first,
                                                       number:sing)
                                                       )]),
                     content:(love_rel,
                              arg1:X,
                              arg2:Y)).


loves  ~~> synsem:(category:(head:vform:fin,
                              subcat:[(category:(head:case:acc),
                                       content:index: Y),
                                      (category:(head:case:nom),
                                       content:index: (X,
                                                       person:third,
                                                       number:sing)
                                                       )]),
                     content:(love_rel,
                              arg1:X,
                              arg2:Y)).

give ~~> synsem:(category:(head:vform:fin,
                              subcat:[(category:(head:case:dat),
                                       content:index: Z),
                                      (category:(head:case:acc),
                                       content:index: Y),
                                      (category:(head:case:nom),
                                       content:index: (X,
                                                       person:first,
                                                       number:sing)
                                                       )]),
```

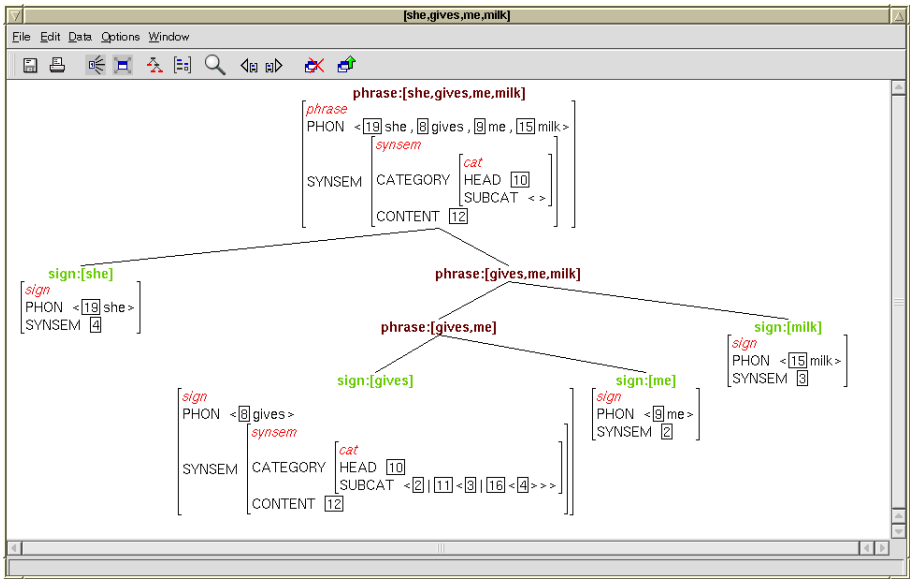
```
                    content:(give_rel,
                             arg1:X,
                             arg2:Y,
                             arg3:Z)).

gives ~~>  synsem:(category:(head:vform:fin,
                             subcat:[(category:(head:case:dat),
                                      content:index: Z),
                                     (category:(head:case:acc),
                                      content:index: Y),
                                     (category:(head:case:nom),
                                      content:index: (X,
                                                      person:third,
                                                      number:sing)
                                                      )]),
                    content:(give_rel,
                             arg1:X,
                             arg2:Y,
                             arg3:Z)).


% phrase structure rules
subject_head_rule ##
(phrase, synsem:(category:(head:H,
                           subcat:[]),
                 content:Content),
 dtr1:Subj, dtr2:Head)
   ===>
cat> (Subj, synsem:Synsem),
cat> (Head, synsem:(category:(head:H,
                              subcat:[Synsem]),
                    content:Content)).

head_complement_rule ##
(phrase, synsem:(category:(head:H,
                           subcat:(ne_list, Rest)),
                 content:Content),
 dtr1:Comp, dtr2:Head)
   ===>
cat> (Head, synsem:(category:(head:H,
                              subcat:[First|Rest]),
                    content:Content)),
cat> (Comp, synsem:First).
```

The first seven lines contain the same technical declarations as our earlier fragments of English, and we still call the signature file `signature`. The grammar now contains more lexical entries of pronouns. Two pronouns (*me* and *her*) receive a disjunctive case specification: They are either *acc*(usative) or *dat*(ive). Adding the lexical entry of the ditransitive verb *give* introduces new kinds of binary branching structures to our syntax. In the sentence *She gives me milk*, *gives* first combines with *me* by the `head_complement_rule`, and *gives me* then combines with *milk*, again by the `head_complement_rule`.



Output-related Issues The disjunctive specification of the possible CASE values of the pronouns *me* and *her* in the lexicon has a remarkable effect on the output behavior of TRALE, which indirectly tells us something about its computational properties. The query `lex her.` now returns two descriptions instead of one; we have a non-disjunctive description for every possible CASE value. We can conclude that the disjunction is promoted to word level by the system.

The behavior we have observed is not simply determined by the fact that two feature structures of different shape are in the denotation of the grammar. To illustrate this, consider the pronoun *I* as specified in our lexicon and its possible GENDER values. The sort *gender* is appropriate for GENDER, and *gender* has two subsorts, *masc* and *fem*. Since the lexical entry of *I* does not mention GENDER values (and no other part of the grammar restricts possible GENDER values of pronouns), *I* is ambiguous between GENDER *masc* and GENDER *fem*. The system expresses this when it answers the query `lex i.` by simply not mentioning the GENDER value. This changes when we change the lexical entry of *I* to a denotationally equivalent description with explicit disjunction. Suppose we have the description:

```
i ~~>     synsem:(category:(head:case:nom,
                            subcat:e_list),
                  content:(index: (X,(person:first,
                                      number:sing,
                                      gender:(fem;masc)))),
                  context:(backgr:[(speaker_rel,arg1:X)])).
```

Nothing has changed with respect to the meaning of our grammar; however the computational output of TRALE has changed, since TRALE operates on a syntactic level: `lex i.` now returns two descriptions, one for each possible GENDER value of *I*.

These observations teach us a very important lesson about grammar implementations: When implementing a grammar, we can influence the output behavior of queries by varying the syntactic form in which our grammar is written. Thus the purpose of a grammar and its most appropriate output behavior quite often influence the grammar writer's choice between various possible equivalent formulations! It is in fact a very important skill to know how to write grammars in a form that will lead to an output behavior which is most suitable for a given task. Achieving this goal can quickly become difficult when we go beyond simple specifications of lexical entries and take into account the interaction of various different complex grammar principles.

The output of a grammar is not only a matter of the grammar specification itself. There are also a number of functions of the graphical interface software GRiSU, which are in some sense more superficial but also very helpful, which you might want to use to fine tune the displaying of parsing results. In contrast to the output behavior with disjunctions, GRiSU has nothing to do with the underlying computations. GRiSU only responds to your demands concerning how you want to see the data displayed which TRALE transmits to the graphical user interface. A display function of GRiSU which you might want to try at this point allows you to suppress certain uninformative features in the graphical output. For example we have said that the DTRS feature in our current grammars is only needed for technical reasons, and we do not use it in our specifications. Thus it would be clearer and more economical if this feature were not displayed in the output. To achieve this, add the line

```
hidden_feat(dtrs).
```

to the file `theory.pl` just behind the technical specifications at the beginning. After saving the changes and recompiling the grammar, you will discover that GRiSU no longer mentions the DTRS attribute and DTRS values in answers to queries. Should you discover while working with a grammar that you want to see the hidden features for any reason, you can choose the option 'Show Hidden Nodes' in the menu of GRiSU windows. Attributes which are hidden for whatever reason are then displayed in light gray, and by clicking on them you prompt GRiSU to show descriptions of their values.

If you like to work with the graphical tree output of GRiSU rather than with full AVM mode you might also want to add

```
hidden_feat(dtr1).
hidden_feat(dtr2).
```

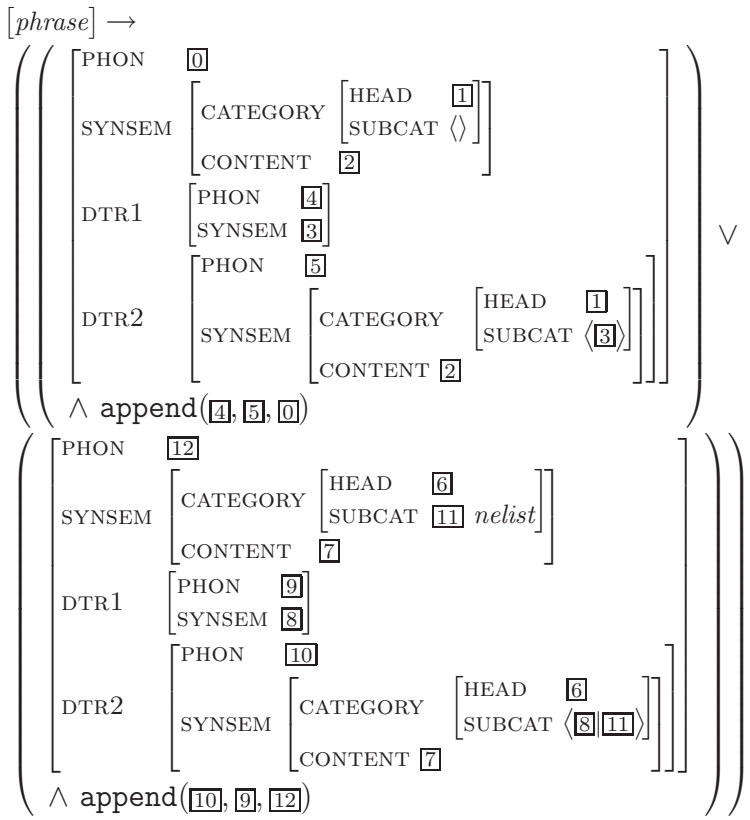
to `theory.pl`, which suppresses displaying the syntactic daughter attributes and their value in the output when an answer to a query first comes up. If you prefer the tree display, your output will be more compact and less redundant. However most of these display specifications are a matter of taste, and you will have to find out yourself what you can work with best.

Phrase Structure Rules Re-interpreted as an ID Specification Now that we know a little more about interpreting and manipulating output, let us return to the logical properties of our grammar implementation. Our third TRALE grammar, just like the second one, only specifies the lexicon and two phrase structure rules with descriptive decorations. There are still no implicational principles of the kind that we know from the theories of HPSG grammars. In order to understand the effect of the annotated phrase structure rules on the feature structure models of our TRALE grammar, we must reformulate the grammar by recreating the effect of the phrase structure rules with a standard grammatical principle.

As we have previously indicated, in terms of HPSG all decorated phrase structure rules taken together specify something similar to an ID Principle. To be more precise, each phrase structure rule may be interpreted as an ID Schema. Our current ID Schema-like phrase structure rules differ from normal ID Schemata in that they specify many more properties of phrases than ID Schemata usually do. For every type of phrase they incorporate specifications which linguists would like to specify independently in separate principles of grammar. The reason why they are normally specified as independent principles is that these principles formulate generalizations over the entire grammar rather than just over particular phrase types. We would like to do the same in our grammar implementations and specify a separate Head Feature Principle, a separate Semantics Principle, and a separate Subcategorization Principle. However, before we take that step, let us consider an HPSG specification of the phrase structure rules of our implementation first.

We will choose the linguistic AVM notation as our description language because of its easy readability. It should be simple to see how to reformulate the principle in MoMo notation. Note the implicit appeal to the existential closure convention in (32). All variables are supposed to be bound with wide scope existential quantification.

(32) *An HPSG description replacing the phrase structure rules of the TRALE grammar*



A phrase is licensed by our grammar if it satisfies one of the disjuncts in the consequent of the given principle. Note that the tags 1, 2, and 3 in the first disjunct of the principle's consequent and the tags 6, 7 and 8 in the second disjunct of the principle correspond to the variables `H`, `Content` and `Synsem` in the description annotation of the phrase structure rules. The specifications achieved by these variables enforce the effect of the HEAD FEATURE PRINCIPLE, the SEMANTICS PRINCIPLE, and the SUBCATEGORIZATION PRINCIPLE in modularly built grammars.

The daughter values of each phrase handled by the TRALE specification of the grammar with the descriptions behind `cat>` in the phrase structure rules proper are restricted by the description of the values of DTR1 and DTR2 in the consequent of the principle. In TRALE the phrase structure rule itself fixes the order of the phonological strings of the daughters in the overall phrase by imposing a linear order on the daughters. According to our principle the order of the phonological contributions of the daughters to the phrase is determined using the tags 0, 4, 5, 9, 10 and 12, and the relation `append`. We presuppose of course the presence in the grammar of an APPEND PRINCIPLE of the kind which we discussed earlier in the course.[15]

[15] In fact, an entirely precise reformulation in feature logic of the meaning of phrase structure rules would also have to guarantee the finiteness of the phonological strings and the graph-theoretic tree structure of phrase structure. We could modify the standard APPEND PRINCIPLE somewhat in order to achieve this. For the sake of simplicity we will ignore these difficult details here.

Note the notation used for lists in the second disjunct of the consequent: [8] refers to the first element on the SUBCAT list of the second daughter, whereas [11] refers to the entire SUBCAT list of the mother, which is identical to the tail of the SUBCAT list of the second daughter. The notation ⟨[8]|[11]⟩ with the operator | is supposed to express that [8] refers to the first element of the list, and [11] refers to the entire tail of the list.

Which "normal" HPSG principles are inherent to the technical device of phrase structure rules and cannot be factored out, and which ones can be removed? Since every phrase structure rule adds the possibility of a new type of phrase in the denotation of the grammar, they necessarily do the job of the disjuncts in the consequent of an ID PRINCIPLE. The second property inherent to them is word order. Stating a phrase structure rule means stating the relative linear order of the phonologies of their daughters. This property is one of the major differences to ID SCHEMATA, which in general do not say anything about the relative linear order of the phonologies of their daughters in the overall phrase. On the contrary, the possibility of having an independent word order module is one of the advantages in analytical flexibility offered by HPSG in contrast with traditional phrase structure grammars. This is exploited in the HPSG literature on so-called *linearization grammars* [Kathol, 1995]. Linearization grammars envision an analysis of discontinuous constituents by positing a certain independence of the phonological combinatorics from the syntactic combinatorics in grammars. The (relative) independence of the syntactic and the phonological combinatorics in linearization grammars means that the phonology of phrases is not necessarily a concatenation of the phonologies of their daughters, but there may be a more complicated relationship between them. For example, the phonology of a phrase might still consist of the sum of the phonologies of its daughters, but the phonological string of one daughter might be broken up and made discontinuous in the phonology of the phrase by the insertion of phonological material from another daughter.

Let us consider a concrete example: According to linearization grammars one might argue that in the German sentence *Karl hat den Hund gefüttert, auf den er seit einer Woche aufpasst* the noun phrase *den Hund* and its restrictive relative clause *auf den er seit einer Woche aufpasst* form a syntactic constituent. This means that the complex NP with the phonology *den Hund, auf den er seit einer Woche aufpasst* and the verb with the phonology *gefüttert* are syntactic sister nodes in the syntactic tree. However, a linearization rule—in combination with a few other systematic architectural features of linearization grammars—allows that the phonology of the verb *gefüttert* in this kind of constellation may intervene between the phonology of the nominal phrase, *den Hund*, and its relative clause, *auf den er seit einer Woche aufpasst*, in the phonology of their mother node. The optionality of this phonological intervention also predicts the alternative word order *den Hund, auf den er seit einer Woche aufpasst, gefüttert*.

When we use the phrase structure component of TRALE we gain in computational efficiency by losing the generality of HPSG's phonological combinatorics. TRALE responds to this situation by once more offering linguists alternative computational devices: The final MiLCA release of TRALE contains a *linearization parser* that provides a special syntax for the specification of linearization rules, which are not as strictly tied to the syntactic combinatorics as the word order rules that automatically come with phrase structure rules.

Returning to our analysis of which parts from (32) must necessarily be contained in specifications of phrase structure rules, we conclude that this only holds for those aspects which concern word order and phrasal combinatorics. In general, if we are careful about how we formulate phrase structure rules, they may be read as ID SCHEMATA which fix the relative word order of their daughters. All other aspects of (32) can be reformulated as independent implicational principles.

In Section 3.1.2 we already began to go in this direction with our second TRALE grammar. We established that the SEMANTICS PRINCIPLE captures the independent linguistic observation *The semantics of a phrase is identical to the semantics of its semantic head*, which, in our small grammars, is identical to the syntactic head. To mirror this linguistic generalization in the structure of our grammar, we factored out the SEMANTICS PRINCIPLE from the two phrase structure rules and stated it as a principle of a reformulated version of our second TRALE grammar. In the present section we were finally able to state the phrase structure rules of our TRALE grammar in terms of an ID PRINCIPLE. With the ID PRINCIPLE we can of course quite easily do exactly the same thing that we did with the phrase structure rules: We can factor out the generalization about the semantics of phrases.

In (33) we will factor out the SEMANTICS PRINCIPLE from (32). The result is an ID PRINCIPLE which no longer mentions CONTENT values, and a second description, which we call SEMANTICS PRINCIPLE.

(33) *HPSG descriptions replacing the phrase structure rules of the TRALE grammar*

ID PRINCIPLE, preliminary version

$$[\mathit{phrase}] \rightarrow$$

$$\left(\left(\left(\begin{array}{l}\text{PHON} \quad \boxed{0} \\ \text{SYNSEM} \quad \left[\text{CATEGORY} \left[\begin{array}{l}\text{HEAD} \quad \boxed{1} \\ \text{SUBCAT} \quad \langle\rangle\end{array}\right]\right] \\ \text{DTR1} \quad \left[\begin{array}{l}\text{PHON} \quad \boxed{4} \\ \text{SYNSEM} \quad \boxed{3}\end{array}\right] \\ \text{DTR2} \quad \left[\begin{array}{l}\text{PHON} \quad \boxed{5} \\ \text{SYNSEM} \quad \left[\text{CATEGORY} \left[\begin{array}{l}\text{HEAD} \quad \boxed{1} \\ \text{SUBCAT} \quad \langle\boxed{3}\rangle\end{array}\right]\right]\end{array}\right]\end{array}\right) \wedge \texttt{append}(\boxed{4}, \boxed{5}, \boxed{0})\right) \vee$$

$$\left(\left(\begin{array}{l}\text{PHON} \quad \boxed{12} \\ \text{SYNSEM} \quad \left[\text{CATEGORY} \left[\begin{array}{l}\text{HEAD} \quad \boxed{6} \\ \text{SUBCAT} \quad \boxed{11}\ \mathit{nelist}\end{array}\right]\right] \\ \text{DTR1} \quad \left[\begin{array}{l}\text{PHON} \quad \boxed{9} \\ \text{SYNSEM} \quad \boxed{8}\end{array}\right] \\ \text{DTR2} \quad \left[\begin{array}{l}\text{PHON} \quad \boxed{10} \\ \text{SYNSEM} \quad \left[\text{CATEGORY} \left[\begin{array}{l}\text{HEAD} \quad \boxed{6} \\ \text{SUBCAT} \quad \langle\boxed{8}\,|\,\boxed{11}\rangle\end{array}\right]\right]\end{array}\right]\end{array}\right) \wedge \texttt{append}(\boxed{10}, \boxed{9}, \boxed{12})\right)\right)$$

SEMANTICS PRINCIPLE

$$[phrase] \rightarrow \begin{bmatrix} \text{SYNSEM CONTENT} & \boxed{1} \\ \text{DTR2 SYNSEM CONTENT} & \boxed{1} \end{bmatrix}$$

If we add two more principles to our new ID PRINCIPLE and SEMANTICS PRINCIPLE, such as a WORD PRINCIPLE with our lexical entries and an APPEND PRINCIPLE, we will have listed all descriptions of a full HPSG specification of the grammar into which we translated our second TRALE grammar at the end of the previous section, where we separated a SEMANTICS PRINCIPLE from the two phrase structure rules. The underlying signature would of course be almost identical with the signature of the second TRALE grammar, except for the fact that we have to declare subsorts of a sort *phonstring* for all words in the grammar, as well as the `append` relation and its arity in the signature. This means, however, that we can now give a complete HPSG formalization of the TRALE grammar of the previous section, with the SEMANTICS PRINCIPLE factored out of the ID PRINCIPLE.

Let us now return to the third grammar which we started in the present section. Taking the signature of our third TRALE grammar (and adding the `append` symbol with arity 3), we now have an explicit HPSG formulation of our third TRALE grammar with the SEMANTICS PRINCIPLE factored out of the phrase structure rules. We can now go further and specify the HEAD FEATURE PRINCIPLE and the SUBCATEGORIZATION PRINCIPLE independent of the ID PRINCIPLE: We simply reformulate our present theory of grammar into an equivalent theory of a different syntactic form. This gives us an HPSG specification with all independently motivated generalizations over the observed linguistic data, neatly separated in independent principles of grammar. The only linguistic aspect which we may not touch if we want to be able to maintain a direct correspondence of the HPSG specification and the TRALE implementation of the grammar is the tight connection between the syntactic and the phonological combinatorics. Each ID SCHEMA must specify the phonology of the mother node as a fixed concatenation of the phonologies of the daughter signs. But this property is a property of all phrase structure grammars which work with traditional phrase structure rules, and at the moment we are simply formulating our HPSG grammars in a manner similar to a phrase structure grammar. The restrictions on possible analyses of word order are thus not difficult to accept as long as we do not pursue a linearization-based HPSG analysis of a natural language. As soon as we do this, we have to resort to the linearization parsers of TRALE.

Let us now take our present HPSG specification of the third TRALE grammar in (33) and pull out the HEAD FEATURE PRINCIPLE from the ID PRINCIPLE. We observe the fact that the HEAD value of a phrase is always identical to the HEAD value of the sign under DTR2 as a separate requirement:

(34) ID Principle, *improved preliminary version*

$$
\begin{array}{l}
[\textit{phrase}] \rightarrow \\
\left(\left(\left(\begin{array}{l}
\left[\begin{array}{ll}\textsc{phon} & \boxed{0}\\ \textsc{synsem} & \left[\textsc{category subcat } \langle\rangle\right]\end{array}\right]\\
\textsc{dtr1} \left[\begin{array}{ll}\textsc{phon} & \boxed{4}\\ \textsc{synsem} & \boxed{3}\end{array}\right]\\
\textsc{dtr2} \left[\begin{array}{ll}\textsc{phon} & \boxed{5}\\ \textsc{synsem} & \left[\textsc{category subcat } \langle\boxed{3}\rangle\right]\end{array}\right]\\
\wedge\ \mathtt{append}(\boxed{4},\boxed{5},\boxed{0})
\end{array}\right)\right) \vee \\
\left(\left(\begin{array}{l}
\left[\begin{array}{ll}\textsc{phon} & \boxed{12}\\ \textsc{synsem} & \left[\textsc{category subcat } \boxed{11}\ \textit{nelist}\right]\end{array}\right]\\
\textsc{dtr1} \left[\begin{array}{ll}\textsc{phon} & \boxed{9}\\ \textsc{synsem} & \boxed{8}\end{array}\right]\\
\textsc{dtr2} \left[\begin{array}{ll}\textsc{phon} & \boxed{10}\\ \textsc{synsem} & \left[\textsc{category subcat } \langle\boxed{8}|\boxed{11}\rangle\right]\end{array}\right]\\
\wedge\ \mathtt{append}(\boxed{10},\boxed{9},\boxed{12})
\end{array}\right)\right)
\end{array}
$$

(35) Semantics Principle

$$[\textit{phrase}] \rightarrow \begin{bmatrix}\textsc{synsem content} & \boxed{1}\\ \textsc{dtr2 synsem content} & \boxed{1}\end{bmatrix}$$

(36) Head Feature Principle

$$[\textit{phrase}] \rightarrow \begin{bmatrix}\textsc{synsem category head} & \boxed{1}\\ \textsc{dtr2 synsem category head} & \boxed{1}\end{bmatrix}$$

Since the head Feature Principle takes care of the identity of head values, they do not have to be mentioned in the ID Principle any longer.

We are now ready to take the final step and pull out the Subcategorization Principle from the current ID Principle in (34). Here we should make the observation that the first element on the subcat list of the head daughter, which is the value of dtr2, is identical to the synsem value of the non-head daughter; and the subcat list of the mother node equals the tail of the subcat list of the head daughter. Since we will state this in the Subcategorization Principle, we no longer have to express it in the ID Principle:

(37) ID Principle, *final version*

$$
\begin{array}{l}
[\mathit{phrase}] \rightarrow \\
\left(\left(\left(\begin{array}{ll}
\textsc{phon} & \boxed{0} \\
\textsc{synsem} & [\textsc{category subcat } \langle\rangle] \\
\textsc{dtr1} & [\textsc{phon } \boxed{4}] \\
\textsc{dtr2} & [\textsc{phon } \boxed{5}]
\end{array}\right)\right) \vee \right. \\
\left.\left(\left(\begin{array}{ll}
\textsc{phon} & \boxed{12} \\
\textsc{synsem} & [\textsc{category subcat } \mathit{nelist}] \\
\textsc{dtr1} & [\textsc{phon } \boxed{9}] \\
\textsc{dtr2} & [\textsc{phon } \boxed{10}]
\end{array}\right)\right)\right) \\
\wedge\ \mathtt{append}(\boxed{10}, \boxed{9}, \boxed{12})
\end{array}
$$

Note that we still have to mention the SUBCAT value of the syntactic mother nodes in the two ID Schemata. This is necessary because we have to ensure that the phonology of the first daughter (the non-head daughter) precedes the phonology of the second daughter (the head daughter) in the phonology of the phrase, exactly in those phrases in which the non-head daughter is the subject. Otherwise we have a head complement phrase, and the phonology of the head must precede the phonology of the non-head in the phonology of the phrase. In our small fragment we know that phrases with an empty SUBCAT list have only just realized their subject, and phrases with a non empty SUBCAT list have realized a complement.

Here come the other principles:

(38) Semantics Principle

$$
[\mathit{phrase}] \rightarrow \begin{bmatrix}
\textsc{synsem content} & \boxed{1} \\
\textsc{dtr2 synsem content} & \boxed{1}
\end{bmatrix}
$$

(39) Head Feature Principle

$$
[\mathit{phrase}] \rightarrow \begin{bmatrix}
\textsc{synsem category head} & \boxed{1} \\
\textsc{dtr2 synsem category head} & \boxed{1}
\end{bmatrix}
$$

(40) Subcategorization Principle

$$
[\mathit{phrase}] \rightarrow \begin{bmatrix}
\textsc{synsem category subcat} & \boxed{1} \\
\textsc{dtr1 synsem} & \boxed{2} \\
\textsc{dtr2 synsem category subcat} & \langle \boxed{2} | \boxed{1} \rangle
\end{bmatrix}
$$

We have not spelled out the WORD PRINCIPLE and APPEND PRINCIPLE here, but we note that they are still presupposed for the full HPSG specification of our reformulated third TRALE grammar.

We may now quite easily translate back our theoretical grammar specification with an independent ID PRINCIPLE, a SEMANTICS PRINCIPLE, a HEAD FEATURE PRINCIPLE and a SUBCATEGORIZATION PRINCIPLE (plus WORD PRINCIPLE and APPEND PRINCIPLE) into a TRALE grammar with the following properties: The ID PRINCIPLE is represented as phrase structure rules (one for each of its ID SCHEMATA); the SEMANTICS PRINCIPLE, HEAD FEATURE PRINCIPLE and SUBCATEGORIZATION PRINCIPLE do not change. They remain the same as in their theoretical specification; the WORD PRINCIPLE is represented by stating its lexical entries in TRALE's special syntax for lexical entries; and the APPEND PRINCIPLE can be ignored, since its purpose in the theoretical specification is realized through the use of phrase structure rules. We have thus established a transparent correspondence between a fully specified HPSG grammar and its implementation in TRALE, and we can reliably investigate the properties of our HPSG grammar using the computational devices provided by TRALE.

With a simple example, we have succeeded with our initial goal for grammar implementations: We have worked out how to use the tools of TRALE to get computational feedback about our theories of language through their implementation.

Exercises

Exercise 38 *Implement our modular HPSG specification in (37)–(40) of the original third TRALE grammar (signature[16], theory.pl[17]) in TRALE (the lexicon may just be copied from the theory file of the third TRALE grammar). Since it is an equivalent reformulation of the grammar we started with, queries to the system should still receive the same answers as before.*

GLOSSARY

3.1.4 Relations as Definite Clauses in TRALE

ABSTRACT

With three versions of the fourth grammar we will introduce definite clause attachments to TRALE descriptions. Using definite clause attachments with grammar implementations in TRALE is the closest we can get to the relations in the description language of HPSG.

In this section we will discuss three very closely related variants of the same grammar. For the sake of simplicity we will refer to all of them collectively as the *fourth grammar*,

[16] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik3/signature
[17] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik3/theory.pl

although they really constitute the fourth to the sixth grammar. The justification for treating them as if they were one grammar is that they have exactly the same empirical coverage: They license exactly the same set of sentences in English, but they do this in slightly different ways. For every new variation of the grammar we will argue that it is a theoretical improvement on the previous formulation, and every improvement will introduce new means of specifying grammars in TRALE.

The version of the fourth grammar which we will start with (signature[18], theory1.pl[19]) is essentially an implementation in TRALE of the modular reformulation of the third grammar which we constructed at the end of the previous section. All linguistic principles which we formulated as independently motivated generalizations over the structure of English in our discussion of the logical reformulation of the original phrase structure rules are contained in this version, as well as the two (minimized) phrase structure rules which now correspond to simple ID SCHEMATA, and a specification of the lexicon.

The only extension in empirical coverage introduced by the fourth grammar is lexical: The word *think* is new in the lexicon. This slight change leads to a major new property of the grammar. For the first time, it licenses infinitely many sentences. This effect is caused by the fact that *think* subcategorizes for a sentential complement, which may of course contain *think* as its verb again, and so on.

For the semantic representations of *think*, we need a new binary relation in the sort hierarchy under *relations*. The necessary modification can be found under the sort *more_arg_rel* in the new signature, see Figure 3.5. In addition, there is a new subsort of *cont* called *arg*, which is an immediate supersort of *index* and of *relations*. Its purpose is to generalize within the appropriateness specifications over *index*-valued and *relations*-valued argument attributes of semantic relations. There are no further changes in the signature.

The theory file of our fourth TRALE grammar also shows a few modifications which diverge from the third grammar. In the technical declarations at the beginning, we added three statements about hiding features in the GRiSU output: We do not want to be shown the attributes DTRS, DTR1 and DTR2 and descriptions of their values in AVMs. With bigger output descriptions from parse queries, it is quite convenient to hide tree-configurational descriptions from AVMs, and we are not interested in the DTRS feature anyway.

The previous lexical entries all remain unchanged. The new entries for the two forms of *think* are worth looking at. The finite forms of *think* require an NP subject in nominative case with agreement properties adjusted to the limited number of pronouns which occur in our small fragment of English. The second syntactic argument, which is subcategorized for by the first element on the SUBCAT list, is a saturated, finite verbal projection.

Directly after the lexical specifications of the theory we find two phrase structure rules. These are the phrase structure rules which correspond to the ID SCHEMATA of the final version of the ID PRINCIPLE of Section 3.1.3. The SEMANTICS PRINCIPLE, the HEAD FEATURE PRINCIPLE, and the SUBCATEGORIZATION PRINCIPLE of Section 3.1.3 follow in the familiar MoMo syntax, which is a superset of the TRALE description language

[18] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik4/1vers/signature`
[19] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik4/1vers/theory1.pl`

```
type_hierarchy
bot
   list
     ne_list hd:bot tl:list
     e_list
   sign phon:ne_list synsem:synsem
      phrase dtr1:sign dtr2:sign dtrs:list
      word
   synsem category:cat content:cont context:conx
   cat head:head subcat:list
   head
      noun case:case
      verb vform:vform
   vform
      fin
      bse
   case
      nom
      acc
      dat
   cont
      nom_obj index:index
      arg
        index
        relations arg1:arg
          un_rel
            walk_rel
            female_rel
            speaker_rel
          more_arg_rel arg2:arg
            bin_rel
              love_rel
              think_rel
            give_rel arg3:arg
   conx backgr:list
   index person:person number:number gender:gender
   person
      first
      third
   number
      sing
      plur
   gender
      masc
      fem
.
```

Figure 3.5: The signature of our fourth TRALE grammar

syntax. Note the implicit existential quantification over the variables in the descriptions.

```
% Multifile declarations.
:- multifile '##'/2.
:- multifile '*>'/2.
:- multifile '~~>'/2.

% load phonology and tree output
:- [trale_home(tree_extensions)].

% specifications for the GRiSU output display
hidden_feat(dtrs).
hidden_feat(dtr1).
hidden_feat(dtr2).

% specify signature file
signature(signature).

% lexical entries

i ~~> (word, synsem:(category:(head:case:nom,
                               subcat:e_list),
                     content:(index: (X,(person:first,
                                         number:sing))),
                     context:(backgr:[(speaker_rel,arg1:X)]))).

me ~~> (word, synsem:(category:(head:case:(acc;dat),
                                subcat:e_list),
                      content:(index: (X,(person:first,
                                          number:sing))),
                      context:(backgr:[(speaker_rel,arg1:X)]))).

she ~~> (word, synsem:(category:(head:case:nom,
                                 subcat:e_list),
                       content:(index: (X,(person:third,
                                           number:sing,
                                           gender:fem))),
                       context:(backgr:[(female_rel,arg1:X)]))).

her ~~> (word, synsem:(category:(head:case:(acc;dat),
                                 subcat:e_list),
                       content:(index: (X,(person:third,
                                           number:sing,
```

```
                                        gender:fem))),
                    context:(backgr:[(female_rel,arg1:X)]))).

milk ~~> (word, synsem:(category:(head:noun,
                                subcat:e_list),
                    content:(index:(person:third,
                                    number:sing)))).

walk ~~>  (word, synsem:(category:(head:vform:fin,
                                subcat:[(category:(head:case:nom),
                                         content:index:(X,
                                                        person:first,
                                                        number:sing)
                                         )]),
                    content:(walk_rel,
                                arg1:X))).

walks ~~>  (word, synsem:(category:(head:vform:fin,
                                subcat:[(category:(head:case:nom),
                                         content:index:(X,
                                                        person:third,
                                                        number:sing)
                                         )]),
                    content:(walk_rel,
                                arg1:X))).

love ~~> (word, synsem:(category:(head:vform:fin,
                                subcat:[(category:(head:case:acc),
                                         content:index: Y),
                                        (category:(head:case:nom),
                                         content:index: (X,
                                                         person:first,
                                                         number:sing)
                                         )]),
                    content:(love_rel,
                                arg1:X,
                                arg2:Y))).

loves  ~~> (word, synsem:(category:(head:vform:fin,
                                subcat:[(category:(head:case:acc),
                                         content:index: Y),
                                        (category:(head:case:nom),
                                         content:index: (X,
```

```
                                                    person:third,
                                                    number:sing)
                                                    )]),
                    content:(love_rel,
                             arg1:X,
                             arg2:Y))).

give ~~> (word, synsem:(category:(head:vform:fin,
                                  subcat:[(category:(head:case:dat),
                                           content:index: Z),
                                          (category:(head:case:acc),
                                           content:index: Y),
                                          (category:(head:case:nom),
                                           content:index: (X,
                                                    person:first,
                                                    number:sing)
                                                    )]),
                    content:(give_rel,
                             arg1:X,
                             arg2:Y,
                             arg3:Z))).

gives ~~>  (word, synsem:(category:(head:vform:fin,
                                  subcat:[(category:(head:case:dat),
                                           content:index: Z),
                                          (category:(head:case:acc),
                                           content:index: Y),
                                          (category:(head:case:nom),
                                           content:index: (X,
                                                    person:third,
                                                    number:sing)
                                                    )]),
                    content:(give_rel,
                             arg1:X,
                             arg2:Y,
                             arg3:Z))).

think  ~~>  (word, synsem:(category:(head:vform:fin,
                                   subcat:[(category:(head:vform:fin,
                                                      subcat:[]),
                                           content: Y),
                                          (category:(head:case:nom),
                                           content:index: (X,
```

```
                                                    person:first,
                                                    number:sing)
                                                    )]),
                    content:(think_rel,
                             arg1:X,
                             arg2:Y))).

thinks ~~>  (word, synsem:(category:(head:vform:fin,
                                subcat:[(category:(head:vform:fin,
                                                   subcat:[]),
                                         content: Y),
                                        (category:(head:case:nom),
                                         content:index: (X,
                                                    person:third,
                                                    number:sing)
                                                    )]),
                    content:(think_rel,
                             arg1:X,
                             arg2:Y))).


% phrase structure rules

subject_head_rule ##
(phrase, synsem:category:subcat:[],
            dtr1:Subj, dtr2:Head)
   ===>
cat> Subj,
cat> Head.

head_complement_rule ##
(phrase, synsem:category:subcat:ne_list,
            dtr1:Comp, dtr2:Head)
   ===>
cat> Head,
cat> Comp.


% Principles

% Semantics Principle
phrase *> (synsem:content:C,
           dtr2:synsem:content:C).
```

```
% Head Feature Principle
phrase *> (synsem:category:head:H,
           dtr2:synsem:category:head:H).


% Subcategorization Principle (first version)
phrase *> (synsem:category:subcat:PhrSubcat,
           dtr1:synsem:Synsem,
           dtr2:synsem:category:subcat:[Synsem|PhrSubcat]).
```

The SUBCATEGORIZATION PRINCIPLE in our present grammar is written for a linguistic analysis which uses binary branching structures. Even complements of verbs are syntactically realized one at a time.

As long as we are content to write grammars of this kind, our SUBCATEGORIZATION PRINCIPLE as it stands is unproblematic. However, let us suppose that we want to develop a grammar of a bigger fragment of English. To do so we might start with a fragment of English as small as the one that we have at the moment. However, at some point during a stepwise extension of the coverage of the grammar we might decide that for certain linguistic reasons we want to switch to an analysis which realizes all complements of verbal heads as syntactic sisters of the verb, as in Pollard and Sag's theory of English. In the sentence *She gives her milk*, the complements *her* and *milk* would then both be syntactic sisters of the head of the construction *gives*.

What we want to modify in this situation is the syntactic tree structure of the grammar. Unfortunately, with a SUBCATEGORIZATION PRINCIPLE which is tied as directly to binary branching trees as the one we currently have, this is not possible. We need to rewrite the SUBCATEGORIZATION PRINCIPLE as well. From a linguistic point of view it would seem wise to formulate the SUBCATEGORIZATION PRINCIPLE from the very beginning in such a way that it has as few side effects on the theory of syntactic tree configurations as possible. We can achieve some degree of independence if we generalize our formulation and go in the direction of the most common formulation of the SUBCATEGORIZATION PRINCIPLE which can be found in the literature. This roughly says that in (headed) phrases, the SUBCAT list of the head daughter is in the `append` relation with the list of SYNSEM values of those elements that are syntactically realized as its sisters, and with the SUBCAT list of the phrase itself.

The question which immediately arises is what to do with the `append` relation. We know how to express relations in HPSG, but we have not yet seen relations in TRALE. In fact, for computational reasons relations are not part of the logical description language of TRALE at all. Instead TRALE offers a definite clause extension of the description language. This means that we may formulate definite clauses and attach their relation symbols as relational constraints to TRALE descriptions. For attaching relational constraints to descriptions we have a new operator: `goal`.

The definite clause language of TRALE is closely related to the definite clause lan-

guage of Prolog, and if you already have some experience with Prolog, writing relational constraints in TRALE will be very easy. The major difference is that the the first-order terms of Prolog are replaced with descriptions of feature structures in the definite clause layer of TRALE.

To understand what all of that means, it is best to look at an example. We will reformulate the SUBCATEGORIZATION PRINCIPLE with a relational constraint based on a definite clause definition of `append`. The relational constraint may be read as an approximation to the `append` relation as we know it from HPSG. (the complete grammar with the revised SUBCATEGORIZATION PRINCIPLE is available in Section 6.4.2, or for downloading: signature[20], theory2.pl[21]).

```
% Subcategorization Principle (second version)
phrase *> (synsem:category:subcat:PhrSubcat,
           dtr1:synsem:Synsem,
           dtr2:synsem:category:subcat:HeadSubcat)
goal
   append([Synsem],PhrSubcat,HeadSubcat).


% Goal definitions
append(L,[],L) if true.
append([],(L,ne_list),L) if true.
append([H|T1],(L,ne_list),[H|T2]) if
  append(T1,L,T2).
```

The relational attachment to the description says that the `append` predicate is a goal which holds for the three given argument values: the first argument is a singleton list which includes the entity denoted by `Synsem`, the second argument is the entity denoted by `PhrSubcat`, and the third argument is `HeadSubcat`. The meaning of `append` is given in terms of definite clauses. In our theory file the definite clauses immediately follow the SUBCATEGORIZATION PRINCIPLE. In order for the constraint to hold, the system has to prove that the goal is satisfied.

The way in which definite clauses are written up is determined to some extent by procedural properties. What becomes computationally relevant is, of course, influenced by the particular way in which a predicate is used as a goal. The leading idea is that we want calls to relational goals to be resolved as quickly as possible. This can easily lead to positing more clauses then logically necessary. For instance, although for `append` we could make do with two clauses from a logical point of view, procedurally it is better to use three clauses. These clauses distinguish between two base cases and a recursive case. Each clause may be read as an implication from right to left. In order of their occurrence they say that `append(L,[],L)` always holds without any further proof. A call of the

[20] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik4/2vers/signature`
[21] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik4/2vers/theory2.pl`

`append` goal is therefore successful if its first and last argument can be identified, and the second argument is an empty list. The goal is also held true without further proof if the first argument is an empty list, the second is a non-empty list, and the third argument is identical with the second argument (`append([],(L,ne_list),L)`). The recursive clause is defined for the case when the first argument *and* the second argument denote non-empty lists. If this is the case then the system has to prove that `append` holds between the tail of the list in the first argument, the non-empty list in the second argument, and the tail of the third argument, supposing that the first element of the lists in the first and third argument are identical.

The clauses for `append` have two base cases instead of one, (such as, for instance, in HPSG's Append Principle), in order to let the system determine as quickly as possible whether the relational attachment in the implementation of the Subcategorization Principle has been met. The system has completed the proof as soon as one of the lists in the first two arguments is found to be empty and the other two lists can be identified. If we were to write the definite clauses in parallel to the Append Principle, more computational steps could be necessary to perform the same task, depending on the context in which the goal is called by the system.

For more information on definite clauses in TRALE, the reader may want to consult the sections on Definite Clauses and on the TRALE-Extensions of descriptions in the TR/ALE manual.[22] When reading in the TR/ALE manual, please keep in mind that the logic and interpretation of the description language is slightly different from HPSG—as long as you keep this in mind and do not get confused by seemingly wrong statements about the meaning of descriptions and their relationship to feature structures, reading in the TR/ALE manual should be very useful at this point.

So far the elements on the subcat lists have been stated in an order which might have been surprising to readers familiar with the theory of English of Pollard and Sag, or with the current HPSG literature. We have ordered the elements in order of decreasing obliqueness: the element with the most oblique case on the list always came first, and the least oblique element, in our lexicon always an element in the nominative case, came last. In virtually all current HPSG grammars in the literature, we see the reverse order. The order is relevant for other parts of linguistic theory which rely on it, such as Binding Theory, or the theory of argument control, or analyses of argument raising.

There was a straightforward reason why we started with a reversed order in our computational implementation: It makes possible the very simple first Subcategorization Principle, which uses the list operator | to distinguish between the first element of a list and its tail, and is employed in a formulation of the principle which says that it is always the first element on the subcat list of a head which is realized as a syntactic daughter. This leaves the subject in the nominative case to be realized last as the last element of the subcat list, which is necessary for an easy implementation of syntactic argument realization in a phrase structure-like grammar. All in all, by reversing the canonical order of elements on subcat lists we were able to formulate the Subcategorization Principle

[22] `http://www.ale.cs.toronto.edu/docs/man/ale_trale_man/index.html`

without reference to relational goals which are computationally relatively expensive. All true recursion was put into the phrase structure component with its extremely efficient specialized parsing algorithms.

As we see, at this point efficiency comes at the price of deviation from linguistic theory. As long as it does not lead to further complications in implemented grammars, we might be prepared to pay that price occasionally. When grammars become larger and the consequences of computationally motivated changes in the theory more difficult to see, it is a good idea to be very conservative with modifications of this sort. Since we now know how to simulate HPSG's relations by definite clauses, and since we have definite clauses for `append`, we can now also reverse the order of relative obliqueness on SUBCAT lists and return to standard assumptions in the linguistic literature. Our third version of the fourth grammar, (signature[23], theory3.pl[24]), does that. Below we will repeat only one lexical entry and the SUBCATEGORIZATION PRINCIPLE with the necessary definite clauses from the new theory file.

We notice immediately that appending a list containing the SYNSEM value of the syntactically realized complement to the SUBCAT value of the mother (as opposed to doing it the other way around), has more computational consequences.

```
% lexical entries


...

gives ~~>  (word, synsem:(category:(head:vform:fin,
                              subcat:[(category:(head:case:nom),
                                       content:index: (X,
                                                       person:third,
                                                       number:sing)),
                                      (category:(head:case:acc),
                                       content:index: Y),
                                      (category:(head:case:dat),
                                       content:index: Z)]),
                    content:(give_rel,
                             arg1:X,
                             arg2:Y,
                             arg3:Z))).

...
```

[23] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik4/3vers/signature
[24] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik4/3vers/theory3.pl

```
% Subcategorization Principle (alternate version)

phrase *> (synsem:category:subcat:PhrSubcat,
           dtr1:synsem:Synsem,
           dtr2:synsem:category:subcat:HeadSubcat)
goal
   append(PhrSubcat,[Synsem],HeadSubcat).


% Goal definitions

append(X,Y,Z) if
   when(  ( X=(e_list;ne_list)
          ; Y=e_list
          ; Z=(e_list;ne_list)
               ),
            undelayed_append(X,Y,Z)).

undelayed_append(L,[],L) if true.
undelayed_append([],(L,ne_list),L) if true.
undelayed_append([H|T1],(L,ne_list),[H|T2]) if
  append(T1,L,T2).
```

The definite clause characterization of `append` is now defined with obvious emphasis on computational considerations which are necessary to ensure termination during parsing. As our readers may try out themselves, leaving the previous formulation of our definite clauses for `append` in a grammar with the elements on SUBCAT lists in standard order of obliqueness results in non-termination of parsing.

The reason for the system's behaving this way can be seen quite easily when we investigate what the new formulation of the single definite clause for `append` is doing. `append(X,Y,Z)` can only be proven at a point in computation when one of the following conditions hold: We have already inferred that `X` is either `e_list` or `ne_list` (i.e, we have gathered more precise information than just the general knowledge that there is a `list`); or that `Y` is of sort `e_list`; or that `Z` is either `e_list` or `ne_list`. Once the system is able to infer that much—but not before—it has to prove `undelayed_append(X,Y,Z)`, which is defined by the definite clauses which we used for simple `append` earlier. The difference is in the recursive case, in which `undelayed_append` calls `append` once again, thus ensuring that something has to be inferred about the arguments of the predicate before the system can execute computations in the recursive step.

The purpose of these so-called *delays* is clear: They make sure that the system performs a recursive operation only at a point where other components of the system have already drawn enough inferences to ensure that the system does not go into an infinite loop. Going into the recursion of `append` without enough information about its arguments would lead

the system into an infinite loop in the context in which the relational attachment of `append` is used in our final version of the SUBCATEGORIZATION PRINCIPLE.

Learning to write appropriate delays and to write definite clauses to simulate the relations in grammars belong to the most important set of skills one needs to acquire for grammar implementations. As is to be expected, this requires a lot of practice, and, in this particular case, a lot of grammar writing and debugging.

Exercises

Exercise 39 *The TRALE grammar (signature[25], theory.pl[26]) is a small grammar which is very similar to those that we have been working with. However, there are a few mistakes built into this grammar. You can see their effects if you parse sentences such as:*

1. *she walks*
2. *she loves her*
3. *i walk*
4. *she gives me milk*
5. *she loves me*

These sentences all get an unexpected number of answers from the system, which is due to the mistakes in the grammar. Other sentences get the right number of answers from the system. Find the mistakes in the theory file of the grammar, fix them, and write short comments into the file about what was wrong. Post the corrected theory file to your personal group and give the file an intuitive name. Something like `fixed-spook-theory.pl` *would be a good choice.*

To find the mistakes pay attention to what the TRALE compiler is telling you, and think about what typical typos might look like in grammars. Debugging your grammar is what always takes most of the time in any grammar implementation effort!

GLOSSARY

[25] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Spook/signature`
[26] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Spook/theory.pl`

3.2 Grammar Development

Abstract

We are now ready to combine linguistic grammar specifications with grammar implementation. We start by outlining the coverage and main linguistic structures of a core fragment of English, providing a specification of that core fragment, and implementing it in TRALE. The core fragment will form the nucleus for later additions and extensions. Extensions may add further theoretical modules of grammar (such as lexical generalizations) or analyses of theoretically interesting linguistic phenomena (such as unbounded dependency constructions). Any extension of the grammar specification can then be implemented as an extension of the core fragment.[27]

What is a *fragment grammar* or, shorter, a *fragment*?

In our context, which is provided by the topics of logical grammar specifications and their computational implementation, a fragment grammar consists of a complete and precise specification of a subset of the grammatical expressions of a natural language. In order to make the specification falsifiable the specification of a fragment requires an accurate characterization of the intended empirical domain, an indication of which grammar formalism the fragment grammar uses, and a specification of the grammar in the chosen formalism on an appropriate level of abstraction.

In linguistics papers we usually find a much looser understanding of the term fragment grammar. The fragment often consists of an in-depth analysis of a particular empirical phenomenon with precise statements about principles which formulate generalizations over the given empirical domain. Such a deep analysis is rarely embedded in an exact analysis of basic data which need to be described first in order to rigorously test the predictions of a grammar specification. Instead many papers refer to standard assumptions in the literature at the time of writing. On closer inspection, these "standard assumptions" may in fact turn out to be rather diverse and may exhibit vital differences in detail. The particular choice of a version of the "standard" theory might thus determine the fundamental structure of the resulting grammar.

In computer implementations we are virtually forced to adopt a concept of grammar fragments which adheres to the first, much more rigid understanding of the nature of a fragment grammar. Focusing on the analysis of a highly specialized empirical phenomenon without a specification of a minimum of basic grammatical notions is hardly viable in computer implementations. The basics might be kept really simple or even simplistic, but they have to be in place before we can think of making queries about sentences or constructions exhibiting more complex phenomena. To a certain degree, implementations of grammar fragments thus force linguists to be explicit about their assumptions. An advantage of this property of grammar implementations can be that it promotes a certain

[27] The specification of the core fragment and its extensions closely follows a teaching grammar jointly developed by Manfred Sailer and Frank Richter.

awareness of the plethora of seemingly innocent assumptions which go into a complete fragment. More often than not they are less innocent than they seem. In this respect implementing grammars can be a very sobering experience.

Grammar implementations force the computational linguist to be precise and to make decisions. However, as a careful theoretical linguist one should always be aware of the fact that a working computer implementation is not a sufficient indication of a clean and explicit linguistic theory. We already saw earlier in our discussion (Section 3.1.2) that the output of a parser first has to be interpreted by the user in order to determine whether it really has a reasonable *meaning*. Seemingly correct output might still have an absurd meaning or be meaningless altogether if the formal foundations of the parsing system fail to be clearly specified. In the latter case in particular a system might be very useful for certain practical tasks, but it loses much value for linguistic theory. Even if it performs extremely well for a specific purpose, it can be completely useless for linguistic theorizing.

Treated with a good degree of care, fragment grammars are an interesting link between theoretical linguistics and applied computational linguistics. We will be very conscious about what we do, and we will make our linguistic assumptions as well as computational necessities and motivations explicit. In this spirit we now turn to an outline of the coverage and basic assumptions about linguistic structures in our core fragment. The core fragment provides the basics of a grammar which describes simple sentential constructions of English. With an outline of the coverage and fundamental structures of the fragment in hand, we will present a specification in HPSG. In comments to our specification, we will explain why certain design decisions are made. From the abstract specification we then proceed to its implementation in TRALE. Anybody may falsify the correctness of the grammar implementation by showing that it misses the specification in some respect.

3.2.1 Fragment I – The Core fragment

Abstract

We will specify the syntax and goals of the core fragment, which underly all later extensions which cover more specialized linguistic phenomena.

The focus of the core fragment is on basic syntactic structures which support an analysis of finite sentential constructions. There is no real semantic analysis and no serious semantic representation. Under the CONTENT attribute, which is the classical place for semantic representations in HPSG, we place structures which indicate the main predicates and the semantic roles of their arguments. It is inspired by the thematic structures that form part of the sketch of a semantics in [Pollard and Sag, 1994]. Morphological and phonological components are entirely absent from the core fragment.

The syntax of the fragment combines ideas from various HPSG publications, without embracing one particular proposal wholesale. In order to keep the syntactic combinatorics simple, we have chosen a strictly binary branching phrase structure. As we will see, this leads to an unusual syntactic structure in at least one empirical area. We are not suggesting that binary branching structures are an empirically motivated solution which we would

want to defend on linguistic grounds. It is only adopted as a working strategy. It is quite likely that one would want to change that strategy in serious extensions of the coverage of the core fragment.

Let us now enumerate typical example sentences which are licensed by the core fragment. Each type of construction is followed by exemplary syntactic tree structures and additional comments. The tree structures and their labels abbreviate attributes and attribute values in the HPSG specification of the fragment.

The core fragment contains simple finite sentences with subjects and complements:

(41) a. Mary walks.

b. Peter likes Mary.

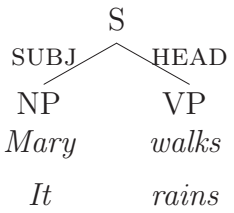
c. She gives it to Mary.

d. It rains.

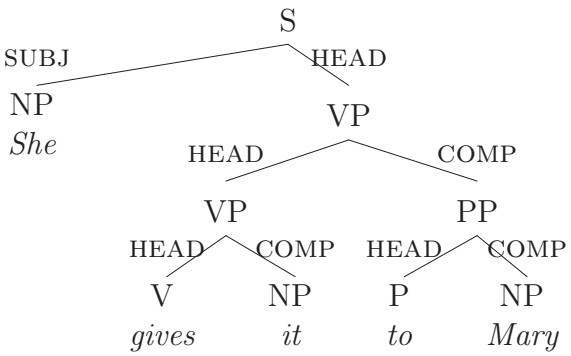
The pronoun *it* in sentence (41c) is referential; in sentence (41d), *it* is non-referential. Verbs determine the referentiality of their syntactic arguments.

(42) The structure of the examples in (41):

a.



b.

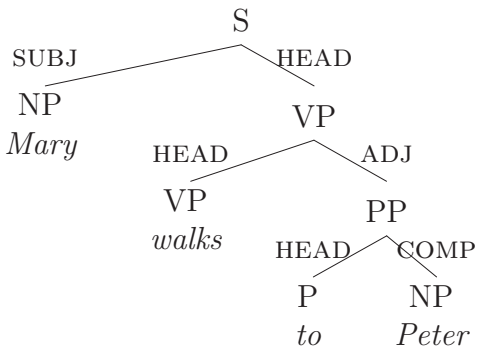


Simple finite sentences with complements as well as adjuncts belong to the core fragment:

(43) a. Mary walks here.

b. Mary walks to Peter.

(44) The structure of the examples in (43):

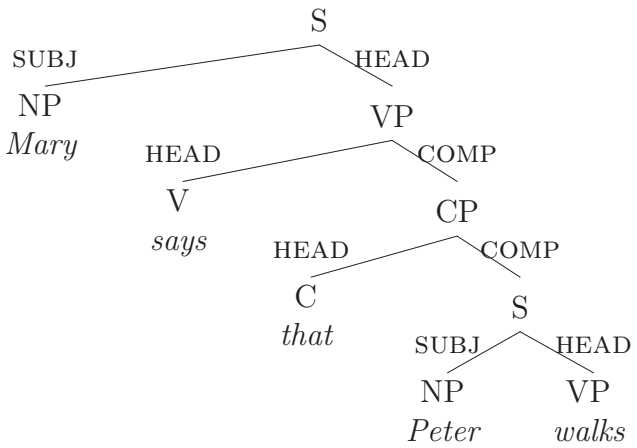


Finite complement clauses are another instance of complements:

(45) Mary says that Peter walks.

The grammar needs a mechanism to distinguish between *marked* sentences, which are sentences that are introduced by a complementizer such as *that*, and *unmarked* sentences, which are sentences that do not have a complementizer. The verb *say* in (45) requires a marked sentential complement in our fragment of English. Note that we assume that markers such as *that* subcategorize the sentences they introduce. They are the head of their sentences.

(46) The structure of the example in (45):



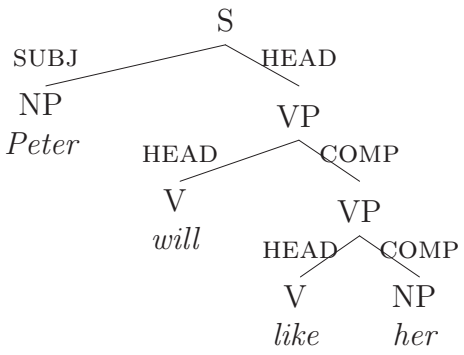
We include sentences with auxiliaries:

(47) a. Peter will like her.

b. It will rain.

Initially *will* is our only auxiliary in the grammar. It is an instance of a subject raiser: It raises the subject of the subcategorized VP and realizes it as its own syntactic subject, while the raised entity remains the logical subject of the main verb of the VP.

(48) The structure of the examples in (47):



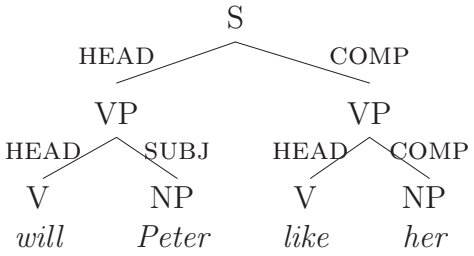
Finally we want to describe sentences with so-called subject-aux inversion, which are sentences in which the auxiliary precedes the subject:

(49) a. Will Peter like her?

b. Will it rain?

Due to the strictly binary branching syntactic structure, the tree representation of sentences with subject-aux inversion is surprising. The auxiliary first realizes its subject in post auxiliary position, and then it realizes its complement VP. In standard HPSG analyses one would assume a ternary structure in which both the subject and the VP are sisters of the auxiliary.

(50) The structure of the examples in (49):



To describe all of the sentences above our fragment must include treatments of the following linguistic notions: part of speech, morphosyntactic properties, subject verb agreement, government, syntactic constituenthood, constituent structure, syntactic headedness, a simple mechanism for selection, valency and thematic structure, grammatical functions (subject, object, adjunct), and argument raising.

3.2.1.1 Specification of the Core Fragment

Abstract

The specification of the core fragment consists of a signature, exemplary lexical entries for the kinds of verbs, nouns, prepositions and adverbs in the fragment, and all necessary principles of grammar.

Without further ado, we will state the signature:

The signature

```
type_hierarchy
top
   sign  phon:list(phonstring)  synsem:synsem
      word  arg_st:list(synsem)
      phrase  daughters:const_struc
   synsem  loc:loc  nonloc:nonloc
   loc  cat:cat cont:cont
   cat  head:head  val:val
   head  pred:boolean mod:synsem_none
      func_verb  vform:vform  marking:marking
         verb  aux:boolean  inv:boolean  marking:unmarked mod:none
         functional  marking:marked
      noun  case:case mod:none
      prep  pform
      adv  mod:synsem
   val  subj:list(synsem)  comps:list(synsem)
   cont
      psoa
         move_rel
            walk_rel  walker:ref
         like_rel  liker:ref  liked:ref
         say_rel   sayer:ref  said:psoa
         give_rel  giver:ref  gift:ref   given:ref
         rain_rel
         future_rel     soa_arg:psoa
         direction_rel  movement:move_rel  goal:ref
         here_rel   located:psoa
      nom_obj index:index
   index  num:num  pers:pers:  gen:gen
      ref
      nonref
         it
         there
   nonloc
```

```
const_struc  hdtr:sign  ndtr:sign
   hs_struc
   hc_struc
   ha_struc
   sai_struc
list
   elist
   nelist first:top  rest:list
vform
   fin
   inf
   base
   pas
   psp
case
   nom
   acc
pform
   lexical
   non_lexical
      to
marking
   unmarked
   marked
      that
boolean
   plus
   minus
pers
   first
   second
   third
num
   sg
   pl
gen
   fem
   masc
   neut
synsem_none
   none
   synsem
phonstring
   #peter#
   #mary#
```

```
      #he#
      #she#
      #her#
      #him#
      #you#
      #they#
      #them#
      #it#
      #rain#
      #rains#
      #walk#
      #walks#
      #like#
      #likes#
      #say#
      #says#
      #give#
      #gives#
      #will#
      #that#
      #here#
      #to#
relations
  append/3
.
```

To distinguish the sort symbols for the subsorts of *phonstring* from other sorts for which one would like to use the same symbol as for the grapheme of a word in the grammar (e.g. the PFORM value *to*), we add the symbol # to graphemes for words. Thus the symbol for PHON lists, *#to#*, is different from the sort label for PFORM values, *to*.

The theory We will begin the specification of the theory with examples of the different kinds of lexical entries in the grammar. The lexical entries are disjuncts in the consequent of the WORD PRINCIPLE stated below in (62). The logical structure of the WORD PRINCIPLE and the reasons for formulating lexical entries as disjuncts in its consequent are explained in detail in Section 2.5.1.

Lexical entries – nouns

(51) Lexical entry of the name *Peter*:

$$\begin{bmatrix} word \\ \text{PHON} & \langle \#peter\# \rangle \\ \text{SYNS} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} noun \\ \text{PRED} & minus \end{bmatrix} \end{bmatrix} \\ \text{CONT} & \begin{bmatrix} \text{INDEX} & \begin{bmatrix} ref \\ \text{NUM} & sg \\ \text{PERS} & third \\ \text{GEN} & masc \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{ARG\_ST} & \langle\rangle \end{bmatrix}$$

Subjects and complements of words, represented as values of the feature ARG_ST, are specified in the argument structure of all lexical entries. An ARGUMENT REALIZATION PRINCIPLE takes care of the distribution of the arguments over the lists representing the subject and the complement requirements of a word. Since *Peter* has an empty argument structure, it follows that it has an empty subject list and an empty complements list.

(52) Lexical entry of the (non referential) pronoun *it*:

$$\begin{bmatrix} word \\ \text{PHON} & \langle \#it\# \rangle \\ \text{SYNS} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} noun \\ \text{PRED} & minus \end{bmatrix} \end{bmatrix} \\ \text{CONT} & \begin{bmatrix} \text{INDEX} & \begin{bmatrix} it \\ \text{NUM} & sg \\ \text{PERS} & third \\ \text{GEN} & neut \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{ARG\_ST} & \langle\rangle \end{bmatrix}$$

Non referential *it* has an INDEX value of sort *it*. The other pronouns in the fragment are referential and have INDEX values of sort *ref*. The immediate subsort *there* of *nonref* in the signature indicates that there are more non referential words in English which will be treated analogously in an extended grammar.

Lexical entries – verbs Note that third person singular agreement of finite verbs with suffix *-s* is handled as a description of admissible elements on the SUBJ list. In the case of *walks*, the identification of the element on the SUBJ list with the single element on the ARG_ST list by virtue of the ARGUMENT REALIZATION PRINCIPLE then also leads to the identification of the INDEX value of the subject with the thematic role WALKER of the verb.

(53) Lexical entry of the verb *walks*:

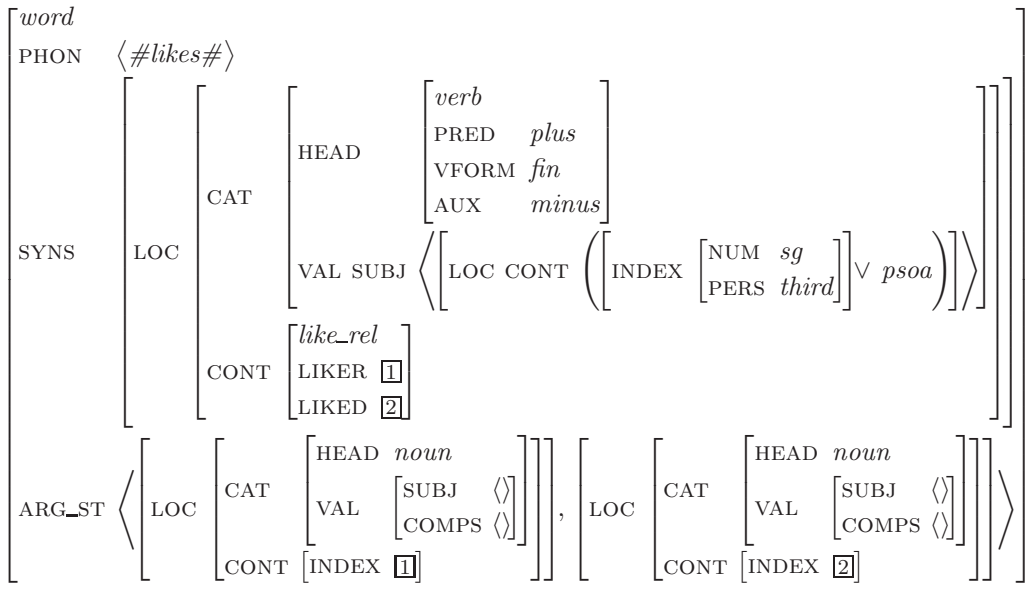
$$\begin{bmatrix} \textit{word} \\ \text{PHON} \quad \langle \#\textit{walks}\# \rangle \\ \text{SYNS} \quad \begin{bmatrix} \text{LOC} \begin{bmatrix} \text{CAT} \begin{bmatrix} \text{HEAD} \begin{bmatrix} \textit{verb} \\ \text{PRED} \quad \textit{plus} \\ \text{VFORM} \quad \textit{fin} \\ \text{AUX} \quad \textit{minus} \end{bmatrix} \\ \text{VAL} \begin{bmatrix} \text{SUBJ} \left\langle \begin{bmatrix} \text{LOC CONT} \left(\begin{bmatrix} \text{INDEX} \begin{bmatrix} \text{NUM} \quad \textit{sg} \\ \text{PERS} \quad \textit{third} \end{bmatrix} \end{bmatrix} \vee \textit{psoa} \right) \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT} \begin{bmatrix} \textit{walk\_rel} \\ \text{WALKER} \ \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{ARG\_ST} \left\langle \begin{bmatrix} \text{LOC} \begin{bmatrix} \text{CAT} \begin{bmatrix} \text{HEAD} \quad \textit{noun} \\ \text{VAL} \begin{bmatrix} \text{SUBJ} \quad \langle\rangle \\ \text{COMPS} \quad \langle\rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT} \begin{bmatrix} \text{INDEX} \quad \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix}$$

Non finite verb forms do not impose any requirements through a direct description of their subjects. Their SUBJ list is not mentioned in lexical entries (see also the lexical entry for the base form *give* below).

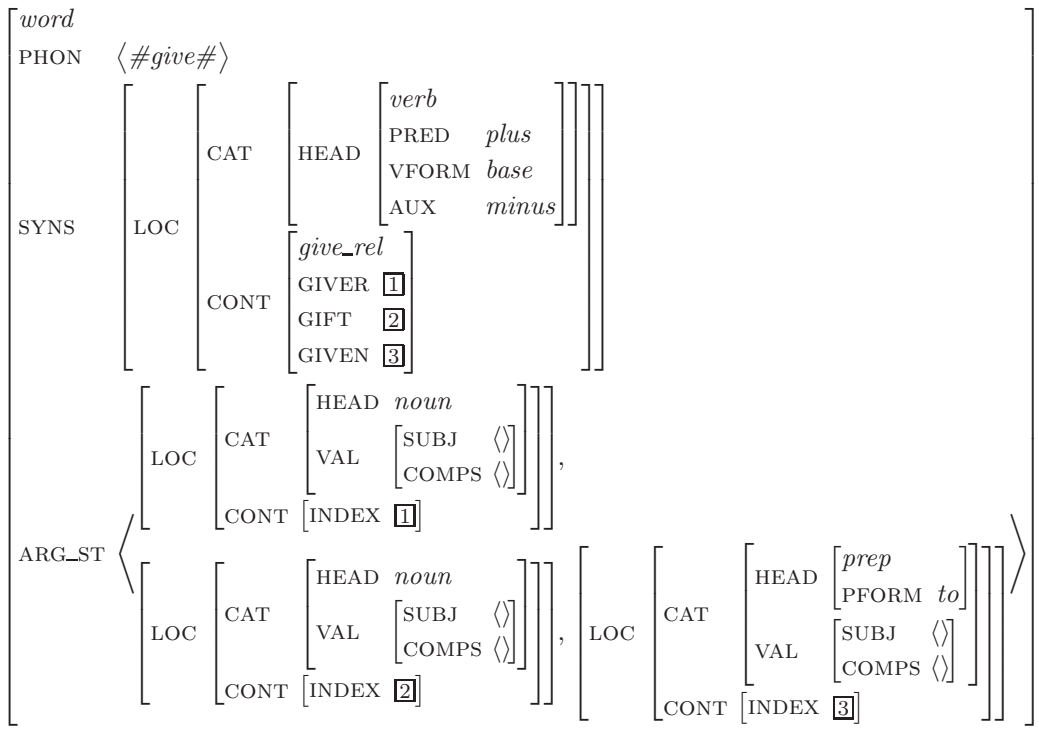
(54) Lexical entry of the verb *rain*:

$$\begin{bmatrix} \textit{word} \\ \text{PHON} \quad \langle \#\textit{rain}\# \rangle \\ \text{SYNS} \quad \begin{bmatrix} \text{LOC} \begin{bmatrix} \text{CAT} \begin{bmatrix} \text{HEAD} \begin{bmatrix} \textit{verb} \\ \text{PRED} \quad \textit{plus} \\ \text{VFORM} \quad \textit{base} \\ \text{AUX} \quad \textit{minus} \end{bmatrix} \end{bmatrix} \\ \text{CONT} \quad \textit{rain\_rel} \end{bmatrix} \end{bmatrix} \\ \text{ARG\_ST} \left\langle \begin{bmatrix} \text{LOC} \begin{bmatrix} \text{CAT} \begin{bmatrix} \text{HEAD} \quad \textit{noun} \\ \text{VAL} \begin{bmatrix} \text{SUBJ} \quad \langle\rangle \\ \text{COMPS} \quad \langle\rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT} \begin{bmatrix} \text{INDEX} \quad \textit{it} \end{bmatrix} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

(55) Lexical entry of the verb *likes*:



(56) Lexical entry of the verb *give*:



(57) Lexical entry of the future auxiliary verb *will*:

$$\begin{bmatrix}\text{PHON} & \langle \#will\# \rangle \\ \text{SYNS} & \begin{bmatrix}\text{LOC} & \begin{bmatrix}\text{CAT} & \begin{bmatrix}\text{HEAD} & \begin{bmatrix}\textit{verb} & \\ \text{PRED} & \textit{plus} \\ \text{VFORM} & \textit{fin} \\ \text{AUX} & \textit{plus}\end{bmatrix}\end{bmatrix} \\ \text{CONT} & \begin{bmatrix}\textit{future\_rel} & \\ \text{SOA\_ARG} & \boxed{2}\end{bmatrix}\end{bmatrix}\end{bmatrix} \\ \text{ARG\_ST} & \left\langle \boxed{1}, \begin{bmatrix}\text{LOC} & \begin{bmatrix}\text{CAT} & \begin{bmatrix}\text{HEAD} & \begin{bmatrix}\textit{verb} & \\ \text{VFORM} & \textit{base}\end{bmatrix} \\ \text{VAL} & \begin{bmatrix}\text{SUBJ} & \langle \boxed{1} \rangle \\ \text{COMPS} & \langle\rangle\end{bmatrix}\end{bmatrix} \\ \text{CONT} & \boxed{2}\end{bmatrix}\end{bmatrix} \right\rangle\end{bmatrix}$$

The auxiliary *will* is an argument raiser. It identifies the subject of its infinitival complement with the first element on its ARG_ST list, which becomes its own syntactic subject.

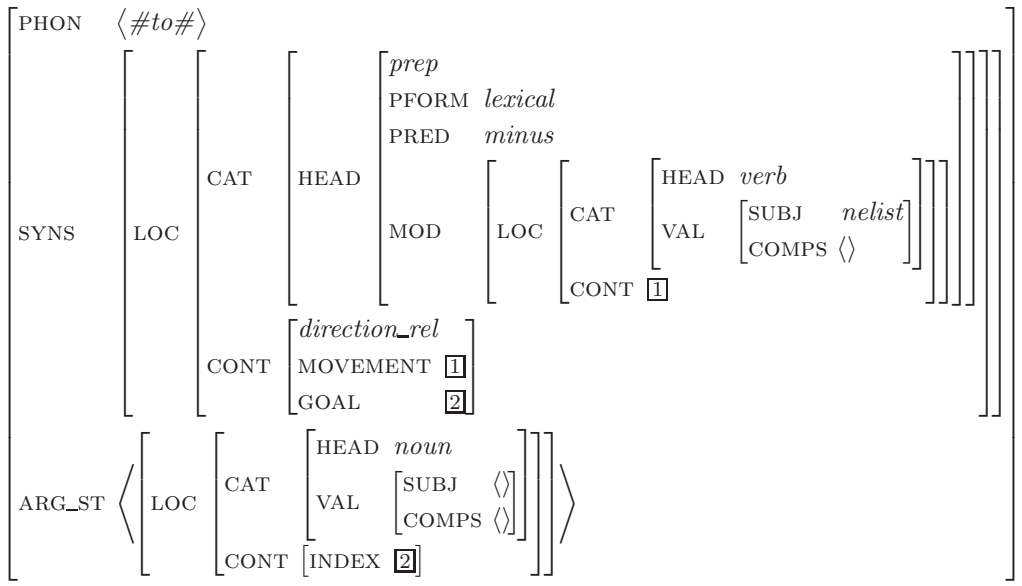
Lexical entries – prepositions The first type of preposition is represented by the case marking preposition *to*; it does not contribute any semantics of its own to phrases in which it occurs:

(58) Lexical entry of the (non lexical) preposition *to*:

$$\begin{bmatrix}\text{PHON} & \langle \#to\# \rangle \\ \text{SYNS LOC CAT HEAD} & \begin{bmatrix}\textit{prep} & \\ \text{PFORM} & \textit{to}\end{bmatrix}\end{bmatrix}$$

Lexical prepositions have their own semantics:

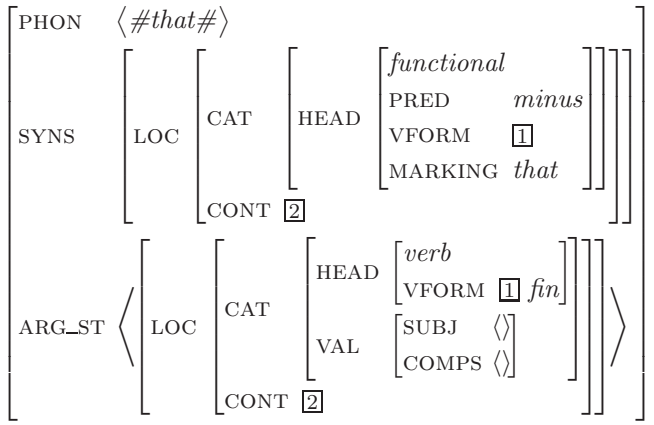
(59) Lexical entry of the (lexical) preposition *to*:



Adjuncts select the phrases they modify via their MOD attribute. The MOD value of the lexical preposition *to* selects verb phrases with an empty COMPS list which have not yet realized their subjects (non empty SUBJ list). This means that syntactically they are realized above complements but below the subject of verbal projections. Note that the appropriateness specifications in the signature determine that the MOVEMENT value inside the CONTENT of *to* must be of a subsort of *move_rel*.

Lexical entries – complementizer

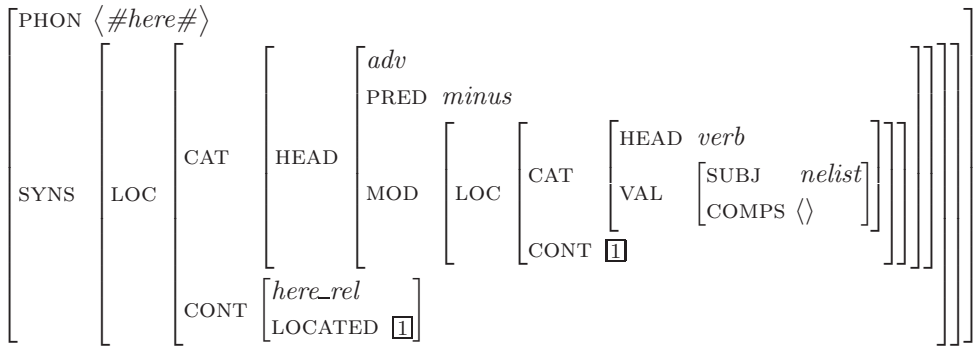
(60) The lexical entry of *that*:



Unlike the marker analysis of the complementizer *that* of Pollard and Sag, we assume that *that* heads the sentence it introduces. In our fragment it selects a finite, saturated verbal projection, has the same VFORM value as the selected clause and also inherits its CONTENT value.

Lexical entries – adverb

(61) Lexical entry of *here*:



The MOD value description of *here* has the same syntactic effect as the MOD value description of the lexical preposition *to* above.

Principles In our sketches of the WORD PRINCIPLE and the ID PRINCIPLE, we will use meta-variables to abbreviate the disjuncts in the consequents of the principles. We will not state precisely how many lexical entries of the kind described above occur in the WORD PRINCIPLE, but we have exactly four ID SCHEMATA in the consequent of the ID PRINCIPLE. These ID SCHEMATA are presented in full detail below ((64)–(67)).

(62) The WORD PRINCIPLE:

$word \rightarrow (\mathrm{LE}_1 \vee \ldots \vee \mathrm{LE}_n)$

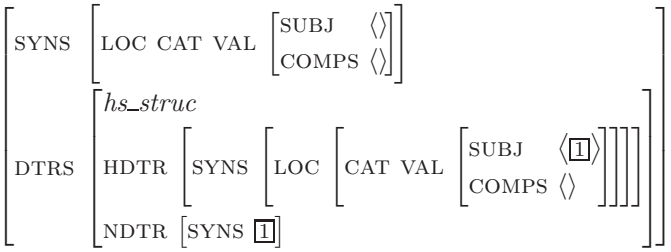
The entries LE_1 to LE_n are, of course, the lexical entries specified above (p. 182–187), with the possible addition of a few others which should be formed analogously in order to obtain a more interesting lexical basis for the sentences licensed by the grammar.

(63) The ID PRINCIPLE:

$phrase \rightarrow (\mathrm{HSS} \vee \mathrm{HCS} \vee \mathrm{SAIS} \vee \mathrm{HAS})$

Note that the syntactic realization of subjects and complements, which is usually governed by a SUBCATEGORIZATION PRINCIPLE, is built into the ID SCHEMATA.

(64) The HEAD-SUBJECT SCHEMA:



(65) The HEAD-COMPLEMENT SCHEMA:

$$\begin{bmatrix} \textsc{syns} & \begin{bmatrix} \textsc{loc} & \begin{bmatrix} \textsc{cat} & \textsc{val} & \begin{bmatrix} \textsc{subj} & \boxed{1} \\ \textsc{comps} & \boxed{3} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \textsc{dtrs} & \begin{bmatrix} \textit{hc\_struc} \\ \textsc{hdtr} & \begin{bmatrix} \textsc{syns} & \begin{bmatrix} \textsc{loc} & \begin{bmatrix} \textsc{cat} & \textsc{val} & \begin{bmatrix} \textsc{subj} & \boxed{1} \\ \textsc{comps} & \langle \boxed{2} \mid \boxed{3} \rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \textsc{ndtr} & \begin{bmatrix} \textsc{syns} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

(66) The SUBJECT-AUX-INVERSION SCHEMA:

$$\begin{bmatrix} \textsc{syns} & \begin{bmatrix} \textsc{loc} & \begin{bmatrix} \textsc{cat} & \textsc{val} & \begin{bmatrix} \textsc{subj} & \langle\rangle \\ \textsc{comps} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \textsc{dtrs} & \begin{bmatrix} \textit{sai\_struc} \\ \textsc{hdtr} & \begin{bmatrix} \textit{word} \\ \textsc{syns} & \begin{bmatrix} \textsc{loc} & \begin{bmatrix} \textsc{cat} & \begin{bmatrix} \textsc{head} & \begin{bmatrix} \textsc{inv} & \textit{plus} \end{bmatrix} \\ \textsc{val} & \begin{bmatrix} \textsc{subj} & \langle \boxed{1} \rangle \\ \textsc{comps} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \textsc{ndtr} & \begin{bmatrix} \textsc{syns} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

(67) The HEAD-ADJUNCT SCHEMA:

$$\begin{bmatrix} \textsc{syns} & \begin{bmatrix} \textsc{loc cat val} & \begin{bmatrix} \textsc{subj} & \boxed{1} \\ \textsc{comps} & \langle\rangle \end{bmatrix} \end{bmatrix} \\ \textsc{dtrs} & \begin{bmatrix} \textit{ha\_struc} \\ \textsc{hdtr} & \begin{bmatrix} \textsc{syns} & \boxed{2} \begin{bmatrix} \textsc{loc} & \begin{bmatrix} \textsc{cat} & \textsc{val} & \begin{bmatrix} \textsc{subj} & \boxed{1} \\ \textsc{comps} & \langle\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \textsc{ndtr} & \begin{bmatrix} \textsc{syns loc cat head mod} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Since there are no phrase structure rules in pure HPSG grammars we need to specify word order in a separate principle of grammar. Our CONSTITUENT ORDER PRINCIPLE orders the phonology of the syntactic daughters of a phrase on the PHON list of their mother node, depending on what kind of phrase we are looking at. In head-subject structures the phonology of the non-head daughter precedes the phonology of the head daughter. In all other types of phrases, the ordering of the phonology is the other way round:

(68) The CONSTITUENT ORDER PRINCIPLE:

$$\textit{phrase} \rightarrow \left(\begin{array}{l} \begin{bmatrix} \textsc{phon} & \boxed{1} \\ \textsc{dtrs} & \begin{bmatrix} \textsc{hdtr phon} & \boxed{2} \\ \textsc{ndtr phon} & \boxed{3} \end{bmatrix} \end{bmatrix} \\ \wedge \Big(\big[\textsc{dtrs} \ (\ \textit{hc\_struc} \vee \textit{ha\_struc} \vee \textit{sai\_struc} \ ) \big] \rightarrow \texttt{append}(\boxed{2}, \boxed{3}, \boxed{1}) \Big) \\ \wedge \Big(\big[\textsc{dtrs} \ (\ \textit{hs\_struc}) \big] \rightarrow \texttt{append}(\boxed{3}, \boxed{2}, \boxed{1}) \Big) \end{array} \right)$$

(69) The HEAD FEATURE PRINCIPLE:

$$phrase \rightarrow \begin{bmatrix} \text{SYNSEM LOC CAT HEAD } \boxed{1} \\ \text{DTRS HDTR } \left[\text{SYNS LOC CAT HEAD } \boxed{1}\right] \end{bmatrix}$$

(70) The SEMANTICS PRINCIPLE:

$$phrase \rightarrow \left(\begin{array}{l} \begin{bmatrix} \text{SYNS LOC CONT } \boxed{1} \\ \text{DTRS } \begin{bmatrix} \neg\ ha\_struc \\ \text{HDTR } \left[\text{SYNS LOC CONT } \boxed{1}\right] \end{bmatrix} \end{bmatrix} \\ \vee \begin{bmatrix} \text{SYNS LOC CONT } \boxed{1} \\ \text{DTRS } \begin{bmatrix} ha\_struc \\ \text{NDTR } \left[\text{SYNS LOC CONT } \boxed{1}\right] \end{bmatrix} \end{bmatrix} \end{array} \right)$$

The syntactic head is always the semantic head of a phrase with the exception of head-adjunct phrases, where the adjunct is the semantic head. For this reason the CONTENT value of the adjunct daughter is identical to the CONTENT value of the mother in head adjunct phrases.

(71) The INV PRINCIPLE:

$$\left[\text{SYNS LOC CAT HEAD } \left[\text{INV } plus\right]\right] \rightarrow \left[\text{SYNS LOC CAT HEAD } \begin{bmatrix} \text{VFORM } fin \\ \text{AUX } plus \end{bmatrix}\right]$$

If a verb occurs inverted (auxiliaries in subject-aux inversion constructions), then it must be finite and an auxiliary. In other words, non auxiliaries may not occur inverted.

(72) The FUNCTIONAL PREPOSITION PRINCIPLE:

$$\begin{bmatrix} word \\ \text{PHON } nelist \\ \text{SYNS LOC CAT HEAD } \begin{bmatrix} prep \\ \text{PFORM } non\_lexical \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{SYNS} & \left[\text{LOC } \left[\text{CAT HEAD } \begin{bmatrix} \text{MOD } none \\ \text{PRED } minus \end{bmatrix} \right]\right] \\ \text{ARG\_ST} & \left\langle \left[\text{LOC } \left[\text{CAT VAL } \begin{bmatrix} \text{SUBJ } \langle\rangle \\ \text{COMPS } \langle\rangle \end{bmatrix} \right]\right] \right\rangle \end{bmatrix}$$

All functional (or case marking) prepositions take exactly one saturated syntactic argument, and they identify their semantics with the semantics of their argument. The FUNCTIONAL PREPOSITION PRINCIPLE is in fact an instance of a lexical generalization, a generalization about the properties of an entire class of lexical items. It helps to keep lexical entries small.

(73) The MOD PRINCIPLE:

$$\begin{bmatrix} phrase \\ \text{DTRS } \neg\ ha\_struc \end{bmatrix} \rightarrow \left[\text{DTRS NDTR } \left[\text{SYNS LOC CAT HEAD } \left[\text{MOD } none\right]\right]\right]$$

According to the MOD PRINCIPLE, adjunct daughters are the only non-heads in our fragment which may select their corresponding head daughter with the MOD attribute, because all other types of daughters do not have a *synsem* valued MOD attribute.

(74) The Argument Realization Principle:

a. $\begin{bmatrix} word \\ \text{SYNS LOC CAT HEAD } \begin{bmatrix}\text{PRED } plus\end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{SYNS LOC CAT VAL } \begin{bmatrix}\text{SUBJ } \langle \boxed{1} \rangle \\ \text{COMPS } \boxed{2}\end{bmatrix} \\ \text{ARG\_ST } \langle \boxed{1} \mid \boxed{2} \rangle \end{bmatrix}$

b. $\begin{bmatrix} word \\ \text{SYNS LOC CAT HEAD } \begin{bmatrix}\text{PRED } minus\end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{SYNS } \begin{bmatrix}\text{LOC CAT VAL } \begin{bmatrix}\text{SUBJ } \langle\rangle \\ \text{COMPS } \boxed{1}\end{bmatrix}\end{bmatrix} \\ \text{ARG\_ST } \boxed{1} \end{bmatrix}$

The Argument Realization Principle determines how the elements on the argument structure of words are distributed over the subject and complements list. Their distribution is determined by whether or not a word is predicative.

(75) The Structural Case Principle:

a. For finite structures:

$$\exists \boxed{1} \left(\begin{bmatrix} phrase \\ \text{DTRS } \begin{bmatrix}\text{HDTR } \begin{bmatrix}\text{SYNS LOC CAT } \begin{bmatrix}\text{HEAD } \begin{bmatrix}\text{VFORM } fin\end{bmatrix} \\ \text{VAL } \begin{bmatrix}\text{SUBJ } \langle \boxed{1} \rangle\end{bmatrix}\end{bmatrix}\end{bmatrix} \\ \text{NDTR } \begin{bmatrix}\text{SYNS } \boxed{1} \begin{bmatrix}\text{LOC CAT HEAD } noun\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix} \right) \rightarrow \begin{bmatrix}\text{DTRS NDTR } \begin{bmatrix}\text{SYNS LOC CAT HEAD CASE } nom\end{bmatrix}\end{bmatrix}$$

b. All other cases:

$$\begin{bmatrix} phrase \\ \text{DTRS } \begin{bmatrix} hc\_struc \\ \text{NDTR } \begin{bmatrix}\text{SYNS LOC CAT HEAD } noun\end{bmatrix}\end{bmatrix}\end{bmatrix} \rightarrow \begin{bmatrix}\text{DTRS NDTR } \begin{bmatrix}\text{SYNS LOC CAT HEAD CASE } acc\end{bmatrix}\end{bmatrix}$$

According to the Structural Case Principle, the nominal subjects of finite words receive nominative case. All nouns in complement position receive accusative case in our small fragment of English.

(76) The Subject Principles:

a. $\begin{bmatrix} val \\ \text{SUBJ } nelist \end{bmatrix} \rightarrow \begin{bmatrix}\text{SUBJ } \left\langle \begin{bmatrix} synsem \end{bmatrix} \right\rangle\end{bmatrix}$

b. $\begin{bmatrix} \text{HEAD } verb \\ \text{VAL SUBJ } \langle\rangle \end{bmatrix} \rightarrow \begin{bmatrix}\text{HEAD VFORM } fin\end{bmatrix}$

The Subject Principles say that if there is a subject at all, then there is exactly one subject; and subjects can be realized in verbal projections only if a verbal projection is finite.

Relations There is only one relation in the core fragment, the append relation:

(77) The APPEND PRINCIPLE:

$$\forall \boxed{1}\ \forall \boxed{2}\ \forall \boxed{3} \left(\texttt{append}(\boxed{1}, \boxed{2}, \boxed{3}) \leftrightarrow \left(\begin{array}{l} \left(\boxed{1}[\mathit{elist}] \wedge \boxed{2} = \boxed{3} \right) \\ \vee \\ \exists \boxed{4}\ \exists \boxed{5}\ \exists \boxed{6} \left(\begin{array}{l} \boxed{1}\begin{bmatrix} \text{FIRST} & \boxed{4} \\ \text{REST} & \boxed{5} \end{bmatrix} \wedge \boxed{3}\begin{bmatrix} \text{FIRST} & \boxed{4} \\ \text{REST} & \boxed{6} \end{bmatrix} \\ \wedge\ \texttt{append}(\boxed{5}, \boxed{2}, \boxed{6}) \end{array} \right) \end{array} \right) \right)$$

Exercises

Exercise 40 *Take the sentence* Peter likes Mary*, which is licensed by the grammar specified in this section.*

First indicate the syntactic tree structure which the principles of the grammar determine for this sentence. Second, for each principle of the grammar indicate what it contributes to the properties of the sentence Peter likes Mary *as we find it in the denotation of the grammar. Do not forget to indicate those principles which do not have any effect on the properties of the sentence.*

GLOSSARY

3.2.1.2 Implementation of the Core Fragment

ABSTRACT

The TRALE implementation of the grammar specified in the previous section is our largest implemented grammar so far. We will discuss new types of differences between the implementation and the specification of the grammar, and we will introduce a number of new features of TRALE.

The first differences compared to our earlier grammar implementations occur in the system specifications at the very beginning of the theory file and concern the display behavior of the system in response to queries. We quote the beginning of `theory.pl`:[28]

```
% Multifile declarations.
:- multifile if/2.

% load phonology and tree output
:- [trale_home(tree_extensions)].

% specify signature file
signature(signature).
```

[28] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Core-Fragment/theory.pl

```
% grisu output specifications

hidden_feat(dtrs).          % shown by the tree
hidden_feat(daughters).     % shown by the tree

synsem <<< arg_st.
vform <<< aux.
vform <<< inv.
subj <<< comps.
liker <<< liked.
sayer <<< said.
giver <<< gift.
gift <<< given.
>>> phon.
```

The statements using the operators '<<<' and '>>>' specify the relative vertical order of attributes in the AVM descriptions of the GRiSU output. Attribute order specifications concern attributes which are appropriate to the same sort, because these attributes may occur in a relative order in one and the same AVM box. If nothing else is said, the system chooses alphabetical order, but this might not be the intuitive order which a human reader with certain linguistic conventions in mind might expect. For example '>>> phon' ("PHON precedes everything") ensures that PHON is always the topmost attribute mentioned in descriptions of signs, which is the conventional order in linguistics. Analogously, 'liker <<< liked' places the attribute which represents the agent role of *like* above the attribute for the patient role, which is again a more intuitive order for the human reader.

The Signature Let us think about the signature, before we turn to differences in the specification of grammatical principles. We do not quote the entire signature[29] of the implementation of the specified grammar here, but we refer the reader to the available file for downloading. Instead, we simply point out the differences and explain them.

The first difference is at the very beginning: The root sort of the grammar specification is called top, appealing to the idea that it is a supersort of all sorts in the sort hierarchy, and describes everything. While the name of the root sort is arbitrary in system independent signatures—we might even choose not to have a root sort at all, although in linguistic applications of the formalism this has not been done yet—, this is not the case in TRALE. TRALE requires first that a root sort exist, and, second, that it be named bot. The name bot comes from the culture of computer science and its tradition of computing with feature logics. In that tradition, bot(tom) is a well-chosen name for the most general sort, because it expresses the idea that we do not know anything yet about an entity, because we have not yet acquired any information about it.

[29] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Core-Fragment/signature

The second difference might look coincidental and just a matter of taste, but there is in fact more to it than just aesthetics. The grammar specification chooses one possible collection of symbols for the sorts and attributes for lists:

```
list
   elist
   nelist first:top  rest:list
```

As we know, any five sort symbols and two attribute symbols can be used to specify a sort hierarchy with appropriateness conditions for lists, as long as we put them in a configuration isomorphic to the one just quoted. It might then seem like an irrelevant coincidence that the implementation declares the list hierarchy with different but equally common symbols:

```
list
  ne_list hd:bot tl:list
  e_list
```

However, what we observe here is a system internal convention which in this case serves computational purposes. Lists are a very frequent data structure whose processing is made much more efficient by being treated separately. In order to recognize that certain sorts and attributes stand for lists, TRALE conventionally fixes them to be the ones just quoted (the sort which is appropriate for `hd` can be freely chosen, of course). Given the freedom of TRALE signatures, the user may of course choose any other set of symbols to use for lists, but this may lead to a loss in efficiency.

Similar considerations are responsible for another difference: In our specification we need sort symbols for all atomic phonological 'strings' in the fragment. According to linguistic conventions these are introduced as the most specific subsorts of a supersort `phonstring`. TRALE treats the phonological or orthographic strings as *extensional sorts* (or, to use TRALE's terminology, as *extensional types*). Extensional sorts do not have to be declared in the signature. For more information about their properties, please consult the sections Extensionality and a_/1 Atoms in the TR/ALE manual.[30] The system knows about the sorts for the admissible orthographic strings only by virtue of the lexical entries which define them for the system. Making the symbols for the orthographic strings clearly identifiable outside the signature provides another motivation for a distinguished syntax of lexical entries.

The last difference between the specified and the implemented signature which we find in appropriateness declarations on the level of signs is one which we have encountered before. The TRALE signature contains an additional statement for the appropriateness function, declaring the list-valued attribute DTRS appropriate for *sign*. We have already explained before that DTRS is needed for system internal representations. Future releases of TRALE probably no longer use it.

[30] `http://www.ale.cs.toronto.edu/docs/man/ale_trale_man/index.html`

An additional syntactic device may be observed at another place in the signature. The hierarchy under `synsem_none` appears as follows:

```
synsem_none
   none
   &synsem
```

The symbol '`&`' in front of the sort symbol `synsem` tells the system that `synsem` has more than one immediate supersort, and the second immediate supersort has been defined earlier in the signature. As you may confirm yourself by investigating the signature, `synsem` is also an immediate subsort of `bot`. In MoMo, the use of '`&`' is optional in these cases. A purely mathematical specification of the partial order of sorts, independent of any software requirements, does not need special notational conventions of this kind.

The next difference to be discussed is not really a mathematical difference, but rather a difference in the level of abstraction at which the specifications are presented. The HPSG grammar specification uses parametric sorts for lists such as *list(phonstring)* and *list(synsem)*. As explained in Section 2.5.2, we regard these signature specifications as abbreviations of ordinary sorts and principles of grammars which enforce the relevant restrictions on the members of lists by appropriate relations. TRALE specifications do not know parametric sorts either, but an appeal to abbreviatory conventions will of course not suffice here, we have to spell out everything. TRALE signatures use ordinary list sorts and their usual appropriateness specifications. Just as in the theoretical grammar specifications, the effect of parametric sorts is simulated by means of grammatical principles. Instead of relational expressions TRALE employs their counterpart, relational attachments and definite clause specifications of the necessary relations. When we discuss the theory file of the implementation we will see concrete examples of how parametric lists are encoded indirectly in TRALE. In any case, we conclude that the treatment of parametric sorts in TRALE parallels the resolution of the abbreviatory conventions which we appealed to when we explained them in our HPSG formalism.

The final difference between the two signature specifications occurs at the very end of the file and is the most profound of all: TRALE signatures do not contain relations. Relations do not belong to the description language. Relational expressions are replaced by relational attachments to descriptions following the operator `goal`, and they are defined as definite clauses. A small *caveat* will be added later when we focus on implementing the theory of the core fragment.

The Theory The implementation in the TRALE grammar of the principles of the HPSG specification is for the most part either straightforward or follows immediately from properties of TRALE which we have already discussed. The consequent of the Word Principle is expressed by stating its lexical entries in TRALE's syntax for lexical entries. The ID Schemata of the ID Principle are expressed as grammar rules. The Constituent Order Principle is implicit in the phrase structure component of the TRALE grammar. With the exception of the mod Principle, the Semantics Principle and the Structural Case Principle all other principles remain exactly as specified.

Before we discuss the differences which we may observe in these three principles, let us first take a look at a more difficult issue. As we have already vaguely indicated, there is an exception to the disallowing of relational expressions in TRALE descriptions. There is a special use of relations as *functional descriptions* in TRALE. These functional descriptions are properly integrated with the description language. They are used in our current fragment of English in the principles which simulate the parametric sort requirements for SUBJ lists, COMPS lists and ARG_ST lists.

The principles which simulate the intended effect of parametric sorts follow after the system settings at the beginning of the theory file. They say that the members of SUBJ lists are objects of sort *synsem*, the members of COMPS lists are objects of sort *synsem*, and the members of ARG_ST lists are objects of sort *synsem*. These principles make use of functional descriptions.

Consider what our theory of lists of *synsem* objects would look like in MoMo syntax. We need a unary relation (a relation of arity 1), `list_of_synsems`, a LIST-OF-SYNSEMS PRINCIPLE, and principles for each one of the lists.

```
VX(list_of_synsems(X) <*> (X:e_list;
                           ^Y(X:hd:synsem,X:tl:Y,list_of_synsems(Y)))).

val *> ^Y(subj:Y, list_of_synsems(Y)).
val *> ^Y(comps:Y, list_of_synsems(Y)).
word *> ^Y(arg_st:Y, list_of_synsems(Y)).
```

Now compare the HPSG formalization with its implementation in TRALE:

```
% Simulation of Parametric Sorts as Principles of Grammar

fun list_of_synsems(-).

list_of_synsems(X) if
   when( (X=(e_list;ne_list) ),
           und_list_of_synsems(X)).

und_list_of_synsems(X) if (X=e_list).
und_list_of_synsems(X) if (X=[(synsem)|Y]),list_of_synsems(Y).

% subj:list(synsem), comps:list(synsem)

val *> (subj:list_of_synsems,
        comps:list_of_synsems).

% arg_st:list(synsem)
word *> arg_st:list_of_synsems.
```

Let us focus first on the definite clauses for `und_list_of_synsems`. They bear a clear resemblance to the LIST-OF-SYNSEMS PRINCIPLE: The first definite clause says that an empty list is in `und_list_of_synsems`. The second clause says that `X` is in `und_list_of_synsems` if we can show that the first element of `X` is of sort *synsem*, and we can prove for the rest of the list, `Y`, that it is in `list_of_synsems`. What do we need `list_of_synsems` for? The distinction between `und_list_of_synsems` and `list_of_synsems` is simply motivated by computational needs. If we tried to prove that `list_of_synsems` holds for some list `X` that we know nothing about, we would run into an infinite proof in the recursion step in which we try to show something about the rest of the completely unknown list. The given definite clause specification prevents an infinite proof by defining the necessary predicate `list_of_synsems` in such a way that the system may enter into proving `und_list_of_synsems` only when it already knows something about the argument of `list_of_synsems`, `X`: The system needs to have already inferred whether the list is of sort *ne_list* or of sort *e_list*.

What we have seen so far is completely analogous of our earlier definite clause definition of `append` on the basis of a second predicate, `undelayed_append`, in Section 3.1.4. The new element in the use of the predicate `list_of_synsems` comes with the declaration

```
fun list_of_synsems(-).
```

which precedes it. This declaration says that `list_of_synsems` may be used as a *functional description*. Functional descriptions are a new and special kind of description provided by the syntax of TRALE. Recall that `list_of_synsems` is a one place predicate. The statement `(-)` at the end of the declaration says that the single argument of `list_of_synsems` is interpreted as standing at the position where the symbol `list_of_synsems` is written in a description. Take the principle which restricts ARG_ST values:

```
word *> arg_st:list_of_synsems.
```

`list_of_synsems` is used as a functional description here. It has one argument, but its argument is thought of as standing at the syntactic position where the symbol `list_of_synsems` is placed. In other words, we read this as saying that the value of ARG_ST is an argument of the predicate `list_of_synsems`. This means that we have to prove for ARG_ST values that `list_of_synsems` holds for them, which is exactly what we want. Functional descriptions are thus a way to use relations as descriptions in TRALE. They may occur at the same syntactic positions where sort symbols are permitted. Crucially, for a functional use of a predicate we have to declare which of its arguments is the implicit, functional argument.

Suppose we want to use an `append` predicate functionally to say that ARG_ST values consist of a concatenation of two lists, `X` and `Y`, under certain circumstances. In functional notation we will write this as

```
... arg_st:append(X,Y).
```

Using the functional notation in this way presupposes that we declare the third argument of `append` to be the implicit, functional argument. To make this declaration we have to write

```
fun append(+,+,-).
```

before the definite clauses which define the `append` predicate.

General Observations on TRALE's Description Language In general, TRALE's description language is significantly less comprehensive than the description language for HPSG. With one exception (which occurs in the implementation of the first part of our STRUCTURAL CASE PRINCIPLE), there is no explicit quantification.[31] Variables in descriptions can be thought of as being implicitly bound by existential quantification. The implication symbol, `*>`, is in fact an operator of TRALE which is preceded and succeeded by a TRALE description. TRALE descriptions, finally, only know about conjunction and disjunction, plus sort symbols, attribute symbols and variables. These symbols can be combined in the way we are used to in order to form more complex descriptions, as exemplified in the descriptions in our grammar implementations.

What is missing, then? There cannot be another implication, `*>`, in the antecedent or in the consequent of a TRALE constraint. There cannot be any relational attachments in antecedents, since relational attachments are attached to complete constraints only. Functional descriptions are not allowed in antecedents. In effect, this means that the predicates which we use in TRALE to simulate the relations of HPSG cannot occur in the scope of negation. It follows that whenever an HPSG principle uses a relation in the scope of negation, it has to be reformulated considerably in its computational implementation. As one may confirm by investigating the grammar of English of Pollard and Sag 1994, this restriction affects a considerable number of important principles of HPSG grammars.

Similarly, TRALE does not have the unary negation symbol, $\sim$, from the complete description language of HPSG. However, since we have implication in the form of the operator, `*>`, which brings with it implicit negation of the antecedent, we can always reformulate expressions using negation into an equivalent expression using conjunction, disjunction, and implication. An example of an equivalent reformulation can be observed in the first disjunct of the consequent of the SEMANTICS PRINCIPLE. The grammar specification says that under certain conditions, the DAUGHTERS value of a phrase may not be of sort *ha_struc*. The implementation of the principle reformulates this by a disjunctive enumeration of the positive possibilities, given the appropriateness specifications of the signature: the DAUGHTERS values in question may be of sort *hs_struc*, *hc_struc*, or *sai_struc*. A similar recoding of a negation occurs in the antecedent of the MOD PRINCIPLE. In general, equivalent reformulations might be more difficult in more complex descriptions, but they are always possible.

[31] The infix operator, depicted as the hat symbol, is, in the STRUCTURAL CASE PRINCIPLE, an existential quantifier which is interpreted non-classically. In standard notation for the quantifier we can understand it as follows: $\exists x_1 \ldots \exists x_n(p) \rightarrow q$ is read as $\exists x_1 \ldots \exists x_n(p) \rightarrow \exists x_1 \ldots \exists x_n(p \wedge q)$. This is of course not valid in classical logic. TRALE provides this option because it is useful for linguistic grammar specifications.

Exercises

Exercise 41 *Why does the system answer the query* `rec[you,walk]` *with* `no`*?*

Exercise 42 *The functional (or non lexical) preposition* to *has a very small lexical entry. In AVM syntax it is simply*

$$\begin{bmatrix} \text{PHON } \langle to \rangle \\ \text{SYNS LOC CAT HEAD } \begin{bmatrix} prep \\ \text{PFORM } to \end{bmatrix} \end{bmatrix},$$

and its TRALE counterpart is a direct translation of this description into TRALE's notation for lexical entries. If you query TRALE for to *with* `lex to`*, however, you get a much more precise description of functional* to *in the denotation of our grammar for an answer (besides a description of lexical* to*, which we ignore here).*

Name the parts of the grammar which the compiler used to infer the more precise description of non lexical `to`*, which it apparently has internally available at run time for the purposes of parsing.*

Exercise 43 *Why do the* `append` *relation and the* Append Principle *not occur in the implementation of our core grammar specification, although we obviously need it in a complete theoretical specification of our core fragment of English?*

GLOSSARY

3.2.2 Lexical Generalizations: Fragment II

Abstract

> *We will extend the core fragment by adding lexical generalizations in the form of lexical rules. With lexical rules we can express systematic relationships between words which have so far been described as unrelated words in separate lexical entries. The domain of our lexical rules will be verb forms.*

The goal of this section is a radical simplification of the verbal lexicon of the core fragment. Despite the simplification, we want to vastly increase the empirical coverage of the verbal domain of the grammar. Idealizing the picture somewhat, we want to be able to specify a single lexical entry for each verb, in which we pair up a phonology and a syntactic argument structure with a single referential meaning. Then we state generalizations over the verb classes to which the verbs belong. The generalizations should predict the existence and form of the different extant inflected forms (which possibly exhibit some additional functional semantic property such as tense) together with the syntactic distribution of the various verb forms.

Taking the verb *walk* as an example, we want to describe the following data pattern, where we suppose exactly one base lexical entry for *walk*:

(78) a. Mary walks.

b. They walk.

c. *Mary walk.

d. *They walks.

e. Mary walked.

f. Mary has walked.

g. Mary has been walking.

h. Mary will be walking.

i. Mary will have been walking.

Similarly, we want to capture simple facts about the passive in English. The same base lexical entries as the ones necessary to derive the data in (78) should be used to state generalizations which predict the data in (79):

(79) a. Mary is liked by Peter.

b. It is given to Peter by Mary.

c. That Peter walks is said by her.

d. It has been given to Peter.

e. *Mary is walked.

GLOSSARY

3.2.2.1 Specification of Fragment II

ABSTRACT

We will specify our extension of the core fragment by stating all necessary extensions of the signature, giving exemplary lexical entries, and stating the lexical generalizations in an informal but intuitive notation for lexical rules.

The necessary additions to the signature of the core fragment consist in new subsorts of *phonstring* to accommodate some new words and their forms, additional semantic relations for a simple representation of tense forms under *psoa*, and a new subsort of *pform* to distinguish the new non lexical preposition *by* needed for the analysis of the passive from other prepositions. We do not repeat the entire signature, but state only the additions to the signature by indicating where the new sorts belong in the sort hierarchy of the core fragment. In comparing the old signature with the additions, it should be immediately obvious how the appropriateness specifications extend to the larger sort hierarchy. The new relation symbols given at the end of the signature are for the treatment of verbal inflexion. All parts of the signature of the core fragment are retained.

```
type_hierarchy
top
   pform
      non_lexical
         by
   cont
      psoa
         tense_rel soa_arg:psoa
            future_rel
            perfect_rel
            present_rel
            past_rel
            cont_rel
   phonstring
      #rained#
      #walked#
      #liked#
      #said#
      #gave#
      #given#
      #have#
      #has#
      #had#
      #be#
      #is#
      #are#
      #was#
      #were#
      #been#
      #by#
relations
  psp_inflect/2
  prp_inflect/2
  past_inflect/3
  fin_inflect/3
.
```

On the basis of our new signature, we can immediately turn to the lexical entries of the grammar. There are no genuinely new syntactic structures in our extended fragment of English. All additions concern new kinds of words with new properties, and their relationship to words which were in the core fragment already.

In the non-verbal domain we keep all lexical entries that we had before.[32] We eliminate

[32]The discussion of one significant modification in one particular previous lexical entry comes up later

all verbal lexical entries which do not describe base forms, i.e. verbs which do not have the VFORM value *base*. To illustrate this we repeat the base forms of two verbs that were included in the core fragment, a base form description of *walk* and a base form description of *like*. Directly after the descriptions of *walk* and *like* we see the lexical entries of two new auxiliaries, the perfect auxiliary *have* and the passive auxiliary *be*. The two auxiliaries are raising verbs, they identify the first element of their ARG_ST lists with the subject of the unsaturated verbal projection which they select. At the same time they are CONTENT raisers: Their CONTENT value is simply identical with the CONTENT value of the selected verbal projection.

Verbs

(80) Base lexical entry of the verb *walk*:

$$\begin{bmatrix} \textit{word} \\ \text{PHON} & \langle \#walk\# \rangle \\ \text{SYNS} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{verb} \\ \text{PRED} & \textit{plus} \\ \text{VFORM} & \textit{base} \\ \text{INV} & \textit{minus} \\ \text{AUX} & \textit{minus} \end{bmatrix} \\ \text{VAL} & \begin{bmatrix} \text{SUBJ} & \langle\rangle \\ \text{COMPS} & \langle\rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT} & \begin{bmatrix} \textit{walk\_rel} \\ \text{WALKER} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{ARG\_ST} & \left\langle \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \textit{noun} \\ \text{VAL} & \begin{bmatrix} \text{SUBJ} & \langle\rangle \\ \text{COMPS} & \langle\rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT} & \begin{bmatrix} \text{INDEX} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

(81) Base lexical entry of the verb *like*:

$$\begin{bmatrix} \textit{word} \\ \text{PHON} & \langle \#like\# \rangle \\ \text{SYNS} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{verb} \\ \text{PRED} & \textit{plus} \\ \text{VFORM} & \textit{base} \\ \text{INV} & \textit{minus} \\ \text{AUX} & \textit{minus} \end{bmatrix} \\ \text{VAL} & \begin{bmatrix} \text{SUBJ} & \langle\rangle \\ \text{COMPS} & \langle\rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT} & \begin{bmatrix} \textit{like\_rel} \\ \text{LIKER} & \boxed{1} \\ \text{LIKED} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{ARG\_ST} & \left\langle \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \textit{noun} \\ \text{VAL} & \begin{bmatrix} \text{SUBJ} & \langle\rangle \\ \text{COMPS} & \langle\rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT} & \begin{bmatrix} \text{INDEX} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \textit{noun} \\ \text{VAL} & \begin{bmatrix} \text{SUBJ} & \langle\rangle \\ \text{COMPS} & \langle\rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT} & \begin{bmatrix} \text{INDEX} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

in an exercise.

(82) Lexical entry of the perfect auxiliary verb *have*:

$$\begin{bmatrix} \text{PHON} & \langle \#have\# \rangle \\ \text{SYNS} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} verb & \\ \text{PRED} & plus \\ \text{VFORM} & base \\ \text{AUX} & plus \end{bmatrix} \end{bmatrix} \\ \text{CONT} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{ARG\_ST} & \left\langle \boxed{1}, \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} verb & \\ \text{VFORM} & psp \end{bmatrix} \\ \text{VAL} & \begin{bmatrix} \text{SUBJ} & \langle \boxed{1} \rangle \\ \text{COMPS} & \langle \rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT} & \boxed{2} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

(83) Lexical entry of the passive auxiliary verb *be*:

$$\begin{bmatrix} \text{PHON} & \langle \#be\# \rangle \\ \text{SYNS} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} verb & \\ \text{PRED} & plus \\ \text{VFORM} & base \\ \text{AUX} & plus \end{bmatrix} \end{bmatrix} \\ \text{CONT} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{ARG\_ST} & \left\langle \boxed{1}, \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} verb & \\ \text{VFORM} & pas \end{bmatrix} \\ \text{VAL} & \begin{bmatrix} \text{SUBJ} & \langle \boxed{1} \rangle \\ \text{COMPS} & \langle \rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT} & \boxed{2} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

Prepositions

(84) Lexical entry of the (non lexical) preposition *by*:

$$\begin{bmatrix} \text{PHON} & \langle \#by\# \rangle \\ \text{SYNS LOC CAT HEAD} & \begin{bmatrix} prep & \\ \text{PFORM} & by \end{bmatrix} \end{bmatrix}$$

Lexical rules In our specification of the grammar fragment, we state lexical rules in a *lingua franca* for lexical rules which has emerged in the HPSG community over the past decade. The specification language is fairly informal. The general pattern of these lexical rules is depicted in (85):

(85)
$$\begin{bmatrix} word & \\ \text{SYNS LOC} & \begin{bmatrix} \text{CAT HEAD} & \begin{bmatrix} head & \\ \text{CASE} & nom \end{bmatrix} \\ \text{CONT} & \delta_1 \end{bmatrix} \end{bmatrix} \quad \longmapsto \quad \begin{bmatrix} word & \\ \text{SYNS LOC} & \begin{bmatrix} \text{CAT HEAD} & \begin{bmatrix} noun & \\ \text{CASE} & acc \end{bmatrix} \\ \text{CONT} & \delta_2 \end{bmatrix} \end{bmatrix}$$

Note that the notation makes crucial use of a new symbol, $\longmapsto$, which is not defined in any of our HPSG description languages. To the left of the arrow we have something that looks like a description of a word, and to the right of the arrow we see a second apparent description of a word.

It is clear that in order to give the informal notation meaning in HPSG, we will ultimately have to say more about how we want to interpret the new notation. Since the reserved symbol, $\longmapsto$, splits what could otherwise be considered a description in two parts, it is not even obvious how exactly to interpret tags which occur on both sides of the new arrow symbol. Remember that tags are variables, and variables are quantified over separately in separate descriptions. In a sense, the intention behind writing the same tag on both sides of the arrow seems to be to convey the idea that *something* is supposed to be the same in two otherwise distinct words. What that something is, we will have to clarify when we explain the lexical rule notation in more detail. For the time being, we will just explain the intended effect of each lexical rule with a loose description in natural language. This will suffice in order to understand the coverage of our fragment. For a full understanding of the linguistic theory behind our lexical generalizations, however, it is insufficient.

Past Participle Lexical Rule The PSP LEXICAL RULE says that for each verb with the VFORM value *base* there is another verb in the language which looks just the same, except for the CONTENT value and the phonology. The CONTENT value of the base form of the verb becomes embedded under a *perfect_rel(ation)*, and the phonology exhibits the inflexion of a past participle. The meaning of the relation `psp_inflect` must be defined in such a way that the phonological form of the base lexical entry stands in the `psp_inflect` relation with the phonological form of its past participle form.

(86) The PSP LEXICAL RULE:

$$\left[\begin{array}{ll}\text{PHON} & \langle \boxed{1} \rangle \\ \text{SYNS LOC} & \left[\begin{array}{ll}\text{CAT HEAD} & \left[\begin{array}{ll} \textit{verb} \\ \text{VFORM} & \textit{base}\end{array}\right] \\ \text{CONT} & \boxed{3}\end{array}\right]\end{array}\right]$$

$$\longmapsto \quad \left[\begin{array}{ll}\text{PHON} & \langle \boxed{2} \rangle \\ \text{SYNS LOC} & \left[\begin{array}{ll}\text{CAT} & \left[\text{HEAD} \left[\begin{array}{ll} \textit{verb} \\ \text{VFORM} & \textit{psp}\end{array}\right]\right] \\ \text{CONT} & \left[\begin{array}{ll} \textit{perfect\_rel} \\ \text{SOA\_ARG} & \boxed{3}\end{array}\right]\end{array}\right]\end{array}\right] \wedge \texttt{psp\_inflect}(\boxed{1}, \boxed{2})$$

Present Participle Lexical Rule The PRP LEXICAL RULE is formulated in analogy to the PSP LEXICAL RULE. The CONTENT value of the base form becomes embedded under *cont(inuous)_rel(ation)*, and the phonology of the present participle form stands in the `prp_inflect` relation with the phonology of the corresponding base form.

(87) The PRP LEXICAL RULE:

$$\begin{bmatrix} \text{PHON } \langle \boxed{1} \rangle \\ \text{SYNS LOC} \begin{bmatrix} \text{CAT HEAD} \begin{bmatrix} verb \\ \text{VFORM } base \end{bmatrix} \\ \text{CONT } \boxed{3} \end{bmatrix} \end{bmatrix}$$

$$\longmapsto \quad \begin{bmatrix} \text{PHON } \langle \boxed{2} \rangle \\ \text{SYNS LOC} \begin{bmatrix} \text{CAT} \begin{bmatrix} \text{HEAD} \begin{bmatrix} verb \\ \text{VFORM } prp \end{bmatrix} \end{bmatrix} \\ \text{CONT} \begin{bmatrix} cont\_rel \\ \text{SOA\_ARG } \boxed{3} \end{bmatrix} \end{bmatrix} \end{bmatrix} \wedge \texttt{prp\_inflect}(\boxed{1}, \boxed{2})$$

Finitivization Lexical Rule The FIN LEXICAL RULE is not quite as simple as the previous two lexical rules, because it needs to take subject verb agreement into account. One base form of a verb usually corresponds to more than one finite verb form, since the phonology of verbs which have third person singular or sentential subjects is often different from the phonology of verbs which agree with subjects that are not third person singular. The relation `fin_inflect` is a ternary relation which relates the phonology of the base form to the phonology of the corresponding finite verb form depending on the CONTENT value of the subject. Note that the CONTENT value of corresponding base and finite verb forms are supposed to look the same, since no changes are indicated

(88) The FIN LEXICAL RULE:

$$\begin{bmatrix} \text{PHON } \langle \boxed{1} \rangle \\ \text{SYNS LOC CAT HEAD} \begin{bmatrix} verb \\ \text{VFORM } base \end{bmatrix} \end{bmatrix}$$

$$\longmapsto \quad \begin{bmatrix} \text{PHON } \langle \boxed{2} \rangle \\ \text{SYNS LOC CAT} \begin{bmatrix} \text{HEAD} \begin{bmatrix} verb \\ \text{VFORM } fin \end{bmatrix} \\ \text{VAL SUBJ} \left\langle \begin{bmatrix} \text{LOC CONT } \boxed{3} \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix} \wedge \texttt{fin\_inflect}(\boxed{1}, \boxed{2}, \boxed{3})$$

Past Tense Lexical Rule Just as in the previous lexical rule, the PAST TENSE LEXICAL RULE needs to take agreement facts into account, because there are verbs in English whose phonological form is not uniform in past tense. The auxiliary *be* distinguishes between *was* for first and third person singular (and sentential subjects) and *were* for all other subject agreement possibilities. For that reason the relation `past_inflect` refers to the CONTENT value of the subject. The CONTENT values of the past tense forms look just like the CONTENT values of the corresponding base forms, with the exception that they are embedded under *past_rel*.

(89) The PAST TENSE LEXICAL RULE:

$$\begin{bmatrix} \text{PHON } \langle \boxed{1} \rangle \\ \text{SYNS LOC} \begin{bmatrix} \text{CAT HEAD} \begin{bmatrix} \textit{verb} \\ \text{VFORM } \textit{base} \end{bmatrix} \\ \text{CONT } \boxed{3} \end{bmatrix} \end{bmatrix}$$

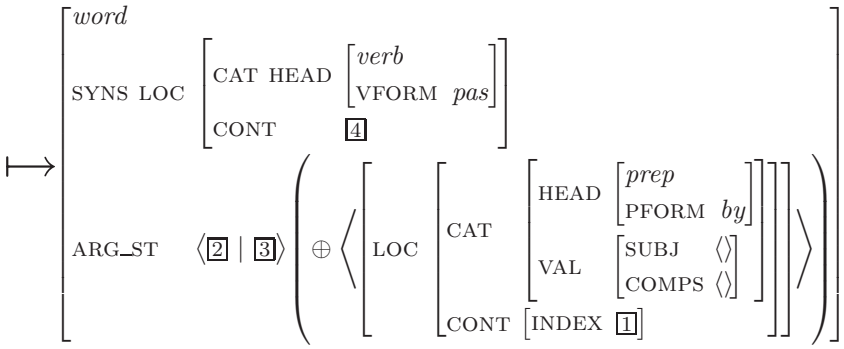
$$\longmapsto \quad \begin{bmatrix} \text{PHON } \langle \boxed{2} \rangle \\ \text{SYNS LOC} \begin{bmatrix} \text{CAT} \begin{bmatrix} \text{HEAD} \begin{bmatrix} \textit{verb} \\ \text{VFORM } \textit{fin} \end{bmatrix} \\ \text{VAL} \begin{bmatrix} \text{SUBJ} \left\langle \begin{bmatrix} \text{LOC CONT } \boxed{4} \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT} \begin{bmatrix} \textit{past\_rel} \\ \text{SOA\_ARG } \boxed{3} \end{bmatrix} \end{bmatrix} \end{bmatrix} \wedge \texttt{past\_inflect}(\boxed{1}, \boxed{2}, \boxed{4})$$

Passive Lexical Rule The PASSIVE LEXICAL RULE is the most interesting lexical rule in our small collection. It relates the past participle forms to corresponding passive forms. Referring to the phonology of the two verb classes is unnecessary, since it is the same. Since auxiliaries lack passive forms, the input description (the description to the left of the lexical rule arrow, $\longmapsto$) says that the words affected by this rule must be non-auxiliaries, i.e., they have the AUX value *minus*. Since the past participle forms of verbs embed their referential content under the SOA_ARG attribute of their *perfect_rel* valued CONTENT, the lexical rule says that the CONTENT of the passive forms looks like the SOA_ARG value of the corresponding past participle form.

Crucial conditions are expressed as statements about the relationship between the ARG_ST lists of the two kinds of participles. Only past participle verbs with at least two elements on the ARG_ST list, and whose subject is nominal, have a corresponding passive participle. The subject of the past participle corresponds to an optional *by*-PP as the last element on the ARG_ST list of the passive participle. The optionality is expressed by the round brackets around the symbol $\oplus$ and the succeeding singleton list containing the SYNSEM value of the *by*-PP. The symbol $\oplus$ is a popular functional infix notation for the `append` relation.

(90) The PASSIVE LEXICAL RULE:

$$\begin{bmatrix} \textit{word} \\ \text{SYNS LOC} \begin{bmatrix} \text{CAT HEAD} \begin{bmatrix} \textit{verb} \\ \text{VFORM } \textit{psp} \\ \text{AUX } \textit{minus} \end{bmatrix} \\ \text{CONT} \begin{bmatrix} \text{SOA\_ARG } \boxed{4} \end{bmatrix} \end{bmatrix} \\ \text{ARG\_ST} \left\langle \begin{bmatrix} \text{LOC} \begin{bmatrix} \text{CAT} \begin{bmatrix} \text{HEAD } \textit{noun} \\ \text{VAL} \begin{bmatrix} \text{SUBJ } \langle\rangle \\ \text{COMPS } \langle\rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT} \begin{bmatrix} \text{INDEX } \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}, \boxed{2} \mid \boxed{3} \right\rangle \end{bmatrix}$$



With the lexical rules we have expressed lexical generalizations as relationships between word classes. We started with the description of a base form of verbs via conventional lexical entries and then made statements about how other admissible verb forms differ from them. The lexical rules are meant to be a characterization of additional words which we want to have in the denotation of our grammar. At this point, it is still an open question how we can technically achieve the intended effect.

The lexical rules are clearly not principles of grammar. As we know from our discussion of the formalism, additional principles can never lead to the admission of new feature structures in the model of the grammar: Principles of grammar are always restrictions on the kinds of structures which are permitted. Similarly, if we have a WORD PRINCIPLE in the grammar, every feature structure of sort *word* must be licensed by a disjunct in its consequent. There is no logical alternative. Thus we seem to have two options: Either lexical rules can somehow be interpreted as (an) additional disjunct(s) in the consequent of the WORD PRINCIPLE, or we give up the WORD PRINCIPLE as we know it and think of other, possibly extra-logical ways of licensing the words in the denotation of our grammar. In the next section we will see that both options have been pursued in the literature.

GLOSSARY

3.2.2.2 Theories of Lexical Rules in HPSG

ABSTRACT

We will introduce two ways of interpreting lexical rules.

The descriptions of words in lexical entries should ideally be left very general. For example, the lexical entry of a verb might only mention the phonology of the verb stem, its part of speech, the lexical semantics of the word, and its syntactic argument structure. Such an impoverished lexical entry in the consequent of the WORD PRINCIPLE would of course license many structural variants of the verb in question which should not be grammatical. A general lexical entry presupposes other principles in the grammar which would exclude the ungrammatical variants. As [Meurers, 2000, pp. 109ff.] explains, *lexical principles* can be used to further constrain the admissible words in the denotation of lexical entries. Lexical principles are principles which constrain the structure of words by generalizing over word classes. In our example of a very general hypothetical lexical entry of a verb, lexical

principles might be formulated which ensure that for all finite verbs in the denotation of the lexical entry (and of all other lexical entries that describe finite verbs), the case of its nominal subject is *nominative*. Our ARGUMENT REALIZATION PRINCIPLE, which maps ARG_ST lists to SUBJ and COMPS lists, is another lexical principle which restricts the possible shape of entire word classes; and so is the FUNCTIONAL PREPOSITION PRINCIPLE.

Lexical principles constrain the form of all words in a word class which exhibits certain properties. The relevant word class is characterized in the antecedent of each lexical principle. Lexical rules express a different type of observation: They are generalizations which link the existence of one verb class to the existence of another verb class. Their different nature begs the question of how to formalize them.

Two views of lexical rules can be distinguished in the HPSG literature. The first view holds that lexical rules express a relationship between lexical entries, and this is the view of [Pollard and Sag, 1994]. According to this view, a linguist specifies a finite set of base lexical entries and a finite set of binary relations between expressions of the logical language in which the base lexical entries are expressed. The lexicon is then defined as the closure of the lexical entries under the defined set of binary relations, the lexical rules. We may imagine the closure operation as taking the base lexical entries and generating all lexical entries which can be derived from them by application of lexical rules. This gives us a new set of lexical entries, which we again take as the input to the lexical rules, because in the first round we might have generated new lexical entries which may again be able to undergo lexical rule application. This process may be reiterated as long as we find new input lexical entries after lexical rule application.

Lexical rules of this kind are also known as meta-level lexical rules (MLRs) because they are not defined in the logical language in which the grammar is specified, but in a meta-language which talks about the language of the grammar. Therefore this view of lexical rules places them outside of the HPSG formalism which we discussed. The new arrow symbol of our notation for lexical rules, $\longmapsto$, may be interpreted as belonging to the meta-language.

The MLR perspective still awaits a formalization in HPSG, and for the time being it remains unclear whether an elegant and general formalization is compatible with our current understanding of the mathematics of HPSG. The closure of lexical entries under MLRs might yield an infinite lexicon, which either necessitates an extension of the logical languages of the formalism to infinitary languages in order to formulate a WORD PRINCIPLE with infinitely many lexical entries, or a modification of the notion of a grammar. For example we could define a grammar as a triple consisting of a signature Σ, a set of Σ descriptions (the theory of the grammar), and a (potentially infinite) set of lexical entries (the lexicon) which is the closure of the base lexical entries under the lexical rules. A set of relational abstract feature structures would be admitted by the grammar if and only if each node in each of its feature structures satisfied each description of the theory, and each node of sort *word* satisfied at least one lexical entry.

In addition to providing a big picture such as the one we have just sketched, we need to be a lot more precise about the specification language of MLRs. In our examples of lexical rules in the previous section, we always gave an explanation of what was meant

by mentioning certain properties of words in the descriptions to the left and to the right of the arrow, and occasionally we also commented on what was meant by *not* mentioning certain properties of the words in the descriptions. A complete theory of MLRs would have to rigorously specify the meta-language of lexical rules, which would include a precise characterization of what it means for a lexical entry to *match* the input description of a lexical rule, and in which way exactly it is transformed into a new description. We will not pursue these difficult issues any further here.

The second view of lexical rules holds that lexical rules express relations between word entities. A lexical rule says that if the model of a grammar contains (feature structure representations of) words of a certain structure, then it must also contain corresponding words of a different given structure. This interpretation of lexical rules is known as description-level lexical rules (DLRs), because it puts lexical rules on the same level with all other principles of the grammar: Lexical rules as DLRs become part of (descriptions in) the theory of the grammar. A formalization of the DLR approach to lexical rules is put forth and discussed by [Meurers, 2000, pp. 123ff.]. The basic idea is to introduce entities of sort *lexical_rule* into the grammar with a *word*-valued IN attribute (the input of the lexical rule) and a *word*-valued OUT attribute (the output of the lexical rule). In the ordinary description language of all principles the linguist describes which properties the input words share with the output words, and where they differ. The WORD PRINCIPLE is extended by exactly one disjunct which adds the possibility that a word in a model of the grammar may also be described by the OUT value of a *lexical_rule* entity.[33] To enhance the practical usability of DLRs, Meurers provides a lexical rule specification language which essentially takes lexical rules as they are usually written by linguists and transforms them into descriptions of *lexical_rule* entities. By and large we can retain the notation of lexical rules which we have used above, plus we obtain a few new symbols for eliminating certain ambiguities which may occur in the traditional lexical rule notation. The lexical rule specifications of this sort are then interpreted as an abbreviatory notation for real, much bigger HPSG descriptions of *lexical_rule* objects.

The advantage of this explanation of lexical rules is that it has been fully formalized, we do not have to go beyond the description language and formalism which we already know, and, in particular, finite descriptions suffice to characterize the lexicon of a natural language with infinitely many (abstract feature structure representations of) words. Just as before with MLRs, we will refrain from going into technical details. If interested, the reader may read up on the topic in [Meurers, 2000], which contains a very readable explanation of lexical rules and the subject of lexical generalizations.

As [Meurers, 2000] argues at length, DLRs achieve exactly the effect which linguists have in mind when they specify lexical rules, and their formalization helps to clarify open questions about ambiguities in the notation which linguists use in their notation of lexical rules. [Meurers and Minnen, 1997] describes a computational implementation of DLRs.

[33] To make this specification possible, the *lexical_rule* entity must be a component of the word that is the output of the lexical rule. For the technical details, see [Meurers, 2000, p. 124].

Exercises

We will study the sentence *Was it given to Peter by Mary.*

Exercise 44 *How are the words* was *and* given *in this sentence licensed by Fragment II? More specifically, which lexical entries and which lexical rules are involved in licensing these two words in our sentence?*

Exercise 45 *Indicate the syntactic structure of the given sentence as it is licensed by Fragment II. A simple bracketing of the sentence depicting the syntactic constituent structure will suffice.*

Exercise 46 *Describe the* CONTENT *value of the sentence. Explain in which way the grammar licenses this* CONTENT *value, taking into account the lexical entry of the verb* give*, the appropriate lexical entry of* be*, the relevant lexical rules and the* SEMANTICS PRINCIPLE.

GLOSSARY

3.2.2.3 Implementation of Fragment II

ABSTRACT

Through implementing the extension of the core fragment, we will see how meta-level lexical rules are used in TRALE.

We have presented two ways of interpreting our lexical rule specifications, which are the only innovations introduced by our extension of the core fragment. The question now is which kind of lexical rules are supported by TRALE, MLRs or DLRs.

The answer to this question is simple: TRALE knows two variants of DLRs, which we will call DLR-1s and DLR-2s. For DLR-1 lexical rules it imports the syntax and functionality of the ALE system with only minor modifications in syntax. For DLR-2 lexical rules it imports the lexical rule compiler, originally written for the ConTroll system (see Section 1.1, *Historical Overview*), which is based on the theory of [Meurers and Minnen, 1997]. TRALE users are free to choose whatever they prefer, or even to use both types of lexical rules in a single grammar. A description of the DLR-1 syntax can be found in the section on *ALE Lexical Rules* of the *User's Guide*;[34] the lexical rule compiler for DLR-2s and the syntax for DLR-2s are explained in the same manual in Section *TRALE Lexical Rule Compiler*.

The reader may ask why the implementation of DLRs uses devices which might at first look as if they did not belong to the feature logic after we said above that DLRs are defined within the feature logic of HPSG. The reason is twofold:

[34] `http://www.ale.cs.toronto.edu/docs/man/ale_trale_man/index.html`

Firstly, as we know, there is a crucial difference between the general and very powerful HPSG formalism and an implementation platform for HPSG grammars such as the TRALE system, which implements parts of the HPSG description language and adds a number of important task specific devices with their own underlying mathematics. The implementation of DLRs is just one more of these devices which may use mechanisms *outside* of the implementation of the feature logic. Just like with an implementation tool such as phrase structure rules, if necessary we are able to translate them back into HPSG's feature logic. Here we will not investigate how such a translation can be achieved.

Secondly, TRALE's LR implementation in terms of DLR-1s is not as *general* as would be necessary to count as a complete implementation of a linguistic theory of description-level lexical rules. The limitations are easy to see in practice. Recall that we said that LRs may easily lead to an infinite number of words in the grammar. Of course, no computer system can handle an infinite number of lexical representations. TRALE's answer to this problem (which is taken over from ALE) is straightforward: It cuts off the recursive application of lexical rules to their output at a (user definable) point, thus excluding an infinite number of words. If nothing is changed by the user, the system will apply lexical rules only up to the second generation of descendants of the base lexical entries, as they are derived by lexical rule application. It immediately follows that if a linguist envisages an infinite lexicon by means of her lexical rule specifications, TRALE's DLR-1s will not be able provide an adequate implementation of the theory. If an infinite number of words is supposed to be derived from a finite set of words in the denotation of base lexical entries by stating lexical rules, it might be worthwhile to consider an implementation in terms of DLR-2s. Of course, this presupposes that the concept of DLR-2s is compatible with the theoretical intentions of the linguist.

Our TRALE implementation (signature[35], theory.pl[36]) of the lexical rules of the specification of Fragment II chooses DLR-1s. The implementation of the PSP LEXICAL RULE then appears as follows:

```
% PSP Lexical Rule

psp_lex_rule ##
(word,
 synsem:loc:(cat:head:(vform:base,
                       aux:Aux,
                       pred:Pred),
             cont:Cont),
 arg_st:Arg)
 **>
(synsem:loc:(cat:head:(vform:psp,
                       aux:Aux,
                       pred:Pred),
```

[35] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Fragment-with-Lex-Gens/signature
[36] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Fragment-with-Lex-Gens/theory.pl

```
              cont:(perfect_rel,
                    soa_arg:Cont)),
 arg_st:Arg)
morphs
be becomes been,
give becomes given,
have becomes had,
say becomes said,
(X,[e]) becomes (X,ed),
X becomes (X,ed).
```

Note that in contrast to the linguistic conventions which we followed in the theoretical specification of the lexical rules, we now have to mention all parts of the description which should be carried over from the input descriptions of the lexical rule to the descriptions that are generated as output. Carrying over parts of the description from the input to the output is done by variables. In order to obtain reasonable output, the DLR-1 implementation of the PSP LEXICAL RULE uses the variables `Aux`, `Pred`, and `Arg` in order to transport the descriptions of the AUX value, of the PRED value, and of the argument structure list from the input to the output. In addition, the content description is modified the same way as in the specification.[37]

What about the phonology of the words and their relational description with `psp_inflect/2` of the specification? Since the phonology of signs is treated in a special way in TRALE anyway, it is not surprising that DLR-1s provide their own mechanism for specifying the orthographical changes from the input descriptions to the output descriptions. After the operator `morphs` there comes a list of transformations from the input to the output: Idiosyncratic changes for certain words are listed first, followed by general patterns which apply to non-idiosyncratic cases. For each lexical entry the compiler executes the first applicable clause it finds without considering the remaining clauses. For details please consult the section on ALE lexical rules in the *User's Guide*.

The DLR-1 implementation of the PRP LEXICAL RULE is entirely parallel to the PSP LEXICAL RULE. The two lexical rules in which verbal inflexion depends on agreement phenomena, the FINITIVIZATION LEXICAL RULE and the PAST TENSE LEXICAL RULE, form another group in terms of their implementation. Recall that in the specification of the lexical rules we introduced relations between the phonology of the input of the rule and the phonology of the output of the rule. These relations had the CONTENT value of the subject of the output as another argument in order to make the phonology of the output dependent on agreement facts. To simulate this effect with the `morphs` operator in an DLR-1 implementation we have to split these lexical rules into two meta-level lexical rules, one for each agreement pattern. Here is the implementation of the FINITIVIZATION LEXICAL RULE as two DLR-1s:

[37] In a DLR-2 specification the same lexical rule would look different and much closer to the original specification!

```
% 3rd_sing_fin_lex_rule
third_sing_fin_lex_rule ##
(word,
 synsem:loc:(cat:head:(vform:base,
                       aux:Aux,
                       pred:Pred),
             cont:Cont),
 arg_st:Arg)
 **>
(synsem:loc:(cat:(head:(vform:fin,
                        aux:Aux,
                        pred:Pred),
                  val:subj:[(loc:cont:(psoa;index:(pers:third,
                                                   num:sg)))]),
             cont:(present_rel,
                  soa_arg:Cont)),
 arg_st:Arg)
morphs
be becomes is,
have becomes has,
X becomes (X,s).

% non_3rd_sing_fin_lex_rule
non_third_sing_fin_lex_rule ##
(word,
 synsem:loc:(cat:head:(vform:base,
                       aux:Aux,
                       pred:Pred),
             cont:Cont),
 arg_st:Arg)
 **>
(synsem:loc:(cat:(head:(vform:fin,
                        aux:Aux,
                        pred:Pred),
                  val:subj:[(loc:cont:index:(num:pl;
                                             (pers:(first;second),
                                             num:sg)))]),
             cont:(present_rel,
                   soa_arg:Cont)),
 arg_st:Arg)
morphs
be becomes are,
X becomes X.
```

The two DLR-1s are identical except for their complementary subject verb agreement coverage, and the phonological forms which depend on agreement.

Note that in the case of past tense the necessity of splitting up the Past Tense Lexical Rule according to the agreement distinction of *be* between *was* and *were* also leads to the occurrence of two past tense lexical entries for each other word. For example *said* has now two derived lexical entries for past tense (just like any other word), one for first and third person singular subjects and one for all other agreement possibilities, although there is no phonological distinction between these forms in English. Since parsing works on the level of descriptions, using LRs might thus have a direct effect on how many solutions to queries we get back from the system!

In general, when working with LRs, it is a good strategy to look at the descriptions of words generated by the compiler given our lexical entries and the constraints in our grammar. These are the descriptions which are relevant for thinking about what the input to the lexical rules looks like. It is also a good idea to check with the `lex` command how many descriptions are available for a given phonological form of a word, and to see if the result matches our intentions.

The Passive Lexical Rule is the most complex lexical rule in our fragment, because it involves rearranging the argument structure of verbs. The order of elements on the argument structure lists of the rule's output is described with an `append` relation. It turns out that this fact does not introduce further complications for an DLR-1 version of the Passive Lexical Rule. We can, in a straightforward manner, use a relational attachment to the output description and directly simulate the original specification. The definite clause definition of `append` uses the familiar delays and is not repeated here.

```
% Passive Lexical Rule

passive_lex_rule ##
(word,
 synsem:(loc:(cat:head:(verb,
                        vform:psp,
                        aux:(Aux,minus),      % no passive of auxiliaries
                        inv:Inv,
                        pred:Pred),
              cont:soa_arg:Cont),
         nonloc:Nonloc),
 arg_st:[(loc:(cat:(head:noun,
                    val:(subj:e_list,
                         comps:e_list)),
              cont:Cont2)),
         Synsem1|
         List])
 **>
(word,
 synsem:(loc:(cat:head:(verb,
                        vform:pas,
                        aux:Aux,
                        inv:Inv,
                        pred:Pred),
              cont:Cont),
         nonloc:Nonloc),
 arg_st:([Synsem1|List];Result))
if append(([Synsem1|List]),[(loc:(cat:(head:(prep,
                                        pform:by),
                                    val:(subj:e_list,
                                         comps:e_list)),
                                cont:Cont2))],Result)
morphs
X becomes X.
```

The description of the ARG_ST list of the output is described disjunctively. The first disjunct in the description of the ARG_ST list of the output captures the case in which the subject is not realized overtly in the passive construction. The second disjunct captures the case in which it is realized as a prepositional by-phrase.

In the first case the ARG_ST list of the output consists of the second element and the other elements behind the second element of the input verb's ARG_ST list; in the second case it is the concatenation (described by the `append` goal) of the list which we have just described with a singleton list whose element is described as the *synsem* object of a

saturated *by*-PP, whose CONTENT value is identical with the CONTENT value of the first element of the ARG_ST list of the input verb. This last identification leads to the correct interpretation of the by-phrase as the semantic subject of the passive verb.

Exercises

Exercise 47 *If you look at the lexical entries of the verbs in our grammar (Signature[38], Theory[39]) which are not auxiliaries, you will notice that their nominal arguments are always fully saturated, i.e., the* SUBJ *list and the* COMPS *lists of the selected nominal arguments must be empty. Stating this fact separately in each lexical entry might look like we are missing a potential generalization in English.*

Formulate this supposed generalization in two steps:

1. *Give an HPSG specification of the observed generalization concerning our small fragment of English.*
2. *Implement your HPSG specification as an extension of our current fragment. To hand in your grammar, please create a new folder in your personal ILIAS group and upload both your signature file and your theory file to this folder.*

Exercise 48 *We have not talked about the new lexical entry of the lexical preposition* to. *If you look at its lexical entry in our implemented grammar and compare it with the corresponding lexical entry in the core fragment, you will see that it becomes much more complex. The new lexical entry comprises a functional description,* `search_rel`, *whose definite clause definition is rather complex.*

Describe in a few sentences what the effect of the functional description in the lexical entry is. (Hint: You might want to investigate what happens if you use the old lexical entry of the lexical preposition to *in our extended fragment)*

GLOSSARY

3.2.3 Unbounded Dependencies: Fragment III

ABSTRACT

The last extension of our fragment of English will add to the grammar an analysis of unbounded dependencies. *A very small extension of the signature will suffice to enable us to formulate generalizations over a large class of new empirical phenomena. The new grammar will raise questions of how closely an implemented grammar should mirror grammars designed in theoretical linguistics.*

[38] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Fragment-with-Lex-Gens/signature`
[39] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Fragment-with-Lex-Gens/theory.pl`

Unbounded dependency constructions (short: *UDCs*), which are also known as *long movement* constructions, have been a longstanding and important topic of all varieties of generative grammar. The term *unbounded dependency construction* refers to a syntactic construction in which at least one constituent appears in a sentence peripheral position which is distinct from the position in which the constituent would appear in 'normal' or 'unmarked' word order. Moreover, the dislocated constituent of an unbounded dependency construction may occupy a syntactic position which may be several sentence boundaries away from the 'unmarked' position of this constituent. It is this last property which gave the construction the name *unbounded* dependency.

Our grammar will encompass the data in (91), which illustrate the topicalization of non-wh elements in English. All grammatical sentences in the examples will be licensed by Fragment III, whereas the sentences preceded by an asterisk will be correctly identified as being ungrammatical. The dislocated constituent will be enclosed in square brackets, and its 'unmarked' position will be signaled by the position of the symbol t in each sentence. The letter t has been chosen to remind the reader of the technical name for the empty category in generative grammar, which is called *trace*.

(91) a. [To Peter] Mary gave it t.

b. [Peter] Mary gave it to t.

c. [Peter] Mary likes t.

d. She says that [Peter] Mary likes t.

e. *She says [Peter] that Mary likes t.

f. [Peter] she says that Mary likes t.

g. *[That] she said t Mary gave it to Peter.

h. *[Gave] she said that Mary t it to Peter.

i. *[To] she said that Mary gave it t Peter.

As the reader may already have inferred from the examples, the empirical scope of the grammar will be restricted to the topicalization of syntactic arguments. In particular, one should note that the extraction of adjuncts is not considered in this grammar fragment.

The classical examples for unbounded dependency constructions in English are *wh-questions* such as *'[Which book] did Peter say that Mary likes* t*?'* In the present extension of our core fragment of English we do not include wh-questions because a satisfactory analysis of wh-questions involves additional syntactic constructions which we do not wish to discuss here.

We will focus instead on *topicalization* and *long topicalization* data. Let us very briefly characterize this phenomenon: In sentences (91a)–(91d) we see that a PP-complement of a verb, an NP-complement of a preposition, and an NP-complement of a verb can be

topicalized within a sentence, even if the sentence in which the topicalization occurs is an embedded sentence (91d). However, topicalized constituents in an embedded that-clause must follow the complementizer ((91d) vs. (91e)). Sentence (91f) is an example of topicalization of a complement from an embedded clause. The sentences in (91g) through (91i) show that certain lexical categories (such as complementizers, finite verbs, and prepositions without their complements) cannot be topicalized.

We will not treat cases of VP topicalization, which are illustrated in (92):

(92) a. [Reading the book] she may have been t.

b. [Win the game] they will t.

For a thorough discussion and an empirical overview of UDCs in English in the context of HPSG the reader is referred to [Pollard and Sag, 1994, Chapter 4]. Our analysis follows this theory in spirit, but is empirically much more restricted and differs in a number of important technical details. In what follows we presuppose a basic knowledge of theories of unbounded dependencies as well as a certain familiarity with SLASH percolation mechanisms for analyzing the bottom, middle and top part of an unbounded dependency construction.

GLOSSARY

3.2.3.1 Specification of Fragment III

ABSTRACT

As with the previous fragments we will present a specification of the signature and the principles of the grammar before we proceed to its implementation in TRALE.

We will keep the signature of Fragment II and extend it only slightly. Just as when we made the step from Fragment I to Fragment II, we will not repeat the entire signature of the previous fragment. Below we will simply enumerate the additions to the signature. A complete version of the new signature (without parametric sorts and without the relation symbols and their arity) can be found in the ultimate TRALE implementation of Fragment III.

We only need two new sorts with their appropriateness specifications and one additional relation symbol in order to add a treatment of topicalization to our previous fragment of English. The new sort *nonlocal*, appropriate for the attribute NONLOCAL at *synsem*, hosts two new attributes with list values. They are called INHERITED and TO_BIND. The attribute INHERITED has the function of housing the LOCAL values of dislocated elements. This makes them accessible to a locally defined percolation mechanism which will ultimately establish the identity of *local* structures between the *filler* and the *gap*. TO_BIND is a technical attribute which controls how the upwards-percolation of dislocated elements is discharged when they are syntactically realized as filler constituents. In order to integrate

head-filler structures, we need a new subsort of the DAUGHTERS value *const_struc* at *phrase*. The new sort for head-filler structures is called *hf_struc*.

Finally, we need a `member` relation with arity 2 for formulating our constraints on extraction. Here is a summary of the new parts of the signature:

```
type_hierarchy
top
   synsem local:local nonlocal:nonlocal
   nonlocal inherited:list(local) to_bind:list(local)
   const_struc
      hf_struc
relations
member/2
.
```

On the basis of our new signature, we can now turn to the specification of lexical entries and new principles of grammar.

Lexical Entries In strong contrast to the previous extension, the one which we are concerned with here is mostly non-lexical. This time the extension focuses on new phrasal syntactic structures, and on additional sub-structures of all types of signs. We do not add any elements with new phonology to the grammar. However, each previous lexical entry needs an additional specification which excludes it from becoming a possible lexical source of an unbounded dependency construction.

For this purpose, we add the following specification to all lexical entries of Fragment II:

(93) Lexical NONLOCAL value specification for all words in Fragment II:

$$\begin{bmatrix} \text{SYNSEM NONLOC} & \begin{bmatrix} \text{INHERITED} & \langle\rangle \\ \text{TO\_BIND} & \langle\rangle \end{bmatrix} \end{bmatrix}$$

The specification [INHERITED ⟨⟩] says that no unbounded dependency originates with the words in question. The specification [TO_BIND ⟨⟩] says that the syntactic phrases in which the words are syntactically realized as daughters are never head filler structures.

The lexical entry for traces is the single exception to the NONLOCAL specification of all other words in our fragment of English, and it is the only new lexical entry in Fragment III:

(94) Lexical entry of the trace:

$$\begin{bmatrix} \textit{word} & & \\ \text{PHON} & \langle\rangle & \\ \text{SYNS} & \begin{bmatrix} \text{LOC} & \boxed{1} \\ \text{NONLOC} & \begin{bmatrix} \text{INHERITED} & \langle \boxed{1} \rangle \\ \text{TO\_BIND} & \langle\rangle \end{bmatrix} \end{bmatrix} & \end{bmatrix}$$

The specification $\left[\text{INHERITED } \langle \boxed{1} \rangle\right]$ marks traces as the bottom part of an unbounded dependency construction. The principles of grammar will have to take care of the proper percolation of the element on the INHERITED list. Like all other words in the grammar, traces cannot bind off INHERITED elements ([TO_BIND ⟨⟩]).

As in the original theory of unbounded dependencies in HPSG of [Pollard and Sag, 1994], our traces are words with an empty phonology which share their LOCAL value with the single element on an INHERITED list. Small differences are caused by a number of simplifications in our fragment. First of all INHERITED values are lists, not sets, because we want to avoid the technical and computational complications which sets cause. Secondly, the original analysis distinguishes between different kinds of unbounded dependencies. In order to handle this Pollard and Sag's *nonlocal* sort has three attributes, dubbed SLASH, QUE, and REL, for different kinds and aspects of unbounded dependency constructions. Since we wish only to discuss unbounded dependencies of the type which is usually analyzed by means of the SLASH attribute, we will omit the distinction (and even the SLASH attribute itself). We will simplify the appropriateness conditions to the minimally necessary ingredients of a theory of topicalization in English: We assign the NONLOCAL value *nonlocal* the familiar two attributes INHERITED and TO_BIND. In Pollard and Sag's full analysis, arcs with these two labels are placed in the SLASH, QUE, and REL values.

Principles The new grammatical principles are concerned with restricting the distribution of traces in phrase structure, guaranteeing the proper percolation of INHERITED values and licensing a new type of phrase, head filler phrases, in order to accommodate extracted constituents in our syntactic structures.

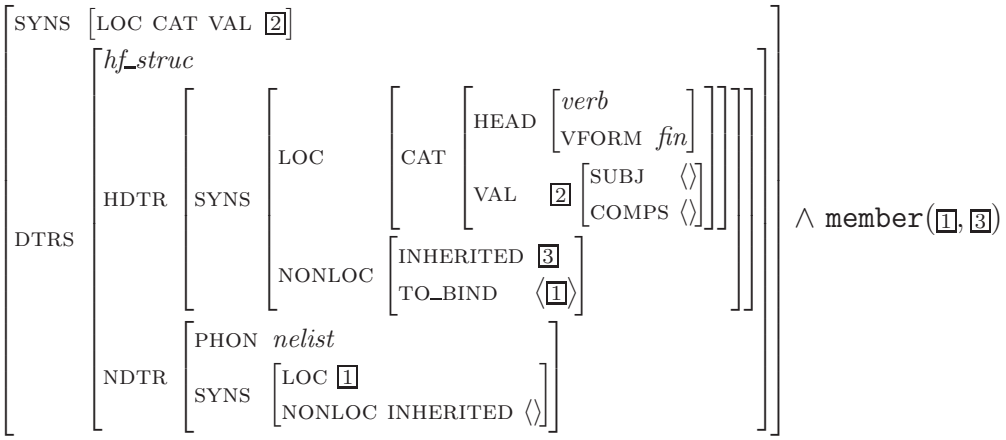
We will begin with a modification of the existing ID PRINCIPLE. One more disjunct in its consequent, the HEAD FILLER SCHEMA (HFS), licenses the new head filler structures:

(95) The ID PRINCIPLE:

$$\textit{phrase} \rightarrow (\text{HSS} \vee \text{HCS} \vee \text{SAIS} \vee \text{HAS} \vee \text{HFS})$$

The meta-variable HFS in the rewritten ID PRINCIPLE stands for the following description of head filler phrases:

(96) The HEAD FILLER SCHEMA:

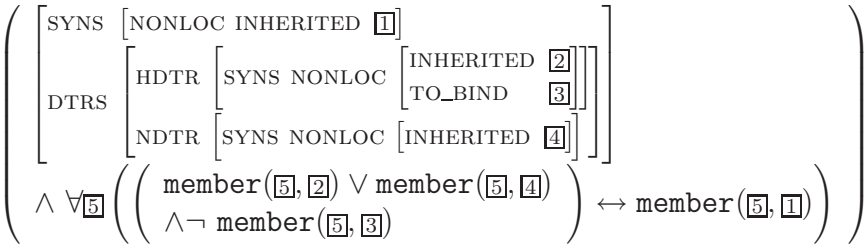


According to the HEAD FILLER SCHEMA, filler daughters are sisters of saturated projections of finite verbs. Their LOCAL value is identical to a member of the INHERITED list of the head daughter. Note the identity of the VALENCE values of the head daughter and the mother node, which is established by the variable 2. This specification is necessary because our grammar has no independent SUBCATEGORIZATION PRINCIPLE, and we have to guarantee that the projection of a verb phrase which is already syntactically saturated remain saturated when it combines with a filler. The *nelist* specification for the phonology of the non-head daughter excludes the possibility of the filler being a trace.

The TO_BIND specification interacts with the NONLOCAL FEATURE PRINCIPLE in such a way that the *local* object denoted by 1 is not on the INHERITED list of the mother phrase of the construction. This means that the constituent with LOCAL value 1 is no longer marked as missing at the mother node of the head filler phrase.

The NONLOCAL FEATURE PRINCIPLE might at first look complicated due to the seemingly complex restrictions which are expressed with the `member` relation. On closer inspection, however, it will turn out to be very straightforward:

(97) The NONLOCAL FEATURE PRINCIPLE (NFP):



The NFP says that the INHERITED list of a phrase consists of those elements of the INHERITED lists of its head daughter and its non-head daughter which do not occur on the TO_BIND list of the head daughter: In other words, all the requirements about missing constituents which are not realized in the form of the local filler daughter are passed upwards.

The PHRASAL TO-BIND PRINCIPLE ensures that nonlocal dependency requirements can only be discharged in head filler structures:

(98) The PHRASAL TO-BIND PRINCIPLE:

$$\left[\begin{array}{l} \textit{phrase} \\ \text{DTRS}\ \left[\text{HDTR}\ \left[\begin{array}{l}\textit{phrase}\\ \text{SYNS NONLOC TO\_BIND}\ \textit{nelist}\end{array}\right]\right] \end{array}\right] \leftrightarrow \left[\text{DTRS}\ \textit{hf\_struc}\right]$$

Since non-empty TO_BIND lists may only occur at head daughters of head filler structures, since every phrase has a head daughter, and since the NFP transports every member of the INHERITED list of each daughter which is not on the TO_BIND list of the head daughter upwards in the syntactic structure, head filler phrases are now the only phrases in which requirements about extracted constituents can be satisfied. As long as a nonlocal requirement is not satisfied by a corresponding filler daughter, the requirement will be preserved at the dominating nodes in the syntactic structure.

The generality of the lexical entry of the trace in (94) has an interesting and perhaps unexpected consequence for the FUNCTIONAL PREPOSITION PRINCIPLE, which we already anticipated in our original formulation of the principle. For convenience we will repeat the principle here:

(99) The FUNCTIONAL PREPOSITION PRINCIPLE:

$$\left[\begin{array}{l} \textit{word} \\ \text{PHON}\ \textit{nelist} \\ \text{SYNS LOC CAT HEAD}\ \left[\begin{array}{l}\textit{prep}\\ \text{PFORM}\ \textit{non\_lexical}\end{array}\right] \end{array}\right] \rightarrow \left[\begin{array}{l} \text{SYNS}\quad \left[\text{LOC}\ \left[\begin{array}{l}\text{CAT HEAD}\ \left[\begin{array}{ll}\text{MOD} & \textit{none}\\ \text{PRED} & \textit{minus}\end{array}\right]\\ \text{CONT}\ \boxed{1}\end{array}\right]\right] \\ \text{ARG\_ST}\ \left\langle\left[\text{LOC}\ \left[\begin{array}{l}\text{CAT VAL}\ \left[\begin{array}{ll}\text{SUBJ} & \langle\rangle\\ \text{COMPS} & \langle\rangle\end{array}\right]\\ \text{CONT}\ \boxed{1}\end{array}\right]\right]\right\rangle \end{array}\right]$$

Note that the antecedent requires a non-empty phonology. This had no effect in the grammar as long as we had no traces. If we did not require a non-empty phonology in the presence of traces, however, traces of an extracted PP headed by a functional preposition would then have to have a non-empty ARG_ST list, and, by virtue of the ARGUMENT REALIZATION PRINCIPLE, also a non-empty COMPS list. But that cannot be, since an extracted PP headed by a functional preposition must be a saturated phrase, which means that the shared valence attributes of the filler and the gap must be saturated. Therefore, a FUNCTIONAL PREPOSITION PRINCIPLE without the non-empty phonology requirement in the antecedent would exclude traces of PPs headed by functional prepositions, although they are grammatical, as is illustrated by the following example:

(100) [To Mary] Peter gave the book t.

Constraints on Extraction The previous new principles were concerned with the middle part and the top part of unbounded dependency constructions. The lexical entry for traces which licenses the bottom part is still completely unrestricted: Any kind of word might occur in the shape of a trace anywhere in the syntactic structure. As we have already observed in the data in (91), this is far too general. We need restrictions on possible extractees and extraction sites. The TRACE PRINCIPLE, which we formulate in three sub-principles, and the SUBJECT CONDITION are examples of such constraints.

In order to restrict the occurrence of traces to complements (with nominal and prepositional heads) of non-functional categories we impose a (structural) TRACE PRINCIPLE:

(101) The TRACE PRINCIPLE:

1. $\begin{bmatrix} \textit{phrase} \\ \text{DTRS NDTR PHON } \langle\rangle \end{bmatrix} \rightarrow \begin{bmatrix} \text{SYNS} & \begin{bmatrix} \text{LOC CAT HEAD } \neg\ \textit{functional} \\ \text{NONLOC TO-BIND } \langle\rangle \end{bmatrix} \\ \text{DTRS} & \textit{hs\_struc} \vee \textit{hc\_struc} \vee \textit{sai\_struc} \end{bmatrix}$
2. $[\textit{phrase}\ ] \rightarrow [\text{DTRS HDTR PHON } \textit{nelist}]$
3. $\begin{bmatrix} \textit{word} \\ \text{PHON } \textit{elist} \end{bmatrix} \rightarrow [\text{SYNS LOC CAT HEAD } \textit{noun} \vee \textit{prep}]$

Clause (1) of the TRACE PRINCIPLE restricts the possible occurrence of traces in argument positions. The three lines of the consequent make the relevant statements: According to the first line, traces may not be complements of functional heads. The third line excludes traces in adjunct position and in the position of fillers. The second line is the most intricate one: It excludes so-called vacuous movement of subjects in an indirect manner by saying that the phrasal mother of a trace may not bind off an unbounded dependency. In other words the mother may not be the daughter of a head-filler structure. Here is an example of the kind of structure which this part of the principle forbids:

(102) [He] t is walking.

Clause (2) says that heads may not be extracted. Clause (3) restricts long movement in our fragment of English to nominal and prepositional signs. Note that for this reason VPs cannot be extracted.

According to the SUBJECT CONDITION, a subject may only be the source of an extraction if (1), it is either extracted itself, or (2), something is extracted out of the subject and at the same time another argument of the same syntactic functor is either itself extracted or something is extracted out of it:

(103) The SUBJECT CONDITION:

$$\forall\boxed{1}\ \forall\boxed{3}\ \exists\boxed{2}\ \exists\boxed{4}$$
$$\left(\begin{bmatrix} \textit{word} \\ \text{SYNS LOC } [\text{CAT HEAD PRED } \textit{plus}] \\ \text{ARG\_ST } \left\langle \boxed{3}\,[\text{NONLOC INHERITED } \textit{nelist}] \middle|\ \boxed{1} \right\rangle \end{bmatrix} \rightarrow \right.$$
$$\left. \left(\boxed{3} \begin{bmatrix} \text{LOC } \boxed{4} \\ \text{NONLOC INHERITED } \langle \boxed{4} \rangle \end{bmatrix} \vee \texttt{member}\left(\boxed{2}\,[\text{NONLOC INHERITED } \textit{nelist}], \boxed{1} \right) \right) \right)$$

The SUBJECT CONDITION is a restriction on predicative words. In our fragment these are the verbs. In particular, our prepositions are not predicative. For this reason their complements may be extracted although the SYNSEM values of the complements are the first elements of the ARG_ST lists of the prepositions.

Word Order Having added a new type of constituent structure to our grammar we also need to extend the CONSTITUENT ORDER PRINCIPLE. We have defined the relative phonological order of syntactic daughters as a function of the constituent structure type of the mother node in Fragment I. In the new CONSTITUENT ORDER PRINCIPLE we simply add a statement which orders the phonology of fillers in front of the phonology of the head of the construction.

(104) The CONSTITUENT ORDER PRINCIPLE:

$$\textit{phrase} \rightarrow \left(\begin{array}{l} \left[\begin{array}{ll} \text{PHON} & \boxed{1} \\ \text{DTRS} & \left[\begin{array}{ll} \text{HDTR PHON} & \boxed{2} \\ \text{NDTR PHON} & \boxed{3} \end{array} \right] \end{array} \right] \\ \wedge \Big(\big[\text{DTRS} \ \big(\ \textit{hc\_struc} \vee \textit{ha\_struc} \vee \textit{sai\_struc} \ \big) \big] \rightarrow \texttt{append}(\boxed{2}, \boxed{3}, \boxed{1}) \Big) \\ \wedge \Big(\big[\text{DTRS} \ \big(\ \textit{hs\_struc} \vee \textit{hf\_struc} \big) \big] \rightarrow \texttt{append}(\boxed{3}, \boxed{2}, \boxed{1}) \Big) \end{array} \right)$$

Parametric Sorts Our extension comprises a new parametric sort specification. As can be seen in the additions to the signature, the list values of the new attributes INHERITED and TO_BIND are supposed to be lists of *local* objects. Just as before we take the specification in the signature to be an abbreviation of the appropriate principles and a new relation of arity 1 (`list_of_locs`), which introduces the intended restrictions on the relevant lists. In MoMo notation we can render the necessary principles as follows:

```
VX(list_of_locs(X) <*> (X:e_list;
                        ^Y(X:hd:loc,X:tl:Y,list_of_locs(Y)))).

nonlocal *> ^Y(inherited:Y, list_of_locs(Y)).
nonlocal *> ^Y(to_bind:Y, list_of_locs(Y)).
```

These principles will, of course, have direct counterparts in the implementation of Fragment III.

GLOSSARY

3.2.3.2 Implementation of Fragment III

ABSTRACT

We will focus on a number of computationally motivated reformulations of the original HPSG specification. In the implementation of the NONLOCAL FEATURE PRINCIPLE *we will see again that the implementation of HPSG principles with relations may deviate significantly from their declarative specifications. The treatment of the trace will be another example of an element of the HPSG specification which is equivalently recoded for processing. The implemented grammar also raises considerations about the adequate empirical coverage of a computational grammar which incorporates very general principles of theoretical linguistics.*

Fragment III (`signature`[40], `theory.pl`[41]) is by far the most complex grammar of a natural language which we have worked with so far. The introduction of unbounded dependencies immediately forces the computational linguist to give serious thought to the runtime behavior of the grammar. Procedural mistakes in the implementation of the grammar can easily lead to infinite compile time, or, if compilation succeeds, to infinite runtime for some or all interesting queries. The only safe and systematic way to overcome the inherent difficulties of working with a very expressive description language such as ours is to emphasize the precise logical specification of the grammar, and to obtain a clear understanding of the meaning of the complete grammar specification, and of the procedural properties of the constraint system and the necessary Prolog components of TRALE. A significant part of these skills can only be acquired by practice and by experience with mistakes.

For the discussion of the implementation of the Fragment III signature we will restructure slightly the order of presentation in the specification of the previous section. We will begin with the straightforward realization of the new parametric sort specification (*list(local)*). Then we will proceed to the principles which govern the top and the middle part of unbounded dependencies. Due to the use of phrase structure rules in TRALE, this will automatically take care of word order as well. Finally we will discuss the implementation of the constraints on extraction, and we will see that these constraints are partially built into the definition of the empty categories in the TRALE grammar. For computational reasons the descriptions of the traces in the implementation are much more detailed than in the HPSG specification, but we will argue that we will nevertheless obtain a denotationally equivalent grammar.

Parametric Sorts The encoding of the parametric sort specifications of the INHERITED and TO_BIND values follows the by now familiar pattern. The HPSG relation `list_of_locs` is approximated by the definite clause definition of the Prolog predicate `list_of_locs`. In the definite clause we employ a delay statement to guarantee termination of the goal

[40] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/UDC-Grammatik/signature`
[41] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/UDC-Grammatik/theory.pl`

during runtime. The functional notation for the predicate `list_of_locs` is chosen for the simplicity of the notation.

```
fun list_of_locs(-).   % allow functional notation for list_of_locs

list_of_locs(X) if
   when( (X=(e_list;ne_list) ),  % apply only with sufficient information
           und_list_of_locs(X)). % about the list

und_list_of_locs(X) if (X=e_list).
und_list_of_locs(X) if (X=[(loc)|Y]),list_of_locs(Y).

nonloc *> (inherited:list_of_locs,
           to_bind:list_of_locs).
```

Principles (including Word Order) The HEAD FILLER SCHEMA assumes the shape of a phrase structure rule with the relevant description annotations. We call it the HEAD FILLER RULE. As is usual with phrase structure rules, the HEAD FILLER RULE automatically incorporates the word order requirements which the HPSG specification had to state separately in the CONSTITUENT ORDER PRINCIPLE.

Before we can state the HEAD FILLER RULE we must define a definite clause which simulates the `member` relation. In another principle later on we will want to use the `member` predicate in functional notation, which is why we already include a statement which allows functional notation with respect to the second argument. The predicate is called `our_member`, in order to avoid a naming conflict with existing predicates in the system.

```
% functional definition of our-member/2 for the Subject Condition

fun our_member(+,-).

our_member(Hd,[Hd|_]) if true.
our_member(Hd,[_|Tl]) if our_member(Hd,Tl).
```

With the `our_member` predicate in hand, the description of the HEAD FILLER SCHEMA can be translated faithfully into the format of a TRALE rule:

```
% Head Filler Rule
head_filler_rule ##
(phrase,
 synsem:loc:cat:val:Val,
 daughters:(hf_struc,
            hdtr:Hdtr,
            ndtr:Ndtr))
```

```
   ===>
cat> (Ndtr, phon:ne_list,
            synsem:(loc:Local,
                    nonloc:inherited:e_list)),
cat> (Hdtr, synsem:(loc:cat:(head:(verb,
                                   vform:fin),
                             val:(Val,(subj:e_list,
                                       comps:e_list))),
                    nonloc:(to_bind:[Local],
                            inherited:List))),
goal> our_member(Local,List).
```

Note the syntactic form of the operator `goal` in the context of rules: The syntactic form `goal>` is necessary here in parallel to the `cat>` operators of the syntactic daughters, and it is conjoined to the rule by a comma.

The Nonlocal Feature Principle is the principle which requires the most drastic recoding. Expressing the percolation of the INHERITED values of daughters contingent upon the TO_BIND value of the head daughter is easy in the declarative logical environment of HPSG, since HPSG provides universal quantification; a relational expression (`member`) which receives a classical interpretation in the scope of negation and the logical constant for equivalence ($\leftrightarrow$). None of these are available in TRALE. Even a brief investigation of the problem will suffice to convince the reader that an adequate encoding of the specified percolation mechanism with definite clauses must approach the problem from a very different angle. The entire mechanism must be restated positively and without explicit quantification, and the new formulation should even avoid substitutes for negation, such as negation by failure, in order to achieve a logically clean and computationally well-behaved encoding. The one advantage which we gain over an informal characterization of the problem from the exact HPSG rendering of the principle is a precisely specified target which we could ultimately use for a formal proof of the logical equivalence of the recoding.

The solution to the given recoding problem may be found by considering the case distinctions made by the Nonlocal Feature Principle for the intended feature value percolation. There are in fact only two cases: Either every element of the INHERITED values of the daughters ascends to the mother node, in which case the TO_BIND list is empty (1), or the TO_BIND list contains exactly one element. In this case, this element must appear on the INHERITED list of one of the daughters. Every element on the daughters' INHERITED lists will be percolated upwards, with the exception of the single element which we have found on the TO_BIND list (2).

It is precisely this binary (and positively expressed) case distinction which is encoded in the new four place predicate `nfp`. In the first case, the INHERITED lists are concatenated in an obvious way by the `append` predicate to yield the INHERITED value of the mother node. In the second case, in which the single element of the TO_BIND list is removed from the combined INHERITED lists, we select the element of the TO_BIND list from the concatenated INHERITED lists and pass the result upwards to the mother node.

```
% The Nonlocal Feature Principle

phrase *>
   (synsem:nonloc:inherited:Result,
 daughters:(hdtr:synsem:nonloc:(inherited:List1,
                                to_bind:Subtrac),
  ndtr:synsem:nonloc:inherited:List2))
goal
   nfp(List1,List2,Subtrac,Result).

nfp(List1,List2,[],Result) if append(List1,List2,Result).     % (1)
nfp(L1,L2,([Ele]),Result) if (append(L1,L2,L3),select(Ele,L3,Result)). % (2)

% definition of select/3

select(Ele,L1,L2) if
   when(  ( L1=(e_list;ne_list)
               ),
            und_select(Ele,L1,L2)).

und_select(Ele,[Ele|Tl],Tl) if true.
und_select(Ele,[Hd|Tl1],[Hd|Tl2]) if select(Ele,Tl1,Tl2).
```

The predicate `select/3` is defined with the usual precautions which ensure that at runtime enough is known about the crucial list argument to guarantee termination. Note that in the present context we could in principle omit the delay, because a delayed `append` goal must be performed prior to evaluating `select`. If this were the only instance of using `select/3` in a grammar, efficiency considerations might lead one to simplify the predicate and eliminate the delay mechanism.

For the sake of completeness we also quote the definition of `append/3` on which the Nonlocal Feature Principle is based:

```
append(X,Y,Z) if
   when(  ( X=(e_list;ne_list)
          ; Y=e_list
          ; Z=(e_list;ne_list)
               ),
            undelayed_append(X,Y,Z)).

undelayed_append(L,[],L) if true.
undelayed_append([],(L,ne_list),L) if true.
undelayed_append([H|T1],(L,ne_list),[H|T2]) if
  append(T1,L,T2).
```

The PHRASAL TO-BIND PRINCIPLE is much easier to capture in TRALE than the NONLOCAL FEATURE PRINCIPLE. Since the HPSG specification takes the syntactic shape of an equivalence, the principle needs to be split into two implications in TRALE. This is of course a simple logical exercise:

```
% Phrasal to-bind Principle

(phrase,
 daughters:hdtr:(phrase,
                 synsem:nonloc:to_bind:ne_list))
*> (daughters:hf_struc).

(phrase,
 daughters:(hs_struc;hc_struc;sai_struc;ha_struc))
*> daughters:hdtr:synsem:nonloc:to_bind:e_list.
```

Constraints on Extraction and Lexical Entries The constraints on extraction, which comprise the lexical entries for traces, complete the TRALE translation of our theory of topicalization. In the HPSG specification of the grammar the simple theory of extraction consists of two principles, a structural TRACE PRINCIPLE and the SUBJECT CONDITION.

The TRACE PRINCIPLE restricts the distribution of empty categories in syntactic structures. From a practical computational point of view, this is important for excluding the licensing of empty elements in syntactic structures in such a way that the grammar would admit any phonological string with an infinite number of syntactic structures.

The SUBJECT CONDITION is a typical principle concerning the contextual conditions under which a certain type of grammatical function allows extraction. It determines in which contexts extractions of subjects or extraction out of subjects is grammatical.

The TRACE PRINCIPLE is the more difficult one of the two. Whereas the SUBJECT CONDITION can be kept almost as it stands, a computationally satisfactory formulation of Clauses (2) and (3) of the TRACE PRINCIPLE requires some thought about the overall structure of Fragment III. Before we deal with them, let us first think about Clause (1):

```
% Structural Trace Principle

% Clause (1)

(daughters:ndtr:phon:e_list) *>
 (synsem:(nonloc:to_bind:e_list,                % no vacuous movement
          loc:cat:head:(verb;noun;prep;adv)),   % not functional
  daughters: (hs_struc;hc_struc;sai_struc)). % follows already independently
                                             % from the implementation
```

As the reader may easily verify, this is identical to the original formulation. We note in passing that the last line of the principle is logically redundant in the present grammar.

Omitting it would not change the collection of sentences which are licensed by the grammar. Fillers cannot be traces due to the HEAD FILLER RULE. Adjuncts cannot be traces because prepositional constituents headed by a lexical preposition cannot be traces; neither can adverbial constituents. And these two types of constituent exhaust the possibilities of adverbials in our grammar. The reason why they cannot be traces will become clear when we investigate Clauses (2) and (3) of the TRACE PRINCIPLE in TRALE.

The effect of Clauses (2) and (3) follows logically from the descriptively very specific lexical entries for traces in the TRALE grammar. The implemented grammar only contains nominal and prepositional traces. A general lexical entry for traces of the kind which occurs in the HPSG specification of Fragment III, and is usually also found in textbooks, does not belong in our TRALE grammar:

```
% empty noun

empty  word,
       phon:e_list,
       synsem:(loc:(Local,
                    cat:head:(noun,
                              pred:minus)),
              nonloc:(inherited:[Local],
                      to_bind:e_list)),
       arg_st:e_list.     % the specification arg_st:e_list excludes
                          % heads in nominal phrases from being
                          % extracted because it entails that the
                          % extracted element is fully saturated

% empty preposition

empty  word,
       phon:e_list,
       synsem:(loc:(Local,
                 cat:head:(prep,
                           mod:none,
                           pred:minus,
                           pform:(to;by))),% restricted to the 2 functional
              nonloc:(inherited:[Local],    % prepositions in the fragment
                      to_bind:e_list)),
       arg_st:e_list.     % the specification arg_st:e_list excludes
                          % heads in prepositional phrases from being
                          % extracted because it entails that the
                          % extracted element is fully saturated
```

The description of the nominal and the prepositional traces is very specific and fine-tuned to the lexical elements which are actually in the grammar. A consequence of this is

that the shape of the traces needs to be reconsidered whenever the lexicon of the grammar or its inventory of structures is extended. The traces are syntactically saturated words. This is equivalent to restricting a more general entry of a trace by a principle and to forbidding the extraction of head daughters. Let us be a little more precise about this:

Clause (3) of the TRACE PRINCIPLE follows from our lexical entries, since we only have entries for a nominal and a prepositional trace. However, the lexical entries describe more than the syntactic category of the constituents: The nominal trace also contains additional information which is common to all nouns in the grammar. The prepositional trace contains information which is shared by all functional prepositions. It is not compatible with the lexical preposition in the grammar. The lexical preposition does not need to be covered, since its maximal syntactic projection can only function as an adjunct. Adjuncts, however, do not belong to our analysis of topicalization.

Clause (2) of the TRACE PRINCIPLE follows from the fact that the lexical entries of the traces require the phonologically empty elements to have an empty ARG_ST list. From the ARGUMENT REALIZATION PRINCIPLE it follows that the traces are syntactically saturated: They have empty valence lists. In our grammar, however, the head daughter of a prepositional and of a nominal projection are never saturated. Therefore head daughters of these categories cannot be extracted.

Note that we could, of course, redundantly state Clauses (2) and (3) of the TRACE PRINCIPLE, as we have just stated the redundant parts of Clause (1). As long as we clearly document the relationship between the specification and the implemented grammar, this is not necessary.

The SUBJECT CONDITION does not require major reformulations. It suffices to move the originally universally quantified variables from the antecedent to the consequent, where they are implicitly existentially bound in TRALE. In the same spirit the descriptions in the consequent need to be reformulated as descriptions of words. The equivalence of the two formulations of the SUBJECT CONDITION should be transparent.

```
% Subject Condition

(word,
 synsem:loc:cat:head:pred:plus,
 arg_st:hd:nonloc:inherited:ne_list) *>
( (arg_st:[(loc:Loc,
    nonloc:inherited:[Loc])|_]);
  (arg_st:[_|our_member((nonloc:inherited:ne_list))])).
```

Discussion With the SUBJECT CONDITION we have completed the implementation of Fragment III, which comprises a fairly sophisticated analysis of topicalization as an instance of an unbounded dependency construction. Due to the complexity of its analysis, Fragment III gave us an opportunity to investigate a few examples of how complex HPSG specifications have to be reformulated in a computational environment. The NONLOCAL

Feature Principle in particular is a very good illustration of how to approach a principle which originally makes heavy use of those parts of HPSG's description language which are not included in TRALE's description language. The crucial skill which a computational linguist working in this field has to acquire is the ability to apply a solid understanding of the formal foundations of HPSG to a translation process from one formal language to another. The chances of success in this translation process derive from the close relationship between HPSG's description language and the computational tools of TRALE. These tools allow the user to capture even those aspects of HPSG principles which go beyond the description language of TRALE. TRALE's phrase structure rules and the layer of definite clause attachments to TRALE descriptions are two of the most important additional components of the TRALE system in this respect. The ability of the linguist to understand the meaning of an HPSG grammar and to recreate that grammar through the use of programming skills in Prolog determines the success of grammar implementations.

The implemented grammar gives us an opportunity to check the predictions of our theory of topicalization in English. With a little creativity in inventing queries to the system it is not very hard to detect interesting properties of Fragment III.

Following common practice and received wisdom of HPSG theories regarding unbounded dependency constructions, the theory of topicalization admits the extraction of subjects (under certain conditions) and the simultaneous extraction of several constituents. In the linguistic literature on unbounded dependencies, both kinds of constructions are attested by examples. Trying out these predictions of our grammar with the given lexicon leads to very remarkable sentences. Here are a few examples of sentences which are predicted by Fragment III:

(105) a. It she says that to Peter Mary gave.

b. To Peter she says that it Mary gave.

c. To Peter it she is giving.

d. It to Peter she is giving.

e. Peter Mary says that is walking.

f. Peter it to her Mary says that gives.

g. It she is giving to Peter.

On first sight it might be hard to understand the supposed meaning or to find the predicted syntactic structure of some of these sentences.[42] It is easy to verify that TRALE's behavior in response to these queries is indeed consistent with the predictions of our HPSG

[42]The reader is very much encouraged to parse these sentences in TRALE. Another interesting task might be to understand the reason for each particular number of answers to the parse queries with the given sentences. Hint: Disjunctively formulated principles might multiply identical solutions by duplicating denotationally identical solutions.

specification of Fragment III. It is equally obvious that we might not like to use a grammar such as this one for the purposes of language generation in a practical system, as long as it is liable to produce the sentences above. On the other hand, excluding some of these sentences in a general way might also exclude those sentences in the literature which motivated a formulation of a theory of topicalization which predicts these sentences.

The problem is that there are many important factors which determine the grammaticality of a particular sentence. Many of these factors we have not considered at all in our grammar. For example, the information structure of a sentence might be crucial for its grammaticality. Different kinds of nominal phrases can be extracted with different degrees of success. It is certainly necessary to distinguish pronouns from proper names, and to distinguish both of these from complex nominal phrases. It makes a difference in which context a sentence occurs. Semantic factors have to be considered. It is obvious that this list can be continued.

In a practical system one might not want to integrate all of these intricate grammaticality factors. It might not be necessary, or it might even be counterproductive to building the system in such a way that it can satisfactorily perform the intended tasks. At some point the purpose of the system must be considered, providing the crucial criteria for deciding whether or not a theoretical component of the grammar of natural languages should be included. On the basis of these decisions one might then have to introduce additional mechanisms, either to the grammar or to the overall system to which the grammar belongs. The task of these mechanisms will be to reduce the output of the system to what is adequate in the envisaged user environment. It should be clear, however, that software design decisions of this sort do not render the underlying linguistic theory invalid. Quite on the contrary, instead of being a burden to software design, a linguistically well-designed underlying grammar might be much easier to maintain and to extend in the long run. Sound theories should eventually also lead to better products.

GLOSSARY

Chapter 4

The Appendix of Pollard and Sag (1994)

Abstract

The appendix of [Pollard and Sag, 1994] contains an informal but quite precise specification of the grammar of English which they develop and discuss in their book. It is a concrete example of the signature and the set of principles which constitute their grammar of English. As such it became the most important point of reference in the literature of what constitutes an HPSG grammar.

Due to its importance, we will reiterate the specification of Pollard and Sag's grammar in this chapter. In the first section we will revise what Pollard and Sag call the *Partitions* and *Feature Declarations*. What they call the Partitions is in fact a notation for stating the set of sorts and the partial order on sorts of the signature of an HPSG grammar. The Feature Declarations specify the set of attributes and the appropriateness conditions of HPSG signatures. In the following sections, we will revise the principles of Pollard and Sag. Whereas Pollard and Sag only state them in natural language in their book, we use the AVM-variant of the syntax of RSRL to render their statements in a formal language. This gives us a clearer picture of what kind of formal constructs are involved in making these principles of grammar totally explicit. They can then serve as examples even before the reader has studied those parts of the textbook which define the syntax of their formal language and its semantics.

4.1 The Sort Hierarchy

Abstract

The Section The Sort Hierarchy *in Pollard and Sag's appendix contains the signature of their grammar, without the specification of the relation symbols and the arity of the relations.*

We will revise the *Partitions* and *Feature Declarations* of Pollard and Sag with only a few minor modifications. The modifications serve to fix shortcomings of the original grammar, or are intended to make their grammar fully compatible with our presentation of the formalism in this textbook.

Partitions contains the specification of the set of sorts, the set of species, and the sort hierarchy of the grammar. The list and set specification of the present version of these is non-parametric. In other words, we make the assumption that there are no parametric sorts in the grammar. Instead the effect of parametric sorts is achieved by principles which guarantee that the relevant lists and sets in the grammar only contain objects of the intended sorts. How this works is explained at length in Section 2.5.2.

Partitions

Partitions of *object*: *boolean (bool), case, category (cat), constituent-structure (con-struc), content (cont), context (conx), contextual-indices (c-inds), gender (gend), head, index (ind), list, local (loc), marking, mod-synsem, nonlocal (nonloc), nonlocal1 (nonloc1), number (num), person (per), phoneme-string (phonstring), preposition-form (pform), quantifier-free-parametrized-state-of-affairs (qfpsoa), semantic-determiner (semdet), set, sign, verbform (vform)*

Partitions of *bool*: *plus (+), minus (-)*

Partitions of *case*: *nominative (nom), accusative (acc)*

Partitions of *con-struc*: *coordinate-structure (coord-struc), headed-structure (head-struc)*

Partitions of *head-struc*: *head-adjunct-structure (head-adj-struc), head-complement-structure (head-comp-struc), head-filler-structure (head-filler-struc), head-marker-structure (head-mark-struc)*

Partitions of *cont*: *nominal-object (nom-obj), parametrized-state-of-affairs (psoa), quantifier (quant)*

Partitions of *nom-obj*: *nonpronoun (npro), pronoun (pron)*

Partitions of *pron*: *personal-pronoun (ppro), anaphor (ana)*

Partitions of *ana*: *refl, recp*

Partitions of *gend*: *feminine (fem), masculine (masc), neuter (neut)*

Partitions of *head*: *functional (func), substantive (subst)*

Partitions of *func*: *determiner (det), marker (mark)*

Partitions of *subst*: *adjective (adj), noun, preposition (prep), relativizer (reltvzr), verb*

Partitions of *ind*: *referential (ref), there, it*

Partitions of *list*: *empty-list (elist / $\langle\rangle$), nonempty-list (nelist)*

Partitions of *marking*: *marked, unmarked*

Partitions of *marked*: *complementizer (comp), conjunction (conj), ...*

Partitions of *comp*: *that, for*

Partitions of *mod-synsem*: *none, syntax-semantics (synsem)*

Partitions of *num*: *singular (sing), plural (plur)*

Partitions of *per*: *1st, 2nd, 3rd*

Partitions of *phonstring*: *Mary, who, gives, bagels, ...*

Partitions of *pform*: *to, of, ...*

Partitions of *qfpsoa*: *control-qfpsoa, ...*

Partitions of *control-qfpsoa*: *influence, commitment, orientation*

Partitions of *influence*: *persuade, appeal, cause, ...*

Partitions of *commitment*: *promise, intend, try, ...*

Partitions of *orientation*: *want, hate, expect, ...*

Partitions of *semdet*: *forall, exists, the, ...*

Partitions of *set*: *empty-set (eset / {}), nonempty-set*

Partitions of *sign*: *word, phrase*

Partitions of *vform*: *base (bse), finite (fin), gerund (ger), infinitive (inf), passive-participle (pas), past-participle (psp), present-participle (prp)*

Feature Declarations

The *Feature Declarations* are the specification of the appropriateness function in the signature of the grammar. Since we do not have parametric sorts, the parametric lists and parametric sets become plain lists and sets. Another modification concerns the value of the appropriateness function for ADJUNCT-DTR at *head-adjunct-struc* and FILLER-DTR at *head-filler-struc*. In both cases, we choose *sign* instead of *phrase*, because we want to license words in addition to phrases as adjunct daughters and as filler daughters in syntactic structures. Pollard and Sag's grammar erroneously excludes these, since the authors seem to presuppose that relevant words such as proper names can project to phrases before they occur in the position of a filler daughter. However, given their ID SCHEMATA, this is not possible. The easiest way to fix the mistake is to allow words in these syntactic positions.

category	HEAD	*head*
	SUBCAT	*list*
	MARKING	*marking*

c-inds	SPEAKER	*ref*
	ADDRESSEE	*ref*
	UTTERANCE-LOCATION	*ref*
	...	

commitment COMMITTOR *ref*

context BACKGROUND *set*
CONTEXTUAL-INDICES *c-inds*

control-qfpsoa SOA-ARG *psoa*

coord-struc CONJ-DTRS *set*
CONJUNCTION-DTR *word*

functional SPEC *synsem*

head-adjunct-struc HEAD-DTR *phrase*
ADJUNCT-DTR *sign*
COMP-DTRS *elist*

head-filler-struc HEAD-DTR *phrase*
FILLER-DTR *sign*
COMP-DTRS *elist*

head-mark-struc HEAD-DTR *phrase*
MARKER-DTR *word*
COMP-DTRS *elist*

head-struc HEAD-DTR *sign*
COMP-DTRS *list*

index PERSON *person*
NUMBER *number*
GENDER *gender*

influence INFLUENCE *ref*
INFLUENCED *ref*

local CATEGORY *category*
CONTENT *content*
CONTEXT *context*

nelist FIRST *object*
REST *list*

neset FIRST *object*
REST *set*

nom-obj INDEX *index*
RESTR *set*

nonlocal TO-BIND *nonlocal1*
INHERITED *nonlocal1*

nonlocal1 SLASH *set*
REL *set*
QUE *set*

noun CASE *case*

orientation EXPERIENCER *ref*

phrase DAUGHTERS *con-struc*

preposition PFORM *pform*

psoa QUANTS *list*
NUCLEUS *qfpsoa*

quantifier DET *semdet*
RESTIND *npro*

sign PHONOLOGY *list*
SYNSEM *synsem*
QSTORE *set*
RETRIEVED *set*

substantive PRD *boolean*
MOD *mod-synsem*

synsem LOCAL *local*
NONLOCAL *nonlocal*

verb VFORM *vform*
AUX *boolean*
INV *boolean*

4.2 The Principles

Abstract

This section contains the grammatical principles of Pollard and Sag's grammar of English. They are listed in exactly the same order in which they occur in the appendix of their book. However, we append the necessary relation principles to their list of principles, which they omit in the book. The relation principles are important, since they determine the meaning of a significant number of relation symbols which are needed in a precise formulation of most of Pollard and Sag's principles.

In addition to repeating the principles as they are stated informally in Pollard and Sag's book, we present their formalization in a syntax of AVM matrices.

A number of principles which are necessary in a complete formulation of the intended grammar are missing from this list. There is no WORD PRINCIPLE, since there is none in the book. For the grammar to license reasonable sentence structures, a lexicon needs to be inferred from the discussion in Pollard and Sag's text. On this basis an adequate WORD PRINCIPLE for their grammar could be formulated.

Since we do not use parametric sorts, it is also necessary to add principles which restrict the elements of various kinds of lists and sets in the way in which Pollard and Sag intend to restrict them with parametric sorts. It is obvious that this is a very simple exercise. We have omitted these principles here.

Pollard and Sag's grammar also uses sets as values of certain attributes such as SLASH and QUE. Sets are specified using a part of the sort hierarchy and appropriateness conditions which look just like lists. For the relevant structures to behave like sets, we would really need to formulate and add a small number of 'set principles' to the grammar. How this can be done is explained in [Richter, 2004, Chapter 4.4]. These additional principles are omitted here as well. Note that many relations on sets are written in a functional infix notation for better readability. This is just a notational variant of no formal significance and comparable to the functional notation of relations in TRALE.

Finally, the reader will notice an occasional notation using double brackets, $\ll$ and $\gg$, in places where the reader might have expected (descriptions of) lists. This curious notation occurs in the description of arguments of relations. These are descriptions which denote lists as well as *chains* of entities. *Chains* are an additional construct of the complete RSRL formalism for HPSG grammars, which we have not discussed in our course. *Chains* are abstract objects very similar to lists, but they are not lists in the structures described by the grammar: They are abstract entities which look like lists. The elements on these quasi-lists, however, are entities which do occur in the structures described (such as *synsem* entities or *local* entities). Readers who are interested might want to read up on this topic in [Richter, 2004], which includes a discussion of the function of *chains* in HPSG grammars, and why they are necessary. In a more superficial reading of the principles, which is sufficient for understanding the intended meaning of these descriptions, *chains* can be taken for lists.

The Head Feature Principle

> In a headed phrase, the values of SYNSEM | LOCAL | CATEGORY | HEAD and DAUGHTERS | HEAD-DAUGHTER | SYNSEM | LOCAL | CATEGORY | HEAD are token-identical.

Formalization:

$$\begin{bmatrix} \textit{phrase} \\ \text{DTRS } \textit{headed-struc} \end{bmatrix} \rightarrow \begin{bmatrix} \text{SYNSEM LOC CAT HEAD} & \boxed{1} \\ \text{DTRS HEAD-DTR SYNSEM LOC CAT HEAD} & \boxed{1} \end{bmatrix}$$

The Subcategorization Principle

> In a headed phrase, the values of DAUGHTER | HEAD-DAUGHTER | SYNSEM | LOCAL | CATEGORY | SUBCAT is the concatenation of the list value of SYNSEM | LOCAL | CATEGORY | SUBCAT with the list consisting of the SYNSEM values (in order) of the elements of the list value of DAUGHTER | COMPLEMENT-DAUGHTERS.

Formalization:

$$\begin{bmatrix} \textit{phrase} \\ \text{DTRS} \;\; \textit{headed-struc} \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} \textit{phrase} \\ \text{SS LOC CAT SUBCAT } \boxed{1} \\ \text{DTRS} \begin{bmatrix} \textit{headed-struc} \\ \text{HEAD-DTR} \begin{bmatrix} \textit{sign} \\ \text{SS LOC CAT SUBCAT } \boxed{3} \end{bmatrix} \\ \text{COMP-DTRS } \boxed{2} \end{bmatrix} \end{bmatrix}$$

$$\wedge \; \texttt{sign-ss-append}(\boxed{1}, \boxed{2}, \boxed{3})$$

The ID Principle

> Every headed phrase must satisfy exactly one of the ID schemata.

Formalization:

$$\left[\text{DTRS} \;\; \textit{headed-struc}\right] \rightarrow$$

$$(\text{Schema1} \vee \text{Schema2} \vee \text{Schema3} \vee \text{Schema4} \vee \text{Schema5} \vee \text{Schema6})$$

The Marking Principle

> In a headed phrase, the MARKING value is token-identical with that of the MARKER-DAUGHTER if any, and with that of the HEAD-DAUGHTER otherwise.

Formalization:

$$\left[\text{DTRS} \;\; \textit{headed-struc}\right] \rightarrow$$

$$\left(\begin{array}{l} \begin{bmatrix} \text{SS LOC CAT MARKING } \boxed{1} \\ \text{DTRS MARKER-DTR SS LOC CAT MARKING } \boxed{1} \end{bmatrix} \vee \\ \left(\begin{bmatrix} \text{SS LOC CAT MARKING } \boxed{1} \\ \text{DTRS HEAD-DTR SS LOC CAT MARKING } \boxed{1} \end{bmatrix} \wedge \neg \left[\text{DTRS} \;\; \textit{head-mark-struc}\right] \right) \end{array} \right)$$

The Spec Principle

In a headed phrase whose nonhead daughter (either the MARKER-DAUGHTER or COMPLEMENT-DAUGHTERS | FIRST) has a SYNSEM | LOCAL | CATEGORY | HEAD value of sort *functional*, the SPEC value of that value must be token-identical with the phrase's DAUGHTERS | HEAD-DAUGHTER | SYNSEM value.

Formalization:

$$\forall \boxed{1}\, \forall \boxed{2}$$
$$\left(\begin{array}{l} \left(\begin{array}{l} \left[\text{DTRS} \left[\left[\text{MARKER-DTR} \ \boxed{1} \right] \vee \left[\text{COMP-DTRS} \ \langle \boxed{1}, \textit{list} \rangle \right] \right] \right] \\ \wedge \ \boxed{1} \left[\text{SS LOC CAT HEAD} \begin{bmatrix} \textit{functional} \\ \text{SPEC} \ \boxed{2} \end{bmatrix} \right] \end{array} \right) \rightarrow \\ \left[\text{DTRS HEAD-DTR SS} \ \boxed{2} \right] \end{array} \right)$$

The Nonlocal Feature Principle

In a headed phrase, for each nonlocal feature F = SLASH, QUE or REL, the value of SYNSEM | NONLOCAL| INHERITED | F is the set difference of the union of the values on all the daughters, and the value of SYNSEM | NONLOCAL | TO-BIND | F on the HEAD-DAUGHTER.

Formalization:

$$\left[\text{DTRS} \ \textit{headed-struc} \right] \rightarrow$$
$$\left[\begin{array}{l} \text{SS NONLOC INHERITED} \begin{bmatrix} \text{SLASH} & \boxed{5} \setminus \boxed{1} \\ \text{QUE} & \boxed{6} \setminus \boxed{2} \\ \text{REL} & \boxed{7} \setminus \boxed{3} \end{bmatrix} \\ \text{DTRS} \ \boxed{4} \left[\text{HEAD-DTR SS NONLOC TO-BIND} \begin{bmatrix} \text{SLASH} & \boxed{1} \\ \text{QUE} & \boxed{2} \\ \text{REL} & \boxed{3} \end{bmatrix} \right] \end{array} \right]$$
$$\wedge \ \texttt{collect-dependencies}(\boxed{4}, \boxed{5}, \boxed{6}, \boxed{7})$$

The Trace Principle

The SYNSEM value of any trace must be a (noninitial) member of the SUBCAT list of a substantive word.

Formalization:

$$\forall x\ \forall \boxed{2}$$
$$\left(\begin{array}{l}
\exists \boxed{1} \\
\left(\begin{array}{l}
\left[\begin{array}{l}\text{PHON } \textit{nelist} \\ \text{SS } \left[\begin{array}{l}\text{LOC CAT } \left[\begin{array}{l}\text{HEAD VFORM } \textit{finite} \\ \text{SUBCAT } \langle\rangle\end{array}\right] \\ \text{NONLOC INHERITED } \left[\begin{array}{l}\text{QUE } \{\} \\ \text{REL } \{\} \\ \text{SLASH } \{\}\end{array}\right]\end{array}\right]\end{array}\right] \\
\wedge\ x\left[\begin{array}{l}\textit{sign} \\ \text{PHON } \langle\rangle \\ \text{SS } \boxed{2} \left[\begin{array}{l}\text{LOC } \boxed{1} \\ \text{NONLOC } \left[\begin{array}{l}\text{INHERITED } \left[\begin{array}{l}\text{QUE } \{\} \\ \text{REL } \{\} \\ \text{SLASH } \{\boxed{1}\}\end{array}\right] \\ \text{TO-BIND } \left[\begin{array}{l}\text{QUE } \{\} \\ \text{REL } \{\} \\ \text{SLASH } \{\}\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right]
\end{array}\right) \rightarrow \\
\exists y\ \exists \boxed{3} \\
\left(y\left[\begin{array}{l}\textit{word} \\ \text{SS LOC CAT } \left[\begin{array}{l}\text{HEAD } \textit{substantive} \\ \text{SUBCAT } \langle \textit{object} | \boxed{3}\rangle\end{array}\right]\end{array}\right] \wedge \texttt{member}(\boxed{2}, \boxed{3})\right)
\end{array}\right)$$

The Subject Condition

> If the initial element of the SUBCAT value of a word is slashed, then so is some other element of that list.

Formalization:

$$\forall \boxed{1}$$
$$\left(\begin{array}{l}\left[\begin{array}{l}\textit{word} \\ \text{SS LOC CAT SUBCAT } \left\langle \left[\text{NONLOC INHERITED SLASH } \textit{neset}\right] | \boxed{1}\right\rangle\end{array}\right] \rightarrow \\ \exists x\ \texttt{member}\left(x\left[\text{NONLOC INHERITED SLASH } \textit{neset}\right], \boxed{1}\right)\end{array}\right)$$

The Weak Coordination Principle

> In a coordinate structure, the CATEGORY and NONLOCAL values of each conjunct daughter are subsumed by those of the mother.

We omit Pollard and Sag's WEAK COORDINATION PRINCIPLE from the list of formalized principles. As the authors themselves note on page 203, the WEAK COORDINATION PRINCIPLE makes assumptions about the formalism that are incompatible with the assumptions in the rest of the book. The differences are so significant that it is impossible

to state this principle in the same formal framework: In the HPSG formalism of this book, there cannot be any non-trivial subsumption relation between the node labels of feature structures, because they are all maximally specific.

The Singleton Rel Constraint

> For any sign, the SYNSEM | NONLOCAL | INHERITED | REL value is a set of cardinality at most one.

Formalization:

$$[\mathit{sign}] \rightarrow$$
$$\Big[\text{SS NONLOC INHERITED REL } \Big[\{\} \vee \{\mathit{object}\}\Big]\Big]$$

The Relative Uniqueness Principle

> For any phrase, a member of the set value of SYNSEM | NONLOCAL | INHERITED | REL may belong to the value of that same path on at most one daughter.

Formalization:

$$\forall \boxed{1}\ \forall \boxed{3}$$
$$\left(\begin{array}{l} \exists \boxed{2} \left(\begin{array}{l} \left[\begin{array}{l} \mathit{phrase} \\ \text{SS NONLOC INHERITED REL } \boxed{2} \\ \text{DTRS } \boxed{3}\ \mathit{headed\text{-}struc} \end{array}\right] \\ \wedge\ \texttt{member}(\boxed{1}, \boxed{2}) \end{array}\right) \rightarrow \\ \exists \boxed{4} \left(\begin{array}{l} \texttt{all-dtrs-rels}(\boxed{3}, \boxed{4}) \\ \wedge\ \texttt{at-most-once}(\boxed{1}, \boxed{4}) \end{array}\right) \end{array}\right)$$

The Clausal Rel Prohibition

> For any *synsem* object, if the LOCAL | CATEGORY | HEAD value is *verb* and the LOCAL | CATEGORY | SUBCAT value is $\langle\rangle$, then the NONLOCAL | INHERITED | REL value must be $\{\}$.

Formalization:

$$\left[\text{LOC CAT } \begin{bmatrix} \text{HEAD} & \mathit{verb} \\ \text{SUBCAT} & \langle\rangle \end{bmatrix}\right] \rightarrow$$
$$\big[\text{NONLOC INHERITED REL } \{\}\big]$$

The Binding Theory

Principle A. A locally o-commanded anaphor must be locally o-bound.
Principle B. A personal pronoun must be locally o-free.
Principle C. A nonpronoun must be o-free.

Formalization:

Principle A:

$$\forall x \left(\exists y\, \texttt{loc-o-command}\Big(y, {}^{x}\big[\textsc{loc cont}\ \ \textit{ana}\big]\Big) \rightarrow \exists z\, \texttt{loc-o-bind}(z, x)\right)$$

Principle B:

$$\forall x \left({}^{x}\big[\textsc{loc cont}\ \ \textit{ppro}\big] \rightarrow \neg\exists y\, \texttt{loc-o-bind}(y, x)\right)$$

Principle C:

$$\forall x \left({}^{x}\big[\textsc{loc cont}\ \ \textit{npro}\big] \rightarrow \neg\exists y\, \texttt{o-bind}(y, x)\right)$$

The Control Theory

If the SOA-ARG value of a *control-qfpsoa* is token-identical with the CONTENT value of a *local* entity whose CATEGORY | SUBCAT value is a list of length one, then the member of that list is (1) reflexive, and (2) coindexed with the INFLUENCED (respectively, COMMITTOR, EXPERIENCER) value of the *control-qfpsoa* if the latter is of sort *influence* (respectively, *commitment*, *orientation*).

Formalization:

$$\forall\boxed{1}\,\forall\boxed{2}\,\forall\boxed{3}\left(\begin{array}{l}\exists\boxed{4}\left(\boxed{1}\begin{bmatrix}\textit{influence}\\ \textsc{soa-arg}\ \ \boxed{4}\end{bmatrix}\wedge\boxed{2}\begin{bmatrix}\textit{local}\\ \textsc{cat subcat}\ \ \langle\boxed{3}\rangle\\ \textsc{cont}\ \ \boxed{4}\end{bmatrix}\right)\rightarrow\\ \exists\boxed{5}\left(\boxed{3}\Big[\textsc{loc cont}\ \begin{bmatrix}\textit{reflexive}\\ \textsc{index}\ \ \boxed{5}\end{bmatrix}\Big]\wedge\boxed{1}\big[\textsc{influenced}\ \ \boxed{5}\big]\right)\end{array}\right)\wedge$$

$$\forall\boxed{1}\,\forall\boxed{2}\,\forall\boxed{3}\left(\begin{array}{l}\exists\boxed{4}\left(\boxed{1}\begin{bmatrix}\textit{commitment}\\ \textsc{soa-arg}\ \ \boxed{4}\end{bmatrix}\wedge\boxed{2}\begin{bmatrix}\textit{local}\\ \textsc{cat subcat}\ \ \langle\boxed{3}\rangle\\ \textsc{cont}\ \ \boxed{4}\end{bmatrix}\right)\rightarrow\\ \exists\boxed{5}\left(\boxed{3}\Big[\textsc{loc cont}\ \begin{bmatrix}\textit{reflexive}\\ \textsc{index}\ \ \boxed{5}\end{bmatrix}\Big]\wedge\boxed{1}\big[\textsc{committor}\ \ \boxed{5}\big]\right)\end{array}\right)\wedge$$

$$\forall\boxed{1}\,\forall\boxed{2}\,\forall\boxed{3}\left(\begin{array}{l}\exists\boxed{4}\left(\boxed{1}\begin{bmatrix}\textit{orientation}\\ \textsc{soa-arg}\ \ \boxed{4}\end{bmatrix}\wedge\boxed{2}\begin{bmatrix}\textit{local}\\ \textsc{cat subcat}\ \ \langle\boxed{3}\rangle\\ \textsc{cont}\ \ \boxed{4}\end{bmatrix}\right)\rightarrow\\ \exists\boxed{5}\left(\boxed{3}\Big[\textsc{loc cont}\ \begin{bmatrix}\textit{reflexive}\\ \textsc{index}\ \ \boxed{5}\end{bmatrix}\Big]\wedge\boxed{1}\big[\textsc{experiencer}\ \ \boxed{5}\big]\right)\end{array}\right)$$

The Semantics Principle

> Semantics Principle, clause (a):
> In a headed phrase, the RETRIEVED value is a list whose set of elements is disjoint from the QSTORE value set, and the union of those two sets is the union of the QSTORE values of the daughters.
>
> Semantics Principle, clause (b):
> If the semantic head's SYNSEM | LOCAL | CONTENT value is of sort *psoa*, then the SYNSEM | LOCAL | CONTENT | NUCLEUS value is token-identical with that of the semantic head, and the SYNSEM | LOCAL | CONTENT | QUANTS value is the concatenation of the RETRIEVED value and the semantic head's SYNSEM | LOCAL | CONTENT | QUANTS value; otherwise the RETRIEVED value is the empty list, and the SYNSEM | LOCAL | CONTENT value is token-identical with that of the semantic head.

Formalization:

Clause (a):

$$\left[\textsc{dtrs}\ \textit{headed-struc}\right] \rightarrow$$

$$\begin{bmatrix} \textsc{dtrs} & \boxed{1} \\ \textsc{qstore} & \boxed{2} \\ \textsc{retrieved} & \boxed{3} \end{bmatrix}$$

$$\wedge\ \boxed{3} \overset{s}{=} \boxed{4}$$

$$\wedge\ \texttt{collect-dtrs-qstores}(\boxed{1}, \boxed{5}\ [\boxed{4} \uplus \boxed{2}])$$

Clause (b):

$$\left(\begin{array}{l}\left[\begin{array}{l}\textit{phrase}\\ \textsc{ss loc cont}\ \ \textit{psoa}\end{array}\right]\rightarrow\\ \forall\boxed{1}\\ \left(\begin{array}{l}\left[\textsc{dtrs}\ \left[\begin{array}{l}\textit{head-adj-struc}\\ \textsc{adjunct-dtr}\ \boxed{1}\end{array}\right]\ \vee\ \left[\neg\left[\textit{head-adj-struc}\right]\wedge\left[\textsc{head-dtr}\ \boxed{1}\right]\right]\right]\rightarrow\\ \exists\boxed{2}\ \exists\boxed{3}\ \exists\boxed{4}\ \exists\boxed{5}\\ \left(\begin{array}{l}\left[\begin{array}{l}\textsc{ss loc cont}\ \left[\begin{array}{l}\textsc{quants}\ \ \boxed{2}\\ \textsc{nucleus}\ \boxed{3}\end{array}\right]\\ \textsc{retrieved}\ \ \boxed{4}\end{array}\right]\wedge\boxed{1}\left[\textsc{ss loc cont}\ \left[\begin{array}{l}\textsc{quants}\ \ \boxed{5}\\ \textsc{nucleus}\ \boxed{3}\end{array}\right]\right]\\ \wedge\ \texttt{append}(\boxed{4},\boxed{5},\boxed{2})\end{array}\right)\end{array}\right)\end{array}\right)$$

$\wedge$

$$\left(\begin{array}{l}\left[\begin{array}{l}\textit{phrase}\\ \textsc{ss loc cont}\ \ \neg\textit{psoa}\end{array}\right]\rightarrow\\ \forall\boxed{1}\\ \left(\begin{array}{l}\left[\textsc{dtrs}\ \left[\begin{array}{l}\textit{head-adj-struc}\\ \textsc{adjunct-dtr}\ \boxed{1}\end{array}\right]\ \vee\ \left[\neg\left[\textit{head-adj-struc}\right]\wedge\left[\textsc{head-dtr}\ \boxed{1}\right]\right]\right]\rightarrow\\ \exists\boxed{2}\\ \left(\left[\begin{array}{l}\textsc{ss loc cont}\ \ \boxed{2}\\ \textsc{retrieved}\ \ \langle\rangle\end{array}\right]\wedge\boxed{1}\left[\textsc{ss loc cont}\ \ \boxed{2}\right]\right)\end{array}\right)\end{array}\right)$$

The Quantifier Binding Condition

> Let X be a *quantifier*, with RESTINDEX | INDEX value Y, on the QUANTS list of a *psoa* Z, and P a path in Z whose value is Y. Let the address of X in Z be QUANTS | Q | FIRST. Then P must have a prefix of one of the following three forms:
>
> (1) QUANTS | Q | FIRST | RESTINDEX | RESTRICTION;
>
> (2) QUANTS | Q | REST; or
>
> (3) NUCLEUS.

Formalization:

psoa $\rightarrow$

$$\neg\exists\boxed{1}\ \exists\boxed{2}\ \exists\boxed{3}\ \exists\boxed{4}\\ \left(\begin{array}{l}\left[\textsc{quants}\ \ \boxed{4}\right]\\ \wedge\ \texttt{to-the-right}\left(\boxed{3}\left[\textsc{restind index}\ \ \boxed{1}\right],\boxed{2},\boxed{4}\right)\\ \wedge\ \texttt{index-of-a-quantifier}(\boxed{1},\boxed{2})\end{array}\right)$$

The Principle of Contextual Consistency

The CONTEXT | BACKGROUND value of a given phrase is the union of the CONTEXT | BACKGROUND values of the daughters.

Formalization:

$[phrase] \rightarrow$

$$\begin{bmatrix}\text{SS LOC CONTEXT BACKGROUND } \boxed{1} \\ \text{DTRS } \boxed{2}\end{bmatrix}$$
$$\wedge\ \texttt{collect-dtrs-backgrounds}(\boxed{2}, \boxed{1})$$

Relation Principles

all-dtrs-rels/2

The relation `all-dtrs-rels` relates all entities x in the denotation of *headed-struc* to a list or chain y of the INHERITED REL values of their daughters.

ALL DTRS RELS PRINCIPLE

$$\forall x \forall y \left(\texttt{all-dtrs-rels}(x, y) \leftrightarrow \left(\begin{array}{l} \exists\boxed{1}\,\exists\boxed{2}\,\exists\boxed{3} \\ \left(\begin{bmatrix} x \\ \textit{head-comp-struc} \\ \text{HEAD-DTR SS NONLOC INHERITED REL } \boxed{1} \\ \text{COMP-DTRS } \boxed{2} \end{bmatrix} \right) \vee \\ \left(\wedge\ \texttt{enumerate-rels}(\boxed{2}, \boxed{3}) \wedge\ y \ll \boxed{1} \mid \boxed{3} \gg \right) \\ \exists\boxed{1}\,\exists\boxed{2} \\ \left(\begin{bmatrix} x \\ \textit{head-adj-struc} \\ \text{HEAD-DTR SS NONLOC INHERITED REL } \boxed{1} \\ \text{ADJUNCT-DTR SS NONLOC INHERITED REL } \boxed{2} \end{bmatrix} \right) \vee \\ \left(\wedge\ y \ll \boxed{1}, \boxed{2} \gg \right) \\ \exists\boxed{1}\,\exists\boxed{2} \\ \left(\begin{bmatrix} x \\ \textit{head-filler-struc} \\ \text{HEAD-DTR SS NONLOC INHERITED REL } \boxed{1} \\ \text{FILLER-DTR SS NONLOC INHERITED REL } \boxed{2} \end{bmatrix} \right) \vee \\ \left(\wedge\ y \ll \boxed{1}, \boxed{2} \gg \right) \\ \exists\boxed{1}\,\exists\boxed{2} \\ \left(\begin{bmatrix} x \\ \textit{head-mark-struc} \\ \text{HEAD-DTR SS NONLOC INHERITED REL } \boxed{1} \\ \text{MARKER-DTR SS NONLOC INHERITED REL } \boxed{2} \end{bmatrix} \right) \\ \left(\wedge\ y \ll \boxed{1}, \boxed{2} \gg \right) \end{array} \right) \right)$$

append/3

`append/3` is a relation between three lists, where the list in the third argument is obtained by concatenating the list in the second argument to the list in the first argument.

APPEND PRINCIPLE

$$\forall x \forall y \forall z \left(\texttt{append}(x, y, z) \leftrightarrow \left(\begin{array}{l} (x \langle\rangle \ \wedge \ y = z \ \wedge \ {}^{y}[list]) \\ \vee \ \exists \boxed{1} \exists \boxed{2} \exists \boxed{3} \ (x \langle \boxed{1} | \boxed{2} \rangle \wedge z \langle \boxed{1} | \boxed{3} \rangle \wedge \texttt{append}(\boxed{2}, y, \boxed{3})) \end{array} \right) \right)$$

at-most-once/2

In the relation `at-most-once` with arguments x and y, x occurs at most once in the sets (or lists or chains) that are members of y.

AT MOST ONCE PRINCIPLE

$$\forall x \ \forall y \left(\begin{array}{l} \texttt{at-most-once}(x, y) \leftrightarrow \\ \left(\begin{array}{l} y \ll\gg \ \vee \\ \exists a \, \exists z \, (y \ll a | \, z \gg \wedge \neg \texttt{member}(x, a) \wedge \ \texttt{at-most-once}(x, z)) \ \vee \\ \exists a \, \exists z \, (y \ll a | \, z \gg \wedge \texttt{member}(x, a) \wedge \ \texttt{successive-not-member}(x, z)) \end{array} \right) \end{array} \right)$$

collect-dependencies/4

The relation `collect-dependencies` relates a *headed-struc* entity, v, to the set unions of the INHERITED SLASH values (x), the INHERITED QUE values (y), and the INHERITED REL values (z) of its syntactic daughters.

COLLECT DEPENDENCIES PRINCIPLE

$$
\begin{array}{l}
\forall v\,\forall x\,\forall y\,\forall z \\
\left(\begin{array}{l}
\texttt{collect-dependencies}(v,x,y,z) \leftrightarrow \\
\left(\begin{array}{l}
\exists\boxed{1}\,\exists\boxed{2}\,\exists\boxed{3}\,\exists\boxed{4}\,\exists\boxed{5}\,\exists\boxed{6}\,\exists\boxed{7}\,\exists c_1\,\exists c_2\,\exists c_3 \\
\left(\begin{array}{l}
{}^{v}\left[\begin{array}{ll}\textit{head-comp-struc} & \\ \text{HEAD-DTR SS NONLOC INHERITED} & \left[\begin{array}{ll}\text{SLASH} & \boxed{1}\\ \text{QUE} & \boxed{2}\\ \text{REL} & \boxed{3}\end{array}\right]\\ \text{COMP-DTRS} & \boxed{4}\end{array}\right] \\
\wedge\ \texttt{enumerate-slashes}(\boxed{4},\boxed{5}) \wedge\ {}^{x}\left[\bigcup c_1 \ll \boxed{1} \mid \boxed{5} \gg\right] \\
\wedge\ \texttt{enumerate-ques}(\boxed{4},\boxed{6}) \wedge\ {}^{y}\left[\bigcup c_2 \ll \boxed{2} \mid \boxed{6} \gg\right] \\
\wedge\ \texttt{enumerate-rels}(\boxed{4},\boxed{7}) \wedge\ {}^{z}\left[\bigcup c_3 \ll \boxed{3} \mid \boxed{7} \gg\right]
\end{array}\right) \vee \\
\\
\exists\boxed{1}\,\exists\boxed{2}\,\exists\boxed{3}\,\exists\boxed{4}\,\exists\boxed{5}\,\exists\boxed{6} \\
\left(\begin{array}{l}
{}^{v}\left[\begin{array}{ll}\textit{head-adj-struc} & \\ \text{HEAD-DTR SS NONLOC INHERITED} & \left[\begin{array}{ll}\text{SLASH} & \boxed{1}\\ \text{QUE} & \boxed{2}\\ \text{REL} & \boxed{3}\end{array}\right]\\ \text{ADJUNCT-DTR SS NONLOC INHERITED} & \left[\begin{array}{ll}\text{SLASH} & \boxed{4}\\ \text{QUE} & \boxed{5}\\ \text{REL} & \boxed{6}\end{array}\right]\end{array}\right] \\
\wedge\ {}^{x}\left[\boxed{1} \cup \boxed{4}\right] \wedge\ {}^{y}\left[\boxed{2} \cup \boxed{5}\right] \wedge\ {}^{z}\left[\boxed{3} \cup \boxed{6}\right]
\end{array}\right) \vee \\
\\
\exists\boxed{1}\,\exists\boxed{2}\,\exists\boxed{3}\,\exists\boxed{4}\,\exists\boxed{5}\,\exists\boxed{6} \\
\left(\begin{array}{l}
{}^{v}\left[\begin{array}{ll}\textit{head-filler-struc} & \\ \text{HEAD-DTR SS NONLOC INHERITED} & \left[\begin{array}{ll}\text{SLASH} & \boxed{1}\\ \text{QUE} & \boxed{2}\\ \text{REL} & \boxed{3}\end{array}\right]\\ \text{FILLER-DTR SS NONLOC INHERITED} & \left[\begin{array}{ll}\text{SLASH} & \boxed{4}\\ \text{QUE} & \boxed{5}\\ \text{REL} & \boxed{6}\end{array}\right]\end{array}\right] \\
\wedge\ {}^{x}\left[\boxed{1} \cup \boxed{4}\right] \wedge\ {}^{y}\left[\boxed{2} \cup \boxed{5}\right] \wedge\ {}^{z}\left[\boxed{3} \cup \boxed{6}\right]
\end{array}\right) \vee \\
\\
\exists\boxed{1}\,\exists\boxed{2}\,\exists\boxed{3}\,\exists\boxed{4}\,\exists\boxed{5}\,\exists\boxed{6} \\
\left(\begin{array}{l}
{}^{v}\left[\begin{array}{ll}\textit{head-mark-struc} & \\ \text{HEAD-DTR SS NONLOC INHERITED} & \left[\begin{array}{ll}\text{SLASH} & \boxed{1}\\ \text{QUE} & \boxed{2}\\ \text{REL} & \boxed{3}\end{array}\right]\\ \text{MARKER-DTR SS NONLOC INHERITED} & \left[\begin{array}{ll}\text{SLASH} & \boxed{4}\\ \text{QUE} & \boxed{5}\\ \text{REL} & \boxed{6}\end{array}\right]\end{array}\right] \\
\wedge\ {}^{x}\left[\boxed{1} \cup \boxed{4}\right] \wedge\ {}^{y}\left[\boxed{2} \cup \boxed{5}\right] \wedge\ {}^{z}\left[\boxed{3} \cup \boxed{6}\right]
\end{array}\right)
\end{array}\right)
\end{array}\right)
\end{array}
$$

collect-dtrs-backgrounds/2

The relation `collect-dtrs-backgrounds` contains tuples of objects in the denotation of *headed-struc* with the set union of all the BACKGROUND sets of their daughters.

COLLECT DTRS BACKGROUNDS PRINCIPLE

$$\forall x\, \forall y \left(\begin{array}{l} \texttt{collect-dtrs-backgrounds}(x, y) \leftrightarrow \\ \left(\begin{array}{l} \exists \boxed{1}\, \exists \boxed{2}\, \exists \boxed{3} \\ \left(\begin{array}{l} {}^{x}\left[\begin{array}{l} \textit{head-comp-struc} \\ \text{HEAD-DTR SS LOC CONTEXT BACKGROUND } \boxed{1} \\ \text{COMP-DTRS } \boxed{2} \end{array}\right] \\ \wedge\ \texttt{enumerate-backgrounds}(\boxed{2}, \boxed{3}) \wedge\ {}^{y}[\bigcup z \ll \boxed{1} \mid \boxed{3} \gg] \end{array} \right) \vee \\ \\ \exists \boxed{1}\, \exists \boxed{2} \\ \left({}^{x}\left[\begin{array}{l} \textit{head-filler-struc} \\ \text{HEAD-DTR SS LOC CONTEXT BACKGROUND } \boxed{1} \\ \text{FILLER-DTR SS LOC CONTEXT BACKGROUND } \boxed{2} \end{array}\right] \wedge\ {}^{y}[\boxed{1} \cup \boxed{2}] \right) \vee \\ \\ \exists \boxed{1}\, \exists \boxed{2} \\ \left({}^{x}\left[\begin{array}{l} \textit{head-mark-struc} \\ \text{HEAD-DTR SS LOC CONTEXT BACKGROUND } \boxed{1} \\ \text{MARKER-DTR SS LOC CONTEXT BACKGROUND } \boxed{2} \end{array}\right] \wedge\ {}^{y}[\boxed{1} \cup \boxed{2}] \right) \vee \\ \\ \exists \boxed{1}\, \exists \boxed{2} \\ \left({}^{x}\left[\begin{array}{l} \textit{head-adj-struc} \\ \text{HEAD-DTR SS LOC CONTEXT BACKGROUND } \boxed{1} \\ \text{ADJUNCT-DTR SS LOC CONTEXT BACKGROUND } \boxed{2} \end{array}\right] \wedge\ {}^{y}[\boxed{1} \cup \boxed{2}] \right) \end{array} \right) \end{array} \right)$$

collect-dtrs-qstores/2

The relation `collect-dtrs-qstores` relates entities in the denotation of the sort *headed-struc* to a set that is the union of the QSTORE values of its daughters.

COLLECT DTRS QSTORES PRINCIPLE

$$\forall x\,\forall y \left(\begin{array}{l} \texttt{collect-dtrs-qstores}(x,y) \leftrightarrow \\ \quad \exists\boxed{1}\,\exists\boxed{2}\,\exists\boxed{3}\,\exists c \left(\begin{array}{l} {}^{x}\left[\begin{array}{l}\textit{head-comp-struc}\\ \text{HEAD-DTR QSTORE } \boxed{1}\\ \text{COMP-DTRS } \boxed{2}\end{array}\right] \\ \wedge\, \texttt{enumerate-qstores}(\boxed{2},\boxed{3}) \wedge\, {}^{y}[\bigcup c \ll \boxed{1} \mid \boxed{3} \gg] \end{array} \right) \vee \\ \quad \exists\boxed{1}\,\exists\boxed{2} \left(\begin{array}{l} {}^{x}\left[\begin{array}{l}\textit{head-adj-struc}\\ \text{HEAD-DTR QSTORE } \boxed{1}\\ \text{ADJUNCT-DTR QSTORE } \boxed{2}\end{array}\right] \\ \wedge\, {}^{y}[\boxed{1} \cup \boxed{2}] \end{array} \right) \vee \\ \quad \exists\boxed{1}\,\exists\boxed{2} \left(\begin{array}{l} {}^{x}\left[\begin{array}{l}\textit{head-filler-struc}\\ \text{HEAD-DTR QSTORE } \boxed{1}\\ \text{FILLER-DTR QSTORE } \boxed{2}\end{array}\right] \\ \wedge\, {}^{y}[\boxed{1} \cup \boxed{2}] \end{array} \right) \vee \\ \quad \exists\boxed{1}\,\exists\boxed{2} \left(\begin{array}{l} {}^{x}\left[\begin{array}{l}\textit{head-mark-struc}\\ \text{HEAD-DTR QSTORE } \boxed{1}\\ \text{MARKER-DTR QSTORE } \boxed{2}\end{array}\right] \\ \wedge\, {}^{y}[\boxed{1} \cup \boxed{2}] \end{array} \right) \end{array} \right)$$

enumerate-backgrounds/2

For a list of signs in its first argument, x, `enumerate-backgrounds` contains a list or a chain of the BACKGROUND values of the signs on x in its second argument, y.

ENUMERATE BACKGROUNDS PRINCIPLE

$$\forall x\,\forall y \left(\begin{array}{l} \texttt{enumerate-backgrounds}(x,y) \leftrightarrow \\ \quad \left(\begin{array}{l} (x\,\langle\rangle \wedge y \ll\gg) \vee \\ \exists\boxed{1}\,\exists\boxed{2}\,\exists\boxed{3} \\ \left(\begin{array}{l} x\left\langle \left[\text{SS LOC CONTEXT BACKGROUND } \boxed{1}\right] \middle| \boxed{2} \right\rangle \wedge y \ll \boxed{1} \mid \boxed{3} \gg \\ \wedge\, \texttt{enumerate-backgrounds}(\boxed{2},\boxed{3}) \end{array} \right) \end{array} \right) \end{array} \right)$$

enumerate-qstores/2

The relation `enumerate-qstores` relates a list of signs to a list or chain of the QSTORE values of these signs.

ENUMERATE QSTORES PRINCIPLE

$$\forall x\,\forall y \left(\begin{array}{l} \texttt{enumerate-qstores}(x,y) \leftrightarrow \\ \quad \left(\begin{array}{l} \left(x\,\langle\rangle \wedge y \ll\gg \right) \vee \\ \exists\boxed{1}\,\exists\boxed{2}\,\exists\boxed{3} \left(\begin{array}{l} x\left\langle \left[\text{QSTORE } \boxed{1}\right] \middle| \boxed{2} \right\rangle \wedge\ y \ll \boxed{1} \mid \boxed{3} \gg \\ \wedge\, \texttt{enumerate-qstores}(\boxed{2},\boxed{3}) \end{array} \right) \end{array} \right) \end{array} \right)$$

enumerate-ques/2

For a list of signs x in its first argument, `enumerate-ques` contains a list or a chain of the QUE values of the signs on x in its second argument.

ENUMERATE QUE PRINCIPLE

$$\forall x\, \forall y \left(\begin{array}{l} \texttt{enumerate-ques}(x, y) \leftrightarrow \\ \left(\begin{array}{l} (x \langle\rangle \wedge y \ll\gg) \vee \\ \exists\boxed{1}\, \exists\boxed{2}\, \exists\boxed{3} \\ \left(\begin{array}{l} x \left\langle \left[\text{SS NONLOC INHERITED QUE } \boxed{1} \right] \middle| \boxed{2} \right\rangle \wedge y \ll\boxed{1}\middle|\boxed{3}\gg \\ \wedge \texttt{enumerate-ques}(\boxed{2}, \boxed{3}) \end{array} \right) \end{array} \right) \end{array} \right)$$

enumerate-rels/2

For a list of signs x in its first argument, `enumerate-rels` contains a list or a chain of the REL values of the signs on x in its second argument.

ENUMERATE REL PRINCIPLE

$$\forall x\, \forall y \left(\begin{array}{l} \texttt{enumerate-rels}(x, y) \leftrightarrow \\ \left(\begin{array}{l} (x \langle\rangle \wedge y \ll\gg) \vee \\ \exists\boxed{1}\, \exists\boxed{2}\, \exists\boxed{3} \\ \left(\begin{array}{l} x \left\langle \left[\text{SS NONLOC INHERITED REL } \boxed{1} \right] \middle| \boxed{2} \right\rangle \wedge y \ll\boxed{1}\middle|\boxed{3}\gg \\ \wedge \texttt{enumerate-rels}(\boxed{2}, \boxed{3}) \end{array} \right) \end{array} \right) \end{array} \right)$$

enumerate-slashes/2

For a list of signs x in its first argument, `enumerate-slashes` contains a list or a chain of the SLASH values of the signs on x in its second argument.

ENUMERATE SLASH PRINCIPLE

$$\forall x\, \forall y \left(\begin{array}{l} \texttt{enumerate-slashes}(x, y) \leftrightarrow \\ \left(\begin{array}{l} (x \langle\rangle \wedge y \ll\gg) \vee \\ \exists\boxed{1}\, \exists\boxed{2}\, \exists\boxed{3} \\ \left(\begin{array}{l} x \left\langle \left[\text{SS NONLOC INHERITED SLASH } \boxed{1} \right] \middle| \boxed{2} \right\rangle \wedge y \ll\boxed{1}\middle|\boxed{3}\gg \\ \wedge \texttt{enumerate-slashes}(\boxed{2}, \boxed{3}) \end{array} \right) \end{array} \right) \end{array} \right)$$

index-of-a-psoa/2

The relation `index-of-a-psoa` is the set of tuples which, for each psoa object y, contains all pairs of y (in the second argument) and indices x (in the first argument) which are components of y. These are the indices that can be reached by some path starting from y.

The sets $\mathcal{S}$ and $\mathcal{A}$, and the function $\mathcal{F}$ to which the description refers are the set of sorts, the set of attributes, and the appropriateness function of the grammar of Pollard and Sag. The description uses an abbreviatory meta-notation for the (potentially very large) disjunction over appropriate descriptions of psoas with NUCLEUS values which have an index-valued attribute, and for the disjunction over appropriate descriptions of psoas with NUCLEUS values which have a psoa-valued attribute in which we might find further indices.

INDEX OF A PSOA PRINCIPLE

$$\forall x\, \forall y \left(\begin{array}{l} \texttt{index-of-a-psoa}(x,y) \leftrightarrow \\ \left(\begin{array}{l} \exists \boxed{1} \exists \boxed{2} \left(\begin{array}{l} {}^{y}\left[\textsc{quants}\ \boxed{1}\right] \\ \wedge\, \texttt{member}(\boxed{2},\boxed{1}) \\ \wedge\, \texttt{index-of-a-quantifier}(x,\boxed{2}) \end{array} \right) \vee \\ \\ \bigvee \left\{ {}^{y}\left[\textsc{nucleus}\ \begin{bmatrix}\sigma \\ \alpha\ \ x\end{bmatrix}\right] \middle| \begin{array}{l} \sigma \in \left\{ \sigma' \in \mathcal{S} \middle| \sigma' \sqsubseteq \mathit{qfpsoa} \right\}, \text{and} \\ \alpha \in \left\{ \alpha' \in \mathcal{A} \middle| \mathcal{F}\left\langle \sigma, \alpha' \right\rangle \sqsubseteq \mathit{index} \right\} \end{array} \right\} \vee \\ \\ \bigvee \left\{ \exists \boxed{1} \left(\begin{array}{l} {}^{y}\left[\textsc{nucleus}\ \begin{bmatrix}\sigma \\ \alpha\ \ \boxed{1}\end{bmatrix}\right] \\ \wedge\, \texttt{index-of-a-psoa}(x,\boxed{1}) \end{array} \right) \middle| \begin{array}{l} \sigma \in \left\{ \sigma' \in \mathcal{S} \middle| \sigma' \sqsubseteq \mathit{qfpsoa} \right\}, \text{and} \\ \alpha \in \left\{ \alpha' \in \mathcal{A} \middle| \mathcal{F}\left\langle \sigma, \alpha' \right\rangle \sqsubseteq \mathit{psoa} \right\} \end{array} \right\} \end{array} \right) \end{array} \right)$$

index-of-a-quantifier/2

The relation `index-of-a-quantifier` is the set of tuples which, for each quantifier object y, contains all pairs of y (in the second argument) and indices x (in the first argument) which are components of y. This means that there is a path that leads from y to x.

INDEX OF A QUANTIFIER PRINCIPLE

$$\forall x\, \forall y \left(\begin{array}{l} \texttt{index-of-a-quantifier}(x,y) \leftrightarrow \\ \left(\begin{array}{l} {}^{y}\left[\textsc{restind}\ \textsc{index}\ \ x\right] \vee \\ \\ \exists \boxed{1} \exists \boxed{2} \left(\begin{array}{l} {}^{y}\left[\textsc{restind}\ \textsc{restr}\ \boxed{1}\right] \\ \wedge\, \texttt{member}(\boxed{2},\boxed{1}) \\ \wedge\, \texttt{index-of-a-psoa}(x,\boxed{2}) \end{array} \right) \end{array} \right) \end{array} \right)$$

loc-o-bind/2

"One referential *synsem* object *(locally) o-binds* another provided it (locally) o-commands and is coindexed with the other. A referential *synsem* object is *(locally) o-free* provided it is not (locally) o-bound. Two *synsem* entities are *coindexed* provided their LOCAL | CONTENT | INDEX values are token-identical." [Pollard and Sag, 1994, p. 401]

LOCAL O BIND PRINCIPLE

$$\forall x\,\forall y\left(\begin{array}{l}\texttt{loc-o-bind}(x,y)\ \leftrightarrow\\ \exists\boxed{1}\,\texttt{loc-o-command}\Big(^{x}\big[\textsc{loc cont index}\ \boxed{1}\big],\,^{y}\big[\textsc{loc cont index}\ \boxed{1}\big]\Big)\end{array}\right)$$

loc-o-command/2

A referential *synsem* object *locally o-commands* another *synsem* object provided they have distinct LOCAL values and either (1) the second is more oblique than the first, or (2) the second is a member of the SUBCAT list of a *synsem* object that is more oblique than the first.

LOCAL O COMMAND PRINCIPLE

$$\forall x\,\forall y\left(\begin{array}{l}\texttt{loc-o-command}(x,y)\ \leftrightarrow\\ \exists\boxed{1}\,\exists\boxed{2}\,\exists\boxed{3}\,\exists s\\ \left(\begin{array}{l}\Big(^{x}\big[\textsc{loc}\ \boxed{1}\big[\textsc{cont index}\ \textit{ref}\big]\big]\wedge\,^{y}\big[\textsc{loc}\ \boxed{2}\big]\wedge\neg\boxed{1}=\boxed{2}\Big)\wedge\\ \left(\begin{array}{l}\texttt{more-oblique}(y,x)\vee\\ \Big(\texttt{more-oblique}\Big(^{s}\big[\begin{array}{l}\textit{synsem}\\ \textsc{loc cat subcat}\ \boxed{3}\end{array}\big],x\Big)\wedge\texttt{member}(y,\boxed{3})\Big)\end{array}\right)\end{array}\right)\end{array}\right)$$

member/2

The `member` relation is a relation that holds between an entity and a list, a set, or a chain, in case the entity is on the list, in the set, or on the chain. The description below uses a notation with a double bar to describe chains.

MEMBER PRINCIPLE

$$\forall x\,\forall y\left(\begin{array}{l}\texttt{member}(x,y)\leftrightarrow\\ \left(\begin{array}{l}y\,\|\,x|\,\textit{chain}\|\ \vee\ y\,\langle x|\,\textit{list}\rangle\ \vee\ ^{y}\big[\begin{array}{l}\textit{set}\\ \textsc{first}\ x\end{array}\big]\end{array}\right)\vee\\ \exists z\left(\begin{array}{l}(y\,\|\,\textit{object}|\,z\|\ \wedge\ \texttt{member}(x,z))\ \vee\\ (y\,\langle\,\textit{object}|\,z\rangle\ \wedge\ \texttt{member}(x,z))\ \vee\\ \Big(^{y}\big[\begin{array}{l}\textit{set}\\ \textsc{rest}\ z\end{array}\big]\wedge\texttt{member}(x,z)\Big)\end{array}\right)\end{array}\right)$$

more-oblique/2

"One *synsem* object is *more oblique* than another provided it appears to the right of the other on the SUBCAT list of some word." [Pollard and Sag, 1994, p. 401]

MORE OBLIQUE PRINCIPLE

$$\forall x\,\forall y\left(\texttt{more-oblique}(x,y)\ \leftrightarrow\exists w\,\exists\boxed{1}\left(\begin{array}{l}^{w}\big[\begin{array}{l}\textit{word}\\ \textsc{ss loc cat subcat}\ \boxed{1}\end{array}\big]\\ \wedge\ \texttt{to-the-right}(x,y,\boxed{1})\end{array}\right)\right)$$

o-bind/2

> "One referential *synsem* object *(locally) o-binds* another provided it (locally) o-commands and is coindexed with the other. A referential *synsem* object is *(locally) o-free* provided it is not (locally) o-bound. Two *synsem* entities are *coindexed* provided their LOCAL | CONTENT | INDEX values are token-identical." [Pollard and Sag, 1994, p. 401]

O Bind Principle

$$\forall x\,\forall y\left(\texttt{o-bind}(x,y) \leftrightarrow \exists\boxed{1}\,\texttt{o-command}\left(^{x}\left[\textsc{loc cont index}\ \boxed{1}\right],\ ^{y}\left[\textsc{loc cont index}\ \boxed{1}\right]\right)\right)$$

o-command/2

> A referential *synsem* object *o-commands* another *synsem* object provided they have distinct LOCAL values and either (1) the second is more oblique than the first, (2) the second is a member of the SUBCAT list of a *synsem* object that is o-commanded by the first, or (3) the second has the same LOCAL | CATEGORY | HEAD value as a *synsem* object that is o-commanded by the first.

O Command Principle

$$\forall x\,\forall y\left(\begin{array}{l}\texttt{o-command}(x,y)\ \leftrightarrow\\ \exists\boxed{1}\,\exists\boxed{2}\,\exists\boxed{3}\,\exists\boxed{4}\,\exists s_1\exists s_2\\ \left(\begin{array}{l}\left(^{x}\left[\textsc{loc}\ \boxed{1}\left[\textsc{cont index}\ \textit{ref}\right]\right]\wedge\ ^{y}\left[\textsc{loc}\ \boxed{2}\left[\textsc{cat head}\ \boxed{4}\right]\right]\wedge\neg\boxed{1}=\boxed{2}\right)\\ \wedge\\ \left(\begin{array}{l}\texttt{more-oblique}(y,x)\vee\\ \left(\texttt{o-command}\left(x,\ ^{s_1}\left[\begin{array}{l}\textit{synsem}\\ \textsc{loc cat subcat}\ \boxed{3}\end{array}\right]\right)\wedge\texttt{member}(y,\boxed{3})\right)\ \vee\\ \texttt{o-command}\left(x,\ ^{s_2}\left[\begin{array}{l}\textit{synsem}\\ \textsc{loc}\ \left[\textsc{cat head}\ \boxed{4}\right]\end{array}\right]\right)\end{array}\right)\end{array}\right)\end{array}\right)$$

set-difference/3 (\)

> The relation `set-difference` expresses the set difference of two sets. It is usually written in infix notation. Its formalization refers to a relation `set-properties` which spells out what it means for a configuration of entities in the denotation of the grammar to have set properties. We do not define it in this textbook. It is explained in [Richter, 2004, Chapter 4.4].

Set Difference Principle

$$\forall x\forall y\forall z\left(\begin{array}{l}\texttt{set-difference}(x,y,z)\leftrightarrow\\ \left(\begin{array}{l}\forall a\,(\texttt{member}(a,z)\ \leftrightarrow\ (\texttt{member}(a,x)\ \wedge\ \neg\texttt{member}(a,y)))\ \wedge\\ \texttt{set-properties}(^{x}[\textit{chain}\vee\textit{set}])\wedge\\ \texttt{set-properties}(^{y}[\textit{chain}\vee\textit{set}])\wedge\\ \texttt{set-properties}(^{z}[\textit{chain}\vee\textit{set}])\end{array}\right)\end{array}\right)$$

set-list-chain-member-equality/2 ($\stackrel{s}{=}$)

This relation, notated in infix notation as $\stackrel{s}{=}$, says that two entities which are chains, lists or sets, have the same members.

SET LIST CHAIN MEMBER EQUALITY PRINCIPLE

$$\forall x \forall y \left(\begin{array}{l} \texttt{set-list-chain-member-equality}(x, y) \leftrightarrow \\ \left(\begin{array}{l} \forall a \, (\texttt{member}(a, x) \leftrightarrow \texttt{member}(a, y)) \; \wedge \\ {}^{x}[chain \vee list \vee set] \; \wedge \; {}^{y}[chain \vee list \vee set] \end{array} \right) \end{array} \right)$$

set-union/3 ($\cup$)

The relation `set-union` expresses the set union of two sets. It is usually written in the usual infix notation. Its formalization refers to a relation `set-properties` which spells out what it means for a configuration of entities in the denotation of the grammar to have set properties. We do not define it in this textbook. It is explained in [Richter, 2004, Chapter 4.4].

SET UNION PRINCIPLE

$$\forall x \forall y \forall z \left(\begin{array}{l} \texttt{set-union}(x, y, z) \leftrightarrow \\ \left(\begin{array}{l} \forall a \, (\texttt{member}(a, z) \; \leftrightarrow \; (\texttt{member}(a, x) \; \vee \; \texttt{member}(a, y))) \; \wedge \\ \texttt{set-properties}({}^{x}[chain \vee set]) \wedge \\ \texttt{set-properties}({}^{y}[chain \vee set]) \wedge \\ \texttt{set-properties}({}^{z}[chain \vee set]) \end{array} \right) \end{array} \right)$$

sign-ss-append/3

`sign-ss-append` is a relation between three entities in a structure such that each triple for which the following is true is in it: The last argument, z, is obtained by appending the SYNSEM values of the list of signs in the second argument, y, in the order of the occurrence of the signs on y to the list in the first argument, x. Metaphorically, we could say that `sign-ss-append` extracts the SYNSEM values from a list in y and appends the resulting list of *synsem* entities to the list x to obtain list z. Of course, there is no process or directionality involved in this.

SIGN SYNSEM APPEND PRINCIPLE

$$\forall x \; \forall y \; \forall z \left(\begin{array}{l} \texttt{sign-ss-append}(x, y, z) \; \leftrightarrow \\ \\ \exists \boxed{1} \, \exists \boxed{2} \, \exists \boxed{3} \\ \left(\begin{array}{l} (x \, \langle \rangle \wedge y \, \langle \rangle \wedge z \, \langle \rangle) \; \vee \\ (x \, \langle \boxed{1} \mid \boxed{2} \rangle \wedge z \, \langle \boxed{1} \mid \boxed{3} \rangle \wedge \texttt{sign-ss-append}(\boxed{2}, y, \boxed{3})) \; \vee \\ \left(y \left\langle \left[\textsc{synsem} \; \boxed{1} \right] \middle| \boxed{2} \right\rangle \wedge z \, \langle \boxed{1} \mid \boxed{3} \rangle \wedge \texttt{sign-ss-append}(x \, \langle \rangle \, , \boxed{2}, \boxed{3}) \right) \end{array} \right) \end{array} \right)$$

successive-not-member/2

In the relation `successive-not-member` with arguments x and y, x is not an element of any of the sets (or lists or chains) on y.

Successive Not Member Principle

$$\forall x\,\forall y\left(\begin{array}{l}\texttt{successive-not-member}(x,y)\leftrightarrow\\ \left(\begin{array}{l}y\ll\gg\ \vee\\ \exists a\,\exists z\left(\ y\ll a|\,z\gg\wedge\neg\texttt{member}(x,a)\wedge\ \texttt{successive-not-member}(x,z)\ \right)\end{array}\right)\end{array}\right)$$

successive-union/2 ($\bigcup$)

`successive-union` is a binary relation whose first argument contains a list (or chain) of sets (or chains with set properties) and whose second argument contains the set union of these sets (or chains with set properties).

Successive Union Principle

$$\texttt{successive-union}(x,y)\leftrightarrow\left(\begin{array}{l}x\ll\gg\ \wedge\,^{y}[\mathit{echain}\ \vee\ \mathit{eset}]\vee\\ x\ll s|\,r\gg\ \wedge\texttt{successive-union}(r,z)\ \wedge\ \texttt{set-union}(s,z,y)\end{array}\right)$$

to-the-right/3

In the relation `to-the-right` with arguments x, y and z, x is to the right of y on list z.

To The Right Principle

$$\forall x\forall y\forall z\left(\texttt{to-the-right}(x,y,z)\leftrightarrow\left(\begin{array}{l}\exists\boxed{1}\ (z\,\langle y|\,\boxed{1}\rangle\wedge\ \texttt{member}(x,\boxed{1}))\\ \vee\exists\boxed{1}\ (z\,\langle\mathit{object}|\,\boxed{1}\rangle\wedge\ \texttt{to-the-right}(x,y,\boxed{1}))\end{array}\right)\right)$$

4.3 The ID Schemata

Abstract

This section contains the six ID Schemata *of the grammar of Pollard and Sag. Just as we did with the principles in the previous section, the informal version of the schemata is followed by a symbolization in a language of AVM matrices. Remember that the* ID Schemata *are to be read as the disjuncts in the consequent of the* ID Principle *listed among the principles of grammar in the previous section..*

The formalizations of the first two ID Schemata which follow each quote from the book by Pollard and Sag below also integrate additional restrictions which [Pollard and Sag, 1994, fn. 17, p. 402] mentions in a footnote. The idea is that these are language-specific restrictions of English which might not apply in variants of the relevant schemata in other

languages. The footnote says: "In the parochial versions of Schemata 1 and 2, the SYNSEM | LOCAL | CATEGORY | HEAD | INV value, if any, must be *minus*; in the parochial versions of Schema 3, it must be *plus*."

SCHEMA 1 (HEAD-SUBJECT SCHEMA)

> The SYNSEM | LOCAL | CATEGORY | SUBCAT value is $\langle\rangle$, and the DAUGHTERS value is an object of sort *head-comp-struc* whose HEAD-DAUGHTER is a phrase whose SYNSEM | NONLOCAL | TO-BIND | SLASH value is $\{\}$, and whose COMPLEMENT-DAUGHTERS value is a list of length one.

Formalization:

$$\begin{bmatrix} \text{SS LOC CAT} & \begin{bmatrix} \text{HEAD} & \Big[\big[\text{INV}\ \ \textit{minus}\big] \vee \neg\ \textit{verb}\Big] \\ \text{SUBCAT} & \langle\rangle \end{bmatrix} \\ \text{DTRS} & \begin{bmatrix} \textit{head-comp-struc} & \\ \text{HEAD-DTR} & \begin{bmatrix} \textit{phrase} \\ \text{SS NONLOC TO-BIND SLASH} \ \{\} \end{bmatrix} \\ \text{COMP-DAUGHTERS} & \langle \textit{object} \rangle \end{bmatrix} \end{bmatrix}$$

SCHEMA 2 (HEAD-COMPLEMENT SCHEMA)

> The SYNSEM | LOCAL | CATEGORY | SUBCAT value is a list of length one, and the daughters value is an object of sort *head-comp-struc* whose HEAD-DAUGHTER value is a word.

Formalization:

$$\begin{bmatrix} \text{SS LOC CAT} & \begin{bmatrix} \text{HEAD} & \Big[\big[\text{INV}\ \ \textit{minus}\big] \vee \neg\ \textit{verb}\Big] \\ \text{SUBCAT} & \langle \textit{object} \rangle \end{bmatrix} \\ \text{DTRS} & \begin{bmatrix} \textit{head-comp-struc} \\ \text{HEAD-DTR}\ \ \textit{word} \end{bmatrix} \end{bmatrix}$$

SCHEMA 3 (HEAD-SUBJECT-COMPLEMENT SCHEMA)

> The SYNSEM | LOCAL | CATEGORY | SUBCAT value is $\langle\rangle$, and the DAUGHTERS value is an object of sort *head-comp-struc* whose HEAD-DAUGHTER value is a word.

Formalization:

$$\begin{bmatrix} \text{SS LOC CAT} & \begin{bmatrix} \text{HEAD INV}\ \ \textit{plus} \\ \text{SUBCAT} \quad \langle\rangle \end{bmatrix} \\ \text{DTRS} & \begin{bmatrix} \textit{head-comp-struc} \\ \text{HEAD-DTR}\ \ \textit{word} \end{bmatrix} \end{bmatrix}$$

SCHEMA 4 (HEAD-MARKER SCHEMA)

> The DAUGHTERS value is an object of sort *head-marker-struc* whose HEAD-DAUGHTER | SYNSEM | NONLOCAL | TO-BIND | SLASH value is {}, and whose MARKER-DAUGHTER | SYNSEM | LOCAL | CATEGORY | HEAD value is of sort *marker*.

Formalization:

$$\begin{bmatrix} \text{DTRS} & \begin{bmatrix} \textit{head-marker-struc} \\ \text{HEAD-DTR SS NONLOC TO-BIND SLASH } \{\} \\ \text{MARKER-DTR SS LOC CAT HEAD } \textit{marker} \end{bmatrix} \end{bmatrix}$$

SCHEMA 5 (HEAD-ADJUNCT SCHEMA)

> The DAUGHTERS value is an object of sort *head-adjunct-struc* whose HEAD-DAUGHTER | SYNSEM value is token-identical to its ADJUNCT-DAUGHTER | SYNSEM | LOCAL | CATEGORY | HEAD | MOD value and whose HEAD-DAUGHTER | SYNSEM | NONLOCAL | TO-BIND | SLASH value is {}.

Formalization:

$$\begin{bmatrix} \text{DTRS} & \begin{bmatrix} \textit{head-adjunct-struc} \\ \text{HEAD-DTR SS } \boxed{1} \begin{bmatrix} \text{NONLOC TO-BIND SLASH } \{\} \end{bmatrix} \\ \text{ADJUNCT-DTR SS LOC CAT HEAD MOD } \boxed{1} \end{bmatrix} \end{bmatrix}$$

SCHEMA 6 (HEAD-FILLER SCHEMA)

> The DAUGHTERS value is an object of sort *head-filler-struc* whose HEAD-DAUGHTER | SYNSEM | LOCAL | CATEGORY value satisfies the description [HEAD *verb*[VFORM *finite*], SUBCAT ⟨⟩], whose HEAD-DAUGHTER | SYNSEM | NONLOCAL | INHERITED | SLASH value contains an element token-identical to the FILLER-DAUGHTER | SYNSEM | LOCAL value, and whose HEAD-DAUGHTER | SYNSEM | NONLOCAL | TO-BIND | SLASH value contains only that element.

Formalization:

$$\begin{bmatrix} \text{DTRS} & \begin{bmatrix} \textit{head-filler-struc} \\ \text{FILLER-DTR SS LOC } \boxed{1} \\ \text{HEAD-DTR SS} & \begin{bmatrix} \text{LOC CAT} & \begin{bmatrix} \text{HEAD VFORM } \textit{finite} \\ \text{SUBCAT } \langle\rangle \end{bmatrix} \\ \text{NONLOC} & \begin{bmatrix} \text{INHERITED SLASH } \boxed{2} \\ \text{TO-BIND SLASH } \{\boxed{1}\} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$\wedge$ `member`($\boxed{1}$, $\boxed{2}$)

4.4 The Raising Principle

Abstract

We will present the Raising Principle *and briefly discuss its special status in the grammar. It is not a principle in the technical sense, but rather a meta-principle for grammar writing.*

The Raising Principle is phrased as follows:

> Let E be a lexical entry in which the (description of the) SUBCAT list L contains (a description corresponding to) a member X (of L) that is not explicitly described in E as an expletive. Then in (the description of) the CONTENT value, X is (described as) assigned no semantic role if and only if L (is described as if it) contains a non-subject whose own SUBCAT value is ⟨X⟩.

Pollard and Sag list the Raising Principle separately from all their other principles. The reason for this is that the Raising Principle is not formulated as a principle on the shape of linguistic entities, but rather as a principle on the well-formedness of lexical entries. In other words, this principle is supposed to restrict the syntactic shape which descriptions of words might have in the grammar. It is thus a meta-principle, and cannot be expressed within the formalism itself. If one should want to formalize it at all, one would need a formalism which can talk about the descriptions in the theory of a grammar. As it stands, one can understand the Raising Principle as a rule for grammar writers which tells them how to write lexical entries.

Chapter 5

Resources: MoMo, TRALE, MERGE

Abstract

The purpose of this chapter is to serve as a springboard to the code and user's manuals of MoMo and the TRALE system, and as an additional pointer to the MERGE, a large grammar of English in TRALE. Section 5.1 contains the links to MoMo, and Section 5.2 provides access to everything TRALE-related. Section 5.3 is a link to MERGE.

5.1 MoMo

Abstract

The MorphMoulder (MoMo) is an implementation of Relational Speciate Re-entrant Language (RSRL) without chains. *RSRL is a comprehensive logical formalism for HPSG. MoMo is a tool for studying the syntax and semantics of RSRL by writing RSRL grammars and building models of grammars. MoMo can visualize models and relationships between signatures, descriptions and models.*

The MoMo manual and MoMo itself are available online and for downloading. The online edition of MoMo can be started directly from your Internet browser, as long as Java is active in your browser. Clicking on the link below will start MoMo immediately. You may use all features of MoMo with the online edition. However, without giving the online edition of MoMo explicit permission to read from and write to your local file system, MoMo cannot save files to or read files from your local directories.

If you choose to download MoMo, you may use the automatic installation program included in the package. The installation program will install MoMo under Solaris and under Linux systems, and will also give you the option of giving the online edition of MoMo permission to read and write in your directories. You may then choose between the online and offline edition any time you wish to run MoMo, and the online edition can be used with full functionality. Please consult the manual for details.

If you are not sure whether or not you want to install MoMo, we recommend that you try out the online edition first to find out if it is of interest to you.

- The MoMo Manual
  - Online Edition[1] (experimental version of March 2004)
  - Postcript[2] for downloading and printing (most recent version)
- The MoMo Program
  - Online Edition[3]
  - MoMo Code[4] for downloading and installing on your computer

5.2 TRALE

Abstract

The TRALE system is a complete grammar development environment written in SICStus Prolog. It comes with extensive documentation for the grammar writer, as well as for those interested in the TRALE system itself.

The TRALE manuals comprise four major documents: The *TRALE User's Manual* contains information on what TRALE is, how to install TRALE, and how to use TRALE and all of its components and various software tools linked to the TRALE system. The *TRALE Reference Manual* contains a specification of the data structures and algorithms of TRALE. The *TRALE User's Manual* and the *TRALE Reference Manual* are available as online editions in HTML format and in postscript format and PDF format for downloading. The *TRALE Source Code* contains the complete annotated source code of TRALE. The *ALE Source Code* contains the complete annotated source code of ALE, which is the core engine on which TRALE is built. The annotated sources are available in HTML format only.

The online editions of all these documents are connected by a system of hyperlinks, which help the reader to find corresponding topics and related information in the four documents. The online editions of the manuals use a color coding scheme to distinguish different kinds of information.

The second link below is for downloading a tared and gzipped directory file with the TRALE core system and the TRALE MiLCA Environment. Please consult the *TRALE User's Manual* on how to install TRALE, and for information concerning the system requirements of TRALE.

[1] `http://milca.sfs.uni-tuebingen.de/A4/Course/Momo/manual/momodocu.html`
[2] `http://milca.sfs.uni-tuebingen.de/A4/Course/Momo/manual/momo-manual.ps`
[3] `http://milca.sfs.uni-tuebingen.de/A4/Course/Momo/Online-Edition/MOMO/all_web.html`
[4] `http://milca.sfs.uni-tuebingen.de/A4/Course/Momo/download/momo.tar`

- The TRALE Manuals and Annotated Source Code[5]
- TRALE Sources with TRALE MiLCA Environment[6] for downloading and installing

5.3 MERGE

Abstract

The MERGE is a large grammar of English, provided here for studying grammar development in TRALE.

Some further information on MERGE may be found in Section 6.9 of the present textbook.

- File with the sources of MERGE[7]
- Documentation of the implementation of the MERGE grammar in Trale[8]

[5] http://www.ale.cs.toronto.edu/docs/
[6] http://milca.sfs.uni-tuebingen.de/A4/Course/trale/
[7] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/MERGE/merge-v-1-0-0.tar.gz
[8] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/MERGE/merge-doc.pdf

Chapter 6

Grammars

Abstract

This chapter contains the source code of all TRALE grammars which are discussed in Chapter 3, Grammar Implementation. *Since the grammars are extensively discussed in Chapter 3, there will be no further comment on them here. For each grammar there is a separate section for the signature and for the theory file.*

6.1 Grammar 1

Abstract

The first grammar licenses exactly one sentence, She walks. *It does not have any principles of grammar, instead it has a single phrase structure rule.*

6.1.1 Signature

Abstract

Note the link to a file with the source code at the bottom of the page.

```
type_hierarchy
bot
   list
     ne_list hd:bot tl:list
     e_list
   sign phon:ne_list synsem:synsem
      phrase dtr1:sign dtr2:sign dtrs:list
      word
   synsem category:cat content:cont context:conx
   cat head:head subcat:list
```

```
   head case:case vform:vform
   vform
      fin
      bse
   case
      nom
      acc
   cont relation:rel_name arg1:index arg2:index index:index
   conx backgr:list
   index person:person number:number gender:gender
   person
      first
      second
      third
   number
      sing
      plur
   gender
      masc
      fem
   rel_name
      walk
      female
.
```

Download[1]

6.1.2 Theory

Abstract

Note the link to a file with the source code at the bottom of the page.

```
% Multifile declarations.
:- multifile '##'/2.
:- multifile '~~>'/2.

% load phonology and tree output
:- [trale_home(tree_extensions)].

% specify signature file
signature(signature).
```

[1] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik1/signature

```
% phrase structure rule
subject_head_rule ##
(phrase, synsem:(category:(head:H,
                           subcat:[]),
                 content:Content),
 dtr1:Subj, dtr2:Head)
   ===>
cat> (Subj, synsem:Synsem),
cat> (Head, synsem:(category:(head:H,
      subcat:[Synsem]),
    content:Content)).

% lexical entries
she ~~> (synsem:(category: (head:case:nom,
                            subcat:e_list),
                 content: (index: (X, (person:third,
                                       number:sing,
                                       gender:fem))),
                 context: (backgr:[(relation:female,arg1:X)]))).

walks ~~> (synsem: (category: (head:vform:fin,
                               subcat:[(category:head:case:nom,
                                        content:index: (X,(person:third,
                                                           number:sing
                                                          ))))]),
                   content:(relation:walk,
                            arg1:X))).
```

Download[2]

[2] http:///milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik1/theory.pl

6.2 Grammar 2

Abstract

The second grammar is an extension of the first grammar. It increases the number of syntactic constructions, and it modifies the signature due to deficiencies observed with the signature of the first grammar.

6.2.1 Signature

Abstract

Note the link to a file with the source code at the bottom of the page.

```
type_hierarchy
bot
   list
     ne_list hd:bot tl:list
     e_list
   sign phon:ne_list synsem:synsem
      phrase dtr1:sign dtr2:sign dtrs:list
      word
   synsem category:cat content:cont context:conx
   cat head:head subcat:list
   head
      noun case:case
      verb vform:vform
   vform
      fin
      bse
   case
      nom
      acc
   cont
      relations arg1:index
          unary_rel
             walk_rel
             female_rel
          love_rel arg2:index
      nom_obj index:index
   conx backgr:list
   index person:person number:number gender:gender
   person
      first
```

```
      second
      third
   number
      sing
      plur
   gender
      masc
      fem
.
```

Download[3]

6.2.2 Theory

Abstract

Note the link to a file with the source code at the bottom of the page.

```
% Multifile declarations.
:- multifile '##'/2.
:- multifile '~~>'/2.

% load phonology and tree output
:- [trale_home(tree_extensions)].

% specify signature file
signature(signature).

% lexical entries
she ~~> (synsem:(category: (head:case:nom,
                            subcat:e_list),
                 content: (index: (X, (person:third,
                                       number:sing,
                                       gender:fem))),
                 context: (backgr:[(female_rel, arg1:X)]))).

her ~~> (synsem:(category: (head:case:acc,
                            subcat:e_list),
                 content: (index: (X, (person:third,
                                       number:sing,
                                       gender:fem))),
                 context: (backgr:[(female_rel, arg1:X)]))).
```

[3] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik2/signature

```
walks ~~> (synsem: (category: (head:vform:fin,
                               subcat:[(category:head:case:nom,
                                        content:index: (X,(person:third,
                                                           number:sing
                                                          ))),
                    content:(walk_rel,
                             arg1:X))).

loves ~~>  (synsem: (category: (head:vform:fin,
                               subcat:[(category:head:case:acc,
                                        content:index: Y),
                                       (category:head:case:nom,
                                        content:index: (X,
                                                        person:third,
                                                        number:sing))]),
                    content: (love_rel,arg1:X,arg2:Y))).

i ~~> (synsem:(category: (head:case:nom,
                          subcat:e_list),
                    content: (index:(person:first,
                                     number:sing)))).



walk ~~> (synsem: (category: (head:vform:fin,
                               subcat:[(category:head:case:nom,
                                        content:index: (X,(person:first,
                                                           number:sing
                                                          ))),
                    content:(walk_rel,
                             arg1:X))).


% phrase structure rules
subject_head_rule ##
(phrase, synsem:(category:(head:H,
                           subcat:[]),
                 content:Content),
 dtr1:Subj, dtr2:Head)
   ===>
cat> (Subj, synsem:Synsem),
cat> (Head, synsem:(category:(head:H,
```

```
      subcat:[Synsem]),
    content:Content)).


head_complement_rule ##
(phrase, synsem:(category:(head:H,
                           subcat:(ne_list, Rest)),
                 content:Content),
 dtr1:Comp, dtr2:Head)
   ===>
cat> (Head, synsem:(category:(head:H,
      subcat:[First|Rest]),
    content:Content)),
cat> (Comp, synsem:First).
```

Download[4]

6.3 Grammar 3

Abstract

The third grammar extends the lexicon of the previous grammar significantly. It is used for a discussion of properties of TRALE signatures, and of the relationship between phrase structure rules and grammatical principles in HPSG grammars in Chapter 3.

6.3.1 Signature

Abstract

Note the link to a file with the source code at the bottom of the page.

```
type_hierarchy
bot
   list
     ne_list hd:bot tl:list
     e_list
   sign phon:ne_list synsem:synsem
      phrase dtr1:sign dtr2:sign dtrs:list
      word
   synsem category:cat content:cont context:conx
   cat head:head subcat:list
```

[4] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik2/theory.pl

```
   head
      noun case:case
      verb vform:vform
   vform
      fin
      bse
   case
      nom
      acc
      dat
   cont
      relations arg1:index
        unary_rel
           walk_rel
           female_rel
           speaker_rel
        more_arg_rel arg2:index
           love_rel
           give_rel arg3:index
      nom_obj index:index
   conx backgr:list
   index person:person number:number gender:gender
   person
      first
      third
   number
      sing
      plur
   gender
      masc
      fem
.
```

Download[5]

6.3.2 Theory

Abstract

Note the link to a file with the source code at the bottom of the page.

```
% Multifile declarations.
```

[5] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik3/signature

```
:- multifile '##'/2.
:- multifile '~~>'/2.

% load phonology and tree output
:- [trale_home(tree_extensions)].

% specify signature file
signature(signature).

% lexical entries
i ~~>     synsem:(category:(head:case:nom,
                            subcat:e_list),
                  content:(index: (X,(person:first,
                                      number:sing))),
                  context:(backgr:[(speaker_rel,arg1:X)])).

me ~~>    synsem:(category:(head:case:(acc;dat),
                             subcat:e_list),
                   content:(index: (X,(person:first,
                                       number:sing))),
                   context:(backgr:[(speaker_rel,arg1:X)])).

she ~~>   synsem:(category:(head:case:nom,
                            subcat:e_list),
                  content:(index: (X,(person:third,
                                      number:sing,
                                      gender:fem))),
                  context:(backgr:[(female_rel,arg1:X)])).

her ~~>   synsem:(category:(head:case:(acc;dat),
                             subcat:e_list),
                   content:(index: (X,(person:third,
                                       number:sing,
                                       gender:fem))),
                   context:(backgr:[(female_rel,arg1:X)])).

milk ~~>  synsem:(category:(head:noun,
                            subcat:e_list),
                  content:(index:(person:third,
                                  number:sing))).

walk ~~>  synsem:(category:(head:vform:fin,
                             subcat:[(category:(head:case:nom),
```

```
                                        content:index:(X,
                                                       person:first,
                                                       number:sing)
                                                       )]),
                        content:(walk_rel,
                                 arg1:X)).

walks ~~>  synsem:(category:(head:vform:fin,
                             subcat:[(category:(head:case:nom),
                                      content:index:(X,
                                                     person:third,
                                                     number:sing)
                                                     )]),
                        content:(walk_rel,
                                 arg1:X)).

love ~~> synsem:(category:(head:vform:fin,
                             subcat:[(category:(head:case:acc),
                                      content:index: Y),
                                     (category:(head:case:nom),
                                      content:index: (X,
                                                      person:first,
                                                      number:sing)
                                                      )]),
                        content:(love_rel,
                                 arg1:X,
                                 arg2:Y)).

loves  ~~> synsem:(category:(head:vform:fin,
                             subcat:[(category:(head:case:acc),
                                      content:index: Y),
                                     (category:(head:case:nom),
                                      content:index: (X,
                                                      person:third,
                                                      number:sing)
                                                      )]),
                        content:(love_rel,
                                 arg1:X,
                                 arg2:Y)).

give ~~> synsem:(category:(head:vform:fin,
                             subcat:[(category:(head:case:dat),
                                      content:index: Z),
```

```
                                        (category:(head:case:acc),
                                         content:index: Y),
                                        (category:(head:case:nom),
                                         content:index: (X,
                                                         person:first,
                                                         number:sing)
                                                         )]),
                        content:(give_rel,
                                 arg1:X,
                                 arg2:Y,
                                 arg3:Z)).

gives ~~>  synsem:(category:(head:vform:fin,
                             subcat:[(category:(head:case:dat),
                                      content:index: Z),
                                     (category:(head:case:acc),
                                      content:index: Y),
                                     (category:(head:case:nom),
                                      content:index: (X,
                                                      person:third,
                                                      number:sing)
                                                      )]),
                        content:(give_rel,
                                 arg1:X,
                                 arg2:Y,
                                 arg3:Z)).


% phrase structure rules
subject_head_rule ##
(phrase, synsem:(category:(head:H,
                           subcat:[]),
                 content:Content),
 dtr1:Subj, dtr2:Head)
   ===>
cat> (Subj, synsem:Synsem),
cat> (Head, synsem:(category:(head:H,
          subcat:[Synsem]),
content:Content)).


head_complement_rule ##
(phrase, synsem:(category:(head:H,
```

```
                          subcat:(ne_list, Rest)),
                content:Content),
 dtr1:Comp, dtr2:Head)
   ===>
cat> (Head, synsem:(category:(head:H,
subcat:[First|Rest]),
content:Content)),
cat> (Comp, synsem:First).
```

Download[6]

6.4 Grammar 4

Abstract

The fourth grammar is in fact a series of three grammars. All three grammars have exactly the same empirical coverage, but the grammars are formulated differently and are progressively apt for further extensions. They are the first grammars which contain grammatical principles besides phrase structure rules.

6.4.1 Version 1

Abstract

This grammar is essentially the same as the third grammar, except that the grammatical principles previously built into the phrase structure rules have been pulled out of them and have been formulated as separate principles.

6.4.1.1 Signature

Abstract

Note the link to a file with the source code at the bottom of the page.

```
type_hierarchy
bot
   list
     ne_list hd:bot tl:list
     e_list
   sign phon:ne_list synsem:synsem
      phrase dtr1:sign dtr2:sign dtrs:list
      word
```

[6] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik3/theory.pl

```
synsem category:cat content:cont context:conx
cat head:head subcat:list
head
   noun case:case
   verb vform:vform
vform
   fin
   bse
case
   nom
   acc
   dat
cont
   nom_obj index:index
   arg
     index
     relations arg1:arg
       un_rel
         walk_rel
         female_rel
         speaker_rel
       more_arg_rel arg2:arg
         bin_rel
           love_rel
           think_rel
         give_rel arg3:arg
conx backgr:list
index person:person number:number gender:gender
person
   first
   third
number
   sing
   plur
gender
   masc
   fem
```

.

Download[7]

[7] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik4/1vers/signature

6.4.1.2 Theory

Abstract

Note the link to a file with the source code at the bottom of the page.

```
% Multifile declarations.
:- multifile '##'/2.
:- multifile '*>'/2.
:- multifile '~~>'/2.

% load phonology and tree output
:- [trale_home(tree_extensions)].


% specifications for the GRiSU output display
hidden_feat(dtrs).
hidden_feat(dtr1).
hidden_feat(dtr2).

% specify signature file
signature(signature).


% lexical entries

i ~~> (word, synsem:(category:(head:case:nom,
                               subcat:e_list),
                     content:(index: (X,(person:first,
                                         number:sing))),
                     context:(backgr:[(speaker_rel,arg1:X)]))).

me ~~> (word, synsem:(category:(head:case:(acc;dat),
                                subcat:e_list),
                      content:(index: (X,(person:first,
                                          number:sing))),
                      context:(backgr:[(speaker_rel,arg1:X)]))).

she ~~> (word, synsem:(category:(head:case:nom,
                                 subcat:e_list),
                       content:(index: (X,(person:third,
                                           number:sing,
                                           gender:fem))),
                       context:(backgr:[(female_rel,arg1:X)]))).
```

```
her ~~> (word, synsem:(category:(head:case:(acc;dat),
                                 subcat:e_list),
                        content:(index: (X,(person:third,
                                            number:sing,
                                            gender:fem))),
                        context:(backgr:[(female_rel,arg1:X)]))).

milk ~~> (word, synsem:(category:(head:noun,
                                  subcat:e_list),
                        content:(index:(person:third,
                                        number:sing)))).

walk ~~>  (word, synsem:(category:(head:vform:fin,
                                   subcat:[(category:(head:case:nom),
                                            content:index:(X,
                                                           person:first,
                                                           number:sing)
                                            )]),
                         content:(walk_rel,
                                  arg1:X))).

walks ~~>  (word, synsem:(category:(head:vform:fin,
                                   subcat:[(category:(head:case:nom),
                                            content:index:(X,
                                                           person:third,
                                                           number:sing)
                                            )]),
                         content:(walk_rel,
                                  arg1:X))).

love ~~> (word, synsem:(category:(head:vform:fin,
                                  subcat:[(category:(head:case:acc),
                                           content:index: Y),
                                          (category:(head:case:nom),
                                           content:index: (X,
                                                           person:first,
                                                           number:sing)
                                           )]),
                         content:(love_rel,
                                  arg1:X,
                                  arg2:Y))).
```

```
loves  ~~> (word, synsem:(category:(head:vform:fin,
                            subcat:[(category:(head:case:acc),
                                     content:index: Y),
                                    (category:(head:case:nom),
                                     content:index: (X,
                                                     person:third,
                                                     number:sing)
                                                     )]),
                    content:(love_rel,
                             arg1:X,
                             arg2:Y))).

give ~~> (word, synsem:(category:(head:vform:fin,
                            subcat:[(category:(head:case:dat),
                                     content:index: Z),
                                    (category:(head:case:acc),
                                     content:index: Y),
                                    (category:(head:case:nom),
                                     content:index: (X,
                                                     person:first,
                                                     number:sing)
                                                     )]),
                    content:(give_rel,
                             arg1:X,
                             arg2:Y,
                             arg3:Z))).

gives ~~>  (word, synsem:(category:(head:vform:fin,
                            subcat:[(category:(head:case:dat),
                                     content:index: Z),
                                    (category:(head:case:acc),
                                     content:index: Y),
                                    (category:(head:case:nom),
                                     content:index: (X,
                                                     person:third,
                                                     number:sing)
                                                     )]),
                    content:(give_rel,
                             arg1:X,
                             arg2:Y,
                             arg3:Z))).

think  ~~>  (word, synsem:(category:(head:vform:fin,
```

```
                                subcat:[(category:(head:vform:fin,
 subcat:[]),
                                         content: Y),
                                        (category:(head:case:nom),
                                         content:index: (X,
                                                         person:first,
                                                         number:sing)
                                                         )]),
                     content:(think_rel,
                              arg1:X,
                              arg2:Y))).

thinks ~~>  (word, synsem:(category:(head:vform:fin,
                                subcat:[(category:(head:vform:fin,
 subcat:[]),
                                         content: Y),
                                        (category:(head:case:nom),
                                         content:index: (X,
                                                         person:third,
                                                         number:sing)
                                                         )]),
                     content:(think_rel,
                              arg1:X,
                              arg2:Y))).


% phrase structure rules

subject_head_rule ##
(phrase, synsem:category:subcat:[],
            dtr1:Subj, dtr2:Head)
   ===>
cat> Subj,
cat> Head.

head_complement_rule ##
(phrase, synsem:category:subcat:ne_list,
            dtr1:Comp, dtr2:Head)
   ===>
cat> Head,
cat> Comp.
```

```
% Principles

% Semantics Principle
phrase *> (synsem:content:C,
           dtr2:synsem:content:C).

% Head Feature Principle
phrase *> (synsem:category:head:H,
           dtr2:synsem:category:head:H).

% Subcategorization Principle (first version)
phrase *> (synsem:category:subcat:PhrSubcat,
           dtr1:synsem:Synsem,
           dtr2:synsem:category:subcat:[Synsem|PhrSubcat]).
```

Download[8]

6.4.2 Version 2

Abstract

The Subcategorization Principle *of the first version of the fourth grammar is now reformulated with a relational attachment.*

6.4.2.1 Signature

Abstract

Note the link to a file with the source code at the bottom of the page.

```
type_hierarchy
bot
   list
     ne_list hd:bot tl:list
     e_list
   sign phon:ne_list synsem:synsem
      phrase dtr1:sign dtr2:sign dtrs:list
      word
   synsem category:cat content:cont context:conx
   cat head:head subcat:list
   head
      noun case:case
      verb vform:vform
```

[8] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik4/1vers/theory.pl

```
vform
   fin
   bse
case
   nom
   acc
   dat
cont
   nom_obj index:index
   arg
     index
     relations arg1:arg
       un_rel
         walk_rel
         female_rel
         speaker_rel
       more_arg_rel arg2:arg
         bin_rel
           love_rel
           think_rel
         give_rel arg3:arg
conx backgr:list
index person:person number:number gender:gender
person
   first
   third
number
   sing
   plur
gender
   masc
   fem
.
```

Download[9]

6.4.2.2 Theory

Abstract

Note the link to a file with the source code at the bottom of the page.

[9] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik4/2vers/signature

```
% Multifile declarations.
:- multifile '##'/2.
:- multifile '*>'/2.
:- multifile '~~>'/2.
:- multifile 'if'/2.

% load phonology and tree output
:- [trale_home(tree_extensions)].

% specifications for the GRiSU output display
hidden_feat(dtrs).
hidden_feat(dtr1).
hidden_feat(dtr2).

% specify signature file
signature(signature).


% lexical entries

i ~~> (word, synsem:(category:(head:case:nom,
                               subcat:e_list),
                     content:(index: (X,(person:first,
                                         number:sing))),
                     context:(backgr:[(speaker_rel,arg1:X)]))).

me ~~> (word, synsem:(category:(head:case:(acc;dat),
                                subcat:e_list),
                      content:(index: (X,(person:first,
                                          number:sing))),
                      context:(backgr:[(speaker_rel,arg1:X)]))).

she ~~> (word, synsem:(category:(head:case:nom,
                                 subcat:e_list),
                       content:(index: (X,(person:third,
                                           number:sing,
                                           gender:fem))),
                       context:(backgr:[(female_rel,arg1:X)]))).

her ~~> (word, synsem:(category:(head:case:(acc;dat),
                                 subcat:e_list),
                       content:(index: (X,(person:third,
                                           number:sing,
```

```
                                        gender:fem))),
                    context:(backgr:[(female_rel,arg1:X)]))).

milk ~~> (word, synsem:(category:(head:noun,
                                subcat:e_list),
                    content:(index:(person:third,
                                    number:sing)))).

walk ~~>  (word, synsem:(category:(head:vform:fin,
                                subcat:[(category:(head:case:nom),
                                         content:index:(X,
                                                        person:first,
                                                        number:sing)
                                                        )]),
                    content:(walk_rel,
                                arg1:X))).

walks ~~>  (word, synsem:(category:(head:vform:fin,
                                subcat:[(category:(head:case:nom),
                                         content:index:(X,
                                                        person:third,
                                                        number:sing)
                                                        )]),
                    content:(walk_rel,
                                arg1:X))).

love ~~> (word, synsem:(category:(head:vform:fin,
                                subcat:[(category:(head:case:acc),
                                         content:index: Y),
                                        (category:(head:case:nom),
                                         content:index: (X,
                                                         person:first,
                                                         number:sing)
                                                         )]),
                    content:(love_rel,
                                arg1:X,
                                arg2:Y))).

loves  ~~> (word, synsem:(category:(head:vform:fin,
                                subcat:[(category:(head:case:acc),
                                         content:index: Y),
                                        (category:(head:case:nom),
                                         content:index: (X,
```

```
                                                    person:third,
                                                    number:sing)
                                                    )]),
                    content:(love_rel,
                             arg1:X,
                             arg2:Y))).

give ~~> (word, synsem:(category:(head:vform:fin,
                                  subcat:[(category:(head:case:dat),
                                           content:index: Z),
                                          (category:(head:case:acc),
                                           content:index: Y),
                                          (category:(head:case:nom),
                                           content:index: (X,
                                                    person:first,
                                                    number:sing)
                                                    )]),
                    content:(give_rel,
                             arg1:X,
                             arg2:Y,
                             arg3:Z))).

gives ~~>  (word, synsem:(category:(head:vform:fin,
                                  subcat:[(category:(head:case:dat),
                                           content:index: Z),
                                          (category:(head:case:acc),
                                           content:index: Y),
                                          (category:(head:case:nom),
                                           content:index: (X,
                                                    person:third,
                                                    number:sing)
                                                    )]),
                    content:(give_rel,
                             arg1:X,
                             arg2:Y,
                             arg3:Z))).

think  ~~>  (word, synsem:(category:(head:vform:fin,
                                   subcat:[(category:(head:vform:fin,
 subcat:[]),
                                           content: Y),
                                          (category:(head:case:nom),
                                           content:index: (X,
```

```
                                                        person:first,
                                                        number:sing)
                                                        )]),
                    content:(think_rel,
                             arg1:X,
                             arg2:Y))).

thinks ~~>  (word, synsem:(category:(head:vform:fin,
                                subcat:[(category:(head:vform:fin,
 subcat:[]),
                                         content: Y),
                                        (category:(head:case:nom),
                                         content:index: (X,
                                                        person:third,
                                                        number:sing)
                                                        )]),
                    content:(think_rel,
                             arg1:X,
                             arg2:Y))).


% phrase structure rules

subject_head_rule ##
(phrase, synsem:category:subcat:[],
            dtr1:Subj, dtr2:Head)
   ===>
cat> Subj,
cat> Head.

head_complement_rule ##
(phrase, synsem:category:subcat:ne_list,
            dtr1:Comp, dtr2:Head)
   ===>
cat> Head,
cat> Comp.


% Principles

% Semantics Principle
phrase *> (synsem:content:C,
           dtr2:synsem:content:C).
```

```
% Head Feature Principle
phrase *> (synsem:category:head:H,
           dtr2:synsem:category:head:H).

% Subcategorization Principle (second version)
phrase *> (synsem:category:subcat:PhrSubcat,
           dtr1:synsem:Synsem,
           dtr2:synsem:category:subcat:HeadSubcat)
goal
   append([Synsem],PhrSubcat,HeadSubcat).


% Goal definitions
append(L,[],L) if true.
append([],(L,ne_list),L) if true.
append([H|T1],(L,ne_list),[H|T2]) if
  append(T1,L,T2).
```

Download[10]

6.4.3 Version 3

Abstract

Finally, the order of the elements on the SUBCAT *list is reversed to conform to the order of elements usually assumed in the linguistic literature. This leads to further computational complication and a second reformulation of the* Subcategorization Principle.

6.4.3.1 Signature

Abstract

Note the link to a file with the source code at the bottom of the page.

```
type_hierarchy
bot
   list
     ne_list hd:bot tl:list
     e_list
   sign phon:ne_list synsem:synsem
      phrase dtr1:sign dtr2:sign dtrs:list
```

[10] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik4/2vers/theory.pl

```
   word
synsem category:cat content:cont context:conx
cat head:head subcat:list
head
   noun case:case
   verb vform:vform
vform
   fin
   bse
case
   nom
   acc
   dat
cont
   nom_obj index:index
   arg
     index
     relations arg1:arg
       un_rel
         walk_rel
         female_rel
         speaker_rel
       more_arg_rel arg2:arg
         bin_rel
           love_rel
           think_rel
         give_rel arg3:arg
conx backgr:list
index person:person number:number gender:gender
person
   first
   third
number
   sing
   plur
gender
   masc
   fem
.
```

Download[11]

[11] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik4/3vers/signature

6.4.3.2 Theory

Abstract

Note the link to a file with the source code at the bottom of the page.

```
% Multifile declarations.
:- multifile '##'/2.
:- multifile '*>'/2.
:- multifile '~~>'/2.
:- multifile if/2.

% load phonology and tree output
:- [trale_home(tree_extensions)].

% specifications for the GRiSU output display
hidden_feat(dtrs).
hidden_feat(dtr1).
hidden_feat(dtr2).

% specify signature file
signature(signature).


% lexical entries

i ~~> (word, synsem:(category:(head:case:nom,
                               subcat:e_list),
                     content:(index: (X,(person:first,
                                         number:sing))),
                     context:(backgr:[(speaker_rel,arg1:X)]))).

me ~~> (word, synsem:(category:(head:case:(acc;dat),
                                subcat:e_list),
                      content:(index: (X,(person:first,
                                          number:sing))),
                      context:(backgr:[(speaker_rel,arg1:X)]))).

she ~~> (word, synsem:(category:(head:case:nom,
                                 subcat:e_list),
                       content:(index: (X,(person:third,
                                           number:sing,
                                           gender:fem))),
                       context:(backgr:[(female_rel,arg1:X)]))).
```

```
her ~~> (word, synsem:(category:(head:case:(acc;dat),
                                 subcat:e_list),
                       content:(index: (X,(person:third,
                                           number:sing,
                                           gender:fem))),
                       context:(backgr:[(female_rel,arg1:X)]))).

milk ~~> (word, synsem:(category:(head:noun,
                                  subcat:e_list),
                        content:(index:(person:third,
                                        number:sing)))).

walk ~~>  (word, synsem:(category:(head:vform:fin,
                                   subcat:[(category:(head:case:nom),
                                            content:index:(X,
                                                           person:first,
                                                           number:sing)
                                            )]),
                         content:(walk_rel,
                                  arg1:X))).

walks ~~>  (word, synsem:(category:(head:vform:fin,
                                   subcat:[(category:(head:case:nom),
                                            content:index:(X,
                                                           person:third,
                                                           number:sing)
                                            )]),
                         content:(walk_rel,
                                  arg1:X))).

love ~~> (word, synsem:(category:(head:vform:fin,
                                  subcat:[(category:(head:case:nom),
                                           content:index: (X,
                                                           person:first,
                                                           number:sing)),
                                          (category:(head:case:acc),
                                           content:index: Y)]),
                         content:(love_rel,
                                  arg1:X,
                                  arg2:Y))).

loves  ~~> (word, synsem:(category:(head:vform:fin,
```

```
                              subcat:[(category:(head:case:nom),
                                       content:index: (X,
                                                       person:third,
                                                       number:sing)),
                                      (category:(head:case:acc),
                                       content:index: Y)]),
                     content:(love_rel,
                              arg1:X,
                              arg2:Y))).

give ~~> (word, synsem:(category:(head:vform:fin,
                              subcat:[(category:(head:case:nom),
                                       content:index: (X,
                                                       person:first,
                                                       number:sing)),
                                      (category:(head:case:acc),
                                       content:index: Y),
                                      (category:(head:case:dat),
                                       content:index: Z)]),
                     content:(give_rel,
                              arg1:X,
                              arg2:Y,
                              arg3:Z))).

gives ~~>  (word, synsem:(category:(head:vform:fin,
                              subcat:[(category:(head:case:nom),
                                       content:index: (X,
                                                       person:third,
                                                       number:sing)),
                                      (category:(head:case:acc),
                                       content:index: Y),
                                      (category:(head:case:dat),
                                       content:index: Z)]),
                      content:(give_rel,
                              arg1:X,
                              arg2:Y,
                              arg3:Z))).

think  ~~>  (word, synsem:(category:(head:vform:fin,
                               subcat:[(category:(head:case:nom),
                                       content:index: (X,
                                                       person:first,
                                                       number:sing)),
```

```
                                        (category:(head:vform:fin,
 subcat:[]),
                                        content: Y)]),
                     content:(think_rel,
                              arg1:X,
                              arg2:Y))).

thinks ~~>  (word, synsem:(category:(head:vform:fin,
                                subcat:[(category:(head:case:nom),
                                        content:index: (X,
                                                        person:third,
                                                        number:sing)),
                                        (category:(head:vform:fin,
 subcat:[]),
                                        content: Y)]),
                     content:(think_rel,
                              arg1:X,
                              arg2:Y))).


% phrase structure rules
subject_head_rule ##
(phrase, synsem:category:subcat:[],
            dtr1:Subj, dtr2:Head)
   ===>
cat> Subj,
cat> Head.

head_complement_rule ##
(phrase, synsem:category:subcat:ne_list,
            dtr1:Comp, dtr2:Head)
   ===>
cat> Head,
cat> Comp.


% Principles

% Semantics Principle
phrase *> (synsem:content:C,
 dtr2:synsem:content:C).

% Head Feature Principle
```

```
phrase *> (synsem:category:head:H,
 dtr2:synsem:category:head:H).

% Subcategorization Principle (alternate version)
phrase *> (synsem:category:subcat:PhrSubcat,
           dtr1:synsem:Synsem,
           dtr2:synsem:category:subcat:HeadSubcat)
goal
   append(PhrSubcat,[Synsem],HeadSubcat).


% Goal definitions

append(X,Y,Z) if
   when(  ( X=(e_list;ne_list)
          ; Y=e_list
          ; Z=(e_list;ne_list)
               ),
           undelayed_append(X,Y,Z)).

undelayed_append(L,[],L) if true.
undelayed_append([],(L,ne_list),L) if true.
undelayed_append([H|T1],(L,ne_list),[H|T2]) if
  append(T1,L,T2).
```

Download[12]

6.5 Spook

Abstract

> *The spook grammar contains a (small) number of mistakes which are the subject of the exercise at the end of Section 3.1.4,* Relations as Definite Clauses in TRALE.

6.5.1 Signature

Abstract

> *Note the link to a file with the source code at the bottom of the page.*

[12] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Grammatik4/3vers/theory.pl

```
type_hierarchy
bot
   list
     ne_list hd:bot tl:list
     e_list
   sign phon:ne_list synsem:synsem
      phrase dtr1:sign dtr2:sign dtrs:list
      word
   synsem category:cat content:cont context:conx
   cat head:head subcat:list
   head
      noun case:case
      verb vform:vform
   vform
      fin
      bse
   case
      nom
      acc
      dat
   cont
      relations arg1:index
        unary_rel
           walk_rel
           female_rel
           speaker_rel
        more_arg_rel arg2:index
           love_rel
           give_rel arg3:index
      nom_obj index:index
   conx backgr:list
   index person:person number:number gender:gender
   person
      first
      third
   number
      sing
      plur
   gender
      masc
      fem
.
```

Download[13]

6.5.2 Theory

Abstract

Note the link to a file with the source code at the bottom of the page.

```
% Multifile declarations.
:- multifile '##'/2.
:- multifile '~~>'/2.

% load phonology and tree output
:- [trale_home(tree_extensions)].

% specify signature file
signature(signature).

% lexical entries
i ~~>     synsem:(category:(head:case:nom,
                            subcat:e_list),
                  content:(index: (X,(person:first,
                                      number:sing))),
                  context:(backgr:[(speaker_rel,arg1:X)])).

me ~~>    synsem:(category:(head:case:(acc;dat),
                            subcat:e_list),
                   content:(index: (X,(person:first,
                                       number:sing))),
                   context:(backgr:[(speaker_rel,arg1:X)])).

she ~~>   synsem:(category:(head:case:nom,
                            subcat:e_list),
                  content:(index: (X,(person:third,
                                      number:sing;
                                      gender:fem))),
                  context:(backgr:[(female_rel,arg1:X)])).

her ~~>   synsem:(category:(head:case:(acc;dat),
                            subcat:e_list),
                   content:(index: (X,(person:third,
                                       number:sing,
```

[13] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Spook/signature

```
                                        gender:fem))),
                    context:(backgr:[(female_rel,arg1:X)])).

milk ~~>  synsem:(category:(head:noun,
                            subcat:e_list),
                    content:(index:(person:third,
                                    number:sing))).

walk ~~>  synsem:(category:(head:vform:fin,
                             subcat:[(category:(head:case:nom),
                                      content:index:(X,
                                                     person:first,
                                                     number:sng)
                                                     )]),
                    content:(walk_rel,
                             arg1:X)).

walks ~~>  synsem:(category:(head:vform:fin,
                             subcat:[(category:(head:case:nom),
                                      content:index:(X,
                                                     person:third,
                                                     number:sing)
                                                     )]),
                    content:(walk_rel,
                             arg1:X)).

love ~~> synsem:(category:(head:vform:fin,
                             subcat:[(category:(head:case:acc),
                                      content:index: Y),
                                     (category:(head:case:nom),
                                      content:index: (X,
                                                      person:first,
                                                      number:sing)
                                                      )]),
                    content:(love_rel,
                             arg1:X,
                             arg2:Y)).

loves  ~~> synsem:(category:(head:vform:fin,
                             subcat:[(category:(head:case:acc),
                                      content:index: Y),
                                     (category:(head:case:nom),
                                      content:index: (X,
```

```
                                                    person:third,
                                                    number:sing)
                                                    )]),
                    content:(love_rel,
                             arg1:X,
                             arg2:Y)).

give ~~> synsem:(category:(head:vform:fin,
                           subcat:[(category:(head:case:dat),
                                    content:index: Z),
                                   (category:(head:case:acc),
                                    content:index: Y),
                                   (category:(head:case:nom),
                                    content:index: (X,
                                                    person:first,
                                                    number:sing)
                                                    )]),
                    content:(give_rel,
                             arg1:X,
                             arg2:Y,
                             arg3:Z)).

gives ~~>  synsem:(category:(head:vform:fin,
                           subcat:[(category:(head:case:dat),
                                    content:index: Z),
                                   (category:(head:case:acc),
                                    content:index: Y),
                                   (category:(head:case:nom),
                                    content:index: (X,
                                                    person:third,
                                                    number:sing)
                                                    )]),
                    content:(give_rel,
                             arg1:X,
                             arg2:Y,
                             arg3:Z)).


% phrase structure rules
subject_head_rule ##
(phrase, synsem:(category:(head:H,
                           subcat:[]),
                 content:Content),
```

```
 dtr1:Subj, dtr2:Head)
   ===>
cat> (Subj, synsem:Synsem),
cat> (Head, synsem:(category:(head:H,
subcat:[Synsem]),
content:Content)).


head_complement_rule ##
(phrase, synsem:(category:(head:H,
                           subcat:(ne_list, Rest)),
                 content:Content),
 dtr1:Comp, dtr2:Head)
   ===>
cat> (Head, synsem:(category:(head:H,
subcat:[First|Rest]),
content:Content)),
cat> (Comp, synsem:First).
```

Download[14]

6.6 Core Fragment

Abstract

The core fragment is a small grammar of English sentences. It covers the most basic facts of the syntax of clauses. This grammar can therefore serve as a basis for the development of grammars which analyze more intricate syntactic phenomena, and for testing and investigating them with computational tools.

6.6.1 Signature

Abstract

Note the link to a file with the source code at the bottom of the page.

```
type_hierarchy
bot
   list
     ne_list hd:bot tl:list
     e_list
   sign phon:list synsem:synsem
```

[14] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Spook/theory.pl

```
      word arg_st:list
      phrase daughters:const_struc dtrs:list
   synsem loc:loc nonloc:nonloc
   loc cat:cat cont:cont
   nonloc
   cat head:head val:val
   head pred:boolean mod:synsem_none
      func_or_verb vform:vform marking:marking
         verb aux:boolean inv:boolean marking:unmarked mod:none
         functional marking:marked
      noun case:case mod:none
      prep pform:pform
      adv mod:synsem
   val subj:list comps:list
   cont
      psoa
         move_rel
            walk_rel walker:index
         like_rel liker:ref liked:ref
         say_rel sayer:ref said:psoa
         give_rel giver:ref gift:ref given:ref
         rain_rel
         future_rel soa_arg:psoa
         direction_rel movement:move_rel goal:ref
         here_rel located:psoa
      nom_obj index:index
   index pers:pers num:num gen:gen
         ref
         nonref
            it
            there
   const_struc hdtr:sign ndtr:sign
      hs_struc
      hc_struc
      ha_struc
      sai_struc
   vform
      fin
      inf
      pas
      psp
      prp
      base
```

```
   case
      nom
      acc
   pform
      lexical
      non_lexical
         to
   marking
      unmarked
      marked
         that
         for
   boolean
      plus
      minus
   pers
      first
      second
      third
   num
      sg
      pl
   gen
      masc
      fem
      neut
   synsem_none
      none
      &synsem
.
```

Download[15]

6.6.2 Theory

Abstract

Note the link to a file with the source code at the bottom of the page.

```
% Multifile declarations.
:- multifile if/2.
```

[15] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Core-Fragment/signature

```
% load phonology and tree output
:- [trale_home(tree_extensions)].

% specify signature file
signature(signature).

% grisu output specifications

hidden_feat(dtrs).          % shown by the tree
hidden_feat(daughters).     % shown by the tree

synsem <<< arg_st.
vform <<< aux.
vform <<< inv.
subj <<< comps.
liker <<< liked.
sayer <<< said.
giver <<< gift.
gift <<< given.
>>> phon.


%=========================================================================

% Simulation of Parametric Sorts as Principles of Grammar

fun list_of_synsems(-).

list_of_synsems(X) if
   when( (X=(e_list;ne_list) ),
           und_list_of_synsems(X)).

und_list_of_synsems(X) if (X=e_list).
und_list_of_synsems(X) if (X=[(synsem)|Y]),list_of_synsems(Y).

% subj:list(synsem)
% comps:ist(synsem)

val *> (subj:list_of_synsems,
        comps:list_of_synsems).

% arg_st:list(synsem)
word *> arg_st:list_of_synsems.
```

```
%=====================================================================
% Phrase Structure Rules, encoding the ID-Principle and the
%                                     Constituent Order Principle and
%                                     Subcategorization Principle


% Head Subject Rule

head_subject_rule ##
(phrase,
 synsem:loc:cat:val:(subj:e_list,
                     comps:e_list),
 daughters:(hs_struc,
            hdtr:Hdtr,
            ndtr:Ndtr))
   ===>
cat> (Ndtr, synsem:Synsem),
cat> (Hdtr, synsem:loc:cat:val:(subj:[Synsem],
                                comps:e_list)).


% Head Complement Rule

head_complement_rule ##
(phrase,
 synsem:(loc:cat:val:(subj:Subj,
                     comps:List)),
 daughters:(hc_struc,
            hdtr:Hdtr,
            ndtr:Ndtr))
   ===>
cat> (Hdtr, synsem:loc:cat:val:(subj:Subj,
                                comps:[Synsem|List])),
cat> (Ndtr, synsem:Synsem).


% Head Adjunct Rule

head_adjunct_rule ##
(phrase,
```

```
 synsem:loc:cat:val:(subj:List,
                     comps:e_list),
 daughters:(ha_struc,
            hdtr:Hdtr,
            ndtr:Ndtr))
   ===>
cat> (Hdtr, synsem:(Synsem,
                  loc:cat:val:(subj:List,
                               comps:e_list))),
cat> (Ndtr, synsem:loc:cat:(head:mod:Synsem,
                            val:(subj:e_list,
                                 comps:e_list))).


% Subject Aux Inversion Rule

subject_aux_inversion_rule ##
(phrase,
 synsem:(loc:cat:val:(subj:e_list,
                      comps:List)),
 daughters:(sai_struc,
            hdtr:Hdtr,
            ndtr:Ndtr))
   ===>
cat> (Hdtr, word,
      synsem:loc:cat:(head:inv:plus,
                      val:(subj:[Synsem],
                           comps:List))),
cat> (Ndtr, synsem:Synsem).


%=====================================================================
% PRINCIPLES

% Head Feature Principle

phrase *> (synsem:loc:cat:head:Head,
           daughters:hdtr:synsem:loc:cat:head:Head).

% Semantics Principle

phrase *> ((synsem:loc:cont:Content,
          daughters:((hs_struc;hc_struc;sai_struc),
```

```
                    hdtr:synsem:loc:cont:Content);
          (synsem:loc:cont:Content,
          daughters:(ha_struc,
                    ndtr:synsem:loc:cont:Content))).

% INV Principle

(synsem:loc:cat:head:inv:plus) *> (synsem:loc:cat:head:(vform:fin,
                                                        aux:plus)).

% MOD Principle

(phrase,
 daughters:(hs_struc;hc_struc;sai_struc))
  *> daughters:ndtr:synsem:loc:cat:head:mod:none.


% Argument Realization Principle

(word,
 synsem:loc:cat:head:pred:plus) *> (synsem:loc:cat:val:(subj:[Synsem],
                                                        comps:List),
                                    arg_st:[Synsem|List]).

(word,
 synsem:loc:cat:head:pred:minus) *> (synsem:loc:cat:val:(subj:e_list,
                                                        comps:List),
                                    arg_st:List).


% Structural Case Principle

Synsem^(phrase,
 daughters:(hdtr:synsem:loc:cat:(head:vform:fin,
                                 val:subj:[Synsem]),
            ndtr:synsem:(Synsem,
                         loc:cat:head:noun)))
*> (daughters:ndtr:synsem:loc:cat:head:case:nom).

(phrase,
 daughters:(hc_struc,
            ndtr:synsem:loc:cat:head:noun))
*> (daughters:ndtr:synsem:loc:cat:head:case:acc).
```

```
% Functional Preposition Principle

(word,
phon:ne_list,
synsem:loc:cat:head:(prep,
                     pform:non_lexical))
*>
(synsem:loc:(cat:(head:(mod:none,pred:minus)),
             cont:Cont),
arg_st:([(loc:(cat:val:(subj:[],comps:[]),
               cont:Cont))])
).


% Subject Principles

(val,subj:ne_list) *> subj:[_].

(head:verb,val:subj:e_list) *> head:vform:fin.


%=============================================================================
% LEXICON

% NOUNS

peter ~~> (synsem:loc:(cat:head:(noun,
                                 pred:minus),
                       cont:index:(ref,
                                   num:sg,
                                   pers:third,
                                   gen:masc)),
           arg_st:e_list).

mary ~~> (synsem:loc:(cat:head:(noun,
                                 pred:minus),
                      cont:index:(ref,
                                  num:sg,
                                  pers:third,
                                  gen:fem)),
          arg_st:e_list).
```

```
he ~~> (synsem:loc:(cat:head:(noun,
                              case:nom,
                              pred:minus),
                    cont:index:(ref,
                                num:sg,
                                pers:third,
                                gen:masc)),
        arg_st:e_list).

she ~~> (synsem:loc:(cat:head:(noun,
                               case:nom,
                               pred:minus),
                     cont:index:(ref,
                                 num:sg,
                                 pers:third,
                                 gen:fem)),
         arg_st:e_list).

it ~~> (synsem:loc:(cat:head:(noun,
                                pred:minus),
                    cont:index:(ref,
                                num:sg,
                                pers:third,
                                gen:neut)),
        arg_st:e_list).

it ~~> (synsem:loc:(cat:head:(noun,
                              pred:minus),
                    cont:index:(it,
                                num:sg,
                                pers:third,
                                gen:neut)),
         arg_st:e_list).

him ~~> (synsem:loc:(cat:head:(noun,
                               case:acc,
                               pred:minus),
                     cont:index:(ref,
                                 num:sg,
                                 pers:third,
                                 gen:masc)),
          arg_st:e_list).
```

```
her ~~> (synsem:loc:(cat:head:(noun,
                               case:acc,
                               pred:minus),
                     cont:index:(ref,
                                 num:sg,
                                 pers:third,
                                 gen:fem)),
         arg_st:e_list).

you ~~> (synsem:loc:(cat:head:(noun,
                               pred:minus),
                     cont:index:(ref,
                                  pers:second)),
         arg_st:e_list).

they ~~> (synsem:loc:(cat:head:(noun,
                                case:nom,
                                pred:minus),
                       cont:index:(ref,
                                   pers:third,
                                   num:pl)),
          arg_st:e_list).

them ~~> (synsem:loc:(cat:head:(noun,
                                case:acc,
                                pred:minus),
                      cont:index:(ref,
                                   pers:third,
                                   num:pl)),
            arg_st:e_list).



% VERBS

walk ~~>  (synsem:loc:(cat:head:(verb,
                                 vform: base,
                                 pred: plus,
                                 aux: minus),
                        cont:(walk_rel,
                              walker:Index)),
            arg_st:[(loc:(cat:(head:noun,
```

```
                              val:(subj:e_list,
                                   comps:e_list)),
                            cont:index:Index))]).

walks ~~>  (synsem:loc:(cat:(head:(verb,
                                   vform:fin,
                                   pred:plus,
                                   aux:minus),
                                   val:subj:[(loc:cont:(psoa;index:(pers:third,
                                                                    num:sg)))]),
                           cont:(walk_rel,
                                 walker:Index)),
            arg_st:[(loc:(cat:(head:noun,
                               val:(subj:e_list,
                                    comps:e_list)),
                          cont:index:Index))]).


rain ~~>  (synsem:loc:(cat:head:(verb,
                                   vform: base,
                                   pred: plus,
                                   aux: minus),
                           cont:rain_rel),
            arg_st:[(loc:(cat:(head:noun,
                               val:(subj:e_list,
                                    comps:e_list)),
                             cont:index:it))]).

rains ~~>  (synsem:loc:(cat:(head:(verb,
                                   vform:fin,
                                   pred:plus,
                                   aux:minus),
                              val:subj:[(loc:cont:(psoa;index:(pers:third,
                                                               num:sg)))]),
                           cont:rain_rel),
            arg_st:[(loc:(cat:(head:noun,
                               val:(subj:e_list,
                                    comps:e_list)),
                             cont:index:it))]).

like ~~>  (synsem:loc:(cat:head:(verb,
                                   vform: base,
                                   pred: plus,
```

```
                                  aux: minus),
                          cont:(like_rel,
                                liker:Index1,
                                liked:Index2)),
              arg_st:[(loc:(cat:(head:noun,
                                 val:(subj:e_list,
                                      comps:e_list)),
                            cont:index:Index1)),
                       (loc:(cat:(head:noun,
                                  val:(subj:e_list,
                                       comps:e_list)),
                             cont:index:Index2))]).

likes ~~>  (synsem:loc:(cat:(head:(verb,
                                   vform: fin,
                                   pred: plus,
                                   aux: minus),
                              val:subj:[(loc:cont:(psoa;index:(pers:third,
                                                               num:sg)))]),
                          cont:(like_rel,
                                liker:Index1,
                                liked:Index2)),
              arg_st:[(loc:(cat:(head:noun,
                                 val:(subj:e_list,
                                      comps:e_list)),
                            cont:index:Index1)),
                       (loc:(cat:(head:noun,
                                  val:(subj:e_list,
                                       comps:e_list)),
                             cont:index:Index2))]).


say ~~>  (synsem:loc:(cat:head:(verb,
                                 vform: base,
                                 pred: plus,
                                 aux: minus),
                        cont:(say_rel,
                              sayer:Index,
                              said:Psoa)),
              arg_st:[(loc:(cat:(head:noun,
                                 val:(subj:e_list,
                                      comps:e_list)),
                            cont:index:Index)),
```

```
                    (loc:(cat:(head:(functional,
                                     vform:fin,
                                     marking:that),
                                val:(subj:e_list,
                                     comps:e_list)),
                           cont:Psoa))]).

says ~~>  (synsem:loc:(cat:(head:(verb,
                                  vform: fin,
                                  pred: plus,
                                  aux: minus),
                             val:subj:[(loc:cont:(psoa;index:(pers:third,
                                                              num:sg)))]),
                        cont:(say_rel,
                              sayer:Index,
                              said:Psoa)),
           arg_st:[(loc:(cat:(head:noun,
                              val:(subj:e_list,
                                   comps:e_list)),
                         cont:index:Index)),
                   (loc:(cat:(head:(functional,
                                    vform:fin,
                                    marking:that),
                              val:(subj:e_list,
                                   comps:e_list)),
                          cont:Psoa))]).

give ~~>  (synsem:loc:(cat:head:(verb,
                                 vform: base,
                                 pred: plus,
                                 aux: minus),
                        cont:(give_rel,
                              giver:Index1,
                              gift:Index2,
                              given:Index3)),
           arg_st:[(loc:(cat:(head:noun,
                              val:(subj:e_list,
                                   comps:e_list)),
                         cont:index:Index1)),
                   (loc:(cat:(head:noun,
                              val:(subj:e_list,
                                   comps:e_list)),
                         cont:index:Index2)),
```

```
                    (loc:(cat:(head:(prep,
                                     pform:to),
                                val:(subj:e_list,
                                     comps:e_list)),
                           cont:index:Index3))]).


gives ~~>  (synsem:loc:(cat:(head:(verb,
                                   vform:fin,
                                   pred:plus,
                                   aux:minus),
                                   val:subj:[(loc:cont:(psoa;index:(pers:third,
                                                                    num:sg)))]),
                       cont:(give_rel,
                             giver:Index1,
                             gift:Index2,
                             given:Index3)),
            arg_st:[(loc:(cat:(head:noun,
                               val:(subj:e_list,
                                    comps:e_list)),
                          cont:index:Index1)),
                    (loc:(cat:(head:noun,
                               val:(subj:e_list,
                                    comps:e_list)),
                          cont:index:Index2)),
                    (loc:(cat:(head:(prep,
                                     pform:to),
                               val:(subj:e_list,
                                    comps:e_list)),
                          cont:index:Index3))]).


%%% AUXILIARIES

% LE of will (future auxiliary)

will ~~> (synsem:loc:(cat:head:(verb,
                                vform:fin,
                                pred:plus,
                                aux:plus),
                      cont:(future_rel,
                            soa_arg:Cont)),
     arg_st:[(Synsem),(loc:(cat:(head:(verb,
```

```
                                        vform:base),
                                   val:(subj:[(Synsem)],
                                        comps:e_list)),
                                   cont:Cont))]).


% PREPOSITIONS

to ~~> (word, synsem:loc:cat:head:(prep,
                                   pform:to)).

to ~~> (word,
        synsem:loc:(cat:head:(prep,
                              mod:(loc:(cat:(head:verb,
                                             val:(subj:ne_list,
                                                  comps:e_list)),
                                        cont:(Cont1))),
                              pred:minus,
                              pform:lexical),
                         cont:(direction_rel,
                               movement:Cont1,
                              goal:Cont2)),
        arg_st:[(loc:(cat:(head:noun,
                           val:(subj:e_list,
                                comps:e_list)),
                      cont:index:Cont2))]).


% ADVERBS

here ~~> (synsem:loc:(cat:head:(adv,
                                pred:minus,
                                mod:loc:(cat:(head:verb,
                                              val:subj:ne_list),
                                         cont:Cont)),
                      cont:(here_rel,
                            located:Cont)),
        arg_st:[]).


% COMPLEMENTIZERS

that ~~> (synsem:loc:(cat:head:(functional,
```

```
                                pred:minus,
                                vform:Vform,
                                mod:none,
                                marking:that),
                        cont:Cont),
          arg_st:[(loc:(cat:(head:(verb,
                                   vform:(Vform,fin)),
                             val:(subj:e_list,
                                  comps:e_list)),
                        cont:Cont))]).

%===========================================================================
```

Download[16]

6.7 Fragment with Lexical Generalizations

Abstract

This grammar extends the core fragment with a theory of lexical generalizations. It focuses on the use of lexical rules in grammar implementations.

6.7.1 Signature

Abstract

Note the link to a file with the source code at the bottom of the page.

```
type_hierarchy
bot
   list
     ne_list hd:bot tl:list
     e_list
   sign phon:list synsem:synsem
      word arg_st:list
      phrase daughters:const_struc dtrs:list
   synsem loc:loc nonloc:nonloc
   loc cat:cat cont:cont
   nonloc
   cat head:head val:val
   head pred:boolean mod:synsem_none
```

[16] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Core-Fragment/theory.pl

```
      func_or_verb vform:vform marking:marking
         verb aux:boolean inv:boolean marking:unmarked mod:none
         functional marking:marked
      noun case:case mod:none
      prep pform:pform
      adv mod:synsem
   val subj:list comps:list
   cont
      psoa
         move_rel
            walk_rel walker:index
         like_rel liker:ref liked:ref
         say_rel sayer:ref said:psoa
         give_rel giver:ref gift:ref given:ref
         rain_rel
         tense_rel soa_arg:psoa
            future_rel
            perfect_rel
            present_rel
            past_rel
            cont_rel
         direction_rel movement:move_rel goal:ref
         here_rel located:psoa
      nom_obj index:index
   index pers:pers num:num gen:gen
         ref
         nonref
            it
            there
   const_struc hdtr:sign ndtr:sign
      hs_struc
      hc_struc
      ha_struc
      sai_struc
   vform
      fin
      inf
      pas
      psp
      prp
      base
   case
      nom
```

```
      acc
   pform
      lexical
      non_lexical
         by
         to
   marking
      unmarked
      marked
         that
         for
   boolean
      plus
      minus
   pers
      first
      second
      third
   num
      sg
      pl
   gen
      masc
      fem
      neut
   synsem_none
      none
      &synsem
.
```

Download[17]

6.7.2 Theory

Abstract

Note the link to a file with the source code at the bottom of the page.

```
% Multifile declarations.
%:- multifile '##'/2.
%:- multifile '*>'/2.
%:- multifile '~~>'/2.
```

[17] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Fragment-with-Lex-Gens/signature

```
:- multifile if/2.

% load phonology and tree output
:- [trale_home(tree_extensions)].

% specify signature file
signature(signature).

% grisu output specifications

hidden_feat(dtrs).          % shown by the tree
hidden_feat(daughters).     % shown by the tree

synsem <<< arg_st.
vform <<< aux.
vform <<< inv.
subj <<< comps.
liker <<< liked.
sayer <<< said.
giver <<< gift.
gift <<< given.
>>> phon.


%=========================================================================

% Simulation of Parametric Sorts as Principles of Grammar

fun list_of_synsems(-).

list_of_synsems(X) if
   when( (X=(e_list;ne_list) ),
           und_list_of_synsems(X)).

und_list_of_synsems(X) if (X=e_list).
und_list_of_synsems(X) if (X=[(synsem)|Y]),list_of_synsems(Y).

% subj:list(synsem)
% comps:ist(synsem)

val *> (subj:list_of_synsems,
        comps:list_of_synsems).
```

```
% arg_st:list(synsem)
word *> arg_st:list_of_synsems.



%=====================================================================
% Phrase Structure Rules, encoding the ID-Principle and the
%                                     Constituent Order Principle and
%                                     Subcategorization Principle


% Head Subject Rule

head_subject_rule ##
(phrase,
 synsem:loc:cat:val:(subj:e_list,
                     comps:e_list),
 daughters:(hs_struc,
            hdtr:Hdtr,
            ndtr:Ndtr))
   ===>
cat> (Ndtr, synsem:Synsem),
cat> (Hdtr, synsem:loc:cat:val:(subj:[Synsem],
                                comps:e_list)).


% Head Complement Rule

head_complement_rule ##
(phrase,
 synsem:(loc:cat:val:(subj:Subj,
                     comps:List)),
 daughters:(hc_struc,
            hdtr:Hdtr,
            ndtr:Ndtr))
   ===>
cat> (Hdtr, synsem:loc:cat:val:(subj:Subj,
                                comps:[Synsem|List])),
cat> (Ndtr, synsem:Synsem).


% Head Adjunct Rule
```

```
head_adjunct_rule ##
(phrase,
 synsem:loc:cat:val:(subj:List,
                     comps:e_list),
 daughters:(ha_struc,
            hdtr:Hdtr,
            ndtr:Ndtr))
   ===>
cat> (Hdtr, synsem:(Synsem,
                  loc:cat:val:(subj:List,
                               comps:e_list))),
cat> (Ndtr, synsem:loc:cat:(head:mod:Synsem,
                            val:(subj:e_list,
                                 comps:e_list))).  % added


% Subject Aux Inversion Rule

subject_aux_inversion_rule ##
(phrase,
 synsem:(loc:cat:val:(subj:e_list,
                  comps:List)),
 daughters:(sai_struc,
            hdtr:Hdtr,
            ndtr:Ndtr))
   ===>
cat> (Hdtr, word,
      synsem:loc:cat:(head:inv:plus,
                      val:(subj:[Synsem],
                           comps:List))),
cat> (Ndtr, synsem:Synsem).


%=============================================================================
% PRINCIPLES

% Head Feature Principle

phrase *> (synsem:loc:cat:head:Head,
           daughters:hdtr:synsem:loc:cat:head:Head).

% Semantics Principle
```

```
phrase *> ((synsem:loc:cont:Content,
            daughters:((hs_struc;hc_struc;sai_struc),
                       hdtr:synsem:loc:cont:Content);
           (synsem:loc:cont:Content,
            daughters:(ha_struc,
                       ndtr:synsem:loc:cont:Content)))).

% INV Principle

(synsem:loc:cat:head:inv:plus) *> (synsem:loc:cat:head:(vform:fin,
                                                        aux:plus)).

% MOD Principle

(phrase,
 daughters:(hs_struc;hc_struc;sai_struc))
  *> daughters:ndtr:synsem:loc:cat:head:mod:none.


% Argument Realization Principle

(word,
 synsem:loc:cat:head:pred:plus) *> (synsem:loc:cat:val:(subj:[Synsem],
                                                        comps:List),
                                    arg_st:[Synsem|List]).

(word,
 synsem:loc:cat:head:pred:minus) *> (synsem:loc:cat:val:(subj:e_list,
                                                        comps:List),
                                    arg_st:List).


% Structural Case Principle

Synsem^(phrase,
 daughters:(hdtr:synsem:loc:cat:(head:vform:fin,
                                 val:subj:[Synsem]),
            ndtr:synsem:(Synsem,
                         loc:cat:head:noun)))
*> (daughters:ndtr:synsem:loc:cat:head:case:nom).

(phrase,
 daughters:(hc_struc,
```

```
            ndtr:synsem:loc:cat:head:noun))
*> (daughters:ndtr:synsem:loc:cat:head:case:acc).


% Functional Preposition Principle

(word,
phon:ne_list,
synsem:loc:cat:head:(prep,
                     pform:non_lexical))
*>
(synsem:loc:(cat:(head:(mod:none,pred:minus)),
               cont:Cont),
arg_st:([(loc:(cat:val:(subj:[],comps:[]),
                cont:Cont))])
).


% Subject Principles

(val,subj:ne_list) *> subj:[_].

(head:verb,val:subj:e_list) *> head:vform:fin.


%=============================================================================
% LEXICON

% NOUNS

peter ~~> (synsem:loc:(cat:head:(noun,
                              pred:minus),
                        cont:index:(ref,
                                    num:sg,
                                    pers:third,
                                    gen:masc)),
           arg_st:e_list).

mary ~~> (synsem:loc:(cat:head:(noun,
                              pred:minus),
                       cont:index:(ref,
                                   num:sg,
                                   pers:third,
```

```
                                    gen:fem)),
          arg_st:e_list).

he ~~> (synsem:loc:(cat:head:(noun,
                              case:nom,
                              pred:minus),
                    cont:index:(ref,
                                num:sg,
                                pers:third,
                                gen:masc)),
        arg_st:e_list).

she ~~> (synsem:loc:(cat:head:(noun,
                               case:nom,
                               pred:minus),
                     cont:index:(ref,
                                 num:sg,
                                 pers:third,
                                 gen:fem)),
         arg_st:e_list).

it ~~> (synsem:loc:(cat:head:(noun,
                              pred:minus),
                    cont:index:(ref,
                                num:sg,
                                pers:third,
                                gen:neut)),
         arg_st:e_list).

it ~~> (synsem:loc:(cat:head:(noun,
                              pred:minus),
                    cont:index:(it,
                                num:sg,
                                pers:third,
                                gen:neut)),
         arg_st:e_list).

him ~~> (synsem:loc:(cat:head:(noun,
                               case:acc,
                               pred:minus),
                     cont:index:(ref,
                                 num:sg,
                                 pers:third,
```

```
                                  gen:masc)),
          arg_st:e_list).

her ~~> (synsem:loc:(cat:head:(noun,
                               case:acc,
                               pred:minus),
                     cont:index:(ref,
                                 num:sg,
                                 pers:third,
                                 gen:fem)),
          arg_st:e_list).

you ~~> (synsem:loc:(cat:head:(noun,
                               pred:minus),
                     cont:index:(ref,
                                 pers:second)),
          arg_st:e_list).

they ~~> (synsem:loc:(cat:head:(noun,
                                case:nom,
                                pred:minus),
                      cont:index:(ref,
                                  pers:third,
                                  num:pl)),
           arg_st:e_list).

them ~~> (synsem:loc:(cat:head:(noun,
                                case:acc,
                                pred:minus),
                      cont:index:(ref,
                                  pers:third,
                                  num:pl)),
           arg_st:e_list).



% VERBS

walk ~~>  (synsem:loc:(cat:head:(verb,
                                 vform: base,
                                 pred: plus,
                                 aux: minus),
                       cont:(walk_rel,
```

```
                            walker:Index)),
            arg_st:[(loc:(cat:(head:noun,
                               val:(subj:e_list,
                                    comps:e_list)),
                          cont:index:Index))]).


rain ~~>  (synsem:loc:(cat:head:(verb,
                                 vform: base,
                                 pred: plus,
                                 aux: minus),
                        cont:rain_rel),
            arg_st:[(loc:(cat:(head:noun,
                               val:(subj:e_list,
                                    comps:e_list)),
                          cont:index:it))]).

like ~~>  (synsem:loc:(cat:head:(verb,
                                 vform: base,
                                 pred: plus,
                                 aux: minus),
                       cont:(like_rel,
                             liker:Index1,
                             liked:Index2)),
            arg_st:[(loc:(cat:(head:noun,
                               val:(subj:e_list,
                                    comps:e_list)),
                          cont:index:Index1)),
                    (loc:(cat:(head:noun,
                               val:(subj:e_list,
                                    comps:e_list)),
                          cont:index:Index2))]).


say ~~>  (synsem:loc:(cat:head:(verb,
                                 vform: base,
                                 pred: plus,
                                 aux: minus),
                      cont:(say_rel,
                            sayer:Index,
                            said:Psoa)),
            arg_st:[(loc:(cat:(head:noun,
                               val:(subj:e_list,
```

```
                                        comps:e_list)),
                              cont:index:Index)),
                        (loc:(cat:(head:(functional,
                                         vform:fin,
                                         marking:that),
                                   val:(subj:e_list,
                                        comps:e_list)),
                              cont:Psoa))]).


give ~~>  (synsem:loc:(cat:head:(verb,
                                 vform: base,
                                 pred: plus,
                                 aux: minus),
                          cont:(give_rel,
                                giver:Index1,
                                gift:Index2,
                                given:Index3)),
             arg_st:[(loc:(cat:(head:noun,
                                val:(subj:e_list,
                                     comps:e_list)),
                           cont:index:Index1)),
                     (loc:(cat:(head:noun,
                                val:(subj:e_list,
                                comps:e_list)),
                           cont:index:Index2)),
                     (loc:(cat:(head:(prep,
                                      pform:to),
                                val:(subj:e_list,
                                     comps:e_list)),
                           cont:index:Index3))]).


%%% AUXILIARIES

% LE of is (passive auxiliary)

be ~~> (synsem:loc:(cat:head:(verb,
                              vform:base,
                              pred:plus,
                              aux:plus),
                    cont:Cont),
        arg_st:[(Synsem),(loc:(cat:(head:(verb,
```

```
                                          vform:pas),
                                     val:(subj:[(Synsem)],
                                          comps:e_list)),
                                cont:Cont))]).

% LE of will (future auxiliary)

will ~~> (synsem:loc:(cat:head:(verb,
                                vform:fin,
                                pred:plus,
                                aux:plus),
                          cont:(future_rel,
                                soa_arg:Cont)),
      arg_st:[(Synsem),(loc:(cat:(head:(verb,
                                        vform:base),
                                   val:(subj:[(Synsem)],
                                        comps:e_list)),
                              cont:Cont))]).


have ~~> (synsem:loc:(cat:head:(verb,
                                vform:base,
                                pred:plus,
                                aux:plus),
                          cont:Cont),
      arg_st:[(Synsem),(loc:(cat:(head:(verb,
                                        vform:psp),
                                   val:(subj:[(Synsem)],
                                        comps:e_list)),
                              cont:Cont))]).

be ~~> (synsem:loc:(cat:head:(verb,
                               vform:base,
                               pred:plus,
                               aux:plus),
                         cont:Cont),
      arg_st:[(Synsem),(loc:(cat:(head:(verb,
                                        vform:prp),
                                   val:(subj:[(Synsem)],
                                        comps:e_list)),
                              cont:Cont))]).
```

```
% PREPOSITIONS

by ~~> (word, synsem:loc:cat:head:(prep,
                                   pform:by)).

to ~~> (word, synsem:loc:cat:head:(prep,
                                   pform:to)).

to ~~> (word,
        synsem:loc:(cat:head:(prep,
                              mod:(loc:(cat:(head:verb,
                                             val:(subj:ne_list,
                                                  comps:e_list)),
                                        cont:(Cont1))),
                              pred:minus,
                              pform:lexical),
                    cont:search_rel(Cont2,Cont1)),
        arg_st:[(loc:(cat:(head:noun,
                           val:(subj:e_list,
                                comps:e_list)),
                      cont:index:Cont2))]).

fun search_rel(+,+,-).

search_rel(C,X,Y) if
  when( (X=(move_rel;tense_rel;like_rel;say_rel;give_rel;rain_rel;
            direction_rel;here_rel) ),
          (und_search_rel(C,(X,(move_rel;tense_rel)),Y);fail)).

und_search_rel(C,X,Y) if (X=move_rel),(Y=(direction_rel,
                                          movement:X,
                                          goal:C)).
und_search_rel(C,X,Y) if (X=(future_rel,
                             soa_arg:Cont)),(Y=(future_rel,
                                                soa_arg:search_rel(C,Cont))).
und_search_rel(C,X,Y) if (X=(perfect_rel,
                             soa_arg:Cont)),(Y=(perfect_rel,
                                                soa_arg:search_rel(C,Cont))).
und_search_rel(C,X,Y) if (X=(present_rel,
                             soa_arg:Cont)),(Y=(present_rel,
                                                soa_arg:search_rel(C,Cont))).
und_search_rel(C,X,Y) if (X=(past_rel,
```

```
                              soa_arg:Cont)),(Y=(past_rel,
                                                 soa_arg:search_rel(C,Cont))).
und_search_rel(C,X,Y) if (X=(cont_rel,
                              soa_arg:Cont)),(Y=(cont_rel,
                                                 soa_arg:search_rel(C,Cont))).


% ADVERBS

here ~~> (synsem:loc:(cat:head:(adv,
                                pred:minus,
                                mod:loc:(cat:(head:verb,
                                              val:subj:ne_list),
                                         cont:Cont)),
                        cont:(here_rel,
                              located:Cont)),
          arg_st:[]).


% COMPLEMENTIZERS

that ~~> (synsem:loc:(cat:head:(functional,
                                pred:minus,
                                vform:Vform,
                                mod:none,
                                marking:that),
                        cont:Cont),
          arg_st:[(loc:(cat:(head:(verb,
                                   vform:(Vform,fin)),
                              val:(subj:e_list,
                                   comps:e_list)),
                        cont:Cont))]).



%=============================================================================
% Lexical Rules

% PSP Lexical Rule

psp_lex_rule ##
(word,
 synsem:loc:(cat:head:(vform:base,
```

```
                        aux:Aux,
                        pred:Pred),
              cont:Cont),
 arg_st:Arg)
 **>
(synsem:loc:(cat:head:(vform:psp,
                        aux:Aux,
                        pred:Pred),
              cont:(perfect_rel,
                    soa_arg:Cont)),
 arg_st:Arg)
morphs
be becomes been,
give becomes given,
have becomes had,
say becomes said,
(X,[e]) becomes (X,ed),
X becomes (X,ed).

% Present Participle Lexical Rule

prp_lex_rule ##
(word,
 synsem:loc:(cat:head:(vform:base,
                        aux:Aux,
                        pred:Pred),
              cont:Cont),
 arg_st:Arg)
 **>
(synsem:loc:(cat:head:(vform:prp,
                        aux:Aux,
                        pred:Pred),
              cont:(cont_rel,
                    soa_arg:Cont)),
 arg_st:Arg)
morphs
be becomes being,
(X,[e]) becomes (X,ing),
X becomes (X,ing).


% Passive Lexical Rule
```

```
passive_lex_rule ##
(word,
 synsem:(loc:(cat:head:(verb,
                        vform:psp,
                        aux:(Aux,minus),      % no passive of auxiliaries
                        inv:Inv,
                        pred:Pred),
              cont:soa_arg:Cont),
         nonloc:Nonloc),
 arg_st:[(loc:(cat:(head:noun,
                    val:(subj:e_list,
                         comps:e_list)),
              cont:Cont2)),
         Synsem1|
         List])
 **>
(word,
 synsem:(loc:(cat:head:(verb,
                        vform:pas,
                        aux:Aux,
                        inv:Inv,
                        pred:Pred),
              cont:Cont),
         nonloc:Nonloc),
 arg_st:([Synsem1|List];Result))
if append(([Synsem1|List]),[(loc:(cat:(head:(prep,
                                        pform:by),
                                    val:(subj:e_list,
                                         comps:e_list)),
                                cont:Cont2))],Result)
morphs
X becomes X.


append(X,Y,Z) if
   when(  ( X=(e_list;ne_list)
          ; Y=e_list
          ; Z=(e_list;ne_list)
               ),
            undelayed_append(X,Y,Z)).

undelayed_append(L,[],L) if true.
undelayed_append([],(L,ne_list),L) if true.
```

```
undelayed_append([H|T1],(L,ne_list),[H|T2]) if
  append(T1,L,T2).


% 3rd_sing_fin_lex_rule

third_sing_fin_lex_rule ##
(word,
 synsem:loc:(cat:head:(vform:base,
                       aux:Aux,
                       pred:Pred),
             cont:Cont),
 arg_st:Arg)
 **>
(synsem:loc:(cat:(head:(vform:fin,
                       aux:Aux,
                       pred:Pred),
                  val:subj:[(loc:cont:(psoa;index:(pers:third,
                                                   num:sg)))]),
             cont:(present_rel,
                  soa_arg:Cont)),
 arg_st:Arg)
morphs
be becomes is,
have becomes has,
X becomes (X,s).

% non_3rd_sing_fin_lex_rule

non_third_sing_fin_lex_rule ##
(word,
 synsem:loc:(cat:head:(vform:base,
                       aux:Aux,
                       pred:Pred),
             cont:Cont),
 arg_st:Arg)
 **>
(synsem:loc:(cat:(head:(vform:fin,
                       aux:Aux,
                       pred:Pred),
                  val:subj:[(loc:cont:index:(num:pl;
                                             (pers:(first;second),
                                             num:sg)))]),
```

```
             cont:(present_rel,
                   soa_arg:Cont)),
 arg_st:Arg)
morphs
be becomes are,
X becomes X.


% past_lex_rule no1

non_third__first_sing_past_lex_rule ##
(word,
 synsem:loc:(cat:head:(vform:base,
                       aux:Aux,
                       pred:Pred),
             cont:Cont),
 arg_st:Arg)
 **>
(synsem:loc:(cat:(head:(vform:fin,
                       aux:Aux,
                       pred:Pred),
                  val:subj:[(loc:cont:index:(num:pl;
                                             (pers:second,
                                              num:sg)))]),
             cont:(past_rel,
                   soa_arg:Cont)),
 arg_st:Arg)
morphs
be becomes were,
give becomes gave,
have becomes had,
say becomes said,
(X,[e]) becomes (X,ed),
X becomes (X,ed).


% past_lex_rule no2

third__first_sing_past_lex_rule ##
(word,
 synsem:loc:(cat:head:(vform:base,
                       aux:Aux,
                       pred:Pred),
```

```
             cont:Cont),
 arg_st:Arg)
 **>
(synsem:loc:(cat:(head:(vform:fin,
                        aux:Aux,
                        pred:Pred),
                  val:subj:[(loc:cont:(psoa;index:(pers:(first;third),
                                                   num:sg)))]),
             cont:(past_rel,
                   soa_arg:Cont)),
 arg_st:Arg)
morphs
be becomes was,
give becomes gave,
have becomes had,
say becomes said,
(X,[e]) becomes (X,ed),
X becomes (X,ed).


%===========================================================================
```

Download[18]

6.8 UDC Grammar

Abstract

With unbounded dependency constructions, a theory of a well-researched but very difficult empirical phenomenon is added to our small grammar of English. The present grammar is particularly good for investigating the tensions between the logical specification of an HPSG grammar and computational needs of grammar implementation. The grammar serves for investigating the meaning-preserving reformulations which are necessary to achieve an implementation true to the original specification of a complex theory of grammar.

6.8.1 Signature

Abstract

Note the link to a file with the source code at the bottom of the page.

[18] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/Fragment-with-Lex-Gens/theory.pl

```
type_hierarchy
bot
   list
     ne_list hd:bot tl:list
     e_list
   sign phon:list synsem:synsem
      word arg_st:list
      phrase daughters:const_struc dtrs:list
   synsem loc:loc nonloc:nonloc
   loc cat:cat cont:cont
   nonloc inherited:list to_bind:list
   cat head:head val:val
   head pred:boolean mod:synsem_none
      func_or_verb vform:vform marking:marking
         verb aux:boolean inv:boolean marking:unmarked mod:none
         functional marking:marked
      noun case:case mod:none
      prep pform:pform
      adv mod:synsem
   val subj:list comps:list
   cont
      psoa
         move_rel
            walk_rel walker:index
         like_rel liker:ref liked:ref
         say_rel sayer:ref said:psoa
         give_rel giver:ref gift:ref given:ref
         rain_rel
         tense_rel soa_arg:psoa
            future_rel
            perfect_rel
            present_rel
            past_rel
            cont_rel
         direction_rel movement:move_rel goal:ref
         here_rel located:psoa
      nom_obj index:index
   index pers:pers num:num gen:gen
         ref
         nonref
            it
            there
   const_struc hdtr:sign ndtr:sign
```

```
      hs_struc
      hc_struc
      ha_struc
      hf_struc
      sai_struc
   vform
      fin
      inf
      pas
      psp
      prp
      base
   case
      nom
      acc
   pform
      lexical
      non_lexical
         by
         to
   marking
      unmarked
      marked
         that
         for
   boolean
      plus
      minus
   pers
      first
      second
      third
   num
      sg
      pl
   gen
      masc
      fem
      neut
   synsem_none
      none
      &synsem
.
```

Download[19]

6.8.2 Theory

Abstract

Note the link to a file with the source code at the bottom of the page.

```
% Multifile declarations.
:- multifile if/2.

% load phonology and tree output
:- [trale_home(tree_extensions)].

% specify signature file
signature(signature).

% grisu output specifications

hidden_feat(dtrs).          % shown by the tree
hidden_feat(daughters).     % shown by the tree

synsem <<< arg_st.
vform <<< aux.
vform <<< inv.
subj <<< comps.
liker <<< liked.
sayer <<< said.
giver <<< gift.
gift <<< given.
>>> phon.


%=========================================================================

% Simulation of Parametric Sorts as Principles of Grammar

fun list_of_synsems(-).

list_of_synsems(X) if
   when( (X=(e_list;ne_list) ),
           und_list_of_synsems(X)).
```

[19] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/UDC-Grammatik/signature

```
und_list_of_synsems(X) if (X=e_list).
und_list_of_synsems(X) if (X=[(synsem)|Y]),list_of_synsems(Y).

% subj:list(synsem)
% comps:ist(synsem)

val *> (subj:list_of_synsems,
        comps:list_of_synsems).

% arg_st:list(synsem)
word *> arg_st:list_of_synsems.


fun list_of_locs(-).

list_of_locs(X) if
   when( (X=(e_list;ne_list) ),
           und_list_of_locs(X)).

und_list_of_locs(X) if (X=e_list).
und_list_of_locs(X) if (X=[(loc)|Y]),list_of_locs(Y).


% inherited:list(loc)
% to_bind:list(loc)

nonloc *> (inherited:list_of_locs,
           to_bind:list_of_locs).


%=====================================================================
% Phrase Structure Rules, encoding the ID-Principle and the
%                                      Constituent Order Principle and
%                                      Subcategorization Principle


% Head Subject Rule

head_subject_rule ##
(phrase,
 synsem:loc:cat:val:(subj:e_list,
                     comps:e_list),
```

```
 daughters:(hs_struc,
            hdtr:Hdtr,
            ndtr:Ndtr))
   ===>
cat> (Ndtr, synsem:Synsem),
cat> (Hdtr, synsem:loc:cat:val:(subj:[Synsem],
                                comps:e_list)).


% Head Complement Rule

head_complement_rule ##
(phrase,
 synsem:(loc:cat:val:(subj:Subj,
                      comps:List)),
 daughters:(hc_struc,
            hdtr:Hdtr,
            ndtr:Ndtr))
   ===>
cat> (Hdtr, synsem:loc:cat:val:(subj:Subj,
                                comps:[Synsem|List])),
cat> (Ndtr, synsem:Synsem).


% Head Adjunct Rule

head_adjunct_rule ##
(phrase,
 synsem:loc:cat:val:(subj:List,
                     comps:e_list),
 daughters:(ha_struc,
            hdtr:Hdtr,
            ndtr:Ndtr))
   ===>
cat> (Hdtr, synsem:(Synsem,
                  loc:cat:val:(subj:List,
                               comps:e_list))),
cat> (Ndtr, synsem:loc:cat:(head:mod:Synsem,
                            val:(subj:e_list,
                                 comps:e_list))).  % added


% Subject Aux Inversion Rule
```

```
subject_aux_inversion_rule ##
(phrase,
 synsem:(loc:cat:val:(subj:e_list,
  comps:List)),
 daughters:(sai_struc,
            hdtr:Hdtr,
            ndtr:Ndtr))
   ===>
cat> (Hdtr, word,
      synsem:loc:cat:(head:inv:plus,
                      val:(subj:[Synsem],
                           comps:List))),
cat> (Ndtr, synsem:Synsem).


% Head Filler Rule

head_filler_rule ##
(phrase,
 synsem:loc:cat:val:Val,
 daughters:(hf_struc,
            hdtr:Hdtr,
            ndtr:Ndtr))
   ===>
cat> (Ndtr, phon:ne_list,
            synsem:(loc:Local,
                    nonloc:inherited:e_list)),
cat> (Hdtr, synsem:(loc:cat:(head:(verb,
                                   vform:fin),
                             val:(Val,(subj:e_list,
                                       comps:e_list))),
                    nonloc:(to_bind:[Local],
                            inherited:List))),
goal> our_member(Local,List).


% functional definition of our-member for the Subject Condition

fun our_member(+,-).

our_member(Hd,[Hd|_]) if true.
our_member(Hd,[_|Tl]) if our_member(Hd,Tl).
```

```
%=====================================================================
% PRINCIPLES

% Head Feature Principle

phrase *> (synsem:loc:cat:head:Head,
           daughters:hdtr:synsem:loc:cat:head:Head).

% Semantics Principle

phrase *> ((synsem:loc:cont:Content,
daughters:((hs_struc;hc_struc;hf_struc;sai_struc),
         hdtr:synsem:loc:cont:Content);
            (synsem:loc:cont:Content,
 daughters:(ha_struc,
                      ndtr:synsem:loc:cont:Content)))).

% INV Principle

(synsem:loc:cat:head:inv:plus) *> (synsem:loc:cat:head:(vform:fin,
                                                        aux:plus)).

% MOD Principle

(phrase,
 daughters:(hs_struc;hc_struc;sai_struc;hf_struc))
  *> daughters:ndtr:synsem:loc:cat:head:mod:none.


% Argument Realization Principle

(word,
 synsem:loc:cat:head:pred:plus) *> (synsem:loc:cat:val:(subj:[Synsem],
                                                        comps:List),
                                    arg_st:[Synsem|List]).

(word,
 synsem:loc:cat:head:pred:minus) *> (synsem:loc:cat:val:(subj:e_list,
                                                        comps:List),
                                    arg_st:List).
```

```
% Structural Case Principle

Synsem^(phrase,
 daughters:(hdtr:synsem:loc:cat:(head:vform:fin,
                                 val:subj:[Synsem]),
            ndtr:synsem:(Synsem,
                         loc:cat:head:noun)))
*> (daughters:ndtr:synsem:loc:cat:head:case:nom).

(phrase,
 daughters:(hc_struc,
            ndtr:synsem:loc:cat:head:noun))
*> (daughters:ndtr:synsem:loc:cat:head:case:acc).


% Functional Preposition Principle

(word,
phon:ne_list,
synsem:loc:cat:head:(prep,
              pform:non_lexical))
*>
(synsem:loc:(cat:(head:(mod:none,pred:minus)),
              cont:Cont),
arg_st:([(loc:(cat:val:(subj:[],comps:[]),
               cont:Cont))])
).


% Subject Principles

(val,subj:ne_list) *> subj:[_].

(head:verb,val:subj:e_list) *> head:vform:fin.


% The Nonlocal Feature Principle

phrase *>
   (synsem:nonloc:inherited:Result,
 daughters:(hdtr:synsem:nonloc:(inherited:List1,
                                to_bind:Subtrac),
```

```
  ndtr:synsem:nonloc:inherited:List2))
goal
   nfp(List1,List2,Subtrac,Result).

nfp(List1,List2,[],Result) if append(List1,List2,Result).
nfp(L1,L2,([Ele]),Result) if (append(L1,L2,L3),select(Ele,L3,Result)).

append(X,Y,Z) if
   when(  ( X=(e_list;ne_list)
          ; Y=e_list
          ; Z=(e_list;ne_list)
               ),
            undelayed_append(X,Y,Z)).

undelayed_append(L,[],L) if true.
undelayed_append([],(L,ne_list),L) if true.
undelayed_append([H|T1],(L,ne_list),[H|T2]) if
  append(T1,L,T2).

select(Ele,L1,L2) if
   when(  ( L1=(e_list;ne_list)
               ),
            und_select(Ele,L1,L2)).

und_select(Ele,[Ele|Tl],Tl) if true.
und_select(Ele,[Hd|Tl1],[Hd|Tl2]) if select(Ele,Tl1,Tl2).


% Phrasal to-bind Principle

(phrase,
 daughters:hdtr:(phrase,
                 synsem:nonloc:to_bind:ne_list))
*> (daughters:hf_struc).

(phrase,daughters:(hs_struc;hc_struc;sai_struc;ha_struc))
 *> daughters:hdtr:synsem:nonloc:to_bind:e_list.


% Structural Trace Principle

% Clause (1)
```

```
(daughters:ndtr:phon:e_list) *>
 (synsem:(nonloc:to_bind:e_list,                % no vacuous movement
         loc:cat:head:(verb;noun;prep;adv)),    % not functional
  daughters: (hs_struc;hc_struc;sai_struc)). % follows already independently
                                             % from the implementation

% Clause (2) and (3)
% The effect clauses (2) and (3) logically follows from the more specific
% lexical entry of traces in the implementation (see below): This grammar
% only contains nominal and prepositional traces. Moreover, they are
% syntactically saturated. This is equivalent to restricting a more
% general entry of a trace by a principle and to forbidding the extraction
% of head daughters.


% Subject Condition

(word,
 synsem:loc:cat:head:pred:plus,
 arg_st:hd:nonloc:inherited:ne_list) *>
( (arg_st:[(loc:Loc,
    nonloc:inherited:[Loc])|_]);
  (arg_st:[_|our_member((nonloc:inherited:ne_list))])).



%=============================================================================
% LEXICON

% NOUNS

peter ~~> (synsem:(loc:(cat:head:(noun,
    pred:minus),
    cont:index:(ref,
                                num:sg,
               pers:third,
                                gen:masc)),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:e_list).

mary ~~> (synsem:(loc:(cat:head:(noun,
    pred:minus),
```

```
    cont:index:(ref,
                                num:sg,
                pers:third,
                                gen:fem)),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:e_list).

he ~~> (synsem:(loc:(cat:head:(noun,
  case:nom,
  pred:minus),
    cont:index:(ref,
                                num:sg,
                pers:third,
                                gen:masc)),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:e_list).

she ~~> (synsem:(loc:(cat:head:(noun,
  case:nom,
  pred:minus),
    cont:index:(ref,
                                num:sg,
                pers:third,
                                    gen:fem)),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:e_list).

it ~~> (synsem:(loc:(cat:head:(noun,
    pred:minus),
    cont:index:(ref,
                                num:sg,
                pers:third,
                                gen:neut)),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:e_list).

it ~~> (synsem:(loc:(cat:head:(noun,
    pred:minus),
    cont:index:(it,
```

```
                                num:sg,
                pers:third,
                                gen:neut)),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:e_list).

him ~~> (synsem:(loc:(cat:head:(noun,
  case:acc,
  pred:minus),
    cont:index:(ref,
                                num:sg,
                pers:third,
                                gen:masc)),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:e_list).

her ~~> (synsem:(loc:(cat:head:(noun,
  case:acc,
  pred:minus),
    cont:index:(ref,
                                num:sg,
                pers:third,
                                gen:fem)),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:e_list).

you ~~> (synsem:(loc:(cat:head:(noun,
  pred:minus),
    cont:index:(ref,
                               pers:second)),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:e_list).

they ~~> (synsem:(loc:(cat:head:(noun,
  case:nom,
  pred:minus),
    cont:index:(ref,
                               pers:third,
                               num:pl)),
```

```
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:e_list).

them ~~> (synsem:(loc:(cat:head:(noun,
        case:acc,
        pred:minus),
    cont:index:(ref,
                              pers:third,
                              num:pl)),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:e_list).

empty  word,
       phon:e_list,
       synsem:(loc:(Local,
                  cat:head:(noun,
            pred:minus)),
              nonloc:(inherited:[Local],
                      to_bind:e_list)),
       arg_st:e_list.           % the specification arg_st:e_list excludes
                                % heads in nominal phrases from being
                                % extracted because it entails that the
                                % extracted element is fully saturated


% VERBS

walk ~~>  (synsem:(loc:(cat:head:(verb,
         vform: base,
                                 pred: plus,
                                 aux: minus),
    cont:(walk_rel,
        walker:Index)),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:[(loc:(cat:(head:noun,
         val:(subj:e_list,
                                  comps:e_list)),
                         cont:index:Index))]).
```

```
rain ~~>  (synsem:(loc:(cat:head:(verb,
          vform: base,
                                    pred: plus,
                                   aux: minus),
    cont:rain_rel),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:[(loc:(cat:(head:noun,
                  val:(subj:e_list,
                                    comps:e_list)),
                           cont:index:it))]).

like ~~>  (synsem:(loc:(cat:head:(verb,
          vform: base,
                                  pred: plus,
                                  aux: minus),
    cont:(like_rel,
          liker:Index1,
                               liked:Index2)),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:[(loc:(cat:(head:noun,
         val:(subj:e_list,
                                    comps:e_list)),
                           cont:index:Index1)),
                     (loc:(cat:(head:noun,
         val:(subj:e_list,
                                    comps:e_list)),
                           cont:index:Index2))]).


say ~~>  (synsem:(loc:(cat:head:(verb,
          vform: base,
                                     pred: plus,
                                    aux: minus),
    cont:(say_rel,
          sayer:Index,
                           said:Psoa)),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:[(loc:(cat:(head:noun,
         val:(subj:e_list,
                                    comps:e_list)),
```

```
                         cont:index:Index)),
                    (loc:(cat:(head:(functional,
                                     vform:fin,
    marking:that),
         val:(subj:e_list,
                                  comps:e_list)),
                         cont:Psoa))]).


give ~~>  (synsem:(loc:(cat:head:(verb,
         vform: base,
                                 pred: plus,
                                 aux: minus),
    cont:(give_rel,
          giver:Index1,
                              gift:Index2,
                              given:Index3)),
         nonloc:(inherited:e_list,
        to_bind:e_list)),
            arg_st:[(loc:(cat:(head:noun,
         val:(subj:e_list,
                                  comps:e_list)),
                         cont:index:Index1)),
                    (loc:(cat:(head:noun,
         val:(subj:e_list,
                                  comps:e_list)),
                         cont:index:Index2)),
                    (loc:(cat:(head:(prep,
                                     pform:to),
         val:(subj:e_list,
                                  comps:e_list)),
                         cont:index:Index3))]).


%%% AUXILIARIES

% LE of is (passive auxiliary)

be ~~> (synsem:(loc:(cat:head:(verb,
              vform:base,
                               pred:plus,
                               aux:plus),
                     cont:Cont),
```

```
         nonloc:(inherited:e_list,
          to_bind:e_list)),
       arg_st:[(Synsem),(loc:(cat:(head:(verb,
                                         vform:pas),
                                   val:(subj:[(Synsem)],
                                        comps:e_list)),
        cont:Cont))]).

% LE of will (future auxiliary)

will ~~> (synsem:(loc:(cat:head:(verb,
             vform:fin,
                            pred:plus,
                            aux:plus),
                      cont:(future_rel,
                           soa_arg:Cont)),
         nonloc:(inherited:e_list,
          to_bind:e_list)),
       arg_st:[(Synsem),(loc:(cat:(head:(verb,
                                         vform:base),
                                   val:(subj:[(Synsem)],
                                        comps:e_list)),
              cont:Cont))]).


have ~~> (synsem:(loc:(cat:head:(verb,
             vform:base,
                            pred:plus,
                            aux:plus),
                      cont:Cont),
         nonloc:(inherited:e_list,
          to_bind:e_list)),
       arg_st:[(Synsem),(loc:(cat:(head:(verb,
                                         vform:psp),
                                   val:(subj:[(Synsem)],
                                        comps:e_list)),
        cont:Cont))]).

be ~~> (synsem:(loc:(cat:head:(verb,
             vform:base,
                            pred:plus,
                            aux:plus),
                      cont:Cont),
```

```
         nonloc:(inherited:e_list,
        to_bind:e_list)),
      arg_st:[(Synsem),(loc:(cat:(head:(verb,
                                        vform:prp),
                                  val:(subj:[(Synsem)],
                                       comps:e_list)),
       cont:Cont))]).



% PREPOSITIONS

by ~~> (word, synsem:(loc:cat:head:(prep,
                                    pform:by),
                      nonloc:(inherited:[],
                              to_bind:[]))).

to ~~> (word, synsem:(loc:cat:head:(prep,
                                    pform:to),
                      nonloc:(inherited:[],
                              to_bind:[]))).

to ~~> (word,
        synsem:(loc:(cat:head:(prep,
                               mod:(loc:(cat:(head:verb,
                                              val:(subj:ne_list,
                                                   comps:e_list)),
                                         cont:(Cont1))),
                pred:minus,
                                    pform:lexical),
      cont:search_rel(Cont2,Cont1)),
        nonloc:(inherited:[],
                         to_bind:[])),
        arg_st:[(loc:(cat:(head:noun,
                               val:(subj:e_list,
                                    comps:e_list)),
                      cont:index:Cont2))]).

fun search_rel(+,+,-).

search_rel(C,X,Y) if
  when( (X=(move_rel;tense_rel;like_rel;say_rel;give_rel;rain_rel;
            direction_rel;here_rel) ),
```

```
          (und_search_rel(C,(X,(move_rel;tense_rel)),Y);fail)).

und_search_rel(C,X,Y) if (X=move_rel),(Y=(direction_rel,
                                          movement:X,
                                          goal:C)).
und_search_rel(C,X,Y) if (X=(future_rel,
                             soa_arg:Cont)),(Y=(future_rel,
                                                soa_arg:search_rel(C,Cont))).
und_search_rel(C,X,Y) if (X=(perfect_rel,
                             soa_arg:Cont)),(Y=(perfect_rel,
                                                soa_arg:search_rel(C,Cont))).
und_search_rel(C,X,Y) if (X=(present_rel,
                             soa_arg:Cont)),(Y=(present_rel,
                                                soa_arg:search_rel(C,Cont))).
und_search_rel(C,X,Y) if (X=(past_rel,
                             soa_arg:Cont)),(Y=(past_rel,
                                                soa_arg:search_rel(C,Cont))).
und_search_rel(C,X,Y) if (X=(cont_rel,
                             soa_arg:Cont)),(Y=(cont_rel,
                                                soa_arg:search_rel(C,Cont))).

empty  word,
       phon:e_list,
       synsem:(loc:(Local,
                    cat:head:(prep,
    mod:none,
            pred:minus,
                            pform:(to;by))),  % restricted to the 2 functional
              nonloc:(inherited:[Local],      % prepositions in the fragment
                      to_bind:e_list)),
       arg_st:e_list.           % the specification arg_st:e_list excludes
                                % heads in prepositional phrases from being
                                % extracted because it entails that the
                                % extracted element is fully saturated


% ADVERBS

here ~~> (synsem:(loc:(cat:head:(adv,
         pred:minus,
                             mod:loc:(cat:(head:verb,
 val:subj:ne_list),
                                      cont:Cont)),
```

```
                    cont:(here_rel,
                         located:Cont)),
           nonloc:(inherited:e_list,
                          to_bind:e_list)),
        arg_st:[]).


% COMPLEMENTIZERS

that ~~> (synsem:(loc:(cat:head:(functional,
         pred:minus,
         vform:Vform,
 mod:none,
                                  marking:that),
                       cont:Cont),
                 nonloc:(inherited:e_list,
                         to_bind:e_list)),
          arg_st:[(loc:(cat:(head:(verb,
 vform:(Vform,fin)),
   val:(subj:e_list,
                                 comps:e_list)),
      cont:Cont))]).



%=============================================================================
% Lexical Rules

% PSP Lexical Rule

psp_lex_rule ##
(word,
 synsem:(loc:(cat:head:(vform:base,
       aux:Aux,
                       pred:Pred),
      cont:Cont),
         nonloc:Nonloc),
 arg_st:Arg)
 **>
(synsem:(loc:(cat:head:(vform:psp,
       aux:Aux,
                       pred:Pred),
             cont:(perfect_rel,
```

```
                    soa_arg:Cont)),
         nonloc:Nonloc),
 arg_st:Arg)
morphs
be becomes been,
give becomes given,
have becomes had,
say becomes said,
(X,[e]) becomes (X,ed),
X becomes (X,ed).

% Present Participle Lexical Rule

prp_lex_rule ##
(word,
 synsem:(loc:(cat:head:(vform:base,
       aux:Aux,
                        pred:Pred),
     cont:Cont),
         nonloc:Nonloc),
 arg_st:Arg)
 **>
(synsem:(loc:(cat:head:(vform:prp,
       aux:Aux,
                        pred:Pred),
              cont:(cont_rel,
                    soa_arg:Cont)),
         nonloc:Nonloc),
 arg_st:Arg)
morphs
be becomes being,
(X,[e]) becomes (X,ing),
X becomes (X,ing).


% Passive Lexical Rule

passive_lex_rule ##
(word,
 synsem:(loc:(cat:head:(verb,
       vform:psp,
                     aux:(Aux,minus),      % no passive of auxiliaries
                     inv:Inv,
```

```
                    pred:Pred),
  cont:soa_arg:Cont),
       nonloc:Nonloc),
 arg_st:[(loc:(cat:(head:noun,
                    val:(subj:e_list,
                         comps:e_list)),
              cont:Cont2)),
         Synsem1|
         List])
 **>
(word,
 synsem:(loc:(cat:head:(verb,
    vform:pas,
                    aux:Aux,
                    inv:Inv,
                    pred:Pred),
  cont:Cont),
         nonloc:Nonloc),
 arg_st:([Synsem1|List];Result))
if append(([Synsem1|List]),[(loc:(cat:(head:(prep,
                                        pform:by),
                                     val:(subj:e_list,
                                          comps:e_list)),
                                  cont:Cont2))],Result)
morphs
X becomes X.


% 3rd_sing_fin_lex_rule

third_sing_fin_lex_rule ##
(word,
 synsem:(loc:(cat:head:(vform:base,
       aux:Aux,
                        pred:Pred),
     cont:Cont),
         nonloc:Nonloc),
 arg_st:Arg)
 **>
(synsem:(loc:(cat:(head:(vform:fin,
       aux:Aux,
                        pred:Pred),
  val:subj:[(loc:cont:(psoa;index:(pers:third,
```

```
                                                        num:sg)))]),
             cont:(present_rel,
                  soa_arg:Cont)),
         nonloc:Nonloc),
 arg_st:Arg)
morphs
be becomes is,
have becomes has,
X becomes (X,s).

% non_3rd_sing_fin_lex_rule

non_third_sing_fin_lex_rule ##
(word,
 synsem:(loc:(cat:head:(vform:base,
       aux:Aux,
                       pred:Pred),
     cont:Cont),
         nonloc:Nonloc),
 arg_st:Arg)
 **>
(synsem:(loc:(cat:(head:(vform:fin,
       aux:Aux,
                       pred:Pred),
  val:subj:[(loc:cont:index:(num:pl;
                                          (pers:(first;second),
                                          num:sg)))]),

             cont:(present_rel,
                   soa_arg:Cont)),
         nonloc:Nonloc),
 arg_st:Arg)
morphs
be becomes are,
X becomes X.


% past_lex_rule no1

non_third__first_sing_past_lex_rule ##
(word,
 synsem:(loc:(cat:head:(vform:base,
       aux:Aux,
                       pred:Pred),
```

```
     cont:Cont),
         nonloc:Nonloc),
 arg_st:Arg)
 **>
(synsem:(loc:(cat:(head:(vform:fin,
       aux:Aux,
                        pred:Pred),
  val:subj:[(loc:cont:index:(num:pl;
                                                (pers:second,
                                                num:sg)))]),
             cont:(past_rel,
                   soa_arg:Cont)),
         nonloc:Nonloc),
 arg_st:Arg)
morphs
be becomes were,
give becomes gave,
have becomes had,
say becomes said,
(X,[e]) becomes (X,ed),
X becomes (X,ed).


% past_lex_rule no2

third__first_sing_past_lex_rule ##
(word,
 synsem:(loc:(cat:head:(vform:base,
       aux:Aux,
                        pred:Pred),
     cont:Cont),
         nonloc:Nonloc),
 arg_st:Arg)
 **>
(synsem:(loc:(cat:(head:(vform:fin,
       aux:Aux,
                        pred:Pred),
  val:subj:[(loc:cont:(psoa;index:(pers:(first;third),
                                                    num:sg)))]),
             cont:(past_rel,
                   soa_arg:Cont)),
         nonloc:Nonloc),
 arg_st:Arg)
```

```
morphs
be becomes was,
give becomes gave,
have becomes had,
say becomes said,
(X,[e]) becomes (X,ed),
X becomes (X,ed).


%=====================================================================
```

Download[20]

6.9 The MERGE

Abstract

The MERGE is a large grammar of English which was developed by W. Detmar Meurers, Kordula De Kuthy and their colleagues at Ohio State University. It is based on a large grammar of English which was written in the LKB.

The MERGE is much larger than any other grammar which we have studied in this course so far. However, after carefully working through this textbook, the reader should by now have acquired sufficient knowledge of the logical foundations of HPSG grammars and of their implementation in TRALE to be in a position to understand even large grammars of the size of MERGE. The links below provide access to the source code of MERGE, which is available in the format of a tared and gzipped directory containing all the necessary files for loading, compiling and running the grammar in TRALE. It is accompanied by extensive documentation in PDF format, *Documentation of the implementation of the MERGE grammar in Trale*.

This document explains the structure of the implemented grammar and the differences compared to its precursor, the *English Resource Grammar* of the LKB system. It provides an overview of the coverage of the grammar, and a specification. It is strongly recommended to read through this document and first get an overview of the grammar before working with it. By studying MERGE the reader will learn more about additional features of the TRALE system, as well as getting to know techniques of grammar engineering which only come into play in large grammars.

[20] `http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/UDC-Grammatik/theory.pl`

Links

- File with the sources of MERGE[21]
- Documentation of the implementation of the MERGE grammar in Trale[22]

[21] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/MERGE/merge-v-1-0-0.tar.gz
[22] http://milca.sfs.uni-tuebingen.de/A4/Course/Grammars/MERGE/merge-doc.pdf

Chapter 7

Glossary

Admission *Admission*, or *feature structure admission*, is a relation between sets of descriptions of our feature logics and sets of (abstract) feature structures. *Admission* is the stronger one of two notions of denotation which we use in our feature logics, and it is based on of the first notion of denotation, satisfaction. A feature structure is admitted by a set of descriptions in the case that every node in the feature structure satisfies each description in the set.

An alternative terminology for *admission* is modeling. If a feature structure is *admitted* by a set of descriptions, we sometimes say that the feature structure models (or is licensed by) the set of descriptions.

ALE *ALE* (Attribute Logic Engine) is a grammar implementation platform for grammars inspired by HPSG. It evolved around the logics presented in a book by Bob Carpenter in the early 1990s. The ALE system underlies the TRALE system which we use in this course.

Anaphor A *synsem* object is an anaphor, provided its LOCAL | CONTENT value is of sort *ana*. Anaphors are a type of pronouns which require the presence of another syntactic element in a close syntactic vicinity which refers to the same object in the world. Typical *anaphors* of English are *herself*, *himself*, and *each other*.

Appropriateness Function The *appropriateness function* is an important component of each signature of an HPSG grammar. The *appropriateness function* determines which attribute is appropriate to which sorts, and it also fixes the sort values which we get for an attribute at a given sort.

Conditions imposed on the appropriateness function by the definition of signatures yield the effect of multiple inheritance.

Atomic Sort A sort is called *atomic* in the case that there are no attributes appropriate to it (nor to its subsorts, if there are any).

Attributes The *attributes*, or *attribute symbols*, of an HPSG grammar are used to capture the properties of linguistic objects. From a more technical point of view we can picture attributes as symbols of the description language of grammars and as labels for the arcs in feature structures. The word *attribute* is used as a synonym of the word *feature* in the HPSG literature.

Attribute Inheritance *Attribute inheritance* is a very common term for describing the fact that an attribute which is appropriate to a sort is also appropriate to all subsorts of this sort in the sort hierarchy.

Basis Set The *basis set* of an abstract feature structure is a set of path. It shows which paths one can follow in the feature structure starting from its (abstract) root node.

Binary Relation A binary relation is a relation with two arguments. See also relation.

Boxes *Boxes* are syntactic entities in our formal languages for HPSG grammars. They roughly correspond to the attribute value matrices of the linguistic literature.

Cartesian Product Let S_1 and S_2 be sets. We call $\{(s_1, s_2)|s_1 \in S_1, s_2 \in S_2\}$ the (binary) Cartesian Product of S_1 and S_2, and write $S_1 \times S_2$. In a tuple (s_1, s_2), s_1 is called the first component and s_2 is called the second component of (s_1, s_2). Binary Cartesian products are generalized to n-fold Cartesian products by taking n sets and forming $S_1 \times \ldots \times S_n$.

Coindexed Two *synsem* objects are *coindexed*, provided that their LOCAL | CONTENT | INDEX values are token-identical.

Constraint-based Grammar Frameworks The term *constraint-based grammar frameworks* refers to grammar frameworks which in contrast to other frameworks use a collection of constraints for characterizing the structure of natural languages. It does not include any claim about how these constraints conspire to make predictions. In this textbook we have a more specific use of the term in mind: We use it to refer to grammar frameworks which view the constraints as expressions of a logical language and employ a logical interpretation of the expressions in the characterization of the models denoted by grammars. In this sense, all grammar formalisms which we define in the text are *constraint-based*, and they are distinguished from unification-based grammar frameworks.

Constraint Satisfaction See satisfaction.

CONTENT **Raiser** We call words whose CONTENT value is identical to the CONTENT value of one of their syntactic arguments CONTENT *raisers*, in analogy to raising verbs, which subcategorize for elements which are arguments of their complements. The semantics of CONTENT *raisers* is identical to the semantics of one of their syntactic arguments.

ConTroll The *ConTroll* system is a grammar implementation platform for HPSG grammars which was developed in Tübingen in the mid-1990s. Its most prominent goal was to get as close to typical linguistic HPSG grammar specification as possible. The experiences with the *ConTroll* system went directly into the creation of the subsequent TRALE system.

Declarative *Declarative* grammar frameworks characterize language by making statements about them which are interpreted simultaneously, without any order imposed on their application, and without referring to procedures which interpret the statements. Constraint-based as well as unification-based grammar frameworks are *declarative*. Transformational grammars are typically not stated declaratively.

Description The term *description* is often used in a technical sense in our textbook, and refers to those formulae of our formal languages which do not contain any free occurrences of variables. A variable is said to occur free in a formula if it is not bound by a quantifier.

Equivalence Class An *equivalence class* of an equivalence relation $\eta = \langle S, \circ \rangle$ is defined as a non-empty subset S' of S such that for each $x \in S'$ and each $y \in S'$, $x \circ y$, but for no element z of $S \backslash S'$ and any $x \in S'$, $z \circ x$.

Equivalence Relation An equivalence relation is a set s together with a relation, $\circ$, in S, where $\circ$ has the following properties:

1. $\circ$ is a binary relation,
2. for each $x \in S$, $x \circ x$ ($\circ$ is reflexive),
3. for each $x \in S$, for each $y \in S$, for each $z \in S$, if $x \circ y$ and $y \circ z$ then $x \circ z$ ($\circ$ is transitive),
4. for each $x \in S$, for each $y \in S$, if $x \circ y$ then $y \circ x$ ($\circ$ is symmetric).

Exhaustive Models *Exhaustive models* are a particular class of models of HPSG grammars, which Paul King introduced to explain the meaning of grammars. Exhaustive models are thought of as containing all possible utterance tokens of a language.

Features See attributes.

Feature Declarations The *feature declarations* of Pollard and Sag's original grammar of English correspond to the declaration of the set of attribute symbols (or simply attributes), and the appropriateness function of an HPSG grammar.

Feature Introduction Condition The *feature introduction condition* is a restriction on the relationship between sorts in the sort hierarchy, and attributes in TRALE signatures. It takes the shape of a restriction on the appropriateness function:

For each attribute which is appropriate to a sort there must be a unique sort σ in the sort hierarchy, such that (a), the attribute is appropriate to σ, and, (b), σ subsumes all other sorts to which the attribute is appropriate.

Feature Structures *Feature structures* are a central concept of many HPSG formalisms, although they are not a necessary part of HPSG. The situation is made even more puzzling by numerous different possible ways of making the notion of *feature structure* mathematically precise. Different definitions of *feature structures* determine their properties in very different ways, which make them appropriate or inappropriate for different purposes. Whenever *feature structures* are used in the linguistic literature it is therefore advisable to find out what particular kind of *feature structure* is meant.

In the formalisms of this textbook *feature structures* are always defined as being sort-resolved and totally well-typed. *Feature structures* with these properties are suitable to be used as complete representations of expressions of natural languages. As such they serve as elements in the denotations of grammars. A good illustration of this situation is to imagine *feature structures* as structured objects which are mathematical idealizations of expressions of languages.

There is an important technical distinction between *concrete* and *abstract* feature structures. Concrete feature structures are defined as a kind of finite state automata, whereas abstract feature structures may be viewed as equivalence classes of *concrete feature structures*, although this is not at first obvious from the definitions. The advantage of *concrete feature structures* is that they are very easy to picture. This is the reason why they are typically used in introductory examples, and also in the MoMo program. Unfortunately it is the more difficult *abstract feature structures* which are of theoretical interest, since their properties can be considered to correspond to object types of expressions of natural languages.

As explained in the section 2.2.2, *Meaning*, of the textbook, sets of *abstract feature structures* are just one possible way known to date of making the meaning of HPSG grammars precise. A semantics in terms of *abstract feature structures* makes the philosophical decision to regard the objective of a constraint-based grammar as a characterization of the object types of the natural language. The object types of the language are represented as *abstract feature structures*. The notion of *object types* and alternative conceptions of the meaning of a grammar are discussed in the section 2.2.2, *Meaning*.

A final area of application for *feature structures* is computation. It is important to realize that the reasons for applying them in computation are completely independent of

the reasons for considering them as representations of object types of natural languages. This means that one can legitimately employ *feature structures* in a computational implementation for HPSG grammars without believing in a model theory of HPSG with *feature structure* representations. *Feature structures* have a long history in computational linguistics. They have been widely studied as data structures for grammar implementations, and there are efficient algorithms for computing with *feature structures*. However, the *feature structures* of computational linguistics are typically partial *feature structures*: These *feature structures* are usually neither sort-resolved nor totally well-typed. The *feature structures* used for computation correspond rather to the descriptions of our formalisms of constraint-based HPSG, although they have to be conceived of as descriptions with a much more restricted syntax than the descriptions we use. The *feature structures* of TRALE are of this kind.

Finite Set A finite set S is a set with finitely many elements.

Function Let S_1 and S_2 be sets. A (partial) *function* f is a recipe that assigns some $s_1 \in S_1$ exactly one $s_2 \in S_2$. We write $f(s_1) = s_2$ and call s_2 the image of s_1 under f.
Partial functions include the special case of total function.

Functional Preposition *Functional preposition* is another term for non lexical preposition. It stresses the functional nature of these lexical items.

Grammar The general notion of a *grammar* receives a very specific mathematical interpretation in constraint-based theories of language. According to the strictly constraint-based view, a *grammar* consists of a signature and a set of grammatical principles, which is usually called the *theory* of the grammar. A crucial task of the logical signature is to provide the vocabulary of non-logical symbols for writing the principles of the theory.

Grammar Development Environment A *grammar development environment* (GDE) is a programming environment for implementing grammars. GDEs are usually inspired by a particular linguistic framework, and support the special needs of this framework. A GDE may comprise specification languages for grammars, (graphical) user interfaces, predicates for grammar development and grammar testing, and debugging tools. The LKB and TRALE are examples of GDEs which were developed to support the implementation of grammars inspired by HPSG.

GRISU *GRISU* is the grammar visualization tool which is used as the graphical user interface of TRALE. Its main purpose in combination with TRALE is to display parse results and offer the user the possibility of inspecting AVMs. Its functionalities are described at length in the user's manual.

iff *iff* is a conventional abbreviation for *if and only if*.

Implementation Platform An *implementation platform* for grammars is a programming environment for developing grammars on a computer. An *implementation platform* provides a specification language for grammars and a set of tools for developing, testing, and viewing grammars. The main purpose of an *implementation platform* is usually to write grammars for parsing and generating sentences.

Infinitary Disjunction An *infinitary disjunction* is a disjunction with infinitely many disjuncts. Our formal languages only allow finite expressions. Therefore they do not comprise *infinitary disjunctions*.

Infinite Feature Structure An *infinite feature structure* is a feature structure with infinitely many nodes.

Insertion Σ insertions play the role of variable assignment functions in our formal languages. Σ insertions map variables and the reserved symbol \$ on sequences of attributes. \$ is always mapped on the empty sequence of attributes.

Label Function The *label function* of an abstract feature structure assigns a species to each (abstract) node of the feature structure.

Lexical Entries *Lexical entries* are descriptions of words, or, to be more precise, descriptions which are satisfied by feature structures of sort *word*. *Lexical entries* are disjuncts in the consequent of the WORD PRINCIPLE.

Lexical Generalization Grammatical principles which characterize the properties of entire classes of words or relate the properties of one class of words to the properties of another class of words are sometimes called *lexical generalizations*. Examples of *lexical generalizations* are the FUNCTIONAL PREPOSITION PRINCIPLE of the series of grammars which starts in Section *The Core Fragment* (Section 3.2.1.1), and the lexical rules of Fragment II (Section 3.2.2.1).

Lexical Principle *Lexical principles* are a particular kind of lexical generalizations. They are lexical generalizations which are formulated as implicational descriptions, just as all other principles of grammar. Lexical rules are a second kind of lexical generalization, but they are not notated as implicational descriptions.

Lexical Preposition *Lexical prepositions* are prepositions of English with a clear semantic contribution, such as the preposition *to* in *She walked over to Mary*. They can be distinguished from non lexical prepositions, which do not have a discernible semantics of their own.

Lexical Rules *Lexical rules* express generalizations over the words in natural languages. The idea is to capture relationships which are observed between lexical items with the purpose of not having to list each form of each word separately in the grammar.

In constraint-based grammar theories such as HPSG, there are at least two technically different ways of making the notion of *lexical rules* precise. They are discussed in Section 3.2.2.2, *Theories of Lexical Rules in HPSG*.

Lexicon We call the collection of lexical entries of a grammar the *lexicon*. Note that this means that the *lexicon* consists of a collection of descriptions.

Licensing The term *licensing* is used as a synonym for admission and describes a relationship between sets of descriptions and sets of feature structures. To say that a feature structure is licensed by (a set of) descriptions means the same as saying that a feature structure is admitted by (a set of) descriptions.

Linearization Grammars HPSG grammars in which the relationship between the phonology of a phrase and the phonology of its daughters is more complex than a relationship of concatenation are often called *linearization grammars*. In grammars based on phrase structure rules, the phonology of phrases is always a concatenation of the phonologies of their daughters. For this reason a linearization grammar cannot be expressed in parsing systems which rely on phrase structure rules. TRALE provides a topological parser for the implementation of *linearization grammars*.

List Signature We use the term *list signature* to refer to the part of a sort hierarchy together with the two attribute symbols and the relevant part of the appropriateness function of a signature that are interpreted as lists and provide the non-logical symbols for the description of lists. Obviously the particular choice of sort symbols and of attribute symbols in list signatures is arbitrary. MoMo provides a special notation which simplifies the notation of descriptions of lists by allowing an alternative notation with brackets for descriptions which would normally have to employ the non-logical symbols of *list signatures*.

LKB *LKB* is a very popular and successful grammar implementation platform for HPSG-inspired grammars. It has been and is still being developed by a very active group of researchers whose main interest is in the fast and efficient processing of large grammars.

Modeling *Modeling* is used as a synonym for being licensed by or being admitted by a set of descriptions. Saying that a feature structure *models* a set of descriptions is therefore equivalent to saying that it is admitted or licensed by by the set of descriptions.

Model Theory *Model theory* is a branch of mathematics which is concerned with structures that interpret logical languages. It investigates the relationships between logical

expressions and the interpreting structures, and it investigates the properties of the interpreting structures.

Multiple Inheritance *Multiple inheritance* is a term which is often used in discussions on the properties of the sort hierarchies of HPSG grammars. *Multiple inheritance* is a consequence of ordering the sorts in the partial order of the sort hierarchy and of the properties of the appropriateness function. Informally speaking *multiple inheritance* describes the fact that a sort which is a subsort of several other sorts, which do not subsume each other, *inherits* its possible attribute values from all of these sorts. This is due to the fact that in accordance with the definition of the appropriateness function, all attribute values at a given sort must be at least as specific as the corresponding attribute value specification at each supersort; and all attributes which are defined for a supersort must be appropriate to it.

In informal discussions of the topic, *multiple inheritance* refers to "inheritance of information" from different supersorts σ_1 and σ_2 of some sort σ_0, where σ_1 and σ_2 do not stand in a mutual subsort relationship. Thus, according to the appropriateness function, σ_0 will "inherit" all appropriateness conditions of both σ_1 and σ_2. Moreover, if there is an implicational description δ in the grammar whose antecedent is a description of feature structures in the denotation of sort σ_1 or σ_2, the possible shape of feature structures of sort σ_0 will be directly restricted by it in the sense that they have to satisfy the consequent of δ. In this sense it is sometimes loosely said that σ_0 inherits the restrictions on σ_1 and σ_2.

Non-derivational *Non-derivational* grammar frameworks do not use *derivation* rules such as phrase structure rules to characterize the structure of languages. Instead they use statements which are applied simultaneously without a procedural order, as can be observed with derivations.

Non-pronoun A *synsem* object is a *non-pronoun*, provided its LOCAL | CONTENT value is of sort *npro*. Within HPSG grammar, the term *non-pronoun* refers to nominal signs which are not pronouns. Pronouns are in turn divided into anaphors and personal pronouns. *Non-pronouns* behave differently from pronouns with respect to where they allow co-referring elements to occur in their syntactic environment.

Non lexical Preposition *Non lexical prepositions*, or functional prepositions, is another term for the case marking prepositions of English, such as the preposition *to* in *He gave a book to Mary*. *Non lexical prepositions* do not seem to make a semantic contribution to utterances.

Object Types The *object types*, or simply *types* of a natural language, are unique mathematical idealizations of expressions of the language. Uniqueness means that for any utterance of a certain form there is exactly one object type which represents it. This stands

in contrast to a theory of meaning which tries to capture utterance tokens: There might be an arbitrary number of utterance tokens for the exact same expression of the language.

The *object types* are formulated as abstract feature structures in the HPSG framework.

Ordered Pair See pair.

Pair A *pair* of structures can be viewed as an enumeration of two structures.

Parametric Sorts *Parametric sorts* are a device which is used in HPSG to simplify the notation for lists with elements of certain sorts in the signature of grammars. Their purpose, notation, and an interpretation of this notation are explained in Section 2.5.2 of the textbook.

Partial Function Let S_1 and S_2 be sets. A partial function f is a recipe which assigns to some $s_1 \in S_1$ exactly one $s_2 \in S_2$. We write $f(s_1) = s_2$ and call s_2 the image of s_1 under f.

In contrast to total functions, partial functions are undefined on some elements of S_1. In other words, there are elements $s \in S_1$ which f fails to map on an element of S_2.

Partial Order A *partial order* is a set S together with a relation, $\circ$, on S, where $\circ$ has the following properties:

1. $\circ$ is a binary relation,
2. $\circ$ is reflexive: for each $x \in S, x \circ x$,
3. $\circ$ is transitive: for each $x \in S$, for each $y \in S$, for each $z \in S$, if $x \circ y$ and $y \circ z$ then $y \circ z$,
4. $\circ$ is antisymmetric: for each $x \in S$, for each $y \in S$, if $x \circ y$ and $y \circ x$ then $x = y$.

$\circ$ is thus a reflexive, transitive and antisymmetric relation on the carrier set, S. We write $\langle S, \circ \rangle$. In comparison to a preorder, a *partial order* has the additional property of being antisymmetric.

Partitions The *partitions* in Pollard and Sag's original grammar of English can be interpreted as a notation for the sort hierarchy of their grammar.

Path A *path* is defined as a (possibly empty) sequence of attributes.

Personal Pronoun A *synsem* object is a *personal pronoun*, provided its LOCAL | CONTENT value is of sort *ppro*. *Personal pronouns* are pronouns such as *she* and *her*. In contrast to anaphors, they typically forbid the presence of syntactic elements which refer to the same entity in the real world in a certain syntactic vicinity.

Power Set The *Power Set* of a set S is the set of all subsets of S. Note that the set of subsets of any set includes the empty set. The *power set* of $\{1, 2\}$ is thus the set $\{\{\}, \{1\}, \{2\}, \{1, 2\}\}$.

Preorder A *preorder* is a set S together with a relation, $\circ$, on S, where $\circ$ has the following three properties:

1. $\circ$ is a binary relation,
2. $\circ$ is reflexive: for each $x \in S, x \circ x$,
3. $\circ$ is transitive: for each $x \in S$, for each $y \in S$, for each $z \in S$, if $x \circ y$ and $y \circ z$ then $y \circ z$.

The relation $\circ$ is thus a reflexive and transitive relation on the carrier set, S. We write $\langle S, \circ \rangle$ for *preorders*.

Principles The *principles* of a grammar are the generalizations about language which linguists formulate in order to capture their empirical observations. In constraint-based grammar formalisms the *principles* are typically formulated as logical implications.

Quadruple A *quadruple* of structures can be viewed as an enumeration of four structures.

Quintuple A *quintuple* of structures can be viewed as an enumeration of five structures.

Reducts The *reducts* of an abstract feature structure are the abstract feature structures one obtains by following an arbitrary path from the root node of the original feature structure and taking the abstract node at which one arrives as the root node of a new (typically smaller) abstract feature structure.

Re-entrancy Relation The *re-entrancy relation* of an abstract feature structure is an equivalence relation over the basis set of the feature structure. The equivalence classes of the *re-entrancy relation* are the (abstract) nodes of the feature structure.

Referential A *synsem* object is *referential*, provided its LOCAL | CONTENT | INDEX value is of sort *ref*.

Relation Suppose that $S, S_1 \ldots S_n$ are sets. An n-ary relation R in $S_1 \ldots S_n$ is a subset of $S_1 \times \ldots \times S_n$. We write this as $R \subseteq S_1 \times \ldots \times S_n$. A very common special case of this are binary relations R' in S, $R' \subseteq S \times S$.

Relation Extension The *relation extension* is a part of the relational abstract feature structures. It is needed in order to give relational formulae a denotation.

Root Node The *root node* of a feature structure is a distinguished node from which all other nodes in the feature structure can be reached by following a sequence of attribute arcs. It corresponds to the start node of a finite state automaton.

Satisfaction *Satisfaction* is a relation between an expression of a logical language and a structure. When we say in a feature logic that a structure *satisfies* a logical expression, we mean that the structure is in a sense described by the expression. This is also what is meant by *constraint satisfaction*. In our feature logics for HPSG, *satisfaction* is the weaker one of two notions of denotation. *Satisfaction* is defined relative to the root node of feature structures, whereas admission is a relation for which all nodes of a feature structure are relevant.

Septuple A *septuple* of structures can be viewed as an enumeration of seven structures.

Set *Sets* are a fundamental concept of mathematics. Mathematics as a whole can be based on the notion of sets. There are different interesting ways of axiomatizing sets. However, for the purpose of working with sets in our course, it is not necessary to study possible axiomatizations of sets. In fact, doing so would in all likelihood be more confusing than enlightening.

For us, an intuitive understanding of the notion of a set along the lines of what Georg Cantor (1845–1918) wrote over a century ago suffices: "A *set* is a collection of certain non-identical objects perceived or thought of by us. The objects involved are called the *elements* of the set."[1]

Signature The *signature* of logical languages provides the set(s) of non-logical symbols from which expressions of the language can be formed. In HPSG, signatures provide additional structure by imposing structure on the set of sorts in the sort hierarchy, and relating sorts and attributes in the appropriateness function.

Smallest Set Speaking of a *smallest set* is a typical technical (and conventionalized) way of expressing an inductive definition in mathematics. When we say that something is the smallest set for which certain properties hold, we mean to say that the conditions we stipulate for something to be a member of that set are exhaustive. We obtain the members of the set by following the given rules exactly, and nothing which cannot be obtained in this way may be included in the set.

Sort Hierarchy The *sort hierarchy* of HPSG grammars is a partial order of the set of sort symbols. The maximal specific sorts in the *sort hierarchy* are those sorts which do not

[1] Eine *Menge* ist eine Zusammenfassung von bestimmten wohlunterschiedenen Objekten unserer Anschauung oder unseres Denkens zu einem Ganzen. Die dabei auftauchenden Objekte werden *Elemente* der Menge genannt.

stand above any other sort in the sort hierarchy. These sorts are used to label the nodes in feature structures.

Sort-Resolvedness *Sort-resolvedness* is a property of the feature structures which are used as models of grammars. A feature structure is said to be *sort-resolved* if and only if every node of the feature structure is labeled by a maximally specific sort or species. These are those sorts in the sort hierarchy which do not have any proper subsort.

Sort Symbols The *sort symbols* are stated specifically for each HPSG grammar. They are symbols for describing the objects in the denotation of the grammar. The *sort symbols* are ordered in a taxonomic hierarchy, the sort hierarchy.

Species *Species* is a special name for those sorts in the sort hierarchy which do not have any proper subsort. The *species* label the nodes of feature structures. They are also called maximally specific sorts.

Strong Generative Capacity In a paper on the formalization of HPSG, Carl Pollard defined the *strong generative capacity* (SGC) of a grammar as a set of structures which are thought of as mathematical idealizations of the utterances of natural languages. The SGC of a grammar represents the predictions of the grammar, and replaces the earlier explanation of the meaning of HPSG grammars in terms of the abstract feature structures in Pollard and Sag's HPSG book from 1994. Another option for explaining the meaning of HPSG grammars are Paul King's exhaustive models. It should be noted that Pollard's use of the term *strong generative capacity* is only loosely related to the Chomskyan terminology of weak and strong generative capacity.

Syntactic Sugar Additional conventions for writing down expressions of a formal language are sometimes called *syntactic sugar*. This is intended to simplify the notation.

Tag The boxed numbers (□) in the AVM descriptions of HPSG are usually called *tags*. In our formalization of the description language, tags are treated as variables.

Theory The term *theory* is quite often used in a very specific technical sense when we talk about constraint-based grammars. In contrast to its general meaning, we often use it to refer to the set of principles of a grammar.

Total Function Let S_1 and S_2 be sets. A total function f is a recipe which assigns to each $s_1 \in S_1$ exactly one $s_2 \in S_2$. We write $f(s_1) = s_2$ and call s_2 the image of s_1 under f.

In contrast to Partial functions, *total functions* assign to each element in S_1 an element in S_2. Partial functions might fail to assign an image in S_2 to some elements of S_1. They are undefined on those elements of S_1.

Totally well-typed In HPSG *total well-typedness* is a property which is ascribed to feature structures as models of linguistic objects.

Ontologically, total well-typedness expresses the assumption that all well-formed linguistic objects are complete objects. Linguistic objects do not have an aspect of partiality. All linguistic objects always have all types of properties which they may have. In other words, each possible feature of each object necessarily has one of its possible values. The situation might arise in which we cannot observe all features and all values, but we know that they are always there.

Mathematically, the idea of the completeness of linguistic signs is expressed by imposing the condition of total well-typedness on the modeling domain of grammars. The modeling domains which we chose for all our grammar formalisms consisted of sets of (slightly different kinds of) feature structures. We say that a feature structure is *totally well-typed* if and only if each node in the feature structure has outgoing arcs for each of the attributes which the signature declares appropriate to a node of the given sort, and there are only appropriate outgoing arcs at each node of the feature structure.

See also well-typed.

TRALE *TRALE* is a grammar implementation platform for HPSG grammars. It is based on the ALE system, but importantly adds a number of logical assumptions about the interpretation of constraints which are specific to HPSG. In addition, the TRALE environment provides many additional functions for grammar development.

Triple A *triple* of structures can be viewed as an enumeration of three structures.

Tuple A *tuple* of structures can be viewed as an enumeration of a fixed number of structures.

Unbounded Dependency Constructions *Unbounded dependency constructions* have been extensively discussed in the literature on generative grammar frameworks, and HPSG is no exception to this rule. Despite the extensive literature on the subject *unbounded dependency constructions* (which are also known as *long movement constructions*) are still a very lively field of research.

With the term *unbounded dependencies* linguists refer to constructions in which a constituent appears in a sentence peripheral position distinct from the position in which the constituent would appear in 'normal' or 'unmarked' word order. The dislocated constituent may be several sentence boundaries away from its 'unmarked' position. Here are three examples:

1. [Which book]$_i$ do you think Mary read t_i first?
2. [This man]$_i$ nobody would believe that anybody would nominate t_i for president.
3. [Who]$_i$ did you say Hillary claimed that Bill dated t_i last summer?

Unification *Unification* is an algebraic operation on objects in an algebra. It became popular in linguistics for handling phrase structure grammars which were augmented with features and feature values.

Unification-based Grammar Frameworks Originally the term *unification-based grammar frameworks* was used to refer to grammar frameworks in which a unification operation was of central importance, such as Generalized Phrase Structure Grammar, Lexical Functional Grammar, Categorial Unification Grammar and early Head-Driven Phrase Structure Grammar. Nowadays the term is occasionally used more loosely, and is also applied to grammar frameworks which use features and feature values in some way, or to frameworks whose typical grammar implementation platforms make use of algorithms and programming languages, such as Prolog, which rely on unification. According to this more relaxed terminology even present-day HPSG, which does not involve unification in its mathematical foundations, might be classified as *unification-based*. In this textbook we are more strict and restrict the term *unification-based* to those frameworks which actually employ unification.

Well-typed The notion of *well-typedness* is slightly weaker than the related notion of total well-typedness.

From an ontological point of view, it makes precise the assumption that objects of a certain kind may only have properties of a limited range. Some of the possible properties may be missing from a particular token of the relevant kind of object, but a token will never have any unexpected properties.

In the modeling domain of feature structures well-typedness is expressed on the basis of the presence of attribute arcs at nodes. We say that a feature structure is *well-typed* if and only if every attribute arc at each node of a sort σ is labeled by an attribute which is appropriate to sort σ according to the signature.

$\mathbb{N}$ In our text, the symbol $\mathbb{N}$ is used for the set of natural numbers including zero. Note that the use of the symbol $\mathbb{N}$ varies in the literature. Sometimes the set which we describe with the symbol $\mathbb{N}$ is described by $\mathbb{N}_0$, and $\mathbb{N}$ is reserved for the set of positive integers. This choice depends on mere conventions and on convenience in a given context.

$\mathbb{N}^+$ In our text, the symbol $\mathbb{N}^+$ is used for the set of natural numbers excluding zero.

Note that we use $\mathbb{N}$ for the set of natural numbers *including* zero.

Bibliography

[Aldag, 1997] Aldag, Bjørn 1997. A proof theoretic investigation of prediction in HPSG. Magisterarbeit, Seminar für Sprachwissenschaft: Universität Tübingen. 126

[Bresnan, 1982] Bresnan, Joan (ed) 1982. *The Mental Representation of Grammatical Relations*. Cambridge, MA, USA: MIT Press. 8

[Carpenter, 1992] Carpenter, Bob 1992. *The Logic of Typed Feature Structures*. Cambridge University Press. Cambridge, Massachusetts, USA. 9

[Chomsky, 1981] Chomsky, Noam 1981. *Lectures on Government and Binding*. Dordrecht, The Netherlands: Foris Publications. 8

[Ebbinghaus et al., 1992] Ebbinghaus, Heinz-Dieter, Flum, Jörg, and Thomas, Wolfgang 1992. *Einführung in die mathematische Logik*. B.I.-Wissenschaftsverlag, 3rd edition. 25

[Gallin, 1975] Gallin, Daniel 1975. *Intensional and Higher-Order Modal Logic*. North-Holland, Amsterdam. 124

[Gazdar et al., 1985] Gazdar, Gerald, Klein, Ewan, Pullum, Geoffrey K., and Sag, Ivan 1985. *Generalized Phrase Structure Grammar*. Harvard University Press. Cambridge Massachusetts. 8

[Höhle, 1999] Höhle, Tilman N. 1999. An Architecture for Phonology. In Robert D. Borsley and Adam Przepiórkowski (eds), *Slavic in HPSG*, 61–90. CSLI Publications. 14

[Kathol, 1995] Kathol, Andreas 1995. *Linearization-Based German Syntax*. PhD thesis, Ohio State University. 156

[King, 1999] King, Paul J. 1999. Towards Truth in Head-driven Phrase Structure Grammar. In Valia Kordoni (ed), *Tübingen Studies in Head-Driven Phrase Structure Grammar*, (= *Arbeitspapiere des SFB 340, Nr. 132, Volume 2*), 301–352. Eberhard-Karls-Universität Tübingen. 35, 37

[Meurers, 2000] Meurers, Walt Detmar 2000. *Lexical Generalizations in the Syntax of German Non-Finite Constructions, (= Arbeitspapiere des SFB 340, Nr. 145)*. PhD thesis, Eberhard-Karls-Universität Tübingen. 206, 208

[Meurers and Minnen, 1997] Meurers, W. Detmar and Minnen, Guido 1997. A computational treatment of lexical rules in HPSG as covariation in lexical entries. *Computational Linguistics*, 23.4:543–568. 208, 209

[Moshier, 1988] Moshier, Michael Andrew 1988. *Extensions to Unification Grammar for the Description of Programming Languages*. PhD thesis, University of Michigan. 50

[Pollard and Sag, 1987] Pollard, Carl and Sag, Ivan A. 1987. *Information-Based Syntax and Semantics. Vol.1: Fundamentals*. CSLI Lecture Notes 13. 8, 9

[Pollard and Sag, 1994] Pollard, Carl and Sag, Ivan A. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press. 2, 4, 7, 8, 10, 13, 20, 26, 50, 61, 62, 63, 70, 84, 85, 86, 87, 92, 103, 104, 109, 114, 115, 175, 207, 217, 219, 233, 252, 253, 254, 256

[Pollard, 1999] Pollard, Carl J. 1999. Strong generative capacity in HPSG. In Gert Webelhuth, Jean-Pierre Koenig, and Andreas Kathol (eds), *Lexical and Constructional Aspects of Linguistic Explanation*, 281–297. CSLI Publications. 35, 36

[Richter, 2004] Richter, Frank 2004. *A Mathematical Formalism for Linguistic Theories with an Application in Head-Driven Phrase Structure Grammar*. Phil. dissertation (2000), Eberhard-Karls-Universität Tübingen. 38, 238, 254, 255

[Sag and Wasow, 1999] Sag, Ivan A. and Wasow, Thomas 1999. *Syntactic Theory: A formal introduction*. Stanford, CA: CSLI Publications. 111