Institutt for Lingvistiske Fag Universitet i Oslo

Norwegian Word Order in Head-Driven Phrase Structure Grammar

— Phenomena, Analysis, and Implementation —

HOVEDFAGSOPPGAVE

(June 20, 2003)

submitted by:

Liv Ellingsen
Rostedsgate 2
0178 Oslo (Norge)
livel@hf.uio.no

Contents

Tusen takk

This thesis is a result of far more than a year's work and sweat on my part. Over this time, a lot of people have helped me along, whether they know it or not. I would therefore like to thank them all, now that the days of swearing seem to come to an end.

First of all, I would like to thank my supervisor Jan Tore Lønning for the time and patience he has laid down in order to get me through this thesis. I am very grateful for his continuous advice and open door throughout the whole process, and, not to forget, for the occasional bit of firm but friendly pressure.

Secondly, I would like to thank good friends in and outside of Oslo for making the time spent outside of Blindern enjoyable and relaxing, and friends at Blindern and at Hundremeter'n for adding fun and interest to my studies. Thanks to Elin for all the coffee, to Hege for a great last year and to Leiv for the company.

The main thank, though, goes to my parents Aud and Per, who very much saved this thesis towards the end by spending vast amounts of time, energy, care and love on me when I needed it, in the middle of their already stressful daily life. They found the time to type up two of the sections of this thesis when I was not able to do this myself.

Finally, thanks to Siv and Frøydis for great hospitality and a never-ending supply of dandelions. And to Stephan: takk i lige måde!

1 Introduction

This thesis is based on a course in computational linguistics at the University of Oslo, where we used the text book *Syntactic Theory — A Formal Introduction* by Ivan A. Sag and Thomas Wasow (Sag and Wasow, 1999). The book presents the grammar framework Head-Driven Phrase Structure Grammar (HPSG) and its application to the English language.

One part of the course consisted of implementing a grammar fragment for Norwegian based on the English grammar described in the book. With no tools (but Prolog) at hand, constructing the grammar and its type system from scratch, the grammar fragments ended up small and rudimentary. Still, even at this basic level, interesting questions arose concerning Norwegian word order in contrast to the English model grammar.

It turned out that although many basic constructions can be transferred directly from English to Norwegian, Norwegian also includes basic clause patterns that are not found in English, and that thus need different analyses and implementations. The starting point of interest was the strict V2 constraint in Norwegian declarative main clauses that is central to many Germanic languages but not found in English, in combination with the possibility of topicalization. To be able to combine these two phenomena, subject inversion is far more common in Norwegian than in English, and well worth a closer look.

As a natural continuation, my field of interest soon expanded to cover the structural differences between Norwegian main and subordinate clauses as well. How to account for the latter's rigid SVO structure and at the same time allow for the inverted, topicalized structures of the former in the simplest possible way turned out to be a fascinating problem.

Finally, the differing placement of adverbials in the two clause types forms an important part of the thesis' objectives. Sentence adverbials can not only be placed in different positions in the two clause types, their distributional possibilities also vary from one group of adverbials to the other, within the same clause type. A challenge, in other words, both to describe and to implement.

In the essence, the goal of this thesis is to develop an analysis for Norwegian that covers all of the above-mentioned phenomena concerning Norwegian word order, and additionally to present an efficient implementation of the analysis in HPSG, using a specialized grammar writing tool.

Overview

The first main section of the thesis introduces the exact data for Norwegian that the analysis and the implementation will cover (section 2). As background, two formal frameworks used for describing the clause structure of the mainland Scandinavian languages are reviewed, one (Field Grammar) because it introduces ideas that are later used in the analysis, and the other (Transformational Grammar) because it represents a very common view on

the mechanisms of Scandinavian clause structure that I still choose to oppose against.

The four sections that follow the description of the language data (sections 3 to 6) all present the different theoretical frameworks and tools that I have used to develop the analysis and implement it as an HPSG grammar fragment of Norwegian.

The first of these sections presents the main characteristics of the grammar framework HPSG. It is a non-transformational, constraint-based and mostly lexicalist framework, and the name 'head-driven' reflects the importance of the information encoded in the lexical head of a phrase. Typed feature structures are used to model linguistic objects, and multiple inheritance type hierarchies are used extensively to express generalizations across lexical items as well as phrases and constructions (section 3).

The following section gives a very brief introduction to the semantic framework of Minimal Recursion Semantics (MRS), the semantic representation that is used in the grammar implementation. MRS is not an independent semantic theory, but rather a meta language developed to fit the needs of large computational grammars. It is thus developed with semantic transfer and parsing and generation of natural language in mind. Though not the main subject of this thesis, semantic representations are included in the implementation because MRS is already incorporated in the basic grammar Matrix, on which the Norwegian grammar has been built (section 4).

The third of the four background sections describes the grammar development tool that has beeen used to construct the grammar fragment, namely the Linguistic Knowledge Builder (LKB) system. The original course implementation in Prolog was abandoned at an early stage in favor of the opportunity to use this more advanced grammar development tool. It aids the construction of the various grammar parts and the grammar's type hierarchy, and it also includes various ways of processing grammars, including both parsing and generation. The LKB as a development environment made it possible to build a far more extensive and sophisticated grammar, covering more phenomena than originally intended (section 5).

As the fourth piece of background, section 6 presents the LinGO Grammar Matrix, the basic type hierarchy on which the language-specific Norwegian grammar fragment is built. The types are mainly taken from the HPSG LinGO English Resource Grammar (ERG), but the goal of the Matrix is to develop into a language-independent starter kit for grammar developers that wish to build HPSG grammars for their own language. Said goal and the fact that the type hierarchy is based on English fit very nicely into this thesis' wish to develop a grammar fragment for Norwegian in combination with taking a closer look at some of the differences in the clause structure between Norwegian and English.

As the main part of the thesis, section 7 presents my own analysis of Norwegian word order and the relevant main vs. subordinate clause distinctions. These distinctions include the V2 constraint, topicalization, the strict SVO structure of subordinate clause structures, declarative vs. interrogative

clauses and, finally, adjunct placement in main and subordinate clause structures. Firstly, the theoretical foundations of the analysis are introduced, and it is shown how it accounts for the data described in preceding sections. The analysis takes as its starting point both the linguistic frameworks of Field Grammar and HPSG. It leans on existing work for English, but Norwegian-specific solutions, required by the word order differences between the two languages, are presented. Advantages and weaknesses of these solutions to the challenges of Norwegian clause structure are then discussed.

The implementation of the analysis is presented in more technical detail in section 8. Aspects of the type hierarchy are combined with comments on the underlying motivations and interactions with other parts of the grammar. To keep the section down to a manageable size, only the the relevant aspects of the grammar that are directly related to the analyses of the main phenomena are explained in some detail. This primarily includes the constructions that inherit from the basic constructional types in the Matrix and the hierarchy of adjunct head types used to implement the distributional possibilities of the different groups of adjuncts. Both the changes that had to be made to the types in the Matrix to support my implementation and the types for the Norwegian grammar inheriting from these (adjusted) basic Matrix types are described.

For the really interested and skillful, even further technical details are supplied in various appendices.

2 Surface structure of Norwegian sentences

This section presents the data that I wish to take into account when implementing a grammar fragment for Norwegian with focus on clause structure. I describe the possible structures of Norwegian main and subordinate clauses, revealing the differences between the two clause types that include the V2 constraint, topicalization and the placement of adverbials in a clause. Two attempts at describing or explaining Norwegian (and Swedish and Danish) clause structure in a formal framework are presented at the end of the section. Norsk referansegrammatikk (Faarlund et al., 1997) and the Oslo-Bergen corpora of tagged Norwegian texts[1] have been the main sources used in this section.

Norwegian belongs to the group of Germanic languages and exhibits great similarities to its siblings on many linguistic levels. Syntactically, it shares properties both with the closely related mainland Scandinavian languages Danish and Swedish and with more distant related Germanic languages such as English and German.

2.1 The main clause

2.1.1 Position of the finite verb

One of the main structural characteristics of a Norwegian declarative main clause is the fact that only one phrase may precede the finite verb. This is called the V2 constraint, a characteristic it shares with almost all other Germanic languages with the exception of English. There are few constraints on what type of phrase can be fronted, so although the subject will be expected to front an otherwise unmarked main clause, different types of objects and adverbials can also be placed in this position.

(1) a. Gyrd smilte da Inge lo.
Gyrd smiled when Inge laughed

b. Da Inge lo smilte Gyrd.

c. *Da Inge lo Gyrd smilte.

Also common for the V2 languages is the fact that direct yes/no questions have the finite verb in the first position, while wh-questions follow the V2 pattern, fronted by a wh-word.

(2) a. Smilte Gyrd da Inge lo?

b. Hvem smilte da Inge lo?

c. Når smilte Gyrd?

[1] Web adress:`http://www.tekstlab.uio.no/norsk/bokmaal/`

2.1.2 Objects

With the exception of some personal pronouns, Norwegian words and phrases are not marked morphologically for case. Syntactic roles are instead handed out according to the ordering of constituents in the sentence. The result is a relatively fixed constituent order, as is found in most Germanic languages that have moved away from case marking, the mainland Scandinavian languages and English included. The objects in a Norwegian sentence follow the order of *Indirect Object - Direct Object - Prepositional Object* when placed behind the finite verb, as seen in the sentences in (3)[2].

(3) a. Hun ga Inge boka.
she-nom gave Inge the-book

b. Hun ga boka til Inge.

c. *Hun ga til Inge boka.

2.1.3 The subject

As a consequence of the V2 rule and the possibility of fronting other elements than the subject, the subject itself must have more than one possible position in a Norwegian declarative main clause. If another element in the sentence is structurally stressed by fronting, the subject must find a place for itself after the finite verb. Due to the mentioned lack of case marking in Norwegian, the syntactic role of the subject still has to be determined by the placement of the element in the sentence. The subject will always be placed in the position after the finite verb, preceding all objects, if the first position of the sentence is already taken.

(4) a. Til Inge ga hun boka.

b. Boka ga hun til Inge.

2.1.4 Complex verbals

For main clauses that include both a finite verb and one or more non-finite verbs, the V2 constraint still holds for the finite matrix verb. The infinite verbs are placed before their objects (or other nominal components, like predicatives), following the same pattern as for English main clauses and contrasting with for instance German, where the infinite verbs typically are placed at the end of the sentence. But contrasting again with English, the Norwegian subject will keep its position right after the finite verb in constructions where another element is fronted, and so the finite verb and the infinite verbs can be separated by the subject, as shown in sentence (5b) and sentence (5c). Wh-questions and yes/no questions follow the same patterns.

[2] *Gyrd* is here replaced by the case-marked pronoun *hun* to avoid multiple readings.

(5) a. Gyrd hadde gitt boka til Inge.
Gyrd had given the-book to Inge

b. Boka hadde Gyrd gitt til Inge.

c. Til Inge hadde Gyrd gitt boka.

(6) a. Hvem hadde gitt boka til Inge?

b. Hva hadde Gyrd gitt til Inge?

c. Hadde Gyrd gitt boka til Inge?

2.1.5 Adverbials

The syntactic function adverbial covers a large and far from homogeneous group of expressions. Adverbials can modify a verb, an adjective, adverbs and adverbial phrases, or a clause as a whole.

In this thesis I will concentrate on a small collection of so-called sentence adverbials[3],due to their distributional possibilities in a Norwegian clause. All the sentence adverbials used in my grammar are of the grammatical category adverb, although other sentence adverbials can belong to as different categories as adjective phrases, prepositional phrases and subordinate clauses. I will not go further into the possible semantic or syntactic classifications of adverbs or adverbials than to give a brief presentation of some of the characteristics of what are traditionally referred to as sentence adverbials, versus bound adverbials and free adverbials.

Bound adverbials are obligatory modifiers, bound by the valence of the verb. The type of adverbial is selected for by the verb, as seen in the sentences in (7). Bound adverbials are placed after the verb(s), and usually after a possible object. They can be fronted, and if the verb they are modifying is fronted in combination with the word *gjør/gjorde* (*do/did*), the bound adverbial must be fronted together with it.

(7) a. Hun bor i Oslo .
she live in Oslo

b. *Hun bor.

c. *Hun bor gjennom Oslo.
she live through Oslo

d. I Oslo bor hun.

[3]Because of their distributional possibilities, names like *nexus adverbial* (Diderichsen (1962), see section 2.3) and *central adverbial* have also been used in attempts at classification of sentence adverbials. Different attempts at classification are presented in Heggelund (1981). The classification used in this thesis is based on the classification made in Norsk referansegrammatikk (Faarlund et al., 1997), leaning heavily on Heggelund (1981).

e. Bo i Oslo gjør hun.

f. *Bo gjør hun i Oslo.

Free adverbials are not bound by the verb they are modifying in any way. Possibilities for placement include at the front of a main clause, at sentence end, or between the finite verb and the infinite verbs or the objects of a main clause, on both sides of the subject. Their precise distribution varies considerably according to their grammatical category and type of semantic contribution. Time adverbials are among the more frequent inter-verbal free adverbials, as shown in the sentences in (8), while for instance locative prepositional phrases that function as free adverbials often are restricted to stand at sentence end or in the fronted position. Free adverbials can choose freely whether they want to move with the main verb when fronted, as shown in sentence (8e) and sentece (8f).

(8) a. Gyrd ga boka til Inge i dag.
Gyrd gave the-book to Inge today

b. I dag ga Gyrd boka til Inge .

c. Boka ga Gyrd i dag til Inge.

d. Boka ga i dag Gyrd til Inge.

e. Ga boka til Inge gjorde Gyrd i dag.

f. Ga boka til Inge i dag gjorde Gyrd.

(9) a. Gyrd ga boka til Inge ved bordet.
Gyrd gave the-book to Inge at the-table

b. *Gyrd ga ved bordet boka til Inge.

c. Ved bordet ga Gyrd boka til Inge.

Sentence adverbials are also facultative, and they modify the whole clause to which they attach. They are only loosely bound to the verb, and they can not be fronted together with the main verb of the sentence, as shown in sentence (10h). They usually express denial, confirmation or doubt, or they tie the clause up to some part of the context.

Sentence adverbials are usually placed in the pre- or the post-subject position between the finite verb and the infinite verbs or the objects of a main clause, as shown in the sentences in (10). They can only seldom stand at the end of a sentence unless they are extraposed. Some sentence adverbials are always light elements, and can therefore not be fronted, as seen in sentence (10d). Another group can happily be stressed, and so also fronted, as seen in sentence (11d).The last and small group includes the sentence adverbials that can stand at sentence end in addition to be able to be fronted, as shown in sentence (12e).

(10) a. Gyrd ga sikkert boka til Inge.
Gyrd gave surely the-book to Inge

b. Til Inge ga Gyrd sikkert boka.

c. Boka ga sikkert Gyrd til Inge.

d. *Sikkert ga Gyrd boka til Inge.

e. *Gyrd ga boka til Inge sikkert.

f. ?Gyrd ga boka sikkert til Inge.

g. Ga boka til Inge gjorde sikkert Gyrd.

h. *Ga sikkert boka til Inge gjorde Gyrd.

(11) a. Gyrd ga heldigvis boka til Inge.
Gyrd gave luckily the-book to Inge

b. Boka ga Gyrd heldigvis til Inge.

c. Til Inge ga heldigvis Gyrd boka.

d. Heldigvis ga Gyrd boka til Inge.

e. *Gyrd ga boka til Inge heldigvis.

f. ?Gyrd ga boka heldigvis til Inge.

(12) a. Gyrd ga allikevel boka til Inge.
Gyrd gave still the-book to Inge

b. Boka ga Gyrd allikevel til Inge.

c. Til Inge ga allikevel Gyrd boka.

d. Allikevel ga Gyrd boka til Inge.

e. Gyrd ga boka til Inge allikevel.

f. ?Gyrd ga boka allikevel til Inge.

The sentences (10f), (11f) and (12f) are not marked clearly ungrammatical. Even though a general rule claims that no other element is allowed between the objects or between an infinite verb and an object in a Norwegian clause, it can not be denied that adverbials sometimes are placed in this position. It is difficult to distinguish adverbials that can do this succesfully from those that definitely can not, as it is not just determined by the type of adverbial, but also by the environment into which it is placed.

The important factors concerning this question are stress and length of the elements involved. A light and short adverbial that is normally placed at sentence end, for instance a preposition with no complement, will more likely

than a longer one be able to migrate to the left of a long, heavy object without leaving the sentence ungrammatical. Following this, it is possible that also long and/or heavy sentence adverbials are more successful at migrating to the right of a lighter and shorter object than shorter and lighter ones.

I have no strong intuition about neither of the three sentences mentioned above, but I find sentence (13) (where the adverbial is extremely light and short) unacceptable, although *jo* has the same distribution as *sikkert* in all other cases.

(13) *Gyrd ga boka jo til Inge
Gyrd gave the-book after-all to Inge

This phenomenon might be related to phenomena like heavy-NP shift and scrambling, and it is probable that it is a question of performance rather than of competence, i.e. a question of style rather than a rule belonging to the core grammar, at least in the case of sentence adverbials. For simplicity, I will rule out these dubious cases in the grammar.

In this thesis I will also not take into account an important generalization that can be made concerning adverbials interleaving objects, namely the fact that light pronouns not just can, but usually demand to stand to the left of any sentence adverbials. This phenomenon is called object shift, and takes place in all the Scandinavian mainland languages. Where sentence (13) is ungrammatical, sentence (14a) is clearly grammatical. The sentence in (14b) is also a possible construction, but it implies that a certain stress is laid on *den*, so that it is no longer a light element.

(14) a. Gyrd ga den jo til Inge

b. (*)Gyrd ga jo den til Inge

2.2 The subordinate clause

Common for many of the Germanic languages is that the structure of the subordinate clause differs from that of the main clause. In German and Dutch, the V2 pattern is forsaken and the finite verb placed at the end of the clause in a subclause. Strict structural differences between the two clause types are also found in the mainland Scandinavian languages, although the changes might not seem too dramatic on the surface. The subordinate clause structure following a subjunction like *da* (*when*) in sentence (15) is for instance apparently the same as in a main clause.

(15) [da] Gyrd ga boka til Inge.

One of the main structural differences is that the subject only has one possible position in the subclause, to the left of the finite verb. Objects and adverbials can not be fronted in the same way as in a main clause, as shown

in sentence (16a) and (16b)[4]. This rigid structure accounts for labeling Norwegian an SVO language, a claim widely accepted in contemporary linguistic theories.

(16) a. (*) [da] boka ga Gyrd til Inge.[5]

b. * [da] boka Gyrd ga til Inge.

The other main difference is the distribution of adverbials in the clause. The possible adverbial positions after the finite verb, before any of the objects in a main clause do not exist in a subordinate clause. As a consequence, the questionable placement of sentence adverbials between objects in the main clause is here clearly ungrammatical.

(17) a. * [da] Gyrd ga sikkert boka til Inge.

b. * [da] Gyrd ga boka sikkert til Inge.

c. * [da] Gyrd ga den sikkert til Inge.

Instead, and breaking down the V2 constraint, sentence adverbials can be placed before the finite verb, after the subject, as shown in sentence (18a).

(18) a. [da] Gyrd sikkert har gitt boka til Inge.

The place before the subject is in theory also open for adverbials, but only a small collection of adverbials occupy this position on a regular basis in real life, and then often in connection with specific subjunctions. The sentence adverbial *ikke* (*not*) is among the most frequent occupants of the position, in combination with subjunctions like *hvis* (*if*).

Other frequently occurring adverbials include for instance *bare* (*only*) and *også* (*too*). These adverbials are troublesome, as they belong to the group of focussing adverbs, a group of adverbs that can modify entities on different levels in a sentence. Due to this, their distributional possibilities are more complex than for sentence adverbials, which can only modify the clause as a whole. When placed in the pre-subject position, it can be hard to decide whether a focussing adverb modifies the subject or the clause.

None of the (very common) sentence adverbials that I have chosen for my grammar implementation can be found in the pre-subject position of a subordinate clause during search in the Oslo-Bergen corpora of tagged Norwegian texts. I therefore choose to view sentence-modifying adverbials placed in this position as an exception rather than as a rule, and I will not look further into the question of whether any type of clause-modifying adverbials can fill this position.

[4]For the well-informed reader: the small but important exception to this rule follows on page 12.

[5]Only correct for the reading where the book gave Gyrd to Inge.

(19) a. ?? [da] sikkert Gyrd har gitt boka til Inge.

b. [hvis] ikke Gyrd har gitt boka til Inge.

Under some circumstances it is possible for a clause to have the structure of a main clause and still function as a subclause. This includes all subclauses headed by the subjunction *at*, where it is optional whether the clause should have main or subordinate clause structure.

(20) a. [at] Gyrd sikkert har gitt boka til Inge.

b. [at] boka har Gyrd sikkert gitt til Inge.

2.3 Field Grammar

The traditional way of describing clause structure in Norwegian is based on the field-based approach to syntax originating with Diderichsens Field Grammar for Danish from 1946 (Diderichsen, 1962). The version presented here is taken from Norsk referansegrammatikk (Faarlund et al., 1997) and is slightly adjusted for Norwegian.

The two different clause structures described in the previous sections result in two different clause patterns, one applying to main clauses (see figure 1) and one to subordinate clauses (see figure 3). Each pattern is divided into three main fields, where the two last fields are divided into positions according roughly to the sort of elements that can occupy that position in a clause.

The first position, in the main clause schemata called the fundamental field or the front field, in the subclause schemata called the complementizer field[6], does only contain one element. In the main clause it will be the fronted element, in the subclause the subjunction. Following this field in both patterns comes first the nexus field and then the content field. The main difference between the patterns is located in the nexus field.

2.3.1 Main clauses

The nexus field of a main clause is divided into positions for verb, adverbials and a nominal, in the order of v-a-n-a as seen in figure 1. The V2 constraint is reflected by the claim that the v-position must always be filled by the finite verb in a main clause, in combination with the claim that the F-position must always be filled in a declarative main clause The nominal position is the obligatory subject position after the finite verb.. Positions might be empty, and we see that the n-position is empty if a subject is fronted. A-positions can be filled with an arbitrary number of adverbials[7].

[6] Translation of the field names from Norwegian following Platzack (1985) and Sells (2001)

[7] In cases of multiple adverbials in one position, or in cases where both a-positions are filled, the ordering of the adverbials is not free. See Nilsen (2000) for further information on this topic.

The content field includes positions for infinite verbs, objects and other adverbials, in this order. All can contain more than one element, and all can be empty, depending on the construction of the sentence.

Interrogative clauses also follow this pattern, as seen in figure 2. In yes/no questions the fundamental field has to be empty, while wh-questions require it to be filled by a wh-word.

Fund. field	Nexus field				Content field		
F	v	a1	n	a2	V	N	A
Inge	ga	-	-	-	-	boka til Gyrd	-
Boka	ga	sikkert	Inge	-	-	til Gyrd	-
Til Gyrd	ga	-	Inge	sikkert	-	boka	-
Boka	har	-	Inge	sikkert	gitt	til Gyrd	-
Heldigvis	kan	jo	Inge	sikkert	gi	boka til Gyrd	i dag

Figure 1: Diderichsen's main clause scheme, adjusted for Norwegian

Fund. field	Nexus field				Content field		
F	v	a1	n	a2	V	N	A
-	Ga	-	Inge	sikkert	-	boka til Gyrd?	
Hva	ga	sikkert	Inge		-	til Gyrd?	
Hvem	har	-	-	sikkert	gitt	boka til Gyrd?	

Figure 2: Interrogative clauses in Diderichsen's scheme

2.3.2 Subordinate clauses

The nexus field of a subordinate clause is also divided into positions for verb, adverbials and a nominal, but as seen in the pattern shown in figure 3 the ordering here is a-n-a-v. The n-position is the only possible subject position in this pattern, while the v-position is the only possible position for the finite verb, just as in the main clause pattern. The content field equals the content field of the main clause pattern.

2.4 Transformational generative theories

The name 'complementizer field', here used for the first field in the subordinate clause pattern in field grammar, originates in formal theories of grammar that view the mechanisms behind the ordering of constituents in a clause

Comp. field	Nexus field				Content field		
f	a1	n	a2	v	V	N	A
da	-	Inge	sikkert	ga	-	boka til Gyrd	-
at	-	Inge	sikkert	har	gitt	boka til Gyrd	-
dersom	ikke	Inge	-	har	gitt	boka til Gyrd	-

Figure 3: Diderichsen's subordinate clause scheme, adjusted for Norwegian

differently, though slightly related. The idea, that is especially adopted by GB theoreticians, is that the varying structures of a language are derivations of one basic structure, made by transformations on this structure. Conditions for transformations, imposed on the grammar by the grammar writer, are supposed to limit the number of possible derivations to the set of derivations that are considered grammatical for the language.

When it comes to the Scandinavian languages, subordinate clause structure is considered to be the basic structure, mainly due to its strict SVO word order and the fact that the verbal always is placed together as one unit. The inverted and non-inverted main clause structure is so supposed to be a derivation of the subordinate clause structure.

The complementizer in a subordinate clause occupies a position labelled Comp or C that is open both to complementizers and to exponents of tense, such as the finite verb. The inverted structure is derived from the basic structure by moving the finite verb to the C position. Any remaining constituent from the basic structure is then moved to the left of C to derive a main clause structure with a fronted subject, object or adverbial.

Platzack (1985) presents an overwiev of some of the transformational generative analyses of the V2 phenomenom in Swedish, Danish and Norwegian. In one of these analyses, the distribution of certain sentence adverbials in main and subordinate clauses are said to support the movement-to-C analysis. It claims that possible adverbial positions before and after the subject in a subordinate clause structure result in the corresponding adverbial positions before and after the inverted subject in a main clause structure.

As earlier mentioned, the adverbial position before the subject in a subordinate clause is questionable, and seldom used in an unambiguous manner[8]. Since one of the advantages to the movement analysis should be that the part of the structure that is not moved should remain constant, this might be an argument against this analysis. The adverbial position before the inverted subject in a main clause is used on a regular basis, occupied by adverbials that would leave a subordinate clause ungrammatical, should they occur

[8]The example given in Platzack (1985), *..., da pludselig en af mine Venner traade ind i Stuen* (*..., when suddenly one of my friends entered the room*), is admittedly a grammatical construction, also in Norwegian. One other example of a subordinate clause beginning with *da plutselig* was also found in the Oslo-Bergen Corpora of tagged Norwegian texts. Interestingly enough, the combination *da plutselig* was also found in one other construction, namely as one constituent fronting a declarative main clause, before the finite verb.

before the subject there.

In Hellan and Nordgård (2000), it is also argued against this type of analysis and the existence of the C position, based on the distribution of sentence adverbials as well as on object shift (for similar reasons as mentioned above) and the variation in clause patterns found in different Norwegian dialects. Their view on clause structure, "in favor of recognizing the patterns surveyed as somehow basic by themselves" (Hellan and Nordgård, 2000, pg. 144), is adopted for the work on this thesis. I will consequently try to implement the phenomena described in this section by the help of a non-transformational analysis, and so I have chosen one of the important non-transformational grammar frameworks to aid me, namely Head-driven Phrase Structure Grammar, which is presented in detail in the next section.

3 Head-Driven Phrase Structure Grammar

This section gives an introduction to Head-driven Phrase Structure Grammar (HPSG), the formal grammar framework that I have used to develop my analysis of Norwegian. The first main section presents the theory of typed feature structures and the construction and inner workings of the type hierarchy. In the last main section, the main charachteristics of the theory is described, includin the sign-based architecture, the general principles, the grammar rules and the lexical types For a further introduction to the framework, see Sag and Wasow (1999).

HPSG is one among many grammar frameworks contrasting with the transformational approach to generative grammar in that they are constraint-based and mostly lexicalist. 'Constraint-based' implies that the grammars are based on constraint satisfaction rather than transformational derivation, and 'lexicalist' (in this context) that words are the atomic entity of the syntax, their internal structure not playing any role in the construction of sentences.

The ancestor of HPSG, Generalized Phrase Structure Grammar[9] (GPSG), was based on the assumption that standard context-free phrase structure grammars could be enriched so as to make them suitable for describing natural language syntax[10]. Some of the central ideas in GPSG, like a feature-based analysis of filler-gap dependencies to treat long-distances dependencies, can be found almost unchanged also in its descendant, while new developments included among other things the attempts at incorporating data type theory and situation semantics into the theory. As HPSG evolved from GPSG, 'Head-Driven' was chosen to reflect the recognized importance of information encoded in the lexical heads of syntactic phrases.

3.1 Who and where

Attempts at computational implementation of HPSG was made early, as in a project supported by Hewlett-Packard Laboratories in Palo Alto, California, from 1980 to 1991. Contributions to its theoretical framework was made (among others) at the Center for the Study of Language and Information (CSLI) at Stanford, California, and the theoretical basis was developed mainly by Carl Pollard and Ivan Sag in Pollard and Sag (1987) and Pollard and Sag (1994), by Ivan Sag and Tom Wasow in Sag and Wasow (1999) and by Jonathan Ginzburg and Ivan Sag in Ginzburg and Sag (2001). Additional work can be found in major articles by various researchers.

Multiple inheritance hierarchies were used in early work on HPSG[11] to simplify the lexicon, expressing cross-classifying generalizations about words. The same approach was later applied by other linguists to treat generalizations about phrases and constructions in terms of cross-classifying type

[9] See Gazdar et al. (1985)

[10] This assumption has also been the basis idea for many other generative linguistic theories, such as Lexical Functional Grammar and Government and Binding.

[11] See Flickinger et al. (1985), Flickinger (1987), Pollard and Sag (1987) and more.

hierarchies. In Sag and Wasow (1999) the type system and type-based inheritance is a fully integrated part of the theory, in fact becoming one of its main characteristics. The possibilities of the type system is used extensively to provide analysis for interrogative constructions in English in Ginzburg and Sag (2001), in combination with the use of default constraints[12] introduced by Lascarides and Copestake (1999).

3.2 Means and mechanisms

3.2.1 Untyped feature structures

Feature Structures are sets of *feature-value pairs*, used to model linguistic objects and represent grammatical information. There is a finite set of possible features, which are all atomic symbols. The value of a feature is either an atomic symbol or a new feature structure. They can be thought of and represented as directed graphs which have one unique root node, and edges leaving this node labeled with feature names. The edges leaving the root node point to daughter nodes properly labeled (if the value is an atomic symbol), or with new edges departing from it (in the cases where the value is a new feature structure). A feature structure has to have a finite number of nodes, and it can not contain cycles. The features labeling the edges leaving a node must be unique within that node.

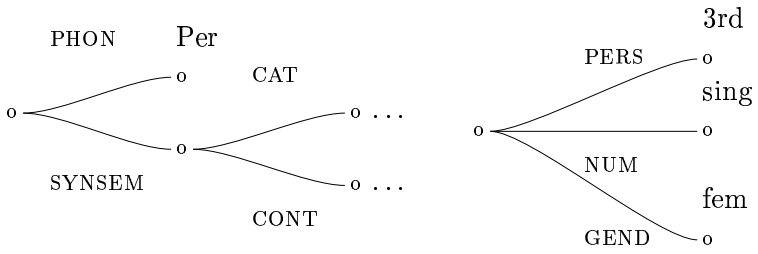

Figure 4: Directed acyclic graphs

Even if this representation is very useful for understanding some aspects of working with feature structures, like value sharing and unification, the structures are more conventionally represented as *attribute-value matrices* (AVMs) as seen in figure 5.

Feature structures can be *reentrant*, meaning that the structures contain features that share the same value. This is not the same case as when two or more edges in the graph lead to different nodes labeled with the same label, but the thought that the edges share exactly the same node in the graph, as illustrated in figure 6. In an AVM notation, reentrancy is indicated by boxed integers as seen in figure 7.

[12]Default constraints will not be used or further discussed in this thesis.

$$\begin{bmatrix} \text{PHON} & \text{'Per'} \\ \text{SYNSEM} & \begin{bmatrix} \text{CAT} & [\ ] \\ \text{CONT} & [\ ] \end{bmatrix} \end{bmatrix} \quad \begin{bmatrix} \text{PERS} & \text{3rd} \\ \text{NUM} & \text{sing} \\ \text{GEND} & \text{fem} \end{bmatrix}$$

Figure 5: AVMs

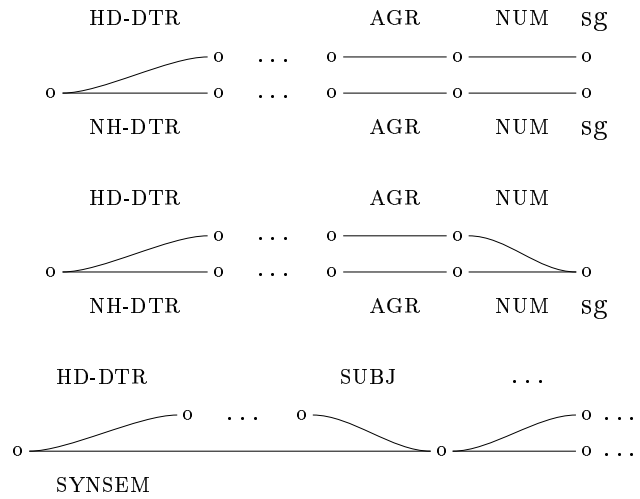

Figure 6: Re-entrant and non-reentrant directed graphs

$$\begin{bmatrix} \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \mid \text{HEAD} & \boxed{1} \\ \text{HD-DTR} \mid \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \mid \text{HEAD} & \boxed{1} \end{bmatrix}$$

Figure 7: Reentrancy in AVM notation

3.2.2 Untyped subsumption

Feature structures can be partially ordered by specificity, using the notion of *subsumption*. A more general feature structure F_1 can be said to subsume a less general structure F_2 ($F_1 \sqsubseteq F_2$). F_2 will then contain all the information present in F_1, and possibly also additional information, as seen in figure 8.

$$\begin{bmatrix} \text{GEND} & \text{fem} \end{bmatrix} \sqsubseteq \begin{bmatrix} \text{GEND} & \text{fem} \\ \text{PERS} & \text{3rd} \end{bmatrix}$$

$$\begin{bmatrix} \text{HD-DTR} \mid \text{SS} \mid \text{CAT} \mid \text{AGR} & \boxed{1} \\ \text{SS} \mid \text{CAT} \mid \text{AGR} & \boxed{1} \end{bmatrix} \sqsubseteq \begin{bmatrix} \text{HD-DTR} \mid \text{SS} \mid \text{CAT} \mid \text{AGR} & \boxed{1}\ \text{sg} \\ \text{SS} \mid \text{CAT} \mid \text{AGR} & \boxed{1} \end{bmatrix}$$

Figure 8: Subsumption

Explained using DAG notation, the root node in the graph representing F_2 must have at least the same number of edges leaving it as the root node in the graph representing F_1, labeled with the same feature names and pointing to nodes with the same labels as in F_1. If two edges lead to the same node in F_1, the corresponding edges in F_2 must do the same (reentrancy present in F_1 must also be included in F_2). Once this has been verified, these constraints in turn apply to the sub-graphs starting from each node hit by an edge from the root node, and the procedure is repeated until each sub-graph of F_1 has been tested against the equivalent sub-graph of F_2. F_2 can contain additional edges, and can also specify reentrancy not present in F_1.

If we find that all paths in F_1 are also present in F_2, that the value of the corresponding paths in F_1 and F_2 is the same for all paths in F_1, and that all paths that are reentrant in F_1 are also reentrant in F_2, then F_1 subsumes F_2. If not, the two feature structures are inconsistent (depicted by $\perp$).

3.2.3 Unification of untyped feature structures

Unification (depicted as $\sqcup$) is the operation of combining two feature structures into the most general feature structure which contains all the information from the two original structures. The unification of two feature structures F_1 and F_2 is then the most general feature structure F_3 which is subsumed by both F_1 and F_2. Unification is said to fail if there are one or more features in the two structures which have conflicting values (depicted by $\perp$).

3.2.4 Typed feature structures and the type hierarchy

> "Typed feature structure grammars are essentially based on one data structure - the typed feature structure, and one operation - unification. The type system contains the allowable structures

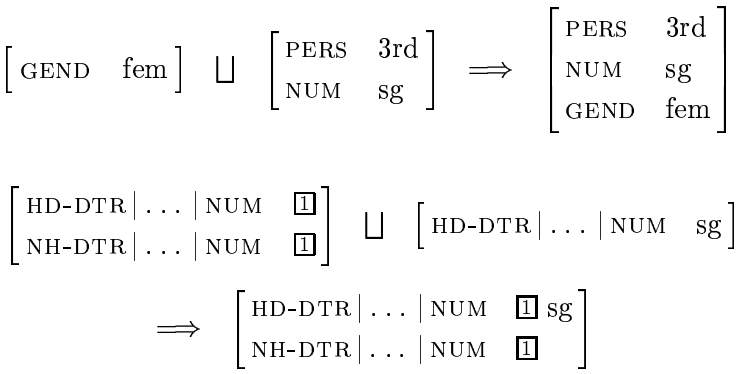

Figure 9: Unification of feature structures

> and provides a way of capturing linguistic generalizations. This combination is powerful enough to allow the grammar developer to write grammars and lexicon that can be used to parse and generate natural language." Copestake (2001)

Typing of feature structures has been developed to achieve two goals: as a method to constrain what can be the value of a feature, and to be able to capture generalizations across feature structures.

In a DAG representing a typed feature structure, every node in the graph must have a single type. Atomic values are replaced by (atomic) types. Every feature structure is labeled by a type, and each type has *appropriateness conditions* that define which features are appropriate for it and what constraints are laid on the feature values. The constraint on a type t is itself a feature structure of type t, as seen in figure 10.

Type	**Appropriate features**	**Constraint**
png	PERS NUM GEND	$_{png}\left[\begin{array}{ll}\text{PERS} & \textit{person}\\ \text{NUM} & \textit{number}\\ \text{GEND} & \textit{gender}\end{array}\right]$
person		$_{person}[\ ]$

Figure 10: Appropriateness conditions for types, with appropriate features and type constraints

The types are organized in a *type hierarchy* with a single most general top type, as sketched in figure 11. All types that exist must have a known position in the hierarchy (closed world assumption). The hierarchy can not contain cycles, but do allow for multiple inheritance. By multiple inheritance all the parents of the type must unify. The properties of a supertype are inherited by all of its subtypes, meaning that e.g. features found appropriate

for one type will also be appropriate for all the subtypes of this type, and that constraints on instances of a particular type will also be inherited to all its subtypes (monotonic inheritance). The constraint on a type must consequently be subsumed by the constraints on all its parents. Features can only be introduced at one place in the type hierarchy, and so feature structures with a common feature will all be descendants of a common supertype. If two types in the hierarchy are compatible, it is stipulated that they must and will have a unique most general common descendant (called *the greatest lower bound*).

Adding to the notion of subsumption according to this, a type x subsumes a type y either if they are the same type, or if y is a subtype of x. This holds for all *well-formed* instances of the the types x and y, as introduced in section 3.2.5.

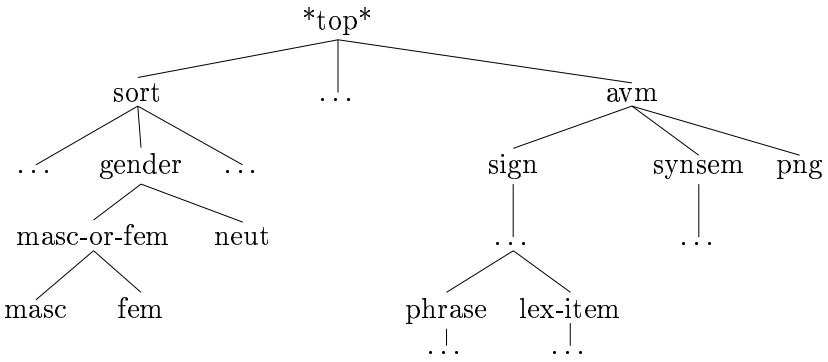

Figure 11: Excerpt from possible type hierarchy

3.2.5 Well-formedness and type inference

Not all typed feature structures (TFSs) described in this and following sections will be *well-formed* with regard to its set of type constraints. One of the very reasons for type constraints is to keep descriptions of the TFSs down to a manageable size by allowing generalizations. Non-well-formed structures are made into well-formed ones via *type inference*.

For a TFS to be well-formed, all substructures in the TFS must fulfill two criteria concerning appropriate features for a type and type constraint[13]: 1) The top-level features of every substructure must be the appropriate features for the type on the structure's root node, none missing and none added. 2) Every substructure must also satisfy the constraint on the type on its root node, meaning every substructure of the TFS must be subsumed by the constraint corresponding to the type on the substructure's root node.

Type inference returns the most general well-formed structure which a non-well-formed structure subsumes, if there is such a structure. To achieve this, the type on each node in the structure is first specialized to the most

[13]from Copestake, 2001, pg. 68.

general type for which all the node's features are appropriate, as seen in figure 12. Missing features are added to the node if they are appropriate features for the type on the node. Then it is made sure that all the resulting substructures are subsumed by their type constraint. For this purpose the constraint and the substructure is unified using well-formed unification. The result of a well-formed unification of two TFSs F_1 and F_2 is the most general well-formed TFS which is subsumed by both F_1 and F_2. The resulting TFS is certain to be subsumed by the type constraint, and it replaces the old substructure. All the information needed to build a well-formed structure is derived from the type system, and local constraints are expanded into full constraints following the principles of inheritance described in section 3.2.4. The type hierarchy must still be finite after type inference.

$$_{*top*}\left[\text{GEND}\quad\text{gender}\right] \Rightarrow _{png}\left[\text{GEND}\quad\text{gender}\right] \Rightarrow \begin{array}{l}_{png}\Big[\begin{array}{ll}\text{PERS} & \text{person}\\ \text{NUM} & \text{number}\\ \text{GEND} & \text{gender}\end{array}\Big]\end{array}$$

$$\begin{array}{l}_{png}\Big[\begin{array}{ll}\text{PERS} & \text{sort}\\ \text{NUM} & \text{number}\\ \text{GEND} & \text{gender}\end{array}\Big]\end{array} \Rightarrow \begin{array}{l}_{png}\Big[\begin{array}{ll}\text{PERS} & \text{person}\\ \text{NUM} & \text{number}\\ \text{GEND} & \text{gender}\end{array}\Big]\end{array}$$

Figure 12: Non-well-formed TFSs made into well-formed TFSs through type inference.

3.3 Main characteristics of HPSG

3.3.1 A sign-based architecture

The HPSG tradition uses a sign-based conception of grammar. The notion of 'sign' originates with Ferdinand de Saussure (1857-1913), a 'sign' in his view being an entity which associates form with meaning. Words and phrases can thus both be viewed as signs, insofar as they both combine form with meaning, and this might ideally allow us an equal treatment of the two variants of linguistic expressions. The type hierarchy of an HPSG grammar will typically contain the type *sign* (as in the sketched type hierarchy in figure 11), which will then be a supertype of both lexical and phrasal types, containing a feature like PHON (phonology) and a feature SYNSEM, which gives us both syntactic and semantic information.

Natural subtypes of *sign* could be *lex-item* and *phrase*, where *lex-item* can branch out into a hierarchy of lexical types discussed in the section 3.3.4. With the help of the type *phrase* and its subtypes, it is possible to no longer treat grammar rules as a special kind of theoretical entity, but to model them as feature structures using exactly the same description language, mech-

anisms and machinery as for words and lexemes, making the similarities between lexical and phrasal signs visible. With the help of features like DAUGHTERS or HEAD-DTR and NON-HEAD-DTR the subtypes of *phrase* can describe different kinds of linguistic phrase structure trees in terms of feature structures. Generalizations across different sorts of rules can be captured in the simple way that they hold for certain types of phrases and not for others, as for example the property headedness which is mentioned in the next section. Different types of phrases will be organized in the type hierarchy in the same way as sorts and lexical entities, as seen in figure 13.

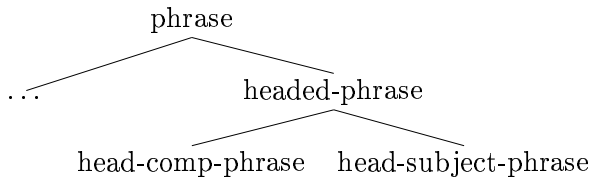

Figure 13: Excerpt of possible type hierarchy below *phrase*

3.3.2 General principles

Principles of the theory are formulated as general principles capable of covering all the needed cases without worrying about unimportant differences between the constituents involved from time to time.

As an example, one of the main principles, the Head Feature Principle, is a generalization over the fact that certain properties of the mother in a phrase are always identical to those of the most important daughter of the phrase, for this reason called the head daughter. In terms of the HPSG theory, this means that certain feature values of the mother are always identical to the values of the corresponding features of the head daughter, i.e. that the values are shared between mother and daughter. To reflect this, the signs are given a feature HEAD, which value is a feature structure containing all the (head) features that always have shared values in the two signs. The head feature principle can then be expressed simply as seen in figure 14.

$$\left[\text{HEAD } \boxed{1}\right] \rightarrow \ldots \text{ H}\left[\text{HEAD } \boxed{1}\right] \ldots$$

Figure 14: The Head Feature Principle

Thanks to the type hierarchy, it is possible to apply this general principle in a simple and elegant way to a grammar by implementing it as a type.

For the practical implementation in the type hierarchy, the head features are collected as appropriate features for a type *head*, with various subtypes for the different constraints laid on different lexical types, or for additional appropriate features. A TFS of type *sign* would then on some level have the

appropriate feature HEAD, which would be constrained to have a value of type *head*.

We build the general principle into our grammar by adding the type of *headed-phrase* to the type hierarchy, with the appropriate feature HD-DTR (head-daughter) for recognizing the daughter with which the HEAD value is to be shared, as seen in figure 15. By adding the constraint of shared HEAD values without any further specification of the phrase, we get a general type implementing a general principle from which a diversity of other phrase types can inherit. Grammar rules for which the head feature principle should apply will be subtypes of *headed-phrase*, inheriting its constraint in addition to applying own constraints.

$$\textit{headed-phrase}\left[\begin{array}{ll}\text{SYNSEM} & \left[\text{LOCAL}\,|\,\text{CAT}\,|\,\text{HEAD}\ \boxed{1}\right] \\ \text{HD-DTR} & \left[\text{SYNSEM}\,|\,\text{LOCAL}\,|\,\text{CAT}\,|\,\text{HEAD}\ \boxed{1}\right]\end{array}\right]$$

Figure 15: Constraints on TFSs of the type *headed-phrase*

Other principles are implemented in a similar way, including the valence principle, the semantic inheritance principle, the semantic compositionality principle, the GAP principle and others. See for instance Sag and Wasow (1999) for further details.

3.3.3 Grammar rules

The trouble of writing the grammar rules of an HPSG-based grammar is greatly reduced as a result of the linguistic generalizations made possible by working with phrases as feature structures on the same level as lexical items and incorporating them into the type hierarchy. The general principles and constraints can be introduced separately and on a higher level as the more construction-specific constraints imposed on realistic grammars of natural languages. Construction-specific types with construction-specific constraints inherit appropriate constraints from the types implementing the general principles needed in the construction at hand. The head-subject rule depicted as general rule in figure 16, would so in the type hierarchy inherit from the type *headed-phrase* in addition to imposing its own constraint shown in figure 17.

Parts of the Matrix, the basic type hierarchy presented in section 6, is exactly what is mentioned here: the general principles implemented as basic types, on which more language specific grammar rules can be build.

$$\left[\text{SUBJ}\ \langle\ \rangle\right] \ \rightarrow\ \boxed{1},\quad \text{H}\left[\text{SUBJ}\ \boxed{1}\right]$$

Figure 16: Head-Subject Phrase Structure Rule, head-final version

$$\textit{head-subj-phrase}\left[\begin{array}{ll}\text{SYNSEM} & \left[\text{LOCAL}\,|\,\text{CAT}\,|\,\text{VAL}\,|\,\text{SUBJ}\ \langle\ \rangle\right] \\ \text{HD-DTR} & \left[\text{SYNSEM}\,|\,\text{LOCAL}\,|\,\text{CAT}\,|\,\text{VAL}\,|\,\text{SUBJ}\ \langle\ \boxed{1}\ \rangle\right] \\ \text{NH-DTR} & \boxed{1}\end{array}\right]$$

Figure 17: Constraint on TFSs of the type *head-subject-phrase*, order-independent

3.3.4 Lexical types and the lexicon

HPSG is considered a theory based on strong lexicalism, in the sense that most of the syntactic information is located in the lexicon instead of in other components like the grammar rules (like e.g. standard CFG). Grammar rules and principles are tried kept simple and with a wide application throughout the grammar, with questions like e.g. subcategorization and possible modification all determined in the lexicon and the single lexical entry.

The lexicon is organized via a hierarchy of lexical types, each expressing lexical generalization that make building the lexicon much simpler and less time-consuming. It catches shared properties across different word classes and lexemes, the most general type encompassing all in *lex-item*. Multiple inheritance allows us to organize information in multiple dimensions, like part-of-speech and argument selection, each lexeme then inheriting from subtypes from both of these dimensions as seen in figure 18. Figure 19 illustrates some of the many types needed to build the type *iv-lxm* in figure 20 according to this.

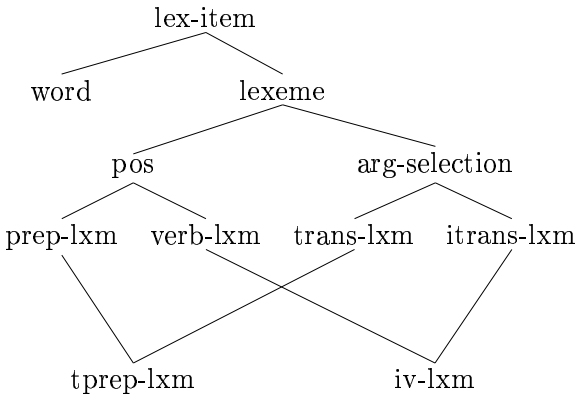

Figure 18: Excerpt of possible type hierarchy below *lex-item*

$$\textit{verb-lxm}\left[\text{LOCAL}\mid\text{CAT}\mid\underset{\textit{verb}}{\text{HEAD}}\left[\ \right]\right] \qquad \underset{\textit{verb}}{\left[\begin{array}{l}\text{VFORM}\ \textit{vform}\\ \text{PRED}\ \textit{bool}\\ \text{AUX}\ \textit{bool}\end{array}\right]}$$

$$\textit{itrans-lxm}\left[\text{LOCAL}\mid\text{CAT}\mid\underset{\textit{itrans}}{\text{VAL}}\left[\ \right]\right] \qquad \underset{\textit{itrans}}{\left[\text{COMPS}\ \langle\ \rangle\right]}$$

Figure 19: Constraints on a few types needed to build *iv-lxm*

$$\textit{iv-lxm}\left[\text{LOCAL}\left[\text{CAT}\left[\begin{array}{l}\text{HEAD}\ \underset{\textit{verb}}{\left[\begin{array}{l}\text{VFORM}\ \textit{vform}\\ \text{PRED}\ \textit{bool}\\ \text{AUX}\ \textit{bool}\end{array}\right]}\\ \text{VAL}\ \underset{\textit{itrans}}{\left[\begin{array}{l}\text{SUBJ}\ \langle\ [\ \ ]\ \rangle\\ \text{SPR}\ \langle\ \rangle\\ \text{COMPS}\ \langle\ \rangle\end{array}\right]}\end{array}\right]\right]\right]$$

Figure 20: Simplified example of *iv-lxm*, based on the two dimensions part-of-speech (*verb*) and argument selection (*itrans*)

4 Minimal Recursion Semantics

This section gives an introduction to Minimal Recursion Semantics (MRS), the semantic representation incorporated in the Matrix and also used in the LinGO ERG. The introduction is brief, as the semantic representations produced by the implemented grammar fragment are not the main subject for this thesis. Still, semantics are an integrated part of the implementation and in a few cases help to constrain syntactic analyses, hence a basic knowledge of the theory behind MRS and the mechanisms at work is necessary for a better understanding of the grammar. Details on MRS can be found in Copestake et al. (1999).

As its origins in the Verbmobil project would suggest, Minimal Recursion Semantics is a framework for semantic theory developed to suit the needs of large computational grammars, and designed to be used for parsing, generation and semantic transfer, all in combination with linguistically precise, hand-built grammars. MRS is derived from existing theories of semantics, but it is constructed as a meta language for describing semantic structures, not as an independent semantic theory in itself. As such, it is depending on an underlying object language, like predicate calculus with generalized quantifiers, which is the object language assumed in Copestake et al. (1999).

The framework is surface-oriented and, as the name specifies, aims at non-recursive structures in the representations. Its goal is to be able to decompose, relate and compare semantic structures in an easy way, plus to allow for underspecifiacation in the representation without losing neither expressive accuracy nor compuational efficiency and control.

The possibility of underspecification of the representations is important, for instance when it comes to quantifier scope. Resolving quantifier scope ambiguities is a difficult task, and additionally often quite unnecessary for some classes of NLP tasks, as for example machine translation. MRS makes it possible to ignore scope when it is not needed, but also to retrieve it when necessary.

4.1 Who and where

MRS was used for the semantic representation in the English HPSG grammar in the Verbmobil project. It has been presented in several papers, including Copestake et al. (1995), Copestake (1995), Copestake et al. (1999) and lately in Copestake et al. (2001)

4.2 The representation

The basic units in MRS are *elementary predications* (EPs), a single relation with its arguments.

EPs are never embedded into each other, and to be able to flatten the semantic structures even further without losing control, each EP is supplied

with a handle that serves as the label for the EP[14]. A scopal EP will have labels in its scopal argument slots, and all EPs filling the same scopal slot will have the same label. Hence, EPs sharing the same label are assumed to be conjoined. The conjunction symbol is omitted to flatten the structure and so avoid that logical properties of the conjunction create formal differences between otherwise equivalent structures.

The conventional representation of sentence (21a) in predicate calculus is shown in (21b). In (21c), the logical form is replaced by a flat list of EPs, i.e. the conjunction is removed, and each EP is given a label (*h0* to *hn*). The conjoined EPs all share the same label, and the two argument slots in the EP for the quantifier *alle* are filled with the labels of the EPs that occupied this slot in (21b).

(21) a. Alle store gule løvetenner blomstrer.
all big yellow dandelions bloom

b. alle(x,(stor(x) & (gul(x) & (løvetann(x)))), blomstre(x))

c. *h0*:alle(x, *h1*, *h2*), *h1*:stor(x), *h1*:gul(x), *h1*:løvetann(x),
h2:blomstre(x)

What distinguishes the MRS representation from a plain rewriting of the logical form, is the fact that handles in the argument positions of an EP can be left underspecified. The example sentence in (21a) does not lend itself to illustrating the usefulness of underspecification, as the only possible scope-resolved representation of the sentence is shown in (21b). Sentence (22a), on the other hand, allows for the two differing representations shown in (22b) and (22c), according to which quantifier is given wide scope.

(22) a. Alle gutter beundrer ei blid jente
all boys admire some smiling girl

b. alle(x, gutt(x), ei(y, (blid(y) & jente(y)), beundre(x,y))

c. ei(y, (blid(y) & jente(y)), alle(x,gutt(x), beundre(x,y))

By letting the handle variables *hA* and *hB* replace specified labels in the scopal argument slot of the quantifier EPs, the representation shown in (23) is a generalization that covers (22b) as well as (22c) . The conditions on retrieving the fully spedified representation from an underspecified representation are that 1) no argument can be left unsatisfied and 2) an EP can only fill one argument position.

(23) *h0*:alle(x, *h1*, *hA*), *h1*:gutt(x), *h2*:blid(x), *h2*:jente(y), *h4*:ei(y, *h2*, *hB*)
h4:beundre(x,y)

[14]The word label is used for the handles labeling EPs, whereas the word handle is used for variable handle arguments in argument positions of an EP, as presented later in the section.

In (23), the restriction of each quantifier is directly filled by a label, because this is the only possible representation for the sentence. This can not be made a general rule, though, because more complex cases demand further underspecification to make the underspecified representation cover all possible representations of a sentence. Still, complete and unconstrained underspecification would lead to unwanted representations, as shown in (24a) and (24b), based on (23).

(24) a. *h0*:alle(x, *hA*, *hB*), *h1*:stor(x), *h1*:gul(x), *h1*:løvetann(x), *h2*:blomstre(x)

b. *h0*:alle(x, *h2*, *h1*), *h1*:stor(x), *h1*:gul(x), *h1*:løvetann(x), *h2*:blomstre(x)

To avoid this, it is possible to express constraints on handle variable instantiation. The form of handle constraint used in Copestake et al. (1999) is called a 'qeq' constraint ($=_q$, for equivalence modulo quantification), a specific form of the outscopes relationship. An EP *E* immediately outscopes another EP *E'* if the value of one of the handle-taking arguments of *E* is the label of *E'*. If a handle argument *hA* is qeq a label *h1*, either that argument slot must be filled directly by *h1*, or it must be filled by the label of another quantifier that has *h1* directly or again indirectly in its body argument.

A MRS structure thus contains a bag of handle constraints in addition to the bag of EPs. A full MRS structure also contains a global and a local top handle. The global top handle is a handle that is not outscoped by any other handle in the structure, whereas the local top handle is the topmost label in an MRS that is not the label of a floating EP, i.e. an EP that can be inserted into a $=_q$ relationship, like quantifiers[15]. The full MRS structure for an underspecified representation of (23) with qeq constraints is shown in (25).

(25) $<$ *h0*, *h1*,
{ *h2*:alle(x, *hA*, *hB*), *h3*:gutt(x), *h4*:blid(x), *h4*:jente(x),
h5:ei(y, *hC*, *hD*) *h6*:beundre(x,y) },
{ *hA* $=_q$ *h3*, *hC* $=_q$ *h4*, *h1* $=_q$ *h6* } $>$

An MRS is said to be scope-resolved if every label is equated with either a handle argument or the global top of the MRS, and at least one label is equated with the global top. The result of these constraints is that the scope resolved MRS has the structure of a tree, with one single root node and no nodes that have multiple parents. An underspecified MRS corresponds to a set of expressions in the object language, whereas a scope-resolved MRS corresponds to exactly one expression in the object language. An MRS is well-formed if it represents at least one expression in the object language.

[15] In the implementation of the LinGO ERG, though, the global top is redundant for reasons explained in Copestake et al. (1999), and so only the local top is implemented. This applies to my own grammar fragment based on the Matrix as well, so when only one top is used throughout this thesis, this is always the local top.

4.3 Semantic composition

The composition rules for MRSs (in phrase structure grammars) are quite simple. Most lexical items supply a single EP to the representation. When a phrase is constructed, the bags of EPs from the daughters are appended to one bag, and so are the bags of handle constraints.

If none of the EPs are scopal (i.e. the phrase is intersective), the handles of the daughters are equated with each other and with the top handle of the phrase. The MRS representation of the phrase *blide jenter* shown in (26a), hence, is the result of the composition of the two MRS representations shown in (26b).

If one of the EPs is scopal, the handle argument of the scopal EP is defined to be qeq the top handle of the scoped over phrase. Thus the MRS representation of the phrase *alle gutter* shown in (27a) is a result of the composition of the two MRS representations shown in (27b).

(26) a. $< h0, \{h0{:}\text{blid}(x),\ h0{:}\text{jente}(x)\}, \{\} >$

b. $< h0, \{h0{:}\text{blid}(x)\}, \{\}>$
$< h2, \{h2{:}\text{jente}(x)\}, \{\} >$

(27) a. $< h0, \{h1{:}\text{alle}(x,\ hA,\ hB),\ h2{:}\text{gutt}(x)\}, \{hA =_q h2\}>$

b. $< h0, \{h1{:}\text{alle}(x,\ hA,\ hB)\}, \{\}>$
$< h2, \{h2{:}\text{gutt}(x)\}, \{\} >$

The conindexation of the ordinary variables has here been taken for granted, but this conindexation must also be specificly defined in lexical entries or constructions of the grmmar, in order to yield correct representations.

4.4 MRS in typed feature structures

MRS is designed with feature-based grammars in mind, and can so be easily expressed in terms of typed feature structures and unification. As mentioned, it is already integrated in the LinGO ERG, proving to work well with this kind of large typed feature structure grammars.

The EPs can be implemented as feature structures, organized in a hierarchy of relational types, where individual predicates can either be encoded as specific types of relations, or as the value of a feature, such as PRED or RELN. The appropriate features for the types include a feature for the handle that labels the EP and one feature each for the EP's argument positions. The different types of relations can be organized in the type hierarchy according to the type and number of arguments they take. The two types of relations shown in figure 21 can be the types for the representation found in (27b) for *gutt* and *alle*, respectively.

An MRS structure is defined as a type whose appropriate features are at least TOP, LIZST or RELS for the bag of relations and H-CONS for the bag of

$$\textit{nom-rel}\begin{bmatrix}\text{HNDL} & \textit{handle}\\ \text{INST} & \textit{ref-ind}\\ \text{PRED} & \text{'gutt}\end{bmatrix} \qquad \textit{quant-rel}\begin{bmatrix}\text{HNDL} & \textit{handle}\\ \text{BV} & \textit{ref-ind}\\ \text{RESTR} & \textit{handle}\\ \text{BODY} & \textit{handle}\\ \text{PRED} & \text{'alle}\end{bmatrix}$$

Figure 21: EPs as typed feature structures

handle constraints, as shown in figure 22. The bags are implemented as lists, but the order of list elements is not semantically relevant. A qeq constraint is implemented as a type with appropriate features SC-ARG and OUTSCPD. Coindexation of feature structures is used for the linking of variables.

$$\textit{mrs}\begin{bmatrix}\text{TOP} & \boxed{1}\\ \text{LIZST} & \left\langle \textit{event-rel}\begin{bmatrix}\text{HNDL} & \boxed{6}\\ \text{ARG1} & \boxed{3}\\ \text{PRED} & \text{'smile}\end{bmatrix}, \textit{nom-rel}\begin{bmatrix}\text{HNDL} & \boxed{5}\\ \text{INST} & \boxed{3}\\ \text{PRED} & \text{'gutt}\end{bmatrix}, \textit{quant-rel}\begin{bmatrix}\text{HNDL} & \boxed{2}\\ \text{BV} & \boxed{3}\\ \text{RESTR} & \boxed{4}\\ \text{BODY} & \textit{handle}\\ \text{PRED} & \text{'en}\end{bmatrix}\right\rangle\\ \text{H-CONS} & \left\langle \textit{qeq}\begin{bmatrix}\text{SC-ARG} & \boxed{1}\\ \text{OUTSCPD} & \boxed{6}\end{bmatrix}, \textit{qeq}\begin{bmatrix}\text{SC-ARG} & \boxed{4}\\ \text{OUTSCPD} & \boxed{5}\end{bmatrix}\right\rangle\end{bmatrix}$$

Figure 22: MRS structure

Phrase types can be defined with regard to constraints on semantic composition, for instance whether a phrase type handles intersective or scopal elements. Semantic contributions to the phrase made by the construction are simply treated as an additional, implicit daughter in the structure, introduced via the feature C-CONT. In an implementation, the appending of the two bags of EPs and handle constraints can be achieved by the help of difference lists.

Further details on the relational types and the linking of syntactic and semantic information made by constructional phrase types in my implemented grammar fragment can be found in section 6, describing the Matrix type hierarchy, and in sections 7 and 8, of course.

5 The LKB system

This section provides a rough introduction to the Linguistic Knowledge Building system (LKB), the grammar development environment that I have employed to implement a fragment of an HPSG grammar for Norwegian. The first subsection presents the practical implementation of a grammar, including notational issues and a survey of grammar components, whereas the second subsection deals with the processing of a grammar when made, including parsing, generation, efficiency issues and how to test the grammar. If detailed information on the system, its various parts and its use is needed, see first Copestake (2001) and secondly further references given in the text.

The LKB system is an open-source grammar development environment for typed feature structure grammars, to be used with constraint-based linguistic formalisms. It contains tools to help the grammar writer construct the various parts of the grammar, and it includes support for parsing and generation with the grammars when built. The system can be downloaded from the LKB-website.[16] The system is used for teaching, research and application development on a wide variety of sites.

5.1 Who and where

The main developers of the LKB system are Ann Copestake, John Carrol, Rob Malouf and Stephan Oepen. It was initially developed at the University of Cambridge Computer Laboratory as part of the ACQUILEX projects (1991 $\rightarrow$), at first as a tool for construction of typed feature structure lexicons. LKB then stood for Lexical Knowledge Base. The system has since been heavily updated, mainly at the Center for the Study of Language and Information (CSLI) at Stanford University, California (as a part of the LinGO project), and it now serves for building and maintaining all grammar components, including syntactic and semantic knowledge. A theoretical and practical introduction to the system and its use with typed feature structure grammars is given in Copestake (2001), which also functions as a user manual.

5.2 Implementing a grammar

5.2.1 A basic grammar

The parts of a typed feature structure grammar built in LKB are divided into different files recognized and used by the system. The minimal set of grammar components and their default file names are as follows:

- `types.tdl` the type hierarchy, type definitions with information about inheritance, appropriate features and type constraints for each type

[16] `http://cslipublications.stanford.edu/lkb.html`

- `lexicon.tdl` instances of the lexical types with orthography and optionally specific semantic predicates, lexical items used in parsing and generation
- `rules.tdl` instances of the construction types, grammar rules used by the parser and generator
- `lrules.tdl` lexical rules, redundancy rules implementing productive lexical processes
- `irules.tdl` lexical rules with associated ortographemic variation, typically inflection
- `roots.tdl` the start symbol(s) of the grammar, well-formedness conditions on full results

In addition, grammar configurations for the LKB for each grammar are defined in a small number of auxiliary lisp files that can be edited by the grammar writer to adjust e.g. path settings, interface functions and more.

5.2.2 Notation

The abbreviation TDL stands for Type Description Language, the description language used for constructing all the parts of an LKB grammar.

The definition of a type consists of the name of the type being defined, the operator ':=', one or more parent types from which the type inherits, each separated by '&', and finally an optional list of appropriate features and specific constraints for the new type. An excerpt from a possible type file describing parts of the type hierarchy on page 22 is provided in figure 23.

LKB notation:	AVM strucure:
`avm =: top.`	$\underset{avm}{[\ ]}$

`sign := avm &`
`[ STEM list,`
`SYNSEM synsem ].`
	$\underset{sign}{\begin{bmatrix}\text{STEM} & \textit{list} \\ \text{SYNSEM} & \textit{synsem}\end{bmatrix}}$

`rule := sign &` `[ RULE-NAME string ].`	$\underset{rule}{\begin{bmatrix}\text{STEM} & \textit{list} \\ \text{SYNSEM} & \textit{synsem} \\ \text{RULE-NAME} & \textit{string}\end{bmatrix}}$

Figure 23: LKB notation vs. AVM notation

Reentrancy is introduced using the sign '#' followed by an identifier that names the reentrancy uniquely within one definition. Nested TFSs can each

```
headed-phrase := phrase &
 [ SYNSEM.LOCAL.CAT.HEAD #head,
   HD-DTR.SYNSEM.LOCAL.CAT.HEAD #head ].
```

headed-phrase

$$\begin{bmatrix} \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \mid \text{HEAD} & \boxed{1} \\ \text{HD-DTR} \mid \text{SYNSEM} \mid \text{LOCAL} \mid \text{CAT} \mid \text{HEAD} & \boxed{1} \end{bmatrix}$$

Figure 24: Paths and reentrancy in TDL notation vs. AVM notation

be enclosed in its own set of square brackets (`[ ]`), or the path-notation seen in figure 24 can be used.

String constants are always subtypes of the type *string*, and do not need to be defined in the type hierarchy. They can not themselves have subtypes, and are so mutually incompatible. Strings are enclosed by double quotes, as in `"jente"`.

Lists are not a built-in data type, but a special list notation is available in TDL that is expanded into a standard TFS representation by the LKB reader. Notation is present for fully defined lists, underspecified lists of unknown length and for difference lists that can be concatenated by unification. Figure 25 shows the abbreviatory TDL list notation versus the full AVM notation.

The type *list* used for the standard list representation has two daughter types, *ne-list* (*non-empty-list*) with the appropriate features FIRST and REST and *null*. This specification of a non-empty list is needed to provide indirect recursion in the type hierarchy without making it infinite, as it would be if *list* was to be defined recursively with the appropriate features FIRST and REST, and with the type *list* itself as value for the feature REST. By specifying different types of lists, the feature REST in *ne-list* can unproblematically be defined to have a value of type *list*, as type inference will not try to specify the type further without additional constraints deciding on which daughter type is to be selected.

5.2.3 Grammar components

Type hierarchies used with the LKB must have all the properties described in section 3.2.4. Type inference takes place as described in section 3.2.5 to convert the TDL specifications into well-formed TFSs.

Apart from non-linguistic types like lists and strings, the type hierarchy includes lexical types and types for lexical and syntactical rules as described in section 3.3. The other main parts of the grammar are defined from these types, in the same description language as the types themselves. The lexicon is constructed from the lexical types, each entry pairing strings with their syntactic and semantic form. Lexical and syntactic rules are defined from lexical and constructional types. Figure 26 and 27 show a lexical entry and a grammar rule written in TDL, respectively.

LKB notation	AVM strucure:
`< hei, du >`	$\begin{bmatrix} \textit{ne-list} \\ \text{FIRST} \quad \text{hei} \\ \text{REST} \quad \begin{bmatrix} \textit{ne-list} \\ \text{FIRST} \quad \text{du} \\ \text{REST} \quad \textit{null} \end{bmatrix} \end{bmatrix}$
`< hei, du, ... >`	$\begin{bmatrix} \textit{ne-list} \\ \text{FIRST} \quad \text{hei} \\ \text{REST} \quad \begin{bmatrix} \textit{ne-list} \\ \text{FIRST} \quad \text{du} \\ \text{REST} \quad \textit{list} \end{bmatrix} \end{bmatrix}$
`< hei . du >`	$\begin{bmatrix} \textit{ne-list} \\ \text{FIRST} \quad \text{hei} \\ \text{REST} \quad \text{du} \end{bmatrix}$
`<! hei, du !>`	$\begin{bmatrix} \textit{diff-list} \\ \text{LIST} \quad \begin{bmatrix} \textit{ne-list} \\ \text{FIRST} \quad \text{hei} \\ \text{REST} \quad \begin{bmatrix} \textit{ne-list} \\ \text{FIRST} \quad \text{du} \\ \text{REST} \quad \boxed{1} \end{bmatrix} \end{bmatrix} \\ \text{LAST} \quad \boxed{1} \end{bmatrix}$

Figure 25: Lists and difference lists in LKB notation vs. AVM notation

```
jente-noun :=  fem-noun-lxm &
 [ STEM < "jente" >,
   SYNSEM.LOCAL.CONT.RELS.LIST.FIRST.PRED 'jente].
```

Figure 26: Lexical entry in TDL

```
head-complement-rule := head-comp-phrase.
```

Figure 27: Grammar rule in TDL

A simple ortographemic component in the LKB admits a certain use of inflectional lexical rules, unary rules mapping from lexemes to words or lexemes. They differ from non-inflectional lexical rules only in that they add orthographemic information describing the affixation. Affixation patterns are written as seen in figure 28, essentially using pairs of regular expressions, where the first expression must match letters in the word stem, while the second expression supplies the corresponding letters in the affixed version of the word. '*' represents a null-character, while characters following an '!' are macro symbols denoting a set of letters. The key words '`suffix`' and '`affix`' in the line of subrules determine what sort of affixation the rule is treating.

Irregular morphemes are handled separately, viz. in a separate file where each entry provides a triple of (i) the inflected form, (ii) a rule specification and (iii) the stem of the irregular morpheme.

```
%(letter-set (!t bcdfghjklmnpqrsxtvwxz))
%(letter-set (!v aiouyæøå))

noun-fem_irule :=
%suffix (e a) (!t !ta) (!v !va)
sg-noun-word-aff &
 [ ARGS < fem-noun-lxm >,
   SYNSEM.LOCAL.AGR.PNG.DEF def ].
```

Figure 28: Morphological rule in TDL

It is possible to define labels for certain TFS configurations in a separate file, deciding for instance how to label the nodes in a parse tree. Though abbreviatory labels like 'S' and 'VP' can be used, these linguistic categories are not elements of the theory in unification grammar. The labels are merely defined by the grammar writer as abbreviations for certain typed feature structures.

The start symbol or symbols of the grammar are defined in a similar way, not as a type, but as a set of TFSs in a separate file, as seen in figure 29.

```
root := phrase &
 [ SYNSEM.LOCAL.CAT [ HEAD verb & [ VFORM fin ],
                      VAL [ SUBJ < >,
                            COMPS < > ]].
```

Figure 29: Possible root TFS

5.3 Processing a grammar

5.3.1 MRS

The LKB system contains a separate module for reading out the semantics from a TFS, converting it into an MRS object and processing it. The module is able to scope-resolve MRS structures (see section 4), to compare different structures, to export them to other formats or simply to reduced them to elementary dependency structures. The basic MRS structure for the sentence *Jenta smiler*[17] (*The girl smiles*) is shown in figure 30, and an indexed version is seen in figure 31.

5.3.2 Parsing and generation

The default parser used in the LKB system implements a variant of active chart parsing, as described in Oepen and Callmeier (2000). The parsing algorithm is purely bottom up, bidirectional and key-driven, i.e. the daughter marked as the key daughter by the grammar writer is checked first when a grammar rule is applied to two or more daughters. Chart parsing in general is described in detail in Jurafsky and Martin (2000).

The chart generator used in the LKB is described in Carrol et al. (1999). It assumes a lexicalist grammar, but it does allow for constructions that introduce additional semantics. It also combines the chart generation with a special treatment of intersective modification for increased generator efficiency. The generator requires a grammar that uses a flat, non-nested semantic representation, where the semantics of a phrase is described as a list of relations with coindexed variables. This applies to MRS and similar formalisms.

5.3.3 Processing efficiency

When implementing larger grammars, past the level of toy grammars, even for interactive grammar development processing efficiency can become a pressing issue. The LKB system supplies several mechanisms for keeping processing costs down, but is at the same time somewhat tuned to the LinGO ERG grammar in terms of where most work on efficiency issues has been invested. For instance, the ERG includes only about 70 rules, but a type hierarchy with thousands of types, thus naturally deciding which look-up mechanism in the LKB is the more advanced. Because of this, the choices of design made by the grammar writer can still have a great influence on the efficiency of processing a grammar.

Efficiency techniques implemented in the LKB includes (i) the checking of high-frequency failure paths before full unification is attempted, (ii) key-driven parsing and generation, (iii) using the locality principle[18] in HPSG to

[17] The example is taken from my Norwegian grammar fragment.

[18] The HPSG locality principle guarantees that it is not possible for a phrase to access information from daughters of its own daughters or from any level below them unless it

```
[
  INDEX: e1 [ EVENT
             E.TENSE:  PRESENT
             E.MOOD:  INDICATIVE
             E.ASPECT:  ACTIVE ]
  LISZT: <
         [ jente
           HNDL: h3
           INST: x2 [ REF-IND
                      PNG.DEF:  DEF
                      PNG.PERS:  PERS
                      PNG.NUM:  SING
                      PNG.GEND:  FEM ]
           LABEL: v4 ]
         [ def-rel
           BV: x2
           HNDL: h7
           LABEL: v8
           RESTR: h5
           SCOPE: h6 ]
         [ smile-rel
           ARG1: x2
           EVENT: e1
           HNDL: h9
           LABEL: v10 ]         >
  HCONS: <           h5 QEQ h3         > ]
```

Figure 30: MRS output for *Jenta smiler*

```
<e1:PRESENT:INDICATIVE:ACTIVE,
{jente(h3, x2:DEF:PERS:SING:FEM, v4),
def-rel(x2, h7, v8, h5, h6),
smile-rel(x2, e1, h9, v10)},
{h5 QEQ h3}>
```

Figure 31: Indexed MRS output for *Jenta smiler*

avoid copying and tree reconstructions (see Malouf et al., 2000), and (iv) optional packing with special treatment of the problems that arise trying to use this technique with unification-based grammars (see Oepen and Carroll, 2000).

5.3.4 Testing the grammar

The LKB system includes a simple batch parse mechanism which can be used with simple grammars, but non-trivial grammar engineering tasks has been interfaced to a package called [incr tsdb()].[19]

The package includes components to take care of the needs of both grammar writers and system developers working with large scale grammars. The test data are stored in a database together with the needed information about each test item, where the need for information is decided upon by the grammar writer/system developer. During parsing and generation, different performance measures are taken, stored and made into a profile for the grammar, for later inspection. Possibly interesting measures might include the number of readings per test item, time and memory usage, ambiguity and so forth, all depending on what part of the grammar or system one wants to inspect. The package also includes graphical tools for inspecting and comparing the profiles after changing the grammar.

is explicitly carried up through its daughters.

[19] Web adress: `http://www.coli.uni-sb.de/itsdb/`

6 The LinGO Grammar Matrix

In this section I present the LinGO Grammar Matrix, the basis grammar upon which I have built my own grammar fragment for Norwegian. Its main partitions of types are presented section-wise, with special emphasis on those aspects that will be revisited in section 7. The sections introduce the basic feature geometry, types for semantic composition, general classes of rules and constructional types. For a broader introduction to the Matrix initiative, see Bender et al. (2002).

The Matrix is a grammar 'starter kit' for HPSG grammars, freely distributed to interested grammar writers . It is a subset of types from two existing large-scale HPSG grammars for English and Japanese, leaning heavily on the English LinGO ERG grammar implemented mainly by Dan Flickinger (see Copestake and Flickinger, 2000), and aiming to ensure consistency with other work on HPSG and semantic representation.

The goal of the Matrix is to supply grammar writers using the LKB system with a basic language-independent type hierarchy, as a starting point from which more sophisticated language-specific grammars can be built. Using this starter kit, grammar writers working with different languages can quickly approach a level where linguistically interesting grammars can be built, avoiding time-consuming sessions with toy grammars.

Depending on feedback from these users, the Matrix can also function as an experiment to see to which extent the selected types can be fruitfully shared across grammars for different languages, and so with increased plausability be considered language-independent. Additionally, comparision of different grammars might be simpliefied when they are built on a common base.

The Matrix embodies the already mentioned goal of making the type hierarchy express generalizations across different linguistic objects. It contains a total of 210 types, whose implemention faithfully follows main-stream HPSG theory. On account of this, this section will not go into minute details describing all types and features included in the Matrix, but will instead try to give an overview of its main partitions of types with some instructive examples.

6.1 Who and where

The Matrix 0.1 was released in January 2002, and updates were made in June 2002. It is being developed at CSLI Stanford, primarily by Emily Bender (see Bender et al., 2002). The download-able Matrix package[20] includes configuration and parameter files for the LKB.

[20]Recent version found at: `http://lingo.stanford.edu/ftp/matrix.tgz`

6.2 Basic feature geometry

The types for basic feature geometry implemented in the Matrix are shown in an abbreviated version[21] in figure 32. The main partitions of types include:

- atomic types vs. attribute value matrices
- types for lists, difference lists and list operations (see also figure 25)
- generalized types for various levels of linguistic description in HPSG

Appropriate features and types are only defined in the Matrix if it is suspected that they could be language-independent. The basic linguistic object, the sign, is implemented as shown in figure 33. Elementary features like HEAD, VAL(ENCE) and CAT(EGORY) will be defined as appropriate features for suitable types, as they are part of the theoretical basis of HPSG and vital to the grammar implementation regardless of language. Types that are constrained to be the value of these features, i.e. *head*, *val* and *cat*, will also need to be defined, though maybe in a very underspecified version. This because crucial appropriate features (or types) in some languages will be lacking in others, like e.g. the head-features CASE or AUX(ILIARY) for languages without case or a system of auxiliary verbs. It is then up to a grammar writer to fill in the additional features required for the language with which she is working.

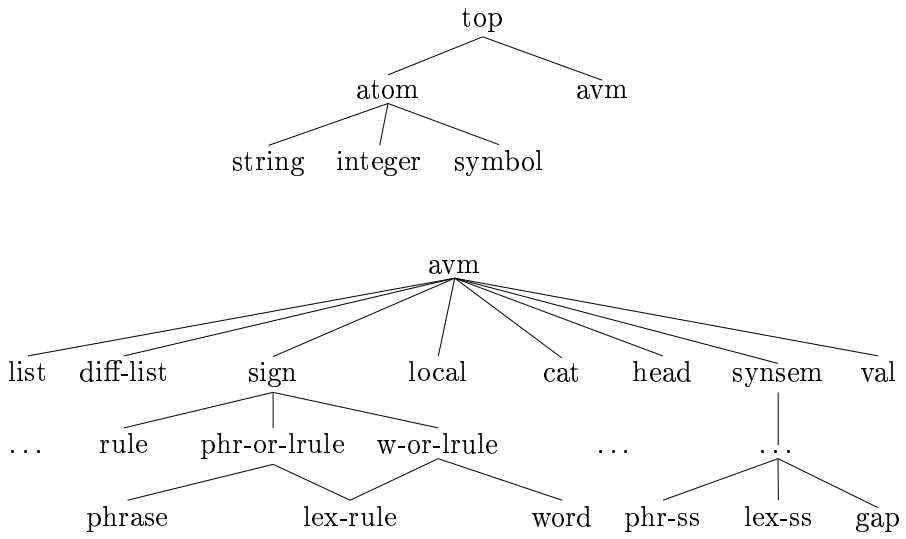

Figure 32: Basic feature geometry, excerpt from the Matrix

[21] The type hierarchies depicted in this section are abbreviated sub-hierarchies of the real Matrix type hierarchy and meant to give only an idea about the contents, not a full overview. Important types are left out, and intermediate steps (e.g. used for processing efficiency) have been skipped.

$$\textit{sign}\begin{bmatrix} \text{STEM} & \textit{list} \\ \text{SYNSEM} & \textit{synsem} \\ \text{ARGS} & \textit{list} \\ \text{INFLECTED} & \textit{bool} \\ \text{ROOT} & \textit{bool} \end{bmatrix}$$

Figure 33: The type *sign*

$$\textit{mrs}\begin{bmatrix} \text{TOP} & \textit{handle} \\ \text{INDEX} & \textit{indiviual} \\ \text{LISZT} & \textit{diff-list} \\ \text{H-CONS} & \textit{diff-list} \end{bmatrix}$$

Figure 34: The type *mrs*

6.3 Types for semantic composition

The types for semantic composition are based on Minimal Recursion Semantics (see section 4). Following this, the type *mrs* needs to contain a top handle, a bag of elementary predications and a bag of scope constraints.The HPSG implementation also introduce a feature INDEX into the MRS, which is an instance or event variable. The type *mrs* is defined with the appropriate features and constraints, as shown in figure 34. It has two subtypes, *psoa* and *nom-obj*, where the value of the feature INDEX is constrained to be of type *event* in a *psoa* and of type *index* in a *nom-obj*, respectively.

The bag of elementary predications is implemented as the value of the feature LISZT, a difference list that collects semantic relations of type *relation*, each a feature structure with its own handle, predicate and argument information. The bag of scope constraints is the value of the feature H-CONS, again implemented as a difference list that houses elements of the type *qeq* (although other types of handle constraints are in principle allowed).

6.4 General classes of rules

The general classes of rules implement fundamental principles in HPSG, like the head-feature principle, the semantic compositionality principle and the non-local feature principle. The rule types are split along different dimensions:

- phrase structure rules and lexical rules
- derivational and inflectional lexical rules
- unary and binary rules

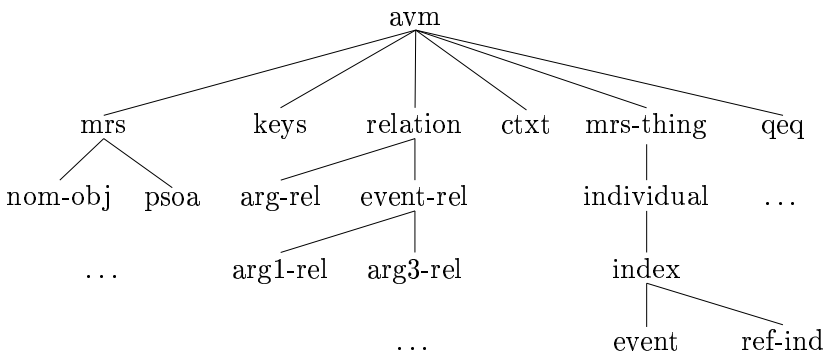

Figure 35: Types for semantic composition, excerpt from the Matrix

- headed and non-headed rules
- head-initial and head-final rules

The types from different dimensions cross-classify to yield the types needed to build the constructoinal types of the grammar, as illustrated in figure 36. The Matrix uses binary-branching structures only, a constraint which can be easily extended by defining additional types following the patterns of the basic definitions.

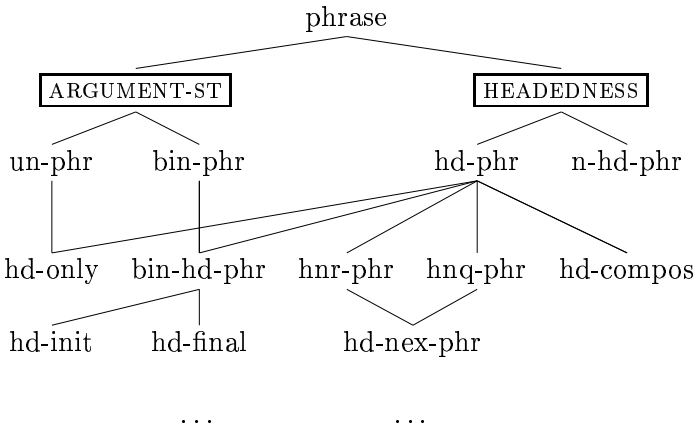

Figure 36: General classes of rules, excerpt from the Matrix

Figure 37 shows an example of the mother's amalgamation of semantic information from the daughters in a rule by means of difference lists. The value of C-CONT is the semantic information supplied by the rule itself and is treated much like an inherent additional daughter.

6.5 Constructional types

The rules of a grammar inherit from the constructional types in the type hierarchy. The basic constructional types in the Matrix include the following:

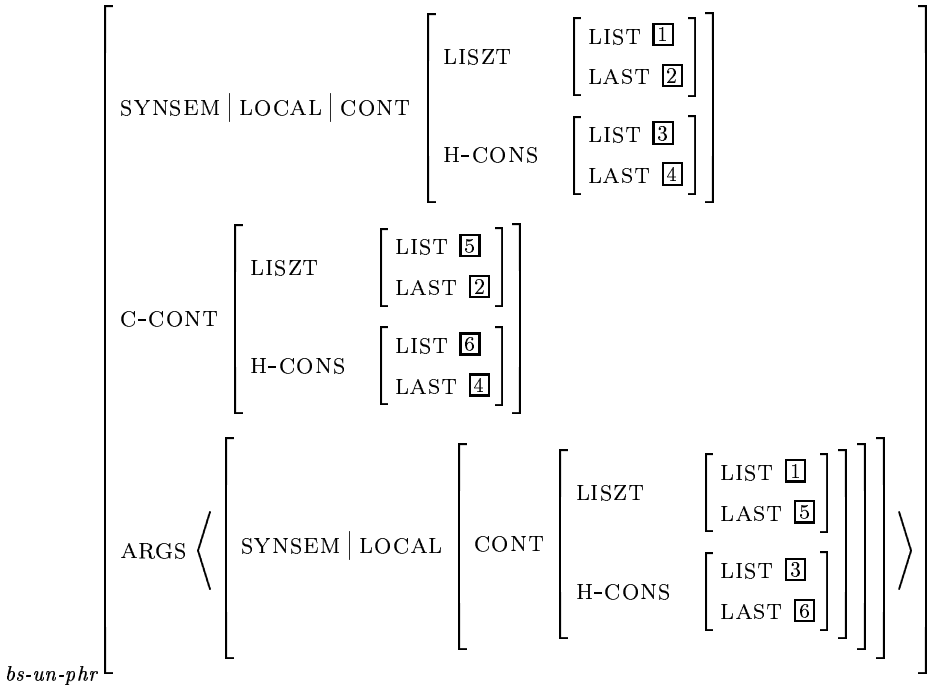

Figure 37: Abbreviated version of the type *basic-unary-phrase*

- head-specifier phrase
- head-complement phrase
- head-subject phrase
- head-modifier phrases
- extracted-argument phrases
- head-filler phrase

None of these phrase types are specified with regard to whether the ordering of their arguments should be head-final or head-initial. All of them except for the head-filler phrase inherit from the type *headed-phrase*, and with the exception of the unary phrase types for extracting arguments, all phrases are defined as binary.

As the constructional types supplied by the Matrix will be relevant to the application to Norwegian presented in section 7, I will in the following comment on some of them in particular, also focussing on properties that will be further discussed there.

The *basic-head-subject-phrase* combines a sign with a non-empty SUBJ list with its subject (as in *Gyrd smiler*/*Gyrd smiles*). Accordingly, the

synsem element on the head-daughter's SUBJ-list is unified with the SYNSEM value of the non-head daughter, as shown in figure 38

The mother structure is specified to have an *anti-synsem* element on its SUBJ-list, in order to distinguish structures with a lexically empty SUBJ-list from structures that originally had a *synsem* element on its SUBJ-list. The head daughter is specified to have an empty COMPS-list, meaning that any elements on the COMPS-list should be removed, e.g. by building a VP with the *basic-head-comp-phrase*, before the *basic-head-subject-phrase* is applied.

$$\textit{basic-hd-subj-phr}\left[\begin{array}{l}\text{SS}\,|\,\text{LOC}\,|\,\text{CAT}\,|\,\text{VAL}\left[\begin{array}{l}\text{COMPS}\ \langle\ \rangle\\ \text{SPR}\ \boxed{1}\\ \text{SUBJ}\ \left\langle\ \textit{anti-synsem}\ \right\rangle\end{array}\right]\\ \text{HD-DTR}\,|\,\text{SS}\,|\,\text{LOC}\,|\,\text{CAT}\,|\,\text{VAL}\left[\begin{array}{l}\text{SPR}\ \boxed{1}\\ \text{SUBJ}\ \left\langle\ \boxed{2}\ \right\rangle\end{array}\right]\\ \text{NH-DTR}\,|\,\text{SS}\ \boxed{2}\end{array}\right]$$

Figure 38: Abbreviated *basic-head-subj-phrase*

The *basic-head-comp-phrase* combines a sign with a non-empty COMPS-list with its first complement (as in the verb phrase *ser henne/see her*). Accordingly, the first *synsem* element on the head-daughter's COMPS-list is unified with the SYNSEM value of the non-head-daughter, as shown in figure 39.

The mother structure shares its COMPS-value with the REST-value of the head daughter's COMPS-list, i.e. the head-daughters COMPS-list minus the first element. Values for SUBJ and SPR are shared directly with the head-daughter.

$$\textit{basic-hd-comp-phr}\left[\begin{array}{l}\text{SS}\,|\,\text{LOC}\,|\,\text{CAT}\,|\,\text{VAL}\left[\begin{array}{ll}\text{COMPS} & \boxed{2}\\ \text{SPR} & \boxed{3}\\ \text{SUBJ} & \boxed{4}\end{array}\right]\\ \text{HD-DTR}\,|\,\text{SS}\,|\,\text{LOC}\,|\,\text{CAT}\,|\,\text{VAL}\left[\begin{array}{ll}\text{COMPS} & \textit{ne-list}\left[\begin{array}{ll}\text{FIRST} & \boxed{1}\\ \text{REST} & \boxed{2}\end{array}\right]\\ \text{SPR} & \boxed{3}\\ \text{SUBJ} & \boxed{4}\end{array}\right]\\ \text{NH-DTR}\,|\,\text{SS}\ \boxed{1}\end{array}\right]$$

Figure 39: Abbreviated *basic-head-comp-phrase*

The *basic-head-spec-phrase* combines a sign with a non-empty SPR-list with its specifier (as in the noun phrase *ei jente/a girl*). The first *synsem* element on the head-daughter's SPR-list is unified with the SYNSEM value of the non-head-daughter. Additionally, the SPEC-list of the non-head-daughter must contain a sign with a HEAD and COMPS value that unifies with the HEAD and COMPS values of the head-daughter. Unifying the entire synsem of the head daugter with SPEC on the non-head daughter, as is foreseen in Pollard and Sag (1994), would result in a cyclic feature structure, which is ruled out by the LKB formalism.

The *head-mod-phrase-simple* combines a sign with a non-empty MOD-list with a sign matching the constraints on the MOD-list element (as in the phrase *røde jenter/red girls*). The appropriate values of the non-head daughter's MOD-list element are unified with the corresponding values of the head-daughter, as seen in figure 40

The mother structure is specified to have an empty COMPS-list, hard-wiring the assumption that modification takes place after complementation. Values for SUBJ and SPR are shared directly with the head-daughter.

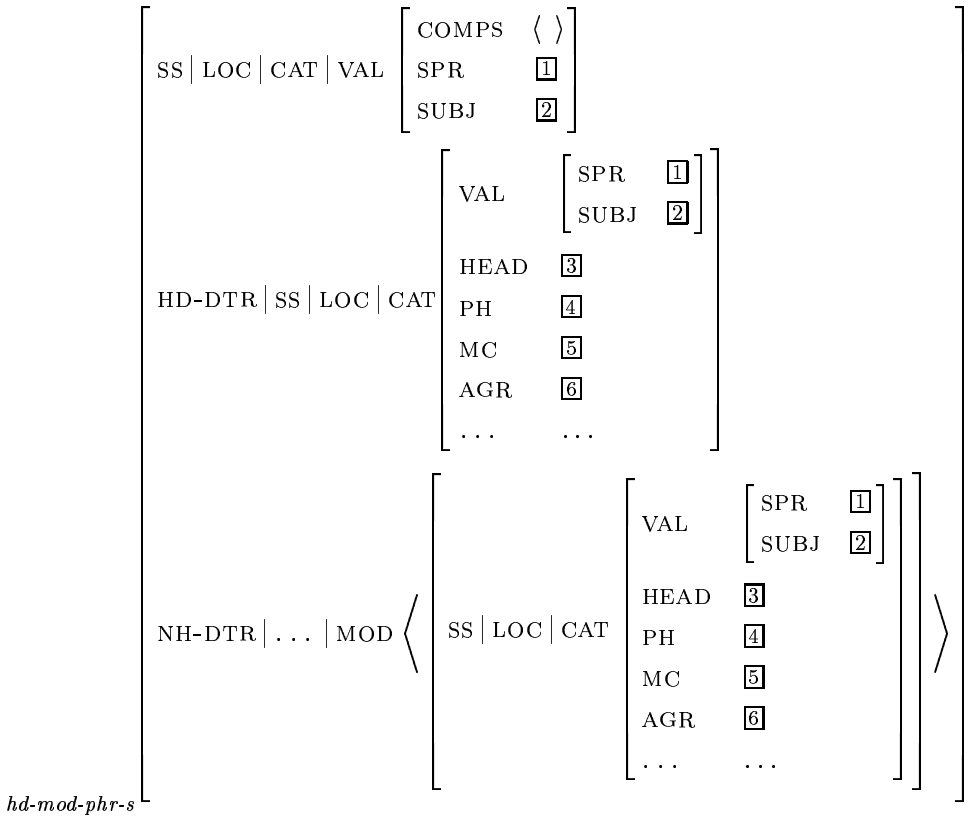

Figure 40: Abbreviated *head-mod-phrase-simple*

The types *head-adj-phrase* and *adj-head-phrase* are defined from *head-*

mod-phrase-simple by letting them inherit from *head-final* and *head-initial*, respectively, and by adjusting values like POSTHEAD and MODIFIED accordingly.

Two other subtypes, *scopal-mod-phrase* and *isect-mod-phrase*, are also inheriting from *head-mod-phrase-simple*, making it possible to make a distinction in the semantics between intersective and scopal modification. An intersective modifier shares its handle with the sign it is modifying, thus always having the same scope as the head it attaches to, and it takes its modifee's event or index variable as its argument. A scopal modifier has its own handle that is not shared by anything. It takes a handle as its argument that is qeq to the handle of its modifee, while e.g. quantifiers can scope inside or outside the modifier at will.

The *basic-head-filler-phrase* combines a structure with one element on its SLASH-list with a filler whose LOCAL-value can be unified with the *local*-element on this list. This mechanism is used for inserting topicalized elements at the front of a sentence.

The LOCAL-value of the first element on the ARGS-list is unified with the value of the NON-LOCAL.SLASH-value of the second element on the ARGS-list. The mother structure is left with an empty difference list as the value of SLASH, hard-wiring the assumption that at most one extraction occurs per clause.

The *basic-extracted-arg-phrase* is a prerequisite for the *basic-head-filler-phrase*. For an element to be topicalized and inserted at the first position of the sentence, it must first be removed from its original position. There are different phrase types according to what element should be extracted from the original structure. These types are *basic-extracted-comps-phrase*, *basic-extracted-subj-phrase* and *extracted-adj-phrase*.

In the type *basic-extracted-comps-phrase*, the first element on the head daughter's COMPS-list is constrained to be of the type GAP, and the gap's SLASH-value is shared with the head-daughters SLASH-value. The SLASH-value is passed subsequently from each head daughter to its mother by the constraints on the type *head-valence-phrase*.

The *basic-extracted-subj-phrase* constrains the head daughter's SUBJ-value to be of type GAP, which is also shared with its SLASH-value. The mother structure is constrained to have empty SUBJ-, SPR- and COMPS-lists, and to have the value MC - (indicating a non-main clause), meaning that the phrase can only be applied when all other elements have been combined with the head, and that only subclause structures allow subject extraction.

The *extracted-adj-phrase*, finally, introduces an element on the mother's SLASH-list that can modify the head daughter sign, i.e. that has an element on its MOD-list that shares important feature values with the head-daughter (mostly in analogue to the *head-mod-phrase-simple* on page 49).

7 Norwegian word order in HPSG

This section presents an implementation of word order phenomena in Norwegian clauses in the HPSG framework. The implementation is based on existing work for English, but Norwegian-specific solutions are presented where the word order differs between the two languages. Advantages and weaknesses of these solutions to the challenges of Norwegian clause structure are discussed. The actual implementation in the LKB grammar development environment is presented in more technical detail in section 8.

The grammar fragment described in this section covers the different word order phenomena in Norwegian main and subordinate clauses described in section 2, including:

- The V2-constraint
- Topicalization
- Adjunct placement in main and subordinate clause structures

7.1 The main clause

7.1.1 Norwegian vs. English

As the Matrix is based mainly on the LinGO ERG grammar for English, it is natural to compare the two languages to see what structural similarities one might utilize and what differences one has to take into account when building a Norwegian grammar.

The clause structure in an unmarked, declarative main clause shows the same placement of subject, finite verb and object(s) in both languages, namely SVO.

(28) Inge beundrer Tore.
Inge admires Tore.

Introducing a complex verb form, English and Norwegian are still alike in that they keep the verbal together as a unit.

(29) Inge har beundret Tore.
Inge has admired Tore.

Regarding the objects of the sentence, the case-less languages Norwegian and English show the same rigid ordering of the objects (IO - DO - PPO). The ordering can not be changed without affecting the semantics of the sentence.

(30) Inge har skjenket Tore en bok/*har skjenket en bok Tore.
Inge has given Tore a book/*has given a book Tore.

Both languages allow topicalization, i.e. stressing an element by fronting it, but structural differences arise from the process. English simply extracts the element from its place in the sentence and places it in front of an otherwise unchanged structure. Norwegian, on the other hand, strictly keeps the finite verb in second place, applying a V2 constraint to all declarative main clauses. The subject is then placed directly after the finite verb, before any infinite verbs and objects.

(31) Tore har Inge skjenket en bok.
Tore, Inge has given a book.

Due to the many similarities in the clause structure of the two languages, it should be possible to lean on already existing solutions to a great extent when building a hierarchy of needed constructional types for Norwegian. Constructional types for argument extraction, topicalization, the combination of a head with its complements and a head with its subject should be transferrable to a Norwegian grammar assuming that the necessary changes are made to account for the differences presented above.

7.1.2 The inverted subject

Also, the English grammar contains constructions where the ordering of the finite verb and the subject is inverted, though not in normal declarative main clauses. In English, only a finite auxiliary can precede the subject. This inverted order is present in questions as well as in other types of constructions, as shown in the sentences (32) to (34). The various cases are thoroughly discussed in Ginzburg and Sag (2001).

(32) Who would you like best?

(33) Would you like him?

(34) Had I met him before, I wouldn't have come.

Norwegian uses the same inverted verb-subject structure in many of these cases, but in addition applies the inverted structure to all main clauses where an element other than the subject has been topicalized, as seen in the different structures in sentence (34) and (37).

(35) Hvem ville du like best?

(36) Ville du like ham?

(37) Hadde jeg møtt ham før, hadde jeg ikke kommet.

As described in section 6, the basic constructional types are not quite tuned to this construction. The *basic-head-subject-phrase* has an empty COMPS list, and its head daughter's COMPS list is of type *olist*. Practically,

this means that all head complement rules are supposed to apply before any head subject rule, to build a VP. This accounts for the word order of strict SVO languages as well as for SOV languages, as subtypes of the *basic-head-complement-phrase* can be defined to be either head-initial or head-final. The inverted structure presents this approach with a problem, though, as the head and its complements are separated in the structure.

Different methods can be applied to make the subject appear to the right of the finite verb, before any of its complements. In the following three sections three possible solutions to the problem are presented. The first two solutions are taken from Sag and Wasow (1999) and Ginzburg and Sag (2001), respectively, whereas the third solution is the one chosen for the analysis in this thesis.

Sag and Wasow (1999)

In Sag and Wasow (1999), the problem is solved by a lexical rule that removes the subject from the SPR list[22] of the lexical entry for an inverted auxilliary as shown in figure 41. As a result of this, all elements on the ARG-ST list of the verb appear on its COMPS list, and the subject will thus end up as the first element on this list.

Since the inverted subject always appears to the right of the finite auxilliary, before the rest of the complements, this would be a possible solution in combination with a head-complement-rule that generates the complements of the COMPS list in the order of which they appear on it.

$$\textit{word}\left[\begin{array}{ll}\text{SYN} & \left[\begin{array}{ll}\text{HEAD} & \left[\begin{array}{ll}\text{FORM} & \text{fin}\\ \text{AUX} & +\end{array}\right]\\ \text{SPR} & \langle\ \text{NP}\ \rangle\end{array}\right]\\ \text{SEM} & \left[\text{MODE}\ \ \text{prop}\right]\end{array}\right] \rightarrow \textit{word}\left[\begin{array}{ll}\text{SYN} & \left[\begin{array}{ll}\text{HEAD} & \left[\text{INV}\ +\right]\\ \text{SPR} & \langle\ \rangle\end{array}\right]\\ \text{SEM} & \left[\text{MODE}\ \ \text{ques}\right]\end{array}\right]$$

Figure 41: Inversion Lexical Rule from Sag and Wasow (1999)

The solution presents the inverted structure in English as an exception to the base case, where a head-final head-subject phrase applies to the finite verb phrase (auxiliary or non-auxiliary) to combine the subject with the verb phrase in a non-inverted structure. In Norwegian, following the strict V2 constraint, this phrase must be blocked against any topicalization, whereas in English, a freely chosen element can still be topicalized. For the inverted structure, the opposite situation arises: only a wh-word can be topicalized in English, whereas any element can be topicalized in Norwegian.

[22]The feature SUBJ is not introduced in the simplified teaching grammar presented in this book.

As the inverted subject always appears in the same position in Norwegian as it does in English, the lexical rule solution is possible to implement. The differences mentioned above would have to be taken into account in the implementation of Norwegian, in addition to the fact that all finite verbs can give rise to an inverted structure in Norwegian, not only the auxiliaries as in English.

A practical problem arises from this solution, because the inverted structure is also present in regular main clauses with a topicalized element in Norwegian. This makes it possible to use a lexical rule to invert the order of subject and verb as shown in figure 41, and then at the next level topicalize the inverted subject back to the sentence-initial position, as all complements of the verb in a Norwegian main clause can be topicalized. This construction is better avoided, as it creates spurious parses and is unefficient, doubling the number of phrase applications compared to what is really needed to generate the phrase[23].

A second and more pressing problem is the fact that the Norwegian main clause allows for certain types of adjuncts both directly before and directly after an inverted subject, but not between the complements of the verb, should there be more than one. This makes it necessary to know whether the first element on the COMPS list was originally a subject or not. It is possible to solve this problem by marking the subject element with a specific type as well, in combination with the lexical rule, but the question is then if this is not just another technical detour that camouflages the problem rather than solving it.

Ginzburg and Sag (2001)

In Ginzburg and Sag (2001, pg.36), the subject is not camouflaged as one of the verb's complements. The type *subject-auxilliary-inversion-phrase* (*sai-ph*) is subject to the constraint seen in figure 42. Both the element from the head daughter's SUBJ list and the elements from its COMPS list are generated at the same time in this type of phrase, introducing both the subject and the complements as sisters of the lecixal head.

$$\left[\text{SUBJ } \langle\ \rangle\right] \rightarrow \text{H}\underset{word}{\left[\begin{array}{ll}\text{INV} & + \\ \text{AUX} & + \\ \text{SUBJ} & \langle\ \boxed{1}\ \rangle \\ \text{COMPS} & \boxed{A}\end{array}\right]}, \quad \boxed{1}, \quad \boxed{A}$$

Figure 42: Constraints on the type *sai-ph* from Ginzburg and Sag (2001)

Using a similar approach with a flat structure for Norwegian, we would

[23] A similar problem is in the LinGO ERG grammar taken care of by defining types for extractable and non-extractable structures. The inverted subject then defined as unextractable.

avoid the problems and technical detours needed to keep track of the inverted subject. To achieve this practically, the hierarchy of constructional types must include phrases more complex than just binary branching ones. The number of constructional types must be expanded to cover all possible permutations of inverted subject[24], finite verb and complements in a sentence.

Building constructional types that have more than two daughters into the hierarchy of phrase types is not a problem in the LKB system. But where the type *sai-ph* is a generalized phrase type, encompassing any possible number of complements, the number of daughters has to be specified for each constructional (leaf) type built in the LKB. Appropriate types for each construction shown below would have to be specified. This results in a larger number of constructional types than what is needed when working with binary branching structures only, but it is a possibility.

V_{finv} – Subj
V_{finv} – Subj – Comp
V_{finv} – Subj – Comp – Comp

Problems to this approach arise when it is combined with sentence-modifying adjuncts. Adjuncts placed at the sentence end can recursively modify a phrase with the help of a binary modifier phrase, and might not cause a problem. Modifiers inside the phrase represent a larger problem, as the number of adjuncts, in contrast to complements, theoretically is unrestricted. It is possible to let the adjuncts modify the lexical verb recursively before applying the flat constructional phrase, but this does not hold for the cases where the inverted subject precedes the adjunct. A theoretically endless number of constructional phrases would have to be postulated following the pattern shown below.

V_f – Subj – Adj – Comp
V_f – Adj – Subj – Comp
V_f – Adj – Subj – Adj – Comp

V_f – Subj – Adj – Adj – Comp
V_f – Adj – Adj – Subj – Comp
V_f – Adj – Adj – Subj – Adj – Adj – Comp

V_f – Subj – Adj – Adj – Adj – Comp
...

Even with a restricted number of adjuncts, using flat constructions leaves us with a very large number of possible constructions and corresponding types. It therefore seems wise to work solely with binary branching construction types, as is common practise in many implemented HPSG fragments and mainly intended in the Matrix.

[24] This approach still uses a head-final head-subject phrase as the base case.

The inverted head-subject phrase

The use of binary branching structures is compatible with the lexical rule approach from Sag and Wasow (1999). But to a greater extent than this approach can offer, a Norwegian grammar also needs an easy way to keep track of the subject of the clause. The easiest solution to this is to avoid any redefinition of the subject and simply use a head-subject phrase that combines the lexical head (i.e. the finite verb) with the inverted subject to its right directly, as shown in figure 43[25]. This approach recognizes the construction for what it is: a construction so common in the Norwegian language that it deserves to be recognized as a part of the core grammar and not to be rewritten in any way.

$$\begin{bmatrix} \text{SUBJ} & \langle\,\rangle \\ \text{COMPS} & \boxed{2} \end{bmatrix} \rightarrow \text{H}\begin{bmatrix} \text{VFORM} & \text{fin} \\ \text{SUBJ} & \langle\,\boxed{1}\,\rangle \\ \text{COMPS} & \boxed{2} \end{bmatrix}, \quad \boxed{1}$$

Figure 43: A head-initial head-subject phrase to be used with inverted structures

Changing the order of the head and the dependent is not a practical problem. Additionally, the *basic-head-subject-phrase* must be adjusted to pass the COMPS value up from daughter to mother instead of constraining it to be an empty list, so that it is possible for a head-complement phrase to apply to this structure afterwards. The practical result can thus be made the same as in the lexical rule approach and the *sai-phrase*, only without the detour and camouflage of the lexical rule, without the disadvantages of using flat structures and with the added bonus of increased control.

The phrase that is built by applying a head-initial head-subject phrase to the inverted structure does not correspond to any recognized linguistic entity such as the NP or the VP. On the contrary, it destroys the traditional connection between the lexical head and its complements as we find it in a traditional VP, a unit that has been proved to function as one linguistic object in many constructions. Following this, it is normally assumed that only complements are sisters of the lexical head, complements thus being deeper embedded in a phrase than subjects or specifiers.

Introducing the inverted head-subject phrase means breaking this assumption. It means choosing to take the surface structure of Norwegian clauses as a starting point for the analysis and rejecting the view of one underlying (syntactic) structure for all constructions. Figure 44 illustrates how constructional phrases can be applied in an inverted structure, following this analysis.

[25] For simplicity, the sketched feature structures in this section only include the features crucial to the problem at hand. For details on the placement and organization of the features, see section 6 or 8.

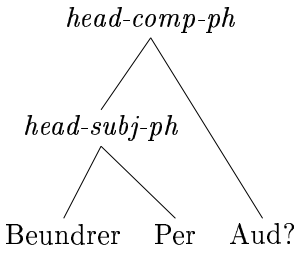

Figure 44: Inverted main clause structure

To be able to account for the many constructions where the traditional VP plays an independent part, as one linguistic object, a grammar for Norwegian must also be able to construct it. Luckily, it is possible to build a grammar that is both able to construct a VP and to apply a head-subject phrase before a head-complement phrase. The head-complement phrase implemented in the current grammar is for instance able to combine an infinite verb with its complements where no head-subject phrase has been at work (see page 75), and further variations of the analysis are possible.

Alternative approaches

It is possible to insist that a structure corresponding to a VP should always be built at some level in the analysis, but an implementation might then require rather heavy machinery. To motivate such descriptive power, the analysis would have to cover the wanted phenomena far better than its simpler sibling (on some measure of linguistic adequacy).

One such candidate is the transformational analysis presented in section 2.4, which would include head-movement in a practical implementation. This might be doable, but also more resource-demanding than an analysis without head-movement. Still, the most important argument against this solution is that it does not cover the selected phenomena in a satisfying mannner, as is discussed in section 2.4.

Other possibilities could for instance be based on work on discontinuous constituents, analyses and machinery developed for languages with a far more free word order than Norwegian, as presented in, among others, Müller (2000). For a language with such a rigid word order as Norwegian, though, this would presumably be an extravagant use of resources as long as a less complicated analysis can do the same job.

7.1.3 The non-inverted subject

The ordering of the finite verb and the subject in a non-inverted Norwegian sentence is equal to the ordering found in English as well as in the other Scandinavian languages. The supertypes for head-complement phrases and head-subject phrases defined in the Matrix could therefore theoretically be

applied to this structure in their original versions, resulting in standard-looking parse trees like the one shown in figure 45.

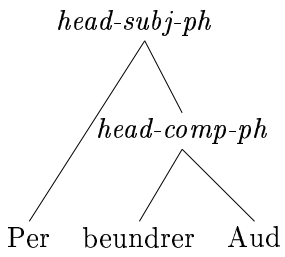

Figure 45: Non-inverted main clause structure

To account for this non-inverted structure, a head-final subtype of the *basic-head-subject-phrase* is needed in addition to the head-initial version described in section 37.

The original *basic-head-complement-phrase* that applies to the lexical head before any head-subject phrase (i.e. where the SUBJ value is passed up from head daughter to mother) can be used in combination with this head-final head-subject phrase. The head-complement phrase then realizes the complements as sisters of the lexical head and builds a VP.

If this solution is chosen, the *basic-head-subject-phrase* still needs to be redefined in the Matrix as described in section 37 to be able to account for both the inverted and the non-inverted structure. The head-initial phrase type for the inverted structure must pass up the COMPS value from the head daughter to the mother, while the head-final phrase type for the non-inverted structure must constrain the COMPS value of the mother and the head daughter to be empty lists. This means the head-final head-subject phrase has to have the same constraints as its supertype originally had, to ensure that no head-complement phrase can apply after it.

Because the *basic-head-complement-phrase* passes up the SUBJ value from head daughter to mother (as it has to, with this suggested implementation of the non-inverted structure), it must be made sure that the head-initial head-subject phrase can only apply to the lexical head. If not, structures where the subject interleaves or is placed after the complements would be considered grammatical due to the fact that the head-initial head-subject phrase constrains the mother to inherit its head daughter's COMPS value. A possible result is shown in figure 46.

It is possible to solve the problem of the two different verb-subject structures in Norwegian main clauses in this manner, and an implementation of this solution would build structures like the one shown in figure 45. Adjustments would have to be made, for instance to avoid topicalization in non-inverted senteces. This is not the exact solution chosen for the implementation of the grammar fragment for this thesis, though, and the alternative approach used in the implementation and its consequenses are described in more detail in the following section.

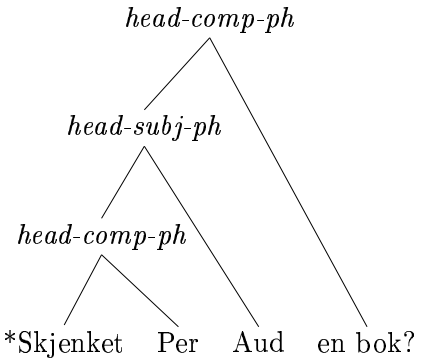

Figure 46: Implementation trap

7.1.4 Topicalization

Topicalization, i.e. stressing an element by placing it at the initial position of the sentence, is used in a large part of Norwegian main clauses, as it is in most of the Germanic languages. In HPSG the phenomenon is implemented by introducing constructional types for extracting an argument from its normal place in the structure and for filling the extracted argument in again at the wanted position. The types for extraction add an element to the mother's SLASH list that is equivalent to the complement or subject that one wants to extract (it is then removed from the valence list), or to an element that is able to modify the head daughter, in the case of adjunct extraction. The head-filler construction type fills in an element equal to the one from the head daughter's SLASH list at the first position of the sentence, i.e. to the left of the head daughter.

Both complements, adjuncts and subjects can be extracted from a structure. The basic types for extraction and the later re-introduction of an element is presented in section 6, and the grammar fragment uses the basic types as intended when it comes to topicalization of complements and adjuncts. Technical challenges and solutions related to topicalization of adjuncts are described in section 7.4 and 8.2.

Topicalization of the subject is normally used for cases where the subject is extracted from a subordinate clause and then topicalized in the main clause that has the subordinate clause as its complement. This phenomenon also exists in Norwegian, but my grammar implementation additionally uses subject extraction and topicalization to part main clause structures from subordinate clause structures.

A topicalized subject

The grammar fragment implements ideas from Diderichsen's two schemata for Norwegian clauses described in section 2.3. The differing clause patterns in Norwegian are divided into two basic patterns, one for the main clause structure and one for the subordinate clause structure. The challenge pre-

sented by the main clause structure is then to describe one basic pattern, despite the varying position of the subject.

If one follows Diderichsen's main clause pattern, the differing positions do not have to present a problem. It is possible to decide that the fundamental subject position is the position to the right of the verb, and that it has no special claim on the sentence-initial position. As a consequence, the subject must be extracted from its place in the original structure in the same way as complements and adjuncts are, and then inserted via a head-filler phrase to be placed in the initial position, as shown in figure 47. In most cases it is then possible to separate main and subordinate clause structures from each other from the very beginning of an analysis, as further described in section 7.2.

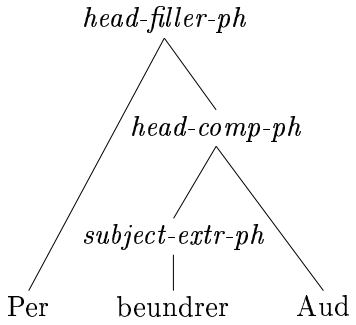

Figure 47: Main clause structure with topicalized subject

This analysis correctly predicts the structural ambiguity that arises from topicalized Norwgian main clauses, as both the subject and an object can occupy the sentence-initial position and none of them has to be marked for case. Both analyses shown in figure 48 are theoretically correct readings of the sentence *Per beundrer Aud*, even though it will be disambiguated to the first reading only in an unmarked case.

Structural vs. perceived ambiguity in Norwegian main clauses

The possibilitiy of topicalization, combined with the V2 constraint and the lack of case marking on nouns, results in vast possibilities for constructing syntactically ambiguous sentences in Norwegian. Still, syntactically ambiguous sentences (caused by these mechanisms) are not too often perceived as such.

The main source of real-life disambiguation is a rule saying that the sentence-initial element is given the syntactic function of subject in an otherwise unmarked sentence, as shown in sentence (38) and (39)[26].

[26]The example sentences in this section are collected from the Oslo-Bergen Corpora of tagged Norwegian texts. In the cases where a sentence is repeated with changed word order, the first sentence is the original sentence from the corpora. Sentence (42) and (43) have been somewhat simplified.

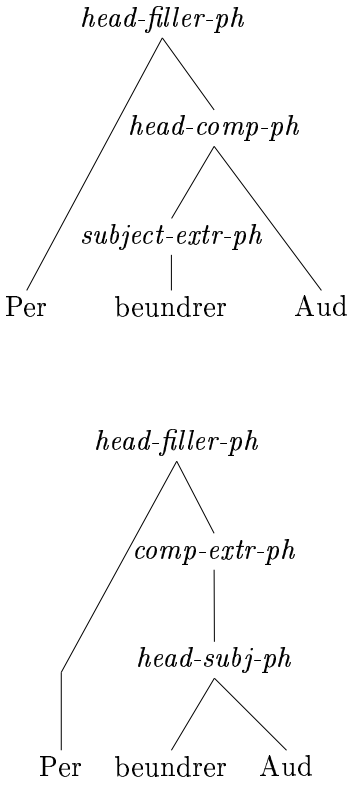

Figure 48: Structural ambiguity in Norwegian main clauses

(38) Jagland-S valgte København-O.
Jagland chose København

(39) København-S valgte Jagland-O.

A syntactically ambiguous sentence can also be disambiguated by stress[27] or by deduction from the semantic contribution of the words and/or the context of the utterance together with knowledge of the world. Pure syntactic means can also be of some help in the disambiguation process, for instance the fact that definite objects are topicalized more often than indefinite ones. Knowledge about the world and the feature definiteness can both be applied to suggest a disambiguation of the sentences (40) and (41)

(40) Svaret-O/?S fant Tranmæl-S/?O i teorien.
the-answer found Tranmæl in the-theory

(41) Tranmæl-S/?O fant svaret-O/?S i teorien.

Some computational lexica include extra-grammatical knowledge of the world, for instance whether a verb takes an animate or inanimate subject, and syntactic means like definiteness are a necessary part of most grammars. To utilize this information for disambiguation can bring a great improvement for grammar-based parsing, but for the scope of this thesis I conclude that syntactically, the sentences (38) to (41) are all ambiguous, even if they are disambiguated by the users of the language on a syntactic or a non-syntactic level[28].

The most efficient approach for a grammar implementation would be to use the rule for unmarked sentences and always mark down the initial element as the subject in sentences where it is not possible to decide specifically that an object or an adjunct has been topicalized. This would grant us a high score of right parses and spare us a great deal of (semantically) strange ones. But as long as efficiency is not the primary issue, it would also rule out quite legal, possible and probable readings of sentences. For this reason, the current grammar treats sentences like *Per beundrer Aud* as ambigous.

But even if syntax cannot solve all the problems in the world, some means that are unquestionable and easy to exploit are given to us also at the syntactic level. Complex verb forms do not give rise to the possible syntactic ambiguity described above at all, because the objects are complements of the infinite verb and thus extracted from the non-finite verb's complement list instead of from the finite verb itself, as shown in sentence (42).

[27] A topicalized object will often be stressed if the sentence is not marked in any other way.

[28] Disambiguation can seldom be made with one hundred percent accurracy. Thus, in a perfect system, some readings should be given a higher *probability score* than others. In her master thesis at the University of Oslo (to appear), Lilja Øvrelid explores different factors that influence this probability score. The factors include definitness and animacy, among others.

(42) Risikoen-O må leverandørselskapene-S bære.
the-risk must the-supply-companies carry

By simple verb forms, structural ambiguity is avoided if nexus adverbials are placed in the nexus field after the subject, effectively fencing in the canonical subject position after the finite verb and disambiguating the sentence as shown in sentence (43).

(43) Sjokoladen-O spiser jo tyvene-S sjelden.
the-chocolate eat after-all the-thieves seldom

In sentences where the subject or the object is a personal pronoun, the syntactical roles can be distinguished through case marking, as in sentence (44) and sentence (45).

(44) Boken-O finner du-S i Fjærland.
the-book find you-nom in Fjærland

(45) Det løftet-O holdt jeg-S.
that promise held I-nom

Still, the slow disappearance of case-specific personal pronouns[29] leads to structural ambiguity in sentences that would earlier have been unambiguous.

(46) Faren-S/O så han-S/O knapt.
the-father/the-danger saw he-nom/akk barely

7.2 The subordinate clause

Having established the main clause pattern following Diderichsen's clause schemata, the pattern for subordinate clauses is relatively easy to establish. The subordinate clause lacks the variation in the basic structure that makes life with main clauses complicated. The subject has only one possible position in the structure, before the finite verb.

Due to the rigid SVO structure, the basic constructional types from the Matrix adequate for describing English sentences could also be applied here, with the exception of the head-filler construction, of course. The head-final subtype of the modified *basic-head-subject-phrase* that correspond to the original basic type described in section 7.1.3 can thus be applied in this case.

Since the grammar treats the position after the finite verb as the fundamental position for the subject in a main clause, a head-final head-subject phrase can be defined to apply in subordinate clause structures only. Thus the head-final head-subject phrase can mark the new structure as [MC -] (main clause -), while the head-initial version can mark the structure [MC +], dividing the two structures into the two patterns as shown in figure 49 and 50.

[29] The nominative forms of the personal pronouns *han* (3rd, sing, masc), *hun* (3rd, sing, fem) and *de* (3rd, pl) are gradually replacing the accusative forms *ham*, *henne* and *dem*

The main clause structure with a topicalized subject is set to be [MC +] only by the application of a head-filler phrase, since subject extraction can take place in subordinate clauses as well as in main clauses.

To obtain a uniform constituent structure for both clause types, my grammar implementation applies the head-subject phrase before the head-complement phrase also in the subordinate structure. This results in structures like the one shown in figure 49 and was done for practical reasons rather early in the implementation process.

This solution may present a problem for VP coordination, as in the subordinate clause *[om] Tranmæl fant svaret og beundret Gyrd.* An intuitive solution might be to coordinate the two VPs *fant svaret* and *beundret Gyrd*, but this may be difficult in combination with the chosen analysis for subordinate clauses. Still, coordination data in general is far from conclusive. Choices of design made later in the process also probably make the task of managing two different strategies for the ordering of phrase application easier. But because redesigning this ordering would have consequences for a large amount of small details in other parts of the grammar, this experiment would exceed the scope of this thesis.

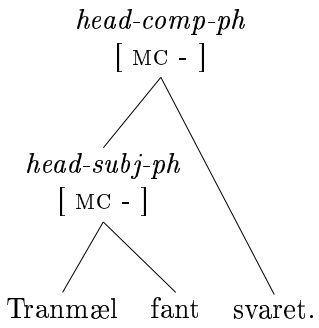

Figure 49: Subordinate clause structure

7.3 Declarative vs. interrogative clauses

Describing sentences in terms of main or subordinate clause structure (MC +/-) in this manner leaves for something to be wished for, as shown in figure 50. The structure in *Fant Tranmæl svaret?* has the characteristic inverted structure of a main clause, and this could be further illustrated by adding adjuncts before or after the subject, as in *Fant Tranmæl allikevel svaret?*. Still, we need to distinguish the two sentences *Fant Tranmæl svaret?* and *Tranmæl fant svaret* from each other semantically, as they represent different speech acts, one clearly indicating a question and the other one a proposition.

As the distinction between a question and a proposition in this case is expressed purely by the syntactic structure of the clause, it is possible to classify the constructions as a question or a proposition during a parse.

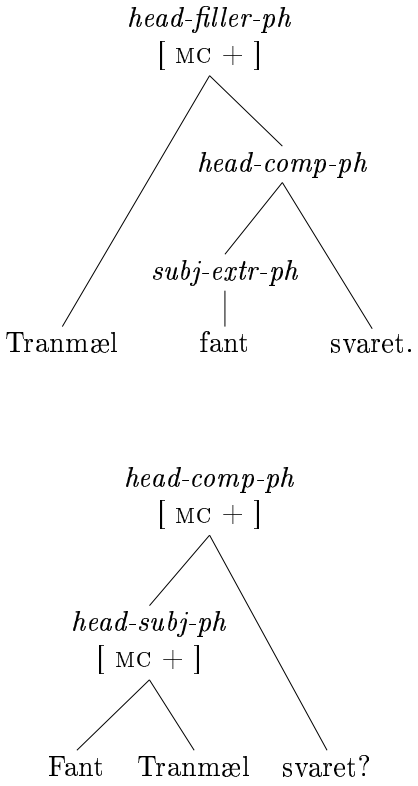

Figure 50: Main clause structures

For instance, a head-filler construction results in a proposition as long as the topicalized element is not a wh-word (as in *Hvem fant svaret?*). A main clause construction with no topicalized element (a V1 construction) can either be a command or a question, depending among other factors on the verb form.

To account for the different speech acts, maximal projections are divided into types of clauses according to their differing properties. Clauses differ from non-clauses in that they are given a message value. Following the Matrix, the possible message types are *question*, *command* and *proposition*[30]. The clause types distinguished from each other by these message values are *declarative-clause* (*proposition*), *interrogative-clause* (*question*) and *imperative-clause* (*command*), the constraints on the first two types shown in figure 51.

$$\textit{interrogative-clause}\left[\text{SS}\,|\,\text{LOC}\,|\,\text{KEYS}\,|\,\text{MESSAGE}\quad\left\langle\,!\;\textit{question}\;!\,\right\rangle\right]$$

$$\textit{declarative-clause}\left[\text{SS}\,|\,\text{LOC}\,|\,\text{KEYS}\,|\,\text{MESSAGE}\quad\left\langle\,!\;\textit{proposition}\;!\,\right\rangle\right]$$

Figure 51: Constraints on the type *interrogative-clause* and *declarative-clause* from the Matrix

The main vs. subclause *structural* difference that has hitherto been described is as explained encoded by the feature MC, the feature value + indicating a main clause structure and the value - encoding a subclause structure. This value can often be decided upon before the full sentence is parsed, in contrast to the proposition vs. question *semantic* difference encoded via the message type. The clause types are cross-classified with the constructional types that result in maximal projections, providing the suitable type of message for the construction, as sketched in figure 52.

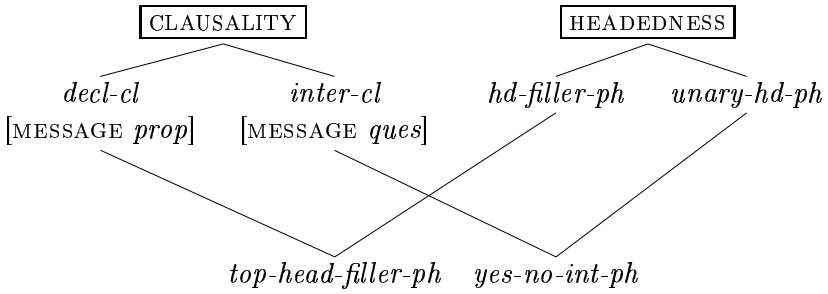

Figure 52: Cross-classification of clause types and constructional types

[30]This hierarchy has been greatly extended and put to use for analyzing several complex constructions of English in Ginzburg and Sag (2001), where clausality is treated as a totally independent dimension to be cross-classified with types from the dimension of headedness. My simplified inventory of messages is, of course, compatible with Ginzburg and Sag.

The inverted structure for *Fant Tranmæl svaret* shown in figure 50 is marked as [MC +], but it has not yet got a message value. This must be so, as the structure can function as a non-maximal projection, a possible head daughter for an adjunct-extracting phrase that will then again be a potential head daughter for a head-filler phrase. In this case, the construction might end as a declarative clause, as in the clause *Allikevel fant Tranmæl svaret* shown in figure 53.

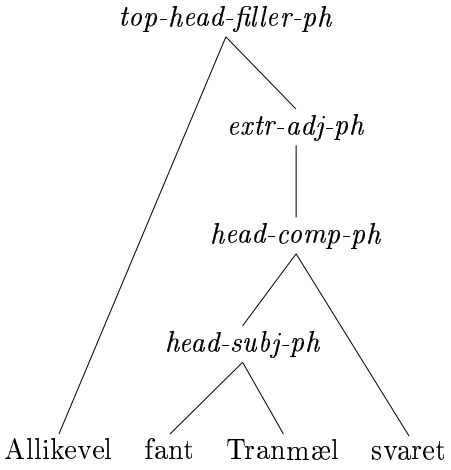

Figure 53: Structure with topicalized adjunct

To recognize the inverted, non-topicalized structure as a maximal projection, a unary phrase applies to it and gives it a message value as explained in detail in section 8.1 and shown in figure 54. The structure must be marked [MC +] to ensure that the construction has got an inverted subject, and its SLASH list must be empty, i.e. no elements have been extracted from the structure. It must also not have a message value already, to avoid applying the phrase more than once or after a head-filler phrase. After the application of the *yesno-int-phrase* the phrase can function as a maximal projection, and it is blocked against application of any extraction rules.

The grammar is implemented so that only maximal projections are given a message value. For technical reasons, subordinate clauses are at the moment not given a message value until they combine with a subjunction.

7.4 Adjuncts

As mentioned in section 2.1.5, possible placement of adjuncts in a sentence is also a way of distinguishing a main clause structure from a subordinate clause structure, in addition to the placement of the subject. Adverbs are allowed to occupy different positions in the two different structures, and this must be taken into account when building the phrases of the grammar. In addition, different types of adverbs may have different constraints on placement within the same structure.

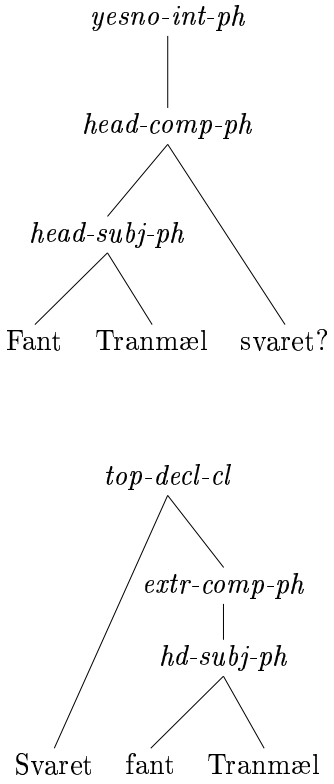

Figure 54: Interoggative vs. declarative clause type

Constructions where adjuncts behave differently according to whether they belong to the group of free adverbials or to the group of sentence adverbials (as shown in section 2.1.5) are not implemented in the current grammar. The adjuncts can so be classified with regard to their distribution only. As this distribution is not decided upon by syntactic category of the adjuncts or other features present in the grammar, it is natural to implement the distribution as a dimension on its own, independent of the other dimensions treating syntactic or semantic categorization.

Practically, and following section 2.1.5, what we need is: i) a type of adjunct that can only stand in the nexus field of a sentence, ii) a type of adjunct that can only stand at sentence end, and iii) a type of adjunct that can occupy all positions. Additionally, it must be defined whether the group of adjuncts can be fronted or not. This divides group 1 into two sub-groups, as some of the adjuncts that can stand in the nexus field can be fronted whereas others can not. A more fine-grained classification would become necessary if one wishes to implement groups of free adverbials whose distribution differ from the ones mentioned. This is possible, but not done in the present grammar.

The group of adjuncts that can only be placed in the nexus field needs a constraint stating that its members can only modify an element that has not yet been combined with any of its complements. This is done via the feature NUC (nucleus phrase), which is set to + for all lexical items of type *verb* and then changed to - when a head-complement phrase or a complement-extraction phrase is applied. The adjunct can so modify the verb both before and after a head-subject phrase has been applied, but it can not be placed between or after complements of the verb. The two subtypes of this type of adjunct do not have any additional constraints on their own as they are separated into two types only to signal which type can and which type can not be fronted.

As explained in section 2.3, the placement of the adjunct positions in the nexus field varies between main and subordinate clauses. This difference is straightforwardly accounted for in the phrase types that handle modification. In main clauses, the nexus adjunct can only be placed after the head, before or after an inverted subject. Thus a head-initial head-adjunct phrase that takes a finite verb phrase as its head daughter and a nexus adjunct as its non-head daughter applies to main clause structures only and marks the stucture as [MC +], as shown in figure 55. In subordinate clauses, the nexus adjuncts can only be placed before the head, so a corresponding head-final head-adjunct phrase applies to subordinate clause structures only and marks the structure as [MC -], as shown in figure 56.

The MC value is shared between the head daughter and the mother in the head-adjunct phrases, i.e. this constraint demands that the head daughter has either got the same MC value as the mother is given by the head-adjunct phrase, or it has got the underspecified value *bool*, which is compatible with the value the mother is given. More technical details concerning the head-adjunct phrase types are described in section 8.1.

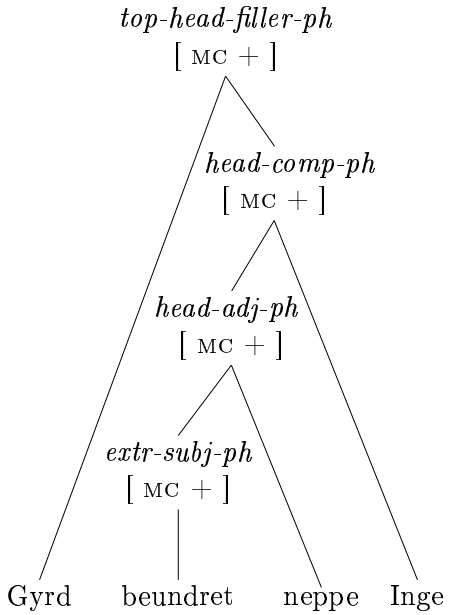

Figure 55: Main clause structure with nexus adjunct

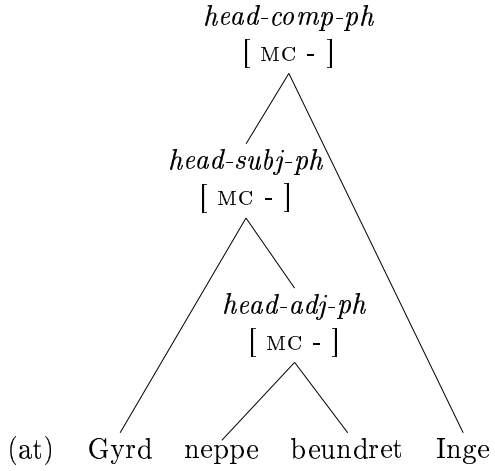

Figure 56: Subordinate clause structure with nexus adjunct

As explained in section 2.2, the decision has been made that adjuncts can not be placed in the position before the subject in a subordinate clause in this grammar[31]. The constraints ensuring this are also laid on the constructional types described in section 8.1, and hence do not influence the lexical types of the adjuncts.

The group of adjuncts that can be placed solely at sentence end needs the opposite constraint of the nexus adjuncts, specifying that its elements can only modify an element that has already been combined with all its complements, i.e. that has an empty COMPS list or a COMPS list of type *olist*. The modification must take place before the head-filler phrase is applied, and this is ensured through the constraint that the MESSAGE list of the modified element must be empty[32]. An inverted structure with no topicalization can thus be modified before the unary *yesno-int-phrase* is applied and the structure is given a message value, as shown in figure 57.

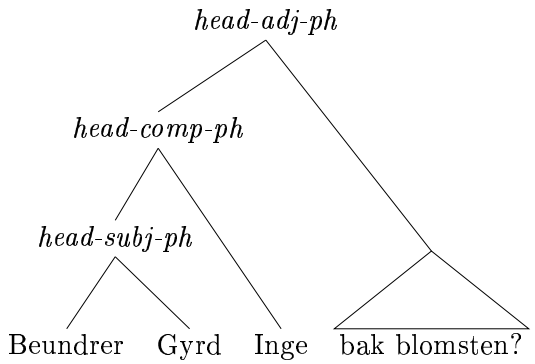

Figure 57: Inverted clause structure with sentence-final adjunct

The last group, whose elements can be placed in all possible adjunct positions of a clause, must be constrained to modify an element that has either not been combined with any of its complements or with all of them. This task is solved through the interaction of underspecification and multiple inheritance in the type hierarchy. The type of adjunct is defined as a common supertype for both the nexus adjunct type and for the sentence-final adjunct type. By application of a head-modifier phrase, the type can be forced down to one of its further specified subtypes as further described in 8.1. Details on the type hierarchy are given in section 8.2.

To front an adjunct, an extracted-adjunct phrase must be applied to a clause structure, as shown in figure 53. The extracted-adjunct phrases must be constrained to only extract the types of adjuncts that can be fronted.

[31] If this should be allowed, it would be an advantage to apply the head-subject phrase first in subordinate clauses, as is done in the current grammar. This would ensure a uniform treatment of sentence-modifying adjuncts.

[32] This is one of the reasons why subordinate clauses are not given a message value before they combine with a subjunction in the current grammar. As the head-subject phrase is applied first, the adjuncts at sentence end must be able to modify the whole structure in the same way as by the V1 interrogative clause shown in figure 56.

Adjuncts can be extracted from the positions in the structure where they normally occur. This implies that a clause with a topicalized sentence adverbial results in several different parse tree constructions, one for each position the adverbial can have in the clause, as shown in figure 58. All of the generated structures are correct, since it is impossible to decide from which position the adverbial has been extracted. Still, it is at this point no advantage to generate all possible trees for a clause of this type, as the only difference between them will be the order of the phrase application. Any semantic contribution that the adverbial might make when it is placed in a non-fronted position (like stress) is for instance lost when it is fronted. For this reason, the grammar constrains the adverbial and the extracted-adjunct phrases in a way that makes sure that adverbials can only be extracted from one position in the clause, as described in section 8.2.

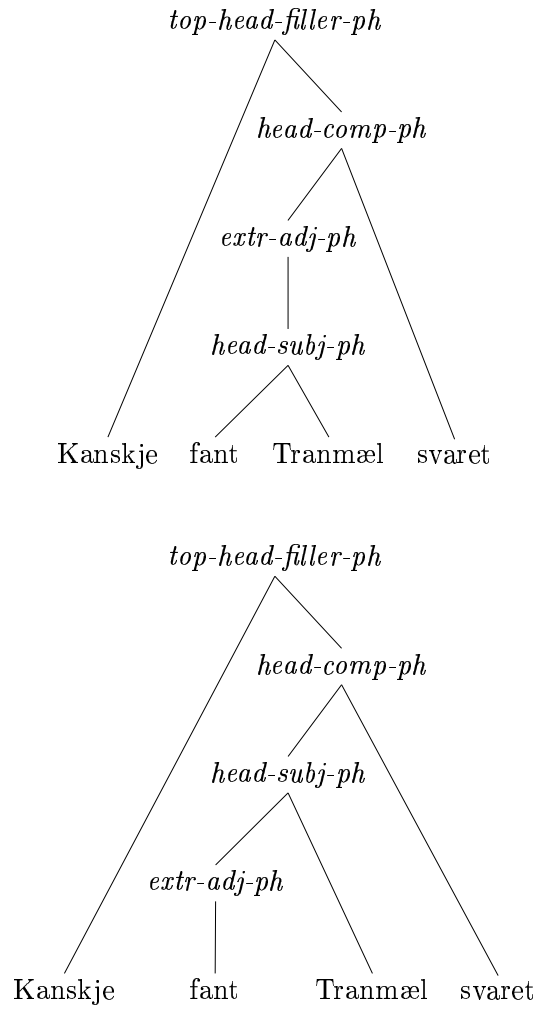

Figure 58: Possible structures with topicalized sentence adverbial

8 Norwegian word order in the Matrix and LKB

This section presents the tecnical details of parts of the implementation described in section 7. Changes that need to be made to the Matrix to support this implementation and types for the Norwegian grammar inheriting from the (adjusted) basic Matrix types are both described. Important for the implementation of Norwegian word order is primarily the constructional types needed for the grammar rules, which inherit from the basic constructional types presented in section 6.5, and the hierarchy of adjuncts described in section 7.4. Additional grammar features are presented in section 8.3.

To complete the documentation of my implemented grammar fragment, the following sections review all relevant aspects of the implementation in considerable technical detail. Throughout the discussion, aspects of the type hierarchy are combined with comments on the underlying motivations and interactions with other parts of the grammar.

8.1 Constructional types

As explained in section 7, the basic constructional types from the Matrix must be somewhat adjusted to support my account of Norwegian word order. Each constructional type is described in this section, first explaining the changes that had to be made to the original type from the Matrix, then presenting the subtypes of the original type that is needed for the Norwegian grammar fragment.

The *basic-head-subject-phrase* from the Matrix needs two subtypes in the grammar to supply the two needed grammar rules, one head-final for the subclause construction and one head-initial for the inverted main-clause construction. The phrases must be marked MC (main clause) - or +, respectively, as shown in figure 59.

Some minor changes need to be made in the original *basic-head-subject-phrase* in the Matrix to support this grammar implementation. As the head-subject-rule applies before the head-complement-rule the COMPS-value of the phrase can not be *null*, and the COMPS-value of the head-daughter can not be *olist*. Instead, the COMPS-value must be underspecified and shared between mother and daughter to pass it up in the structure.

The MC value of the head-daughter is changed from *na* to unspecified, and the value has to be shared between mother and head-daughter. This change is necessary because sentence adverbs can modify a verb before a head-subject-rule is applied. Pre-head modification will then result in the value [MC -], while post-head modification will result in the value [MC +], as described on page 77.

The diff-list values for LISZT and H-CONS in C-CONT are not specified in the *basic-head-subject-phrase*, as they are in the other constructional phrase

types. Since the head-subject-rule does not make any semantic contribution of its own in this grammar, they are both specified to have an empty difference list as value.

The *head-subj-phrase* inherits from *non-clause* to make sure that it can not function as a main clause on its own, even in an intransitive verb construction like *Smiler jenta?* (*Does the girl smile?*). The *yes-no-interrogative-phrase* has to be applied to this structure to yield a main clause of type *interrogative-clause*. Furthermore, the subjunction *at*, which can select for a main-clause construction as its complement is thus blocked from selecting *smilte jenta* as its complement, specifiying its main-clause complement to be of type *declarative-clause*.

```
head-subj-phrase1 := basic-head-subj-phrase &
                     rule &
[  SYNSEM.LOCAL.CAT.MC #mc,
   HEAD-DTR.SYNSEM.LOCAL.CAT.MC #mc,
   C-CONT [ LISZT <! !>,
            H-CONS <! !> ]].

subj-head-phrase := head-subj-phrase1 &
                    head-final &
[ SYNSEM [ LOCAL.CAT.MC - ],
  RULE-NAME 'subjh ].


head-subj-phrase := head-subj-phrase1 &
                    head-initial &
                    non-clause &
[ SYNSEM [ LOCAL.CAT.MC + ],
  RULE-NAME 'hsubj ].
```

Figure 59: TDL definition of the head-subject phrase types

The *basic-head-comp -hrase* from the Matrix only needs one subtype in the grammar, shown in figure 60, and no changes needs to be made to the original type. The subtype specifies that the phrase has the value [NUC -], as the COMPS list is not complete anymore when the phrase type is applied. The subtype also specifies that the head-daugther's SUBJ list has to be of the type *olist*. This type of list specifies that the list must either be empty, or all list elements must be of type *unexpressed-synsem* and have the value [OPT +].

The reason for this implementation is the fact that the *head-comp-phrase* should be able to apply to elements with an empty SUBJ-list (for instance prepositional phrases), to elements with a SUBJ-list containing an element of type *anti-synsem* (for instance after applying a head-subject-rule to a verb)

```
head-comp-phrase := basic-head-comp-phrase &
                    head-initial &
                    rule &
[  SYNSEM.LOCAL.CAT.NUC -,
   HEAD-DTR.SYNSEM.LOCAL.CAT.VAL.SUBJ olist,
   RULE-NAME 'hcomp1 ].
```

Figure 60: TDL definition of the type *head-complement-phrase*

and finally to elements where the SUBJ-list only contains elements of type *unexpressed-reg*.

The last case is a result of a solution chosen for dealing with infinite verb forms in this grammar. The type *verb-lxm* is defined with an element of type *synsem* on its SUBJ-list.

Still, infinite verbs do not have a syntactic subject that is supposed to be present in the surface structure in connection with this verb, but share their semantical subject with the subject or object of the finite verb in the construction. To have two definitions of each verb lexeme, one for finite verbs with a *synsem* subject and one for infinite verbs without a *synsem* subject, is going against the idea of keeping types as simple and general as possible. To treat the raising constructions mentioned above, it is also simpler to keep a synsem element on the SUBJ-list for infinite verbs as well, to handle the semantics of the phrase.

As long as the verb lexemes are defined in the way described above, their SUBJ-list element can not be further specified to be of type *anti-synsem* because *anti-synsem* only inherits from *synsem-min* in addition to *unexpressed*, and it is therefore a sister and not a descendant of *synsem*. The SUBJ-list element can be specified to be of type *unexpressed-reg* though, because *unexpressed-reg* inherits from *synsem* as well as from *unexpressed*. This makes the definition in figure 61 for the type *psp-verb* possible, as it will not clash with the definition of the subtype of *verb-lxm* it is cross-classified with. This definition avoids the chance of the subject of the infinite verb form ever being physically present in the structure, while still transferring semantic information.

```
psp-verb := sign &
[ SYNSEM.LOCAL [ CAT [ HEAD.VFORM psp,
                       VAL.SUBJ < unexpressed > ],
                 CONT.INDEX.E.TENSE pastperf ] ].
```

Figure 61: TDL definition of the type *psp-verb*

Following the definition of *olist*, this presumes that the subject's OPT value is undefined in the verb lexemes, and then set to + for the finite verb-forms in the inflectional rules.

This approach works for this grammar, but it is well possible that it might complicate matters in a larger grammar. What might be preferred is to use the inflectional rules to supply the infinite verb forms with an *anti-synsem* subject. The hypothesis in the Matrix, that inflectional lexeme-to-word rules pass the synsem value from the argument to the mother unchanged, would then have to be broken.

The extracted-subj phrase in the Matrix needs to be changed in the same way as the *basic-head-subject-phrase*. The COMPS value of mother and head-daughter needs to be shared and unspecified, as subject extraction takes place before the head-comp-rule can apply. The head daughter's MC value can not be set to *na* because this value might already have been decided upon by modification of the head-daughter, and this value must also be shared between mother and daughter.

In the Matrix, the mother's MC value is set to -, to ensure that subject extraction can only take place in subordinate clauses. As this grammar uses subject extraction to build the main clause structure as well, this constraint must be removed.

The INDEX value of the *gap* on the head-daughter's SUBJ list is changed from *ref-ind* to *individual* to allow for extraction of the formal subject *det*. The word *det* has an INDEX value which is a subtype of *expl-ind*, as in *det regner* (*it rains*).

Only one subtype of the *basic-extracted-subj-phrase* is needed in the grammar, specifying the value of C-CONT in the same way as in *basic-head-subject-phrase*, as shown in figure 62.

```
extr-subj-phrase := basic-extracted-subj-phrase &
                    head-compositional &
                    rule &
[ HEAD-DTR.SYNSEM.MODIFIED notmod-or-lmod,
  C-CONT [ LISZT <! !>,
           H-CONS <! !> ],
  RULE-NAME 'extr-subj ].
```

Figure 62: TDL definition of the type *extr-subj-phrase*

Additionally, the head-daughter is specified to have the value for MODIFIED be *notmod-or-lmod* (not modified or left modified). This is done to avoid spurious parses for sentences containing a modifier in the nexus field. As modifiers can stand both before and after the subject in the nexus field in a main clause, it would otherwise be possible to extract the subject from the structure both before or after the verb has been modified.

By specifying that the head-daughter must be *notmod-or-lmod*, extraction can only take place before the verb has been modified in a main clause, because there only topicalized modifiers may stand to the left of the finite verb. The reason for using *notmod-or-lmod* instead of only *notmod* is to al-

low for subject extraction also in subordinate clauses, as the grammar does not allow adverbials to modify a *subj-head-phrase* (when the subj-head-rule has been applied in a subordinate clause). If the subclause does have modifiers to the left of the verb, the subject must then be extracted after they have modified the verb, since nothing is allowed to modify the verb once the subject is removed.

The *basic-extracted-comp-phrase* from the Matrix does not need to be modified. It needs one subtype that constrains the head daughter's SUBJ value to the type *olist*, for the same reasons as described above for *head-comp-phrase*. The value of NUC is also constrained to be -.

The *adj-head-int-phrase* and *head-adj-int-phrase* from the Matrix do also not need to be modified before they are put to use in my grammar. They both inherit from the *head-mod-phrase*, though, where the COMPS value of the phrase is changed from *null* to an underspecified list shared between mother and head daughter. This is necessary, of course, because modifiers can also modify elements with a non-empty COMPS list.

The phrase types needed for modifying noun-phrases, *n-hadj-int-phrase* and *n-adjh-int-phrase*, only need to specify that the head daughter's HEAD value must be of type *noun*. These phrase types then apply both to the adjectives and prepositonal phrases that can modify noun phrases in the grammar.

The phrase types needed for modifying verb-phrases need additional constraints, which is the reason why the division between verb-modifying and noun-modifying phrases must be made here in the first place[33].

The *s-int-phrase* is a verb-modifying subtype of the type *isect-mod-phrase*, constrained to have a head daughter with a HEAD value of type *verb* and with the value of VFORM set to *fin*. It needs three subtypes that cross-classify with one of the types *adj-head-int-phrase* and *head-adj-int-phrase* to cover all cases present in the grammar, *s-hadj-int-phrase*, *s-main-nex-hadj-int-phrase* and *s-sub-nex-adjh-int-phrase*.

The type *s-hadj-int-phrase* shown in figure 63 is used for the post-head modification of the verb that is common for main and subordinate clauses. The only possible position for such modification is at the sentence-final position, and the type thus only specifies the head daughter's HEAD value to be of type *final-or-strictly-final-mod*. The hierarchy of adjunct head types is presented in detail in section 8.2.

The head-initial type *s-main-nex-hadj-int-phrase* shown in figure 64 then covers all the cases where adjuncts are placed in the nexus field of main clause structures in my grammar. The phrase type is thus specified to have the MC value +, blocking for any nexus adjuncts after the finite verb in a

[33] Such a syntactic distinction is not uncommon and it is done for several reasons, among others partly semantic. See for instance the LinGO ERG.

```
s-hadj-int-phrase := s-int-phrase &
                     head-adj-int-phrase  &
[ NON-HEAD-DTR.SYNSEM.LOCAL.CAT.HEAD
    final-or-strictly-final-mod ].
```

Figure 63: TDL definition of the types *s-hadj-int-phrase*

subordinate clause, and its non-head daughter's HEAD value is constrained to be of type *nex-mod*.

```
s-main-nex-hadj-int-phrase :=  s-int-phrase &
                               head-adj-int-phrase  &
[ SYNSEM.LOCAL.CAT.MC +,
  NON-HEAD-DTR.SYNSEM.LOCAL.CAT.HEAD nex-mod ].
```

Figure 64: TDL definition of the type *s-main-nex-hadj-int-phrase*

As its counterpart, the head-final type *s-sub-nex-adjh-int-phrase* shown in figure 65 handles adjuncts in the nexus field of subordinate clause structures. The value of MC is set to -, thus blocking the pre-head modification of the verb that is not allowed in a main clause. The non-head daughter's HEAD value is set to *nex-mod*. The SUBJ value of the head daughter is set to be a list containing an *expressed-synsem*, in order to block modification to the left of the subject. If modification in this position was considered desirable (see the discussion in section 2.2), this constraint could be relaxed to just the type *olist*, or be left unspecified. The type *expressed-synsem* is used instead of, for example, *canonical-synsem* to ensure that the subject can still be extracted.

```
s-sub-nex-adjh-int-phrase :=  s-sub-int-phrase &
                              adj-head-int-phrase &
[ SYNSEM.LOCAL.CAT.MC -,
  HEAD-DTR.SYNSEM.LOCAL.CAT [ VAL.SUBJ < expressed-synsem > ],
  NON-HEAD-DTR.SYNSEM.LOCAL.CAT.HEAD nex-mod ].
```

Figure 65: TDL definition of the type *s-adjh-int-phrase*

If a distinction between scopal and intersective modifiers was needed for a semantic reason (see section 4), corresponding types could be defined for scopal modifiers as well in a straightforward fashion.

The *basic-extracted-adj-phrase* from the Matrix does not need to be modified. It needs two subtypes in this grammar, both shown in figure 66.

The type *adv-extracted-adj-phrase* extracts adjuncts that can be placed in the nexus field of a sentence. The extracted argument is specified to have

```
adv-extracted-adj-phrase := extracted-adj-phrase &
[ SYNSEM.NON-LOCAL.SLASH.LIST.FIRST.CAT.HEAD emph-mod,
  HEAD-DTR.SYNSEM.LOCAL [ CAT.VAL.SUBJ < anti-synsem >,
                          KEYS.MESSAGE 0-dlist ]].

adv-fin-extracted-adj-phrase := extracted-adj-phrase &
[ SYNSEM.NON-LOCAL.SLASH.LIST.FIRST.CAT.HEAD
                                    strictly-final-mod,
  HEAD-DTR.SYNSEM.LOCAL.KEYS.MESSAGE 0-dlist].
```

Figure 66: TDL definition of the subtypes for adjunct extraction

a HEAD value of type *emph-mod* (see section 8.2 for details on the adjunct head types).

The head-daughter's *anti-synsem* subject is specified to avoid spurious parses, as the adjunct otherwise can be extracted both before and after the head-subject-rule has been applied.

Setting the head daughter's MESSAGE value to *0-dlist* is done to avoid extracting an adjunct from a sentence (i.e. once the fillhead-rule or the yes-no-rule has been applied).

The type *adv-fin-extracted-adj-phrase* extracts an adjunct from the final position of a sentence. The extracted argument is specified to have a HEAD value of type *strictly-final-mod*.

The *basic-head-filler-phrase* from the Matrix does not need any modification. As wh-questions are not yet implemented in the grammar, it also needs only one subtype, resulting in a propositional phrase.

The type *fillhead-phrase* inherits from the types *head-final*, *declarative-clause* and *head-compositional*. The type fills in an element according to the value of the head daughter's SLASH list. The head daughter must be of type *verb* and have MC value +, which is shared with the mother. As the phrase is a maximal projection, a message relation is introduced on the LISZT difference list via the value of C-CONT.LISZT. This message relation is shared with the mother's MESSAGE value which is a proposition.

The type also adds a qeq handle constraint to the mother's H-CONS list via the C-CONT.H-CONS value of the phrase. This qeq element ensures that the SOA handle of the message of the phrase takes scope over the key handle of the phrase.

The *yes-no-interrogative-phrase* is a unary rule that inherits from the types *unary-phrase* and *interrogative-clause* from the Matrix. Together with the *fillhead-phrase*, this is the only other phrase type that yields a maximal projection, i.e. that can function as a root element. It takes as its argument an inverted finite verb phrase with an empty COMPS list where no topicalisation has taken place and where no argument has been extracted from the

structure. This is ensured by constraining the argument's SLASH value to be *0-dlist* and its MESSAGE value also to be of type *0-dlist*.

The mother is given a *quest-rel* element on its MESSAGE difference list to show that the phrase is a question. This is shared with the element on the C-CONT.LISZT difference list to add it to the LISZT difference list of the phrase.

8.2 Adjuncts

In the Matrix phrase types are defined for both intersective and scopal modification, cross-classifying with the types for head-initial and head-final phrases to yield all the possible combinations of the four types. Hence what the grammar needs is lexical types for the different types of adjuncts, specifying what type of constituent they modify.

The implementation of adjuncts does as already mentioned not directly follow the classification of adjuncts (i.e. adverbials) found in section 2.1.5, where the distinction is made between free adverbials on one side and sentence adverbials on the other side. This exact classification is not implemented in my grammar because there are no syntactic phenomena that depend on it within the scope of this thesis. In the interest of transparency, the current implementation avoids irrelevant distinctions, though from the discussion in section 2.1.5 it should be clear how a more fine-grained distinction could be incorporated quite easily.

Instead, the type hierarchy shown in figure 67 covers only the different distributional possibilities associated with the corresponding types of adverbials in section 2.1.5. That is to say, that if a sentence adverbial and a free adverbial have the same distributional possibilities in a sentence, their implementation is identical in this grammar. If such a classification was needed it could easily be defined as a new sub-hierarchy, to be cross-classified with the distributional types at need, without loss of generality. Aditional semantic classifications could be treated in the same way.

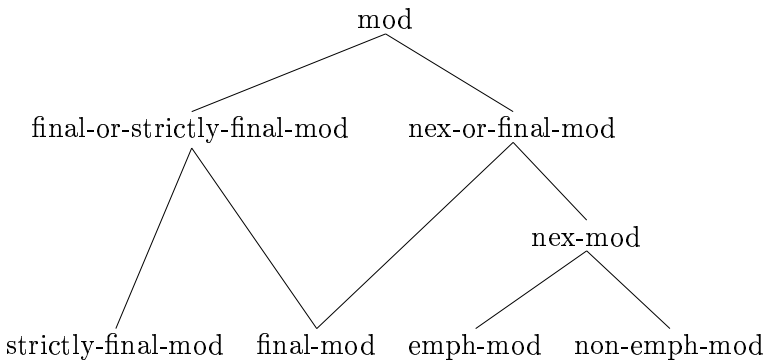

Figure 67: Type hierarchy below *mod*

The only sub-hierarchy that is implemented to cross-classify with the distributional types in this grammar are grammatical categories like adverb

(*adv*) and preposition (*prep*), subtypes of *head.* as shown in figure 68.

```
adv :=  adverbee.
nex-or-final-adv := adv & nex-or-final-mod.
final-adv := nex-or-final-adv & final-mod.
non-emph-adv := adv & non-emph-mod.
emph-adv := nex-or-final-adv & emph-mod.

prep := head.
final-prep := prep & strictly-final-mod.
```

Figure 68: TDL definition of the types below *adv* and *prep*

The top type of the sub-hierarchy of distributional types is the type *mod* (modifier). It inherits only from the type *head* and it has no specific constraints on its own.

The type *nex-or-final-mod* only inherits from the type *mod* and it has also no constraints on its own. It is a type encompassing all distributional possibilities an adjunct can have in a sentence, as the types specified to be able to stand in the pre- and post-subject positions in the nexus field, at sentence end and fronted all are subtypes of this type. A sentence adverbial (or free adverbial) that can stand in every (adjunct) position of the clause, such as *allikevel*, inherits directly from this type.

Instances of its subtypes, *nex-mod* and *final-mod*, can only be placed in the nexus field or at sentence end, respectively, i.e. most sentence adverbials implemented in the grammar are instances of one of the subtypes of *nex-mod.*

The type *nex-mod* is specified to modify an element that has NUC value +, as shown in figure 69. This means that it can not modify an element that has already had a complement removed from its COMPS list, thus blocking adjunction 'inside' of (what corresponds to) the traditional VP.

```
nex-mod :=  nex-or-final-mod &
[ MOD < [ LOCAL.CAT.NUC + ] > ].
non-emph-mod := nex-mod.
emph-mod := nex-mod.
```

Figure 69: TDL definition of the types below *nex-mod*

The subtypes of *nex-mod*, *non-emph-mod* and *emph-mod*, are specified with regard to whether the type can be fronted or not. As a general rule, all adjuncts that can be emphasized can also be fronted, hence the name *emph-mod* is given to the type that can be fronted (such as *kanskje*), whereas instances of the type *non-emph-mod* can not be fronted (such as *sikkert*). Other than their relative position in the type hierary, they have no constraints of their own.

The type *final-mod* also inherits from the second subtype of *mod*, the type

final-or-strictly-final-mod, defined as shown in figure 70. The type *final-or-strictly-final-mod* is specified to modify an element with an empty COMPS list, as it can only stand at sentence end. The two other specifications, that the SUBJ list must be of type *olist* and that the modified element can not have a message value, are consequenses of the decision always to combine the subject and the finite verb first, before applying a head-complement phrase. The message specification is then needed to avoid modification of maximal projections, i.e. a whole clause.

```
final-or-strictly-final-mod := mod &
[ MOD < [ LOCAL [ CAT.VAL [ SUBJ olist,
                            COMPS <> ],
                  KEYS.MESSAGE 0-dlist ]] > ].
```

Figure 70: TDL definition of the types below *final-or-strictly-final-mod*

As shown in section 8.1, two different rules are needed to handle post-head modification by sentence-modifying adjuncts in main clause structures. Both are head-initial, but where the phrase type that applies to nexus adverbials is defined to have a non-head daughter with the HEAD value *nex-mod*, the phrase type handling modification at sentence end is defined to have a non-head daughter with the HEAD value *final-or-strictly-final-mod*.

Intuitively, it might seem that another way of setting up the adjunction rules wold be to assume just one 'nexus or final' head-adjunct construction and constraint it to be [MC +] (because of the adjuncts placed after the finite verb in the nexus field). Given the type hierarchy in figure 67, such a unified rule could require its non-head daughter's HEAD to be of type *nex-or-final-mod* and apply to both nexus adverbials and adverbials at the end of the sentence.

However, actual implementation will sometimes serve to prove naïve intutions wrong and reveals that the strategy sketched above would result in overgeneration. The lexicon, as mentioned earlier, contains adverbs defined to be of the same underspecified type (i.e. *nex-or-final-mod*) as well. If the *s-main-hadj-int-phrase* applies directly to an element with a HEAD value of type *nex-or-final-mod*, wrong parses would result from the underspecification, as crucial information needed to rule out ungrammatical adjunct placement is only supplied by the subtypes of *nex-or-final-mod*. Underspecification of the feature NUC (whose specified value is defined in *nex-mod*) can for instance lead to structures where adjuncts interleave the complements. The division into two different rules thus ensures that only the adequately constrained types are used in combination with the head-adjunct phrases.

As shown in section 8.1, this division also makes it possible to use a single phrase type for modification at sentence end, only contrained to have a non-head daughter with the HEAD value *final-or-strictly-final-mod*. If one head-initial phrase should apply to all head-adjunct constructions in the main clause structure, this phrase would, as mentioned, have to be marked

as [MC +]. Adjuncts at the end of a sentence in subordinate clause structures would then have to be accounted for by their own phrase type, in addition to the head-final phrase type needed to account for the adjuncts placed in the nexus field before the finite verb in subordinate clause structures.

This division between the two subtypes of *final-or-strictly-final-mod*, *final-mod* and *strictly-final-mod*, is made for technical reasons, in order to part adjuncts that can only be placed at sentence end (*strictly-final-mod*) from adjuncts that can be placed in the nexus field as well (*final-mod*) as at sentence end. This means that no adjunct is defined as an instance of the type *final-mod* directly, as an adjunct must be defined as an instance of *strictly-final-mod* if it can only stand at sentence end (in this grammar for instance prepositional phrases such as *ved bordet*). Instances of the type *final-mod* inherit from the type *nex-or-final-mod* and are specified to the type *final-mod* during phrase application.

```
strictly-final-mod := final-or-strictly-final-mod.
final-mod := nex-or-final-mod &
             final-or-strictly-final-mod.
```

Figure 71: TDL definition of the types below *final-or-strictly-final-mod*

The reason for this division is to avoid spurious parses by adjunct extraction and fronting, as discussed on page 71 in section 7.4. Fronting is a possibility for all adjuncts that can be placed at sentence end[34], so adjuncts of the type *final-mod* should have the ability to be extracted. But as some adjuncts, as mentioned earlier, are instances of the type *nex-or-final-mod*, they can correctly unify to instances of both *emph-mod* and *final-mod*.

This makes the implementation of the extraction rule a bit tricky. A *nex-or-final-mod* element can not be extracted directly, as this would include *non-emph-mod* elements. Making one extraction rule each for *emph-mod* and *final-mod* would be a solution, but it would result in spurious parses, as some elements, as mentioned, can unify to instances of both types. To define one common supertype for *final-mod* and *emph-mod*, excluding *non-emph-mod*, would enable us to use one rule only, but would still leave us with spurious parses. In this case the supertype unifies to the type *emph-mod* when the adjunct is extracted from the nexus field in one possible parse of the clause, and then unifies to the type *final-mod* when the adjunct is extracted at sentence end in the second possible parse of the clause.

Finally, deciding that only elements of the type *emph-mod* can be extracted would exclude adjuncts that can only stand at sentence end, like prepositional phrases in this grammar, and the other way around the adjuncts that are only defined as *emph-mod* would be excluded.

The solution chosen here is to introduce the type *final-or-strictly-final-mod*, which has the mentioned subtypes *strictly-final-mod* and *final-mod*,

[34]Not including extraposed adjuncts and sentence adverbs that can end up at sentence end because of lacking objects.

also inheriting from *nex-or-final-mod*. Using this classification, the problem of extracting adjuncts can be solved by using the two adjunct-extraction phrases shown in section 8.1.

The phrase type *adv-extracted-adj-phrase* is therefore specified to have a non-head daughter with a HEAD value of type *emph-mod*, as adjuncts of the type *non-emph-mod* can not be fronted. Adjuncts specified as *nex-or-final-mod* can be fronted, and unify to the subtype *emph-mod*.

To cover the cases of extraction of adjuncts from the final position of a sentence, the phrase type *adv-fin-extracted-adj-phrase* is specified to have a non-head daughter with a HEAD value of type *strictly-final-mod* (i.e. adjuncts that can only be placed at sentence end and not in the nexus field as well). To avoid spurious parses, elements of the type *final-mod* can not be extracted from this position, as any element that unifies to *final-mod* also unifies to the type *emph-mod*, to which the phrase type *adv-extracted-adj-phrase* already applies.

This hierarchy does not exhaust the possible placement pattern for adjuncts. As mentioned in section 2.1.5, especially the precise distribution of free adverbials can vary considerably according to grammatical category and semantic contribution. The free adverbials implemented in this grammar were selected because they cover the full range of distributional possibilities, in addition to being of the grammatical category adverb. In spite of this, it would not be a problem to specify additional distributional types for adjuncts, e.g. one that can only be placed after the subject in the nexus field or only before the subject in the nexus field, or any other combination.

The phenomenon of bound adverbials is implemented in the grammar, but this has no impact on the implementation of the *mod*-hierarchy. A bound adverbial is simply any kind of adjunct that is an oblique complement of a verb, and the interesting implementaion is thus done for the verb's lexical type, specifying which type of adjunct the type of verb requires.

8.3 Additional grammar features

This section briefly presents some of the features of the implemented grammar fragment that have not been mentioned in the main sections describing the grammar. This is because although they do provide a necessary basis for working with word order phenomena, the main goal of the grammar implementation, they do not directly influence the solutions that have been chosen in this field.

The main parts of the grammar presented here include lexical types (i.e. types used for building lexical elements), inflectional types and constructional types. By the lexical types, the hierarchy of adjuncts has already been described in detail, as this is important for the implementation of word order phenomena. Most of the constructional types have already been described in section 8.1. The very few types left for this section were omitted because they construct entities other than clauses or because they handle clause constructions that were not on the to-do list of this thesis.

8.3.1 Lexical types

The lexical types implemented in the grammar are in many cases copied from the LinGO ERG grammar and adjusted for use with the Norwegian grammar.

Nominal types included in the grammar are lexical types for common nouns and for the lexical NPs proper names and pronouns (which are marked for case). Common nouns are defined as lexemes, and get their feature value for number (singular of plural) via inflectional rules.

One lexical type of adjective is defined, with an element with HEAD value of type *noun* on its MOD list. All adjectives are pre-head modifiers, marked as POSTHEAD -. The adjectives are defined as lexemes, which get inflected with regard to number, person, gender and definiteness via a inflectional rule, to ensure agreement with the noun.

Definite and indefinite types of determiners function as specifiers in the noun phrases. The subtypes of the determiners also vary with regard to gender, person and number, as these values need to agree with the corresponding values of the noun.

To build the lexical types for verbs, two sub-hierarcies are put to use. A hierarchy of linking types specify the relationship between valence positions and semantic roles. Another hierarchy of valence types specify the number and type of subcategorized complements. The different lexical verb types thus inherit from the suitable subtypes from both hierarcies. The verbs are also defined as lexemes, so tense and verb form are introduced by inflectional rules.

The lexical verb types include types for atransitive verbs, intransitive verbs, transitive verbs with a NP complement, ditransitive verbs, intranitive verbs with a PP complement, transitive verbs with a PP complement and intransitive verbs with a CP complement. One type of auxiliary verb is also defined.

Only transitive prepositions is defined in the grammar. This type has an element with HEAD value of type *verb-or-noun* on its MOD list, i.e. it is able to modify both noun and verb phrases. Prepositional phrases are posthead modifiers only, and they are so marked POSTHEAD +.

The defined lexical type of adverb can only modify elements with HEAD value of type *verb*, and they can be pre-head as well as post-head modifiers.

Two subtypes of complementizers are defined, one that takes a finite verb phrase as its complement (i.e. a subordinate clause) and one that takes an infinite verb phrase as its complement. Additionally, the type of complementizer that takes a finite verb phrase as its complement has two subtypes, one where the complement can have both main clause structure and subordinate clause structure and one where the complement must have subordinate clause structure.

8.3.2 Inflectional types

The inflectional types in the grammar treat inflection of adjective lexemes, noun lexemes and verb lexemes. Irregular forms are not implemented in this grammar, and of the regular forms only the main paradigmes are implemented. This to ensure a maximum of coverage with a minimum of work.

The adjectives are inflected with regard to number, gender and definiteness. The nouns are inflected with regard to number and definiteness, and feminin nouns can be inflected like masculine nouns in addition to the feminin-specific inflection, as in the rule shown in figure 72.

```
noun-mof-pl_irule :=
%suffix  (e er) (er ere) (!er !erer) (!vm !vmmer) (!tm !tmer)
  (!k !ker)
pl-noun-word-infl-rule &
[ ARGS < masc-or-fem-noun-lxm >,
  SYNSEM.LOCAL.AGR.PNG.DEF indef ].
```

Figure 72: Inflectional rule for masculine and feminine noun lexemes

The inflected forms of the verbs reflect tense, and they include infinitive, presens, past and past participle forms. The fact that there are four different main inflectional patterns for regular verbs in Norwegian is complicating matters slightly. Four parallel set of inflectional rules for the past and past participle forms are thus needed to cover for all the regular verbs, and each verb lexeme inherit from a type defining it to belong to one of the three groups. Each inflectional rule then specify which type of verb lexeme it takes as its argument, as shown in figure 73.

```
verb1-pret_irule :=
%suffix (!s !set) (!s !sa)
past-verb-rule &
[ ARGS < verb1-lxm > ].

verb2a-pret_irule :=
%suffix (!s !ste)
past-verb-rule &
[ ARGS < verb2a-lxm > ].
```

Figure 73: Inflectional rules for verb lexemes

8.3.3 Additional constructional types

The *n-hadj-int-phrase* is a subtype of *head-adj-int-phrase*, constraining head daughter of the phrase to have the HEAD value *noun*. Its sibling *n-adjh-int-phrase* is its head-final equivalent. These two phrase types account

for all modification of noun phrases in the grammar including adjectives and prepositional phrases.

The *bare-np-phrase* is a unary grammar rule, inheriting from the types *head-valence-phrase* and *head-only*. The phrase is needed because all common nouns are defined with a specifier, although some lexical forms of the noun can stand alone in a phrase. The phrase type thus removes the specifier when needed.

The head daughter is a lexical element with a HEAD value *noun* and a non-empty specifier list. Because all nouns must be bound by a quantifier, the quantifier is provided through the C-CONT.LISZT value. The INDEX value of the noun is unified with the bound variable of the quantifier relation. In addition to this, a qeq element is introduced on the C-CONT.H-CONS difference list. Here the handle of the quantifier relation is set to take scope over the top handle of the noun.

The phrase needs two subtypes, the *bare-np-def-phrase* and the *bare-np-indef-phrase*. The first phrase type constraints the noun to have the PNG.DEF value *def* and the quantifier relation to be of type *def-rel*. It also constraints the head daughter to have the MODIFIED value *notmod-or-rmod* as a definite noun can not stand without a specifier if it is left-modified.

The *bare-np-indef-phrase* has to have the PNG.DEF value *indef* and it specifies the quantifier relation to be of type *udef-rel*. Additionally PNG.NUM value to be *plur*. It does not lay constraints on modification as does its sibling.

The *det-subj-phrase* is a lexical rule introducing formal subject while transforming the original subject into a complement. This construction is constrained to take certain types of verbs as its arguments, also laying constraints on the original subject of the verb.

In this simplified version, the verb is constrained to be of type *prep-intrans-verb*. This works well, as the grammar only contains prepositions that can be used in this costruction. The subject of the verb must have the PNG.DEF value *indef*. The mother's COMPS list consist of the daughter's COMPS list with the subject element put at the front of the list. When the new element of mother's SUBJ list has an INDEX value of type *det-ind*, which applies to the non-referntial formal subject *det*.

9 Conclusion

To start the conclusion where the introduction ended: the goal of this thesis has been "[...] to develop an analysis for Norwegian that covers all of the above-mentioned phenomena concerning Norwegian word order, and additionally to present an efficient implementation of the analysis in HPSG, using a specialized grammar writing tool". The phenomena mentioned were the varying structures of Norwegian main clauses due to the V2 constraint and topicalization, and the differences in word order between Norwegian main and subordinate clauses, concerning both the ordering of subject, verbs and complements and of adjuncts.

It must be admitted that the learning curve needed to achieve this goal has been steep at times. It has taken considerable time to master the formal frameworks as well as the technical tools. Additionally, the complex nature of the type hierarchy has made it difficult to test out alternative approaches for different phenomena along the way, as the consequences of small changes in one part of the grammar have been deeply felt in completely different grammar parts. But as comparison of different analyses was not an original goal for the thesis, the alternative approaches mentioned in the text can also be seen merely as interesting topics for further studies.

When this is said, though, it must again be mentioned that both the technical tool LKB and the Matrix have made it possible to build a far more advanced grammar than originally intended. In the case of the Matrix, an extra bonus of mutual benefit is added, as my grammar implementation is among its first test beds, building a non-English grammar by the help of the English-derived starter kit. The comparison of Norwegian constructions against their English equivalents thus contributes to the goal of the Matrix, to develop into a language-independent basic type hierarchy. The information resulting from the implementation process, concerning which types can be directly imported into the Norwegian grammar and which types are so language-specific (English) that they need to be changed to be put to use, can be evaluated and later incorporated into the Matrix by its creators.

Not all factors contributing to the the long time span and steep learning curve of this thesis have been imposed on the innocent student by technicians or formal frameworks. It can not be denied that the main and self-inflicted reason has been its broad scope. The work can be divided into no less than four main parts, including: i) a formal descriptive account of the Norwegian phenomena, their classification and delimination, ii) an analysis of the phenomena within the framework of HPSG, iii) a functional implementation of the analysis in LKB and iv) a comprehensive test suite for the phenomena.

To my best knowledge, this thesis presents the first implemented and documented analysis of Norwegian word order phenomena in HPSG. Though the inspiration found in the formal framework of Field Grammar is shared with other contributors to the treatmeant of Scandinavian clause structure in HPSG, the advantage of the analysis is its (relative) simplicity. A minimal use of phrase marking via the features MC and NUC has shown to be sufficient

to account for the main troublesome constructions in the grammar.

The basic differences in the clause structure of main and subordinate clauses are accounted for by postulating two basically different construction patterns, normally separated already from the first phrase application. The sentence adverbials, that on first glance seemed like a rather unruly class when it came to constituent ordering, have shown themselves as well suited for implementation and underspecification in a type hierarchy. When combined, these solutions fulfill the goal of the thesis, as they supply an implementable analysis that accounts for the all chosen language data.

My implemented grammar fragment is supplied on CD-ROM as a 'virtual' appendix to this thesis. The implementation comprises a type hierarchy of 360 types (in addition to the 219 supplied by the Matrix), 17 constructions and 22 lexical and inflectional rules, 84 lexical entries (typically with only a few examples per lexical class), and 2440 lines of TDL code (in addition to the 1398 lines of code for the Matrix). My development and test corpus included as appendix C contains 388 sentences, of which 150 are ungrammatical to test for overgeneration in the grammar (of which there is none).

Maybe one of the most tempting open question that remains is to see how easily the analysis can be expanded to handle object shift as well. It should not be a problem to use the already present features MC and NUC to supply a slightly more advanced marking of the head-complement, head-subject and head adjunct phrases, in combination with a distinct type for at least light pronomina, so that the interaction between the light pronomina and sentence adverbials in the nexus field (i.e. in a NUC + phrase) can be accounted for. It would be extremely interesting to see how advanced the mark-up would have to be to allow for the grammatical object shift constructions while still blocking the ungrammatical ones. However, this question can not be solved in this conclusion and must be left as an exercise to the reader.

A Parse tree examples

This appendix presents the parse trees for some possible constructions in the grammar fragment. The trees are all saved directly from the LKB, exactly as they are displayed in the program. The parse results for the full test set of sentences can be found in appendix C.

A.1 Parse trees from section 7

All the parse trees depicted in section 7 are included in this section, in addition to a few parse trees for possible constructions described but not depicted[35].

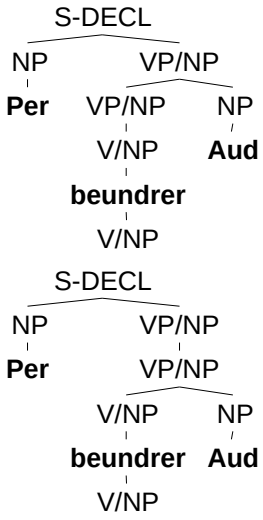

Figure 74: Declarative main clause structures

Figure 74 displays the two resulting parse trees from a declarative main clause with a transitive verb. The first tree corresponds to the default reading of this sentence, where the subject is placed in the sentence initial position. In the second tree, the direct object is extracted from the structure and topicalized.

Figure 75 shows a V1 version of the main clause shown in figure 74. The *yes-no-int-phrase* is applied to recognize and mark the clause as an interrogative clause.

Figure 76 shows the structure of a subordinate clause, here as the complement of the sentence's main verb *sjekket*.

Figure 77 shows the two resulting parse trees when a light sentence adverbial is placed directly after the finite verb. In this case, the element after the

[35]Due to a font problem with the LKB, discovered a bit to late to be fixed, parse trees containing the name *Tranmæl* can not be saved to file. For this reason, other proper names replace the name *Tranmæl* in some examples.

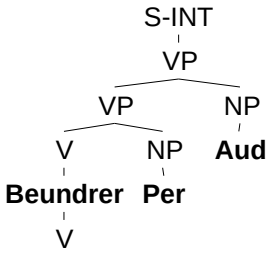

Figure 75: Interrogative main clause structures

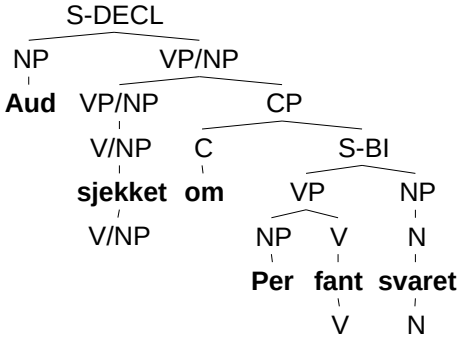

Figure 76: Declarative main clause with subordinate clause complement

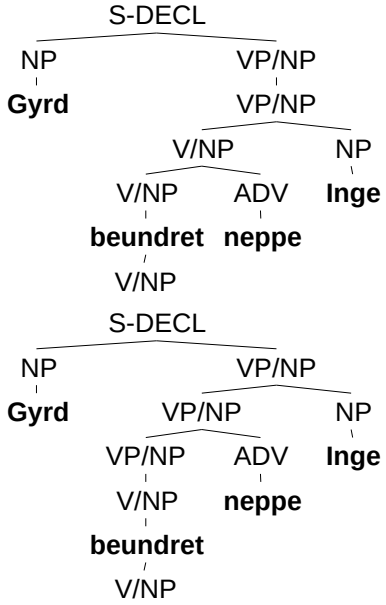

Figure 77: Main clause with light sentence adverbials

adverbial can be the subject as well as the complement of the verb. The first tree has a topicalized object, while the second tree shows the non-stressed reading with the subject at the front of the sentence.

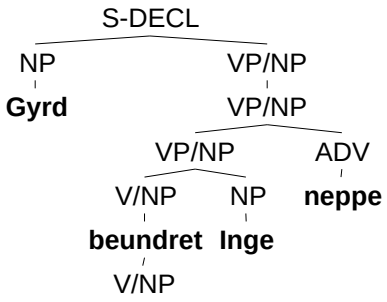

Figure 78: Main clause with light sentence adverbials and topicalized object

In figure 78, the parse result of the sentence is one parse tree only, bedcause an NP element is placed between the finite verb and the light sentence adverbial. As the light and heavy sentence adverbials can only be placed in two positions in the nexus field, directly after the finite verb or after the inverted subject, the NP element must be the subject and the topicalized element must be the object.

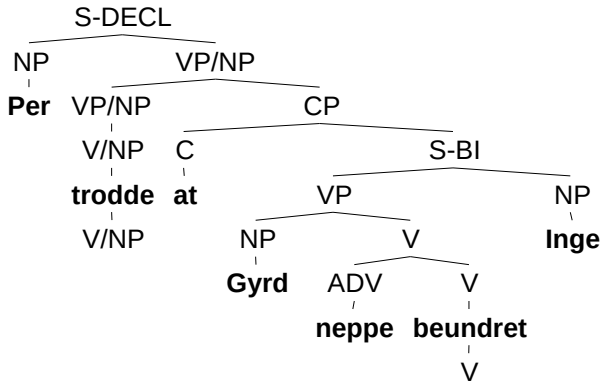

Figure 79: Subordinate clause with light sentence adverbial

Figure 79 shows the pre-head position for light and heavy sentence adverbials in subordinate clause structures.

Figure 80 shows a main clause modified by a sentence adverbial that can also be placed at the final position of a sentence, i.e. after the complements.

In figure 81, this adverbial is topicalized. Note that it is then only extracted from the position after the inverted subject, before the complement.

Figure 82 shows a modified main clause where the adverbial is a prepo-

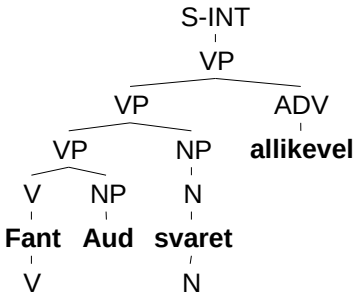

Figure 80: Main clause with *nex-or-final-mod* sentence adverbial

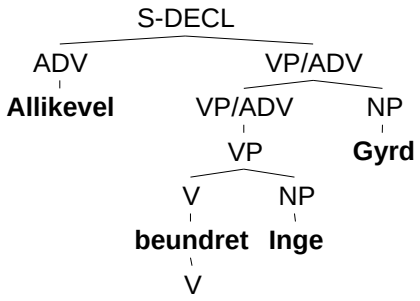

Figure 81: Main clause with topicalized *nex-or-final-mod* sentence adverbial

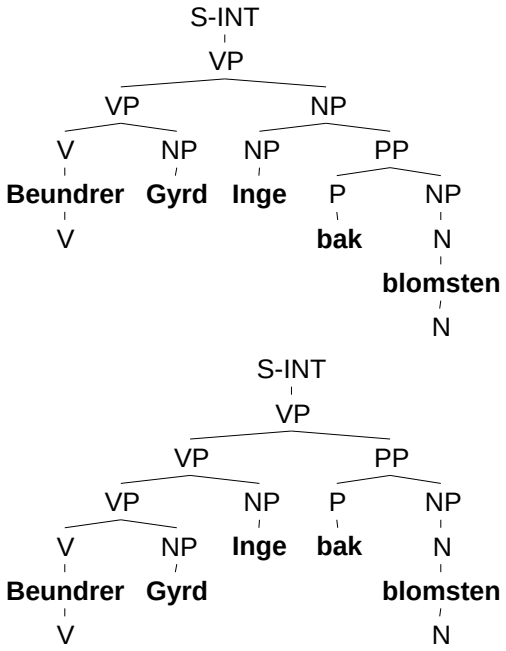

Figure 82: Main clause with PP modifier

sitional phrase. The PPs that are implemented in this grammar are of type *strictly-final-mod* and can so (in addition to being topicalized) only be placed at sentence end if they are modifying the finite verb of the sentence. As the first parse tree shows, they can also modify NPs.

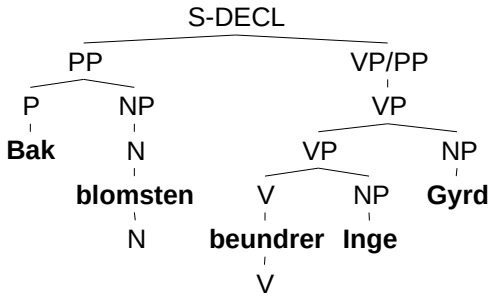

Figure 83: Main clause with topicalized PP modifier

In figure 83, the PP is topicalized. As it can only be placed at the end of the sentence, it is also extracted from this position.

A.2 Additional features

As already mentioned, the exhaustive parse results of the grammar test set of sentences is found in appendix C, logically sorted after subject. The following parse trees are only meant to give a glimpse of some of the features of the grammar that are not described in section 7, in a more visual way than appendix C allows for[36].

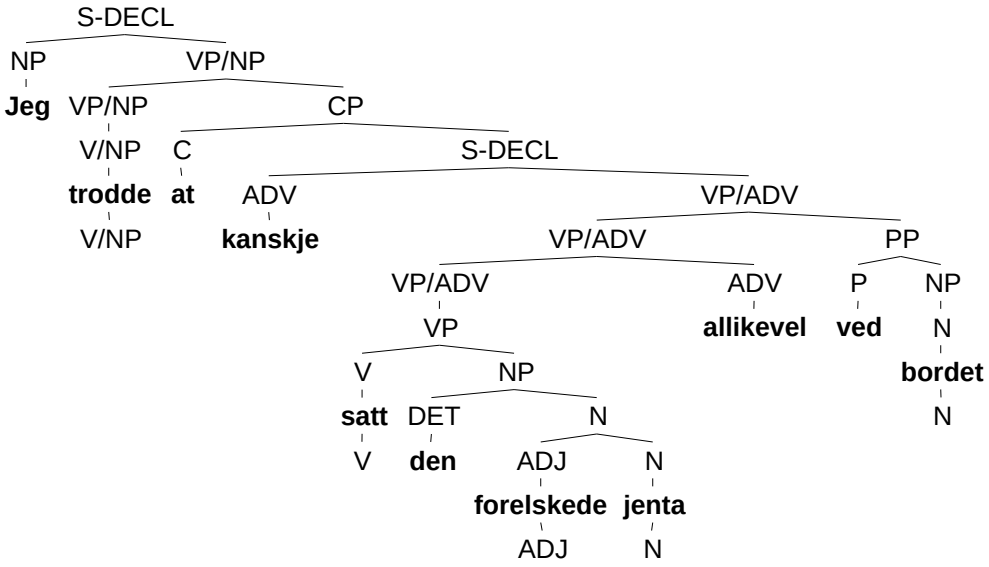

Figure 84: Main clause with main clause complement

Figure 84 and 85 show a important distinction by the use of the complementizer *at*. In contrast to the complementizer *om* that can only have a subordinate clause structure as its complement, *at* can choose between a main and a subordinate clause structure complement. The embedded sentence in figure 84 displays all the signs of a main clause structure, with an inverted subject, a topicalized heavy sentence adverbial and a sentence adverbial placed before the complement, after the finite verb and the inverted subject. In the embedded sentence in figure 85, on the other hand, a light sentence adverbial is placed before the finite verb, after the subject, thus marking the clause as having suborinate clause structure. In figure 85, the subordinate clause is topicalized.

Figure 84 additionally contains a verb that takes a locative PP complement, namely *satt* (which is also a non-regular verb, with base form *sitte*). The PP is not treated as an adjunct, but is combined with the verb phrase via the head-complement rule.

[36]I must confess that the sentences suffer a bit under the attempt at combining far to many phenomena into one structure, and that they for this reason might not be the most natural-sounding utterances under most circumstances. Still, their structural grammati-

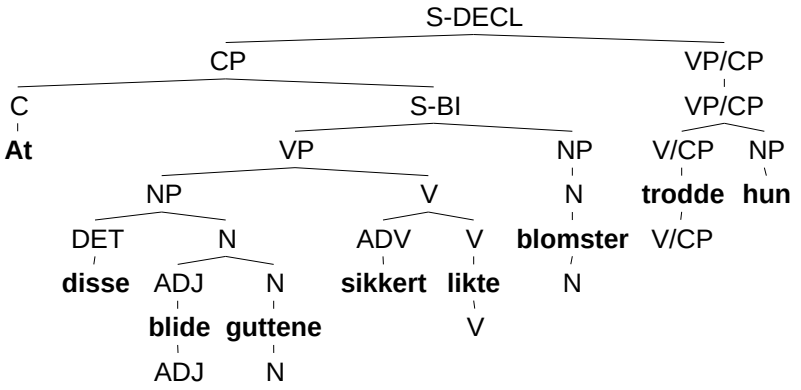

Figure 85: Main clause with subordinate clause complement

Figure 84 and 85 both include varying types of NPs. The simplest form is found in *blomster* (*flowers*) and *bordet* (*the-table*), both common nouns that has been marked for definiteness and number via an inflection rule and that can function as an NP on their own. The bare-np-rule has been applied to both nouns to remove their specifier and so recognize them as NPs.

The more complex noun phrases, *den forelskede jenta* (*the amorous girl*) and *disse blide guttene* (*these cheerful boys*), consist of a common noun, an adjective and a determiner. The common noun has again been marked for definiteness and number via an inflection rule. The adjective modifies the noun and has been inflected with regard to definiteness, number and gender. These feature values have to agree with the corresponding values of the noun, both by the adjective and by the determiner that functions as the specifier of the noun.

Finally, the personal pronouns *jeg* (*I*) and *hun* (*she*) can function as NPs directly. They are both marked for case.

Figure 86 shows a structure where the complement *henne* (*her*) from the subordinate clause has been extracted and topicalized in the main clause. It also includes a construction with the auxiliary verb *hadde* (*had*) and shows that more than one adjunct can be placed in the same adjunct position. All verbs used in the figures has been inflected with regard to tense and verb form. The varying suffixes (for instance *-te* vs. *-et* and *-dde*) are a result of the three different regular patterns for verb inflection in Norwegian.

Figure 87 is included to show the different readings that arise from the possibility of topicalizing complements. In this sentence, the NP *jenten ved bordet* (*the-girl by the-table*) functions as indirect object, direct object and subject, respectively. As a small p.s., the form *jenten* is used instead of the form *jenta* to show that feminine nouns can be inflected by the same rules as masculine nouns, in addition to their feminin-specific inflection.

---

cality can be defended.

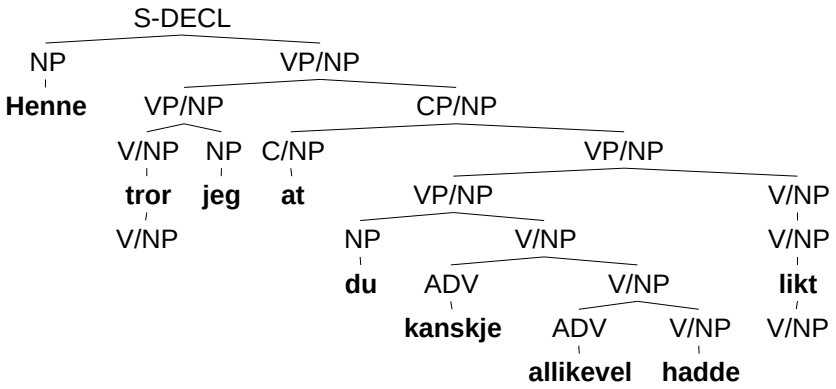

Figure 86: Topicalization of complement from the subordinate clause

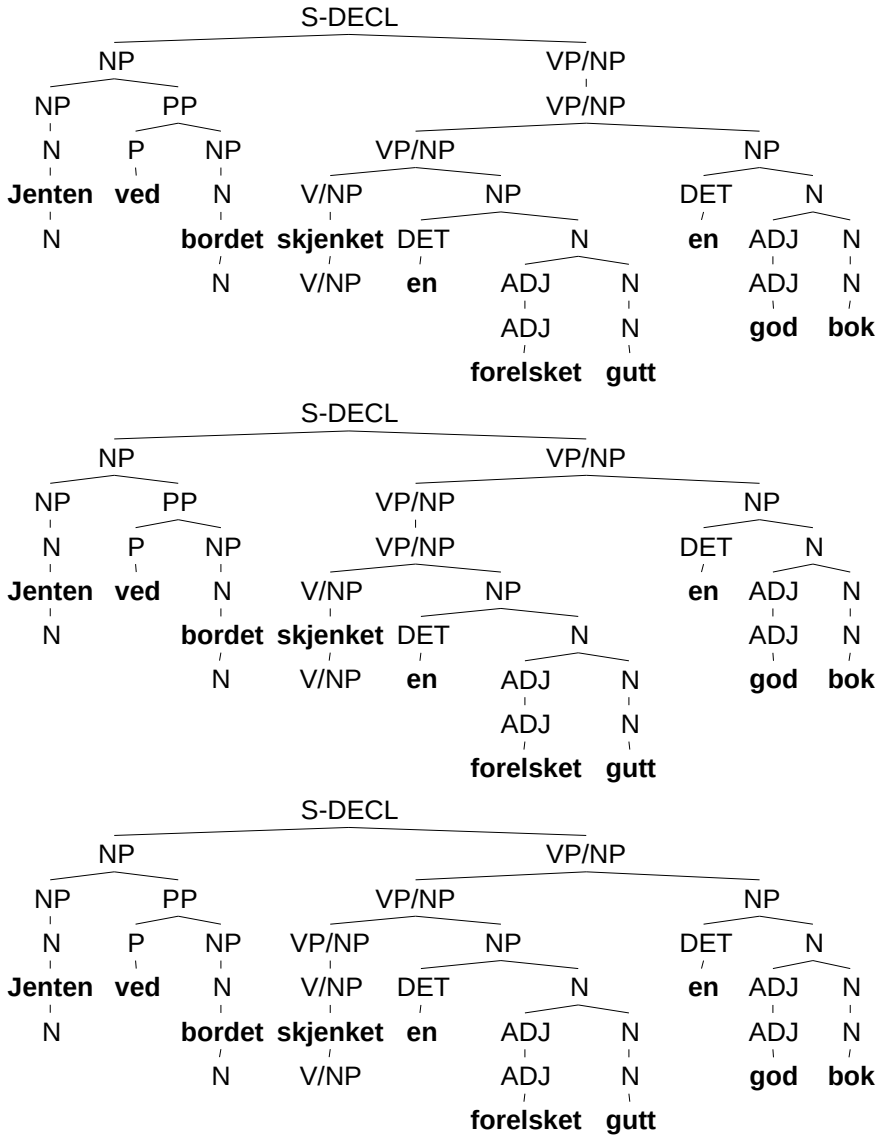

Figure 87: Main clause with ditransitive verb

B LKB Grammar Sources

Following is the file '`norsk.tdl`' from the implemented grammar fragment. The file contains all type definitions for the Norwegian grammar, in many cases using basic types supplied by the Matrix (which is not included here). The actual inventory of rules (instantiating phrase types) and lexical entries (instantiating lexical types) is included towards the end of this section.

```
;;; -*- Mode: tdl; Package: lkb -*-


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; sorts
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

pers := sort.
1st := pers.
2nd := pers.
3rd := pers.

num := sort.
sing := num.
mass := sing.
countsg := sing.
plur := num.

gender := sort.
masc-or-fem := gender.
masc := masc-or-fem.
fem := masc-or-fem.
neut := gender.

definiteness := sort.
indef := definiteness.
def := definiteness.

case := sort.
nom := case.
acc := case.

indicative := mood.

active := aspect.
passive := aspect.

norsk-png := png &
[ PERS pers,
  NUM num,
  GEND gender,
  DEF definiteness ].

present := tense.
past := tense.
presperf := tense.
pastperf := tense.
future := tense.

vform := sort.
```

```
fin := vform.
inf-or-psp := vform.
psp := inf-or-psp.
inf-or-prp := vform.
inf := inf-or-prp & inf-or-psp.
prp := inf-or-prp.
pas := vform.
imp := vform.


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; head types
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

det := head.

nominal :=  head &
[ CASE case ].

verb-or-noun := head.

noun := nominal & verb-or-noun.
noun-nom := noun &
[ CASE nom ].
noun-acc := noun &
[ CASE acc ].

adverbee := head.

adj :=  adverbee.

verb-or-comp := head &
[ VFORM vform ].

verbal := verb-or-comp &
[ AUX bool ].
verb := verb-or-noun & verbal & adverbee.

comp := verbal.
at-comp := comp.
om-comp := comp.

mod := head.
final-or-strictly-final-mod := mod &
[ MOD < [ LOCAL [ CAT.VAL [ SUBJ olist,
                            COMPS <> ],
  KEYS.MESSAGE 0-dlist ]] > ].
strictly-final-mod := final-or-strictly-final-mod.
nex-or-final-mod :=  mod.
nex-mod :=  nex-or-final-mod &
[ MOD < [ LOCAL.CAT.NUC + ] > ].
final-mod := nex-or-final-mod &  final-or-strictly-final-mod.
non-emph-mod := nex-mod.
emph-mod := nex-mod.

adv :=  adverbee.
nex-or-final-adv := adv & nex-or-final-mod.
final-adv := nex-or-final-adv & final-mod.
non-emph-adv := adv & non-emph-mod.
emph-adv := nex-or-final-adv  & emph-mod.
```

```
prep := head.
final-prep := prep & strictly-final-mod.


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; cat
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

nomp-cat-min := cat &
[ HEAD nominal,
  VAL [ SUBJ olist,
        SPR olist,
        COMPS olist ] ].

nomp-cat-acc-min := nomp-cat-min &
[ HEAD.CASE acc ].
nomp-cat-nom-min := nomp-cat-min &
[ HEAD.CASE nom ].

np-cat-nom-min := nomp-cat-nom-min &
[ HEAD noun ].

np-cat-acc-min := nomp-cat-acc-min &
[ HEAD noun ].


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; noun synsems
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

basic-nom-synsem := lex-synsem &
[ LOCAL [ CONT nom-obj,
          CAT.HEAD nominal ]].

nominal-synsem := basic-nom-synsem.

ref-synsem := nominal-synsem &
[ LOCAL [ CONT [ LISZT.LIST.FIRST basic-nom-rel & #rel &
                                  [ INST #ind & ref-ind ],
                 INDEX #ind ],
          KEYS.KEY #rel ]].

noun-synsem := ref-synsem & nonpronominal-synsem &
[ LOCAL [ CONT.INDEX #ind,
          AGR #ind,
          CAT.HEAD noun ],
  NON-LOCAL.SLASH 0-dlist ].

nomod-local := local &
[ CAT.HEAD.MOD < > ].

nomod-synsem := lex-synsem &
[ LOCAL.CAT [ HEAD.MOD <>,
              VAL.SUBJ <> ]].

mod-synsem := lex-synsem &
[ LOCAL.CAT [ HEAD.MOD < synsem >]].
```

```
pronominal-synsem := nominal-synsem &
[ LOCAL [ CAT [ HEAD noun &
                     [ MOD < > ],
                VAL.SUBJ < > ] ] ].

nonpronominal-synsem := nominal-synsem.

non-ref-synsem := nominal-synsem  & nomod-synsem & zero-arg &
[ LOCAL.CAT.VAL [ SPR < >,
                  COMPS < > ] ].

ref-pro-synsem := pronominal-synsem & ref-synsem & nomod-synsem.

non-ref-pro-synsem := pronominal-synsem & non-ref-synsem &
[ LOCAL [ AGR #agr,
          CONT.INDEX #agr ] ].

common-noun-synsem := noun-synsem &
[ LOCAL [ CAT.VAL [ COMPS #rest,
                    SPR < synsem & #first &
                          [ OPT -,
                            LOCAL [ KEYS.KEY.BV #bv,
                                    CAT [ VAL [ SUBJ null,
                                                SPR olist,
                                                COMPS olist ],
                                          HEAD det ]]] > ],
          ARG-S [ FIRST #first,
                  REST #rest ],
          CONT  [ TOP #top,
                  INDEX #bv,
                  LISZT <! nom-rel & [ HNDL #top ]!>,
                  H-CONS <! !>]]].

common-noun-nocomp-synsem := common-noun-synsem & nomod-synsem &
[ LOCAL.CAT.VAL.COMPS <> ].

nonpro-nomod-synsem := nomod-synsem & nonpronominal-synsem.
nomod-basic-onearg-synsem := nomod-synsem & basic-one-arg.
nomod-onearg-synsem := nomod-synsem & one-arg.
nomod-zero-arg-synsem := nomod-synsem & basic-zero-arg.
nonpro-nomod-onearg-synsem := nonpro-nomod-synsem & basic-one-arg.

noun-nocomp-synsem := common-noun-nocomp-synsem & nonpro-nomod-onearg-synsem.

mass-noun-synsem := noun-nocomp-synsem &
[ LOCAL.CONT [ INDEX [ PNG norsk-png,
                       DIVISIBLE + ],
               LISZT <! relation !> ]].


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; lexical NPs
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

np-word := nontopkey &
[ SYNSEM ref-synsem & [ LOCAL.CAT [ HEAD noun,
                                   VAL [ SPR < >,
                                         SUBJ < >,
                                         COMPS < > ] ] ] ].
```

```
np-synsem := noun-synsem & nomod-synsem.

basic-np-sing-word := np-word &
[ SYNSEM.LOCAL [ KEYS.KEY.HNDL #keyhand,
                 CONT [ INDEX.PNG norsk-png   & [ PERS 3rd, NUM sing ],
                        LISZT <! basic-nom-rel &
                                 [ INST #ind ],
                                 quant-rel &
                                 [ BV #ind,
                                   RESTR #rhand ] !>,
                        H-CONS <! qeq & [ SC-ARG #rhand,
                                          OUTSCPD #keyhand ] !> ] ] ].

np-sing-word := basic-np-sing-word &
[ SYNSEM np-synsem &
         [ LOCAL.CONT.LISZT <! relation, def-np-rel !> ] ].

proper-name-sg := np-sing-word &
[ SYNSEM.LOCAL [ KEYS.KEY named-rel ]].

personal-pro := np-word &
[ SYNSEM ref-pro-synsem  &
         [ LOCAL [ KEYS.KEY.HNDL #hand,
                   CONT [ TOP #hand,
                          LISZT <! pron-rel !>,
                          H-CONS <! !> ] ] ] ].

personal-pro-sg := personal-pro &
[ SYNSEM.LOCAL.CONT.INDEX.PNG.NUM sing ].
personal-pro-pl := personal-pro &
[ SYNSEM.LOCAL.CONT.INDEX.PNG.NUM plur ].
personal-pro-nom := personal-pro &
[ SYNSEM.LOCAL.CAT.HEAD.CASE nom ].
personal-pro-acc := personal-pro &
[ SYNSEM.LOCAL.CAT.HEAD.CASE acc ].
personal-pro-1st := personal-pro &
[ SYNSEM.LOCAL.CONT.INDEX.PNG.PERS 1st ].
personal-pro-2nd := personal-pro &
[ SYNSEM.LOCAL.CONT.INDEX.PNG.PERS 2nd ].
personal-pro-3rd := personal-pro &
[ SYNSEM.LOCAL.CONT.INDEX.PNG.PERS 3rd ].
personal-pro-fem := personal-pro &
[ SYNSEM.LOCAL.CONT.INDEX.PNG.GEND fem ].
personal-pro-masc := personal-pro &
[ SYNSEM.LOCAL.CONT.INDEX.PNG.GEND masc ].
personal-pro-neut := personal-pro &
[ SYNSEM.LOCAL.CONT.INDEX.PNG.GEND neut ].

perspron-3rd-sg := personal-pro-sg &  personal-pro-3rd.

perspron-1st-sg-nom := personal-pro-sg & personal-pro-nom & personal-pro-1st.
perspron-2nd-sg-nom := personal-pro-sg & personal-pro-nom & personal-pro-2nd.
perspron-3rd-sg-nom := perspron-3rd-sg & personal-pro-nom.

perspron-1st-sg-acc := personal-pro-sg & personal-pro-acc & personal-pro-1st.
perspron-2nd-sg-acc := personal-pro-sg & personal-pro-acc & personal-pro-2nd.
perspron-3rd-sg-acc := perspron-3rd-sg & personal-pro-acc.

perspron-1st-pl-nom := personal-pro-sg & personal-pro-nom & personal-pro-1st.
perspron-2nd-pl := personal-pro-sg  & personal-pro-2nd.
perspron-3rd-pl := personal-pro-sg & personal-pro-3rd.
```

```
perspron-1st-pl-acc := personal-pro-sg & personal-pro-acc & personal-pro-1st.
perspron-3rd-pl-acc := personal-pro-sg & personal-pro-acc & personal-pro-3rd.


perspron-3rd-sg-nom-fem := perspron-3rd-sg-nom & personal-pro-fem.
perspron-3rd-sg-nom-masc := perspron-3rd-sg-nom & personal-pro-masc.

perspron-3rd-sg-acc-fem := perspron-3rd-sg-acc & personal-pro-fem.
perspron-3rd-sg-acc-masc := perspron-3rd-sg-acc & personal-pro-masc.

perspron-3rd-sg-neut := perspron-3rd-sg & personal-pro-neut.

expletive-det-word := nontopkey &
[ SYNSEM non-ref-pro-synsem &
         [ LOCAL [ KEYS.KEY no-relation,
                   CONT [ INDEX expl-ind,
                          LISZT <! !> ] ]] ].


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; modifier synsems
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

isect-synsem := canonical-synsem &
[ LOCAL [ CAT.HEAD.MOD < [ LOCAL intersective-mod ] >,
          CONT.TOP #hand,
          KEYS.KEY.HNDL #hand ] ].


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; adjectives
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

basic-adj-synsem := lex-synsem &
[ LOCAL [ CAT [ HEAD adj,
                VAL [ SUBJ < >,
                      SPR < synsem &
                            [ LOCAL local-min &
                                    [ CAT [ VAL [ SPR olist,
                                                  COMPS olist ],
                                            MC na ],
                                      KEYS.KEY degree-rel &
                                               [ HNDL #ahand] ],
                              OPT + ] >,
                      COMPS <> ],
                POSTHEAD - ],
          CONT.LISZT.LIST < basic-adj-rel & #key &
                            [ HNDL #ahand], ... >,
          KEYS.KEY #key ] ].

adj-synsem := basic-adj-synsem & isect-synsem &
[ LOCAL [ CAT [ HEAD.MOD < [ LOCAL [ CAT [ HEAD noun,
                                           VAL [ SUBJ < >,
                                                 SPR < synsem &
                                                       [ LOCAL.CAT.HEAD det ] >,
                                                 COMPS olist ],
                                           MC na] ,
```

```
                                  CONT.INDEX #ind & ref-ind,
                                  AGR #agr ] ] > ],
          CONT [ LISZT <! abstr-adj-rel !>,
                 INDEX #ind ],
          KEYS.KEY abstr-adj-rel & [ ARG #ind ],
          AGR #agr] ].

adj-word := word &
[ SYNSEM adj-synsem &
         [ LOCAL [  CONT.LISZT <! abstr-adj-rel !>]]].


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; determiners
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

basic-det-synsem := nomod-synsem &
[ LOCAL [ CAT [ HEAD det,
                VAL [ SPR < [ LOCAL local-min &
                                    [ CAT.HEAD adv,
                                      KEYS.KEY relation &
                                               [ HNDL #khand]],
                              NON-LOCAL [ QUE 0-dlist,
                                          REL 0-dlist ],
                              OPT + ] >,
                      SPEC < [ LOCAL.CONT.TOP #nhand ] >,
                     COMPS < > ] ],
          CONT nom-obj &
              [ INDEX #index,
                LISZT.LIST < quant-or-wh-rel & #key, ... >,
                H-CONS.LIST < qeq &
                              [ SC-ARG #rhand,
                                OUTSCPD #nhand ], ... > ],
          KEYS.KEY #key & [ HNDL #khand,
                            BV #index,
                            RESTR #rhand],
          ARG-S < > ] ].

det-synsem := basic-det-synsem &
[ LOCAL.CONT [ LISZT <! quant-or-wh-rel !>,
               H-CONS <! qeq !> ] ].

det-word := word &
[ SYNSEM det-synsem ].

det-word-sing := det-word &
[ SYNSEM.LOCAL.KEYS.KEY.BV [ PNG norsk-png & [ PERS 3rd, NUM sing ],
                             DIVISIBLE - ] ].

det-word-fem := det-word &
[ SYNSEM.LOCAL.KEYS.KEY.BV.PNG.GEND fem ].
det-word-masc-or-fem := det-word &
[ SYNSEM.LOCAL.KEYS.KEY.BV.PNG.GEND masc-or-fem ].
det-word-neut := det-word &
[ SYNSEM.LOCAL.KEYS.KEY.BV.PNG.GEND neut ].

det-word-sing-neut := det-word-sing & det-word-neut.
det-word-sing-masc-or-fem := det-word-sing & det-word-masc-or-fem.
det-word-sing-fem := det-word-sing & det-word-fem.
```

```
det-word-sing-indef-fem := det-word-sing-fem & word-indef.
det-word-sing-indef-masc-or-fem := det-word-sing-masc-or-fem & word-indef.
det-word-sing-indef-neut := det-word-sing-neut & word-indef.

det-word-sing-def-masc-or-fem := det-word-sing-masc-or-fem & word-def.
det-word-sing-def-neut := det-word-sing-neut & word-def.

det-word-plur := det-word &
[ SYNSEM.LOCAL.KEYS.KEY.BV.PNG norsk-png & [ PERS 3rd, NUM plur ] ].

det-word-pl-def := det-word-plur & word-def.

det-word-pl-indef := det-word-plur & word-indef.

word-def := word &
[ SYNSEM.LOCAL.KEYS.KEY.BV.PNG.DEF def ].
word-indef := word &
[ SYNSEM.LOCAL.KEYS.KEY.BV.PNG.DEF indef ].


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; index subtypes
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

full-index := index &
[ DIVISIBLE bool ].

full-ref-ind := ref-ind & full-index.

det-ind := expl-ind & full-index.


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; relation types
;;;

basic-nom-rel := norm-relation &
[ INST individual ].

nom-rel := basic-nom-rel &
[ INST ref-ind ].

named-rel := nom-rel.

pron-rel := nom-rel.

quant-or-wh-rel := norm-relation &
[ BV ref-ind,
  RESTR handle ].

quant-rel := quant-or-wh-rel &
[ SCOPE handle ].

number-or-degree-rel := relation.
degree-rel := number-or-degree-rel.
basic-adj-rel := relation.
abstr-adj-rel :=  basic-adj-rel & arg-rel &
[ ARG non-expl ].
adj-rel := abstr-adj-rel.
```

```
support-abstr-rel := event-rel.
support-arg4-rel := support-abstr-rel & arg4-rel.

all-rel := quant-rel.
some-rel := quant-rel.
udef-rel := quant-rel.
def-rel := quant-rel.

def-np-rel := def-rel.

prop-rel := proposition &
[ PRED 'proposition ].

quest-rel := question &
[ PRED 'question ].


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; adjective lexemes
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

adj-lxm := lexeme &
[ SYNSEM adj-synsem ].

adj-word-rule := lex-rule &
[ ARGS < adj-lxm > ].

pl-adj-word-rule := adj-word-rule &
[ SYNSEM.LOCAL.AGR.PNG.NUM plur ].

sg-adj-word-rule := adj-word-rule &
[ SYNSEM.LOCAL.AGR.PNG.NUM sing ].

sg-indef-mof-adj-word-const-rule := const-ltow-rule &
                                    sg-adj-word-rule & sign-indef &
[ SYNSEM.LOCAL.AGR.PNG.GEND masc-or-fem ].

sg-indef-neut-adj-word-infl-rule := infl-ltow-rule &
                                    sg-adj-word-rule & sign-indef &
[ SYNSEM.LOCAL.AGR.PNG.GEND neut ].

pl-adj-word-infl-rule := infl-ltow-rule & pl-adj-word-rule.

sg-def-adj-word-infl-rule := infl-ltow-rule & sg-adj-word-rule & sign-def.

sign-indef := sign &
[ SYNSEM.LOCAL.AGR.PNG.DEF indef ].
sign-def := sign &
[ SYNSEM.LOCAL.AGR.PNG.DEF def ].


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; noun lexemes
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

lexeme := word-or-lexrule &
[ INFLECTED -,
  SYNSEM.LOCAL.KEYS.MESSAGE 0-dlist ].
```

```
noun-lxm := lexeme &
[ SYNSEM common-noun-nocomp-synsem ].

masc-or-fem-noun-lxm := noun-lxm &
[ SYNSEM.LOCAL.AGR.PNG.GEND masc-or-fem ].
neut-noun-lxm := noun-lxm &
[ SYNSEM.LOCAL.AGR.PNG.GEND neut ].
fem-noun-lxm := masc-or-fem-noun-lxm &
[ SYNSEM.LOCAL.AGR.PNG.GEND fem ].
masc-noun-lxm := masc-or-fem-noun-lxm &
[ SYNSEM.LOCAL.AGR.PNG.GEND masc ].

noun-word-rule := lex-rule &
[ ARGS < noun-lxm > ].

pl-noun-word-rule := noun-word-rule &
[ SYNSEM.LOCAL.AGR.PNG.NUM plur ].
sg-noun-word-rule := noun-word-rule &
[ SYNSEM.LOCAL.AGR.PNG.NUM sing ].

sg-noun-word-const-rule := const-ltow-rule & sg-noun-word-rule.
pl-noun-word-const-rule := const-ltow-rule & pl-noun-word-rule.

sg-noun-word-infl-rule := infl-ltow-rule & sg-noun-word-rule.
pl-noun-word-infl-rule := infl-ltow-rule & pl-noun-word-rule.


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; linking types
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

linking-type := lex-synsem.

;; Det snør

atrans-lt := linking-type &
[ LOCAL [ CAT.VAL.SUBJ < [ LOCAL local-min &
                                [ KEYS.KEY.LABEL #label,
                                  CONT.INDEX det-ind ] ] >,
          KEYS.KEY no-relation & [ LABEL #label ] ] ].

arg1-subj-lt := linking-type &
[ LOCAL [ CAT.VAL.SUBJ < [ LOCAL local-min &
                                [ CONT [ TOP #top,
                                         INDEX #subjind & non-expl ]]] >,
          CONT [ TOP #top,
                 LISZT <! #key !> ],
          KEYS.KEY #key & arg1-rel & [ ARG1 #subjind,
                                       HNDL #top ] ] ].

arg13-lt := arg1-subj-lt &
[ LOCAL [ CAT.VAL.COMPS < [ LOCAL local-min &
                                 [ CONT [ TOP #top,
                                          INDEX #objind & non-expl ]] ], ... >,
          CONT.TOP #top,
          KEYS.KEY arg13-rel & [ ARG3 #objind ] ] ].

trans-lt := arg13-lt.
```

```
arg14-lt := arg1-subj-lt &
[ LOCAL [ CAT.VAL.COMPS < [ LOCAL local-min &
                                [ KEYS [ KEY norm-relation,
                                         MESSAGE 1-dlist &
                                             <! [ HNDL #mhand ] !>]]], ... >,
          KEYS.KEY arg14-rel & [ ARG4 #mhand ] ] ].

ditrans-lt := arg1-subj-lt &
[ LOCAL [ CAT.VAL.COMPS < [ LOCAL local-min &
                                [ CONT [ TOP #top,
                                         INDEX #obj2ind & non-expl ]] ],
                          [ LOCAL local-min &
                                [ CONT [ TOP #top,
                                         INDEX #objind & non-expl ] ] ], ... >,
          CONT.TOP #top,
          KEYS.KEY arg123-rel & [ ARG2 #obj2ind,
                                  ARG3 #objind ] ] ].

basic-prep-intrans-lt := linking-type &
[ LOCAL [ CAT.VAL.COMPS < [ LOCAL local-min &
                                [ KEYS.KEY prep-mod-rel &
                                           [ ARG #index ],
                                  CONT.TOP #top ] ],
                          ... >,
          CONT [ TOP #top,
                 INDEX #index,
                 LISZT <! #key !> ],
          KEYS.KEY #key ] ].

prep-intrans-lt := basic-prep-intrans-lt & arg1-subj-lt.

prep-trans-lt := arg13-lt &
[ LOCAL [ CAT.VAL.COMPS < *top*,
                          [ LOCAL local-min &
                                [ CAT.HEAD.MOD < synsem-min >,
                                  CONT.TOP #top,
                                  KEYS.KEY prep-mod-rel &
                                               [ ARG #event ] ] ],
                          ... >,
          CONT [ TOP #top,
                 INDEX #event,
                 LISZT <! #key !> ],
          KEYS.KEY #key ] ].

ssr-lt := linking-type &
[ LOCAL.KEYS.KEY  support-abstr-rel ].

ssr-v-lt := linking-type &
[ LOCAL [ CONT.LISZT <! #key !>,
          KEYS.KEY #key ]].


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; valence types
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

valence-type := lex-synsem.

unsat-subst := valence-type &
[ LOCAL.CAT.VAL.SUBJ < synsem  & [ LOCAL.CAT nomp-cat-nom-min ] > ].
```

```
intrans-subst := unsat-subst & one-arg &
[ LOCAL.CAT.VAL.COMPS < > ].

unsat-two-arg-subst := unsat-subst &
[ LOCAL [ CAT.VAL.COMPS < synsem & [ LOCAL local-min &
                                        [ KEYS.KEY #ckey ] ], ... >,
          KEYS.--COMPKEY #ckey ] ].

two-arg-subst := unsat-two-arg-subst &
[ LOCAL.CAT.VAL.COMPS < *top* > ].

trans-subst := unsat-subst &
[ LOCAL.CAT.VAL.COMPS < synsem & [ LOCAL.CAT np-cat-acc-min ], ... > ].

np-trans-subst := trans-subst & two-arg-subst & two-arg &
[ LOCAL.CAT.VAL.COMPS < synsem > ].

unsat-three-arg-subst := unsat-two-arg-subst &
[ LOCAL [ CAT.VAL.COMPS < synsem,
                          synsem & [ LOCAL local-min &
                                        [ KEYS.KEY #ckey ] ], ... >,
          KEYS.--OCOMPKEY #ckey ] ].

three-arg-subst := unsat-three-arg-subst & three-arg.
basic-three-arg-subst := unsat-three-arg-subst & basic-three-arg.

three-arg-trans-subst := unsat-three-arg-subst & three-arg & trans-subst.
basic-three-arg-trans-subst := unsat-three-arg-subst & basic-three-arg &
       trans-subst.

ditrans-subst := three-arg-trans-subst &
[ LOCAL.CAT.VAL.COMPS < synsem & [ LOCAL [ CAT nomp-cat-acc-min ] ],
                        synsem & [ LOCAL [ CAT nomp-cat-acc-min ] ] > ].

cp-intrans-subst := two-arg-subst &
[ LOCAL.CAT.VAL [ --KEYCOMP #comp,
                  COMPS < #comp & synsem &
                          [ LOCAL [ CAT [ HEAD comp & [ VFORM fin ],
                                          VAL [ SUBJ < >,
                                                COMPS <>,
                                                SPR <> ],
                                          MC - ]]]> ] ].


prep-trans-subst := three-arg-trans-subst &
[ LOCAL.CAT.VAL.COMPS < synsem & [ LOCAL.CAT nomp-cat-acc-min ],
                        synsem & [ LOCAL.CAT [ HEAD prep, VAL.COMPS < > ] ] > ].

inf-or-prp-intrans-subst := unsat-two-arg-subst &
[ LOCAL.CAT.VAL [ --KEYCOMP #comp,
                  COMPS < #comp &
                          [ LOCAL local-basic  &
                                  [ CAT [ HEAD verbal &
                                                [ VFORM inf-or-prp ],
                                          VAL [ SUBJ < synsem >,
                                                COMPS olist  ]]]] > ] ].

inf-intrans-subst := inf-or-prp-intrans-subst &
[ LOCAL.CAT.VAL.COMPS < [ LOCAL local-min &
                                [ CAT.HEAD.VFORM inf ] ] > ].
```

```
ssr-subst := two-arg-subst &
[ LOCAL [ CAT.VAL [ SUBJ < [ LOCAL local-min & [ CONT [ TOP #top,
                                                        INDEX #cont ] ] ] >,
                    COMPS < [ LOCAL local-min &
                                    [ CAT.VAL [ SUBJ < [ LOCAL.CONT.INDEX #cont,
                                                         NON-LOCAL [ SLASH 0-dlist,
                                                                     REL 0-dlist,
                                                                     QUE 0-dlist ]]>,
                                                COMPS olist ],
                                      KEYS.MESSAGE 0-dlist ] ] > ],
          CONT.TOP #top ] ].

ssr-two-arg-subst := ssr-subst & basic-two-arg.

ssr-inf-subst := ssr-subst & inf-intrans-subst.


prep-intrans-subst := two-arg-subst &
[ LOCAL.CAT.VAL.COMPS < synsem & [ LOCAL.CAT [ HEAD prep,
                                               VAL.COMPS < > ]] > ].


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; verb synsems
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

verb-synsem := lex-synsem &
[ LOCAL [ CAT [ HEAD verb,
                VAL [ SUBJ < synsem & #subj >,
                      COMPS #comps ,
                      SPR < anti-synsem >]],
          ARG-S < #subj . #comps > ],
  NON-LOCAL [ QUE 0-dlist,
              REL 0-dlist ] ].

atrans-verb := verb-synsem & intrans-subst & atrans-lt.

unerg-verb := verb-synsem & intrans-subst & arg1-subj-lt.

np-trans-verb := verb-synsem & np-trans-subst & trans-lt.

ditrans-verb := verb-synsem & ditrans-subst & ditrans-lt.

basic-prep-intrans-verb := verb-synsem & prep-intrans-subst & two-arg.

prep-intrans-verb := basic-prep-intrans-verb & prep-intrans-lt.

prep-trans-verb := verb-synsem & prep-trans-subst & prep-trans-lt.


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; NON-LOCAL amalgamation types
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

basic-zero-arg := lex-synsem &
[ LOCAL.ARG-S < >,
  NON-LOCAL [ SLASH 0-dlist,
              REL 0-dlist,
```

```
            QUE 0-dlist ] ].

zero-arg := basic-zero-arg &
[ LOCAL.CONT.H-CONS <! !> ].

basic-one-arg := canonical-synsem &
[ LOCAL.ARG-S < [ NON-LOCAL [ SLASH #slash,
                              REL #rel,
                              QUE #que ],
                  LOCAL.CONT.INDEX individual ] >,
  NON-LOCAL [ SLASH #slash,
              REL #rel,
              QUE #que ] ].

one-arg := basic-one-arg &
[ LOCAL.CONT.H-CONS <! !> ].

basic-two-arg := lex-synsem &
[ LOCAL.ARG-S < [ NON-LOCAL [ SLASH [ LIST #smiddle,
                                      LAST #slast ],
                              REL [ LIST #rmiddle,
                                    LAST #rlast ],
                              QUE [ LIST #qmiddle,
                                    LAST #qlast ] ],
                  LOCAL.CONT.INDEX individual ],
                [ NON-LOCAL [ SLASH [ LIST #sfirst,
                                      LAST #smiddle ],
                              REL [ LIST #rfirst,
                                    LAST #rmiddle ],
                              QUE [ LIST #qfirst,
                                    LAST #qmiddle ] ],
                  LOCAL.CONT.INDEX individual ] >,
  NON-LOCAL [ SLASH [ LIST #sfirst,
                      LAST #slast ],
              REL [ LIST #rfirst,
                    LAST #rlast ],
              QUE [ LIST #qfirst,
                    LAST #qlast ] ] ].

two-arg := basic-two-arg &
[ LOCAL.CONT.H-CONS <! !> ].

basic-three-arg := lex-synsem &
[ LOCAL [ ARG-S < [ NON-LOCAL [ SLASH [ LIST #smiddle2,
                                        LAST #slast ],
                                REL [ LIST #rmiddle2,
                                      LAST #rlast ],
                                QUE [ LIST #qmiddle2,
                                      LAST #qlast ] ],
                    LOCAL.CONT.INDEX individual ],
                  [ NON-LOCAL [ SLASH [ LIST #sfirst,
                                        LAST #smiddle1 ],
                                REL [ LIST #rfirst,
                                      LAST #rmiddle1 ],
                                QUE [ LIST #qfirst,
                                      LAST #qmiddle1 ] ],
                    LOCAL.CONT.INDEX individual ],
                  [ NON-LOCAL [ SLASH [ LIST #smiddle1,
                                        LAST #smiddle2 ],
                                REL [ LIST #rmiddle1,
                                      LAST #rmiddle2 ],
                                QUE [ LIST #qmiddle1,
```

```
                                    LAST #qmiddle2 ] ],
                    LOCAL.CONT.INDEX individual ] > ],
  NON-LOCAL [ SLASH [ LIST #sfirst,
                      LAST #slast ],
              REL [ LIST #rfirst,
                    LAST #rlast ],
              QUE [ LIST #qfirst,
                    LAST #qlast ] ] ].

three-arg := basic-three-arg &
[ LOCAL.CONT.H-CONS <! !> ].

four-arg := lex-synsem &
[ LOCAL [ ARG-S < [ NON-LOCAL [ SLASH [ LIST #smiddle3,
                                        LAST #slast ],
                                REL [ LIST #rmiddle3,
                                      LAST #rlast ],
                                QUE [ LIST #qmiddle3,
                                      LAST #qlast ] ],
                    LOCAL.CONT.INDEX individual ],
                  [ NON-LOCAL [ SLASH [ LIST #sfirst,
                                        LAST #smiddle1 ],
                                REL [ LIST #rfirst,
                                      LAST #rmiddle1 ],
                                QUE [ LIST #qfirst,
                                      LAST #qmiddle1 ] ],
                    LOCAL.CONT.INDEX individual ],
                  [ NON-LOCAL [ SLASH [ LIST #smiddle1,
                                        LAST #smiddle2 ],
                                REL [ LIST #rmiddle1,
                                      LAST #rmiddle2 ],
                                QUE [ LIST #qmiddle1,
                                      LAST #qmiddle2 ] ],
                    LOCAL.CONT.INDEX individual ],
                  [ NON-LOCAL [ SLASH [ LIST #smiddle2,
                                        LAST #smiddle3 ],
                                REL [ LIST #rmiddle2,
                                      LAST #rmiddle3 ],
                                QUE [ LIST #qmiddle2,
                                      LAST #qmiddle3 ] ],
                    LOCAL.CONT.INDEX individual ] >,
          CONT.H-CONS <! !> ],
  NON-LOCAL [ SLASH [ LIST #sfirst,
                      LAST #slast ],
              REL [ LIST #rfirst,
                    LAST #rlast ],
              QUE [ LIST #qfirst,
                    LAST #qlast ] ] ].


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; verb lexemes
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

verb-lxm := lexeme &
[ SYNSEM verb-synsem &
         [ LOCAL.CAT.NUC + ] ].


non-atrans-verb-lxm := lexeme &
```

```
[ SYNSEM verb-synsem &
         [ LOCAL [ CONT [ INDEX #ind,
                          LISZT.LIST.FIRST event-rel &
                                #rel & [ EVENT #ind ] ],
                   KEYS.KEY #rel ]]].

verb-word-rule := lex-rule &
[ ARGS < verb-lxm > ].

verb1-lxm := verb-lxm.
verb2a-lxm := verb-lxm.
verb2b-lxm := verb-lxm.
verb2c-lxm := verb-lxm.

;;; 'Det snoer', 'det regner'

verb1-atrans-lxm := verb1-lxm &
[ SYNSEM atrans-verb ].
verb2c-atrans-lxm := verb2c-lxm &
[ SYNSEM atrans-verb ].

;;; 'Jenta smiler'

verb1-unerg-lxm := verb1-lxm & non-atrans-verb-lxm &
[ SYNSEM unerg-verb  ].
verb2a-unerg-lxm := verb2a-lxm & non-atrans-verb-lxm &
[ SYNSEM unerg-verb  ].
verb2b-unerg-lxm := verb2b-lxm & non-atrans-verb-lxm &
[ SYNSEM unerg-verb  ].
verb2c-unerg-lxm := verb2c-lxm & non-atrans-verb-lxm &
[ SYNSEM unerg-verb  ].

;;; 'Gutten beundrer jenta'

verb1-trans-lxm := verb1-lxm & non-atrans-verb-lxm &
[ SYNSEM np-trans-verb ].
verb2a-trans-lxm := verb2a-lxm & non-atrans-verb-lxm &
[ SYNSEM np-trans-verb ].
verb2b-trans-lxm := verb2b-lxm & non-atrans-verb-lxm &
[ SYNSEM  np-trans-verb ].
verb2c-trans-lxm := verb2c-lxm & non-atrans-verb-lxm &
[ SYNSEM np-trans-verb ].

;;; 'Gutten skjenker jenta ei bok'

verb1-ditrans-lxm := verb1-lxm & non-atrans-verb-lxm &
[ SYNSEM ditrans-verb ].

;;; 'Hun putter boken på bordet'

verb1-prep-trans-lxm := verb1-lxm & non-atrans-verb-lxm &
[ SYNSEM prep-trans-verb ].

;;; 'Jenta sitter ved bordet'

verb2b-prep-intrans-lxm := verb2b-lxm & non-atrans-verb-lxm &
[ SYNSEM prep-intrans-verb ].

;;; 'Jenta trodde at gutten smilte', 'Gutten sjekker om hun spydde'

cp-prop-intrans-lxm := non-atrans-verb-lxm &
[  SYNSEM cp-prop-intrans-verb  ].
```

```
om-at-cp-prop-intrans-lxm := cp-prop-intrans-lxm.

;;; 'Jeg mener at hun smilte'

at-cp-prop-intrans-lxm := om-at-cp-prop-intrans-lxm &
[  SYNSEM.LOCAL.CAT.VAL.COMPS < [ LOCAL.CAT.HEAD at-comp ] > ].

om-cp-prop-intrans-lxm := om-at-cp-prop-intrans-lxm &
[  SYNSEM.LOCAL.CAT.VAL.COMPS < [ LOCAL.CAT.HEAD om-comp ] > ].

;;; 'Han sjekker om/at blomsten har blomstret'

verb1-om-at-cp-prop-intrans-lxm := verb1-lxm & om-at-cp-prop-intrans-lxm.

verb2a-at-cp-prop-intrans-lxm := verb2a-lxm & at-cp-prop-intrans-lxm.

verb2c-at-cp-prop-intrans-lxm := verb2c-lxm & at-cp-prop-intrans-lxm.

;;; 'Jenta proever a smile'

ssr-verb-lxm := non-atrans-verb-lxm &
[ SYNSEM ssr-verb ].

verb2b-ssr-verb-lxm := verb2b-lxm & ssr-verb-lxm.

nomod-verb := sign &
[ SYNSEM.LOCAL.CAT.HEAD.MOD <> ].

fin-verb := sign &
[ SYNSEM.LOCAL.CAT.HEAD.VFORM fin ].
inf-verb := sign &
[ SYNSEM.LOCAL.CAT [ HEAD.VFORM inf,
     VAL.SUBJ < unexpressed > ]].
pas-verb := sign &
[ SYNSEM.LOCAL.CAT.HEAD.VFORM pas].
imp-verb := sign &
[ SYNSEM.LOCAL.CAT.HEAD.VFORM imp].

pres-verb := sign &
[ SYNSEM.LOCAL.CONT.INDEX.E.TENSE present].
past-verb := sign &
[ SYNSEM.LOCAL.CONT.INDEX.E.TENSE past].
prp-verb := sign &
[ SYNSEM.LOCAL.CONT.INDEX.E.TENSE presperf].
psp-verb := sign &
[ SYNSEM.LOCAL [ CAT [ HEAD.VFORM psp,
                       VAL.SUBJ < unexpressed > ],
                 CONT.INDEX.E.TENSE pastperf]].

indicative-verb := sign &
[ SYNSEM.LOCAL.CONT.INDEX.E.MOOD indicative].

active-verb := sign &
[ SYNSEM.LOCAL.CONT.INDEX.E.ASPECT active].
passive-verb := sign &
[ SYNSEM.LOCAL.CONT.INDEX.E.ASPECT passive ].

pres-fin-verb := pres-verb & fin-verb & nomod-verb.
past-fin-verb := past-verb & fin-verb & nomod-verb.

pres-verb-rule := infl-ltow-rule &  pres-fin-verb & indicative-verb & active-verb.
```

```
past-verb-rule := infl-ltow-rule &  past-fin-verb & indicative-verb & active-verb.
psp-verb-rule := infl-ltow-rule &  psp-verb &  nomod-verb.

inf-verb-rule := infl-ltow-rule & inf-verb.

pres-fin-psp-aux-verb-word := pres-fin-verb & indicative-verb & active-verb &
                              psp-aux-verb-word.
past-fin-psp-aux-verb-word := past-fin-verb & indicative-verb & active-verb &
                              psp-aux-verb-word.


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; lexical defaults
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

nonslash := word &
  [ SYNSEM.NON-LOCAL.SLASH 0-dlist ].

nonque := word &
  [ SYNSEM.NON-LOCAL.QUE 0-dlist ].

nonrel := word &
  [ SYNSEM.NON-LOCAL.REL 0-dlist ].

nonmsg := word &
  [ SYNSEM.LOCAL.KEYS.MESSAGE 0-dlist ].

topkey := word &
  [ SYNSEM.LOCAL [ KEYS.KEY #key,
   CONT.--TOPKEY #key ] ].

nontopkey := nonque & nonslash & nonrel & nonmsg & non-affix-bearing.

msg-amalg-word :=   non-affix-bearing & topkey.


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; prepositions
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

basic-prep-synsem := lex-synsem &
[ LOCAL [ CAT [ HEAD prep,
                VAL [ SUBJ <>,
                      SPR <>,
                      COMPS <  synsem &
                               [ LOCAL local-min &
                                       [ KEYS.KEY #ckey,
                                         CONT [ TOP #top,
                                                INDEX #objind ] ]], ... > ],
                POSTHEAD + ],
          CONT [ TOP #top,
                 LISZT.LIST < #key, ... > ],
          KEYS [ KEY #key & [ ARG3 #objind ],
                 ALTKEY #ckey,
                 --COMPKEY #ckey ]]].

basic-mod-n-or-vp-synsem := isect-synsem &
[ LOCAL [ CAT [ HEAD.MOD < [ LOCAL [ CAT [ HEAD verb-or-noun,
```

```
                                        VAL [  SUBJ olist,
                                               SPR olist,
                                               COMPS olist ],
                                        MC #mc ],
                                   CONT [ INDEX #ind & non-expl ],
                                   KEYS.MESSAGE #msg ]]  >,
                POSTHEAD +,
                MC #mc ],
          CONT.INDEX #ind ,
          KEYS [ KEY.ARG #ind,
                 MESSAGE #msg ]]] .

prep-mod-synsem := basic-prep-synsem & basic-mod-n-or-vp-synsem.

trans-prep-synsem := prep-mod-synsem & one-arg &
[ LOCAL [ CAT.VAL.COMPS < #comps & [ LOCAL.CONT.INDEX ref-ind ] >,
          CONT.LISZT <! prep-mod-rel !> ,
          ARG-S < #comps > ]].

basic-prep-word := msg-amalg-word &
[ SYNSEM trans-prep-synsem &
         [ LOCAL.CAT.VAL.COMPS < synsem &
                                 [ LOCAL [ CAT nomp-cat-acc-min &
                                              [ VAL.SPR < > ]]] > ]].

prep-word := basic-prep-word &
[ SYNSEM.LOCAL.CAT.VAL.COMPS < [ LOCAL.CONT.INDEX non-expl ] > ].

final-prep-word := prep-word &
[ SYNSEM.LOCAL.CAT.HEAD final-prep ].


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; complementizers
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

complementizer-word := word &
[ SYNSEM.LOCAL local-min &
               [ CAT [ HEAD comp,
                       VAL.SPR < > ],
                 CONT.INDEX non-expl,
                 KEYS.KEY event-rel ] ].

plain-compl-word := complementizer-word &
[ SYNSEM.LOCAL [ CAT.VAL [ SUBJ < #subj >,
                           COMPS #comps &
                                < [ LOCAL local-min &
                                          [ KEYS.KEY #ckey &
                                                     [ LABEL #label ],
                                            CONT.INDEX #ind ],
                                    OPT - ], ... > ],
                 CONT.INDEX #ind,
                 ARG-S < #subj . #comps >,
                 KEYS [ KEY.LABEL #label,
                        --COMPKEY #ckey ] ] ].

basic-compl-word := complementizer-word &
[ SYNSEM nomod-basic-onearg-synsem &
         [ LOCAL [ ARG-S #comps,
                   CAT [ HEAD [  VFORM #vform ],
```

```
                        MC -,
                        VAL [  COMPS #comps &
                                     < synsem &
                                       [ LOCAL local-min &
                                               [ CAT [ HEAD verb &
                                                           [  VFORM #vform ],
                                                       VAL [ SUBJ < anti-synsem >,
                                                             COMPS <> ]],
                                                 CONT [ TOP #chand ],
                                                 KEYS.KEY #ckey ],
                                          OPT - ] > ] ],
                    CONT [ TOP #mhand,

                               LISZT <! #msg !>,
                         H-CONS <! qeq &
                                   [ SC-ARG #soahand,
                                     OUTSCPD #chand ] !> ],
                    KEYS [ KEY #ckey,
                           MESSAGE 1-dlist &
                               <! #msg & [ HNDL #mhand,
                                           SOA #soahand ] !>,
                           --COMPKEY #ckey ] ] ] ].

at-compl-word := basic-compl-word &
[ SYNSEM.LOCAL [ CAT [ HEAD [ VFORM fin,
                              AUX -  ]],
                 KEYS.MESSAGE 1-dlist &
                     <! proposition !> ] ].

at-mcl-compl-word := at-compl-word &
[ SYNSEM.LOCAL.CAT.VAL.COMPS < [ LOCAL [ CAT.MC +,
                                         KEYS.MESSAGE 1-dlist &
                                                      <! proposition !> ]] > ].

at-subcl-compl-word := at-compl-word &
[ SYNSEM.LOCAL.CAT.VAL.COMPS < [ LOCAL.CAT.MC - ] > ].

om-compl-word := basic-compl-word &
[ SYNSEM.LOCAL [ CAT [ HEAD [ VFORM fin,
                              AUX -   ],
                       VAL.COMPS < [ LOCAL.CAT.MC - ] >],
                 KEYS.MESSAGE 1-dlist &
                     <! proposition !> ] ].

to-compl-word := plain-compl-word &
[ SYNSEM basic-two-arg &
  [ LOCAL.CAT [ HEAD [ VFORM inf],
                VAL [ SUBJ < [ LOCAL.CONT #cont ] >,
                      COMPS < [ LOCAL.CAT [ HEAD verbal & [ VFORM inf ],
                                            VAL [ SUBJ < [ LOCAL.CONT #cont,
                                                           NON-LOCAL [ SLASH 0-dlist,
                                                                       REL 0-dlist,
                                                                       QUE 0-dlist ]] > ,
                                                  COMPS olist ],
                                             MC na ]] > ]]]].

to-compl-nonprop-word := to-compl-word &
[ SYNSEM.LOCAL nomod-local &
              [ CAT [ VAL [ COMPS < [ LOCAL local-min &
                                      [ CONT.TOP #hand ] ] > ],
                            MC na ],
                CONT [ TOP #hand,
```

```
                      LISZT <! !>,
                      H-CONS <! !> ],
                KEYS.MESSAGE 0-dlist ] ].

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; CP verbs
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

basic-cp-prop-ques-verb := verb-synsem &
[ LOCAL [ CAT.VAL [ --KEYCOMP [ LOCAL local-min &
                                    [ CAT [ HEAD verbal ]]]],
          CONT.LISZT.LIST < relation, ... > ] ].

cp-prop-ques-verb := basic-cp-prop-ques-verb.

fin-cp-prop-ques-verb := cp-prop-ques-verb &
[ LOCAL.CAT.VAL.--KEYCOMP.LOCAL.CAT.HEAD.VFORM fin ].

cp-intrans-verb := fin-cp-prop-ques-verb &  cp-intrans-subst & two-arg & arg14-lt &
[ LOCAL.CAT.VAL.COMPS < [ LOCAL.CAT.MC - ] > ].

cp-prop-intrans-verb := cp-intrans-verb.

cp-ques-intrans-verb := cp-intrans-verb.

ssr-verb := verb-synsem & ssr-inf-subst & basic-two-arg & ssr-v-lt &
[ LOCAL [ CAT.VAL.--KEYCOMP.LOCAL.CONT.TOP #chand,
          CONT.H-CONS <! qeq &
                       [ SC-ARG #arghand,
                         OUTSCPD #chand ] !>,
          KEYS.KEY.ARG4 #arghand ] ].


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; auxiliary verbs
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

aux-verb := verb-synsem.

aux-verb-ssr := aux-verb & ssr-subst & basic-two-arg &
[ LOCAL.CAT.VAL [ --KEYCOMP #comp,
                  COMPS < #comp, ... > ] ].

aux-verb-word-super :=  nonmsg &
[ INFLECTED +,
  SYNSEM aux-verb-ssr ].

aux-verb-word := aux-verb-word-super &
[ SYNSEM.LOCAL.CAT.HEAD.AUX + ].

psp-aux-verb-word := aux-verb-word &
[ SYNSEM aux-verb-ssr &
  [ LOCAL [ CAT.VAL.COMPS < [ OPT -,
                              LOCAL local-basic &
                                  [ KEYS.KEY.LABEL #label,
                                    CONT.TOP #hand,
                                    CAT [ HEAD verbal &
                                                [ VFORM psp ],
```

```
                                        VAL [ SUBJ < synsem >,
                                              COMPS olist,
                                              --KEYCOMP.LOCAL.CONT.TOP #chand]]]] >,
            CONT [ LISZT <! #key !>,
                         INDEX #event,
                   H-CONS < ! qeq & [ SC-ARG #arghand,
                                      OUTSCPD #chand ] !> ],
            KEYS.KEY #key & [ EVENT #event,
                              HNDL #hand,
                              LABEL #label,
                              ARG4 #arghand ]]]].


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; adverbs
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

basic-adverb-synsem := basic-zero-arg &
[ LOCAL [ CAT [ HEAD adv &
                     [  MOD < [ LOCAL local-basic &
                                      [ CONT [ INDEX #vevent &
                                                     individual],
                                        KEYS.MESSAGE #msg  ] ] > ],
                VAL [ SUBJ < >,
                      COMPS < > ] ],
          CONT [ INDEX #vevent,
                 LISZT.LIST < #key, ... > ],
          KEYS [ KEY #key & adv-rel,
                 MESSAGE #msg ]] ].

basic-int-adverb-synsem := basic-adverb-synsem & isect-synsem &
[ LOCAL [ CAT.HEAD adv &
                   [ MOD < [ LOCAL.CONT [ TOP #top, INDEX #vevent ]] > ],
          CONT [ INDEX #vevent,
                 LISZT <! relation !>,
                 H-CONS <! !> ],
          KEYS.KEY [ HNDL #top, ARG #vevent ] ] ].

intersect-s-adverb-nospec-synsem := basic-int-adverb-synsem &
[ LOCAL.CAT [ VAL.SPR <>,
              HEAD.MOD < [ LOCAL.CAT [ HEAD verb,
                                       MC #mc ]] >,
              MC #mc ]].

basic-int-adverb-word := topkey &
[ SYNSEM intersect-s-adverb-nospec-synsem ].

non-emph-adverb-word := basic-int-adverb-word &
[ SYNSEM.LOCAL.CAT.HEAD non-emph-adv ].

emph-adverb-word :=  basic-int-adverb-word &
[ SYNSEM.LOCAL.CAT.HEAD emph-adv ].

nex-or-final-adverb-word :=  basic-int-adverb-word &
[ SYNSEM.LOCAL.CAT.HEAD nex-or-final-adv ].

final-adverb-word := basic-int-adverb-word &
[ SYNSEM.LOCAL.CAT.HEAD final-adv ].
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; phrase types
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

det-subj-phrase := lex-rule &
[ INFLECTED +,
  KEY-ARG #keyarg,
  ROOT #root,
  STEM #stem,
  SYNSEM synsem &
         [ LOCAL [ CAT [ HEAD #head & verb,
                         VAL [ SUBJ < synsem &
                                      [ LOCAL.CONT.INDEX det-ind,
                                        NON-LOCAL.SLASH [ LIST #3, LAST #4] ] >,
                               COMPS < #subj, #comps  >,
                               SPR #spr ]],
                   CONT #cont,
                      KEYS #keys ],
           NON-LOCAL.SLASH [ LIST #1, LAST #4 ],
           LEX + ],
  DTR lex-rule &
      [ INFLECTED +,
        STEM #stem,
        KEY-ARG #keyarg,
        ROOT #root,
        SYNSEM prep-intrans-verb &
               [ LOCAL [ CAT [ HEAD #head,
                               VAL [ SUBJ < synsem & #subj &
                                            [ LOCAL.AGR.PNG.DEF indef,
                                              NON-LOCAL.SLASH.LAST #3 ] >,
                                     COMPS < #comps & [ NON-LOCAL.SLASH.LIST #1 ] >,
                                     SPR #spr]],
                         CONT #cont,
                         KEYS #keys ] ]],
  C-CONT.LISZT <! !> ].

yes-no-interrogative-phrase :=  unary-phrase & interrogative-clause &
[ SYNSEM [ LOCAL [ CAT #cat,
                   CONT [ TOP #top,
                          INDEX #index,
                          E-INDEX #e-index],
                   KEYS [KEY #key,
                         ALTKEY #altkey,
                         MESSAGE 1-dlist & <! quest-rel & #msg !>]],
           NON-LOCAL.SLASH 0-dlist],
  ARGS < sign & [ SYNSEM  [ LOCAL [ CAT #cat & [ HEAD verb & [ VFORM fin ],
                                                 VAL [ SUBJ < anti-synsem >,
                                                       COMPS <>],
                                                 MC +],
                                    CONT [ TOP #top,
                                           INDEX #index,
                                           E-INDEX #e-index ],
                                    KEYS [ MESSAGE 0-dlist,
                                           KEY #key,
                                           ALTKEY #altkey ]],
                              NON-LOCAL.SLASH 0-dlist ]] >,
  C-CONT.LISZT <! #msg !> ].

s-int-phrase :=  isect-mod-phrase  &
[  HEAD-DTR.SYNSEM.LOCAL.CAT.HEAD verb & [ VFORM fin ] ].
```

```
s-main-nex-hadj-int-phrase :=  s-int-phrase & head-adj-int-phrase  &
[ SYNSEM.LOCAL.CAT.MC +,
  NON-HEAD-DTR.SYNSEM.LOCAL.CAT.HEAD nex-mod ].

s-hadj-int-phrase :=   s-int-phrase & head-adj-int-phrase  &
[ NON-HEAD-DTR.SYNSEM.LOCAL.CAT.HEAD final-or-strictly-final-mod ].

s-sub-nex-adjh-int-phrase :=  s-int-phrase & adj-head-int-phrase &
[ SYNSEM.LOCAL.CAT.MC -,
  HEAD-DTR.SYNSEM.LOCAL.CAT [ VAL.SUBJ < expressed-synsem > ],
  NON-HEAD-DTR.SYNSEM.LOCAL.CAT.HEAD nex-mod ].

n-hadj-int-phrase :=  head-adj-int-phrase  &
[ HEAD-DTR.SYNSEM.LOCAL.CAT.HEAD noun ].

n-adjh-int-phrase := adj-head-int-phrase &
[ HEAD-DTR.SYNSEM.LOCAL.CAT.HEAD noun ].

bare-np-phrase := head-valence-phrase & head-only &
[ SYNSEM [ LOCAL [ CAT [ VAL [ SUBJ null,
                               SPR null,
                               COMPS null ]],
                   KEYS [ ALTKEY #altkey,
                          MESSAGE #hmsg ]],
           NON-LOCAL.SLASH #slash ],
  C-CONT [ H-CONS <! qeq & [ SC-ARG #scopearg ,
                             OUTSCPD #outscoped ] !>,
           LISZT <! quant-rel & #key & [ BV #bv,
                                         RESTR #scopearg] !>,
           INDEX #bv ],
  HEAD-DTR.SYNSEM [ LOCAL [ CAT [ VAL [ SPR < canonical-synsem &
                                              [ LOCAL.KEYS [ KEY #key ],
                                                NON-LOCAL [ SLASH 0-dlist,
                                                            REL 0-dlist,
                                                            QUE 0-dlist ]] >,
                                        SUBJ olist,
                                        COMPS olist ],
                                 HEAD noun ],
                            KEYS [ KEY basic-nom-rel,
                                   ALTKEY #altkey,
                                   MESSAGE #hmsg ],
                            CONT [ TOP #outscoped,
                                   INDEX #bv ]],
                    LEX +,
                    NON-LOCAL.SLASH #slash ]].

bare-np-def-phrase := bare-np-phrase & rule &
[ SYNSEM.LOCAL.AGR.PNG.DEF def,
  HEAD-DTR.SYNSEM.MODIFIED notmod-or-rmod,
  C-CONT.LISZT.LIST.FIRST def-rel & [PRED 'def-rel],
  RULE-NAME 'bare-np-def ].

bare-np-indef-phrase := bare-np-phrase & rule &
[ SYNSEM.LOCAL.AGR.PNG [ DEF indef,
                         NUM plur ],
  C-CONT.LISZT.LIST.FIRST udef-rel & [ PRED 'udef-rel],
  RULE-NAME 'bare-np-indef ].

head-specifier-phrase := basic-head-spec-phrase & head-final & rule &
[ HEAD-DTR.SYNSEM.LOCAL [ CONT [ TOP #outscoped,
                                 INDEX #bv ],
```

```
                        AGR.PNG #png ],
  NON-HEAD-DTR.SYNSEM.LOCAL [ CONT [ H-CONS.LIST.FIRST qeq &
                                                       [ SC-ARG #scarg ,
                                                         OUTSCPD #outscoped ],
                                   LISZT.LIST.FIRST quant-rel &
                                                       [ BV #bv,
                                                         RESTR  #scarg],
                                   INDEX #bv ],
                              AGR.PNG #png ],
  RULE-NAME 'hspec ].

head-subj-phrase1 := basic-head-subj-phrase & rule &
[ SYNSEM.LOCAL.CAT [ MC #mc ],
  HEAD-DTR.SYNSEM.LOCAL.CAT.MC #mc, C-CONT [ LISZT <! !>,
                                             H-CONS <! !> ]].

subj-head-phrase := head-subj-phrase1 & head-final &
[ SYNSEM [ LOCAL.CAT.MC - ],
  RULE-NAME 'subjh ].

head-subj-phrase := head-subj-phrase1 & head-initial & non-clause &
[ SYNSEM [ LOCAL [ CAT.MC + ]],
  RULE-NAME 'hsubj ].

head-comp-phrase := basic-head-comp-phrase & head-initial &  rule &
[ HEAD-DTR.SYNSEM.LOCAL.CAT.VAL.SUBJ olist,
  NON-HEAD-DTR.SYNSEM.LOCAL.CAT.VAL [ SPR olist, COMPS  olist ],
  RULE-NAME 'hcomp1 ].

extr-subj-phrase := basic-extracted-subj-phrase & head-compositional & rule &
[ HEAD-DTR.SYNSEM.MODIFIED notmod-or-lmod,
  C-CONT [ LISZT <! !>,
   H-CONS <! !> ],
  RULE-NAME 'extr-subj ].

extr-comp-phrase := basic-extracted-comp-phrase & rule &
[ SYNSEM [ LOCAL.CAT.VAL.SUBJ olist ],
  RULE-NAME 'extr-comp ].

adv-extracted-adj-phrase := extracted-adj-phrase &
[ SYNSEM.NON-LOCAL.SLASH.LIST.FIRST.CAT.HEAD emph-mod,
  HEAD-DTR.SYNSEM.LOCAL [ CAT.VAL.SUBJ < anti-synsem >,
  KEYS.MESSAGE 0-dlist ]].

adv-fin-extracted-adj-phrase := extracted-adj-phrase &
[ SYNSEM.NON-LOCAL.SLASH.LIST.FIRST.CAT.HEAD strictly-final-mod,
  HEAD-DTR.SYNSEM.LOCAL.KEYS.MESSAGE 0-dlist].

fillhead-phrase := basic-head-filler-phrase & head-final & rule &
                   declarative-clause & head-compositional &
[ SYNSEM.LOCAL [ CAT [ VAL [ SUBJ < anti-synsem >,
                             COMPS <>],
                       MC #mc & +,
                       NUC - ],
                 KEYS [ KEY.HNDL #chand,
                        MESSAGE  1-dlist & [ LIST.FIRST #msg & prop-rel &
                                                       [ SOA #soahand] ]]],
  ARGS < [ SYNSEM.LOCAL.CAT.VAL.COMPS <>],
         [ SYNSEM.LOCAL.CAT [ HEAD verb & [ VFORM fin ],
                              VAL [ SUBJ < anti-synsem >, COMPS <> ],
                              MC #mc ]] >,
  C-CONT [ LISZT <! #msg !>,
```

```
         H-CONS <! qeq &
                 [ SC-ARG #soahand,
                   OUTSCPD #chand ] !>],
RULE-NAME 'fillhead ].
```

Following is the file '`rules.tdl`' from the implemented grammar fragment. The file contains the phrasal construction types described in section 7. Inflectional and other lexical rules are not included in the appendix.

```
;;; -*- Mode: tdl; Package: lkb -*-

bare-np-indef-rule := bare-np-indef-phrase.
bare-np-def-rule := bare-np-def-phrase.

head-specifier-rule := head-specifier-phrase.

s-main-nex-hadj-int-rule := s-main-nex-hadj-int-phrase.
s-hadj-int-rule := s-hadj-int-phrase.
s-sub-nex-adjh-int-rule := s-sub-nex-adjh-int-phrase.

n-hadj-int-rule := n-hadj-int-phrase.
n-adjh-int-rule := n-adjh-int-phrase.

subj-head-rule := subj-head-phrase.
head-subj-rule := head-subj-phrase.
head-comp-rule := head-comp-phrase.

subj-extr-rule := extr-subj-phrase.
comp-extr-rule := extr-comp-phrase.
adv-adj-extr-rule := adv-extracted-adj-phrase.
adv-fin-adj-extr-rule := adv-fin-extracted-adj-phrase.

yes-no-int-rule := yes-no-interrogative-phrase.
head-filler-rule := fillhead-phrase.
```

Following is the file '`lexicon.tdl`' from the implemented grammar fragment. The file contains the vocabulary provided by the grammar, each entry instantiating exactly one lexical type and supplying only small amounts of additional information, i.e. the base orthography and the key semantic relation.

```
;;; -*- Mode: tdl; Package: lkb -*-

det-expl := expletive-det-word &
[ STEM < "det" > ].

gyrd-pn := proper-name-sg &
[ STEM < "Gyrd" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'gyrd].

inge-pn := proper-name-sg &
[ STEM < "Inge" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'inge].

aud-pn := proper-name-sg &
[ STEM < "Aud" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'aud].

per-pn := proper-name-sg &
[ STEM < "Per" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'per].

Tranmæl-pn := proper-name-sg &
[ STEM < "Tranmæl" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'tranmæl].

jeg-pp := perspron-1st-sg-nom &
[ STEM < "jeg" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'jeg].

du-pp := perspron-2nd-sg-nom &
[ STEM < "du" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'du].

han-pp := perspron-3rd-sg-nom-masc &
[ STEM < "han" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'han].

hun-pp := perspron-3rd-sg-nom-fem &
[ STEM < "hun" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'hun].

vi-pp := perspron-1st-pl-nom &
[ STEM < "vi" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'vi].

dere-pp := perspron-2nd-pl &
[ STEM < "dere" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'dere ].

de-pp := perspron-3rd-pl &
[ STEM < "de" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'de].

meg-pp := perspron-1st-sg-acc &
[ STEM < "meg" >,
```

```
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'meg].

deg-pp := perspron-2nd-sg-acc &
[ STEM < "deg" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'deg].

ham-pp := perspron-3rd-sg-acc-masc &
[ STEM < "ham" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'ham].

henne-pp := perspron-3rd-sg-acc-fem &
[ STEM < "henne" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'henne].

oss-pp := perspron-1st-pl-acc &
[ STEM < "oss" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'oss].

dem-pp := perspron-3rd-pl-acc &
[ STEM < "dem" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'dem].

jente-noun :=  fem-noun-lxm &
[ STEM < "jente" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'jente].

gutt-noun := masc-noun-lxm &
[ STEM < "gutt" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'gutt].

hjelm-noun := masc-noun-lxm &
[ STEM < "hjelm" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'hjelm].

dam-noun := masc-noun-lxm &
[ STEM < "dam" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'dam].

lærer-noun := masc-noun-lxm &
[ STEM < "lærer" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'lærer].

sjel-noun := masc-noun-lxm &
[ STEM < "sjel" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'sjel].

barm-noun := masc-noun-lxm &
[ STEM < "barm" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'barm].

klem-noun := masc-noun-lxm &
[ STEM < "klem" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'klem].

sykkel-noun := masc-noun-lxm &
[ STEM < "sykkel" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'sykkel].

blomst-noun := masc-noun-lxm &
[ STEM < "blomst" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'blomst].
```

```
bok-noun := fem-noun-lxm &
[ STEM < "bok" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'bok].

dør-noun := fem-noun-lxm &
[ STEM < "dør" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'dør].

ball-noun := masc-noun-lxm &
[ STEM < "ball" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'ball].

bane-noun := masc-noun-lxm &
[ STEM < "bane" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'bane].

fly-noun :=  neut-noun-lxm &
[ STEM < "fly" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'fly].

hjem-noun :=  neut-noun-lxm &
[ STEM < "hjem" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'hjem].

bord-noun :=  neut-noun-lxm &
[ STEM < "bord" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'bord].

hus-noun :=  neut-noun-lxm &
[ STEM < "hus" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'hus].

ei-det := det-word-sing-indef-fem &
[ STEM < "ei" >,
  SYNSEM.LOCAL [ CONT.LISZT.LIST.FIRST.PRED 'ei,
                 KEYS.KEY udef-rel]].

et-det := det-word-sing-indef-neut &
[ STEM < "et" >,
  SYNSEM.LOCAL [ CONT.LISZT.LIST.FIRST.PRED 'et,
                 KEYS.KEY udef-rel ]].

en-det := det-word-sing-indef-masc-or-fem &
[ STEM < "en" >,
  SYNSEM.LOCAL [ CONT.LISZT.LIST.FIRST.PRED 'en,
                 KEYS.KEY udef-rel]].

det-det := det-word-sing-def-neut &
[ STEM < "det" >,
  SYNSEM.LOCAL [ CONT.LISZT.LIST.FIRST.PRED 'det,
                 KEYS.KEY def-rel]].

den-det := det-word-sing-def-masc-or-fem &
[ STEM < "den" >,
  SYNSEM.LOCAL [ CONT.LISZT.LIST.FIRST.PRED 'den,
                 KEYS.KEY def-rel]].

de-det := det-word-pl-def &
[ STEM < "de" >,
  SYNSEM.LOCAL [ CONT.LISZT.LIST.FIRST.PRED 'de,
                 KEYS.KEY def-rel]].
```

```
dette-det := det-word-sing-def-neut &
[ STEM < "dette" >,
  SYNSEM.LOCAL [ CONT.LISZT.LIST.FIRST.PRED 'dette,
                 KEYS.KEY def-rel]].

denne-det := det-word-sing-def-masc-or-fem &
[ STEM < "denne" >,
  SYNSEM.LOCAL [ CONT.LISZT.LIST.FIRST.PRED 'denne,
                 KEYS.KEY def-rel]].

disse-det := det-word-pl-def &
[ STEM < "disse" >,
  SYNSEM.LOCAL [ CONT.LISZT.LIST.FIRST.PRED 'disse,
                 KEYS.KEY def-rel]].

alle-det := det-word-plur &
[ STEM < "alle" >,
  SYNSEM.LOCAL [ CONT.LISZT.LIST.FIRST.PRED 'alle,
                 KEYS.KEY all-rel]].

noen-det := det-word-pl-indef &
[ STEM < "noen" >,
  SYNSEM.LOCAL [ CONT.LISZT.LIST.FIRST.PRED 'noen,
                 KEYS.KEY some-rel]].

god-adj := adj-lxm &
[ STEM < "god" >,
  SYNSEM.LOCAL [  CONT.LISZT.LIST.FIRST.PRED 'god]].

blid-adj := adj-lxm &
[ STEM < "blid" >,
  SYNSEM.LOCAL [  CONT.LISZT.LIST.FIRST.PRED 'blid]].

rød-adj := adj-lxm &
[ STEM < "rød" >,
  SYNSEM.LOCAL [  CONT.LISZT.LIST.FIRST.PRED 'rød]].

forelsket-adj := adj-lxm &
[ STEM < "forelsket" >,
  SYNSEM.LOCAL [  CONT.LISZT.LIST.FIRST.PRED 'forelsket]].

regn-v := verb1-atrans-lxm &
[ STEM < "regn" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'regn-rel].

snø-v := verb2c-atrans-lxm &
[ STEM < "snø" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'snø-rel].

blø-v := verb2c-unerg-lxm &
[ STEM < "blø" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'blø-rel].

smil-v := verb2a-unerg-lxm &
[ STEM < "smil" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'smile-rel].

blomstr-v := verb1-unerg-lxm &
[ STEM < "blomstr" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'blomstre-rel].

svai-v := verb1-unerg-lxm &
```

```
[ STEM < "svai" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'svaie-rel].

kast-v := verb1-trans-lxm &
[ STEM < "kast" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'kast-rel].

tap-v := verb2a-unerg-lxm &
[ STEM < "tap" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'tape-rel].

spy-v := verb2c-unerg-lxm &
[ STEM < "spy" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'spy-rel].

les-v := verb2a-trans-lxm &
[ STEM < "les" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'lese-rel ].

lik-v := verb2a-trans-lxm &
[ STEM < "lik" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'like-rel ].

beundr-v := verb1-trans-lxm &
[ STEM < "beundr" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'beundre-rel ].

skjenk-v := verb1-ditrans-lxm &
[ STEM < "skjenk" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'skjenke-rel ].

putt-v := verb1-prep-trans-lxm  &
[ STEM < "putt" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'putte-rel ].

sitt-v := verb2b-prep-intrans-lxm  &
[ STEM < "sitt" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'sitte-rel ].

ha-aux-v := pres-fin-psp-aux-verb-word &
[ STEM < "har" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'ha-rel ].

hadde-aux-v := past-fin-psp-aux-verb-word &
[ STEM < "hadde" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'hadde-rel ].

men-v := verb2a-at-cp-prop-intrans-lxm &
[ STEM < "men" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'mene-rel ].

tro-v := verb2c-at-cp-prop-intrans-lxm &
[ STEM < "tro" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'tro-rel ].

sjekk-v := verb1-om-at-cp-prop-intrans-lxm &
[ STEM < "sjekk" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'sjekke-rel ].

prøv-v := verb2b-ssr-verb-lxm &
[ STEM < "prøv" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'prøve-rel ].
```

```
ved-v := final-prep-word  &
[ STEM < "ved" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'ved-rel ].

på-v := final-prep-word  &
[ STEM < "på" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'på-rel ].

til-v := final-prep-word  &
[ STEM < "til" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'til-rel ].

at-m-c := at-mcl-compl-word  &
[ STEM < "at" >,
  SYNSEM.LOCAL.CAT.HEAD at-comp].

at-sub-c := at-subcl-compl-word  &
[ STEM < "at" >,
  SYNSEM.LOCAL.CAT.HEAD at-comp].

om-c := om-compl-word  &
[ STEM < "om" >,
 SYNSEM.LOCAL.CAT.HEAD om-comp].

å-c := to-compl-nonprop-word  &
[ STEM < "å" > ].

sikkert-adv := non-emph-adverb-word &
[ STEM < "sikkert" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'sikkert-rel ].

jo-adv := non-emph-adverb-word &
[ STEM < "jo" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'jo-rel ].

kanskje-adv := emph-adverb-word &
[ STEM < "kanskje" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'kanskje-rel ].

allikevel-adv := nex-or-final-adverb-word &
[ STEM < "allikevel" >,
  SYNSEM.LOCAL.CONT.LISZT.LIST.FIRST.PRED 'allikevel-rel ].
```

C Test corpora

Following is the test suite that I constructed and used for the grammar implementation. This collection of sentences and non-sentences exemplifies all the syntactic phenomena discussed in the main sections of this thesis, systematically tests for overgeneration (of which there is none), and also demonstrates some of the variability in vocabulary and construction types provided for by the implemented grammar fragment.

The columns of the table are, from left to right, the test item number, the actual sentece (with a leading asterisk to mark ungrammaticality where appropriate), and the number of distinct analyses assigned by the grammar.

i	test item input	analyses
1	*Inge smiler*	1
2	*Gyrd beundrer Inge*	2
3	*Gyrd skjenket Inge en bok*	3
4	*Han skjenket henne en bok*	1
5	*Henne skjenket han en bok*	2
6	*En bok skjenket han henne*	2
7	**Han henne beundret*	0
8	**En bok skjenket henne han*	0
9	*Smiler Inge*	1
10	*Beundrer Inge Gyrd*	1
11	*Skjenket Gyrd Inge en bok*	1
12	*Beundret han henne*	1
13	**Beundret henne han*	0
14	*Jeg sjekker om Gyrd beundrer Inge*	1
15	*Jeg sjekker om Inge skjenker Gyrd en bok*	1
16	*Jeg sjekker om hun beundrer ham*	1
17	**Jeg sjekker om ham beundrer hun*	0
18	*Jeg tror at Gyrd beundrer Inge*	3
19	*Jeg sjekker at Gyrd skjenker Inge en bok*	4
20	*Jeg tror at hun beundrer ham*	2
21	*Jeg tror at ham beundrer hun*	1
22	**Jeg sjekker at beundrer hun ham*	0
23	*Det snør*	1
24	*Det regner*	1
25	*Snør det*	1
26	**Det snør jenter*	0
27	**Jenta snør*	0
28	*En rød blomst blomstrer*	1
29	*De blide jentene smilte*	1
30	*Blomstrer den røde blomsten*	1
31	**Det smiler*	0
32	**Jenta smiler en gutt*	0
33	*Jenta beundrer flyet*	2

34	*Gutten leser boka*	2
35	*Leser gutten boka*	1
36	**Du beundrer*	0
37	**Han leser henne boka*	0
38	*Jenta skjenket gutten en bok*	3
39	*De skjenket dem en bok*	1
40	*Skjenket hun ham boka*	1
41	**De skjenket dem*	0
42	**Han skjenket hun en bok*	0
43	*Jenta sitter ved bordet*	1
44	*Ved bordet sitter jenta*	1
45	*Sitter Gyrd ved bordet*	1
46	**Jenta sitter*	0
47	**Jenta sitter blomsten*	0
48	**Sitter ved bordet Gyrd*	0
49	*Jenta puttet flyet på bordet*	2
50	*På bordet puttet jenta flyet*	1
51	*Puttet jenta flyet på bordet*	1
52	**Vi puttet på bordet*	0
53	**Vi puttet flyet*	0
54	*Jenta trodde at gutten smilte*	2
55	*Jenta sjekket om gutten smilte*	1
56	*Jenta sjekket at gutten smilte*	2
57	*At hun smilte trodde han*	2
58	*Trodde du at Inge smilte*	2
59	*Sjekket du om Gyrd smilte*	1
60	**Jenta trodde om gutten smilte*	0
61	**Jenta trodde at*	0
62	**Hun sjekket gutten smilte*	0
63	*Hun prøvde å smile*	1
64	*Gyrd prøvde å skjenke Inge en bok*	1
65	*Å skjenke Inge en bok prøvde Gyrd*	1
66	*En bok prøvde Gyrd å skjenke Inge*	2
67	*Gyrd prøvde å tro at jenta smilte*	2
68	*Prøvde hun å smile*	1
69	*Prøvde Gyrd å skjenke Inge en bok*	1
70	**Hun prøvde å smilte*	0
71	**Hun prøvde hun å smile*	0
72	*Gyrd har lest boka*	1
73	*Jeg hadde lest boka*	1
74	*Gyrd hadde beundret Inge*	1
75	*Gyrd har skjenket Inge en bok*	1
76	*Jenta hadde sittet ved bordet*	2
77	*Gyrd har prøvd å skjenke Inge en bok*	1
78	*Jeg har sjekket om du beundret henne*	1
79	*Har Gyrd beundret Inge*	1

80	*Hadde Gyrd prøvd å skjenke Inge en bok*	1
81	*Jeg trodde at henne hadde du beundret*	1
82	*Trodde du at jeg hadde beundret henne*	2
83	**Jeg har leser boka*	0
84	*Gyrd smilte sikkert*	1
85	*Gyrd beundret sikkert Inge*	2
86	*Gyrd beundret Inge sikkert*	1
87	*Du beundret sikkert Inge*	1
88	*Gyrd skjenket sikkert Inge en bok*	3
89	*Gyrd skjenket Inge sikkert en bok*	2
90	**Sikkert beundret Inge Gyrd*	0
91	**Du beundret Inge sikkert*	0
92	**Gyrd skjenket Inge en bok sikkert*	0
93	**Hun skjenket Inge sikkert en bok*	0
94	*Gyrd prøvde sikkert å smile*	1
95	*Gyrd trodde sikkert at Inge smilte*	2
96	*Gyrd har sikkert Inge skjenket en bok*	2
97	*Gyrd har Inge sikkert skjenket en bok*	2
98	*Gyrd har sikkert skjenket Inge en bok*	1
99	*Gyrd har sikkert trodd at Inge smilte*	2
100	**Gyrd har trodd sikkert at Inge smilte*	0
101	*Jeg trodde at Inge sikkert smilte*	1
102	*Jeg trodde at Gyrd sikkert beundret Inge*	1
103	*Jeg trodde at Gyrd beundret sikkert Inge*	2
104	*Jeg trodde at Inge beundret Gyrd sikkert*	1
105	*Jeg trodde at du beundret sikkert Inge*	1
106	**Jeg trodde at du beundret Inge sikkert*	0
107	**Jeg trodde at Gyrd skjenket Inge en bok sikkert*	0
108	**Jeg sjekket om Gyrd beundret sikkert Inge*	0
109	**Jeg sjekket om Gyrd beundret Inge sikkert*	0
110	**Jeg trodde at sikkert Inge smilte*	0
111	*Gyrd smilte kanskje*	1
112	*Gyrd beundret kanskje Inge*	2
113	*Gyrd beundret Inge kanskje*	1
114	*Meg beundret du kanskje*	1
115	*Gyrd skjenket kanskje Inge en bok*	3
116	*Gyrd skjenket Inge kanskje en bok*	2
117	*Jeg trodde kanskje at Inge smilte*	2
118	*Du beundret kanskje Inge*	1
119	*Kanskje beundret Inge Gyrd*	1
120	*Beundret kanskje jenta en gutt*	1
121	*Beundret jenta kanskje en gutt*	1
122	**Du beundret Inge kanskje*	0
123	**Gyrd skjenket Inge en bok kanskje*	0
124	*Gyrd prøvde kanskje å smile*	1
125	*Gyrd trodde kanskje at Inge smilte*	2

126	*Gyrd hadde kanskje Inge skjenket en bok*	2
127	*Gyrd hadde Inge kanskje skjenket en bok*	2
128	*Gyrd hadde kanskje skjenket Inge en bok*	1
129	*Gyrd hadde kanskje trodd at Inge smilte*	2
130	**Gyrd hadde trodd kanskje at Inge smilte*	0
131	*Jeg trodde at Inge kanskje smilte*	1
132	*Jeg trodde at Inge kanskje beundret Gyrd*	1
133	*Jeg trodde at Gyrd beundret kanskje Inge*	2
134	*Jeg trodde at Inge beundret Gyrd kanskje*	1
135	*Jeg trodde at du beundret kanskje Inge*	1
136	**Jeg trodde at kanskje Inge smilte*	0
137	**Jeg trodde at du beundret Inge kanskje*	0
138	**Jeg trodde at Gyrd skjenket Inge en bok kanskje*	0
139	**Jeg sjekket om Gyrd beundret kanskje Inge*	0
140	**Jeg sjekket om Gyrd beundret Inge kanskje*	0
141	*Gyrd smilte allikevel*	2
142	*Gyrd beundret allikevel Inge*	2
143	*Gyrd beundret Inge allikevel*	3
144	*Gyrd skjenket allikevel Inge en bok*	3
145	*Gyrd skjenket Inge allikevel en bok*	2
146	*Jeg trodde allikevel at Inge smilte*	2
147	*Du beundret allikevel Inge*	1
148	*Du beundret Inge allikevel*	1
149	*Gyrd skjenket Inge en bok allikevel*	3
150	*Allikevel beundret Inge Gyrd*	1
151	*Beundret Gyrd allikevel Inge*	1
152	*Beundret Gyrd Inge allikevel*	1
153	*Gyrd prøvde allikevel å smile*	1
154	*Gyrd trodde allikevel at Inge smilte*	2
155	*Gyrd hadde allikevel Inge skjenket en bok*	2
156	*Gyrd hadde Inge allikevel skjenket en bok*	2
157	*Gyrd hadde allikevel skjenket Inge en bok*	1
158	*Gyrd hadde allikevel trodd at Inge smilte*	2
159	**Jeg skjenket Inge allikevel en bok*	0
160	**Gyrd hadde trodd allikevel at Inge smilte*	0
161	*Jeg trodde at Inge allikevel smilte*	1
162	*Jeg trodde at Inge allikevel beundret Gyrd*	1
163	**Jeg trodde at allikevel Inge smilte*	0
164	*Jeg trodde at Gyrd beundret allikevel Inge*	2
165	*Jeg trodde at Inge beundret Gyrd allikevel*	7
166	*Jeg trodde at du beundret allikevel Inge*	1
167	*Jeg trodde at du beundret Inge allikevel*	4
168	*Jeg trodde at Gyrd skjenket Inge en bok allikevel*	8
169	*Jeg sjekket om Gyrd beundret Inge allikevel*	2
170	**Jeg sjekket om Gyrd beundret allikevel Inge*	0
171	*Gyrd smilte ved bordet*	1

172	*Jenta beundret gutten ved bordet*	4
173	*Ved blomsten smiler jenta*	1
174	**Jenta beundret ved bordet gutten*	0
175	*Det sitter ei jente ved bordet*	1
176	*Sitter det ei jente ved bordet*	1
177	**Det snør det*	0
178	**Det sitter ei jente*	0
179	*Blomsten sitter jenta ved*	1
180	*Blomsten trodde jeg at jenta satt ved*	1
181	**Ved sitter jenta bordet*	0
182	*Ei jente smilte*	1
183	*En jente smilte*	1
184	*Ei dør smilte*	1
185	*En dør smilte*	1
186	*En dam smilte*	1
187	*En gutt smilte*	1
188	*Et fly smilte*	1
189	*Et hjem smilte*	1
190	**En hjem smilte*	0
191	**Et jente smilte*	0
192	**Et dør smilte*	0
193	**Ei dam smilte*	0
194	**Et gutt smilte*	0
195	**Ei fly smilte*	0
196	**En gutten smiler*	0
197	**Ei jenta smiler*	0
198	**Et flyet smilte*	0
199	*Jenten smiler*	1
200	*Gutten smiler*	1
201	*Dammen smiler*	1
202	*Barmen smiler*	1
203	**Flyen smiler*	0
204	**Hjemmen smilte*	0
205	**Jente smiler*	0
206	**Guttn smiler*	0
207	**Damen smiler*	0
208	**Barmmen smiler*	0
209	*Jenta smilte*	1
210	*Boka smilte*	1
211	**Gutta smilte*	0
212	**Flya smilte*	0
213	**Jentea smilte*	0
214	**Boa smilte*	0
215	*Hjemmet smilte*	1
216	*Flyet smilte*	1
217	**Guttet smilte*	0

218	**Hjemet smilte*	0
219	**Flymet smilte*	0
220	*Den jenta smilte*	1
221	*Den gutten smilte*	1
222	*Det flyet smilte*	1
223	*Dette flyet smilte*	1
224	*Denne jenta smilte*	1
225	*Denne gutten smilte*	1
226	**Det jenta smilte*	0
227	**Den flyet smilte*	0
228	**Det fly smilte*	0
229	**Det flyene smilte*	0
230	**Dette fly smilte*	0
231	**Dette flyene smilte*	0
232	**Dette jenta smilte*	0
233	**Denne flyet smilte*	0
234	**Denne gutt smilte*	0
235	**Denne gutter smilte*	0
236	**De flyet smilte*	0
237	*Gutter smilte*	1
238	*Jenter smilte*	1
239	*Lærere smilte*	1
240	*Dører smilte*	1
241	*Dammer smilte*	1
242	*Barmer smilte*	1
243	**Damer smilte*	0
244	**Dørere smilte*	0
245	**Laererer smilte*	0
246	**Barmmer smilte*	0
247	**Guttr smilte*	0
248	**Jenteer smilte*	0
249	**Hjemmer smilte*	0
250	**Hjemer smilte*	0
251	**Flyer smilte*	0
252	*Fly smilte*	1
253	*Hjem smilte*	1
254	**Gutt smilte*	0
255	*Noen gutter smilte*	1
256	*Noen fly smilte*	1
257	**Noen jente smilte*	0
258	**Noen guttene smilte*	0
259	*Alle jenter smilte*	1
260	*Alle jentene smilte*	1
261	**Alle jente smilte*	0
262	**Alle jenta smilte*	0
263	*Jentene smilte*	1

264	*Guttene smilte*	1
265	*Hjemmene smilte*	1
266	*Barmene smilte*	1
267	*Lærerene smilte*	1
268	*Flyene smilte*	1
269	*Husene smilte*	1
270	**Jenteene smilte*	0
271	**Guttne smilte*	0
272	**Hjemene smilte*	0
273	**Barmmene smilte*	0
274	*Disse jentene smilte*	1
275	*Disse flyene smilte*	1
276	*Disse guttene smilte*	1
277	*De flyene smilte*	1
278	*De guttene smilte*	1
279	*De jentene smilte*	1
280	**De jenter smilte*	0
281	**De gutt smilte*	0
282	**Det flyene smilte*	0
283	**Disse fly smilte*	0
284	**Det fly smilte*	0
285	**Disse disse flyene smilte*	0
286	*Ei blid jente smilte*	1
287	*En blid jente smilte*	1
288	*En blid gutt smilte*	1
289	*Et blidt fly smilte*	1
290	*Ei forelsket jente smilte*	1
291	*Ei blid god jente smilte*	1
292	*Et blidt godt fly smilte*	1
293	**Ei blid gutt smilte*	0
294	**En blidt gutt smilte*	0
295	**Et blid fly smilte*	0
296	**Blid jente smilte*	0
297	**Ei jente blid smilte*	0
298	*Den blide jenta smilte*	1
299	*Den blide gode gutten smilte*	1
300	*Det forelskede flyet smilte*	1
301	*Denne gode gutten smilte*	1
302	*Dette gode blide flyet smilte*	1
303	**Det god flyet smilte*	0
304	**Den blide jente smilte*	0
305	**Gode flyet smilte*	0
306	*Forelskede jenter smilte*	1
307	*Forelskede fly smilte*	1
308	*Blide jenter smilte*	1
309	*Blide gutter smilte*	1

310	*Blide fly smilte*	1
311	*Blide gode fly smilte*	1
312	*Blide gode blide jenter smilte*	1
313	**Forelskete gutter smilte*	0
314	**Forelskede jentene smilte*	0
315	**Forelskede jenta smilte*	0
316	*Noen blide jenter smilte*	1
317	**Noen blide jentene smilte*	0
318	*De blide jentene smilte*	1
319	*De blide gode flyene smilte*	1
320	*Disse gode guttene smilte*	1
321	*Disse forelskede guttene smilte*	1
322	**De blide jenter smilte*	0
323	**Disse blid jentene smilte*	0
324	**Blide jentene smilte*	0
325	*Ei jente ved bordet beundret gutten*	2
326	*Blide jenter ved bordet smilte*	1
327	*Jenta beundret gutten på blomsten*	4
328	*Beundret gutten på blomsten jenta*	1
329	**På blomsten gutten beundret jenta*	0
330	*Jeg beundrer deg*	1
331	*Du beundrer meg*	1
332	*Han beundrer henne*	1
333	*Hun beundrer ham*	1
334	*Vi beundrer dere*	1
335	*Dere beundrer oss*	1
336	*Dere beundrer dem*	1
337	*De beundrer dere*	2
338	*Hun sitter ved ham*	1
339	*Ved ham sitter hun*	1
340	*Henne beundrer hun*	1
341	*Meg beundrer du*	1
342	**Hun beundrer hun*	0
343	**Jeg sitter ved du*	0
344	**Oss beundrer oss*	0
345	*Jeg prøvde å kaste ballen*	1
346	*Jeg prøvde å blø*	1
347	**Jeg prøvde å kast ballen*	0
348	**Jeg prøvde å bløe*	0
349	*Jeg kaster ballen*	1
350	*Jeg blør*	1
351	**Jeg kastr ballen*	0
352	**Jeg bløer*	0
353	*Inge kastet ballen*	2
354	*Inge kasta ballen*	2
355	**Inge kastte ballen*	0

356	**Inge kastde ballen*	0
357	*Inge har kastet ballen*	1
358	*Inge har kasta ballen*	1
359	**Inge har kastt ballen*	0
360	**Inge har kastd ballen*	0
361	*Gyrd tapte*	1
362	**Gyrd tapet*	0
363	**Gyrd tapde*	0
364	*Gyrd har tapt*	1
365	**Gyrd har tapet*	0
366	**Gyrd har tapd*	0
367	*Jeg prøvde å blomstre*	1
368	**Jeg prøvte å blomstre*	0
369	**Jeg prøvdde å blomstre*	0
370	*Jeg har prøvd å blomstre*	1
371	**Jeg har prøvt å blomstre*	0
372	**Jeg har prøvdd å blomstre*	0
373	*Det snødde*	1
374	**Det snøde*	0
375	**Det snøte*	0
376	*Det har snødd*	1
377	**Det har snøt*	0
378	**Det har snød*	0
379	*Hun trodde jeg at beundret meg*	1
380	*Gyrd sjekker jeg om beundrer meg*	1
381	*Henne trodde jeg at du beundret*	1
382	*Henne sjekket jeg om du beundret*	1
383	**Henne trodde jeg at beundret Gyrd*	0
384	**Gyrd trodde jeg at meg beundret*	0
385	*hun trodde jeg at sikkert beundret meg*	1
386	*henne trodde jeg at jeg sikkert beundret*	1
387	**hun trodde jeg at beundret sikkert meg*	0
388	**henne trodde jeg at jeg beundret sikkert*	0

Finally, using the evaluation tool (called [incr tsdb()]; see Oepen and Callmeier (2000)) that is available as on add-on to the LKB system, here is a summary view of some interesting properties of the test set when aggregated by sentence length. The columns are, again from left to right, the total number of sentences in each group, the number of grammatical examples within the group, the average sentence length, the average number of analyses found by the parser, the total number of sentences for which at least one analysis can be found, and (finally) the resulting grammatical coverage.

'livel/mar-03' Coverage Profile						
Aggregate	**total items** ♯	**positive items** ♯	**word string** ϕ	**parser analyses** ϕ	**total results** ♯	**overall coverage** %
$8 \leq length < 10$	7	5	8.20	3.00	5	100.0
$6 \leq length < 8$	88	65	6.57	1.69	65	100.0
$4 \leq length < 6$	121	84	4.40	1.27	84	100.0
$2 \leq length < 4$	172	84	2.63	1.08	84	100.0
Total	**388**	**238**	**4.45**	**1.36**	**238**	**100.0**

(generated by [incr tsdb()] at 19-jun-2003 (16:22 h))

List of Figures

References

Lars Ahrenberg. A grammar combining phrase structure and field structure. In *Proceedings of COLING-90*, volume 1, pages 1–6, Helsinki, August 1990a.

Lars Ahrenberg. Topological fields and word order constraints. In *Edinburgh Working Papers in Cognitive Science*, volume 6, pages 1–19. University of Edinburgh, 1990b.

Lars Ahrenberg. Positions vs. precedence as primitives of constituent order. In *Saetningsskemaet i Generativ Grammatikk*. Institut for Erhvervssprøglig Informatik og Kommunikation, Syddansk Universitet – Kolding, 1999.

Kjell Johan Sæbø Anneliese Pitz. Kontrastive syntax norwegisch-deutsch. Kompendium für Deutsch 220, Germanistisk institutt, Universitetet i Oslo, Oslo, 1997.

Emily M. Bender, Dan Flickinger, and Stephan Oepen. The grammar matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammar. In *Procedings of the 19th International Conference on Computational Linguistics*, Taipei, Taiwan, 2002.

Gosse Bouma, Daniel Flickinger, and Frank van Eynde. Constraint-based lexica. In Frank van Eynde and Daffyd Gibbon, editors, *Lexicon Development for Speech and Language Processing*. Kluwer, Dordrecht, 2000.

John Carrol, Ann Copestake, Daniel Flickinger, and Victor Poznanski. An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the 7th European Workshop on Natural Language Generation (EWNLG'99)*, pages 86–95, Toulouse, 1999.

Ann Copestake. Semantic transfer for verbmobile. Draft, August 1995.

Ann Copestake. *Implementing Typed Feature Structure Grammars*. CSLI Lecture Notes. Center for the Study of Language and Information, Stanford, California, 2001.

Ann Copestake and Dan Flickinger. An open-source grammar development environment and broad-coverage english grammar using hpsg. In *Proceedings of the 2nd Linguistic Resources and Evaluation Conference*, pages 591–600, Athens, Greece, 2000. ELDA.

Ann Copestake, Dan Flickinger, Rob Malouf, Susanne Riehmann, and Ivan Sag. Translation using minimal recusion semantics. In *Proceedings of the Sixth International Conference on Theoretical and Methodological Issues in Machine Translation (TM195)*, Belgium, 1995. Leuven.

Ann Copestake, Dan Flickinger, and Ivan A. Sag. Minimal recursion semantics: An introduction. Manuscript, Stanford University: CSLI, 1999.

Ann Copestake, Alex Lascarides, and Dan Flickinger. An algebra for semantic construction in constraint-based grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)*, Toulouse, France, 2001.

Paul Diderichsen. *Elementær dansk grammatik*. Gyldendal, København, 3 edition, 1962.

Rolf Theil Endresen, Hanne Gram Simonsen, and Andreas Sveen, editors. *Innføring i lingvistikk*. Universitetsforlaget, Oslo, 1999.

Hans-Olav Enger and Kristian Emil Kristoffersen. *Innføring i norsk grammatikk : morfologi og syntaks*. Landslaget for norskundervisning : Cappelen akademisk forlag, Oslo, 2000.

Jan Terje Faarlund, Svein Lie, and Kjell Ivar Vannebo. *Norsk referansegrammatikk*. Universitetsforlaget, Oslo, 1997.

Dan Flickinger, Carl Pollard, and Thomas Wasow. Structure sharing in lexical representations. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 180–187. Association for Computational Linguistics, 1985.

Daniel Flickinger. *The Hierarchical Lexicon*. PhD thesis, Stanford University, 1987.

G. Gazdar, E. Klein, G. K. Pullum, and I. A. Sag. *Generalized Phrase Structure Grammar*. Basil Blackwell, Oxford, 1985.

Gerald Gazdar and Christopher Mellish. *Natural Language Processing in PROLOG: an Introduction to Computational Linguistics*. Addison-Wesley Publishing Company, 1989.

Jonathan Ginzburg and Ivan A. Sag. *Interrogative Investigations: the form, meaning, and use of English Interrogatives*. CSLI Publications, Stanford, California, 2001.

Kjell Tørres Heggelund. *Setningsadverbial i Norsk*. Number 2 in Tromsøstudier i språkvitenskap. Novus Forlag, Oslo, 1981.

Lars Hellan and Torbjørn Nordgård. Invited tutorial on norwegian grammar: Challenges for hpsg. In *The Proceedings of the 7th International Conference on Head-Driven Phrase Structure Grammar*, pages 130–146, Berkeley, 2000. CSLI Publications.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice Hall, 2000.

Alex Lascarides and Ann A. Copestake. Default representation in constraint-based frameworks. *Computational Linguistics*, 25(1):55–105, 1999.

Robert Malouf, John Carroll, and Ann Copestake. Efficient feature structure operations without compilation. *Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):29 – 46, 2000.

Stefan Müller. *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche.* Number 394 in Linguistische Arbeiten. Max Niemeyer Verlag, Tübingen, 1999.

Stefan Müller. Continuous or discontinuous constituents? In *Proceedings of the ESSLLI-2000 Workshop on Linguistic Theory and Grammar Implementation*, pages 133–152, Birmingham, UK, August 14–18, 2000.

Anne Neville and Patrizia Paggio. Developing a danish grammar in the grasp project: A construction-based approach to topology and extraction in danish. Manuscript, 2001.

Øystein Nilsen. *The Syntax of Circumstantial Adverbials.* Number 21 in Tromsø Studies in Linguistics. Novus Press, 2000.

Stephan Oepen and Ulrich Callmeier. Measure for measure: Parser cross-fertilization - towards increased component comparability and exchange. In *Proceedings of the 6th International Workshop on Parsing Technologies*, Trento, Italy, 2000.

Stephan Oepen and John Carroll. Parser engineering and performance profiling. *Natural Language Engineering*, 6(1):81–97, 2000.

Thorbjørn Nordgård og Tor Anders Åfarli. *Generativ syntaks : ei innføring via norsk.* Novus, Oslo, 1990.

Gerald Penn. *The Algebraic Structure of Attributed Type Signatures.* PhD thesis, School of computer science, Carnegie Mellon university, 2002.

Christer Platzack. A survey of generative analyses of the verb second phenomenon in germanic. *Nordic Journal of Linguistic*, pages 49–73, 1985.

Carl J. Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics.* Number 13 in CSLI Lecture Note Series. Stanford University: CSLI Publications, Chicago, 1987.

Carl J. Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar.* University of Chicago Press, Chicago, 1994.

Ivan A. Sag and Thomas Wasow. *Syntactic Theory: A Formal Introduction.* Center for the Study of Language and Information, Stanford, 1999.

Peter Sells. Scandinavian clause structure and object shift. In Miriam Butt and Tracy Holloway King, editors, *Proceedings of the LFG98 Conference*, Brisbane, 1998. The University of Queensland, CSLI Publications.

Peter Sells. Negation in swedish: Where it's not at. In Miriam Butt and Tracy Holloway King, editors, *Proceedings of the LFG00 Conference*, Berkely, 2000. University of California, CSLI Publications.

Peter Sells. *Structure, alignment and optimality in Swedish.* Stanford monographs in linguistics. CSLI publications, Stanford, California, 2001.

Nancy L. Underwood. A typed feature-based grammar of danish. *Nordic Journal of Linguistics*, 20(1):31–62, 1997.