Project Number LRE 61029: Large Scale Grammars for EC Languages
Deliverable E-D8-DK

Documentation of the Danish Lingware

Bradley Music & Costanza Navarretta

July, 1996

Keywords:
ALEP
NLP
implementation
Danish grammatical analysis

LSGRAM Deliverable E-D8-DK July, 1996 (final version) Printed July 30, 1996

Abstract

This is the eighth and final deliverable from the Danish group for the project **LRE 61029, "Large-Scale Grammars for EC Languages"**.

This document describes the implementation of lingware for Danish linguistic analysis done in ALEP. Major aspects of the approach are compared with HPSG, technical details are given including the TFS and processing declarations used, requirements for lexical coding are specified, and the implementation of text handling, morphological analysis, phrase structure analysis and PAS is described.

Authors:
Bradley Music and Costanza Navarretta
Center for Sprogteknologi
Njalsgade 80
DK-2300 Copenhagen S

Contents

Foreword

This is the final documentation and guide for the Danish LSGRAM implementation in ALEP. Although a project-specific deliverable, it also provides useful information on implemented approaches to areas of NLP of Danish for developers other than those who have been directly involved in LSGRAM work who are interested in practical, technical approaches to text handling, morphology, syntax and/or argument structure, especially as implemented in ALEP.

Some familiarity with Two-Level Morphology (TLM) will be assumed for some sections, while an acquaintance with HPSG concepts will make the introductory remarks, the data structures, and schemata descriptions more accessible.

Sections are presented procedurally, describing files or sets of files containing declarations, lexical entries or grammar rules with an overall organization corresponding essentially to the processing sequence.[1]

The introduction, Chapter 1, describes briefly the LSGRAM project as a whole and the Danish LSGRAM results in particular. General issues concerning the implementation approach, with its similarities with and deviations from HPSG, are also addressed as part of the introduction, and occasionally touched upon within the remaining chapters.

Chapter 2 describes declarations used, both linguistic (the makeup of the `sign`) and non-linguistic (affecting processing, e.g. lookup keys, paths).

Chapter 3 briefly describes the implemented coverage at the time of writing.

Chapter 4 describes text-handling, including a description of the locally developed tagging program for identifying difficult word constructs.

Chapter 5 covers the lexicon organization and coding of the lexical signs. The approach using ALEP is heavily oriented towards lexicalized information, and there is a separate lexicon for each major processing phase (morphological analysis, syntactic analysis, semantic analysis, though not for text handling).

Chapter 6 covers the morphographemic and morphosyntactic processing done during the word segmentation and lifting phases, respectively.

Chapter 7 details the rules used during the analysis phase of processing, comprising both word structure and phrase structure.

Chapter 8 describes refinement, the stage of processing where a syntactic structure is enriched with semantic information, such as identification of arguments and manipulation of quantifier and restriction lists.

Although the authors have worked closely throughout the major implementation phases of the project, each has focussed on particular areas of implementation and documentation. Bradley Music implemented declarations, text handling, morphology, word structure, non-lexical refine-

[1] Despite this attempt at ordinal perspicuity, the document has a probable destiny as a source of reference than as flowing prose (though in the unlikely latter case, please to consider it non-fiction).

ment, and some phrase structure rules, and wrote the following Chapters and Sections: 1, 2, 3, 4, 5.1, 5.2, 6, 7.1, and 8.4. Costanza Navarretta implemented lexical entries for analysis and refinement and many of the phrase structure rules, and wrote Sections 5.3 and 5.4, and most of Chapters 7 and 8 (excepting the sections just mentioned).

As regards formatting, reference to HPSG types and attributes will be done with SMALL CAPS. Type names and attribute-value pairs in the Danish implementation are shown using a `teletype font`, while attribute values mentioned without their corresponding attribute names are give in *italics*. Finally, within examples of complex feature structures, the Danish letters *æ, ø, å* are rendered as *ae, oe, aa*, respectively.

The authors would like to thank Bjarne Ørsnes for his valuable comments on HPSG and the refinement specifications during the writing of this report.

Chapter 1

Introduction

1.1 The LSGRAM project

A major goal of the EU-funded LSGRAM project was to engineer theoretically well-founded implementations which function efficiently, demonstrate the functionality of ALEP, and can be used as the foundation for training, research and applications. The project stressed high-level properties of lingware design and implementation, including modularity, main stream linguistic analysis, extensibility, robustness, standardization, as well as thorough testing and documentation.

Three language groups formed the initial group (Spain, England, Germany; *LSGRAM Core*), while 6 new groups joined later (Denmark, Portugal, Greece, France, Holland, Italy; *Extended LSGRAM* or *LSGRAM+*). The result is parallel grammar implementations for 9 separate European languages, based on a common platform, with some extensions to the functionality of the system designed and implemented by the LSGRAM groups.

The following points summarize the results of the LSGRAM project as a whole. Although there is some overlap, each is in itself an important result of the LSGRAM effort.

- **migration**: Existing linguistic resources have been reused (migrated) to the ALEP formalism, including significant lexical resources. As a result, the Danish implementation has a lexical coverage of thousands of base forms of common vocabulary.
- **new resources**: Significant linguistic resources have been generated by LSGRAM which are
  - **well-tested** using standard test suites developed within the project as a whole
  - **well-documented** via the report series
  - based on a **standard platform**
  - based on a **modern linguistic approach**, being HPSG inspired, though with modifications mostly having to do with implementational efficiency
  - **freely available** as a research result.
- **standardization**: The existence of these resources promotes standardization, besides encouraging use of ALEP for other languages. All LSGRAM groups had success with the formalism and platform, and all wish to continue the significant developments already made.
- **user-group**: More and more groups are interested in ALEP, such that although the LSGRAM project has ended, the ALEP user group is expanding.

- **training**: The project has trained and reinforced the expertise of an EU-wide network of experts in grammar development, and specifically in the platform and formalism using the chosen linguistic approach. This is in line with programme goals of dissemination of state-of-the-art NLP technology. The training effect is especially significant for those countries who had not developed significant resources before the start of the project.
- **testing, extension and demonstration of ALEP**: The LSGRAM project has been a fertile testing-grounds for the ALEP system, resulting in feedback which has contributed to a current state of the system which is light years ahead of its state at the start of the project. The system receives good grades for its extensibility and modularity, as well as the possibilities for developers to interact with the processing algorithms to implement implementation-specific optimizations.
- **feasibility** of unification-based approach to NLP: The success of the project should contribute to further development of unification-based approaches to NLP. This is significant, in that it is still widely held that unification-based approaches fail to provide the efficiency needed for serious processing applications. We think the results, particularly those of the German group, give credence to further development scenarios.
- **foundation** for future projects: this is not a throw-away project, but the culmination of a significant and cost-effective investment in resources, both software and lingware. The results are especially reusable because of their modularity, and of course, in particular in the event ALEP becomes an even more widely accepted standard.

1.2 The Danish LSGRAM

The following list of deliverables from the Danish LSGRAM group traces the course of the project:

E-D1-DK *LSGRAM - Danish Design* (Music 1995a)
preliminary design of the Danish implementation

E-D2-DK *Danish Corpus Analysis and Priority List* (Povlsen et al. 1995a)
corpus analysis and initial priority list of phenomena to be implemented

E-D3-DK *Type System and Lexicon Specifications for the Danish Core Grammar* (Braasch & Jørgensen 1995)
initial type system and macro designs for the implementation

E-D4-DK migrated lexical resources

E-D5-DK *Danish Morphology in ALEP* (Music 1995b)
implementation and documentation of morphological analysis

E-D5-DK (supplement) *Tagging Messy Details in ALEP* (Music 1995c)
implementation and documentation of the messy details tagger

E-D6-DK *Design of Syntax Implementation for Danish Phrase Structure and Predicate Argument Structure* (Navarretta 1996)
discussion of source material relative to the planned implementation of syntax and PAS

E-D7-DK final implementation

E-D8-DK *Documentation of the Danish Lingware* (Music & Navarretta 1996)
final documentation of the Danish core grammar implementation (this document)

Deliverable E-D2-DK defined a priority list for coverage based on a corpus analysis of patent documents, the chosen domain. This list was later updated as a result of deliverable E-D6-DK, where material from the LINDA project (see below) and other sources was reviewed and where the implementation could be planned based on concrete specifications.

Deliverable E-D4-DK resulted in a program for migrating from a format used by Eurotra to an ALEP format consistent with the then current Danish TFS defined by E-D3-DK, resulting in over 4000 lexical entries for morphological analysis. The program was later modified to generate refinement entries.

Deliverable E-D5-DK outlines the implementation of Danish morphology. Special use of lifting rules was implemented, such that morphotactics could be done before structural analysis began (see Chapter 6, esp. Section 6.2.2).

As the corpus analysis of deliverable E-D2-DK revealed a large number of messy details (difficult word constructs such as patent specifications, codes, dates, etc.), work was also done in this period on a tagger using regular expression pattern matching for spotting and marking these phenomena. This was implemented as an `awk` program and integrated with ALEP, documented as the supplement to deliverable E-D5-DK.

The final implementation phase of the project took deliverable E-D6-DK as its guidelines, with inspiration from HPSG (mostly Pollard & Sag (1994)), resulting in the present system.

1.3 MLAP and LSGRAM

The specification reports developed by the Danish MLAP project *LINDA* were used directly as a source material for much of the LSGRAM implementation (Neville & Povlsen 1995, Pedersen et al. 1995, Povlsen et al. 1995b, Underwood & Jørgensen 1995). Given that LSGRAM is an implementation project, processing considerations have taken priority when in conflict with a theoretical stand involving significant inefficiency. Some differences have to do with technical possibilities for optimization which are irrelevant for the specifications, such as co-representation of heads (see Section 7.2). Other differences relate to the fact that LINDA, like HPSG, assumes a single processing level, while ALEP provides three processing levels. This has been significant for instance wrt. how much information need be expressed within lexical entries at each level, and where ambiguities are introduced.

1.4 HPSG and LSGRAM

Like the LINDA project, the implementation has taken HPSG as a starting point. By referring to and comparing with HPSG approaches, we hope to make this documentation more generally accessible, since familiarity with HPSG within the NLP community (at least currently) can be taken as a given. This should in no way be interpreted as a promise to conform to HPSG as far as possible, only as a common reference to be used as a means to promote understanding of what has been done.

The remainder of this section describes the relation between key HPSG concepts and the implementation.

1.4.1 Linear precedence (LP) rules

ALEP provides no means of expressing LP rules. Instead, LP and ID are expressed via phrase structure (PS) rules ('syntactic backbone'). ID can be expressed without LP, since there is the

possibility of expressing that daughter nodes in a rule are randomly ordered, effectively overriding the strict LP that is the default. This feature has been used to reduce the number of rules during structural refinement (see Section 8.4).

1.4.2 Principles

Neither grammar formalisms nor implementations need to follow principles, however the more licentious a theoretical or practical approach is, the less interesting its solutions can be for others. The principles anchor the approach, defining limitations and providing a basis for comparison between different approaches.

General but powerful principles are the ideal anchors, providing clear guidelines useful for understanding grammatical relationships. Establishing a few, vague principles is only little better than unrestricted behavior, while a multitude of very specific principles tends to be less generally applicable (and less interesting) between implementations.

Below we outline principles defined here as general constraints on feature structures. These have been developed for the Danish LSGRAM implementation based on those presented in Pollard & Sag (1994). Since phenomena treated by most of the principles within that seminal work have yet to be implemented, only a minority of the principles (approximately one third) are relevant for the current Danish implementation. Of these, the reader will find a general consistency with HPSG. Deviations wrt. Pollard & Sag (1994) are mostly computationally motivated, although some represent generalizations of HPSG principles (see the Marking Principle and the Selection Principle).

An overall cosmetic difference has to do with explicitness. Although ALEP actually has a facility for establishing correspondances (e.g. token-identity) relatively implicitly as part of parse head declarations, correspondances cannot be established there between mother nodes and non-head-daughters, making it impossible to implement all principles via such declarations. All principles within the Danish implementation are made explicit within applicable PS rules.

In Pollard & Sag (1994), "a *headed phrase* is a *phrase* whose DAUGHTERS value is of sort *headed-structure*." (p.399) The LSGRAM equivalent of this is that a *headed phrase* is one where the `head|major` values of the mother node and one of the daughter nodes are token-identical. In fact, every PS rule specifies a headed phrase.

- **Head Feature Principle**

  HSPG: Mother and head-daughter HEAD values are token-identical.

  LSGRAM: Mother and head-daughter `head|major` values are token-identical.

  The change reflects the restructuring of the `head` feature such that it now contains an attribute `major` and the marking features `spr` (see Chapter 7) and `marking` (see **Marking Principle** below). Moving marking features to within the `head` value makes subcategorization based on co-represented structures more efficient by avoiding having to implement a similar, but much more unwieldy structure at the `cat` level (see Chapter 5).

- **Subcategorization Principle**

  HPSG: In a head-complement construction, the SUBCAT value of the head-daughter node is the concatenation of the SUBCAT values of the complement **nodes** with the SUBCAT value of the mother node.

  LSGRAM: In a head-complement construction, the `compls` value of the head-daughter node is the concatenation of the `compls` values of the complement **node** with the `compls` value of the mother node.

The name change from SUBCAT to `compls` is merely cosmetic. The LSGRAM version assumes only a single complement is found by application of a head-complement rule (HPSG allows PS rules spanning a head and several complements), making it possible to interleave adjuncts and complements (see Chapter 7).

- **ID Principle**

  HPSG: Every headed phrase must satisfy exactly one of the ID schemata.

  LSGRAM: Every *grammatical* headed phrase must satisfy exactly one of the ID schemata.

  Assuming that no two schemata are unifiable, the HPSG principle is a truism — even if a given input satisfies two different schemata, one would not claim that the two results were the same *headed phrase* since there would be at least one inconsistent attribute value between them.

  In LSGRAM terms, since each schema is implemented as a set of PS rules, and since something that does *not* match a PS rule cannot be parsed, satisfaction either occurs, resulting in a head phrase, or parsing fails. By making the assumption of grammaticality explicit, this principle can be used to define what is grammatical (i.e. that which matches one of the PS rules (and thereby a schema)) and what is ungrammatical (i.e. matching no PS rule). This has little significance in the current implementation, but will in future versions with robust parsing, where a parse result must be constructed which possibly matches no PS rule.

- **Marking Principle**

  HPSG: In a headed phrase, the MARKING value is token-identical with that of the MARKER daughter if any, and with that of the HEAD-DAUGHTER otherwise.

  LSGRAM: In a headed phrase, the `marking` value is token-identical with that projected by the *selecting* daughter if any, and with that of the *head-daughter* otherwise.

  Another way to state this is that projections can receive a new `marking` value (relative to the head) from anything that can select, viz. specifiers, modifiers, and markers, otherwise they retain their `marking` value. Note that head selection by marking elements is consistent with HPSG. (See the Selection Principle below.)

  The extension of the concept of marking used here makes it possible to exploit the `marking` feature for controlling configurational phenomena of specifying and modifying elements. In Danish, the so-called nexus adverbials may only occur in a given order, and likewise for determiners (Neville & Povlsen 1995).

  The formulation "projected by" is used since it is not actually the specifying/modifying element's marking value that the mother receives, but rather the value of the feature `newmarking`, used exclusively for projecting a marking value. This is based on the observation that it makes little sense for marking elements to be themselves marked, rather they are the source of a new marking for the resulting structure, whence the feature name.

  This distinction means that within an analysis structure, only those nodes actually marked contain a value for the feature `marking`, giving a more perspicuous structure.

- **SPEC Principle (LSGRAM: Selection Principle)**

  HPSG: If a nonhead-daughter in a headed structure bears a SPEC value, it is token-identical to the SYNSEM value of the head daughter. (p.51)

  LSGRAM: If a non-head-daughter in a headed structure bears a (non-nil) `selects` value, it is token-identical to the `synsem` value of the head-daughter.

Although superficially this looks like just a name change, it represents a significant generalization to the HPSG principles, consolidating selection by specifiers, modifiers and markers. Like the generalization of marking above, using a single feature here is a logical step, since specifiers, modifiers and markers are non-overlapping sets which all select their heads.

The distinction between head selection by modifiers and specifiers/markers in HPSG is expressed via the feature names, viz. MOD and SPEC, respectively. In the LSGRAM approach, it is the value of `selects` that is the distinguishing feature (see the type declarations in Section 2.1.9, and Section 7.2.1 for more on the approach).

- **Raising Principle**

  HPSG: An element X within the SUBCAT list L of a lexical item E has no role in the CONTENT value of E if and only if L contains another nonsubject element whose SUBCAT list contains X.

  LSGRAM: For a lexical item L allowing raising, the `content` value of a complement (`subj` or `compls` element) is token-identical with the `content` value of a complement of another element within the `compls` list of L.

  The HPSG approach is a general contraint which would be computationally costly. For this reason, LSGRAM has implemented raising within specific lexical entries, though the result, i.e. the effect on the `content` value, is the same.

- **Content Principle**

  HPSG:
  Definition of semantic heads: The semantic head of a headed phrase is:
  1. the adjunct daughter in a head-adjunct structure,
  2. the head-daughter otherwise.

  CONTENT PRINCIPLE:
  In a headed phrase, **Case 1** if the semantic head's CONTENT value is of sort psoa, then its NUCLEUS is token-identical to the NUCLEUS of the mother;
  Case 2 otherwise, the CONTENT of the semantic head is token-identical to the CONTENT of the mother.

  LSGRAM:
  Definition of semantic heads: The semantic head of a headed phrase is always the head-daughter.

  CONTENT PRINCIPLE:
  In a headed phrase:
  Case 1: In *head-specifier constructions* and *head-adjunct constructions with a quantifier adjunct*, all information within `content` other than `quants` is token-identical between the mother node and the semantic head.
  Case 1a: In a *possessive head-specifier structure*, the quantifier list `quants` of the mother is the concatenation of a definite quantifier force[1] and the `quants` value of the semantic head. The `possessor` value within the semantic head is token-identical to the `content` value of the specifier.
  Case 1b: In a *non-possessive head-specifier structure* or a *head-adjunct constructions with a quantifier adjunct*, the quantifier list `quants` of the mother is the concatenation of the content of the `quants` lists of the specifier or adjunct and the semantic head.

[1] `quantifier:{q_force=>defin}`

> **Case 2**: In *head-adjunct constructions with a non-quantifier adjunct*, all information within `content` other than `restr` is token-identical between the mother node and the semantic head.
> **Case 2a**: In such a head-adjunct construction with a *nominal adjunct*, the restrictions list `restr` of the mother is the concatenation of the `content` value of the adjunct and the `restr` value of the semantic head.
> **Case 2b**: In *all other head-adjunct constructions*, the restrictions list `restr` of the mother is the concatenation of the `restr` values of the adjunct and the semantic head.
>
> **Case 3**: In all *constructions other than head-specifier and head-adjunct*, the `content` of the semantic head is token-identical to the `content` of the mother.

The Content Principle as given in HPSG (Pollard & Sag 1994) was not found to be adequate for the needs of the implementation. One difficult point was the fact that nominals can function as temporal adjuncts, as in *Jeg har møder hele dagen.* (I have meetings all day long.). In the HPSG approach, this would mean that every lexical entry for nominals possibly functioning as adjuncts (here, *dag* (day)) would have to be doubled in order to express their two possible semantics, and that every possible modifier of those nominal adjuncts would have to take into account that their semantics could have two completely different structures. This then would have caused an untenable explosion in refinement lexical entries.

We also found it odd that pre- and post-quantifiers were semantic heads while central quantifiers as specifiers were not, and that prepositional phrases modifying a nominal should have a different content (i.e. semantic) structure from exactly the same prepositional phrases modifying a clause.

In the present implementation, head-daughters are also semantic heads, while information from adjuncts is added to the restrictions list of the modified head via structural refinement rules (see Section 8.4). In general this solution is more flexible, allowing different word classes to act as adjuncts and the same word class to modify different semantic heads without introducing multiple lexical entries in refinement. Although in the present implementation we have only implemented lexical entries for adjuncts modifying nominals and clauses, the implementation easily allows other singleton adjunct entries to be added for modifying adjectives, adverbs, non-saturated verbs and prepositional phrases.

In the head-specifier constructions of Case 1a, the specifier can be either a genitive construction or a possessive pronoun. The definite quantifier force is added here since it is not considered lexical information.

The selection of semantic heads and implementation of the Content Principle are done in the refinement phase within a special set of rules (see Section 8.4).

- **Cohead Feature Principle**

  (This is a new principle introduced by LSGRAM.)

  > LSGRAM: Within all signs, the `head` value and the corresponding `cohead` attribute value are token-identical.

  By *corresponding* is meant the `cohead` attribute which matches the part-of-speech of the sign in question; in other words, the `cohead` attribute which is unifiable with `head`.

  Technically, this is accomplished by ensuring this token-identity within lexical tokens (via lifting rules) and within all mother nodes (via variables within PS rules), respectively. Where the `head` value is simply percolated, the `cohead` value can also be percolated, maintaining the identity via the head-daughter node. A special case of this is where the mother and head-daughter `cat` values are token-identical.

- **NONLOCAL Feature Principle**
  Relative Uniqueness Principle
  Singleton REL Constraint
  Clausal REL Prohibition

  The NONLOCAL feature is not currently used in the LSGRAM implementation.

- **Principles A-C (Binding Theory)**

  No binding theory is currently implemented in LSGRAM.

- **Subject Condition**
  Trace Principle
  Slash Inheritance Principle
  Slash Termination Metarule

  No unbounded dependencies are currently implemented in LSGRAM.

- **Weak Coordination Principle**

  Coordination is not implemented in LSGRAM.

- **The Control Theory**

  This level of semantics (e.g. experiencer, commitment, orientation) is not implemented in LSGRAM.

- **Semantics Principle**
  Quantifier Binding Condition
  Quantifier-Inheritance Principle
  Scope Principle

  Quantifier scope is not implemented in LSGRAM.

- **Principle of Contextual Consistency**

  > HPSG: The CONTEXT|BACKGROUND value of a given phrase is the union of the CONTEXT|BACKGROUND values of the daughters.

  This is not implemented in LSGRAM.

1.4.3 Schemata

Schemata have been implemented as a union of a set of functionally similar PS rules. Examples of the rules themselves are given in Chapter 7. The schema a rule belongs to is marked within parse results using the attribute `syn|str|contr`.

The following HPSG schemata are implemented as sets of PS rules in LSGRAM (see Section 7.3):

- **Head-Subject Schema**
- **Head-Complement Schema**
- **Head-Subject-Complement Schema**
- **Head-Marker Schema**
- **Head-Adjunct Schema**
- **Head-Specifier Schema**

The HPSG **Head-Filler Schema** has yet to be implemented.

In addition to these, schemata for word structure have been implemented. These comprise the following (see Section 7.1):

- **Word Schema**
- **Compound Schema**

Chapter 2

Declarations

This chapter describes the data types used within the implementation for controlling both linguistic and non-linguistic processing.

Linguistic data types include what is usually thought of as the core of the TFS, defining what a linguistic sign consists of.

Data types affecting non-linguistic processing include style and head declarations, paths, lookup keys, and specifiers for partitioning the grammars and lexica.

A convention used in naming types is to begin their name with the letter `t`, with the exception of the topmost type `sign` and the types within `tsem`, the names of which have been determined jointly within the entire LSGRAM project.

2.1 The typed-feature system (TFS)

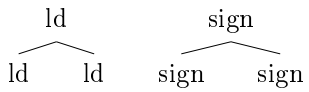
2.1.1 type `sign`

```
type(
sign:{
        procinfo        => type({tprocinfo:{}}),
        ortho           => type({tortho:{}}),
        synsem          => type({tsynsem:{}}),
        nonlocal        => type({tnonlocal:{}})
        },'top-level structure (ld)').
```

A *sign* is a complex feature structure containing complete information for describing a linguistic unit within the implementation. This information consists for the most part of attribute-value pairs of linguistic relevance, but also includes information controlling processing, such as lookup keys.

Other groups have chosen to use the default type `ld` as the topmost type within their implementations. Type `ld` then typically contains, e.g., the attributes `sign` and `procinfo`. The goal here is to separate non-linguistic (i.e. specifiers for grammar/lexicon partitioning) information out of the sign.

However PS rules must be written with daughter nodes expressed as instances of the topmost sign, such that in the alternative approach PS rules are of the form on the left, while the Danish rules are of the form on the right:



Although the named specifiers are not part of type `sign`, the left-hand rule format does not succeed in separating out other non-linguistic information, since LP and ID specifications as represented within such a PS rule are by nature linguistic, and all `ld` nodes still contain the named specifiers.

2.1.2 type `tprocinfo`

```
type(
  tprocinfo:{
    language => atom({ da }),
    unit_num => atom,
    parsehead=> atom({y}),
    specinfo => type({tspecinfo:{}}),
    ruleinfo => type({truleinfo:{}})
      }, '').
```

Information within the typed-feature system (TFS) affecting processing has been collected with the type `tprocinfo`. `language` is a language specification which is used as part of grammar and object partitioning. `unit_num` is the position in the input sequence of the `sign`, i.e. its value is an ordinal representing the number of the sentence within the input document. The attribute `parsehead` is used to indicate to the parser which daughter node to examine first during parsing, i.e. the one with `parsehead=>y`. This makes it easy to choose the parse head directly from within the PS rules by setting this feature within a daughter node. (See also Section 2.2.1.)

```
type(
  tspecinfo:{
    tlm => atom({ y,n }),
    lift => atom({ y,n }),
    ana => atom({ default,lex,ws,ps,n }),
    ref => atom({ default,y,n }),
    synth => atom({ y,n }),
    lower => atom({ y,n }),
    partition => type({tpart:{}})
      }, '' ).

type(
  tpart:{
    main => atom,
    sub  => atom
      }, '' ).
```

Type `tspecinfo` contains the so-called named specifiers for the grammar set in question. Not to be confused with linguistic specifiers, these attributes are used to partition the grammar rules and lexical entries in order to indicate to the parser which rules may be applied at a given point within the input structure, and at a given point during the parse process. For example, during TLM analysis, syntactic analysis and refinement, only lexical entries and phrase structure rules

explicitly containing non-*n* values for `tlm`, `ana` and `ref`, respectively, are applied. This is termed *vertical partitioning*.[1]

At the analysis phase, *horizontal partitioning* is also implemented to give a finer (and more efficient) partitioning to the lexicon and grammar. This is done by allowing several possible values for the feature `ana`, viz. the values `ana=>lex, ana=>ws` and `ana=>ps` are assigned to lexical entries, word structure rules and PS rules, respectively, to be applied during analysis. The special value `ana=>default` indicates lexical entries to be applied by default in cases where no lexical entry with `ana=>lex` can be found which unifies with the input token.

In order for this partitioning to be effective, the system must be told at what point during the processing of a given input each `ana` value is applicable. Since the parser works bottom-up, the initial value to be used must be `ana=>lex`, which is indicated via the relevant `us_environment` object in ALEP (see relevant ALEP documentation) as the initial value to be used when searching for rules to unify with terminal nodes. To make the system 'switch' to the other specifier values, they must be included within the input itself at the appropriate points. So for example, to indicate at the word structure level that only word structure rules (i.e. those with `ana=>ws`) should be applied, `ana=>ws` is set during lifting within all nodes at the word level. Similarly, the topmost input node is assigned `ana=>ps` during lifting to force switching of the specifier to PS rules when parsing at that level.

The Danish implementation has the additional partitioning attribute `partition`. This is because the specifiers are applied to entries and rules using subsumption, not unification, which when doing lexical looking can be less efficient. A main partition and a subpartition are defined: the main partition is normally set to the part-of-speech of the lexical entry (for word and phrase structure rules, it is set to *ws* and *ps*, respectively), while the subpartition value is set to the `lu` value. This makes for very finely grained partitioning.

```
type(
  truleinfo:{
    tlm_id => atom,
    lift_id => atom,
    ana_id => atom,
    ref_id => atom,
    synth_id => atom,
    lower_id => atom
      }, '' ).
```

The type `truleinfo` contains features for indicating which rules have been applied to a node. Each lexical entry and grammar rule (though not TLM rules, since several can be applied to a single input token) has a unique `id` value.

2.1.3 type `tortho`

```
type(
  tortho:{
    string          => list(atom),
    rest            => list(atom)
      },'').
```

Type `tortho` is used to show what the input is corresponding to each structure node. Variable binding within lifting and syntax rules causes the attribute `string` to end up with the morpholog-

[1] This is a slight overgeneralization; since there is no structure created during TLM analysis, no PS rules have the feature `tlm=>y`.

ically processed input string corresponding to the input starting at whatever node the attribute occurs in until the end of the input. The attribute `rest` contains that part of the input which is *not* dominated by the current node. Thus when examining a node within an output structure, by substracting the value of `rest` from that of `string` one can identify the input sequence which is dominated by the node.

2.1.4 type tsynsem

```
tsynsem_opt_compl >
  {
  tsynsem_extracted,
  tsynsem
    }.

type(
  tsynsem_opt_compl:{
    syn         => type({tsyn:{}}),
    sem         => type({tsem:{}})
      },'').

type(
  tsynsem_extracted:{},'').

type(
  tsynsem:{},'').
```

Declaration of `tsynsem` in this way makes it possible to use the types themselves for handling optional complements. `tsynsem_opt_compl` is only used within lexical items themselves, and only for complements given within the `compls` list that are optional. Obligatory complements and all non-complement occurrences of a synsem-structure are specified as being of type `tsynsem`. Only complements within the `compls` list which may be of type `tsynsem_extracted`, i.e. only those of type `tsynsem_opt_compl`, can be extracted. (See Chapter 7 for more on this.)

2.1.5 type tsem

```
type(
  tsem:{
        content         => type({content:{}}),
        context         => type({context:{}})
        },'semantic feature structure').
```

A common specification of the type `tsem` has been defined within the LSGRAM project, and is documented in Theofilidis et al. (1994).

2.1.6 type tsyn

```
type(
  tsyn:{
    cat      => type({
        tpunct_cat:{head=>tpunct_head:{
                      major=>tpunct_major:{pos=>punct,selects=>tmarkee:{}}}},
```

```
        tfunct_cat:{head=>tfunct_head:{
                      major=>tfunct_major:{pos=>(marker;explet)}}},
        tsubst_cat:{head=>tsubst_head:{
                      major=>tsubst_major:{
                        pos=>(n;pron;v;art)}}},
        tsubst_cat:{head=>tsubst_head:{
                      marking=>unmarked,
                      major=>tsubst_major:{
                        pos=>(~(punct;marker;explet;n;pron;v;art))}}}
                     }),
    str      => type({
        tphrasal:{constr=>(~(lexical;compound))},
        tmorphol:{constr=>(lexical;compound)},
        tpunct:{constr=>lexical}
      })
  },'').
```

Type `tsyn` contains information constraining syntactic behavior. Note that although there is a general declaration for types `tcat`, `thead`, and `tmajor`, this declaration contrains them such that, e.g., `tsubst_cat` and `tsubst_head` must cooccur.

Similarly, the value of `str` is restricted in that a `tphrasal` type may not be of construction types *(lexical;compound)*, while a `tmorphol` may only be of those construction types.

2.1.7 type `tcat`

```
tcat >
  {
  tpunct_cat,
  tfunct_cat,
  tsubst_cat
      }.

type(
  tcat:{
    head => type({thead:{}})
      },'').

type(
  tpunct_cat:{
    position  => boolean([{left,right}])
      },'').

type(
  tfunct_cat:{},'').

type(
  tsubst_cat:{
    cohead=>type({tnothing:{},tcohead:{}}),
    subj=>list(type({tsynsem:{}})),
    compls=>list(type({tsynsem_opt_compl:{}})),
    by_ag=>list(type({tsynsem:{}}))
      },'').
```

All cat-types have a head attribute, but only substantive categories can contain subcategorization information, coheads or by-agents. Note the value of `compls`; this is the only occurrence of `tsynsem_opt_compl` within the declarations (other than its own declaration), i.e. only complements may be specified as optional.

2.1.8 type thead

```
thead >
  {
  tpunct_head,
  tfunct_head,
  tsubst_head
    }.

% Information shared by all heads, both functional and substantive.
% No attribute has thead:{} as its value, it is only declared
% to allow inheritance of general features.
type(
  thead:{
    major=>type({tmajor:{}})
      },'').

type(
  tpunct_head:{},'').

type(
  tfunct_head:{},'').

type(
  tsubst_head:{
    % Must be identical to 'newmarking' declaration within
    % tspecmodmark:{}.
    marking=>boolean([
               {prequant,cquant,postquant,card,ord,defin,
                unmarked,at,pp,adv1,adv2,adv3,adv4,adv5,punct}]),
    spr=>boolean([{sat,unsat}]),
    % Must be identical to 'p_lu' declaration within tp_major:{}.
    p_lu_mod => boolean([{
                          af,bag,bag_ved,efter,for,fra,gennem,hos,i,
                          i_forhold_til,imellem,imod,ind_i,ind_under,
                          inden_for,med,med_hensyn_til,mellem,mod,ned_af,
                          om,op_paa,op_til,over,over_for,paa,som,til,
                          ud_af,uden_for,under,ved,vedroerende }])
                      },'').
```

The Danish implementation distinguishes between *major* head features, which always are percolated up the analysis structure, and other, marking-type head features which sometimes are changed when combining the head with other elements. This not only adds greater nuance to the conceptualization of head features, but also gives a technical savings when implementing co-representation. This is because marking features must be included as part of the co-representation, so that if they are not part of the head proper, co-representation must be done at the next highest level, i.e. the `cat` level. This would mean a much greater resource investment in the co-representation approach, since the type `tcat` contains subcategorization information (see above).

Including marking-type features as part of `thead` allows co-representation to occur at this level with much less information structure-shared, and therefore with much greater efficiency.

Via the type `thead`, all nodes have a `major` attribute, while only substantive heads have the marking-type attributes, i.e. `marking`, `spr` and `p_lu_mod`.

Note that the `marking` attribute has values indicating its use for quantifier combinatorics, adverb combinatorics, as well as marking via punctuation and the traditional *at* marker. This represents a generalization of the constraining of mutually exclusive[2] cooccurrence behavior, where the use of more than a single attribute would be superfluous.

The attribute `spr` is used similarly to HPSG to indicate whether or not a noun is missing a specifier via the values *unsat* and *sat*, respectively.

The attribute `p_lu_mod` is used to store the `lu`-value of a post-modifying prepositional phrase (i.e. projection). This is used to eliminate some overgeneration in the case where a verb takes a complement noun followed by an optional complement prepositional phrase. Without this special attribute, overgeneration occurs, since the following prepositional phrase can be analyzed both as a complement to the verb and as a post-modifier to the noun, where the optional prepositional complement has been extracted from the verb's `compls` list. To eliminate the latter analysis, the verb entry is coded with the complement noun containing the negation of the `lu`-value of the optional following preposition, e.g. `p_lu_mod=>(~for)`, whereafter a post-modifying *for*-phrase can never be attached to the complement noun.

2.1.9 type `tmajor`

```
tmajor >
  {
  tpunct_major,
  tfunct_major >
    {
    tmark_major,
    texplet_major
      },

  tsubst_major >
    {
    tpos_major >
      {
      tagr_major >
        {
        tnom_major >
          {
          tn_major,
          tpron_major
            },
        tadj_major,
        tquant_major,
        tadv_major
          },
      tcard_major,
      tord_major,
      tp_major,
```

[2] In the sense that adverbs, quantifiers, and verbs marked by *at* are non-overlapping groups.

```
      tv_major >
        {
        tv_participial,
        tv_fininfin
          }
        },
    tconj_major
      }
    }.


type(
  tmajor:{
    selects => type({tnothing:{},tspecmodmark:{}}),
    pos             => boolean([
                         {n,v,adj,
                          adv,art,pron,quant,
                          conj,p,explet,
                          marker,punct}])},'').
```

As mentioned above, features within `tmajor` are always percolated as part of a projection of the node in question. Via `tmajor`, all node types contain the attributes `pos` (Part-of-Speech) and `selects`.

```
type(
  tnothing:{},'').

tspecmodmark >
        {
     tspecmod >
           {
        tspecifiee,
        tmodifiee },
     tmarkee }.

type(
  tspecmodmark:{
    synsem=>type({tsynsem:{}}),
    % Must be identical to 'marking' declaration within tsubst_head:{}.
    newmarking=>boolean([
               {prequant,cquant,postquant,card,ord,defin,
                unmarked,at,pp,adv1,adv2,adv3,adv4,adv5,punct}])
      },'').

type(
  tspecmod:{},'').

type(
  tspecifiee:{},'').

type(
  tmodifiee:{},'').

type(
```

```
  tmarkee:{},'').
```

These define the possible values of `selects`. Note the definition of `newmarking`, which must be identical to the declaration of the attribute `marking` within type `tsubst_head`. `newmarking` is used to contain the marking value to be passed up the analysis structure via a head-specifier or head-adjunct schema (see the Marking Principle, Section 1.4.2).

The type `tnothing` is generic and used for expressing a nil value for `selects` (see the Selection Principle, Section 1.4.2).

The `selects` values `tspecifiee` and `tmodifiee` are declared as subtypes of `tspecmod`, allowing generalization over specifiers and modifiers (used for head selection, see Section 2.2).

```
% Major head information shared by all substantive heads.
type(
  tsubst_major:{
    prd => atom({yes,no})
      },'').

type(
  tpunct_major:{},'').

type(
  tfunct_major:{},'').
```

Note that only substantive categories can be predicating (attribute `prd`).

```
type(
  tpos_major:{
    posit=>boolean([{front,nexus,end,none}])
      },'').
```

This is the general type for wordclasses using positional information.

```
type(
  tagr_major:{
    % Must be identical to declaration in type tn:{}
    agr=>boolean([{comm,neut},{sing,plur},{def,indef}])
      },'').
```

This is the general type for wordclasses using agreement information.

```
type(
  tnom_major:{
    case => boolean([{gen,nom,acc}])
    },'').
```

This is the general nominal type, for generalizing over nouns and pronouns.

```
type(
  tn_major:{
    % Must be identical to declarations in type tn:{}
    masscount => boolean([{mass,count}]),
    def_enclitic  => atom({y,n})
    },'').
```

This is the major declaration for nouns. Note that in Danish, nouns can take a definite enclitic when they are unmodified. Pre-modified nouns take a definite article before the modifier, just as in English. Due to this phenomenon, it is not adequate to distinguish nouns in terms of definiteness alone, since then there would be no way to distinguish between pre-modified and unmodified definite nouns. The mass-count distinction is coded here as a major head feature.

```
type(
  tadj_major:{
    adjform => atom({base,compar,superl}),
    prd_agr => boolean([{comm,neut},{sing,plur},{def,indef}])
      },'').
```

Major head information specific to adjectives. `adjform` indicates whether the morphological form is base, comparative or superlative.

```
type(
  tpron_major:{
    type => boolean([{pers,poss,interr,rel,refl,demo,recipr,quant,expl,art}])
      },'').
```

As in many other languages, Danish has a variety of pronoun types. These are declared as a boolean, making it possible to constrain rules using boolean operators.

```
type(
  tcard_major:{},'').

type(
  tord_major:{},'').

type(
  tquant_major:{
    quantform => atom({base,compar,superl})
    },'').
```

Cardinals, ordinals and quantifiers. Note that quantifiers can occur in comparative and superlative forms.

```
type(
  tv_major:{
    modal => atom({y,n}),
    v_lu => boolean([{have,vaere,blive,other}]),
    perf_aux => atom({'vaere','have'}),
    voice => atom({act,pass}),
    nex => boolean([{nva,vna,nav}])
      },'').

type(
  tv_participial:{
    partform => boolean([{past,pres}])
      },'').

type(
```

```
  tv_fininfin:{
    type => boolean([{imper,pres,past,infin}])
      },'').
```

Major head declarations for verbs. Boolean declarations here allow rules applying to subsets of verbal types. Note the particularly Danish feature `nex`, indicating the ordering of elements in the nexus field. Normally in Danish main clauses, adjuncts to the finite verb are positioned after the verb (`nex=>nva`), while in subclauses they are positioned before the verb (`nex=>nav`). Topicalization may only occur in main clauses, and causes the subject to move after the verb (`nex=>vna`).

```
type(
  tadv_major:{
    advform => atom({base,compar,superl})
      },'').

type(
  tp_major:{
    p_compl         => type({tsubst_head:{}}),
    % Must be identical to 'p_lu_mod' declaration within tsubst_head:{}.
    p_lu            => boolean([{
                         af,bag,bag_ved,efter,for,fra,gennem,hos,i,
                         i_forhold_til,imellem,imod,ind_i,ind_under,
                         inden_for,med,med_hensyn_til,mellem,mod,ned_af,
                         om,op_paa,op_til,over,over_for,paa,som,til,
                         ud_af,uden_for,under,ved,vedroerende }])
      },'').

type(
  tconj_major:{},'').

type(
  tmark_major:{},'').

type(
  texplet_major:{
    expl_type => atom
      },'').
```

Declarations of major head features for adverbs, prepositions, conjunctions, markers and expletives. Note that the `lu` value and the head information from the complement of a preposition is saved within the feature `p_compl`. This information is used as constraints for combining the prepositional phrase with other word classes. For example, the entry for *magen* (identical), as in *Den er magen til min bil.* (It is identical to my car.) can use this feature to restrict the complement of its complement preposition *til* (to) to being a nominal. Without percolation of this information via `p_compl`, this constraint would not be possible. This information is also used at refinement for determining which entry should be applied to, e.g., *god* in *Han er god til at lave mad.* (He is god at making food (cooking).), vs. *Han er god til madlavning.* (He is good at cooking.).

The `p_lu` value is defined as a boolean. This is used as a constraint to prevent ambiguity in cases where a verb can take a direct object nominal followed by an optional prepositional phrase (see Section 2.1.8).

2.1.10 type tcohead

```
type(
  tcohead:{
    n   => type({tnothing:{},tn_cohead:{}}),
    v   => type({tnothing:{},tv_cohead:{}}),
    p   => type({tnothing:{},tp_cohead:{}}),
    adj => type({tnothing:{},tadj_cohead:{}}),
    adv => type({tnothing:{},tadv_cohead:{}})
      },'').

type(
  tn_cohead:{   head=>type({
                  tsubst_head:{major=>tnom_major:{pos=>(n;pron;art)}}}) },'').
type(
  tv_cohead:{   head=>type({
                  tsubst_head:{major=>tv_major:{pos=>v}}}) },'').
type(
  tp_cohead:{   head=>type({
                  tsubst_head:{major=>tp_major:{pos=>p}}}) },'').
type(
  tadj_cohead:{ head=>type({
                  tsubst_head:{major=>tadj_major:{pos=>adj}}}) },'').
type(
  tadv_cohead:{ head=>type({
                  tsubst_head:{major=>tadv_major:{pos=>adv}}}) },'').
```

Type `tcohead` is used for implementing co-representation, a technique for reducing the number of lexical entries by representing alternatives for, e.g., subcategorization within a single structure. It is in fact a way to implement disjunction over types without actually having a disjunction operator. A typical example of the use of a cohead representation is where certain verbs can take as a direct object either a saturated nominal projection or an *at*-infinitive verbal projection. Examples in Danish are *klargøre* (prepare), *nå* (achieve), and *opgive* (give up). Within the lexical entries for these verbs, the direct object complement coding within the `compls` list has an underspecified `head` value, while the `cohead` value has information specified for the alternative complement noun and verbal projections.

2.1.11 type tstr

```
tstr >
  {
  tpunct,
  tphrasal,
  tmorphol >
    {

    tstem >
      {
      tstem_major >
        {
        tv_all >
          {
          tv,
```

```
          tmodal
            },
        tstem_compoundable >
          {
          tn,
          tadj
            } },

      tstem_minor >
        {
        tadv,
        tpron,
        tquant
          } },

    tword >
      {
      tconj,
      tp,
      tmarker,
      texplet
        } } }.

type(
  tstr:{
    lu => atom,
    constr => boolean([{lexical,compound,word,
                        h_subj,h_compl,h_subj_compl,
                        h_spec,h_adj,h_mark,h_mark_punct,h_fill}])
                        },'').
```

Type `tstr` contains information on construction type and the base form of the node or the head of the node (`lu`). This type is *never* structure-shared between nodes.

Currently there are the 11 construction types shown here, where *h_subj, h_compl, h_subj_compl, h_spec, h_adj, h_mark* and *h_fill* are inspired by similar schemata within HPSG. The values *lexical, compound, word* were added for describing the construction types within word formation rules, while *h_mark_punct* is a special head-marker construction type for combining a head with a punctuation mark.

For strictly morphological (i.e. non-phrasal) elements, the attributes `constr` can have the values *lexical* or *compound*. `constr=>lexical` indicates that the node is a terminal node and thereby must unify directly with a lexical entry. The value `constr=>compound` indicates that the node is a compound, consisting of two or more morphemes. A node with `constr=>word` can only be a minimal projection of type `tphrasal`, i.e. having a daughter of type `tmorphol`. (See Section 7.1.)

```
type(
  tpunct:{
    lemma => atom
    },'').
```

A punctuation sign has no morphographemic information. `lu` and `lemma` are always the same.

```
type(
```

```
  tphrasal:{
    headed => boolean([{left,right}]),
    heading => boolean([{left,right}])},'').
```

This is the `str` value for phrases. `headed` indicates on which side the head-daughter of the node is found, while `heading` tell on which side the node can occur as a head-daughter. These are used e.g. for controlling whether post-modifiers or pre-modifiers are parsed first. An input with both modifier types would otherwise generate a vacuous ambiguity.

```
type(
  tmorphol:{
    lemma => atom,
    seq => boolean([{first,not_first},{last,not_last}])
      },'').
```

`seq` indicates the horizontal position of the node within a word. `lemma` is the form used as a stem during morphological parsing.

```
type(
  tstem:{
    infl => boolean([
              {infl1,infl2,infl3,infl4,infl5,infl6,irreg},

              {e,n,r,s,t,                                      %  5
               en,er,es,et,ne,ns,re,rs,st,te,ts,               % 11
               ede,ene,ens,ere,est,ers,ets,nes,rne,ste,tes,    % 11
               edes,ende,enes,erne,este,rnes,                  %  6
               ernes,                                          %  1
               null                                            %  1
                 }]),
    grf => type({tgrf:{}}),
    contin => atom
  },'').

% General morphographemic characteristics:  syncope, etc.

type(
  tgrf:{
    gemination => atom({y,n}),
    syncope => atom({y,n}),
    fuge => boolean([{e,s,'-',null,'mid-s','mid-null'}]),
    fuge_used => atom({y,n})
      },'').
```

Type `tstem` is the general type with information inherited by all stems, major and minor. Although minor stems do not take all suffixes, major inflectional types, nor syncope or gemination, `infl` and `grf` are declared here so that TLM rules can be written generally to apply to major and minor stems alike.

```
% Major stems
% -----------
```

```
type(
  tstem_major:{},'').

type(
  tv_all:{},'').

type(
  tv:{},'').

type(
  tmodal:{},'').

type(
  tstem_compoundable:{},'').

type(
  tn:{
    % Must be identical to declaration in type tagr_major:{}.
    agr=>boolean([{comm,neut},{sing,plur},{def,indef}]),
    % Must be identical to declaration in type tn_major:{}.
    masscount=>boolean([{mass,count}])
      },
  'agreement features: gender, number, definiteness.  Type (mass vs. count).').

type(
  tadj:{},'').
```

Major stems have relatively complex inflectional morphology (enough to justify specification of inflectional paradigms) and morphographemic changes to the stem.

Note the type `tstem_compoundable`, which is used for controlling which word classes may occur as part of a compound (it is used as a constraint within the TLM rules for compounding). Currently only nouns and adjectives may occur in compounds.

```
% Minor stems
% -----------

type(
  tstem_minor:{},'').

type(
  tadv:{},'').

type(
  tpron:{},'').

type(
  tquant:{},'').

% Words
% -----

type(
  tword:{},'').
```

```
type(
  tconj:{},'').

type(
  tmarker:{},'').

type(
  texplet:{},'').

type(
  tp:{},'').
```

Minor stems have a weak inflectional morphology and no morphographemic changes to the stem.

Words have no inflectional morphology whatsoever.

2.1.12 type tnonlocal

```
type(
  tnonlocal:{},'').
```

This type is not used within the current implementation.

2.2 Processing declarations

2.2.1 Head selection

Head selection declarations are used by the head-out parsing algorithm to select a parse head, i.e. a daughter node from which to begin parsing. These have a significant effect on runtime.

Every word and phrase structure rule must have an applicable head selection declaration, otherwise the rule will not be accessible during parsing.

Word Structure

```
%%   phrasal > morphol

select_analysis_head( [ana_ws],
sign:{
  synsem=>tsynsem:{syn=>tsyn:{
                      str=>tphrasal:{constr=>word},
                      cat=>tsubst_cat:{head=>tsubst_head:{}}}}},
sign:{
  synsem=>tsynsem:{syn=>tsyn:{
                      str=>tmorphol:{constr=>(lexical;compound),
                                     seq=>(first&last)},
                      cat=>tsubst_cat:{head=>tsubst_head:{}}}}}).

select_analysis_head( [ana_ws],
sign:{
```

```
  synsem=>tsynsem:{syn=>tsyn:{
                       str=>tphrasal:{constr=>word},
                       cat=>CAT}}},
sign:{
  synsem=>tsynsem:{syn=>tsyn:{
                       str=>tmorphol:{constr=>lexical,
                                      seq=>(first&last)},
                       cat=>CAT=>tfunct_cat:{}}}}).

%%   morphol > morphol

select_analysis_head( [ana_ws],
sign:{
  synsem=>tsynsem:{syn=>tsyn:{
                       str=>tstem_major:{constr=>compound,seq=>first},
                       cat=>CAT}}},
sign:{
  synsem=>tsynsem:{syn=>tsyn:{
                       str=>tstem_major:{constr=>lexical,seq=>(~first)},
                       cat=>CAT=>tsubst_cat:{head=>tsubst_head:{}}}}}).
```

The set of word structure rules is relatively simple, and only these three head selection rules are necessary for efficient processing. The first and second select a substantive and functional `tmorphol` daughter, respectively. Since the `tphrasal:{constr=>word}` word structure rules all only have a single daughter, a single head selection declaration would have sufficed, however splitting it into two makes it possible to specify more precise information sharing between the mother and (parse) head-daughter that would be possible with a single declaration. This results in head relations that are more constrained.

The third head selection declaration applies to word structure rules for compounding, taking the non-initial daughter as the head. This is most efficient due to the left-branching structure of word formation.

Phrase Structure (PS)

For efficient head selection, the maximal amount of head information must be structure-shared between mother and head-daughter. This means that in rules where the parse head is identical to the linguistic head, as much as possible of the head information within the daughter should be structure-shared with information in the mother within the head-selection declaration.

In the Danish implementation, a set of head-selection declarations have been developed which combine this maximal sharing of information with the flexibility of determining the parse head from the phrase structure rules themselves, such that parse heads can be changed without having to recompile the head declarations (and everything else!). The key to this is the feature `sign|procinfo|parsehead`, which is declared as taking only the value y (or no value). This is used within the PS rules to indicate which daughter is parse head, i.e. the one with `parsehead=>y`. (The parser uses subsumption to test this value.)

Since via the `parsehead` feature the developer can specify any daughter 'type', for example head vs. non-head-daughters or functionals vs. substantives vs. punctuation, the head selection declarations must be defined to allow any type of daughter to be parse head, as well as to take the `parsehead` feature into account and to implement the structure-sharing mentioned. Seven head selection declarations are necessary to take these possibilities into account, two of which are given below as examples.

```
%% a substantive head

select_analysis_head( [ana_ps],
 sign:{
   synsem=>tsynsem:{
     syn=>tsyn:{
         str=>tphrasal:{},
         cat=>tsubst_cat:{
             head=>tsubst_head:{major=>MAJOR}}}}},
 sign:{
   procinfo=>tprocinfo:{parsehead=>y},
   synsem=>tsynsem:{
     syn=>tsyn:{
         str=>tphrasal:{},
         cat=>tsubst_cat:{head=>tsubst_head:{
                              major=>MAJOR=>tsubst_major:{}}}}}}).


%% an substantive adjunct or specifier

select_analysis_head( [ana_ps],
 sign:{
   synsem=>tsynsem:{
     syn=>tsyn:{
         str=>tphrasal:{constr=>(h_adj;h_spec)},
         cat=>tsubst_cat:{
             head=>tsubst_head:{major=>MAJOR}}}},
 sign:{
   procinfo=>tprocinfo:{parsehead=>y},
   synsem=>tsynsem:{
     syn=>tsyn:{
         str=>tphrasal:{},
         cat=>tsubst_cat:{
             head=>tsubst_head:{
                 major=>tsubst_major:{
                     selects=>tspecmod:{
                         synsem=>tsynsem:{
                             syn=>tsyn:{
                                 cat=>tsubst_cat:{
                                     head=>tsubst_head:{
                                         major=>MAJOR=>tsubst_major:{}
                                      }}}}}}}}}}}).
```

Note the use of the variable `MAJOR`, which implements the structure-sharing. Compilation of head selection declarations also takes into account shared structures from the PS rules other than those explicitly given, as long as they are at a depth not exceeding that which is explicitly declared. For example if there is a variable shared between the mother node and the parse head within a PS rule which is a sibling to the feature `major`, it will also be used during compilation of the head relations used at runtime, meaning more constraints on the resulting head relations and therefore more efficient processing.

2.2.2 Keys

The keys below indicate which features are to be used during lexical lookup at each of the three major processing phases, i.e. TLM analysis, syntactic analysis, and refinement.

```
ld_ana_key(
X,
sign:{
  synsem=>tsynsem:{
            syn=>tsyn:{
                   str=>tmorphol:{lemma=>X}}}},
tspecinfo:{tlm=>y} ).

ld_ana_key(
X,
sign:{
  synsem=>tsynsem:{
            syn=>tsyn:{
                   str=>tmorphol:{lu=>X}}}},
tspecinfo:{ana=>lex} ).

ld_ana_key(
X,
sign:{
  synsem=>tsynsem:{
            syn=>tsyn:{
                   str=>tmorphol:{lu=>X}}}},
tspecinfo:{ref=>y} ).
```

Note that for TLM analysis, the `lemma` value is used, while `lu` is used for the other phases. This is because TLM analysis is based on the `lemma` value, which is the morphological stem of the word, so that it makes most sense (and is most efficient) to do lexical lookup during TLM based on that feature. Once a word has been parsed morphologically, the `lu` or base form value is used for lookup during subsequent processing phases, since the morphological stem is no longer interesting.

The two key declarations below are similar to the previous keys for lexical lookup, however these indicate which attribute to use as a key for indexing PS rules.

```
ld_ana_mother_key(
X,
sign:{
  synsem=>tsynsem:{
    syn=>tsyn:{
          str=>tstr:{constr=>X}}}}).

ld_ana_head_key(
X,
sign:{
  synsem=>tsynsem:{
    syn=>tsyn:{
        cat=>X}}}).
```

The head key is used during (headed) rule lookup during analysis. The mother key is used during refinement. Both keys are also used during compilation to compute the head relation and rule

indexing.

2.2.3 Paths

```
ld_morph_feature(
X,
sign:{
        synsem=>tsynsem:{
                  syn=>tsyn:{
                         str=>tmorphol:{lemma=>X}}}}).

ld_specifier_feature(
X,
sign:{
        procinfo=>tprocinfo:{specinfo=>X}}).

ld_two_level_rules_feature(
X,
sign:{
  synsem=>tsynsem:{syn=>tsyn:{str=>X=>tmorphol:{}}}}).
```

Paths tell the system where to find information. The `ld_morph_feature` declaration indicates which attribute is to be used as the basis for TLM analysis (`lemma`). `ld_specifier_feature` indicates the attribute value relative to which the specifiers are declared (see below). `ld_two_level_rules_feature` indicates which feature structure from the TLM lexical entries is to be added to the input (via unification) as a result of TLM analysis.

2.2.4 Specifiers

```
named_spec(tlm,       tspecinfo:{tlm=>y} ).
%named_spec(lift,      tspecinfo:{lift=>y} ).
named_spec(ana_lex,   tspecinfo:{ana=>lex} ).
named_spec(ana_default,   tspecinfo:{ana=>default} ).
named_spec(ana_ws,    tspecinfo:{ana=>ws} ).
named_spec(ana_ps,    tspecinfo:{ana=>ps} ).
named_spec(ref,       tspecinfo:{ref=>y} ).
named_spec(ref_default,   tspecinfo:{ref=>default} ).
%named_spec(lower,     tspecinfo:{lower=>y} ).
```

These are the specifiers used to partition rules and lexical entries. Note that ALEP does not yet utilize specifier declarations for lifting and lowering.

The specifiers `ana_default` and `ref_default` are interpreted specially by the system to indicate lexical entries used as defaults during analysis and refinement, respectively.

Chapter 3

Coverage

3.1 Messy details

The size of the lexicon has been reduced by the implementation of a word-construct tagger which identifies a variety of so-called 'messy details' in the input before lexical lookup occurs (see Chapter 4).

The tagger currently identifies the following phenomena:

- **proper names**
  Herre Povlsen, Fru Jensen
- **dates**
  den 15. marts 1995
- **patent specifications**
  DE-U-85 30 365
  DE 34 26 465 A1 og CH 398 453
- **codes**
  M-12106
  No. 6145A
- **numbers**
  De to vægge, 31 og 32, ...
  fig 13 og 14
  tabel 5
  15%
  25-30 cm.

3.2 Lexica

A great deal of lexical information has been migrated automatically and semi-automatically from the Danish Eurotra resources at CST. Before the implementation of default lexical entries, the lexicon at TLM contained over 4500 base entries, with around 3000 entries at analysis. Now that default entries have replaced several thousand previous entries, the total number of compiled entries is only XX, although the coverage is now much greater than previously.

There are other factors that make the actual lexical coverage many times greater than the number of compiled entries, including the implementation of a morphological component with inflections and compounding, the implementation of optional complement extraction and co-representation of heads, all of which compress the number of lexical entries necessary for parsing a very large number of tokens. The result is a remarkable coverage given the short lifespan of the project.

3.3 Morphology

The morphology implementation includes comprehensive morphological analysis via lexical entries, TLM and lifting rules. The implementation accounts for inflectional suffixation, stem changes, fuge elements, gemination, syncope, and compounding. Limited regularity within irregular forms is exploited to the extent possible by allowing for irregular pseudo-stems taking regular endings.

3.4 Syntax

In Navarretta (1996), the following revised priority list for syntactic implementation is given (the list assumes TH and morphology). Phenomena already implemented at the time of writing are shown in boldface.

1. **Determination**
2. **Pre-modified and post-modified NP constructions**
3. **Active main clauses**
4. **PP constructions, also as complements**
5. **Complement finite and non-finite subclauses**
6. **AP and AdvP constructions**
7. **Verbal complements**
8. **PAS**
9. Relative clauses
10. Coordination

The following additional phenomena were to be implemented time permitting. Phenomena already implemented at the time of writing are shown in boldface.

1. **Passivization**
2. **Other PP,AdjP, AdvP constructions**
3. Other post-modified NP constructions
4. Topicalization/long-distance dependencies (**Topicalized adjuncts**)
5. Simple negation
6. **Der/det-constructions**

Danish has three ways to form passives: *s*-passives (*Æblet spises.* 'the apple is eaten'); *blive + pastparticiple* (*Æblet blev spist.* lit. 'the apple became eaten'); and *være + pastparticiple* (*Æblet er spist.* lit. 'the apple is eaten'). The form with *være* 'be' is less common, and involves a semantic distinction not within the scope of the project. The Danish implementation parses all these forms syntactically but assigns them the same semantics.

By 'other PP, AdjP, AdvP constructions' is meant other than basic constructions. The Danish implementation includes predicative nouns and adjectives, configurational restrictions on adverbials, and a distinction between nexus and other adverbials (see Section 7.2).

Some types of *der/det*-constructions are implemented via syntax rules having the functionality of lexical rules. They are triggered by the presence of *der* or *det*, respectively, and reconfigure the complement structure of their corresponding verb (see Section 7.2).

The implementation of complement structures includes facilities for optional complement extraction.

Attachment ambiguity of PP modifiers is retained.

Verbal structures include modals and auxiliaries.

The combinatorics of determiners and adverbs, respectively, have been implemented via special constraints implemented as values of the `marking` feature (see Chapter 7).

3.5 Semantics

The implementation includes semantics for predicate-argument structure (PAS) of complementizing elements (verbs, nouns, adjectives), control constructions, genitive constructions, adjunct constructions, predicatively used nominal, adjectival and prepositional phrases, pronouns, and quantifiers.

Chapter 4

Text handling

Text handling within ALEP as delivered is the phase of processing where paragraph, sentence and word boundaries are identified and marked up using SGML tags. Users can integrate new processes within the standard TH (text handling) package.

This possibility has been exploited within the LSGRAM project to implement a tagger of word constructs, including so-called messy details. This tagger was originally developed at CST and has been extended by other members of the LSGRAM project (see Bredenkamp et al.). The tagger is documented in Music (1995c), from which much of this chapter originates.

4.1 Messy text constructs and patents

Messy details are text constructs which do not lend themselves well to treatment by traditional techniques for linguistic analysis, whence their 'messiness'. Typical examples are numbers, codes or other (sequences of) wordforms which can occur in many variations (often infinite), making impossible a comprehensive treatment by traditional means.

As part of a corpus analysis of Danish mechanical patent texts (Povlsen et al. 1995a), messy details were classified according to levels, viz. general format level, sentence level and word level phenomena. *General format level* phenomena occur over sentence boundaries, example being headers, meta-comments and tables. Phenomena classified as *sentence level* occur within a single sentence, but cannot be considered word constructs of a fixed nature. These are more 'linguistic' than the usual messy details, but were included as part of the analysis of messy details since they lend themselves to partial analysis via a similar type of pre-processing. Examples of these are very long sentences, parenthetical text and commas, all of which present problems to practical implementations which can be ameliorated by segmentation during pre-processing.

The *word level* phenomena are the ones treated by the tagging program (see the previous chapter for coverage). For any realistic application these types of construct must be processed efficiently, the alternative being coding them individually in some lexicon. Not only must they be given attention, but they should ideally be considered from the start of implementation in order to avoid having to provide a home-grown solution and reimplementing later, and to generate a system where linguistic and non-linguistic processing are integrated and complementary.

For these reasons, processing of messy details was prioritized, making it possible for the LSGRAM project as a whole to have some facility available early on for tagging word constructs. This also avoided the situation where groups develop multiple, mutually incompatible tentative solutions of their own, with a corresponding duplication of work. The result has been a profitable synergy within the project, some of the results can be seen in Bredenkamp et al. (in progress).

Given the goal of processing these phenomena efficiently, questions arise as to where within the chain of processing they should be handled, and with what tool. These are in fact interdependent questions, but are nonetheless discussed in separate sections below. However, not to leave the reader in suspense, the solution arrived at has been to develop an `awk` program integrated at the word recognition phase of TH. The program `tagit_da`[1] is relatively efficient, quickly and easily integrated with ALEP's existing TH functionality, and is easy to modify or replace with other locally-developed software. In addition, the power of `tagit_da` is sufficient to handle most of the phenomena present in the text types tackled by the various LSGRAM groups.[2]

4.2 Integration with ALEP

The most natural place for a user application given the system as delivered would seem to be at the `User application` phase of TH. This is consistent with the projection in EUROTRA-6/3 (1991), Section 6.10 regarding the most likely place for user-defined pre- and post-processing applications. ALEP is provided with a C-library of functions for accessing the tree structure generated as output from word-recognition, where words have been enveloped by the `<W>` and `</W>` SGML tags.

The types of word construct of interest here can be spotted by matching regular expressions over a sequence of words. For this type of application, use of the C-functions was considered less straightforward, since matching patterns over a sequence of words would have to be done in C by either explicitly stepping through the words matching them one at a time against a pattern, or by reconstructing the sentence without the `W`-tags and then doing a single match against the pattern.

By implementing the pattern-matching before word recognition, development was facilitated, since sentences could be matched directly against regular expressions. This placement also complements word segmentation (i.e. morphological analysis), in that wordforms belonging to word constructs found by `tagit_da` need no morphological processing, speeding that process up.

Integration of the tagging program with ALEP is a straightforward affair.

1. Install (i.e. copy) `tagit_da(.awk)` in an appropriate place (e.g. `$ALEP_USER_PATH/src/tx/`). Make sure it's executable.
2. Modify `$ALEPHOME/src/tx/etc/alep_tx_wordana` to call the tagger via the environment variable `ALEP_TH_TAGGER` before `sent_seg` (the word recognition program). Copy the new version to `$ALEPHOME/bin`. The relevant lines of code within `alep_tx_wordana` should be something like this:

```
TAGTMP=$ALEP_TMP/user_tag.$$
$CMD_RM_F $TMP_TREE.sit $TMP_TREE.txt $TAGTMP

if test -z "$ALEP_TH_TAGGER" -o ! -x "$ALEP_TH_TAGGER"
then
  sent_seg_in=$1
else
  $ALEP_TH_TAGGER < $1 > $TAGTMP
  sent_seg_in=$TAGTMP
fi

sent_seg -t $TMP_TREE -i $sent_seg_in -o $2   1> $MSGFILE 2> $MSGFILE
$CMD_RM_F $TMP_TREE.sit $TMP_TREE.txt $TAGTMP
```

[1] Since the program contains data specific for the Danish language, the name has a language suffix added. This convention has been adopted within the LSGRAM project as a whole.

[2] The program has been ported to `perl` by the Spanish LSGRAM group, giving a substantial runtime improvement.

3. Modify `$ALEPHOME/bin/alep_tx_anacombo` to call the usual `Full Text Analysis` procedures if you want tagging when running `Analyse Selected Text`. This is slightly more complicated than the last step.

4. Before starting ALEP, set the environment variable `ALEP_TH_TAGGER` to the tagging program, e.g.

   ```
   setenv ALEP_TH_TAGGER $ALEP_USER_PATH/src/tx/tagit_da.awk
   ```

5. To access the program via ALEP's UI, make a `tx_document` object pointing to the program file. Note that this is not an absolutely necessary step for using the tagger, it is only to make it convenient for finding the tagger file from, e.g., an All-in-One Tool. In this way, the developer need not remember where the program is installed. By using a `tx_document` object in this way, access to the program file is provided via the `Display Document` item of the `Actions` menu. But again, this is not part of the integration, it's only for convenience.

6. Make lift rules for the new tag type(s) (see below).

7. Check the analysis/refinement lexica and grammars to ensure that the lifted versions of the tagged nodes can be parsed. General lexical entries can be used. To help make the interaction between the tagging program, the lifting rules and the lexical entries for these tags as transparent as possible, it's advisable to reuse the tag types as the lexical unit values.

4.3 A look at the program

Many messy details at the word level can be readily identified by pattern-matching techniques based on regular expressions. Various programming languages could be used for this, but one of the most common and straightforward is `awk`, whose entire raison d'être is reformatting of input based on pattern-matching. Compilation problems are also avoided, since `awk` scripts are interpreted.

Unfortunately, during implementation it was discovered that the `awk` delivered with some Unix systems has some limitations when processing the complex regular expressions needed for Danish messy details. The solution was to use the latest version of GNU `awk` (`gawk`), version 2.15.6, which is available to everyone on the internet.

This rest of this section goes through the program code.

In the `BEGIN` section of the program, general declarations are found specifying what characters may be considered word boundaries. When matching patterns against the input, a word boundary character must be present on each side of the sequence to be matched, otherwise the match will fail. These boundary characters are not considered to be part of the match itself, i.e. they are not replaced as part of the matched sequence.

The input and output record separators are defined as the character "<". This assumes that segmentation has only recognized down to the level of sentences. This declaration then causes `gawk` to regard every SGML tag as a record (i.e. sentence) delimiter. Thus an input like

```
<P><S>Det er den 15. marts.</S><S>Saetning nummer 2.</S></P>
```

gets segmented as the 6 records

```
P>
S>Det er den 15. marts.
```

```
/S>
S>Saetning nummer 2.
/S>
/P>
```

by `gawk` before pattern-matching begins. Any record not beginning with `S>` is skipped, all others are considered sentences and are sent through the pattern-matching sequence.

Regular expressions are written as strings, using parentheses for grouping. For clarity and generality, variables can be defined and reused within other regular expressions.

A separate `while`-loop for each pattern-tag combination is given, as shown below. This is not only processed more efficiently by `gawk` than the alternative `for`-loop, but it also makes the pattern-tag combinations and their orderings explicit (as opposed to ordering based on indices).

```
  while (match($0,wda date    wdz)) handlematch("DATE")
  while (match($0,wda number  wdz)) handlematch("NUMB")
```

The variables `wda` and `wdz` are the word boundaries mentioned above. `date` and `number` are set to regular expressions, which are linked to the tags `DATE` and `NUMBER`, respectively, via these while loops.

Since many patterns contain some of the same elements (numbers, for instance), ordering of the patterns is important for controlling which have a higher priority. In the example above, matching of the pattern identifying dates must have priority over that identifying numbers, otherwise the number 15 in the date will be tagged and replaced, and the sequence would then not match against the date pattern.

When a match is found, the function `handlematch()` is called to deal with it. In order to avoid a word matching more than one pattern, the matched sequence within the input record is replaced with a unique flag which itself cannot match other patterns, while the replacement string (possibly a rewritten version of the sequence matched) is recorded in an array. Once all the patterns have been checked against the input, the flags are replaced with their replacement strings, the record is output and processing proceeds to the next record.

The SGML tag `USR` is used for all patterns found, while the attribute `TYPE` is used to distinguish between them. Other attributes are also defined, such as `VAL`, `ORIG` and `LEVEL`. `ORIG` is set to the original input string value; `VAL` is a standard version of the original input sequence matched by the pattern. Dates and numbers are standardized currently in this way, and other special handling can be added easily. `LEVEL` is set to `M` for all patterns found. This information is used at lifting (see below). The data content of the match sequence (i.e. the sequence itself) is combined into a single unit by replacing spaces with underscore.

The result of tagging the example above is the following (with line breaks inserted for clarity):

```
<P>
<S>Det er
<USR LEVEL=W>
<USR LEVEL=M TYPE=DATE ORIG="den 15. marts" VAL="95/03/15">den_15._marts
  </USR></USR>.
</S>
<S>Saetning nummer
<USR LEVEL=W>
<USR LEVEL=M TYPE=NUMB ORIG="2" VAL="2">2</USR></USR>.
</S>
</P>
```

4.4 Patterns for patents

Since sentences are identified within the `gawk`-script as beginning with `S>`, patterns should not be written too loosely in order to avoid matching this tag. Otherwise any extended regular expressions can be used.

The rest of this section gives some examples, with instances of the constructs to be identified given first, and the `gawk` patterns themselves after. The actually patterns will obviously vary from language to language, from text type to text type.[3]

dates

Examples

```
Den 6. december 1941
den 7. nov 1962
d. 27. okt 96
```

Patterns

```
montharray["jan"] = "01"
montharray["feb"] = "02"
montharray["mar"] = "03"
montharray["apr"] = "04"
montharray["jun"] = "06"
montharray["jul"] = "07"
montharray["aug"] = "08"
montharray["sep"] = "09"
montharray["okt"] = "10"
montharray["nov"] = "11"
montharray["dec"] = "12"

day = "([0-9][0-9]?\\.)"
month = "((jan(uar)?)|(feb(ruar)?)|(mar(ts)?)|(apr(il)?)|(maj)|(juni?)|\
          (juli?)|(aug(ust)?)|(sep(t(ember)?)?)|(okt(ober)?)|(nov(ember)?)|\
          (dec(ember)?))"
year = "([0-9][0-9]([0-9][0-9])?)"
date = "([dD]((en)|\\.) "day" "month" "year")"
```

The `montharray` is used for generating the standard representation of each month. When a date is found, the first three letters of the month are used for looking up in the array to get the month's numeric value, which is assigned to the `VAL` attribute.

Note that case is significant. This is necessary, since for some patterns, e.g. proper names, it is important to retain the distinction.

patent specifications

Examples

```
DE-U-85 30 365
```

[3] For information more on extended regular expressions, consult relevant Unix documentation.

```
EP-A-0 385 555
DE 34.26/465,A1
DE 34 26 465 A1 og CH 398 453
DE 34 26 465 A1, CH 398 453, and US 22,55 23 B2
```

Patterns

```
cap = "[A-Z]"
conj = OR3("og","eller","og/eller")
listsign = OR(",",";")

combinumber = "([0-9]+((/|\\.|,| )"cap"?[0-9]+)*)"
combihead = "("cap cap"(-|"cap"|[0-9])*)"
pat = "("combihead" "combinumber")"
patref = "("pat"(("listsign")|( "conj"))+ "pat")"
```

This is modest in comparison with the pattern for patent specifications used the PaTrans system, where much more variation is accounted for.[4]

codes

Examples

```
M 12106
No. 6145A
```

Patterns

```
code = "(([A-Z] [0-9]+)|(No\\. [0-9]+[A-Z]))"
```

numbers

Examples

```
200 g/time   (measurement)
3,0 m3/time  (measurement)
2/3          (fraction)
2-8          (interval)
```

Patterns

```
sign = OR("+","-")
decimal = "(,[0-9]+)"

number = "(("sign" ?)?([0-9]+(\\.[0-9]+)*"decimal"?)"
interval = "("number" ?- ?"number")"
fraction = "("number" ?/ ?"number")"
range = "("number"( ?(-|/) ?"number")?)"

wtmeasure = "((k)?(g)\\.?|(kilo)?(gram)(s)?)"
```

[4] For more on the PaTrans system, contact CST or the authors.

```
volmeasure = OR4("mol","ml","cl","l")
sizemeasure = "(" OR5("km","mm","nm","cm","m") "\\.?[0-9]?)"

measure = "("range" ?" OR3(wtmeasure,volmeasure,sizemeasure)"(/ ?time)?)"
```

Numbers can also occur in parentheses, usually being references to indices within a figure, often following a noun.

Examples `Vandindløbet (1) og luftindløbet (2) ...`
`... fig. (1) ...`
`... fig. 13 og 14 ...`

Patterns

```
refr = "(\\("number"\\))"

xrefnames = OR3("tabel","eks((emp(el)|(ler(ne)?))|\\.)?",
                "fig((ur(er(ne)?)?)|\\.)?")
xrefnumber = "(\\(?(([0-9]+"letter"?)|[0-9]("lowerroman"+"cap"?)|\
                  ("caproman"+"lower"?))?)"
xrefnumberintv = "("xrefnumber"( ?- ?"xrefnumber")?)"
xrefs = "("xrefnames" "xrefnumberintv"((,? "xrefnumberintv")*,? \
            "conj" "xrefnumberintv")?)"
```

Note that the second element of the pattern `xrefs` is optional, so that it captures both single and multiple references.

4.5 Lifting and analysis of the tags

As all other parts of the input structure, the tagging results must be lifted. As with the output of word segmentation, each tag has a `W` (word) and an `M` (morphological) level, which must be lifted using separate rules.

```
ts_ls_rule(
  sign:{
    procinfo=>tprocinfo:{
            specinfo=>tspecinfo:{ana=>ws},
            ruleinfo=>truleinfo:{lift_id=>'USRWlift'}},
    synsem=>tsynsem:{
              syn=>tsyn:{
                     str=>tphrasal:{constr=>word}}}},
  'USR',['LEVEL'=>'W']).

ts_ls_rule(
  sign:{
    ortho=>tortho:{string=>[STR|REST],rest=>REST},
    procinfo=>tprocinfo:{
            specinfo=>tspecinfo:{partition=>tpart:{main=>usr,sub=>TYPE}},
            ruleinfo=>truleinfo:{lift_id=>'USRMlift'}},
    synsem=>tsynsem:{
              syn=>tsyn:{
                     str=>tword:{
                         constr=>lexical,
```

```
                        lu=>TYPE,
                        lemma=>VAL},
                    cat=>tsubst_cat:{
                        head=>tsubst_head:{
                             major=>tsubst_major:{
                                 pos=>(n;quant;adv;adj;v),
                                 selects=>tnothing:{}}
                              }}}}},
  'USR',['LEVEL'=>'M','TYPE'=>TYPE,'VAL'=>VAL],STR).
```

Consistent with the treatment of morphemes, the `lemma` value is set to a processable version of the actual input (i.e. for morphemes, the processable version is the stem upon which morphological processing can be based), while the `lu` (lexical unit) value is set to the general form, here the `TYPE` value. In addition, as with morphemes, the rule adds partitioning information for the user-defined tag based on a main partition `usr` and a subpartition set to the `lu` value (see Music (1995b)). The following lexical entries are given as examples.

```
'user-defined' ~
sign:{named_spec=>mLEXSPECanaref[usr,LU],
    synsem=>tsynsem:{
 syn=>tsyn:{
           str=>tword:{lu=>LU=>('DEGR'/'DOC'/'PAT'/'PATLIST'/
                                'PNAME'/'XREF'/'XREFS'/'REFR')},
           cat=>tsubst_cat:{
               head=>tsubst_head:{
                   spr=>sat,
                   major=>tn_major:{
                       pos=>n,
                       agr=>neut,
                       case=> ~gen,
                       selects=>tnothing:{}
                    }},
               compls=>[]}}
    }}.

'user-defined' ~
sign:{named_spec=>mLEXSPECanaref[usr,LU],
    synsem=>tsynsem:{
        syn=>tsyn:{
            str=>tword:{lu=>LU=>('FRAC'/'INTV'/'MEASURE'/'NUMB')},
            cat=>tsubst_cat:{
                head=>tsubst_head:{
                    spr=>sat,
                    major=>tcard_major:{
                        pos=>quant,
                        selects=>tmodifiee:{
                          newmarking=>card,
                          synsem=>tsynsem:{
                            syn=>tsyn:{
                              cat=>tsubst_cat:{
                                 head=>tsubst_head:{
                                     marking=> (ord;unmarked),
                                     major=>tn_major:{
                                         pos=>n,
```

```
                                    def_enclitic=>n}
                                         }}}}}}},
      compls=>[]}}
}}.
```

The first entry is for temperature measures, document references, patent specifications (including lists of them), proper names, and intratextual references (e.g. “tabel 5”, “figur 4.1”). These are currently all handled as substantives taking no complements. The second entry is for numbers in the form of fractions, intervals, measures (other than temperature), and simple numbers. These are all currently treated as post-quantifying cardinals.

Chapter 5

Lexica

5.1 Lexicon partitioning

The lexica are organized into three major partitions, one for each of TLM analysis, structural (syntactic) analysis, and refinement. Specifiers are used to implement this partioning (see Section 2.2.4). In addition, the Danish implementation partitions via a two special features for the purpose, `sign|procinfo|specinfo|partition|{main,sub}`. This is because the specifiers are applied to entries and rules using subsumption, not unification, which in some situations has proven to be less efficient. A main partition and a subpartition are defined via these two features: the main partition is normally set to the part-of-speech of the lexical entry, while the subpartition value is set to the `lu` value. This makes for very finely grained partitioning.

5.2 Lexical coding: TLM

5.2.1 Major stems

In order for morphological processing to function properly, each entry must be coded with an explicit list of all its legal suffixes, the list acting as a constraint on which suffixes can be attached to the stem (the value of the feature `lemma` is the stem for morphological analysis).

Major stems must be coded with information on gender, inflectional type, stem change and fuge elements by using macros defined for the purpose. Based on this information, the complete list of possible suffixes for the entry is automatically generated as a boolean disjunction during macro expansion.

Nouns

Nouns with no vowel shift or consonant change are generally coded using one of the following templates (with the variable `LU` standing for `LEXICAL_UNIT` in this context):

```
m_TLM_n[LU, INFLECTION, GENDER]
m_TLM_n[LU, INFLECTION, GENDER, GRAPHEMICS]
```

`INFLECTION` for nouns and other major stems is indicated as a macro call of the form `m_<paradigm>[]`, e.g. `m_ninfl2[]`. (The inflectional paradigms are given in Appendix A.)

```
%%  Examples:  work, film, tax, spoon

m_TLM_n[ arbejde,          m_ninfl2[], m_ntr[], m_tgrf_fuge_n_a[]].
m_TLM_n[ film,             m_ninfl4[], m_cmn[]].
m_TLM_n[ skat,             m_ninfl3[], m_cmn[], m_tgrf_gem_fuge_e[]].
m_TLM_n[ ske,              m_ninfl3[], m_cmn[]].
```

All macro names begin with the string `m_` and take zero or more arguments between square brackets. The macro `m_TLM_n[]` expands out into the overall `sign` structure, assigning information on grammar partitioning and setting appropriate features according to the arguments.

The macros for specifying inflection and gender are used to restrict the boolean disjunction of legal suffixes for the entry in question. The optional final argument to `m_TLM_n[]` specifies possible morphographemic changes and/or fuge elements taken by the lexeme. Thus for the noun *arbejde* (work) fuge is indicated as being `...n_a` ('not applicable'), which means that there cannot be a fuge element, i.e. *arbejde* can never be used as a non-final element in a compound. *film* ((roll of) film, movie), *skat* (tax/treasure) and *ske* (spoon), on the other hand, allow compounding, taking a null fuge (the default), fuge *e* and null fuge, respectively.

Of these entries, only *skat* is indicated as allowing a morphographemic change to take place, i.e. *gemination* (consonant doubling; only occurs before *e*). For example, the final *-t* doubles with the addition of the definite suffix *-en*, forming *skatten*. Note that, given the appropriate context, gemination occurs both in compounded and non-compounded forms of the lexeme (see Section 6.3).

Although *arbejde* cannot be used directly within compounds, the exceptional form *arbejds-* can, as in *arbejdspakke* (work package). This is a rare phenomenon in Danish and must be handled with an exceptional lexical entry, where the stem is *arbejd-* and a fuge *-s-* is used.

```
%%  Example:  work (compound form)

m_TLM_n[ arbejde, arbejd,
         infl2&(null), m_ntr[], tgrf:{fuge=>(s;'mid-s'), fuge_used=>y} ].
```

Here the form of *arbejde* used for compounding, *arbejd-*, is given explicitly as the second argument. The third argument indicates that no non-null inflectional endings may be added to the stem, with the fourth argument indicating gender. The final argument is a feature structure directly setting values for the features `fuge` and `fuge_used` which force the entry to be used only as a non-final element of a compound with either a fuge *-s-* or a *mid-s* (see the rules in Section 6.4).

The following feature structure is the entry for *film* from above with the macros expanded out, allowing the forms *films* (film's), *filmen* (the film), *filmene* (the films), *filmens* (the film's), *filmenes* (the films'), and *film* (film), and disallowing gemination and *syncope* (*e*-deletion)[1]. The macros are expanded out at compile time into the feature structures used by ALEP during processing:

```
sign:{
   procinfo =>
   tprocinfo:{
      language => da,
      specinfo =>
      tspecinfo:{
         tlm => y,
         ana => n,
```

[1] This is of course superfluous, since no *e* occurs in *film*.

```
         ref => n,
         partition =>
         tpart:{
            sub => film } } },
   synsem =>
   tsynsem:{
      syn =>
      tsyn:{
         str =>
         tn:{
            lu => film,
            lemma => film,
            infl => (infl4&(s;(en;(ene;(ens;(enes;null))))))),
            grf =>
            tgrf:{
               gemination => n,
               syncope => n,
               fuge => (mid-null;(mid-s;null)) },
            agr => comm,
            masscount => count } } } }
```

Nouns with vowel shift or consonant change are coded using one of the following special templates:

```
m_TLM_nstem_change[SINGULARSTEM, PLURALSTEM, INFLECTION, GENDER]
m_TLM_nstem_change[SINGULARSTEM, PLURALSTEM, INFLECTION, GENDER, GRAPH]

%% Examples:  book, man, account

m_TLM_nstem_change[ bog,     boeg,     m_ninfl3[], m_cmn[]].
m_TLM_nstem_change[ mand,    maend,    m_ninfl4[], m_cmn[], m_tgrf_fuge_s[]].
m_TLM_nstem_change[ konto,   konti,    m_ninfl4[], m_cmn[]].
```

A single such entry is expanded out into two lexical entries, one for the singular form and one using the plural stem (given as the second argument) as the `lemma` value, i.e. the basis for morphological analysis. Note that the macro expansion has to take into account that the plural form of noun inflectional type 4 takes the null suffix, and that inflectional type 6 forms its plural with the suffix *-s*. The expanded entries for *bog* (book) are given below: the first allows the forms *bogs* (book's), *bogen* (the book), *bogens* (the book's), and *bog* (book); the second allows the forms *bøger* (books), *bøgers* (books'), *bøgerne* (the books), and *bøgernes* (the books'). Note that *bog* has no form **bøg*, indicated by the lack of the null suffix in the second entry. Note also in this entry that the plural form may not participate in compounding as a non-final element, since `fuge_used` has been set to *n*, preventing application of the relevant compounding rules during TLM processing.

```
sign:{
   procinfo =>
   tprocinfo:{
      language => da,
      specinfo =>
      tspecinfo:{
         tlm => y,
         ana => n,
         ref => n,
         partition =>
```

```
         tpart:{
            sub => bog } } },
   synsem =>
   tsynsem:{
      syn =>
      tsyn:{
         str =>
         tn:{
            lu => bog,
            lemma => bog,
            infl => (infl3&(s;(en;(ens;null)))),
            grf =>
            tgrf:{
               gemination => n,
               syncope => n,
               fuge => (mid-null;(mid-s;null)) },
            agr => (comm&sing),
            masscount => count } } } }

sign:{
   procinfo =>
   tprocinfo:{
      language => da,
      specinfo =>
      tspecinfo:{
         tlm => y,
         ana => n,
         ref => n,
         partition =>
         tpart:{
            sub => bog } } },
   synsem =>
   tsynsem:{
      syn =>
      tsyn:{
         str =>
         tn:{
            lu => bog,
            lemma => boeg,
            infl => (infl3&(er;(ers;(erne;ernes)))),
            grf =>
            tgrf:{
               gemination => n,
               syncope => n,
               fuge_used => n },
            agr => (comm&plur),
            masscount => count } } } }
```

Verbs

Coding of verbs (non-modal) is a bit simpler, since there are no vowel shifts or consonant changes. The following templates are used:

```
m_TLM_v[LU, LEMMA, INFLECTION]
m_TLM_v[LU, LEMMA, INFLECTION, GRAPH]

%% Examples:  arrive, happen

m_TLM_v[ ankomme,   ankom,    m_vinfl5[], m_tgrf_gem[]].
m_TLM_v[ ske,       ske,      m_vinfl4[]].

%% Irregular inflection:  sit

m_TLM_v[ sidde,     sid,      m_vinfl1[], m_tgrf_gem[], (~(ede;edes))].
m_TLM_v[ sidde,     sad,      m_virreg[]].
```

As seen in the examples, verbs can also be coded for gemination via an optional macro call.

Note the irregularly inflected verb *sidde* (sit), which is inflected as follows: *sidde* ((to) sit), *sid* (sit (imperative)), *sidder* (sit(s) (present)), *siddes* (be/is sat(?) (present/infinitive passive)), *siddende* (sitting (present participle)), *sad* (sat (past)), *siddet* (sat (past participle)). All forms but the past finite *sad* are covered by the first *sidde* entry in the example, which has an additional argument excluding the suffixes *-ede* and *-edes*. The second entry only covers the irregular past tense form *sad* (sat). There is no past passive form.

An advantage of this type of approach to suffixation is that one can express the relationship between some of the inflected forms of a lexeme with an existing inflectional paradigm without requiring that all inflectional forms for the lexeme correspond to the paradigm. Other approaches would require separate entries for every inflectional form of such irregular lexemes; for *sidde*, eight separate entries would be coded instead of the two given here.

Many other irregular forms consistently have a single stem allowing all non-past suffixes, i.e. everything but the past finite active, past finite passive and past participle endings. They can be coded with five arguments to `m_TLM_v[]`, as with *sidde*, but since the pattern is so frequent a macro `m_v_nopast[]` has been defined excluding these suffixes. Note that use of these macros is optional. To illustrate this, in the coding of *sige* (to say) given below, two equivalent possibilities are given for the relatively regular entry, with the two irregular forms *sagde* and *sagt* handled with one entry each:

```
m_TLM_v[ sige,              sig,          (infl2&(null;e;es;er;ende)) ].
                OR
m_TLM_v[ sige,              sig,          m_vinfl2[], m_tgrf[], m_v_nopast[]].

m_TLM_v[ sige,              sagde,        m_virreg[]].
m_TLM_v[ sige,              sag,          (infl2&t) ].
```

Either variation of the first entry covers the forms *sige* (to say), *sig* (say (imperative)), *siger* (say(s) (present finite)), *siges* (be/is said' (present/infinitive passive)), and *sigende* (saying). The second entry allows only *sagde* (said (past finite)), while the third covers *sagt* (said (past participle)).

Thus, as can be seen, by coding the endings as a boolean disjunction, the system is able to capture partial regularities in the inflections of irregular forms.

Modal verbs have a special behavior morphologically: unlike other verbs, modals have no passive, imperative or present participle forms; of the 4 remaining forms, the present is always irregular, the past and infinitive forms always identical, and the past participle is always formed with `-et`. For example, *kunne* has the inflected forms *kunne* (to be able to), *kan* (can), *kunne* (could), and *kunnet* (been able to).

These could be coded with an inflectional paradigm particular to modals, however this was not possible due to a limitation within ALEP having to do with expansion of boolean types[2].

In the Danish implementation, modals are coded with a special type `tmodal`, to distinguish them from other verbs when lifting. This is important since the morphotactics are slightly different for modals; the `-e` suffix for modals indicates the past tense *and* infinitive, while the null suffix indicates present tense.

Because of the irregular present tense form, each modal must be coded with two entries, e.g.

```
m_TLM_modal[ burde,            boer,        m_virreg[]].
m_TLM_modal[ burde,            burd,        (infl1&(e;et)) ].
m_TLM_modal[ kunne,            kan,         m_virreg[]].
m_TLM_modal[ kunne,            kun,         (infl1&(e;et)), m_tgrf_gem[] ].
m_TLM_modal[ maatte,           maa,         m_virreg[]].
m_TLM_modal[ maatte,           maat,        (infl1&(e;et)), m_tgrf_gem[] ].
m_TLM_modal[ skulle,           skal,        m_virreg[]].
m_TLM_modal[ skulle,           skul,        (infl1&(e;et)), m_tgrf_gem[] ].
m_TLM_modal[ turde,            toer,        m_virreg[]].
m_TLM_modal[ turde,            turd,        (infl1&(e;et)) ].
m_TLM_modal[ ville,            vil,         m_virreg[]].
m_TLM_modal[ ville,            vil,         (infl1&(e;et)), m_tgrf_gem[] ].
```

Adjectives

For most adjectives, the following templates are used:

```
m_TLM_adj[LU, INFLECTION, COMPARATIVEPARADIGM]
m_TLM_adj[LU, INFLECTION, COMPARATIVEPARADIGM, GRAPH]

%% Examples:  available, still/relaxed

m_TLM_adj[ disponibel,    m_adjinfl1[], m_no_comp[],      m_tgrf_sync[] ].
m_TLM_adj[ rolig,         m_adjinfl1[], m_comp_ere_st[]].
```

As with the other major stems, adjectives are coded for inflectional paradigm via macros. In addition, many adjectives allow comparative and superlative forms, which also must be indicated lexically. As before, this is done via a macro call which expands out into an explicit list of legal suffixes. Finally, both syncope and gemination are possible morphographemic changes, which also must be indicated in the lexical entry.

disponibel (available) can be inflected as *disponibel* (common, singular), *disponibelt* (neuter, singular), and *disponible* (definite/plural). As can be seen, the lexeme has no comparative form and allows syncope when inflected with a suffix beginning with *-e-*. The expanded entry is given below.

```
sign:{
   procinfo =>
   tprocinfo:{
      language => da,
      specinfo =>
      tspecinfo:{
```

[2]See Section 2.1 for the declaration of `infl` within type `tstem`; there are 7 inflectional paradigms and 35 inflectional endings, making a total of 245 combinatoric possibilities. The built-in limit is 255.

```
         tlm => y,
         ana => n,
         ref => n,
         partition =>
         tpart:{
            sub => disponibel } } },
   synsem =>
   tsynsem:{
      syn =>
      tsyn:{
         str =>
         tadj:{
            lu => disponibel,
            lemma => disponibel,
            infl => (infl1&(e;(t;null))),
            grf =>
            tgrf:{
               gemination => n,
               fuge => null } } } } }
```

Note that when syncope may occur, the value of the attribute `syncope` within the type `tgrf` is left unset. Similarly for gemination, if the change can occur, `gemination` is left unset (see Section 6.3).

rolig (calm) can be inflected as *rolig* (common, singular), *roligt* (neuter, singular), *rolige* (definite/plural), *roligere* (comparative), *roligst* (superlative, indefinite, singular), and *roligste* (superlative, definite/plural). The lexeme takes the suffixes *-ere* and *-st* in the comparative and superlative forms, respectively, with a definite/plural form of the superlative formed by combining with the suffix *-este*. The expanded entry is given below.

```
sign:{
   procinfo =>
   tprocinfo:{
      language => da,
      specinfo =>
      tspecinfo:{
         tlm => y,
         ana => n,
         ref => n,
         partition =>
         tpart:{
            sub => rolig } } },
   synsem =>
   tsynsem:{
      syn =>
      tsyn:{
         str =>
         tadj:{
            lu => rolig,
            lemma => rolig,
            infl => (infl1&(e;(t;(st;(ere;(ste;null)))))),
            grf =>
            tgrf:{
               gemination => n,
```

```
            fuge => null } } } } }
```

Adjectives with irregular comparative and superlative forms must be coded with the help of the special template:

```
m_TLM_adj_comp[ BASEFORM, COMPARATIVESTEM, COMPARATIVEPARADIGM ]

%% Examples:  old

m_TLM_adj[      gammel,     m_adjinfl1[], m_no_comp[],      m_tgrf_sync[]].
m_TLM_adj_comp[ gammel,     aeld,         m_comp_re_st[]].
```

gammel (old) is inflected as *gammel* (common, singular), *gammelt* (neuter, singular), *gamle* (definite/plural), *ældre* (comparative), *ældst* (superlative, indefinite, singular), and *ældste* (superlative, definite/plural). The first entry covers all regular forms and indicates that no comparative form can be derived from the base form *gammel*. The comparative forms are covered with a single entry using the special template which explicitly indicates what the irregular stem is, here *æld-*.

Note that although two entries for *gammel* are necessary for morphological analysis, only a single entry is necessary for structural analysis; whether the surface form is comparative or superlative will already have been determined by the suffixes found during TLM analysis, so that these forms need not have separate entries at later analysis phases.

Note also that *gammel* has degemination (deletion of the repeated consonant) occurring when inflected with a suffix beginning with *e*. This is an entirely predictable process, only occurring in conjunction with syncope. Thus there is a TLM rule for this morphographemic change, it need not be lexically coded (see Section 6.3). The fully expanded entries for *gammel* are as follows:

```
sign:{
   procinfo =>
   tprocinfo:{
      language => da,
      specinfo =>
      tspecinfo:{
         tlm => y,
         ana => n,
         ref => n,
         partition =>
         tpart:{
            sub => gammel } } },
   synsem =>
   tsynsem:{
      syn =>
      tsyn:{
         str =>
         tadj:{
            lu => gammel,
            lemma => gammel,
            infl => (infl1&(e;(t;null))),
            grf =>
            tgrf:{
               gemination => n,
               fuge => null } } } } }
```

```
sign:{
   procinfo =>
   tprocinfo:{
      language => da,
      specinfo =>
      tspecinfo:{
         tlm => y,
         ana => n,
         ref => n,
         partition =>
         tpart:{
            sub => gammel } } },
   synsem =>
   tsynsem:{
      syn =>
      tsyn:{
         str =>
         tadj:{
            lu => gammel,
            lemma => aeld,
            infl => (re;(st;ste)),
            grf =>
            tgrf:{
               gemination => n,
               syncope => n,
               fuge => null,
               fuge_used => n } } } } }
```

5.2.2 Minor stems

These are the templates for coding minor stems (adverbs, pronouns and quantifiers):

```
m_TLM_adv[LU]
m_TLM_adv[LU,LEMMA,INFL]

m_TLM_pron[LU]
m_TLM_pron[LU,LEMMA,INFL]

m_TLM_quant[LU]
m_TLM_quant[LU,LEMMA,INFL]

%% Examples:  somewhat, late

m_TLM_adv[ nogenlunde    ].
m_TLM_adv[ sent          ].
m_TLM_adv[ sent,         sene, (re;st;ste) ].

%% Examples:  all, that/it, this

m_TLM_pron[ alt          ].
m_TLM_pron[ den,         d, (e;en;et;ens;ets) ].
m_TLM_pron[ den,         dem ].
m_TLM_pron[ den,         dere, (s) ].
```

```
m_TLM_pron[ denne,       denne, (null;s) ].
m_TLM_pron[ denne,       dette, (null;s) ].
m_TLM_pron[ denne,       diss,  (e) ].

%% Examples:  other, few

m_TLM_quant[ anden,      ande, (n;t) ].
m_TLM_quant[ anden,      andr, (e) ].
m_TLM_quant[ faa         ].
m_TLM_quant[ faa,        faer, (re) ].
m_TLM_quant[ faa,        faerre, (st;ste) ].
```

As in the examples of major stems, lists of legal suffixes are indicated, though with minor stems the lists are always given explicitly instead of being compiled out from macros. The set of all suffixes possible for minor stems is a subset of the set of major stem suffixes, so it was convenient for all stems to share the same boolean type definition of `infl` (see Section 2.1). The expanded version of the pronoun *den* (that)[3] is as follows:

```
sign:{
   procinfo =>
   tprocinfo:{
      language => da,
      specinfo =>
      tspecinfo:{
         tlm => y,
         ana => n,
         ref => n,
         partition =>
         tpart:{
            sub => den } } },
   synsem =>
   tsynsem:{
      syn =>
      tsyn:{
         str =>
         tpron:{
            lu => den,
            lemma => d,
            seq => (first&last),
            infl => (e;(en;(et;(ens;ets)))) } } } }
```

Although it is a bit strange to consider *d-* as a stem, the entry exploits the bit of regularity there is within the forms of *den* (*de, den, det, dens, dets*), namely that the final characters can be used to determine agreement features of the lexeme, thereby obviating the need for a separate structural analysis lexical entry for each form. Forms with no regular interrelationships or suffixal endings must, as always, be coded as separate entries for structural analysis.

5.2.3 Words

These are the templates for coding lexemes which are neither major nor minor stems, i.e. so-called words:

[3] *den* can also be used as an article! See Section 5.3.4.

```
m_TLM_p[LU].
m_TLM_conj[LU].

%% Examples:  to, ahead of, along

m_TLM_p[ til              ].
m_TLM_p[ foran            ].
m_TLM_p[ langs            ].

%% Examples:  both, or, and

m_TLM_conj[ baade            ].
m_TLM_conj[ eller            ].
m_TLM_conj[ og               ].
```

Since words can take no suffixes, the attribute `infl` is missing from their declarations. Since words do not participate in compounding, their `seq` value is always set to `(first&last)`. The expanded version of *ifølge* (according to) is as follows:

```
sign:{
   procinfo =>
   tprocinfo:{
      language => da,
      specinfo =>
      tspecinfo:{
         tlm => y,
         ana => n,
         ref => n,
         partition =>
         tpart:{
            sub => ifoelge } } },
   synsem =>
   tsynsem:{
      syn =>
      tsyn:{
         str =>
         tp:{
            lu => ifoelge,
            lemma => ifoelge,
            seq => (first&last) } } } }
```

5.3 Lexical coding: analysis

This section presents generally the principles behind coding of the analysis lexicon and describes in particular how the main word classes have been coded.

Many of the entries of the analysis lexicon have been semi-automatically extracted from the Eurotra lexical resources.

In coding the lexicon we have tried to avoid ambiguities where possible. Therefore words which subcategorize for different word classes are coded with co-represented heads (`cohead`) following Theofilidis and Reuther (1995). `cohead` representations permit to avoid lexical disjunction in various domains, including subcategorization. For example the object for the verb *tro* (believe) can

be a nominal, a prepositional phrase or a subordinate clause. These alternatives are represented in one lexical entry, as follows:

```
sign:{
  procinfo=>mLEXSPECana[v,LU],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tv:{lu=>LU=>(tro/vente)},
           cat=>tsubst_cat:{
                  head=>thead:{
                          major=>tv_major:{ pos=>v,
                                            perf_aux=>have}},
                  subj=>[
                    tsynsem:{
                      syn=>tsyn:{
                             cat=>tsubst_cat:{
                                    head=>tsubst_head:{
                                            spr=>sat,
                                            major=>tnom_major:{
                                                     pos=>(n;pron)}}}}}],
                  compls=>[
                    tsynsem_opt_compl:{
                      syn=>tsyn:{
                             cat=>tsubst_cat:{
                                    head=>tsubst_head:{},
                                    cohead=>tcohead:{
                                              n=>tn_cohead:{
                                                   head=>tsubst_head:{
                                                           spr=>sat,
                                                           major=>tnom_major:{
                                                                    pos=>(n;pron)
                                                                      }}},
                                              v=>tv_cohead:{
                                                   head=>tsubst_head:{
                                                           marking=>at,
                                                           major=>tv_fininfin:{
                                                                    pos=>v}}},
                                              p=>tp_cohead:{
                                                   head=>tsubst_head:{
                                                           major=>tp_major:{
                                                                    pos=>p,
                                                                    prd=>no,
                                                                    p_lu=> paa}}},
                                              adj=>tnothing:{},
                                              adv=>tnothing:{}
                                                }}}} ] }}}}.
```

In the analysis lexicon, macros have not been used to the same extent as in the TLM lexicon, instead words having the same syntactic structure are grouped with a disjunction over the `lu` value.[4] The only macros used are `mLEXSPECana[CLASS,LU]` and `mLEXSPECanaref[CLASS,LU]` which assign information on grammar partitioning. The former is used for entries which are

[4] Later with the addition of the default lexical rule functionality, the most frequently occurring complement structures were implemented as defaults, simplifying the lexica by allowing us to delete some of the largest disjunctions.

applied exclusively during analysis, the latter for entries which are applied in both analysis and refinement.

For nouns, verbs and adjectives, optional complement extraction has been implemented. Optional complements within the `compls` list of a lexical entry are of type `tsynsem_opt_compl`, while obligatory complements (corresponding to the "obl" feature in the Eurotra lexicon) are coded with the subtype `tsynsem` in the `compls` list (see Section 7.3.1).

5.3.1 User-defined classes

User-defined classes are codes, numbers, dates, references, etc., found by the tagger which the user can add to the lexicon. The present lexicon contains such phenomena from the patent domain (see Chapter 3 for examples).

Since user-defined classes are actually general tags for phenomena found by the text handling tagger (see Chapter 4), the entries used for them during analysis are already of a general nature, thus no default lexical entries are used.

5.3.2 Verbs

The most common type of verb within our migration source was divalent verbs taking a nominal subject and an obligatory nominal direct object. This is the default then for verbs which cannot unify with other lexical entries.

Verbs with irregular forms must have some head information set explicitly within the lexical entries. This is done via a single logical entry using a named disjunction, where shared information is specified outside the disjunction. An example is the verb *tage* (take/go), where the named disjunction adds the information `tv_fininfin:{type=>past}` to forms with the stem `lemma=>tog`.

```
sign:{
  procinfo=>mLEXSPECana[v,LU],
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tv:{lu=>LU=>'tage', lemma=>(tag/tog) - altx},
         cat=>tsubst_cat:{
             head=>tsubst_head:{
                  major=>tv_major:{perf_aux=>vaere,pos=>v,voice=>act}=>
                     (_/tv_fininfin:{type=>past}) - altx},
             subj=>[
               tsynsem:{
                  syn=>tsyn:{
                       cat=>tsubst_cat:{
                            head=>tsubst_head:{
                                 spr=>sat,
                                 major=>tnom_major:{
                                     pos=>(n;pron),
                                     prd=>no
                                                }}}}}],
             compls=>[
               tsynsem_opt_compl:{
                  syn=>tsyn:{
                            cat=>tsubst_cat:{
                                 head=>tsubst_head:{
```

```
                                major=>tp_major:{
                                    pos=>p,
                                    prd=>no,
                                    p_lu=>fra}},
                        compls=>[]}}},
              tsynsem_opt_compl:{
                 syn=>tsyn:{
                        cat=>tsubst_cat:{
                             head=>tsubst_head:{
                                  major=>tp_major:{
                                      pos=>p,
                                      prd=>no,
                                      p_lu=>til}},
                        compls=>[]}}}
                  ]}}}}.
```

The correspondance between Eurotra `ers_frame` values and their subcategorization structures is as follows (EUROTRA 1991).

Zero-valent verbs are verbs which have no subject (Eurotra `ers_frame=f00`).

Mono-valent verbs (intransitive verbs) correspond to the `ers_frame` values *f10* and *f12*.

Divalent-verbs are grouped according to the type of object they subcategorize for (nominals, at-clauses, interrogative clauses, prepositional clauses and their combinations), corresponding to the `ers_frame` values *f210, f212, f222, f231, f232, f234, f236, f238, f242, f244, f246, f248, f40, f50*.

In the present implementation we have not implemented accusative + past participium as a possible object, thus the `ers_frame` values *f236* and *f246* have been interpreted as if they were values *f232* and *f242*, respectively.

Trivalent verbs correspond to the `ers_frame` values *f60, f70, f72, f90, f92* and *f100*. Tetravalent verbs correspond to the `ers_frame` values *f110, f120* and *f130*.

Modals are coded as subcategorizing for a subject and an infinitive. The subject of the modal is structure-shared with the subject of the infinitive.

The auxiliaries *være* (be), *have* (have) and *blive* (become)[5] subcategorize for a subject and for a past participle. The other readings of the three verbs are coded in the same entry by `cohead` representations. In the following the entry for the verb *have* is given:

```
sign:{
    procinfo=>mLEXSPECana[v,LU],
    synsem=>tsynsem:{
        syn=>tsyn:{
            str=>tv:{lu=>LU=>have,
                     lemma=>(_/havde) - altx,
                     infl=>((~irreg)/irreg) - altx},
            cat=>tsubst_cat:{
                head=>thead:{
                        major=>tv_major:{
                                 pos=>v,nex=>NEX,
                                 modal=>n,perf_aux=>have}
                         =>(_/tv_fininfin:{type=>past} ) - altx },
                subj=>SUBJ=>[tsynsem:{
                                syn=>tsyn:{
```

[5] *blive* as an auxiliary is used to form passive constructions.

```
                    cat=>tsubst_cat:{
                         head=>tsubst_head:{
                            major=>tsubst_major:{prd=>no,
                                                 pos=>(~art)}}}}}],
compls=>[
    tsynsem:{
      syn=>tsyn:{
             cat=>tsubst_cat:{
                    subj=>SUBJ,
                    head=>tsubst_head:{},
                    cohead=>tcohead:{
                              n=>tn_cohead:{
                                   head=>tsubst_head:{
                                           spr=>sat,
                                           major=>tnom_major:{
                                               pos=>n,
                                               prd=>no
                                                 }}},
                              v=>tv_cohead:{
                                   head=>tsubst_head:{
                                           marking=>unmarked,
                                           major=>tv_participial:{
                                               pos=>v,
                                               nex=>NEX,
                                               perf_aux=>have,
                                               partform=>past}}},
                              p=>tnothing:{},
                              adj=>tnothing:{},
                              adv=>tnothing:{}
                                }}}}]}}}}.
```

The auxiliary reading of the verb *ville* is implemented in a similar way, but *ville* only subcategorizes for a non-finite verb.

5.3.3 Nouns

Nouns which have a lexicalized stem change in TLM (e.g. *mand*/*mænd*) only need a single entry in the analysis lexicon.

The vast majority of nouns (within the Eurotra migration source they had `ers_frame=f00`), are nouns taking no complement (simple nouns). This is the default for structural analysis.

The default entry also handles the group of simple nouns which in the Eurotra lexicon and in the LINDA report on PAS subcategorize for a genitive (`ers_frame=f10`). In the Danish LSGRAM implementation, nouns do not subcategorize for genitive phrases; if a genitive phrase specifies a noun, its value is passed to the noun's `sign` in the `possessor` attribute during refinement. It is then up to the refinement entry for the noun to bind the genitive phrase to the correct argument in the argument structure, assuming the noun can subcategorize for a genitive phrase (see Section 5.4.2).

Similarly divalent, trivalent and tetravalent nouns whose first complement can be a genitive phrase are coded in the analysis lexicon as monovalent, divalent and trivalent nouns respectively.

An example of a monovalent noun, subcategorizing for prepositional phrases introduced by the

preposition *af* or by the preposition *for*[6] is the following:

```
sign:{
  procinfo=>mLEXSPECana[n,LU],
  synsem=>tsynsem:{
    syn=>tsyn:{
       str=>tn:{lu=>LU=>betegnelse},
       cat=>tsubst_cat:{
           head=>tsubst_head:{
                major=>tn_major:{pos=>  n,
                                 posit=>none}},
           compls=>[
             tsynsem_opt_compl:{
               syn=>tsyn:{
                   cat=>tsubst_cat:{
                        head=>thead:{
                            major=>tp_major:{
                                pos   =>p,
                                prd   =>no,
                                p_lu  =>(af;for)}},
                                compls=>[]}}}]}}}}.
```

Nouns which can act as adjuncts, i.e. temporal names, such as *dag*, *tid* (day, time), have been coded with the appropriate `posit` attribute, indicating in which position in a sentence they can occur as adjuncts (see Section 7.2). Nouns which cannot occur as adjuncts (the majority of nouns) have the value *none* for the `posit` attribute.

5.3.4 Pronouns

In the present implementation we have considered pronouns to be a large class comprising proper pronouns, the expletive *det* and all articles (of type *expletive* and *article* respectively) to avoid lexical ambiguities among i.a. the personal pronoun *den/det*, the expletive *det* and the articles *den/det* (see section 7.2).

Pronouns have no default lexical entry.

In the present implementation we have coded personal pronouns, expletives articles and possessives. The entry for the article, pronoun, expletive *den/det* is the following:

```
sign:{
  procinfo=>mLEXSPECana[pron,LU],
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tpron:{lu=>LU=>den},
        cat=>tsubst_cat:{
            head=>thead:{
                major=>tpron_major:{
                    pos=>(art;pron),
                    type=>(art;pers;expl)}},
            compls=>[]}}}.
```

Analysis and refinement information is coded within a single entry (note the macro `mLEXSPECanaref[CLASS,LU]`) for unambiguous personal pronouns, as follows:

[6] This is expressed with a boolean disjunction over the `p_lu` value.

```
sign:{
  procinfo=>mLEXSPECanaref[pron,LU],
  synsem=>tsynsem:{
    syn=>tsyn:{
         str=>tpron:{lu=>LU=>jeg,lemma=>(jeg/mig) - altx},
         cat=>tsubst_cat:{
             head=>thead:{
                  major=>tpron_major:{
                      pos=>pron,
                      type=>pers,
                      case=>(nom/acc) - altx}},
             compls=>[]}},
    sem=>tsem:{
         content=>lq_cont:{
                   rd_cont=>r_ppro:{
                       index=>INDEX=> ind_index:{
                                   pers=>p1,
                                   numb=>sing
                                     },
                       restr=>[
                               inst_zero_psoa:{
                                       rel=>rel:{
                                           rel_name=>LU},
                                       inst=>INDEX } ]}}}}}.
```

Possessive pronouns select the nominal phrase they specify by the value of attribute `selects=>tspecifiee|synsem`, to which they add a value for the feature `possessor` during refinement. Because possessive pronouns are specifiers, they also select (from the head-daughter) and project (to the mother) appropriate marking values using the two features `marking` and `newmarking`, respectively.

5.3.5 Adjectives

Adjectives with only an external argument can occur attributively (in Eurotra, `attr=yes`; in HPSG, PRD=NO), predicatively (in Eurotra, `attr=no`; in HPSG, PRD=YES) or in both uses.

Adjectives which can occur attributively select for the nominal they can modify using the `selects=>tmodifiee|synsem` value.

Predicative adjectives are grouped according to the number and the type of complements they take (subject only, subject plus one or two complements, corresponding to the following Eurotra division: adjectives with an external argument, adjectives with an external and one internal argument and adjectives with an external and two internal arguments).

A special group is formed by adjectives allowing expletive *det* constructions which correspond to the `ers_frame` values *a11, a12, a13* and *a14* (see Section 7.3.2).

Adjectives which may occur both attributively and predicatively and which have no complements are extremely common, such that this is the default.

The entry for adjectives which can only occur attributively is the following:

```
sign:{
  procinfo=>mLEXSPECana[adj,LU],
  synsem=>tsynsem:{
```

```
    syn=>tsyn:{
        str=>tadj:{lu=>LU=>(aeronautisk/afvigende/almindelig/anden/anselig/
                            ..../
                            yderlig)},
        cat=>tsubst_cat:{
           head=>thead:{
               major=>tadj_major:{
                   pos=>adj,
                   prd=>no,
                   selects=>tmodifiee:{
                        newmarking=>unmarked,
                        synsem=>tsynsem:{
                           syn=>tsyn:{
                                  cat=>tsubst_cat:{
                                      head=>tsubst_head:{
                                         marking=>unmarked,
                                         major=>tn_major:{
                                              pos=>n}}}}}}},
           compls=>[]}}
       } }.
```

The entry for an adjective with an external and an internal argument has the following structure:

```
sign:{
  procinfo=>mLEXSPECana[adj,LU],
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tadj:{lu=>LU=>'fri'},
        cat=>tsubst_cat:{
            head=>thead:{
                major=>tadj_major:{
                    pos=>adj,
                    selects=>tmodifiee:{
                        newmarking=>unmarked,
                        synsem=>tsynsem:{
                            syn=>tsyn:{
                                cat=>tsubst_cat:{
                                    head=>tsubst_head:{
                                        marking=>unmarked,
                                        major=>tn_major:{pos=>n}
                                      }}}}},
          subj=>[
            tsynsem:{
               syn=>tsyn:{
                   cat=>tsubst_cat:{
                       head=>tsubst_head:{
                            major=>tnom_major:{}}}}}],
          compls=>[
            tsynsem_opt_compl:{
              syn=>tsyn:{
                  cat=>tsubst_cat:{
                      head=>tsubst_head:{
                          major=>tp_major:{
                              pos=>p,
```

```
                        prd=>no,
                        p_lu=>(for;til),
                        p_compl=>thead:{
                            major=>tmajor:{
                                pos=>(n;pron;v)}}}},
                   compls=>[]}}}]
                }}}}.
```

5.3.6 Prepositions

Prepositions have no default lexical entry.

Prepositions can act as adjuncts thus they have the `selects=>tmodifiee` attribute. In the analysis lexicon there's only an entry for each preposition, thus we do not distinguish here among strongly bound prepositions, weakly bound prepositions and predicative prepositions (see Section 7.2), for example:

```
sign:{
  procinfo=>mLEXSPECana[p,LU],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tp:{lu=>LU=>'til'},
           cat=>tsubst_cat:{
               head=>tsubst_head:{
                   major=>tp_major:{
                            pos=>p,
                            p_lu=>til,
                            selects=>tmodifiee:{
                                  synsem=>tsynsem:{
                                     syn=>tsyn:{
                                            cat=>tsubst_cat:{
                                                     compls=>[]}}}}}},
               compls=>[
                 tsynsem:{
                   syn=>tsyn:{
                          cat=>tsubst_cat:{
                                head=>tsubst_head:{},
                                cohead=>tcohead:{
                                          n=>tn_cohead:{
                                               head=>tsubst_head:{
                                                 major=>tnom_major:{case=>acc}}},
                                          v=>tv_cohead:{
                                               head=>tsubst_head:{
                                                 marking=>at,
                                                 major=>tv_fininfin:{
                                                           pos=>v}}},
                                          adv=>tadv_cohead:{
                                                 head=>tsubst_head:{
                                                   major=>tadv_major:{}}},
                                          p=>tnothing:{}}}}]}}}}.
```

5.3.7 Adverbs

Adverbs also act as adjuncts and have the `selects=>tmodifiee` attribute.

A large number of adverbs can modify just about anything, so that the default entry for adverbs is correspondingly underspecified.

The value of the `posit` feature gives the position in which each adverb can occur, when modifying clauses. The `marking` and `newmarking` features are used to control the combinatorics of adverbs in the Actualization field (see Section 7.2.7).

5.3.8 Quantifiers

Quantifiers have no default lexical entry.

Central quantifiers are specifiers, selecting the nominal phrase they specify by the value of the attribute `selects=>tspecifiee|synsem`.

Pre- and post-quantifiers are treated as adjectives, selecting the nominal phrase they modify by the `selects=>tmodifiee|synsem` value.

All quantifiers select and project marking values using the two features `marking` and `newmarking`, respectively.

An example of entry for pre-quantifiers is the following:

```
sign:{
  procinfo=>mLEXSPECana[quant,LU],
  synsem=>tsynsem:{
      syn=>tsyn:{
          str=>tquant:{lu=>LU=>al},
          cat=> tsubst_cat:{
              head=>tsubst_head:{
                  major=>tquant_major:{
                      pos=>quant,
                      agr=>AGR,
                      selects=>tmodifiee:{
                          newmarking=>prequant,
                          synsem=>tsynsem:{
                              syn=>tsyn:{
                                  str=>tphrasal:{},
                                  cat=>tsubst_cat:{
                                      compls=>[],
                                      head=>tsubst_head:{
                                          spr=>sat,
                                          marking=>
                                            ~(prequant;cquant;postquant),
                                          major=>tn_major:{
                                              pos=>n,
                                              agr=>AGR}}}}}}}},
              compls=>[]}}}}.
```

Note that the `agr` feature for quantifiers must be set in the lexicon.

5.3.9 Functionals and punctuation marks

Functionals and punctuation marks have no default lexical entries.

The expletive *der* and the marker *at* are the only functionals we have implemented. We have coded the semantics (i.e. `content` value) of the expletive *der* in the analysis lexicon, thus the specifier macro `mLEXSPECanaref` is used.

```
sign:{
  procinfo=>mLEXSPECanaref[explet,LU],
  synsem=>tsynsem:{
            syn=>tsyn:{
                   str=>texplet:{lu=>LU=>'der'},
                   cat=>tfunct_cat:{
                          head=>thead:{
                                  major=>texplet_major:{
                                           pos=>explet}}}},
            sem=>tsem:{
                content=>lq_cont:{
                    rd_cont=>r_ppro:{
                        index=>expl_index:{
                                   expl_type=> der,
                                   pers=>p3
                                     }}}}}.
```

The marker *at* selects a finite and a non-finite clause by the `selects=>tmarkee|synsem` attribute. The `marking` and `newmarking` features are used to select and project, respectively, appropriate marking values.

Punctuation marks are currently treated as markers, selecting a sentence by the `selects=>tmarkee|synsem` attribute and selecting and projecting marking values using `marking` and `newmarking`. Since the semantics of punctuation marks has not been investigated, the implementation assigns them no semantics. Their status could change in future versions.

The same lexical entry is used for analysis and refinement via the macro `mLEXSPECanaref`.

5.4 Lexical coding: refinement

Much of the refinement lexicon has been semi-automatically extracted from Eurotra, and it has been coded similarly to the analysis lexicon, i.e. words having the same semantic structure are coded with disjunctions over the `lu` value.

The following macros have been used within the refinement lexicon:

- `mLEXSPECrefdefault[CLASS, ID]`, `mLEXSPECref[CLASS, ID]`
  Assigns appropriate partitioning information for default and non-default refinement lexicon entries, respectively.
- `m_tsynsem_GETSEM[SEM]`
  Sets the variable `SEM` to the value of the `content` of the `tsynsem`.
- `mARG0[REL]`, `mARG1[REL,A1]`, `mARG12[REL,A1,A2]`, etc.
  Expand the content of zerovalent, monovalent, divalent, etc., words, structure-sharing `REL` with the relation and the following variables `A1, A2`, etc. with the semantic arguments.

In the present implementation we do not distinguish among the different types of arguments, such as origin, measure, or goal, although this information is available from the Eurotra migration source. So that this information is not lost, it has been saved as comments to specific word entries, such that future extensions to the depth of the LSGRAM semantics can take it into account.

5.4.1 Verbs

Since the LSGRAM semantics representation is so simple, default lexical entries at refinement can handle most cases. The default rules simply assign the subject's semantics as argument 1, the direct object's semantics as argument 2, etc. As an example, here is the default refinement entry for divalent verbs.

```
sign:{
  procinfo=>mLEXSPECrefdefault[v,'default-verb-arg12'],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tv:{lu=>LU},
           cat=>tsubst_cat:{
               subj=>[m_tsynsem_GETSEM[A1]],
               compls=>[m_tsynsem_GETSEM[A2]]}},
    sem=>mARG12[LU,A1,A2]}}.
```

All verbs without this straight-forward assignment of semantics to argument structure must have a non-default entry.

Zerovalent verbs have no arguments, with the semantics only consisting of a relation name. These are identified by an empty `compls` list and an expletive pronoun subject.

```
sign:{
  procinfo=>mLEXSPECref[v,'verb-arg0'],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tv:{lu=>LU=>regne},
           cat=>tsubst_cat:{
               subj=>[tsynsem:{syn=>tsyn:{
                                  cat=>tsubst_cat:{
                                      head=>tsubst_head:{
                                          major=>tpron_major:{
                                                    pos =>pron,
                                                    type=>expl}}}}}]}},
    sem=>tsem:{
           content=>lq_cont:{
                     rd_cont=>r_psoa:{
                               psoa=>rel_psoa:{
                                       rel=>rel:{rel_name=>LU}}
                     }}}}}.
```

Monovalent verbs have a single argument, assigned as arg1 for unergatives, arg2 for unaccusatives. The entry for unaccusative verbs is the following:

```
sign:{
  procinfo=>mLEXSPECref[v,'verb-arg2'],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tv:{lu=>LU=>(aftage/avancere/blegne/blomstre/cirkulere/
                             ...
                             vokse)},
           cat=>tsubst_cat:{
               subj=>[m_tsynsem_GETSEM[A2]],
               compls=>[]}},
    sem=>mARG2[LU,A2]}}.
```

Divalent verbs have two arguments. In the lexicon there are distinct entries for verbs subcategorizing for a non-finite clause and for a prepositional phrase followed by a non-finite clause as the direct object. In the case of non-finite clauses, there is a distinction made between equi and raising verbs (see section 8.2). In the same way in verbs subcategorizing for a prepositional phrase, we distinguish the case where the complement for the prepositional phrase is a non-finite clause (see section 8.2) from the default ones, i.e. where the prepositional complement is a finite clause or a nominal.

The entry for divalent equi verbs is the following:

```
sign:{
  procinfo=>mLEXSPECref[v,'verb-arg12-equi1'],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tv:{lu=>LU=>(behoeve/bekraefte/beslutte/forberede/
                             ...
                             tilsigte/tilstraebe/undgaa/vedtage)},
           cat=>tsubst_cat:{
                  subj=>[m_tsynsem_GETSEM[A1]],
                  compls=>[
                    tsynsem:{
                      syn=>tsyn:{
                             cat=>tsubst_cat:{
                                    head=>tsubst_head:{
                                                marking=>at,
                                                major=>tv_fininfin:{
                                                    pos=>v,
                                                    type=>infin}},
                                    subj =>[m_tsynsem_GETSEM[A1]]}},
                      sem=>tsem:{content=>A2}}]}},
   sem=>mARG12[LU,A1,A2]}}.
```

The entry for divalent equi verbs where the equi construction is part of the prepositional argument is the following:

```
sign:{
  procinfo=>mLEXSPECref[v,'verb-arg12-equi-ppinfinobj'],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tv:{lu=>LU},
           cat=>tsubst_cat:{
                  subj=>SUBJ=>[m_tsynsem_GETSEM[A1]],
```

```
                compls=>[
                  tsynsem:{
                    syn=>tsyn:{
                           cat=>tsubst_cat:{
                                  head=>tsubst_head:{
                                          major=>tp_major:{
                                              p_compl=>tsubst_head:{
                                                     major=>tv_fininfin:{
                                                              pos=>v,
                                                              type=>infin}}}},
                                  subj=>SUBJ}},
                      sem=>tsem:{content=>A2}}]}},
    sem=>mARG12[LU,A1,A2]}}.
```

The entry for divalent raising verbs is as follows:

```
sign:{
  procinfo=>mLEXSPECref[v,'verb-arg2-raising'],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tv:{lu=>LU=>'synes'},
           cat=>tsubst_cat:{
                  subj=>[m_tsynsem_GETSEM[A1]],
                  compls=>[
                    tsynsem:{
                      syn=>tsyn:{
                             cat=>tsubst_cat:{
                                    subj=>[m_tsynsem_GETSEM[A1]],
                                    head=>tsubst_head:{
                                                marking=>at,
                                                major=>tv_fininfin:{
                                                    pos=>v,
                                                    type=>infin}}}},
                      sem=>tsem:{content=>A2}}]}},
   sem=>mARG2[LU,A2]}}.
```

Trivalent control verbs are treated in a similar way.

There are two different entries covering trivalent verbs where dative shifting occurs, e.g.

> *Jeg gav kruset til manden.*
> (I gave the cup to the man.)
>
> *Jeg gav manden kruset.*
> (I gave the man the cup.)

In the former example the first and the second complement in the verb's `compls` list are shared with the verb's arg2 and arg3, respectively. In the second example the first and the second complement in the verb's `compls` list are structure-shared with the verb's arg3 and arg2, respectively.

In the present implementation auxiliaries are substantives (see Section 7.2.2), thus they are treated in refinement as raising verbs. The semantics of the two auxiliaries *have* and *være* is simply to set the value of `context|background` to the concatenation of the perfect aspect (`aspect_psoa:{aspect=>perf}`) with the `context|background` list of the subcategorized for past participle. The semantics of the auxiliary *ville* is similar but in this case the aspect is set to

prospect (aspect_psoa:{aspect=>prosp}). This is done within the lexical entries for auxiliaries. The auxiliary *blive* does not influence the semantics of the subcategorized for past participle.

Modals interpret their subject as arg1 and the following verb as arg2, with the subject of the subcategorized verb structure-shared with the subject of the modal.

5.4.2 Nouns

Nouns can occur attributively and predicatively. In predicative use they do not take arguments unless they follow a copula. In this case they have an arg1 which is structure-shared with the subject of the predication.

Most nouns do not have a PAS (simple nouns). In attributive use they are all handled by the following default rule:

```
sign:{
  procinfo=>mLEXSPECrefdefault[n,'default-noun'],
  synsem=>tsynsem:{
    syn=>tsyn:{
            str=>tn:{lu=>LU},
            cat=>tsubst_cat:{
                head=>tsubst_head:{major=>tn_major:{prd=>no}},
                compls=>[]}},
    sem=>tsem:{
           content=>lq_cont:{
                      rd_cont=>r_npro:{
                           possessor=>POSSESSOR,
                           index=>INDEX=>ind_index:{},
                           restr=>[
                                 inst_zero_psoa:{
                                     inst=>INDEX,
                                     rel=>rel:{rel_name=>LU}}]}}}}}.
```

When used predicatively after a copula construction they are handled by the following default rule:

```
sign:{
  procinfo=>mLEXSPECrefdefault[n,'default-noun-arg1'],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tn:{lu=>LU},
            cat=>tsubst_cat:{
                head=>tsubst_head:{major=>tn_major:{prd=>yes}},
                subj=>[m_tsynsem_GETSEM[A1]],
                compls=>[]}},
    sem=>tsem:{
           content=>lq_cont:{
                      rd_cont=>r_npro:{
                           index=>INDEX=>ind_index:{},
                           possessor=>POSSESSOR,
                           restr=>[
                                inst_arg1_psoa:{
                                     inst=>INDEX,
```

```
                                    rel=>rel:{rel_name=>LU},
                                    arg1=>A1}]}}}}}.
```

Monovalent nouns fall into three groups: nouns whose arg1 is a genitive phrase, nouns whose arg1 is a prepositional phrase and nouns whose arg2 is a prepositional phrase. The three corresponding entries in refinement lexicon are the following:

```
% genitive is arg1
sign:{
  procinfo=>mLEXSPECref[n,'noun-arg1-gen'],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tn:{lu=>LU=>(adfaerd/aktivitet/beliggenhed/betydning/
                             ...
                             vanskelighed/vaekst)}},
            cat=>tsubst_cat:{
                  head=>tsubst_head:{
                         major=>tn_major:{
                                 prd=>no}}}},
   sem=>tsem:{
       content=>lq_cont:{
           rd_cont=>r_npro:{
                index=>INDEX=>ind_index:{},
                possessor=>A1,
                restr=>[inst_arg1_psoa:{
                           rel=>rel:{
                               rel_name=>LU},
                           inst=>INDEX,
                           arg1=>A1}]}}}}}.

% PP is arg1

sign:{
  procinfo=>mLEXSPECref[n,'noun-arg1-pp'],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tn:{lu=>LU=>(balance/beloeb/erstatning/
                             ...
                             variation)},
            cat=>tsubst_cat:{
                  head=>tsubst_head:{
                         major=>tn_major:{
                                 prd=>no}}}},
                  subj=>[],
                  compls=>[m_tsynsem_GETSEM[A1]]}},
   sem=>tsem:{
       content=>lq_cont:{
           rd_cont=>r_npro:{
                index=>INDEX=>ind_index:{},
                possessor=>POSSESSOR,
                restr=>[inst_arg1_psoa:{
                           rel=>rel:{rel_name=>LU},
                           inst=>INDEX,
                           arg1=>A1}]}}}}}.
```

```
% PP is arg2
sign:{
  procinfo=>mLEXSPECref[n,'noun-arg2-pp'],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tn:{lu=>LU=>(hold/perspektiv/
                             ...
                             tvivl/vidne/vifte/virkeliggoerelse/vaerktoej)},
           cat=>tsubst_cat:{
                  head=>tsubst_head:{
                         major=>tn_major:{
                                  prd=>no}}},
                  subj=>[],
                  compls=>[m_tsynsem_GETSEM[A2]]}},
   sem=>tsem:{
       content=>lq_cont:{
           rd_cont=>r_npro:{
                index=>INDEX=>ind_index:{},
                possessor=>POSSESSOR,
                restr=>[inst_arg2_psoa:{
                           rel=>rel:{rel_name=>LU},
                           inst=>INDEX,
                           arg2=>A2}]}}}}}.
```

In the first case the `content` of arg1 is structure-shared with the `possessor` attribute of the nominal. The `possessor` attribute is sat in refinement rules.

Divalent, trivalent and tetravalent nouns are coded in a similar manner. Also in these cases the first argument can be a genitive, thus there are two kinds of entries for the nouns in each group.

A special group is formed by temporal nouns which can act as adjuncts modifying clauses. They have the same semantic structure as other nominals, but when used as clausal adjuncts their semantics is assigned to the restrictions list of the mother node, similar to the treatment of other types of adjuncts. This is not done in the lexicon but in refinement rules (see Section 8.4).

5.4.3 Pronouns

Personal pronouns which are unambiguous syntactically and semantically (e.g. possessive pronouns) have a single entry used for both analysis and refinement.

Ambiguous personal pronouns, such as articles, non-possessives and expletives, have a single analysis entry, with the ambiguity being resolved via the multiple refinement entries.

Articles (which were not distinguished from pronouns in the analysis lexicon) have a treatment in refinement similar to that of central quantifiers, in that they add their `q_force` value to the `quants` list of the nominal phrase they specify.

The three refinement entries for pronouns are given below, for the personal pronouns *et/en/den/det*, the articles *et/en/den/det*, and the expletive *det*, respectively.

```
%% B.1 PRONOUNS type=>(~(art;expl))

sign:{
  procinfo=>mLEXSPECref[pron,'pron-non-art-expl'],
```

```
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tmorphol:{lu=>LU},
        cat=>tsubst_cat:{
           head =>tsubst_head:{
               major=>tpron_major:{
                   pos =>pron,
                   type=>(~(art;expl))}}}},
     sem=>tsem:{
         content=>lq_cont:{
             rd_cont=>r_ppro:{
                index=>INDEX=>ind_index:{
                           pers=>p3,
                           numb=>sing,
                           gend=>neut},
                possessor=>POSSESSOR,
                restr=>[ inst_zero_psoa:{
                                rel=>rel:{
                                      rel_name=>LU},
                                      inst=>INDEX } ]}}}}}.

%% B.2 PRONOUNS type=>art (default)

sign:{
  procinfo=>mLEXSPECrefdefault[pron,'default-pron-art'],
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tmorphol:{lu=>LU},
        cat=>tsubst_cat:{
           head =>tsubst_head:{
               major=>tpron_major:{
                   pos =>art,
                   case=> ~gen,
                   type=>art,
                   prd => _}}}},
    sem=>tsem:{
        content=>lq_cont:{
            quants=>[quantifier:{q_force=>LU}]}}}}.


%% B.3 PRONOUNS type=>expl (default)

sign:{
  procinfo=>mLEXSPECrefdefault[pron,'default-pron-expl'],
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tmorphol:{lu=>LU},
        cat=>tsubst_cat:{
           head =>tsubst_head:{
               major=>tpron_major:{
                   pos =>pron,
                   type=>expl}}}},
     sem=>tsem:{
         content=>lq_cont:{
```

```
                rd_cont=>r_ppro:{
                    index=>expl_index:{
                              expl_type=>det,
                              pers=>p3,
                              numb=>sing}}}}}}.
```

Note that the article and expletive readings are coded as defaults. This is done because articles and expletives are easily spotted during syntactic analysis, while e.g. the personal pronoun reading of *det* in *Det er et stort æble.* (It/That is a big apple.) is more difficult to identify, since a general head-subject rule is used to parse the pronoun as the subject within which it is not possible to access the `type` feature in order to restrict the pronominal type to non-expletives. The entries above means that 'pronouns' identified as expletives or articles will fall through to the correct default readings, while other pronouns will be assigned the preferred personal pronoun reading.

5.4.4 Adjectives

Adjectives taking only an external argument have the following two default entries, the former applied when they occur predicatively, the latter when they occur attributively.

```
%% monovalent adjectives, predicative use

sign:{
  procinfo=>mLEXSPECref[adj,'default-adj-arg1-pred'],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tadj:{lu=>LU},
           cat=>tsubst_cat:{
                  head=>tsubst_head:{
                         major=>tadj_major:{
                                  prd=>yes}},
                  subj=>[m_tsynsem_GETSEM[A1]]}},
    sem=>mARG1[LU,A1]}}.

%% monovalent adjectives, attributive use

sign:{
  procinfo=>mLEXSPECref[adj,'default-adj-arg1-attr'],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tadj:{lu=>LU},
           cat=>tsubst_cat:{
                  head=>tsubst_head:{
                         major=>tadj_major:{
                                  prd=>no,
                                  selects=>tmodifiee:{}}},
                  subj=>[m_tsynsem_GETSEM[A1]]}},
    sem=>tsem:{
           content=>lq_cont:{
               rd_cont=>rd_cont:{
                      restr=>[arg1_psoa:{
                                   rel => rel:{
                                       rel_name => LU,
                                        rel_sort => rel_sort:{} },
```

```
                                        arg1 => A1 }]}}}}}.
```

In predicative use an adjective is just assigned the correct PAS structure, while in attributive use it is an adjunct and therefore must end up with a restrictions list containing a single element which is prepended to the restrictions list of the modified element via structural refinement rules (see Section 8.4). All monovalent adjectives are currently handled via these default rules.

The defaults for attributive and predicative uses of divalent adjectives are very similar to the monovalent defaults.

There are special entries for adjectives taking an *at*-clause as arg1, and for those taking an *at*-clause as arg1 and a dative perceiver as internal argument, for example:

> At snyde er forbudt. (lit. To cheat is forbidden.)
> At arbejde er godt for ham. (lit. To work is good for him.)

The corresponding expletive *det* raising constructions are handled by lexical rules in analysis (see Section 7.3.2), allowing the formulations

> Det er forbudt at snyde. (It is forbidden to cheat.)
> Det er godt for ham at arbejde. (It is good for him to work.)

Adjectives whose internal arguments are prepositional phrases have an entry covering the case in which the prepositional phrase has a nominal phrase as its complement (arg2 is structure-shared with the prepositional phrase), and another entry for non-finite verbs as prepositional complements. Non-finite prepositional complements are arg2 in the PAS structure of the adjective, but at the same time they share the subject (arg1) with the adjective (equi construction), as can be seen in the following entry:

```
sign:{
  procinfo=>mLEXSPECref[adj,'adj-arg12-pred-"at"-pred'],
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tadj:{lu=>LU=>(adgangsberettiget/afskaermende/aktuel/alene/
                               ...
                               oekonomisk/oenskvaerdig)},
           cat=>tsubst_cat:{
                  head=>tsubst_head:{
                         major=>tadj_major:{
                                  prd=>yes}},
                  subj=>[m_tsynsem_GETSEM[A1]],
                  compls=>[
                    tsynsem_opt_compl:{
                      syn=>tsyn:{
                             cat=>tsubst_cat:{
                                    subj=>[m_tsynsem_GETSEM[A1]],
                                    head=>tsubst_head:{
                                            major=>tp_major:{
                                                     pos=>p,
                                                     p_lu=>til,
                                                     p_compl=>tsubst_head:{
                                                        major=>tv_major:{
                                                                 pos=>v}}}}}},
                      sem=>tsem:{content=>A2}}]}},
    sem=>mARG12[LU,A1,A2]}}.
```

Adjectives with an external and two internal arguments are treated in a similar way.

5.4.5 Prepositions

There is no default rule for prepositions.

Prepositions used attributively (i.e. after analysis the `prd` value is *no*) have the following entry:

```
sign:{
 procinfo=>mLEXSPECref[p,'p-attr'],
 synsem=>tsynsem:{
   syn=>tsyn:{
          str=>tp:{lu=>LU},
          cat=>tsubst_cat:{
                 subj=>SUBJ,
                 head => tsubst_head:{
                           major=>tp_major:{
                                    pos => p,
                                    prd => no
                                      }},
                 compls=>[tsynsem:{syn=>tsyn:{
                                         cat=>tsubst_cat:{
                                                subj=>SUBJ,
                                                head=>tsubst_head:{}}},
                                      sem=>SEM}]}},
   sem=>SEM}}.
```

They are treated as having no semantics of their own, inheriting the entire semantics of the complement. They structure-share their `subj` list with that of their complement.

Predicatively used prepositions act as adjuncts modifying clauses and nominal phrases or following a copula. The former type take their complement as arg2, but they do not have an arg1. The latter type have both an arg1 and an arg2, where arg2 is again the prepositional complement and arg1 is structure-shared with the subject for the copula.

Since adjuncts are not semantic heads, the semantics of prepositions need not conform to the semantics of the elements they modify (cf. Theofilidis et al. (1994)). Thus a single entry is sufficient for handling prepositions as adjuncts.

```
sign:{
  procinfo=>mLEXSPECref[p,'p-pred'],
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tmorphol:{lu=>LU},
        cat=>tsubst_cat:{
            head=>tsubst_head:{
                major=>tp_major:{
                    pos => p,
                    prd => yes,
                    selects=>tmodifiee:{}}},
            subj=>[],
            compls=>[m_tsynsem_GETSEM[A2]]}},
    sem=>tsem:{
        content=>lq_cont:{
```

```
            rd_cont=>rd_cont:{
                restr=>[arg2_psoa:{
                             rel=>rel:{
                                  rel_name=>LU,
                                 rel_sort=>rel_sort:{}},
                             arg2 =>A2}]}}}}}.
```

Note that the `subj` list is empty in the above entries.

The entry for prepositional phrases after a copula is the following:

```
sign:{
  procinfo=>mLEXSPECref[p,'p-pred-copula'],
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tmorphol:{lu=>LU},
        cat=>tsubst_cat:{
            head =>tsubst_head:{
                 major=>tp_major:{
                    pos => p,
                    prd => yes}},
            subj=>[m_tsynsem_GETSEM[A1]],
            compls=>[m_tsynsem_GETSEM[A2]]}},
  sem=>mARG12[LU,A1,A2]}}.
```

5.4.6 Adverbs

In the present implementation we have only treated adverbs which modify clauses, so their entry in refinement is similar to that of prepositional phrases modifying clauses. This entry is very general and covers all adverbs. Thus there is no default rule.

```
sign:{
  procinfo=>mLEXSPECref[adv,LU],
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tmorphol:{lu=>LU},
        cat=>tsubst_cat:{
            head =>tsubst_head:{
               major=>tadv_major:{
                   pos => adv,
                   prd =>yes,
                   selects=>tmodifiee:{}}}}},
    sem=>tsem:{
        content=>lq_cont:{
            rd_cont=>r_psoa:{
                    restr=>[rel_psoa:{
                             rel=>rel:{
                                  rel_name=>LU,
                                  rel_sort=>rel_sort:{}}}]}}}}}.
```

5.4.7 Quantifiers

There is no default rule for quantifiers.

In contrast to Theofilidis et al. (1994) no quantifiers are semantic heads. This has a simplifying effect on the treatment of quantifiers at refinement, where a single rule is sufficient for all quantifiers, whether functioning as modifiers or specifiers.

```
sign:{
  procinfo=>mLEXSPECref[quant,'quant'],
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tmorphol:{lu=>LU},
        cat=>tsubst_cat:{
           head =>tsubst_head:{
               major=>tquant_major:{
                   pos => quant,
                   prd => _,
                   selects=>tspecmod:{}}}}},
    sem=>tsem:{
         content=>lq_cont:{
             quants=>[quantifier:{q_force=>LU}]}}}}.
```

The attribute value `selects=>tspecmod` generalizes over specifiers and modifiers (see Section 2.1.9). As can be seen here, all quantifiers have a single element on their `quants` list, which is prepended to the `quants` list of the modified or specified element via a single structural refinement rule (see Section 8.4).

5.4.8 Functionals and punctuation marks

See Section 5.3.9.

Chapter 6

Morphological Analysis

Word segmentation is the morphological analysis phase of processing.

The morphology implementation documented here includes comprehensive morphological analysis via lexical entries, TLM and lifting rules. The key aspects described here are: lexicalization of some stem changes; non-compositional inflectional suffixation; the featurization of inflectional suffixes and fuge elements; and morphotactics during lifting.

6.1 Aspects of the approach

In ALEP, Danish morphology is implemented over several phases, viz. TLM analysis, lifting and structural analysis. Key aspects of the implementation are:

Use of booleans to reduce ambiguity

The implementational priorities of simplicity and efficiency have prompted changes in the *Typed Feature System* (TFS) relative to the document Braasch & Jørgensen (1995) in order to handle the particular ambiguities within the Danish language as efficiently as possible. For example, Danish has a passive inflection not only of the present finite form but of the past finite and infinitive forms as well. One particular verbal suffix *-es* is ambiguous as to whether it forms the present or infinitive passive form. Similarly, for three verbal inflectional types there is a homographic ambiguity between their imperative forms, and either their infinitive or simple past forms, viz. inflectional types *vinfl3, vinfl4, vinfl5*. Within the Danish implementation, the TFS is defined in such a way that these ambiguities can be expressed via a boolean disjunction over an atomic value. This means that instead of lifting into two or more incompatible (i.e. non-unifiable) objects, the ambiguity is restricted to the atomic feature value within a single object, eventually being disambiguated by context. (See Section 6.2.2.)

Word class groups: major stems, minor stems, words

For morphological processing, word classes have been given a tripartite grouping into major stems, minor stems, and so-called words. *Major stems* have relatively complex inflectional morphology (enough to justify specification of inflectional paradigms) and morphographemic changes to the stem. **Nouns**, **verbs** and **adjectives** are major stems. *Minor stems* have a very weak inflectional morphology and no syncope or gemination can apply to their stems. This group includes **adverbs**, **articles**, **pronouns** and **quantifiers**. *Words* have no inflectional morphology whatsoever, consisting only of **conjunctions** and **prepositions**. Other word classes have yet to be implemented.

This grouping allows exploitation of the pseudo-inflectional endings that are to be found in a

number of word classes other than nouns, verbs and adjectives. This results in a cleaner, simpler analysis lexicon, avoiding having to code e.g.: comparative and superlative forms of adverbs or quantifiers; genitive forms of pronouns; and gender-specific forms of quantifiers, pronouns and articles.

The price to be paid is the introduction of somewhat disputable stems being assigned to some of these minor stems. For instance, the comparative and superlative forms of *meget 'much' (quantifier)* are *mere, mest, meste*; once the 'regular' comparative and superlative suffixes have been removed from these, the resulting stem is *me-*. Note however that this stem is only used during morphographemic (TLM) analysis and does not appear in the analysis and refinement entries, which are coded by the lexical unit (*meget*). The approach is adopted here despite the objectionable stems since it does not commit the implementation to any theoretical claims, being simply a practical exploitation of the regularity of the sometimes occurring interpretable suffixes on minor stems which otherwise are ignored given a more thoroughly lexicalistic approach.

Lexicalized vowel shift and consonant change

Vowel shift and consonant change occur in inflected forms of some stems in Danish. These could be processed via TLM rules; since the change is not predictable, each lexeme would have to be coded for the specific stem change applying to it, either by using a diacritic or a feature-based approach. A series of TLM rules would then take this coding into account, applying the stem change.

Consistent with the recommendation of Underwood & Jørgensen (1995), a lexicalized approach has been chosen here, basically because vowel shift and consonant change cannot be considered productive in Danish (Underwood & Jørgensen (1995), p.3), i.e. these morphographemic processes only apply to a relatively small number of tokens, and are not applied to new tokens entering the language. Thus as can be seen below (Section 5.2), entries with changes in the vowel or consonant are coded as separate lexical entries. For nouns, a single logical lexical entry is expanded out into the two physical entries, each specifying its own set of legal suffixes.

Non-compositional suffixation

Danish has a number of inflectional suffixes which could be handled compositionally. However, as argued in Underwood & Jørgensen (1995), Section 1.1.2, this can lead to unnecessary complications in TLM and word formation, unnecessary given the limited inflectional morphology of Danish.

Seen technically, this is a perfect example of an area where practicality has dictated a compromise. Suffixes are analyzed non-compositionally here in order to simplify the TLM implementation, and to make featurization of the suffixes an easier matter (see below). Non-compositionally, then, the complete set of Danish suffixes is the following:

```
null
e n r s t
en er es et ne ns re rs st te ts
ede ene ens ere est ers ets nes rne ste tes
edes ende enes erne este rnes
ernes
```

Featurized inflectional suffixes and fuge elements

In variance with the approach described in Underwood & Jørgensen (1995), suffixes and fuge elements have been featurized; the TLM rules delete the inflectional suffixes and fuge elements, assigning a value to a feature within the preceding major stem which is accessed during lifting for assignment of agreement or other information (see Section 6.2.2).

This featurization results in a simpler set of lexicon and grammar rules. No entries for inflectional suffixes or fuge elements are necessary, so that the lexicon consists entirely of relatively 'substantial'

elements. Fewer head declarations and fewer word formation rules are necessary, the latter now restricted to treating the general case (a word consisting of a single morpheme), compounding, and implementation of some lexical rules.

Each suffix combination is implemented as a separate TLM rule, as opposed to the compositional approach with perhaps one TLM rule each suffixal element. However, since a list of the legal suffixes is explicitly given for each lexical entry, either manually by the grammar developer or via macro expansion, separate TLM rules are not required for suffixation within each word class; in other words, each TLM rule can potentially apply to any word class allowing the suffix. This reduces the potential number of rules compared to other approaches. Finally, since the TLM rules are very explicit, their application is correspondingly very efficient.

Morphotactics during lifting

Since lifting rules can only apply to single nodes at a time, and not, say, to stem-suffix combinations, other implementations must account for the morphotactics of non-featurized inflectional suffixes within word formation rules during structural analysis. Lifting, in this context, is only a conversion between the `lg_ts_chunk` structure to which TLM analysis is applied and the analysis and refinement can apply.

Since now suffixation information is contained within the stem as a feature value, and since lifting rules allow full specification of linguistic attribute information as constraints on application of the rules, morphotactics can be done during lifting. This gives the following advantages:

- **Speed**

  Doing morphotactics in this way is quick.

- **Ease of development**

  By moving processing of morphotactics from structural analysis to lifting, development of morphology was made easier. This was also advantageous during further grammar development, since a full trace was not necessary when trying to discover whether some offending feature was added by a morphotactic rule, since application of these is completely finished before structural analysis begins. In other words, it was an easier matter to examine the results of morphotactics separate from other types of processing and thereby break down the development process into more easily manageable chunks.

- **Conceptualization**

  Morphographemics during TLM analysis, morphotactics during lifting, then practically all morphology is finished by the time structural analysis begins. The exception is the interpretation of compound elements, which in many languages *must* be handled during structural analysis in any case.

6.2 Inflectional morphology

6.2.1 Parsing (TLM)

The Danish character set

```
define(
alphabet,
{a,ä,b,c,d,e,ë,f,g,h,i,ï,j,k,l,m,n,o,ö,p,q,r,s,t,u,ü,v,w,x,y,z,æ,ø,å,'-'}).

define(
consonants,
{b,c,d,f,g,h,j,k,l,m,n,p,q,r,s,t,u,v,w,x,y,z}).
```

```
define(
vowels,
{a,ä,e,ë,i,ï,o,ö,u,ü,y,æ,ø,å}).
```

These are the character sets declared for Danish which are used as constraints within TLM rules. Although Danish does not make use of umlaut, umlauted letters are included here due to the occurrence of foreign (often German) words in patent documents. Hyphen '-' is included since it functions as a fuge element.

Default rules

The usual TLM default rule has been replaced by the following two default rules:

```
tlm_rule(
  default,
  [X1] [Y] [] => [X2] [Y] [],
  _,
  [X1 in alphabet, X2 in alphabet, Y in alphabet]
        ).

tlm_rule(
  default_initial_morpheme,
  [=] [X] [] => [+] [X] [],
  _,
  tmorphol:{seq=>first},
  [X in alphabet]
        ).
```

The important change is the specification of a preceding context, which is necessary for two reasons:

- **Restricting the gratuitous fuge to nouns**
  Danish noun compounds with three or more elements almost always take a 'gratuitous' fuge (*-s-* or null) after the non-initial morphemes. This is regardless of which fuge element is lexically coded within the entry for the morpheme in question, so the phenomena had to be handled within the TLM rules. However in order to restrict the rule allowing this gratuitous fuge from being inserted after word-initial morphemes, the initial morpheme had to be identified accurately. This could only be done by identifying the word boundary preceding the first character of the word, as is done in the rule `default_initial_morpheme`.

- **Restricting compounding to noun and adjective heads**
  Allowing compounds headed by word classes other than nouns or adjectives is currently considered too inefficient and prone to overgeneration. Therefore compounding has been restricted to those compounds with nouns or adjectives as their head.

  This is not as straightforward as it may seem, since in ALEP TLM rules, when inserting an extra character or morpheme boundary, constraints can only be placed on what occurs to the left. What was needed was a way to constrain what occurred to the right of the new morpheme boundary. Simple rules such as the following

  ```
  tlm_rule(
    compound_straw_man1,
    [] [Y] [] => [] [Y] [],
    _,
    tn:{seq=>~first},
  ```

```
  [Y in alphabet]
        ).

    OR

tlm_rule(
  compound_straw_man2,
  [X] [Y] [] => [+] [Y] [],
  _,
  tn:{seq=>~first},
  [X in alphabet, Y in alphabet]
        ).
```

are not sufficient for this, since the usual default rule could also apply to the same character index here by the variable `Y`, causing overgeneration. This problem *cannot* be avoided by using the obligatory operator in either of these rules, since this operator only takes precedence in situations where the surface sequences to be mapped are different, while here, the sequences are the same, it is only the information unified with the resulting morpheme that is different.

To avoid this overgeneration, what was needed was a way to have mutually exclusive contexts for the default rule and some new rule restricting compounding. This meant that preceding context had to be able to be specified within the default rule, restricting it to applying in characters where a morpheme boundary is *not* inserted immediately before. This is necessarily implemented as two default rules, one where the preceding character is within the alphabet, where the value of `seq` is not touched, the other where there was no preceding character, i.e. word-initial context, where the value of `seq` is correspondingly set to `first`[1].

Stems

There are 35 inflectional suffixes in Danish, considered non-compositionally and including the null suffix.

The job of the inflectional TLM rules is to identify the suffixes, featurize them and add relevant morphographemic information to the resulting structure. Considered *morphographemic* is sequence information, the continuation (or following character(s)) of each morpheme, and the identification of the suffixes. `fuge=>n` is also set within these rules, since if there is an inflectional suffix attached to a morpheme, there cannot also be a fuge element (see Section 6.4).

As described above, the macro calls used for coding the lexical entries expand out into an exhaustive list of legal suffixes for each stem (see Section 5.2). Each of the 35 possible suffixes has a single TLM rule within which the value of the attribute `infl` is restricted to being the string value of the suffix itself. As is typical with unification, this specification acts simultaneously as a constraint and an assignment; the disjunctive list within the lexical entry constrains which TLM suffixation rules can potentially apply and vice versa, while the actual application of one of the rules sets the value within the resulting feature structure to be one and only one member of the disjunction.

Note that since word class-specific inflectional information is already taken into account at macro expansion, no word class information is necessary within the TLM rules; the TLM rule accounting for the suffix *-e* can apply to any stem, be it noun, verb, adjective, article, pronoun or quantifier.

As an example, consider the expanded version of the lexical entry for *film* given above, the `str` attribute of which is repeated here:

[1] Note that the specification of the word boundary symbol within the preceding context was not possible with ALEP as delivered, but was made possible by a patch provided by Cray Systems.

```
...
str          =>
     tn:{
        lemma         => film,
        lu            => film,
        infl          =>
             (infl4&(s;(en;(ene;(ens;(enes;null))))))),
        grf           =>
             tgrf:{
                gemination    => n,
                syncope       => n,
                fuge          => (null) } }
...
```

With the input *filmen*, the following TLM rule applies:

```
tlm_rule(
  suffix_en,
  [X] [e,n,=] [] => [] [+] [],
  _,
  tstem:{
    infl=>en,
    grf=>tgrf:{fuge=>'n'},
    contin=>'-en',
    seq=>last},
  [X in alphabet]
        ).
```

After the application of `suffix_en` to the suffix and final word boundary, `default_initial_morpheme` to the initial *f*, and `default` to the characters *i,l* and *m*, `str` has the following value:

```
...
str          =>
     tn:{
        lemma         => film,
        lu            => film,
        infl          =>
             (infl4&en),
        grf           =>
             tgrf:{
                gemination    => n,
                syncope       => n,
                fuge          => n&(null) },
        contin=>'-en',
        seq=>(first&last) }
...
```

Words

Non-stems (i.e. words) have no inflection and thus all have the following TLM rule applied:

```
tlm_rule(
```

```
word_end,
[] [=] [] => [] [+] [],
_,
tword:{
  seq=>(first&last)}
      ).
```

Specification of `seq=>(first&last)` is actually redundant, since words are lexically coded with this attribute value.

6.2.2 Morphotactics (lifting)

General information lifted

Each word class has a separate lifting rule interpreting suffixes for stems and creating the `lg_LS` linguistic structure for all word forms. In addition to the morphological information lifted and assigned by the rules, there is general information added for all nodes, both morphemes and non-morphemes. These include

- **projectional/construction type information**
  All morphemes are assigned `constr=>lexical`, all minimal phrasal projections are assigned `constr=>word`.
- **rule id information**
  The rule id of the lifting rule which has applied.
- **orthographic information**
  This is used in conjunction with the same structures at the phrasal level for identifying the input string from a given point on. The continuation string assigned at TLM analysis is exploited for this, such that within the `ortho` value of the topmost phrasal projection one can see all morphemes, fuge elements, suffixes, suffix-less stems, and tagged word constructs (Music 1995b) found within the input.
- **lexicon and grammar partitioning information**
  To speed up processing, the lexica and grammars have been partitioned (see Section 5.1).

Mass and count nouns

A well-known problem area within HPSG-inspired implementations is the distinction between count and mass nouns, and their influence on syntax. In Danish, as in English, singular indefinite count nouns must cooccur with a preceding specifier. Plurals count nouns, definite count nouns, and mass nouns do not require the presence of a specifier.

For the LSGRAM implementation described here, it was necessary to code the mass/count distinction already within the TLM lexicon. This is because determining whether the specifier in question is subcategorised for depends on both lexical information (i.e. the type of the noun itself, mass vs. count) and on morphological information (i.e. whether a definite enclitic or plural suffix (or a combination of these) is present). This is a similar problem to the interpretation of suffixes as regards agreement information — the (inflectional) type of the noun combined with the suffix found gives the interpretation.

Determination of the subcategorisation information for a preceding specifier is in essence then a morphotactic problem, and therefore is done during lifting. This requires that the mass/count

distinction is coded within the TLM lexicon, since this is the only lexical information available at the time of lifting.

The attribute `spr` is a boolean containing information on what specifiers the noun feature structure combine with. If the value is `spr=>sat`, the structure is either a projection of a count noun which has been combined with a specifier, or a mass noun not requiring a specifier. Of course, since mass nouns allow specifiers to precede them, their `spr` value is not set strictly to `sat`, but `sat` is allowed as a possible value, while for singular indefinite count nouns, `sat` may not be the `spr` value.

Phrase structure rules combining noun feature structures with other types require that the noun feature structure has its `spr` value set to `sat`. This enforces then that singular indefinite count nouns must be combined with a specifier.

Major stems

Lifting rules for major stems are naturally the most complex. Agreement information is assigned to the lifted `lg_LS` objects. The agreement information assigned to suffixes depends on the word class and the inflectional type of the morpheme, since many suffixes are ambiguous between and within word classes.

The inflectional types are preserved as part of the boolean value of the attribute `infl`. This is a very nice advantage to the technical possibility provided by ALEP of declaring booleans over several sets of values (this is also exploited for the attribute `fuge` for compounding, see Section 6.4). Since the inflectional paradigm (for major stems only, of course) is part of the same attribute value as the suffix found via TLM analysis, it is an easy matter to interpret all possible combinations of them within a large named disjunction, with a corresponding named disjunction assigning the features which are the interpretation of the given inflectional type/suffix combination.

The following is the lifting rule for nouns with the `selects` value excluded:

```
ts_ls_rule(
  sign:{
    procinfo=>tprocinfo:{
       specinfo=>tspecinfo:{tlm=>n,partition=>tpart:{main=>n,sub=>LU}},
       ruleinfo=>truleinfo:{lift_id=>'Mlift-noun'}},
    ortho=>tortho:{
       string=>[LEMMA|[CONT|REST]],rest=>REST},
    synsem=>tsynsem:{
       syn=>tsyn:{
          str=>CAT=>tn:{
              constr=>lexical,
              lu=>LU,
              lemma=>LEMMA,
              contin=>CONT,
              agr=>AGR,
              seq=>(_/last/last/_/last/
                      last/last/_/last/last/last/
                      last/last/last/last/last) - altx,
              masscount=>MASSCOUNT=>(
                mass/mass/mass/count/count/
                  count/count/count/count/count/_/
                  _/_/_/_/_ ) - altx,
              infl=>(
                (infl1&null; infl2&null; infl3&null;
```

```
                infl4&null; infl5&null; infl6&null)/
               (infl1&s;    infl2&s;    infl3&s;
                infl4&s;    infl5&s;    irreg&s)/
               (infl6&s)/
               (infl1&null; infl2&null; infl3&null;
                infl4&null; infl5&null; infl6&null)/
               (infl1&s;    infl2&s;    infl3&s;
                infl4&s;    infl5&s;    irreg&s)/
               (infl6&s)/ % ambiguous 1
               (infl6&s)/ % ambiguous 2
               (irreg&null)/
               (infl4&null)/  % ambiguous
               (infl4&s)/     % ambiguous
               (t;et;n;en)/
               (ts;ets;ns;ens)/
               (e;r;er)/
               (ene;rne;erne;ne)/
               (es;rs;ers)/
               (enes;rnes;ernes;nes) ) - altx },
          cat=>tsubst_cat:{
              cohead=>tcohead:{n=>tn_cohead:{head=>COHEAD}},
              head=>COHEAD=>tsubst_head:{
                      marking=>(unmarked/unmarked/unmarked/unmarked/unmarked/
                                  unmarked/unmarked/unmarked/unmarked/unmarked/
                                  defin/defin/unmarked/defin/unmarked/defin)
                        - altx,
                      spr=>((sat;unsat)/(sat;unsat)/(sat;unsat)/unsat/unsat/
                              unsat/unsat/(sat;unsat)/(sat;unsat)/(sat;unsat)/
                              sat/sat/(sat;unsat)/sat/(sat;unsat)/sat)
                        - altx,
                      major=>tn_major:{
                               pos=>n,
                               masscount=>MASSCOUNT,
                               agr=>AGR=>(sing/sing/sing/sing/sing/
                                           sing/plur/_/plur/plur/def&sing/
                                           def&sing/plur/plur&def/plur/plur&def)
                                 - altx,
                               case=>((~gen)/gen/gen/(~gen)/gen/gen/(~gen)/
                                      (~gen)/(~gen)/gen/(~gen)/gen/(~gen)/
                                      (~gen)/gen/gen)
                                 - altx,
                               def_enclitic=>(n/n/n/n/n/n/n/n/n/n/y/y/n/y/n/y)
                                 - altx,
                               selects=>...
  'M',['CAT'=>CAT,'STATUS'=>'OK'],_STR).
```

The value of `selects` has been excluded from this example due to its size. Its value is set according to whether the noun occurs in genitive case or not: genitive nouns get `selects=>tspecifiee:{}`, while non-genitives get `selects=>tmodifiee:{}`, used in cases where a noun functions as a temporal adverbial.

As can be seen from the last line of this rule, application is only attempted when the output from word segmentation has the tag `M`. The value of type `CAT` is assigned to the local variable of the same name and thereby unified with the `str` type of the resulting `lg_LS` object. It is this unification

which constrains application of this lifting rule for nouns from applying to every structure with tag `M`.

The inflectional type and suffix specifications given as the value of `infl` by lexical coding and the application of TLM rules are used within the named disjunction `altx`. Since there are 16 distinct combinations requiring treatment, there are 16 elements of the disjunction, and the rule expands out into the same number of lifting rules.

The agreement information assigned to an inflectional type/suffix combination can be found by locating the corresponding disjunction members within the values for, e.g., `infl` and `agr`. Other features are also assigned values, though these are not considered agreement features, e.g. `case` and `def_enclitic`[2].

Note that inflectional type information is only relevant for ambiguous suffixes, viz. the null and *-s* suffixes, whence the first 10 specifications within the disjunction. For example, inflectional type `infl4` has a null suffix for both the singular and the plural indefinite forms[3]. Likewise the singular and plural genitive indefinite forms are identical, both taking *-s*. Each of these special combinations must be accounted for separately.

As can be seen, the ambiguity of inflectional type `infl4` is expressed by the fact that there are additional entries for the plural interpretations of `(infl4&null)` and `(infl4&s)`.

Lifting rules for verbs and adjectives are similar to this, with named disjunctions defined for interpretation of inflectional type/suffix combinations. Modal verbs have a lifting rule separate from other verbs due to their particular morphotactic behavior (see Section 5.2).

Minor stems

With minor stems, suffixation is simple and unambiguous, resulting in simpler morphotactic rules. Since there are no inflectional types, interpretation is based solely on the suffix. As an example, here is the lifting rule for adverbs:

```
ts_ls_rule(
  sign:{
    procinfo=>tprocinfo:{
            specinfo=>tspecinfo:{tlm=>n,partition=>tpart:{main=>adv,sub=>LU}},
            ruleinfo=>truleinfo:{lift_id=>'Mlift-adverb'}},
    ortho=>tortho:{
             string=>[LEMMA|[CONT|REST]],rest=>REST},
    synsem=>tsynsem:{
              syn=>tsyn:{
                     str=>CAT=>tadv:{
                                  constr=>lexical,
                                  lu=>LU,
                                  lemma=>LEMMA,
                                  contin=>CONT,
                                  infl=>( null/re/st/ste ) - altx },
                     cat=>tsubst_cat:{
                            cohead=>tcohead:{adv=>tadv_cohead:{head=>COHEAD}},
                            head=>COHEAD=>tsubst_head:{
                                    major=>tadv_major:{
                                         pos=>adv,
```

[2] Although the definite enclitic in Danish necessarily implies that the noun is definite, the converse is not true, thus the distinction here between definiteness and the presence of the enclitic.

[3] Examples of inflectional type `infl4`: *film* (movie(s)/film(s)), *tog* (train(s)), *afsnit* (section(s)/paragraph(s)), *behov* (need(s)).

```
                                        selects=>tmodifiee:{},
                                        advform=>
                                        ( base/compar/superl/superl ) - altx,
                                        agr=>
                                        (_/_/indef&sing/
                                         (~(indef&sing))) - altx }}
                                    }}}},
  'M',['CAT'=>CAT,'STATUS'=>'OK'],_STR).
```

As can be seen, some adverbs allow a comparative form, and definite and indefinite superlative forms. Note that adverbs can only modify, they cannot specify. This is expressed by setting `selects=>tmodifiee:{}` for all adverbs via the lifting rule.

Words

Here is the rule for prepositions.

```
ts_ls_rule(
  sign:{
    procinfo=>tprocinfo:{
            specinfo=>tspecinfo:{tlm=>n,partition=>tpart:{main=>p,sub=>LU}},
            ruleinfo=>truleinfo:{lift_id=>'Mlift-tp'}},
    ortho=>tortho:{
             string=>[LEMMA|REST],rest=>REST},
    synsem=>tsynsem:{
              syn=>tsyn:{
                     str=>CAT=>tp:{
                                 constr=>lexical,
                                 lu=>LU,
                                 lemma=>LEMMA},
                     cat=>tsubst_cat:{
                            cohead=>tcohead:{p=>tp_cohead:{head=>COHEAD}},
                            head=>COHEAD=>tsubst_head:{
                                    marking=>unmarked,
                                    major=>tp_major:{
                                        pos=>p,
                                        selects=>tmodifiee:{
                                            newmarking=>MARKING,
                                            synsem=>tsynsem:{
                                              syn=>tsyn:{
                                                cat=>tsubst_cat:{
                                                  head=>tsubst_head:{
                                                    marking=>MARKING}}}}}
                                        }}}}},
  'M',['CAT'=>CAT,'STATUS'=>'OK'],_STR).
```

Although words (according to their definition here) are uninflected, there can still be a distinction between the values for `lemma` and `lu`, for instance where a collocation can be interpreted as a compound preposition which can potentially be abbreviated. An example from Danish is *ved hjælp af, 'vha.'* ('with the help of').

6.3 Stem changes

As vowel shift and consonant change have been lexicalized, the only morphographemic changes left to implement via TLM rules are syncope, gemination and degemination. These get one TLM rule each.

```
% Syncope
% Examples:  disponible<=>disponibel+e, gamle<=>gammel+e

tlm_rule(
  syncope,
  [C1] [] [C2,e] <=> [C1] [e] [C2,+],
  _,
  tstem_major:{
    infl=>(~null),
    grf=>tgrf:{syncope=>y,fuge_used=>n},
    seq=>last},
  [C1 in consonants, C2 in {l,n,r}]
       ).
```

This rule expresses that syncope (*e*-deletion) can only occur interconsonantally, where the following consonant is one of *l,n,r* and is followed by a suffix beginning with the letter *e*. The word must be an inflected major stem. The *e* given in the right-hand context part of the syncope rule is necessarily part of a suffix (as opposed to being the beginning of a following element forming a compound) due to the constraint `seq=>last`.

Note in this rule and the rule for gemination below that the features `syncope` and `gemination` are set to `y` on application of the rule. Lexical items allowing syncope and/or gemination leave the respective features uninstantiated, allowing unification with the feature structures of these rules. Note also that the application of syncope or gemination has no significance for morphotactics, so that once these features have been used to constrain/allow the stem change during TLM analysis, they no longer are used.

```
% Degemination
% Examples:  kartofler<=>kartoffel+er, gamle<=>gammel+e
%
% Syncope must also occur, so there must be (almost)
% the same context for C2 as in the rule for syncope.

tlm_rule(
  degemination,
  [C] [] [C2] <=> [C] [C] [e,C2,+],
  _,
  tstem_major:{
    infl=>(~null),
    grf=>tgrf:{syncope=>y,fuge_used=>'n'},
    seq=>last},
  [C in {f,m}, C2 in {l,n,r}]
       ).
```

Degemination (deletion of a doubled consonant) can occur to a repeated *m* or *f* preceding an *e* to which syncope has applied.

```
% Gemination
% Examples:
% klubben<=>klub+en, stikket<=>stik+et, s{\o}nner<=>s{\o}n+er
% flotte<=>flot+e,
% kommer<=>kom+er
% skattetryk<=>skat+tryk

tlm_rule(
  gemination,
  [V,C] [C] [e] <=> [V,C] [] [+],
  _,
  tstem_major:{
    grf=>tgrf:{gemination=>y}},
  [V in vowels, C in {b,d,f,g,k,l,m,n,p,r,s,t}]
    ).
```

Gemination (consonant doubling) can occur immediately before a suffix beginning with *e*. Only the consonants *b,d,f,g,k,l,m,n,p,r,s,t* may be doubled.

Since gemination occurs both in compounded and non-compounded forms of lexemes allowing the change, the TLM rule for gemination has been made more general (i.e. no constraint on the value of `seq`) compared to those for syncope or degemination, which only occur word-finally.

6.4 Compounding

Coverage

Compounding of nouns and adjectives has been implemented. Other compound types are either infrequent or difficult to process, or both.

Another restriction is based on the fact that adjectives occur infrequently as the middle element of a compound consisting of more than two elements. Thus only nouns are allowed as middle elements.

The implementation still covers the greatest share of compounds occurring within the corpus analyzed by Paggio & Oersnes (1991), which is also consistent with occurrences in the corpus used for LSGRAM. However one obviously weak point is the combination `noun+pastparticiple`, where the past participle is used adjectivally. This is infrequent in our corpus, but must be considered for implementation of a more general description of Danish.

Initial compound elements

By far the single most resource-demanding TLM rule is the one inserting the null fuge element. Since the null fuge can follow either an adjective or a noun, a special type has been defined called `tstem_compoundable` which comprises both nouns and adjectives and is used as a constraint within the TLM rule, in this way avoiding having to specify two separate null-fuge rules.

```
tlm_rule(
  compound_null_fuge,
  [X,Y] [] [Z1,Z2] => [X,Y] [+] [Z1,Z2],
  _,
  tstem_compoundable:{
```

```
    infl=>null,
    grf=>tgrf:{fuge_used=>y,fuge=>null},
    contin=>'-',
    seq=>first&(~last)},
  [X in alphabet, Y in alphabet, Z1 in alphabet, Z2 in alphabet]
       ).
```

It is not strictly necessary to have two letters of context on each side, but they are specified here in an attempt to make the rule more efficient. (There are no nouns or adjectives consisting of a single character.)

The remaining rules for initial elements are restricted to nouns only, and are used for processing the fuge elements *e,s,'-'*. `compound_s_fuge` is given here as an example:

```
tlm_rule(
  compound_s_fuge,
  [X,Y] [s] [Z] => [X,Y] [+] [Z],
  _,
  tn:{
    infl=>null,
    grf=>tgrf:{fuge_used=>y,fuge=>s},
    contin=>'-s-',
    seq=>first&(~last)},
  [X in alphabet, Y in alphabet, Z in alphabet]
       ).
```

Middle compound elements

Each noun in Danish requires a certain following fuge element (either null, *e,s*, or hyphen *'-'*) when used as the non-final element of a compound. However, if the noun is the middle element of a compound (non-initial and non-final), either a 'gratuitous' fuge *-s-* or null fuge is used, termed within the rules as `fuge=>mid-s` and `fuge=>mid-null`, respectively.

The gratuitous null fuge occurs after *s, er* or *or*, otherwise the fuge *-s-* must occur. Specifying the rules for the null fuge is easily done:

```
tlm_rule(
  compound_middle_noun_null_fuge_1,
  [X,r] [] [Y] => [X,r] [+] [Y],
  _,
  tn:{
    infl=>null,
    grf=>tgrf:{fuge_used=>y,fuge=>'mid-null'},
    contin=>'-',
    seq=>(~first)&(~last)},
  [X in {o,e}, Y in alphabet]
       ).

tlm_rule(
  compound_middle_noun_null_fuge_2,
  [s] [] [Y] => [s] [+] [Y],
  _,
  tn:{
```

```
    infl=>null,
    grf=>tgrf:{fuge_used=>y,fuge=>'mid-null'},
    contin=>'-',
    seq=>(~first)&(~last)},
  [Y in alphabet]
        ).
```

The gratuitous fuge *-s-* also requires two rules, since it is not possible to express the fact that the two preceding characters in combination may *not* be *er* or *or*. The constraint can be expressed by a single rule for the context following a non-*r*, while another rule expresses that the post-*r* context is acceptable if what precedes the *r* is neither of *o,e*.

```
tlm_rule(
  compound_middle_noun_s_fuge_1,
  [X] [s] [Y] => [X] [+] [Y],
  _,
  tn:{
    infl=>null,
    grf=>tgrf:{fuge_used=>y,fuge=>'mid-s'},
    contin=>'-s-',
    seq=>(~first)&(~last)},
  [X in consonants, X not in {r,s}, Y in alphabet]
        ).

tlm_rule(
  compound_middle_noun_s_fuge_2,
  [X,r] [s] [Y] => [X,r] [+] [Y],
  _,
  tn:{
    infl=>null,
    grf=>tgrf:{fuge_used=>y,fuge=>'mid-s'},
    contin=>'-s-',
    seq=>(~first)&(~last)},
  [X not in {o,e}, Y in alphabet]
        ).
```

There are other exceptional compounds where the *-s-* does not occur (a null fuge is used instead), but which are not systematic and must be lexically coded in this implementation.

One final rule is necessary for middle elements in order to process the letter immediately following the morpheme boundary. Remember that the default rules do not process a character following an inserted morpheme boundary (see Section 6.2.1). Note that this is restricted to applying to `tn`'s, effectively limiting middle elements to nouns.

```
tlm_rule(
  compound_middle_noun,
  [X] [Y] [] => [+] [Y] [],
  _,
  tn:{
    seq=>(~first)&(~last)},
  [X in alphabet, Y in alphabet]
        ).
```

Final compound elements

Given that the type `tstem_compoundable` is defined comprising adjectives and nouns, the final element of a compound is handled with the following rule:

```
tlm_rule(
  compound_final,
  [X] [Y] [] => [+] [Y] [],
  _,
  tstem_compoundable:{
    seq=>(~first)&last},
  [X in alphabet, Y in alphabet]
        ).
```

Chapter 7

Structural Analysis

Analysis is a term used within ALEP to cover both word formation and syntactic analysis, these occurring as part of the same processing phase.

This chapter begins with a description of word structure analysis in the implementation. Phrase structure analysis is then described in two separate sections, the first focussing on how various word classes are analyzed, the second on the phrase structure rules themselves and the schemata they implement.

7.1 Word structure schemata and rules

The output from lifting is a simple hierarchical structure where a type `tphrasal` has as daughters a sequence of morphemes of type `tmorphol`. In this context then, a word can be defined as a minimal projection of type `tphrasal`.

Word structure rules are used to specify general relationships between morphological information and information necessary for syntactic processing, for giving structure to compounds, and for implementing the equivalent of lexical rules for manipulating the subcategorization information of passive forms.

7.1.1 Word Schema

The Word Schema has been created to generalize over word structure rules within the implementation. The schema itself is fairly vacuous, since the word structure rules vary markedly depending on whether they implement the equivalent of a lexical rule or simply convert the morphemic structure to a phrasal structure.

Word Schema

$$\left[\textsc{sign} \left[\textsc{synsem} \left[\textsc{tphrasal} \left[\textsc{constr} \quad \text{word}\right]\right]\right]\right]$$

$<$

$$\left[\textsc{sign} \left[\textsc{synsem} \left[\textsc{tmorphol} \left[\textsc{constr} \quad \left(\textsc{lexical;compound}\right)\right]\right]\right]\right]$$

According to this schema, all word structure rules are unary rules where the mother is of type `tphrasal` and construction type *word*, while the daughter is of type `tmorphol` and construction type *lexical* or *compound*.

Danish has an unambiguous `s`-passive verb form (i.e. having the suffix `-s`) and two auxiliary passive forms (*være* and *blive* passives). Because auxiliaries are treated as substantives (see section 7.2.2), they are treated as active main verbs.

The passive voice in *være* and *blive* passives is considered inherent to the past participles the auxiliaries combine with. Thus in the present implementation there is a word structure rule changing the subcategorization information of `s`-passives and a word structure rule changing the subcategorization information of past participles for verbs which can be passivized with the auxiliaries *være* and *blive*. This treatment of passives is similar to the one suggested in Pollard and Sag (1987, pp. 215–218) and Pollard and Sag (1994, pp. 153–155).

The two rules for `s`-passive and for passive past-participles are as follows:

```
sign:{
  procinfo=>mWSPECana['phr2morph,Passive fininfin verb'],

  ortho=>ORTHO,
  synsem=>tsynsem:{
            syn=>tsyn:{
                   str=>tphrasal:{constr=>word,lu=>LU},
                   cat=>tsubst_cat:{
                          head=>HEAD,
                          cohead=>tcohead:{ v=>tv_cohead:{head=>HEAD} },
                          subj=>[COMPL],
                          compls=>REST,
                          by_ag=>SUBJ}}
                       % Note:  must preserve all subject synsem info in this way,
                       % which otherwise is lost if make the by_ag value a p
                       % with an empty compls list.
                       % Requires that verb+by_agent rule finds its own "af".
            }}
  <
  [ sign:{
      ortho=>ORTHO,
      synsem=> tsynsem:{
        syn=>tsyn:{
               str=>tv:{constr=>(lexical;compound),seq=>first&last,lu=>LU},
               cat =>tsubst_cat:{
                       head=>HEAD=>tsubst_head:{
                             major=>tv_fininfin:{
                                      pos=>v,
                                      voice=>pass}},
                       subj=>SUBJ,
                       compls=>[COMPL|REST]}}}}].

sign:{
  procinfo=>mWSPECana['phr2morph,Passiv Past Participial verb'],

  ortho=>ORTHO,
        synsem=>tsynsem:{
            syn=>tsyn:{
```

```
                    str=>tphrasal:{constr=>word,lu=>LU},
                    cat=>tsubst_cat:{
                           head=>HEAD=>tsubst_head:{
                              major=>tv_participial:{
                                       pos=>POS,
                                       voice=>pass,
                                       partform=>PARTFORM,
                                       selects=>SELECTS,
                                       perf_aux=>vaere}},
                              cohead=>tcohead:{ v=>tv_cohead:{head=>HEAD} },
                              subj=>[COMPL],
                              compls=>REST,
                              by_ag=>SUBJ}}
                         }}
<
[ sign:{
    ortho=>ORTHO,
    synsem=> tsynsem:{
      syn=>tsyn:{
             str=>tv:{constr=>(lexical;compound),seq=>first&last,lu=>LU},
             cat =>tsubst_cat:{
                     head=>tsubst_head:{
                           major=>tv_participial:{
                                    pos=>POS=>v,
                                    voice=>act,
                                    partform=>PARTFORM=>past,
                                    selects=>SELECTS,
                                    perf_aux=>have}},
                     subj=>SUBJ,
                     compls=>[COMPL|REST]}}}}].
```

The daughters in these rules are `tv`, a subtype of `tmorphol`.

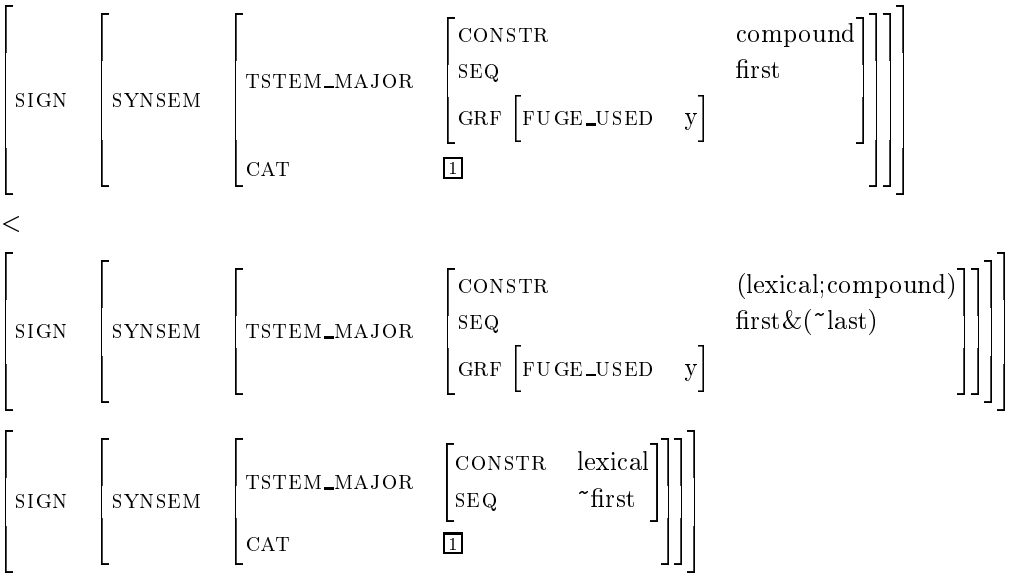
The effect of application of these rules is to structure-share the deep subject with the by-agent feature `by_ag` of the phrase, and to structure-share the first complement with the surface `subj` value, constrained to applying to passive verb forms via a special phrase structure rule.

Note that the `by_ag` feature must be assigned the entire deep subject `tsynsem` value, instead (as might be supposed) a *by*-preposition projection with an empty `compls` list. This is necessary in order to have access to this `tsynsem` information from the rule parsing the by-agent. With the alternate approach, this information is inaccessible from the by-agent phrase structure rule, since all PS rules have access to only two levels, i.e. mother-daughter, but not grandmother-granddaughter.

There are three other rules implementing the Word Schema, these accounting for all cases other than `s`-passive and passive past participle, i.e. active verbs, all non-verbal substantive forms and functional forms. They will not be given here, as they are very simple, structure-sharing the entire `tcat` value between mother and daughter.

7.1.2 Compound Schema

Compound Schema



Given that compounding was not part of the original work plan, it was decided to treat compounds as instances of the final compound element, so that almost all information within the mother node comes from the non-initial daughter.

This is implemented with a single PS rule.

```
sign:{
  procinfo=>mWSPECana['morph2morphmorph'],

  ortho=>tortho:{ string=>STRING,rest=>STRING_REST },
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tstem_major:{
                  constr=>compound,
                  seq=>first,
                  grf=>tgrf:{fuge_used=>y},
                  lu=>LU
                    },
           cat=>CAT}}}
  <
  [
    sign:{
      ortho=>tortho:{ string=>STRING,rest=>STRINGa },
      synsem=>tsynsem:{
        syn=>tsyn:{
               str=>tstem_major:{
                      constr=>(lexical;compound),
                      seq=>first&(~last),
                      grf=>tgrf:{fuge_used=>y}},
               cat=>_CAT}}},
    sign:{
      ortho=>tortho:{ string=>STRINGa,rest=>STRING_REST },
      synsem=>tsynsem:{
        syn=>tsyn:{
               str=>tstem_major:{
```

```
            constr=>lexical,
            seq=>(~first),
            lu=>LU},
    cat=>CAT=>tsubst_cat:{
        head=>tsubst_head:{
            major=>tsubst_major:{}}}}}} ].
```

Via this rule, all compounds are assigned a left-branching structural analysis with all `cat` information being structure-shared from the right-most daughter morpheme. As mentioned, currently syntactic and semantic information from non-final compound elements is ignored.

7.2 Phrase structure analysis and word classes

7.2.1 General comments

The following general decisions regarding syntactic analysis (Navarretta 1996) have been taken:

- On the whole, the Lean Approach (Schmidt et al. (1996)) is followed, given that the ALEP formalism is itself lean.
- In most cases, binary rules are implemented. This approach has been proven to be efficient (see the documentation of the German implementation in Schmidt et al. (1995)) and flexible, allowing a greater possibility for cyclic rule application.
- The HPSG SUBCAT list has been split into a `subj` list and a `compls` list following Pollard and Sag (1994, Chapter 9).
- Saturation constraints are used instead of the two features `max` og `min` described in the Danish MLAP (LINDA) report on Phrase Structure.
- The feature `heading` is only used when necessary. The feature `headed` is not used.
- The attribute `constr` is implemented indicating the construction type (schema) of every node, morphological and phrasal.
- The distinction between functional vs. substantive parts of speech is only maintained where it is most efficient implementationally.

We depart somewhat from the distinction between substantive and functional categories made in Povlsen et al. (1995b) in that the only functional classes in our implementation are markers and the expletive *der*, while punctuation marks are a class of their own. It is important to stress that our choice is mainly made on the basis of implementational considerations to avoid double declarations and ambiguities for word classes belonging to both categories, such as verbs: auxiliaries vs. main verbs, quantifiers: central quantifiers vs. pre- and post-quantifiers, prepositions: valency-bound vs. non valency-bound prepositions (see the sections for each word class).

This treatment of i.a. auxiliaries and valency-bound prepositions is made possible by the functionality of ALEP, wherein different levels of analysis (i.e. analysis and refinement) can be defined, so that some distinctions necessary within a single-level approach such as described by HPSG can be 'put off' until the more efficient refinement phase. In fact, one could argue that this approach is more theoretically sound, since it makes a clearer distinction between behavior at the phrase structure level and at the refinement level. In this approach, syntactically a PP is a PP, whether valency-bound or not, while semantically the preposition of a valency-bound PP is vacuous, differing from predicatively used prepositions.

Consistent with Pollard and Sag (1994) the nominal sign comprises nouns and pronouns.

Different word classes which can act as clausal adjuncts (in the present implementation, these are adverbs, prepositional phrases and nominal phrases) have a head feature `posit` whose value *front*, *nexus* and/or *end*, determines the possible positions in which each adjunct can occur in a clause (see Section 7.3).

The HPSG head features SPEC and MOD are implemented by the single head attribute `selects` taking as a value one of the types `tspecifiee`, `tmodifiee`, `tmarkee` and `tnothing` for specifiers, modifiers, markers and non-selecting nodes, respectively. The feature is used not only to indicate *what* can be selected by a given node, if anything, but also *whether* anything can be selected by the node. Thus nodes with `selects=>tspecifiee`, `selects=>tmodifiee` and `selects=>tmarkee` are potential specifiers, modifiers and markers, while a node with `selects=>tnothing` may neither specify, modify, nor function as a marker (see the description of the Selection Principle in Section 1.4.2; see also Section 7.2.3 below).

In HPSG the SPEC feature is a HEAD feature for functionals, both markers and specifiers. In Povlsen et al. (1995b) it is a category feature. In the LSGRAM approach there is a distinction between marking and specifying, so that SPEC corresponds to the two MAJOR features `selects=>tspecifiee` and `selects=>tmarkee`. The former is used by substantive word classes (genitive phrases, pronouns, articles, central quantifiers) and the latter by markers and punctuation.

The `spr` feature, which in HPSG was a list, is implemented as a boolean feature taking the two values *sat* and *unsat*. In HPSG `spr` is a `cat` feature, while in the present implementation it is a substantive `head` feature.

The HPSG MARKING head feature is implemented by the two head features `marking` and `newmarking`: `marking` is used when selecting, `newmarking` when projecting a marking value. The two features are not only used for markers as in HPSG, but also for determiners, adjuncts and punctuation marks (see the Marking Principle, Section 1.4.2).

In Povlsen et al. (1995b) the features `heading` and `headed` were used in all phrasal rules, with `heading` indicating whether the head-daughter can function as right or left head, and `headed` indicating whether the head-daughter is realized to the left or right. Here, only the `heading` feature is used, controlling the attachment of constructions in the two cases where this is necessary, i.e. to control the attachment of pre- and postmodifying elements to a nominal phrase, and of pre- and postmodifying adjuncts to a clause. Pre-modifying elements are attached to nominals after post-modifying elements. Pre-modifying clausal adjuncts are attached to clauses before post-modifying adjuncts.

In implementing passive constructions we have used a categorial list `by_ag`, for the agent (Theofilidis and Reuther 1995).

The degree of saturation is determined by the saturation of the `subj` and `compls` lists, and of the boolean feature `spr`.

To avoid lexical disjunctions in the analysis lexicon due to differences in subcategorization having to do with complement types, co-representation of heads has been implemented using the attribute `cohead` (Section 2.1), following Theofilidis and Reuther (1995) (see also the Cohead Feature Principle, Section 1.4.2).

7.2.2 Verbs

Verbs include main verbs, auxiliaries and modals. Finite verbs have the head feature `nex` taking one of the three values *nva, nav, vna* (in the LINDA report only *nav* and *vna* were used for distinguishing between main and subordinate clauses). Here, the `nex` feature is used to indicate

in which kind of sentence the finite verb is used, viz. a main clause with the subject on the first position, a main clause with the subject after the finite verb or a subordinate clause.

The `nex` value *nva* is used in main clauses with a normal SVO construction, including imperative clauses which do not take a subject. The `nex` value *vna* is used in main clauses where the first element is not the subject (VSO constructions). The value *nav* is used in subordinate clauses where any actualization adverbs must occur between the subject and the finite verb, and not after the subject and the finite verb (independently of their reciprocal order) as in all types of main clauses.

Treatment of auxiliaries and modals

Modals and auxiliaries are treated similarly. They are not functors, but are substantive categories taking a complement, though their `subj` is inherited from the verbal complement.

This avoids the problem of having to lift the verbs *blive, ville, være* and *have* into separate structures for the substantive and functional readings, one of which must then be eliminated during syntactic analysis. Instead, each of these verbs is lifted into a single underspecified structure with a correspondingly underspecified lexical entry applying. In this way, the ambiguity is put off until much later in the analysis process, having a positive effect on runtime (generally speaking, ambiguities should be put off as long as possible for optimal runtimes).

Another implementational advantage of treating auxiliaries as heads is generalizability, as it then becomes possible to write a grammar rule applying to all finite verbs, whether auxiliary, modal or main verbs. This makes inversion easier to handle: inversion in Danish for example is handled easily by repeating the rule combining a verb with its subject, where the subject occurs to the left or right of the finite verb, respectively. Given a substantive-functional distinction between different verbs, the rule would have to be doubled again, making four rules and losing the generalizability of inversion expressed within a single rule.

In addition, head information within the past participial main verb is no longer interesting once it has combined with an auxiliary. For example, the perfect auxiliary value is changed, since the auxiliaries themselves may take different auxiliaries than whatever main verb they are combined with. Nor is verb-form information interesting at that point, since there is no reason to preserve the information that the main verb was in participial/infinitival form once it has been combined with an auxiliary.

7.2.3 Nouns

Singular count nouns are unsaturated and are subcategorized for by the following specifiers: articles, possessive pronouns, genitive phrases and central quantifiers. Unsaturated nouns (those missing a specifier) are indicated via the feature `spr`, where `spr=>unsat` indicates an unsaturated noun, while `spr=>sat` indicates a saturated noun.

Specifiers structure-share their `selects=>tspecifiee|synsem` feature with the `synsem`-value of the head daughter. The value `tspecifiee` for the attribute `selects` is a necessary and sufficient indication that the node in question is a specifier.

Nouns can be pre- and/or post-modified.

Nouns can be premodified by adjectives, quantifiers and participles (they are all implemented). They can be postmodified by complements, adjuncts, participles, and relative sentences (the latter two are not implemented).

Modifiers structure-share their `selects=>tmodifiee|synsem` feature with the `synsem`-value of the head daughter. The value `tmodifiee` for the attribute `selects` is a necessary and sufficient indication that the node in question is a modifier.

Allowable combinations of determiners, viz. articles, possessives, genitive phrases and all quan-

tifiers, are controlled by the category features `marking` and `newmarking` which are also used for markers and adverbs. The relevant `marking` values for nominals are the following: *prequant, cquant, postquant, card, ord, defin, unmarked.*

Postmodifying elements are attached to nouns before premodifying elements, which are attached before specifiers. To control the order of attachment, the feature `heading` (taking the possible values *left* and *right*) is used together with constraints on the degree of saturation.

Because nouns expressing temporal expressions, such as *dag*, *tid* (day, time), can act as adjuncts modifying clauses, they have the adverbial head feature `posit` declared[1].

Nouns used as adjuncts or as predications after copula constructions are marked as being used in a predicatively way, i.e. the `prd` feature has value *yes*. In all the remaining cases the feature `prd` for nouns is set to *no* marking the attributive use.

7.2.4 Pronouns

Pronouns can be of the following types:

```
type => boolean([{pers,poss,interr,rel,refl,demo,recipr,quant,expl,art}])
```

In the LINDA reports possessive pronouns and central quantifiers are treated as articles, thus as functionals. On the other hand pre- and post- quantifiers and genitive phrases[2], including personal pronouns (e.g. *hans* (his) is the genitive form of the personal pronoun *han* (he)) are substantives.

Maintaining the distinction functionals/substantives for articles and central quantifiers would complicate unnecessarily the implementation, requiring the multiplication of the lexicon entries and of the phrasal rules for words belonging to the same class, or belonging to different classes but having the same function in the syntax. As it was the case for auxiliaries, ambiguities should be put off as long as possible. In the case of possessive pronouns, personal pronouns in genitive form and nominal genitive phrases, the distinction between functionals and substantives also appears odd from a theoretical point of view. Therefore we treat all the above classes as substantives.

In the present implementation we have considered pronouns to be a large class comprising the expletive *det* and all articles (types *expletive* and *article*, respectively). The choice has not a theoretical but an implementational justification in that we avoid ambiguities in lifting and in analysis lexicon among the pronouns *den/det, en/et*, the expletive *det* and the articles *den/det, en/et*. In refinement these types have distinct entries.

7.2.5 Adjectives

Adjectives as a class can occur in both attributive and predicative uses, though some adjectives may only occur in one or the other usage. This is indicated within the entries of the analysis lexicon, where the `prd` value may be unspecified, *no* or *yes*.

Attributive adjectives structure-share their `selects=>tmodifiee|synsem` feature with the `synsem` value of the nominal head daughter.

Adjectives occurring predicatively have a slightly different behavior with regard to agreement than when they occur attributively. In short, definiteness is a significant agreement feature attributively but not predicatively. Thus *godt* (good) used attributively indicates the modifiee is singular, neuter and indefinite, while if used predicatively, it only indicates that the subject is singular and neuter, saying nothing about definiteness. The following table makes this behavior clear.

[1] Genitive nominals have a `selects=>tspecifiee` feature specifying a nominal phrase, non genitive nominals have a `selects=>tmodifiee` feature modifying a verbal phrase. The `selects` feature for nominals is set in lifting.

[2] Nominal genitive phrases as specifiers were outside the scope of the LINDA reports.

"good"	attr. agreement	pred. agreement
god	sing&comm&indef	sing&comm
godt	sing&neut&indef	sing&neut
gode	plur ; def	plur

The agreement information for attributive usage is indicated within the adjective's `selects=>tmodifiee|synsem` value, while agreement for predicative usage is indicated within the adjective's `subj` value. All agreement features are set in lifting, with the exception of the agreement feature for adjectives taking a clause as subject, which is set in the analysis lexicon.

7.2.6 Prepositions

Valency-bound prepositions, weakly bound prepositions, predicative prepositions and prepositions occurring as heads of predicative phrases are not distinguished in the analysis lexicon, to avoid such lexical ambiguity during syntactic analysis. In this we depart from Pedersen et al. (1995) which proposes to code a separate lexical entry for each use of a preposition.

Instead when used for case marking (i.e. strongly bound prepositions), the preposition is marked as being used in an attributive way (`prd` feature has value *no*). When used as heads for adjuncts they are marked as predicative (`prd` feature has value *yes*). In this we follow Pollard and Sag (1987 and 1994). Prepositions are also marked as predicative when occurring after the copula verbs such as *være*, *blive*, in which case the subject of the prepositional phrase is coindexed with the subject of the copula verb.

Because prepositions can be heads for adjuncts modifying clauses they have the adverbial head feature `posit`.

Prepositions have also the two head features `p_lu` and `p_compl`. The former is a boolean and indicates the actual preposition. It is used when argument-taking classes must subcategorize for a particular valency-bound preposition (it is a boolean value because the same word can subcategorize for different prepositions). The `p_compl` feature has a value of the same type as the head feature structure and is used for constraining the class and type of the complements following the preposition and for percolating information about these complements (see Section 7.3 wrt the Head-Complement Schema for prepositions). In this way it is possible to distinguish between prepositional phrases with nominal complements, adverbial complements, and finite and non-finite verbal complements at higher levels within the analysis structure. This information is otherwise inaccessible to once the prepositional phrase is formed. It is necessary because non-finite verbal complements, are control constructions which must be specially treated in refinement, e.g.

Han var træt af at læse.
(He was tired of reading.)

7.2.7 Adverbs

Adverbs in Danish can modify other adverbs, adjectives, verbs, clauses, i.a. However in the present implementation we have only treated clausal adverbs. Adverbs modifying clauses have as other adverbials the head feature `posit` indicating the position they occupy in a clause (see Section 7.3 wrt Head-Adjunct Schema). Adverbs occurring in the Actualization field have a precise reciprocal order, e.g.

Jeg kan jo altså faktisk sjældent komme tidligere.
(I can therefore really seldom come before.)

is a correct clause while the following two are not:

** Jeg kan sjældent jo faktisk altså komme tidligere.*
** Jeg kan jo altså sjældent faktisk komme tidligere.*

To control the order of the adverbials in the Actualization field, the `marking` and `newmarking` head features have been used. Their relevant values for adverbs are *adv1, adv2, adv3, adv4* where adverbs of type *adv1* come first, adverbs of type *adv2* must follow them and so on.

Note that *ikke* (not) is coded as `marking=>adv4`, although it can occur in a different position from the usual *adv4* adverb. However when it does so it is not functioning as a clausal adverb, but is modifying the following adverbial (beyond the scope of the present implementation).

7.2.8 Quantifiers

All quantifiers are substantives, departing from the LINDA report on specification. Pre- and postquantifiers are treated as attributive adjectives and therefore structure-share their `selects=>tmodifiee|synsem` feature with the `synsem` value of the head daughter. Central quantifiers are treated as articles, therefore structure-sharing their `selects=>tspecifiee|synsem` feature with the `synsem` value of the head daughter. This is consistent with the Danish MLAP report on determination (Neville 1995).

Quantifiers are determiners and have the two head features `marking` and `newmarking` which also control the order of determiners.

7.2.9 Functionals and punctuation marks

As explained in Section 7.2.4, the expletive *der* is treated as a special functional category, while the expletive *det* is considered a type of pronoun. In refinement the expletive *det* is recognized on the basis of the constructions it occurs in.

The expletive *der* can introduce active and passive constructions. In active constructions it can be followed by the verb *være*, by unaccusatives and by motion verbs. The following NP (the real subject of the construction) must be indefinite and must not be preceded by pre-quantifiers or strong central quantifiers[3].

Examples of *der*-constructions are the following:

Der er mange mennesker på gaden.
(There are many people in the street.)

Der kommer en mand med en sjov hat.
(A man with a funny hat is coming.)
(lit. there comes a man with a funny hat.)

Der lå ikke noget i det han sagde.
(There was no deeper meaning in what he said.)
(lit. there lay not something in that he said.)

The expletive *der* can also be followed by all passive verb forms. In this case *der* introduces impersonal constructions as in the following examples:

[3] Strong central quantifiers (*enhver* and *hver*) should not be allowed in these constructions. Presently we have not implemented this constraint.

Der arbejdes meget her.
(One works hard here.)
(lit. There is-worked much here.)

Der må ikke nydes drikkevarer i bussen.
(It is forbidden to consume beverages in the bus.)
(lit. There must not be-enjoyed beverages in the-bus.)

In the present implementation we have only treated the most common *der*-constructions, i.e. those followed by s-passives (only when there is a formal subject) and intransitive verbs.

The expletive *det* introduces zero-valent verbs and raising constructions (verbals and adjectivals). Both constructions have been implemented.

Det regner.
(It is raining.)
(lit. It rains.)

Det er godt for ham at holde fri.
(It is good for him to take a day off.)
(lit. It is good for him to hold free.)

There is currently only one marker in the Danish implementation, "at", marking both infinitive verbs (corresponding to the English infinitive marker "to") and subordinate clauses (complementizer, corresponding to the English complementizer "that"). Thus we consider the infinitive marker a complementizer[4].

Currently punctuation is treated as a marker, although this may change in future versions.

7.3 Phrase structure schemata and rules

In implementing phrasal projections we have used binary rules with the exception of unary rules for word structure (including word structure rules for passivization), a unary rule for optional complement extraction, a unary rule for subjectless imperative constructions and a triary rule for the attachment of the by-agent to passive constructions.

The feature `str|constr` has been used to indicate the type of construction (schema) implemented by each rule. At the phrasal level, these constructions all correspond to HPSG schemata (see Section 1.4.3 and Section 2.1).

```
constr => boolean([{lexical,compound,word,
                    h_subj,h_compl,h_subj_compl,
                    h_spec,h_adj,h_mark,h_mark_punct,h_fill}])
```

The value `h_mark_punct` indicates a special type of Head-Marker construction used for punctuation. The Head-Filler Schema (`h_fill` construction type) has not been implemented.

In the sections below, the X-bar versions of HPSG schemata are given where possible. The reader is reminded that these representations use the conventions that X" assumes specifier-saturation (SPR <>), X' assumes the X is still seeking a specifier (SPR <Y">), and X^0 is a lexical item (word) (Polland & Sag (1994), p. 362).

[4] In Pollard and Sag (1994) is discussed the possibility of treating the infinitive marker in English as a complementizer or as a defective auxiliary (pp. 125–27).

7.3.1 Head-Complement Schema

The Head-Complement Schema (Pollard & Sag 1994, p. 362) is the following:

$$\text{XP} \rightarrow \boxed{1},\ \text{X}^0\left[\text{COMPS}\ \left\langle \boxed{1} \right\rangle\right]$$

with the first element on the right side of the formula being the `compls`, the second being the head.

This assumes that in the Head-Complement Schema head information, the `subj` list and `sem` are structure-shared between the head daughter and the mother. We implement the same schema, but with binary rules, thus the first element of the `compls` list of the head daughter is structure-shared with the `synsem` value of the complement daughter.

Although the Head-Complement Schema is stated generally as a binary structure it also comprises unary rules treating optional complement extraction and a triary rule for by-agent attachment. In addition, semantics is handled in refinement rules and lexical entries (Chapter 8). Here is our version:

Head-Complement Schema

$$\left[\begin{array}{l}\text{SIGN}\ \left[\begin{array}{l}\text{SYNSEM}\ \left[\begin{array}{l}\text{STR}\ \left[\text{CONSTR}\ \ \text{h\_compl}\right] \\ \text{CAT}\ \left[\begin{array}{ll}\text{HEAD} & \boxed{1} \\ \text{COHEAD} & \boxed{1} \\ \text{SUBJ} & \boxed{2} \\ \text{COMPLS} & \boxed{4}\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right]$$

$<$

$$\left[\begin{array}{l}\text{SIGN}\ \left[\begin{array}{l}\text{SYNSEM}\ \left[\begin{array}{l}\text{CAT}\ \left[\begin{array}{ll}\text{HEAD} & \boxed{1} \\ \text{SUBJ} & \boxed{2} \\ \text{COMPL} & \left\langle \boxed{3} \mid \boxed{4} \right\rangle\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right],$$

$$\left[\text{SIGN}\ \left[\text{SYNSEM}\ \ \boxed{3}\right]\right]$$

This is applied recursively until the complement list is empty.

There are Head-Complement PS rules covering the following cases:

- verb + complements
- verb + by-agent
- noun + complements
- adjective + complements
- preposition + complements

The first rule is used to attach complements to verbs and, because we treat auxiliaries as heads (see Section 7.2.2), it is also used to attach auxiliaries (and modals) to the main verb. The fact

that the rule is binary allows the processing of possible interleaved adjuncts. The second rule attaches a passive verb (s-passive or a passive past participle) to its agent. The rule is applied after the other complements have been attached (the `compls` list of the verb is saturated). This rule is the only one which is triary (passive verb + *af* (by) + agent).

Nominals can take complements when they are used attributively (`prd` value is *no*). Complements are attached to unsaturated nominals (`spr` value is *unsat*) before adjuncts. Postmodifying elements are attached to nouns before premodifying elements (determiners and attributive adjectives). To control the order of attachment of pre-modifying and postmodifying elements the feature `heading` is used.

Adjectives can take complements when they are used predicatively.

Prepositions precede nominals, finite and non-finite clauses and adverbs. In the rule attaching complements to prepositions, the preposition daughter structure-shares the `head` of the prepositional complement with the `p_compl` head feature.

The Head-Complement rule for adjective+complement is given below as an example.

```
sign:{
  procinfo=>mPSPECana['Adjective+Complement'],

  ortho=>tortho:{ string=>STRING,rest=>STRING_REST },
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tphrasal:{constr=>h_compl},
           cat=>tsubst_cat:{
                  head  =>HEAD,
                  cohead=>tcohead:{ adj=>tadj_cohead:{head=>HEAD} },
                  subj  =>SUBJ,
                  compls=>REST}}}}
  <
  [ sign:{
      procinfo=>tprocinfo:{parsehead=>y},
      ortho=>tortho:{ string=>STRING,rest=>STRINGa },
      synsem=> tsynsem:{
        syn=>tsyn:{
               str=>tphrasal:{},
               cat =>tsubst_cat:{
                       head=>HEAD=>tsubst_head:{
                             major=>tadj_major:{
                                      pos=>adj,
                                      prd=>yes}},
                       subj  =>SUBJ,
                       compls=>[COMPL|REST]}}}},
    sign:{
      ortho=>tortho:{ string=>STRINGa,rest=>STRING_REST },
      synsem=>COMPL=>tsynsem:{
        syn=>tsyn:{
               str=>tphrasal:{},
               cat=>tsubst_cat:{
                      head=>tsubst_head:{
                               major=>tsubst_major:{prd=>no}},
                      compls=>[]}}}}].
```

Optional complement extraction

Optional complement extraction has been implemented for verbs, adjectives and nouns. The following types have been declared (repeated from Section 2.1).

```
tsynsem_opt_compl >
  {
  tsynsem_extracted,
  tsynsem
    }.

type(
  tsynsem_opt_compl:{
    syn         => type({tsyn:{}}),
    sem         => type({tsem:{}})
      },'').

type(
  tsynsem_extracted:{},'').

type(
  tsynsem:{},'').
```

Unary rules for extracting optional complements have been implemented. The rules take a daughter whose initial element on the `compls` list is of type `tsynsem_extracted`. Since this type is a subtype of `tsynsem_opt_compl`, any word with a complement coded as the latter type can have this extraction rule applied. Application of the rule then results in the complement type value being changed to `tsynsem_extracted`, making it easy to see within the resulting analysis structure which complements actually are present and which were extracted.

Note that this method allows for the presence of complements occurring within the complements list before and after the optional complement, these being parsed using the usual Head-Complement (*h_compl*) rule(s). More than one complement within the list can be specified as optional and extracted by this rule.

Here are the three rules, implemented as a disjunction.

```
sign:{
  procinfo=>mPSPECana['Opt.compl.extr'],
  ortho=>ORTHO,
  synsem=>tsynsem:{
    syn=>tsyn:{
         str=>tphrasal:{constr=h_compl},
         cat=>tsubst_cat:{
                  head  =>HEAD,
                  cohead=>COHEAD,
                  subj  =>SUBJ,
                  compls=>REST,
                  by_ag=>BY_AG
                    }}}}
  <
  [ sign:{
      procinfo=>tprocinfo:{parsehead=>y},
      ortho=>ORTHO,
      synsem=>tsynsem:{
        syn=>tsyn:{
```

```
str=>tphrasal:{constr=>(~h_adj), heading=>left},
cat=>tsubst_cat:{
         head  =>HEAD=>(
             tsubst_head:{
                 major=>tn_major:{pos=>n,
                                  prd=>no}} /
             tsubst_head:{
                 major=>tv_major:{pos=>v}} /
             tsubst_head:{
                 major=>tadj_major:{pos=>adj,
                                    prd=>yes}} ),
         cohead=>COHEAD,
         subj  =>SUBJ,
         compls=>[tsynsem_extracted:{}|REST],
         by_ag=>BY_AG }}}} ].
```

7.3.2 Head-Subject Schema

The Head-Subject Schema in HPSG covers *normal* SV constructions (Pollard & Sag 1994, p. 362):

$$\text{X''}\left[\text{SUBJ }\langle\rangle\right] \rightarrow \boxed{1}\text{Y''},\ \text{X''}\left[\text{SUBJ }\left\langle\boxed{1}\right\rangle\right]$$

with the first element on the right side of the arrow being the subject, the second the head. We implement the same schema.

The following is our version of the Head-Subject Schema.

Head-Subject Schema

$$\left[\begin{array}{ll}\text{SIGN} & \left[\begin{array}{ll}\text{SYNSEM} & \left[\begin{array}{ll}\text{STR} & \left[\text{CONSTR}\quad \text{h\_subj}\right] \\ \text{CAT} & \left[\begin{array}{ll}\text{HEAD} & \boxed{1} \\ \text{COHEAD} & \boxed{1} \\ \text{SUBJ} & \langle\rangle \\ \text{COMPLS} & \langle\rangle \\ \text{BY-AG} & \boxed{3}\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right]$$

$<$

$$\left[\begin{array}{ll}\text{SIGN} & \left[\begin{array}{ll}\text{SYNSEM} & \left[\begin{array}{ll}\text{CAT} & \left[\begin{array}{ll}\text{HEAD} & \boxed{1} \\ \text{SUBJ} & \left\langle\boxed{2}\right\rangle \\ \text{COMPLS} & \langle\rangle \\ \text{BY-AG} & \boxed{3}\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right],$$

$$\left[\text{SIGN}\quad \left[\text{SYNSEM}\quad \boxed{2}\right]\right]$$

Note that in addition to `head` information, `by_ag` information is also structure-shared between mother and head daughter. The `subj` list value is structure-shared with the `synsem` of the subject, which is attached to the verb after the complements (i.e. the *compls* list is saturated).

There are Head-Subject PS rules covering the following cases:

- subject + verb
- imperative verb
- expletive *der* + verb
- expletive *det* + verb

The first rule treats normal SV constructions, i.e. both in main and subordinate clauses (the finite verbal type having value *pres* or *past*, the `nex` feature taking one of the two values *nva*, *nav*). The second rule is a unary rule handling imperative clauses which are subjectless (the finite verbal `type` having value *imper*, the `nex` feature having value *nva*). In this rule, of course, there is no `synsem` value for the subject, and the *by-ag* list is empty.

The remaining two binary rules cover expletive constructions: one for *der* expletive constructions, one for *det* expletive constructions. They are both similar to the SV rule, but in the *der* rule, the mother's complement list structure-shares with the `synsem` value of the head daughter's subject list. Here the real subject of the sentence is constrained to be an indefinite noun that is not preceded by a pre-quantifier.

In the *det* rule the real subject of the sentence can be an at-clause and the *by_ag* list must be empty.

The expletive+verb rule is given below as an example.

```
sign:{
  procinfo=>mPSPECana['det(expl)+Verb'],

  ortho=>tortho:{ string=>STRING,rest=>STRING_REST },
  synsem=>tsynsem:{
    syn=>tsyn:{
           str=>tphrasal:{constr=>h_subj},
           cat=>tsubst_cat:{
                  head  =>HEAD,
                  cohead=>tcohead:{ v=>tv_cohead:{head=>HEAD} },
                  subj  =>[],
                  compls=>SUBJ,
                  by_ag=>[]  % can't be passive
                    }}}}
  <
  [ sign:{
      procinfo=>tprocinfo:{parsehead=>y},
      ortho=>tortho:{ string=>STRING,rest=>STRINGa },
      synsem=> tsynsem:{
        syn=>tsyn:{
               str=>tphrasal:{constr=>word},
               cat=>tsubst_cat:{
                       head =>tsubst_head:{
                                major=>tpron_major:{
                                         pos=>pron,
                                         type=>expl}}}}}},
    sign:{
      ortho=>tortho:{ string=>STRINGa,rest=>STRING_REST },
      synsem=>tsynsem:{
```

```
syn=>tsyn:{
       str=>tphrasal:{},
       cat=>tsubst_cat:{
              head=>HEAD=>tsubst_head:{
                    major=>tv_fininfin:{
                              pos=>v,
                              prd=>yes}},
              subj  =>SUBJ=>[
                  tsynsem:{
                     syn=>tsyn:{
                         cat=>tsubst_cat:{
                               head=>tsubst_head:{
                                       marking=>at,
                                       major=>tv_fininfin:{
                                           pos=>v,
                                           nex=>(nav;nva),
                                           type=>(infin;pres;past)}
                                        }}}}],
              compls=>[]}}}} ].
```

7.3.3 Head-Subject-Complement Schema

The Head-Subject-Complement Schema in HPSG is parochial, because it treats English inverted clauses, i.e. interrogative clauses with auxiliaries preposed to the subject. The same is the case for our *h_subj_compl* constructions which treat the extraction of the subject in main clauses where the finite verb is on the first position in the Actualization field, followed by the subject (the Base field is empty when the clause is a simple interrogative clause, otherwise it contains a topicalized element).

Head-Subject-Complement Schema

$$\left[\begin{array}{ll} \text{SIGN} & \left[\begin{array}{ll} \text{SYNSEM} & \left[\begin{array}{ll} \text{STR} & \left[\text{CONSTR} \quad \text{h\_subj\_compl}\right] \\ \text{CAT} & \left[\begin{array}{ll} \text{HEAD} & \boxed{1} \\ \text{COHEAD} & \boxed{1} \\ \text{SUBJ} & \langle\rangle \\ \text{COMPLS} & \boxed{3} \\ \text{BY-AG} & \boxed{4} \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

$<$

$$\left[\text{SIGN} \quad \left[\text{SYNSEM} \quad \boxed{2}\right]\right],$$

$$\left[\begin{array}{ll} \text{SIGN} & \left[\begin{array}{ll} \text{SYNSEM} & \left[\begin{array}{ll} \text{CAT} & \left[\begin{array}{ll} \text{HEAD} & \boxed{1} \\ \text{SUBJ} & \left\langle \boxed{2} \right\rangle \\ \text{COMPLS} & \boxed{3} \\ \text{BY-AG} & \boxed{4} \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

There are Head-Subject-Complement PS rules covering the following two cases:

- verb + subject
- verb + expletive *der*

and are parallel to the rules implementing the corresponding *h_subj* constructions, with the exception that here the subject is following the finite verb and that the `compls` list is not saturated..

In *h_subj_compl* constructions the `nex` feature has value *vna* and the verbal `type` can only take values *pres* or *past*.

7.3.4 Head-Specifier Schema

The following parts of speech can be specifiers: articles, possessive pronouns, central quantifiers, genitive NPs.

The Head-Specifier Schema in HPSG is the following (Pollard & Sag 1994, p. 362):

$$\text{X''} \rightarrow \boxed{1}\text{Y''}\left[\text{SPEC } \boxed{2}\right],\ \boxed{2}\text{X'}\left[\text{SPR } \left\langle \boxed{1} \right\rangle\right]$$

where the former element after the arrow is the specifier and the latter is the head.

The head information is shared between the head daughter and the mother. The X' is elevated to an X'' when a specifier feature is added to it.

In our implementation `spr` is not a list which can be structure-share, but a boolean value. Our version is as follows.

Head-Specifier Schema

$$\left[\begin{array}{ll} \text{SIGN} & \left[\begin{array}{ll} \text{SYNSEM} & \left[\begin{array}{ll} \text{STR} & \left[\text{CONSTR} \quad \text{h\_spec}\right] \\ \text{CAT} & \left[\begin{array}{ll} \text{HEAD } \boxed{1} & \left[\begin{array}{ll} \text{MAJOR} & \boxed{2} \\ \text{SPR} & \text{sat} \\ \text{MARKING} & \boxed{3} \end{array}\right] \\ \text{COHEAD} & \boxed{1} \\ \text{SUBJ} & \boxed{4} \\ \text{COMPLS} & \boxed{5} \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

$<$

$$\left[\begin{array}{ll} \text{SIGN} & \left[\begin{array}{ll} \text{SYNSEM} & \left[\begin{array}{ll} \text{CAT} & \left[\begin{array}{ll} \text{SPECIFIEE} & \left[\begin{array}{ll} \text{SYNSEM} & \boxed{6} \\ \text{NEWMARKING} & \boxed{3} \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right],$$

$$\left[\begin{array}{ll} \text{SIGN} & \boxed{6}\left[\begin{array}{ll} \text{SYNSEM} & \left[\begin{array}{ll} \text{CAT} & \left[\begin{array}{ll} \text{HEAD} & \left[\begin{array}{ll} \text{MAJOR} & \boxed{1} \\ \text{SPR} & \text{unsat} \\ \text{MARKING} & \boxed{7} \end{array}\right] \\ \text{SUBJ} & \boxed{4} \\ \text{COMPLS} & \boxed{5} \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

The `selects=>tspecifiee|synsem` value of the specifier daughter is structure-shared with the `synsem` value of the head daughter. The head daughter selected must be unsaturated, indicated by the feature value `spr=>unsat`, while the value is set to `spr=>sat` in the mother node.

The marking and newmarking features are used to control the reciprocal order of the determiners. There are Head-Specifier PS rules covering the following cases:

- genitive nominals (nouns and pronouns)
- articles
- central quantifiers
- possessive pronouns

The rule for genitive nominals is given below as an example.

```
sign:{
  procinfo=>mPSPECana['spec+N-head  Genitive pron;n'],

  ortho=>tortho:{ string=>STRING,rest=>STRING_REST },
  synsem=>tsynsem:{
    syn=>tsyn:{
          str=>tphrasal:{
              heading=>right,
              constr=>h_spec},
          cat=>tsubst_cat:{
              head=>HEAD=>tsubst_head:{
                  major=>MAJOR,
                  marking=>MARKING,
                  spr=>sat,
                  p_lu_mod=>PLUMOD},
              cohead=>tcohead:{ n=>tn_cohead:{head=>HEAD} },
              subj=>SUBJ,
              compls=>COMPLS}}}}
  <
  [ sign:{
      procinfo=>tprocinfo:{parsehead=>y},
      ortho=>tortho:{ string=>STRING,rest=>STRINGa },
      synsem=>tsynsem:{
        syn=>tsyn:{
              str=>tphrasal:{},
              cat=>tsubst_cat:{
                  head=>tsubst_head:{
                      spr=>sat,
                      major=>tnom_major:{
                          pos=>(pron;n),
                          prd=>no,
                          case=>gen,
                          selects=>tspecifiee:{
                              synsem=>TSYNSEM,
                              newmarking=>MARKING}}}}}},
    sign:{
      ortho=>tortho:{ string=>STRINGa,rest=>STRING_REST },
      synsem=>TSYNSEM=>tsynsem:{
        syn=>tsyn:{
              str=>tphrasal:{
                  constr=>(h_adj;h_compl;word),
```

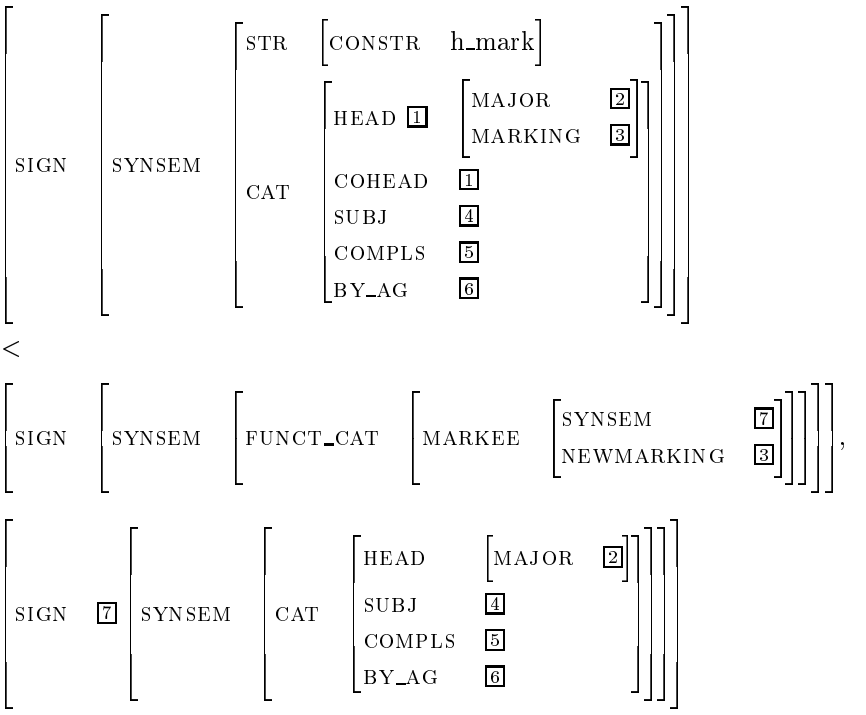
```
    heading=>right},
cat=>tsubst_cat:{
    head=>tsubst_head:{
        major=>MAJOR=>tn_major:{},
        spr=>unsat,
        p_lu_mod=>PLUMOD},
    subj=>SUBJ,
    compls=>COMPLS=>[]}}}}].
```

7.3.5 Head-Marker Schema

The Head-Marker Schema is similar to the Head-Specifier Schema, but in the Head-Marker Schema no `spr` value is projected.

Our version is as follows.

Head-Marker Schema



The `selects=>tmarkee|synsem` value of the marker daughter is structure-shared with the `synsem` value of the head daughter. The `marking` and `newmarking` features are important here for disallowing double marking (e.g. **Jeg ved at at du kommer.* (I know that that you will come.))

Although currently punctuation is treated under the Head-Marker Schema, their status may change in the future. The `marking` value for punctuation is *punct*.

7.3.6 Head-Adjunct Schema

The Head-Adjunct Schema is as follows (Pollard & Sag 1994, p. 384):

$$\text{XP} \rightarrow \text{Y''}\left[\text{MOD } \boxed{3}\right],\ \boxed{3}\text{XP}$$

where the first element after the arrow is the modifier and the second is the head. We implement the same schema[5].

Head-Adjunct Schema

$$\left[\begin{array}{ll}\text{SIGN} & \left[\begin{array}{ll}\text{SYNSEM} & \left[\begin{array}{ll}\text{STR} & \left[\text{CONSTR} \quad \text{h\_adj}\right] \\ \text{CAT} & \left[\begin{array}{ll}\text{HEAD } \boxed{1} & \left[\begin{array}{ll}\text{MAJOR} & \boxed{2} \\ \text{MARKING} & \boxed{3}\end{array}\right] \\ \text{COHEAD} & \boxed{1} \\ \text{SUBJ} & \boxed{4} \\ \text{COMPL} & \boxed{5} \\ \text{BY\_AG} & \boxed{6}\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right]$$

$<$

$$\left[\begin{array}{ll}\text{SIGN} & \left[\begin{array}{ll}\text{SYNSEM} & \left[\begin{array}{ll}\text{SUBST\_CAT} & \left[\begin{array}{ll}\text{HEAD} & \left[\begin{array}{ll}\text{MODIFIEE} & \left[\begin{array}{ll}\text{NEWMARKING} & \boxed{3} \\ \text{SYNSEM} & \boxed{7}\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right],$$

$$\left[\begin{array}{ll}\text{SIGN} & \boxed{7}\left[\begin{array}{ll}\text{SYNSEM} & \left[\begin{array}{ll}\text{CAT} & \left[\begin{array}{ll}\text{HEAD} & \left[\text{MAJOR} \quad \boxed{2}\right] \\ \text{SUBJ} & \boxed{4} \\ \text{COMPLS} & \boxed{5} \\ \text{BY\_AG} & \boxed{6}\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right]$$

The `selects=>tmodifiee|synsem` head feature of the adjunct daughter is structure-shared with the `synsem` of the head-daughter. Note that now that semantics is done in a separate processing phase, the Head-Marker and Head-Adjunct Schemata are very similar, although markers are functionals while all adjuncts are substantives.

There are Head-Adjunct rules covering the following cases:

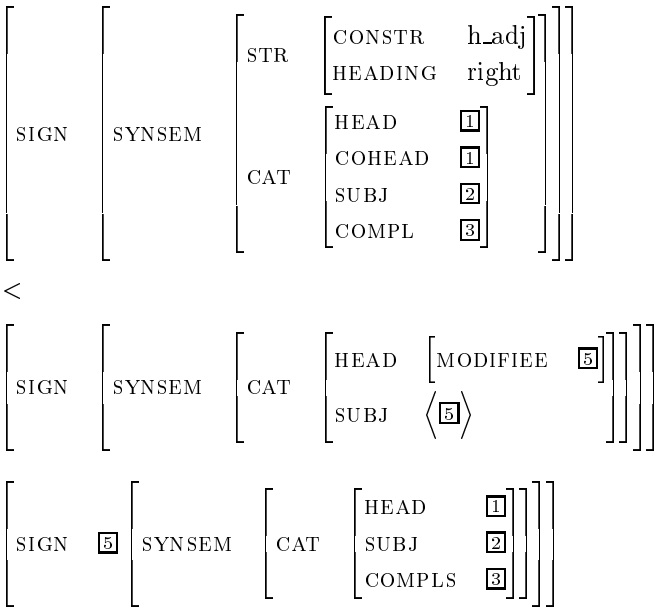
- adjuncts pre-modifying nominals
- adjuncts post-modifying nominals
- adjuncts pre-modifying main clauses (topicalized adjuncts)
- adjuncts postmodifying main and subordinate clauses
- adjuncts in the Actualization field, modifying main clauses
- adjuncts in the Actualization field, modifying subordinate clauses

Adjuncts modifying nominals

The rule for adjuncts premodifying nominals (viz. adjectives, participles, pre- and post-quantifiers) is given below in its schematic and ALEP forms. For the sake of space, the subsequent Head-Adjunct examples will be given in schematic form only.

[5] In our implementation, though, adjuncts are not semantic heads.

Head-Adjunct rule, pre-modification of nominals




```
sign:{
  procinfo=>mPSPECana['adjunct+N'],

  ortho=>tortho:{ string=>STRING,rest=>STRING_REST },
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tphrasal:{constr=>h_adj, heading=>right},
        cat=>tsubst_cat:{
            head  =>HEAD=>tsubst_head:{
                major=>MAJOR,
                spr=>SPR,
                p_lu_mod=>PLUMOD,
                marking=>MARKING},
            cohead=>tcohead:{ n=>tn_cohead:{head=>HEAD}},
            subj  =>SUBJ,
            compls=>COMPLS}}}}
  <
  [ sign:{
      procinfo=>tprocinfo:{parsehead=>y},
      ortho=>tortho:{ string=>STRING,rest=>STRINGa },
      synsem=>tsynsem:{
        syn=>tsyn:{
            str=>tphrasal:{constr=>(word;h_adj)},
            cat=>tsubst_cat:{
                head=>tsubst_head:{
                    major=>tsubst_major:{
                        pos=>(adj;v;quant),
                        selects=>tmodifiee:{
                            newmarking=>MARKING=> ~cquant,
                            synsem=>TSYNSEM=>tsynsem:{sem=>MODSEM}},
                        prd=>no}},
```

```
              subj=>[tsynsem:{
                      sem=>MODSEM}] }}}},

    sign:{
      ortho=>tortho:{ string=>STRINGa,rest=>STRING_REST },
      synsem=>TSYNSEM=>tsynsem:{
        syn=>tsyn:{
            str=>tphrasal:{heading=>right,
                           constr=>(word;h_adj;h_compl;h_spec)},
            cat=>tsubst_cat:{
                head  =>tsubst_head:{
                     major=>MAJOR=>tn_major:{pos=>n},
                     spr=>SPR,
                     p_lu_mod=>PLUMOD},
                subj  =>SUBJ,
                compls=>COMPLS=>[]}}}}].
```

Note that the subject for the premodifying adjective is the modified nominal.

Prepositional phrases can act as adjuncts which post-modify nominals. As it was the case for the rule attaching complements to nominals, also in the rule attaching adjuncts to nominals the feature `heading` is used. The schematic version of the rule for adjuncts which post-modify nominals is the following:

Head-Adjunct rule, post-modification of nominals

$$\begin{bmatrix} \text{SIGN} & \begin{bmatrix} \text{SYNSEM} & \begin{bmatrix} \text{STR} & \begin{bmatrix} \text{CONSTR} & \text{h\_adj} \end{bmatrix} \\ \text{CAT} & \begin{bmatrix} \text{HEAD} & \boxed{1} \\ \text{COHEAD} & \boxed{1} \\ \text{COMPLS} & \langle\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$<$

$$\begin{bmatrix} \text{SIGN} & \boxed{2} \begin{bmatrix} \text{SYNSEM} & \begin{bmatrix} \text{STR} & \begin{bmatrix} \text{HEADING} & \text{left} \end{bmatrix} \\ \text{CAT} & \begin{bmatrix} \text{HEAD} & \boxed{1} \\ \text{COMPLS} & \langle\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix},$$

$$\begin{bmatrix} \text{SIGN} & \begin{bmatrix} \text{SYNSEM} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{MODIFIEE} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The complement list must be saturated (complements are attached to nominals before adjuncts).

Adjuncts modifying clauses

Adverbs, prepositional phrases, temporal nominal phrases (e.g. *hele dagen* (all day)) and participles can function as adjuncts modifying main clauses. We have implemented adverbs, prepositional phrases and nominal phrases as clausal adjuncts.

Adjuncts modifying main clauses can occur in three different positions:

- after the main verb and its complements
  Jeg vil rejse til Italien <u>om en måned.</u>

(I will travel to Italy next month)

- sentence initially, before the finite verb
  Om en måned vil jeg rejse til Italien.
  (Next month I will travel to Italy.)
  (lit. Next month will I travel to Italy).

- in the so called "Actualisation field" after the finite verb or after the postponed subject
  Jeg vil helst rejse om en måned.
  (I would preferably travel to Italy next month.)
  and
  Til Italien vil jeg helst rejse om en måned.
  (To Italy I would preferably travel next month.)
  (lit: To Italy will I preferably travel in a month).

Some adjuncts can occur in all positions, some can only occur in the Actualization field, some cannot occur in the Actualisation field, some can both occur in first position and in the Actualisation field. Few adverbs have different meanings when occurring in the Actualisation field and when not.

In subordinate clauses adjuncts can only occur in the Actualisation field (after the subject and before the finite verb) and/or after the main verb and its complements (final position).

To control the position of the adverbs in clauses we have introduced the head feature `posit` taking the three boolean values *front, nexus, end* (adverbs, prepositions and temporal nominals have it). Adjuncts modifying clauses and occurring before the finite verb or after the saturated verbal projection are attached to the parsing tree when both subject and complement lists (and possibly by-agent list) are saturated. Topicalized adjuncts are attached before non-topicalized ones. As before, the order of attachment is controlled by the feature `heading`.

Adjuncts in the Actualisation field modify the entire clause.

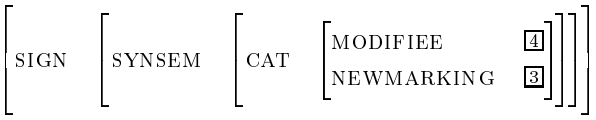
The schematic version of the rule for adjuncts post-modifying clauses is the following:

Head-Adjunct rule, premodification of clauses

$$\left[\begin{array}{ll} \text{SIGN} & \left[\begin{array}{ll} \text{SYNSEM} & \left[\begin{array}{ll} \text{STR} & \left[\begin{array}{ll} \text{CONSTR} & \text{h\_adj} \\ \text{HEADING} & \text{left} \end{array}\right] \\ \text{CAT} & \left[\begin{array}{ll} \text{HEAD}\ \boxed{1} & \left[\begin{array}{ll} \text{MAJOR} & \boxed{2} \\ \text{MARKING} & \boxed{3} \end{array}\right] \\ \text{COHEAD} & \boxed{1} \\ \text{SUBJ} & \langle\rangle \\ \text{COMPLS} & \langle\rangle \\ \text{BY-AG} & \langle\rangle \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

$<$

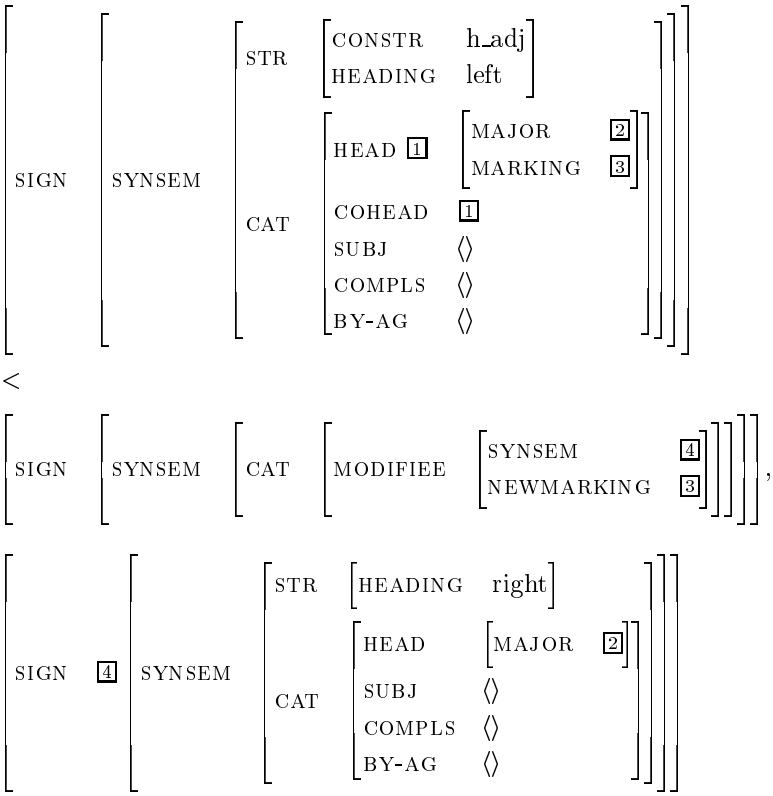
$$\left[\begin{array}{ll} \text{SIGN} & \boxed{4}\left[\begin{array}{ll} \text{SYNSEM} & \left[\begin{array}{ll} \text{STR} & \left[\text{HEADING}\quad \text{left}\right] \\ \text{CAT} & \left[\begin{array}{ll} \text{HEAD} & \left[\text{MAJOR}\quad \boxed{2}\right] \\ \text{SUBJ} & \langle\rangle \\ \text{COMPLS} & \langle\rangle \\ \text{BY-AG} & \langle\rangle \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right],$$



Because the rule covers both main and subordinated clauses, the `nex` value can be *nva*, *vna* or *nav*. The adjunct's `posit` value is *end*. The above rule causes an incorrect ambiguity in the case where adjuncts modify at-clauses (which are complementizer for the main clause). The adjunct is both attached to the main clause and to the at-clause. This could be solved by indicating that a main clause is followed by a marker and disallowing post-modifying attachment of adverbs. A preference mechanism system would be the preferred solution, however.

The schematic version of the rule for adjuncts pre-modifying clauses is parallel to the preceding rule. The `nex` feature has value *vna*. The `posit` value for the adjunct is *front*.

Head-Adjunct rule, pre-modification of clauses



Two different rules cover attachment of clausal adjuncts in the Actualization field, one for main clauses (`nex` value is *nva* or *vna*) and one for subordinate clauses (`nex` value is *nav*). The `posit` value for the adjunct is *nexus*.

Actualization adverbs which modify main clauses can occur in-between complements, thus the `compls` list must not be saturated. This is not the case for subordinate clauses which always occur before the finite verb and its complements, so the `compls` list must be saturated.

Chapter 8

Refinement

This chapter starts with a description of some general principles behind the implementation of refinement for Danish. The implementation of Predicate-Argument Structure is then discussed, comprising the treatment of control constructions and of nominal, adjectival and prepositional phrases predicatively used. The chapter concludes with the treatment of signs which do not take arguments, i.e. adjuncts, quantifiers, genitive phrases, pronouns, articles and expletives.

8.1 General principles for lexical refinement

The refinement processing phase is implemented by refinement structural rules and refinement lexica.

In refinement we have followed the general lines given in Theofilidis et al. (1994), with the exception of the treatment of adjuncts which are not considered semantic heads here (see Sections 1.4.2 and 8.3.1). The structure of `content` is the same as those authors have described with only a couple of changes.

8.1.1 Nominal and non-nominal lexical signs

The Danish implementation distinguishes between nominal and non-nominal `content` specifications (see Pollard and Sag (1994) and Theofilidis et al. (1994)). To handle deverbal and deadjectival nominals we have extended the `psoa` type to include instance-argument psoa for nominals. These are parallel to those of other PAS-taking word classes.

```
psoa > {
  rel_psoa > {
      inst_psoa > {
          inst_zero_psoa,
          inst_arg1_psoa > {
               inst_arg234_psoa > {
                   inst_arg2_psoa,
                   inst_arg34_psoa > {
                        inst_arg3_psoa,
                        inst_arg4_psoa
                     }
                  }
               }
```

```
        },
    arg_psoa > {
        arg1_psoa,
        arg234_psoa > {
            arg2_psoa,
            arg34_psoa > {
                arg3_psoa,
                arg4_psoa
             }
          }
        }
     }
}
```

In this we depart from Pedersen et al. (1995) which suggest using a typological structure for PAS which on the whole is the same for all the signs with a PAS. The structure proposed in the LINDA report is the following:

$$\left[\text{SEM}\,|\,\text{LOCAL}\left[\text{CONT}\,|\,\text{NUC}\begin{bmatrix}\text{PAS} & \textit{psoa}\\ \text{RESTR} & \textit{set\_of\_psoas}\end{bmatrix}\right]\right]$$

.

In the LINDA report, verbal and nominal signs also contain an `index` and may contain quantifiers. The `index` has the two subtypes `ind_index` and `ev_index`. Verbal signs have an `ev_index`. Event-denoting nouns have an event-index and an argument-event, while the remaining nouns have a referential index and an arg1 corresponding to the `instance` feature in Pollard and Sag (1987).

In LSGRAM the event-argument has yet to be implemented.

The structure for non-nominal signs in the present implementation is the following:

$$\left[\text{SEM}\left[\text{CONTENT}\;\textit{lq\_cont}\left[\text{RD\_CONT}\;\textit{r\_psoa}\begin{bmatrix}\text{PSOA} & \textit{psoa}\\ \text{RESTR} & \langle\rangle\end{bmatrix}\right]\right]\right]$$

.

The structure for nominal signs is as follows:

$$\left[\text{SEM}\left[\text{CONTENT}\;\textit{lq\_cont}\left[\text{RD\_CONT}\;\textit{r\_npro}\begin{bmatrix}\text{INDEX} & \boxed{1}\\ \text{POSSESSOR} & \boxed{2}\\ \text{RESTR} & \left\langle\textit{inst\_psoa}\begin{bmatrix}\text{INST} & \boxed{1}\\ \text{INST\_PSOA} & \textit{psoa}\end{bmatrix}\right\rangle\end{bmatrix}\right]\right]\right]$$

.

We have added a `possessor` feature to the type `r_index`, used for percolating information about genitive phrases and possessive pronouns to the specified nominal.

The implemented treatment of quantification and adjuncts is, in part, consistent with the solutions suggested by Theofilidis et al. (1994) (see below).

8.2 Predicate-Argument Structure of Lexemes

The argument-taking word classes are verbs, adjectives, prepositions, deverbal and deadjectival nouns, and predicatively used nouns after a copula.

Because we have implemented optional complement extraction for verbs, nouns, and adjectives in analysis, verbs, nouns and adjectives taking optional complements have only one refinement entry.[1] If the optional complements are not present, the corresponding arguments are simply left uninstantiated.

8.2.1 PAS for verbs

PAS for verbs has for the most part been implemented as proposed in Pedersen et al. (1995).

Verbs which in the analysis lexicon can take both a nominal and a verbal complement (implemented with cohead representations) must have a special entry for verbal infinitive objects in refinement lexicon. Infinitive objects are equi constructions which must be reflected in the semantic structure.

Nominal complements and finite verbal complements are covered by the appropriate default rule, having their semantics structure-shared with an appropriate argument in the argument structure for the verb, e.g.

Jeg foretrækker at du venter her.
(I prefer that you wait here.)

The semantics value of *jeg* is structure-shared with arg1 in the argument structure for *foretrække*, while the at-clause *at du venter her* is structure-shared with arg2.

Non-finite verbal complements are control structures, which must be treated as special cases (see later on control verbs), e.g.

Jeg foretrækker at vente her.
(I prefer to wait here.)

In this case *jeg* is arg1 for *foretrække* and is also the unexpressed subject for the following non-finite clause, *at vente her.*

Verbs which can take a prepositional phrase as complement where the prepositional complement can be a non-finite verb must also be coded with an extra entry. Nominal and finite clause complements are structure-shared with the appropriate argument in the argument structure for the verb, while non-finite clause complements must be treated as control constructions, e.g.

Han synes om at du arbejder.
(He likes that you work.)

vs.

Han synes om at arbejde.
(He likes to work.)

where the subject for the matrix verb, *han*, is also the unexpressed subject for the non-finite clause.

[1] Also in this we differ from the treatment of PAS in Pedersen et al. (1995), since optional complement extraction is a finesse added during implementation.

Head information about prepositional complements is assigned to the prepositional head feature `p_compl` to make it accessible from higher levels of structure (see Section 7.2.6). In Eurotra it was not possible to distinguish among the different kinds of prepositional complements from outside the prepositional phrase.

Control verbs

There are both divalent and trivalent control verbs.

Divalent equi verbs are subject equi verbs. The subject (arg1) of the matrix verb is structure-shared with the unexpressed subject (arg1) of the non-finite complement, e.g.

> *Jeg prøver at komme i aften.*
> (I (will) try to come this evening.)

The non-finite complement is arg2 in the argument structure of the matrix verb (see Section 5.4.1).

Divalent raising verbs are syntactically similar to equi verbs, e.g.

> *Han synes at være træt.*
> (He seems to be tired.)

In the case of raising verbs, however, the subject of the matrix verb is structure-shared with the unexpressed subject of the non-finite complement, but it is not assigned a role in the argument structure for the matrix verb. In the above example the arg1 in the argument structure for *synes* is missing, while arg2 is the non-finite complement *at være træt* which has the subject of the matrix verb, *han*, as arg1.

Trivalent equi verbs are indirect object equi verb.

> *Han forbyder hende at gå i biografen.*
> (He forbids her to go to the cinema.)

The non-finite clause is arg2 for the matrix verb, the indirect object maps onto arg2P (second participant) which is structure-shared with the unexpressed subject of the non-finite complement (its arg1).

Trivalent raising verbs are similar to divalent raising verbs, but they have an additional dative perceiver argument.

> *Han forekommer mig at være træt.*
> (He seems to me to be tired.)

The subject of the matrix verb (in the above case *han*) is structure-shared with the unexpressed subject of the non-finite complement, but it is not assigned a role in the argument structure for the matrix verb. Thus in the PAS for *forekomme* arg1 is missing, while arg2 is the non-finite complement which has the subject of the matrix verb (*han*) as arg1. The dative perceiver *mig* (me) is assigned arg_P.

Auxiliaries and modals

Auxiliaries do not have their own content, they inherit it from the content of the main verb whose subject is structure-shared with the subject of the auxiliary. The auxiliary *være*, *have* and *ville* influence the aspect of the main verb. The semantics of *have* and *være* sets the value of `context|background` to the concatenation of the perfect aspect (`aspect_psoa:{aspect=>perf}`)

with the `context|background` list of the subcategorized for past participle. The semantics of the auxiliary *ville* is similar, but sets the value of `context|background` to the concatenation of the prospect aspect (`aspect_psoa:{aspect=>prosp}`) with the `context|background` list of the subcategorized for infinite verb.

The auxiliary *blive* does not influence the semantics of the subcategorized past participle in this implementation.

Modals have an arg1 and arg2, where arg2 is the infinitive verb construction which follows the modal. The infinitive verb has the same arg1 as the modal (their subjects are structure-shared).

8.2.2 PAS for nouns

We distinguish between nouns used predicatively and attributively. Nouns used predicatively after a copula construction[2] always have an arg1 which is structure-shared with their subject (see Section 5.4.2).

Nouns used attributively have been implemented according to the suggestions within the LINDA report on Predicate Argument Structure (Pedersen et al. 1995), although the content of a nominal sign has been given a different structure.

Zero-valent nouns have an *inst_zero_psoa* structure (see Section 5.4.2).

Deverbal and deadjectival nouns have an argument structure similar to the verb or adjective they are derived from.

Special treatment is given to nouns which can subcategorize for a genitive nominal phrase. The argument represented by the genitive phrase (usually arg1) is structure-shared with the value of the `possessor` feature for the noun where the content of the specifying genitive is held (see Section 8.3.3).

As noted in Pedersen et al. (1995), it is difficult to distinguish between the use of a prenominal genitive as arg1 or arg2 in certain constructions. For example,

> *hans fremstilling af øl*
> (his production of beer)
>
> *Han fremstiller øl.*
> (He produces beer.)

The genitive in the first sentence is arg1, corresponding to the subject of the second sentence. Compare this to the following:

> *øllets fremstilling*
> (the production of beer)
>
> *Øl fremstilles.*
> (Beer is produced.)

Here the genitive is arg2, corresponding to the subject of the passive clause, as shown. In the present implementation the genitive phrase is always structure-shared with arg1.

It is similarly not possible to distinguish between the use of the same noun as a process noun or as a result noun (see Grimshaw (1990))[3].

[2] They are outside the scope of LINDA reports. In the present implementation we only treat predicatively used nouns after copula constructions.

[3] In the latter use the genitive phrase indicates a real relation of possession and not an arg1 or arg2 in the noun's PAS.

8.2.3 PAS for adjectives

All adjectives have arg1 structure-shared with their external argument as described in Pedersen et al. (1995). Adjectives in predicative use can also have internal arguments.

Special treatment has been given to adjectives which can take a finite or a non-finite clause as external argument[4]. The latter constructions are equi constructions, while the former are not. When adjectives take both a non-finite clause as external argument and a dative perceiver as internal argument the dative perceiver must be structure-shared with the unexpressed subject of the non-finite verb, its arg1, e.g.

Det er godt for ham at gå.
(It is good for him to walk.)

At gå er godt for ham.
(Walking is good for him.)
(lit. To walk is good for him)

As for verbs particular entries must be coded for adjectives which take as internal argument a verbal non-finite prepositional complements. When adjectives subcategorize for a preposition followed by a non-finite clause, we have an equi construction, e.g.

Han er god til at lave mad.
(He is good at cooking.)

Here the external argument (subject) of the adjective is structure-shared with the unexpressed subject of the non-finite clause (its arg1, see Section 5.4.4).

8.2.4 PAS for prepositions

Prepositions used predicatively are assumed always to have an arg2 (the complement for the preposition) and not an arg1 as in Pedersen et al. (1995). Only prepositions which are used predicatively with copula verbs have an arg1 which is structure-shared with the subject for the preposition.

Han er i London.
(He is in London.)

In the above example *London* is arg2 and *han* is arg1.

Predicatively used prepositions which do not follow a copula, have no arg1. They act as adjuncts modifying a nominal phrase or a clause (for the treatment of adjuncts see Section 8.3.1).

8.3 Treatment of other semantic phenomena

In the following we will shortly describe the treatment of other semantic phenomena, such as adjuncts, quantification, genitive phrases, case marking prepositions and pronouns.

[4] The raising version of these constructions is handled by two rules in analysis, see Chapter 7.

8.3.1 Treatment of adjuncts

We follow the treatment of adjuncts proposed by Theofilidis et al. (1994), but in this implementation adjuncts are not considered semantic heads. Instead they add their semantic information to the restriction list of the modified sign via structural refinement rules. The decision of not treating adjuncts as semantic heads was made in order to avoid the introduction of multiple lexical entries in refinement for adjuncts belonging to different word classes (nominal and non-nominal) or modifying different word classes (see Chapter 1).

The refinement entry for adverbs is given as an example:

```
sign:{
  procinfo=>mLEXSPECref[adv,LU],
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tmorphol:{lu=>LU},
        cat=>tsubst_cat:{
            head =>tsubst_head:{
               major=>tadv_major:{
                   pos => adv,
                   prd =>yes,
                   selects=>tmodifiee:{
                       synsem=>tsynsem:{
                           sem=>tsem:{
                               content=>lq_cont:{
                                  quants=>QUANTS,
                                  rd_cont=>r_psoa:{
                                       psoa=>PSOA,
                                       restr=>RESTR}}}}}}}}},
    sem=>tsem:{
        content=>lq_cont:{
            rd_cont=>r_psoa:{
                    restr=>[rel_psoa:{
                              rel=>rel:{
                                   rel_name=>LU,
                                   rel_sort=>rel_sort:{}}}]}}}}}.
```

The refinement rules treating adjuncts cover nominal adjuncts, and non nominal adjuncts. Non-nominal adjuncts can modify a nominal and a non-nominal sign. The rule treating nominal adjuncts is the following:

```
sign:{
  procinfo=>mPSPECref['Cont.Princ.(2) {nominaladjunct+head}'],
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tphrasal:{constr=>h_adj},
        cat=>tsubst_cat:{head=>tsubst_head:{major=>MAJOR}}},
    sem=>tsem:{content=>lq_cont:{quants=>QUANTS,
                                 rd_cont=>r_psoa:{psoa=>PSOA,
                                                  restr=>[CONTENT|RESTR]}}}}}
  <
  { sign:{
      procinfo=>tprocinfo:{parsehead=>y},
```

```
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tphrasal:{},
        cat=>tsubst_cat:{
            head =>tsubst_head:{
                major=>tsubst_major:{
                    pos=>n,
                    selects=>tmodifiee:{synsem=>TSYNSEM},
                    prd=>yes}}}},
    sem=>tsem:{content=>CONTENT}}},
sign:{
  synsem=>TSYNSEM=>tsynsem:{
    syn=>tsyn:{
          str=>tphrasal:{},
          cat=>tsubst_cat:{
              head=>tsubst_head:{major=>MAJOR=>tsubst_major:{}}}},
    sem=>tsem:{
         content=>lq_cont:{
            quants=>QUANTS,
            rd_cont=>r_psoa:{psoa=>PSOA,
                             restr=>RESTR}}}}}
 }.
```

8.3.2 Quantifiers

Theofilidis et al. (1994) propose the following two solutions for locally treating quantifiers (p. 11–12):

> within a structure rule accounting for attachment of a quantificational element (being identified, say, on syntactic grounds), the quantificational element's content specification is appended, at the mother node, to syntactic head's 'quants'-list...
>
> The second solution, on the other hand, claims that quantificational elements should be treated in analogy to the treatment of adjuncts in HPSG, that is, they should be considered semantic heads in a quantifier-head construction...
>
> Under this assumption, the lexical content specification of a quantificational element will be of the same type as that of the (syntactic) head it specifies, that is, of type 'lq_cont', but with a further 'quant' item being added to the semantic information that is contributed by the (syntactic) head.

We have opted for the treatment of the `quants` list within latter solution, although adjuncts are no longer semantic heads.

For the time being the `quants` list only contains the quantifiers' `lu` value, i.e. the `quantifier|q_force` is set to the quantifier's `lu` value.

All quantifiers, whether functioning as adjuncts (pre- and post-quantifiers) or specifiers (central quantifiers), add their `lu` to the quantifier list of the nominal sign they specify.

8.3.3 Genitive phrases

Genitive phrases structure-share their content with the `possessor` feature of the nominal they specify and they prepend a *definite* quantifier force to the `quants` list of it. The `possessor` feature

can indicate a real possessor specification for the specified nominal or can refer to an argument in the PAS of the specified noun, in which case it is structure-shared with this argument, which is always arg1 (see Section 8.2.2). The refinement rule treating genitive phrases is the following:

```
sign:{
  procinfo=>mPSPECref['Cont.Princ.(1) h_spec, genitive'],
  synsem=>tsynsem:{
    syn=>tsyn:{
          str=>tphrasal:{constr=>h_spec},
          cat=>tsubst_cat:{}},
    sem=>tsem:{
       content=> lq_cont:{
           quants=>[quantifier:{q_force=>defin}|QUANTS],
           rd_cont=>RD_CONT}}}}
  <
  [ sign:{
      procinfo=>tprocinfo:{parsehead=>y},
      synsem=>tsynsem:{
        syn=>tsyn:{
              str=>tphrasal:{},
              cat=>tsubst_cat:{
                  head=>tsubst_head:{
                      spr=>sat,
                      major=>tnom_major:{
                          pos=>(pron;n),
                          case=>gen,
                          selects=>tspecifiee:{}}}}},
        sem=>tsem:{content=>GENCONTENT}}},
    sign:{
      synsem=>tsynsem:{
        syn=>tsyn:{
              str=>tphrasal:{},
              cat=>tsubst_cat:{}},
        sem=>tsem:{
             content=>lq_cont:{
                 quants=>QUANTS,
                 rd_cont=>RD_CONT=>r_npro:{
                     index=>INDEX,
                     possessor=>GENCONTENT,
                     restr=>RESTR}}}}}].
```

8.3.4 Pronouns and articles

Personal pronouns are coded as `r_ppro` and are given an `ind_index`. Expletives have an `explet_index`.

Possessive pronouns have an `ind_index`. They are treated in the same way as genitive phrases, i.e. their content is structure-shared with the `possessor` feature of the nominal they specify, and they prepend a *definite* quantifier force to the nominal's `quants` list.

Articles are treated in the same manner as central quantifiers (see Section 8.3.2). Enclitic articles prepend their quantification to the quantification list of the nominal they are attached to in lifting. The quantification for enclitic articles is the same as that for non-enclitic definite articles (*den*), so that the two nominal phrases *manden* and *den gode mand* ("the man" and "the good man")

have the same quantification.

8.3.5 Non-predicative prepositions

Non-predicative prepositions do not have their own content, rather they inherit the content of their complement. Their subject is structure-shared with the subject of the complement for handling equi constructions. Thus we have three different entries for prepositions in refinement lexicon: one for the attributive usage and two for the predicative usage (prepositional phrases as modifiers, and prepositional phrases after copula constructions, see Section 5.4.5).

8.4 Structural refinement

The selection of semantic heads and the Content Principle (see Section 1.4.2) have been implemented via a set of structural refinement rules.

The semantic head is selected within the rules by choosing the head-daughter for structure-sharing of central semantic information, while the 3 cases of the Content Principle are implemented by appropriate manipulation of the `restr` and `quants` lists.

The correspondences between cases of the Content Principle and the rules are as follows:

Case 1a (possessive head-specifier constructions)
Two rules: one for genitive nominals, one for possessive pronouns.
Each rule sets the `quants` list of the mother node to the concatenation of a definite quantifier force (`quantifier:{q_force=>defin}`) with the value of the `quants` list of the head-daughter, and structure-shares the specifier's `content` with the specifiee's `possessor` feature.

Case 1b (non-possessive head-specifier constructions, head-adjunct constructions with a quantifier adjunct)
One rule for handling all articles and quantifiers.
The rule sets the `quants` list of the mother node to the concatenation of the values of the `quants` lists of the specifier/modifier and head-d aughter.

Case 2a (head-adjunct construction with a nominal adjunct)
One rule.
The rule sets the `restr` list of the mother node to the concatenation of the `content` value of the modifier and the `restr` list of the and head-daughter.

Case 2b (head-adjunct construction with a non-nominal, non-quantifier adjunct)
Two rules: one for modifying nominals, one for modifying non-nominals.
The rules set the `restr` list of the mother node to the concatenation of the values of the `restr` lists of the modifier and head-daughter. Two rules are necessary, since the semantic structures of nominals and non-nominals differ.

Case 3 (non-head-adjunct/non-head-specifier constructions)
Three rules: one each for unary, binary and triary constructions.
The rules structure-share the `content` of the head-daughter with the mother node.

The rule implementing Case 1a (possessive pronouns) is given here as an example.

```
sign:{
  procinfo=>mPSPECref['Cont.Princ.(1) h_spec, poss.pron'],
  synsem=>tsynsem:{
```

```
    syn=>tsyn:{
          str=>tphrasal:{constr=>h_spec},
          cat=>tsubst_cat:{}},
    sem=>tsem:{
       content=> lq_cont:{
           quants=>[quantifier:{q_force=>defin}|QUANTS],
           rd_cont=>RD_CONT}}}}
  <
  [ sign:{
      procinfo=>tprocinfo:{parsehead=>y},
      synsem=>tsynsem:{
        syn=>tsyn:{
              str=>tphrasal:{},
              cat=>tsubst_cat:{
                  head=>tsubst_head:{
                      major=>tpron_major:{
                          pos=>pron,
                          type=>poss,
                          selects=>tspecifiee:{}}}}},
        sem=>tsem:{content=>GENCONTENT}}},
    sign:{
      synsem=>tsynsem:{
        syn=>tsyn:{
              str=>tphrasal:{},
              cat=>tsubst_cat:{}},
        sem=>tsem:{
             content=>lq_cont:{
                 quants=>QUANTS,
                 rd_cont=>RD_CONT=>r_npro:{
                     index=>INDEX,
                     possessor=>GENCONTENT,
                     restr=>RESTR}}}}}].
```

The rule implementing Case 2b (non-nominal/non-quantifier adjunct) modification of a nominal is given here. Note the curly braces surrounding the daughters, indicating 'disordering' or free-ordering of the nodes, used to allow for the adjunct occurring before or after the head.

```
sign:{
  procinfo=>mPSPECref['Cont.Princ.(2) h_adj, {~(quant;n)+head}, r_index'],
  synsem=>tsynsem:{
    syn=>tsyn:{
        str=>tphrasal:{constr=>h_adj},
        cat=>tsubst_cat:{head=>tsubst_head:{major=>MAJOR}}},
    sem=>tsem:{content=>lq_cont:{quants=>QUANTS,
                                 rd_cont=>r_index:{index=>INDEX,
                                                   possessor=>POSSESSOR,
                                                   restr=>[RESTR|RESTRS]}}}}}
  <
  { sign:{
      synsem=>tsynsem:{
        syn=>tsyn:{
            str=>tphrasal:{},
            cat=>tsubst_cat:{
```

```
            head =>tsubst_head:{
                major=>tsubst_major:{
                    pos=>(~(quant;n)),
                    selects=>tmodifiee:{
                        synsem=>TSYNSEM}}}}},
    sem=>tsem:{content=>lq_cont:{rd_cont=>rd_cont:{restr=>[RESTR]}}}}},
sign:{
  procinfo=>tprocinfo:{parsehead=>y},
  synsem=>TSYNSEM=>tsynsem:{
    syn=>tsyn:{
          str=>tphrasal:{},
          cat=>tsubst_cat:{
              head=>tsubst_head:{
                  major=>MAJOR=>tnom_major:{pos=>(n;pron)}}}},
    sem=>tsem:{
         content=>lq_cont:{
            quants=>QUANTS,
            rd_cont=>r_index:{
                 index=>INDEX,
                 possessor=>POSSESSOR,
                 restr=>RESTRS}}}}
            }.
```

Chapter 9

Conclusion

This concludes the documentation of the Danish grammar developed using the ALEP platform and formalism.

The work was done based on an initial corpus analysis. Processing comprises a text handling component developed for the project by the Danish group and augmented by a number of other LSGRAM groups. Morphological analysis uses the TLM integrated with ALEP, implementing comprehensive processing, including parsing of compounds. Syntactical analysis and identification of predicate-argument structure has been implemented for a number of schemata, consistent with a set of principles constraining feature structures and helping to give the implementation consistency and unity. Finally, thousands of lexical entries have been migrated from other sources and integrated with the running system, not only giving a respectable lexical coverage, but also testing the platform's robustness in terms of its ability to handle large linguistic resources. A new ALEP feature, default lexical rules, has recently been implemented and tested, considerably extending the system's coverage and allowing processing of extra-lexical items. Complement extraction and co-representation of heads within complements are approaches implemented for reducing lexical ambiguity.

An important aspect of this implementation is the fact that processing is done via a single formalism and platform. This is not a heterogeneous collection of processing tools nor an ultra-fast program implementing a single, restricted functionality; this is a harmonic NLP system providing administration of files and processing results, compilation of rules and lexical entries, test suite execution and report generation, with graphical access and control throughout.

Future developments will focus primarily on robustness, synthesis, and exploitation of ALEP's transfer rules with access to all processing levels within the shared-structure analysis result.

Appendix A

Inflectional paradigms for major stems

The tables below show the suffixes for each of the major stems. Star * indicates a non-existent form. A hyphen indicates no suffix, i.e. the null suffix.

Nouns

Nouns inflect for number, genitiveness, definiteness and they have gender, which is only contrastive in singular definite forms. In these cases, alternation (shown with ";") indicates the choice is determined by the gender, either common (*-n, -en, -ns, -ens*) or neuter (*-t, -et, -ts, -ets*).

	sing nongen indef	sing nongen def	sing gen indef	sing gen def	pl nongen indef	pl nongen def	pl gen indef	pl gen def	Examples
ninfl1	-	en;et	s	ens;ets	e	ene	es	enes	stol,bord
ninfl2	-	n;t	s	ns;ts	r	rne	rs	rnes	rede,vindue
ninfl3	-	en;et	s	ens;ets	er	erne	ers	ernes	bygning,stakit
ninfl4	-	en;et	s	ens;ets	-	ene	s	enes	forslag,film
ninfl5	-	en	s	ens	e	ne	es	nes	bager
ninfl6	-	en;et	s	ens;ets	s	ene	s'	enes	job
ninfl7	-	*	s	*	*	*	*	*	Danmark
nirreg									power

Verbs

Indicative and infinitive forms inflect for voice. Finite forms also inflect for tense, and there is a distinct imperative form for most inflectional types. Present and past participial forms also occur.

	fin pres imp act	fin pres ind act	fin pres pass	fin past ind act	fin past pass	nonfin act infin	nonfin pass infin	nonfin pastpart	nonfin prespart	Examples
vinfl1	-	er	es	ede	edes	e	es	et	ende	arbejd
vinfl2	-	er	es	te	tes	e	es	t	ende	hoer
vinfl3	-	r	s	ede	edes	-	s	et	ende	bebo

```
vinfl4 -    r    *    te   *      -       *       t       ende        ske
vinfl5 -    er   es   -    es     e       es      et      ende        kom
virreg                                                               sad
```

Adjectives

Adjectives inflect for number, definiteness and gender. The suffix *-e* indicates either plural, definite, or both. Some inflectional types take a distinct neuter suffix.

```
           sing plur sing
           comm      neut
          indef def  indef      Examples
----------------------------------------
adjinfl1    -    e    t         god gode godt
adjinfl2    -    -    -         stille stille stille
adjinfl3    -    e    -         glad glade glad
adjinfl4    -    -    t         blaa blaa blaat
adjinfl5    et   ede  et        beslaegtet beslaegtede beslaegtet
adjirreg    -    -    -         lille lille/smaa lille
```

Bibliography

[EUROTRA 1991] *EUROTRA Ordbogsmanual*, L.D. Jørgensen and S. Kirchmeier-Andersen (eds.), rule coding manual for Danish v.1.1, 1991.

[EUROTRA-6/3 1991] *EUROTRA-6/3 Text Handling Design Study*, Final Report, SEMA Group Belgium, Commission of the European Communities, 1991.

[Braasch & Jørgensen 1995] A. Braasch and L.D. Jørgensen. *Type System and Lexicon Specifications for the Danish Core Grammar*, Project Number LRE 61029, Deliverable E-D3-DK, Center for Sprogteknologi, 1995.

[Bredenkamp et al.] A. Bredenkamp, F. Fouvry, T. Declerck, B. Music. *Efficient Integrated Tagging of Word Constructs*, COLING-96, Copenhagen, 1996.

[Grimshaw 1990] J. Grimshaw. *Argument Structure*, The MIT Press, Cambridge, Mass, 1990.

[Music 1995a] B. Music. *LSGRAM — Danish Design*, LSGRAM - LRE 61029, Deliverable E-D1-DK, Center for Sprogteknologi, 1995.

[Music 1995b] B. Music. *Danish Morphology in ALEP*. LSGRAM - LRE 61029, Deliverable E-D5-DK, Center for Sprogteknologi, Copenhagen, 1995.

[Music 1995c] B. Music. *Tagging Messy Details in ALEP*, LSGRAM - LRE 61029, Supplement to Deliverable E-D5-DK, Center for Sprogteknologi, Copenhagen, 1995.

[Navarretta 1996] C. Navarretta. *Design of Syntax Implementation for Danish Phrase Structure and Predicate Argument Structure*. LSGRAM - LRE 61029, Deliverable E-D6-DK, Center for Sprogteknologi, Copenhagen, 1996.

[Neville & Povlsen 1995] A. Neville, C. Povlsen. *Specifications for Determination in Danish*. LINDA - MLAP93-09 Deliverable 8, CST, Copenhagen, 1995.

[Paggio & Ørsnes 1991] P. Paggio, B. Ørsnes. *Research on Compounds: A Typology*, unpublished paper, Center for Sprogteknologi, 1991.

[Pedersen et al. 1995] B.S. Pedersen, L.D. Jørgensen, B. Ørsnes. *Specifications for Predicate-Argument Structure in Danish*. LINDA - MLAP93-09: Deliverable 5, Center for Sprogteknologi, Copenhagen, 1995.

[Pollard & Sag 1994] C. Pollard and I.A. Sag. *Head-Driven Phrase Structure Grammar*. CSLI, The University of Chicago Press, Chicago, 1994.

[Povlsen et al. 1995a] C. Povlsen, L.D. Jørgensen, B. Music. *Danish corpus analysis and priority list*. LSGRAM - LRE 61029, Deliverable E-D2-DK, Center for Sprogteknologi, Copenhagen, 1995.

[Povlsen et al. 1995b] C. Povlsen, P. Paggio, N.L. Underwood. *Specifications for Phrase Structure in Danish*. LINDA - MLAP93-09, Deliverable 4.2, Center for Sprogteknologi, 1995.

[Schmidt et al. 1996] P. Schmidt, S. Rieder, A. Theofilidis, T. Declerck. *Lean Formalisms, Linguistik Theory and Applications. Grammar Development in ALEP.* COLING-96, Copenhagen, 1996.

[Schmidt et al. 1995] P. Schmidt, S. Rieder, A. Theofilidis. *LSGRAM Lingware Development Documentation — Core Grammar*, LSGRAM - LRE 61029, Deliverable DC-WP6b (German), IAI, 1995.

[Theofilidis et al. 1994] . Theofilidis, M. Verhagen, T. Badia. *Specifications for a Common Semantic Representation Format*, LSGRAM - LRE 61029, Deliverable D-WP1a (Definition of Output Structure), 1994.

[Underwood & Jørgensen 1995] N.L. Underwood, L.D. Jørgensen. *LINDA - Specifications for Danish Inflectional Morphology*, MLAP93-09, Deliverable 2.1, 1995.