TROLL

(The Trondheim Linguistic Lexicon Project)

October 1989

This report describes work in progress, and some of the chapters and in particular the appendices will be subject to continuous updating. A full new report will appear some time in 1990, and updated versions of chapters and appendices will be available at irregular intervals. For information and comments, write to any of the participants of the project (for the present: Anneliese Pitz, Lars Johnsen, Lars Hellan) or to "TROLL", at the address

Linguistics Department
University of Trondheim
7055 Dragvoll
Norway

Trondheim, October 18, 1989.

Contents

TROLL
(The Trondheim Linguistic Lexicon Project)
October 1989

1. Introduction.

1.1. Purpose and goals.

The purpose of the project TROLL is to produce a system, called by the same name, for encoding lexical information, in principle for any language, and in a way which is neutral with regard to specific applications. That is, any part of the lexicon TROLL should be applicable to an indefinite number of uses, albeit none of these uses is part of the system itself. For purposes of the design of TROLL, we have two applications specifically in mind: using the system as a database for question-answering concerning the lexicon of a language L, and using the system as a repository for the lexical part of automatic parsing programs or speech recognition programs for L.

By the *lexicon* of a language, we understand the total amount of information tied to the *words* of the language. The information concerning a given word comprises not only its phonological/graphemic shape, its morphological structure and core meaning, but also all of its combinatorial properties, both syntactic and semantic; thereby the description of a word W includes all contributions of W to a sentence where W occurs. Hence, the lexicon will contain much information that is often classified as syntactic or combinatory semantic information. Moreover, some of this information is 'regular', in the sense that it may be predictable from other pieces of information in the system. Hence, the amount of information contained in the lexicon goes far beyond what might be thought of as the 'basic' or 'idiosyncratic' properties connected with words.

Given such a scope, it is clear that the construction of a lexicon for a language L is an overwhelmingly complex task. For this reason, key importance is given to the following requirements:

(i) At any stage, the lexicon must be open to *updating*, in two respects:

a. The amount of information that can go into a lexicon is really infinite, hence the lexicon must always be expandable. We may say that in this respect, the lexicon will *grow*.

b. New insights in the various linguistic or organizational aspects of the lexicon may call for modifications of the system, some of them so basic that we may speak of a *regeneration* of the lexicon, i.e.

the 'birth' of a new lexicon. The capacity for such regeneration is to be an essential characteristic of the system from the beginning, and to stay with it at all stages.

(ii) No single group of researchers can hope to achieve, within a reasonable amount of time, a satisfactorily complete lexicon in the sense in question. Hence it is desirable that different groups in different places can cooperate on the construction of the same lexicon. This comes out technically as a requirement of *unifiability* of various pieces of information, possibly created in different places.

The main type of target for the question-answering application is linguistic *concepts*; that is, questions will be asked about linguistic concepts (e.g., "which verbs allow adverb incorporation?", "do all verbs with transitivity alternation allow passive?", etc.). The choice of system of such concepts in TROLL is not unrelated to the parsing application: this application requires a grammar which is *complete* in the generative sense, assigning correct structure and interpretation to all sentences. We assume that by and large, the set of notions contained in the *grammar* behind such a system will include the set of concepts used for the question-answering aspect. This goes for other conceivable applications as well, relative to the parsing application. Hence, at the bottom of the lexicon TROLL, we assume a generative system of linguistic concepts and rules, required in its entirety for the parsing application, and serving as a 'pool' from which notions relevant to other applications are drawn. Section 2 gives a partial outline of this generative system.

1.2. LexScript and the representation of words.

The information-holding system itself is called *LexScript*, to be outlined in section 6. The information held by this system is essentially organized as an attribute-value (AV-) graph. A key type of entity in the system is the lexical *entry*, which contains all information related to a word. The main attributes of the graph constituting an entry for an inflected word of category V, N or A are as follows:

(1)
```
     Id:  {
          Cat:       {
               Major_cat:
               Minor_cat:
               Inst_feat:
          }
          Segmental: {
               Morphon:
               Morphgraph:
          }
          Template:     {
               SAF: ...
               LLF:  ...
               CS: ...
          }
          SemRel:
     }
```

'Id' here is a unique identifier for the entry, technically a number. 'Cat' is categorial information, 'Major_cat' defining main word class, 'Minor_cat' providing features for inflection and other syntactic features. Concerning 'Inst_feat', see section 4.

The attribute 'Segmental' (for 'Segmental properties') assembles those properties which pertain to the segmental manifestation of the entry: 'Morphon' gives morphophonological information, and 'Morphgraph' displays the graphemic counterpart of the former; see section 5.

'Template' covers the syntactic frame of the item and its semantic properties: 'SAF' stands for 'Syntactic Argument Frame', 'LLF' for 'Lexical Logical Form' and 'CS' for 'Conceptual Structure'; see sections 3 and 4.

'SemRel', finally, provides largely pragmatic information concerning the denotata of words; see Appendix 10.

The lexicon will provide information concerning each type of occurrence for a word, subsuming inflectional variants, semantic variants, syntactic frames, allomorphic and allophonic/allographic variants; there will therefore be entries giving complete specification with regard to all of these properties. In addition, the lexicon should capture intuitions concerning class membership of such occurrence types. For instance, forms like eat, eats, ate, eaten, and eating are seen as inflected variants of one and the same entity, commonly called a *lexeme*; the occurrences of eating in John is eating and John is eating fish are seen as frame alternations of one and the same verb; forms like read, readable and readability are seen as derivational variants with one and the same *root*, to mention some examples. In all of these cases, various concrete word forms belong to one and the same more abstract entity, the nature of the entity differing from case to case. To capture these 'belong' relationships, we give special importance to the notion *derivation,* as outlined in the next subsection.

1.3. Derivation of entries.

Common to all derivational relations is that in TROLL, there will be a 'source' or 'input' entry and a 'derived' or 'output' entry, the input entry representing the more abstract type of entity of the dimension instantiated by the relation. For instance, if y is what would be called an inflected form of the lexeme x, there will be a derivational function f such that y=f(x), f inducing the inflected form in question; likewise if y is one among many frame alternants for the verb x, etc.. Such functions will be called *derivational rules*, or *D-rules*, which gives the notion 'derivation' somewhat larger scope than in the terminology which contrasts 'derivation' with 'inflection': in the present terminology, inflection is one subtype of derivation, what is traditionally called '(morphological) derivation' another.

What is output entry of one D-rule may be input to another, giving rise to *sequences of D-rules*, or *derivational sequences*. Some entries are counted as *non*-derived, serving as value of no function. Such an entry will be called the *basic* entry relative to a sequence in which it takes part. At the other end of a sequence, an entry serving as argument to no function is called the *end* entry of the sequence. Derivational sequences will be represented on either of the forms (2a) or (2b), the latter being the prolog-readable version:

(2) a. 'A I B I C I D ...'
  b. 'A ++ B ++ C ++ D ...'

In either version, A refers to the basic entry and B,C,D,... are names of D-rules. In general, the configuration 'X I Y' (or 'X ++Y') can be interpreted to the effect that the rule Y applies to the entry represented by X, 'X I Y' representing a new entry. This interpretation applies iteratively from left to right, 'X' on each iteration covering the whole string to the left of the 'I', while 'Y' is always a single rule. (For a technically more accurate interpretation, see subsection 6.3 below.) For any point X in a sequence 'X I Y I Z I W ...', the part 'Y I Z I W ...' is called the *continuation* of X, while relative to W in 'X I Y I Z I W ...', the part 'X I Y I Z' is called the *history* of W.

We distinguish between D-rules interconnecting entries belonging to the *same lexeme*, and D-rules interconnecting entries belonging to *different lexems*, called *intra-lexeme* and *cross-lexeme* D-rules, respectively. Processes accounting for pairs like laugh-laughter (those traditionally nomed 'derivations') are constructible as cross lexeme rules; the main subtypes of intra-lexeme rules are D-rules inducing inflection, called *inflectional* rules, and D-rules inducing alternations in syntactic frame and perhaps logical form of a given lexeme, called *frame alternation* rules.

It is essential to the present notion of derivation that what we may call the *core meaning* of the word represented by an entry stays unaffected throughout a derivation. Among the semantic changes that are allowed, the most important ones are *aspectual* changes and *addition* of material in the structure part of LLF. As a putative constraint on this part of LLF, we assume the following *persistence* principle:

(3) Logical form persistence:

If the entry T_2 is derived from entry T_1, then the structure part of LLF of T_1 must be properly contained in the LLF of T_2.

The structure of LLF is treated in detail in section 4.

A *family* of derivational sequences is defined as the (largest) set of derivational sequences that have the same basic entry. We distinguish between those families of derivations where all end-entries belong to the *same lexeme*, and those where they do not, called intra-lexeme and cross-lexeme families, respectively. In an intra-lexeme family, all the D-rules that can apply are thus inflections and frame-alternations, and the base will be the entry representing the *stem* of the inflected word forms. In a cross lexeme family, also cross-lexeme D-rules are represented in the derivational sequences.

Of particular interest in TROLL are families whose bases are entries representing *root* morphemes: these are cross-lexeme families, yielding (ignoring compound words) the whole set of morphologically complex words with the base in question as root. We will call these families *root families*. Included here are also the intra-lexeme families branching out from every node representing a possible stem entry. (4) is an informal illustration of what a root family may look like, with the Norwegian verbal root prate 'chat' as base. 'V-infl', 'A-infl', and 'N-infl' stand for rules inducing inflectional patterns for verbs, adjectives, and nouns; som is a label for the rule affixing som to a stem, thereby creating the adjective pratsom, and het is the noun-deriving rule suffixing het, deriving pratsomhet; these sequences constitute only a subset of the full root family of prate:

(4)
```
    base ++  V-infl
    base ++  Pass      ++   V-infl
    base ++  IV_smallcl ++ V-infl
    base ++  IV_smallcl ++ Pass ++      V-infl
    base ++  som       ++   A-infl
    base ++  som  ++   het   ++   N-infl
```

Root families will be the key objects of the database, in the sense that the content of every entry representing a word (at any abstraction level: inflected word, lexeme, ...) will be a function of the history of that entry within some root family. Consequently, in the characterization of a root morpheme, not only the attributes listed in (1) will have to be specified, but also the whole set of *continuations* of the entry. For details and examples, see 6.3-6.5 and, for lists of possible D-sequences, appendices 5 and 6, with introductory remarks in 3.2 and 3.3.

Derivational rules are conceived as having as their scope the totality of an entry, meaning that the operations of a derivational rule may apply to any set of attributes inside the entry. Thus, among the rules referred to in (4), all of the -infl-rules and som and het yield segmental changes, som and het also yield changes under Cat and Template. The rules Pass and IV_smallcl both operate on Template, Pass also on Cat:Minor_cat. The ways in which the various attributes are subjected to D-rules are treated in detail in the various sections throughout.

A general point can be made here, however, viz. that D-rules will often be *parametrized*, with parameters ranging over all the attribures of an entry. That is, a rule may leave open choices as to form of some item to be added, its mode of combination, whether certain effects take place under 'Template', and so on - examples will appear in the various sections. Formally, in the description of the rules, the parameter values are entered in parenthesis behind the rule name, with no upward limitation on the number of parameters available for each rule; see appendices 3a and 3b.

In current linguistic theory, there is so far no full account of what we treat here as *derivations*, and the present proposals may be partly seen as an advancement towards such a theory. For a general remark concerning the relationship between this lexicon and linguistic theory, see otherwise the beginning of section 2.

The derivational rules in the present system may be regarded mainly as *construction rules*, adding or subtracting building blocks of the entries. Once certain aspects of an entry get marked as being subject to the application of a derivational rule, the derivational sequences acquire key importance from the perspective of question-answering to the data-base, since rule names in derivational sequences are possible targets for question-answering. In terms to be introduced in section 6, the derivational sequence thus represents an *intensional* aspect of the system. For the parsing application, on the other hand, it is the end entries which are crucial (representing the extensional aspect). Even for the purpose of a parser searching for a suitable entry to match a string, though, the derivational histories may systematize the search process, and thus be useful also from the parsing perspective. For the design of the lexicon, finally, this type of analysis provides an efficient tool for the classification of entries, and especially for later *re*classification.

1.4. LexEdit.

A special system is devised for the process of loading data into the database, called *LexEdit*. It satisfies certain general requirements:

1. It should be possible to work with for a linguist without any knowledge about programming or database organization.
2. It should be maximally error-proof. For instance, since any use of the keyboard implies the possibility of making mistakes, as much of the information-loading as possible is done via choices from menues. Thus, the listing of D-rule sequences for a given root is done via menues, corresponding to those found in appendices 6a,b.

This presentation is organized as follows. The content of LexScript will be discussed first, according to the division of entries into attributes (sections 2-5); we then treat the technical *implementation* of the system (section 6). The presentation of the content of LexScript starts with a survey of key notions and assumptions in the area of syntax and semantics (section 2); then follows the organization of syntax (section 3), semantics (section 4) and segmentals (section 5), each section including the effects of derivational processes in the types of attributes in question. Remarks are also made concerning the organization of LexEdit for each type of information.

2. The underlying grammatical concepts.

A comment may be made immediately on the relation between the TROLL system and what are commonly regarded as *linguistic theories*. A theory is in a sense a leap into the unknown; the TROLL lexicon states what is known. Still, this lexicon purports to be a tool in (among other tasks) the construction and evaluation of linguistic theories, and in this capacity has to make all the distinctions that will be relevant for these purposes. This is not possible without linguistic theories playing a major role when classifications are made in the lexicon: in constructing the lexicon, one, so to say, has to foresee what the theories may possibly claim, and to do this, one necessarily has to base oneself on some sort of theory, or rudiments of a theory.

In slightly different words, what the entries may be seen to constitute is a description of the *theorems* that ought to be derivable in a theory of the lexicon. That is, for any number of distinct entries listed for a lexeme in the present lexicon, a *theory* should be able to derive them, or counterparts with the same essential properties, as distinct entries. As such, the present lexicon is a kind of descriptive *grid* that any theory should match. Again, of course, no such grid can be constructed without the designer having a fairly clear idea what a possible theory should look like.

We assume the generally accepted division of 'layers' of the grammar, including phonology, morphology, syntax, semantics and pragmatics. The lexicon communicates with all of these levels, and so the lexicon will be partitioned into a corresponding set of levels. Most of the work done in TROLL so far has been on morphology and syntax, somewhat less on semantics and very little concrete on phonology. Our remarks in this section therefore mainly concern syntax, morphology and semantics.

The syntactic framework is broadly speaking that of generative grammar, with elements both from contemporary schools (GB, LFG, GPSG, ...) and from transformational grammar of the 60s and 70s. Of recent works which have had an impact on the TROLL system we may mention Williams 1985, Zubizarreta 1987, Grimshaw 1988, and, most directly, Hellan 1988. In semantics, our assumptions are presumably compatible with most frameworks.

In general, as far as the function of TROLL as a database for question-answering is concerned, our choice of framework and terminology should not have any excluding effect on concepts and terminologies of other frameworks, including the 'framework' of ordinary language as concerns language: correspondences of usage across frameworks will be established, so that a user can approach the database with the terminology familiar to her/him.

We now give an elementary exposition of various of the basic syntactic and semantic notions (2.1), and then sketch elements of a parsing oriented language deploying these notions (2.2). This language serves mainly as a relational encoding of the linguistic concepts to be discussed, but will, in this capacity, provide a supplementary perspective on the information held in the syntactic-semantic part of LexScript outlined in section 3.

2.1. Basic grammatical notions.

2.1.1. 'Instantiators' of words: situations and things.

Semantically, any occurrence of a word of category V, N or A has an **instantiator**. This is the entity in the (referential) world which is correctly described by the word (or an expression semantically headed by the word). For verbs, the instantiator is typically a **situation** (subsuming *events*); for adjectives, it is a *state* (which is also subsumed under situations). For instance, what is correctly described by the sentence John runs is a situation consisting of John running; the verb run is here the 'kernel', or, in a semantic sense, the 'head' of this sentence. For a particular occurrence of the verb, the instantiator is a particular situation; for the verb as such, what we call its instantiator is a *type* of situations, namely running situations in general. Likewise, what is correctly described by John is tall is the state of John being tall; here tall is semantically the head, and is instantiated by 'tall-ness' states (situations) in general. For nouns, the instantiator is either a (concrete) **thing** or a situation. We take the view that what instantiates a (root) noun like goat is not any state or situation of 'goat-ness', but simply the individual goat itself. Verbs and adjectives are thus, loosely speaking, situation-geared, while root nouns are thing-geared.

The constituent denoting the instantiator of a given noun is the NP dominating that noun, if this NP is definite singular. As a notion in the realm of 'denotation' which also covers indefinite and quantified NPs, we introduce the notion *representative*: A representative of an NP is an entity with regard to which the descriptive content of the NP is evaluated. That is to say, a representative of an NP is a denotation of the variable 'x' associated with the expressed quantifier in a standard predicate logic formalization of a sentence involving the NP. (In the terms of Barwise and Cooper 1981, a representative is something like a 'witness' in the 'witness set' of the NP.) Thus, *a representative of an NP is an instantiator of the common noun heading the NP*. (For further remarks, see section 4, especially 4.2.3.)

2.1.2. Predicates and external arguments.

Verbs, adjectives and nouns have in common that the constituent of which they are the *syntactic head* - VP, AP and CnP (common noun phrase), respectively - serves as a **predicate**, i.e., a semantically uni-valenced expression expressing a **property** (or **act**). A property (or act) is something which is 'ascribed to' an entity E, and what is expressed through such an ascription is a **situation**. The construction expressing this ascription is called a **predication**, and the expression denoting the entity E is the 'subject' of the predication. Since this 'subject' expression typically occurs as a sister of the (syntactic) maximal projection of the word, it is called the **external argument (ea)** of the word. (As a derivative way of speaking, we call this expression the external argument also relative to the VP, AP or CnP.)

Syntactically, the CnP is typically contained in its own external argument (namely, the dominating NP), whereas a VP is a sister of its 'subject'. APs behave both ways: when attributive, they are inside their external argument (viz., the dominating NP), when predicative, they are sisters of it (ignoring the possible copula). It follows that the ea of a noun is identical to the NP whose representative is the instantiator of the noun (an idea which is also behind the 'R'-role of Williams 1983 and Zubizarreta 1987, and in effect present in Montague 1974), whereas for a verb or an adjective, the ea is distinct from the constituent (if any) denoting its instantiator. In other words, in the case of nouns, the entity of which the CnP is predicated is exactly the thing *instantiating* the noun, whereas for verbs and adjectives, the instantiating situation is always distinct from the denotation of the external argument.

2.1.3. Thematic roles.

Words occur with a certain number of **arguments** (for more on this notion, see 2.1.4 and 2.1.5). Arguments are linguistic items. Many of them carry **thematic roles (theta-roles)**. One way of characterizing thematic roles may go as follows: In a construction where a V, A or N occurs, various entities are referred to which stand in a certain relation to the **instantiator** of the word. For instance, in <u>John milks the goat</u>, the entities referred to by <u>John</u> and <u>the goat</u> stand, respectively, in the 'doer' and the 'undergoer' relation to the event described; in <u>the book by John about goats</u>, John stands in the creator relation to the thing being classified, which in turn stands in an 'about-ness' relation to goats.

Theta-roles are often regarded as being associated with the words themselves - the verb <u>milk</u> is said to have (or 'assign') an Agent and a Theme role, and the noun <u>book</u> has a Creator role and an 'Aboutee' role. This is consistent with the view presented in the preceding paragraph, in that the individual words clearly are the carriers of a 'recipe' for how the role composition of the instantiator of the word is to be understood. Another intuition, which harmonizes slightly less with that view, is that in the case of situations instantiating verbs, the role bearers seem in a sense to be *constituents* of the situations, which is not quite the same as saying that the roles are relations obtaining between the situations and entities. However, both views have something to recommend them: in the case of situations, roles are 'inward percolable' to varying degrees, depending on how the situation is referred to. With verbs, the roles of a situation usually are expressed by NPs occurring inside the sentence referring to that situation. When *nouns* are used for reference to situations, however, like mistake, the role bearers are often expressed outside the NP referring to the situation, via *light verbs* such as commit, like in John committed the mistake; but they can also be expressed inside the NP, as in John's mistake. Such variation suggests that even with a situation, a role may be conceived as 'outside' it even when its bearer clearly is a constituent of it. Throughout, we will adopt parts of both perspectives; see 4.1.1 for further remarks.

As a terminological point, 'bearing a theta-role' is a property which belongs to entities in the real world: in John buys the goat, it is John, not John, which bears the Agent role. As a derivative way of speaking, though, we will also say that NPs bear theta-roles, meaning that they denote entities bearing the roles in question.

Thematic roles can be divided into **central** and **marginal** roles, relative to a given word **w**: central roles of **w** are those which *must* obtain when the instantiator of **w** obtains, marginal roles *may* obtain. 'Obtaining' in this connection means 'be understood' - it is not always necessary that a central role be syntactically *expressed* (by a phonologically realized item, a trace, PRO or pro). On this issue, see further in 2.1.4 and 2.1.5.

At various points in the literature on thematic roles (which is large, with no contribution having specific impact on the use of the notion here), it has been suggested that thematic roles have a feature-like composition. Although the role labels used here are atomic, nothing precludes in principle the adoption of a feature composition of them; cf. 4.3.1.2.

2.1.4. Syntactic arguments.

We define as an **argument** of a word **w** any constituent whose occurrence is somehow required or regulated by the presence of **w**. A **syntactic argument frame (SAF)** for **w** is a set of arguments which defines an admissible frame of occurrence for **w**. For verbs, the main types of arguments are the *external argument* (corresponding to 'subject' in traditional terminology), the *governee* (or 'direct object', abbreviated 'DO', like the book in read the book and give John the book), the *indirect object* (IO, like John in give John the book), a *predicative* (like happy in John is happy and make John happy), and a *PP-argument* (also called an *indirect argument*, like John in talk about John). (For further types, see 2.2.1 and appendix 1a.) These argument types will also be called *syntactic functions*; they represent *relations* in which the arguments stand to the verb, viz. the relations 'ea of', 'governed by', 'indirect object of', 'predicative constituent of' (not to be confused with 'predicate of', the relation mentioned in 2.1.2), and 'PP-argument of'.

'Govern' in the sense used here is a relation between a verb or preposition on the one hand, and an NP or clausal constituent with a function similar to an NP, on the other. A governor can govern only one XP-item, and adjacency is commonly required between governor and this XP-level governee. According with common assumptions, we assume that when an XP is governed by a governor G, then G also governs the head of the XP.

It is not unreasonable to assume that when a verb governs an NP, this government relation serves as a mark of which of the roles associated with the verb is assigned to this NP. Many roles are signalled by choice of preposition (i.e., when the role-bearer is a PP-argument: as opposed to adjunct PPs, we assume that the NP in a PP-argument gets assigned a role by the verb, which is however *also* expressed by the preposition), and the indirect object position also seems earmarked for a certain role, the bene/malefactive. For any verb, there generally seem to be only two theta-roles in its frame which are assigned without help of these devices, and for these roles, the following principle can be hypothesized to work: the highest (according to a certain 'agentivity' hierarchy) of the two roles is given to the (unique) external argument, the lower one is given to the (unique) governee, i.e., the direct object. (In accordance with this, the external argument and the direct object may be called the *direct arguments* of the verb.)

For some governing words, the governed items are *obligatory*, in the sense that they cannot be omitted, given the presence of the governor. (We keep open the possibility that also *un*governed items can be obligatory in the present sense, like indirect objects for certain verbs.) Obligatoriness is clearly a property distinct from that of being governed, but it is not obvious that it is a homogeneous notion itself. For instance, from a different angle, 'be obligatory' might be interpreted as 'must be expressed': what such a property belongs to is a *thematic role*, whereas 'omissibility' pertains to syntactic items. The next subsection pursues the perspective of realizing thematic roles, and then returns to the topic of obligatoriness.

For nouns the main argument types are PP-arguments, genitives and, in some languages, incorporated nouns, aside from the external argument (see above). For adjectives, PP-arguments are the only type, apart from the external argument. Obligatoriness of arguments is rare for these word classes; one type is discussed in Grimshaw 1988. In this exposition, we mainly discuss verbal argument frames.

The *syntactic argument frame* (SAF) of a given word w is assumed to define those properties of the environment of w in any string where w occurs which are relevant to the syntactic rules and principles of the language in question. It is a moot question whether information about *thematic roles* must be accessible to these rules (cf. e.g. Hellan 1988 to the effect that it must, Grimshaw

1988 that it must not); we open for the possibility that it may, so that a syntactic structure will be assumed to be characterized at least along the following three dimensions: *thematic role, formal category* and *syntactic function.* A lexical syntactic argument frame, which defines the admissible syntactic frame in which a word may occur, accordingly must be characterized along the same three dimensions. That is to say, each argument in a frame is characterized with regard to these three properties.

(The conception of a SAF as involving three layers of information, such that each argument is characterized by category, thematic role and syntactic function, is in accordance with the notion of *Lexical Linking* in Hellan 1988.)

2.1.5. Implicit arguments.

In a given construction, a thematic role may have either of the following possibilities of 'realization':

a. It is not understood at all.

b. It is understood, given the semantics of the head word, but it plays no grammatical role.

c. It is understood, it plays a grammatical role, but it is still a degree lower in 'visibility' than such standard items as trace, pro and PRO.

d. It is understood, and it is expressed by a phonologically realized item or either of the empty categories trace, pro or PRO.

A typical example of possibility b. is the agent role in English middle constructions. Here we will say that the role is **suppressed**. Possibility c. is treated extensively in Hellan 1988, Roberts 1987 and others, and we say that in this case, the role is represented by an **implicit argument**. Examples are the agent in short form passives, the unexpressed agent in an NP like <u>the destruction</u>, to mention a few. Arguments being in principle *linguistic items,* we can *not* say that the role by itself *is* an implicit argument; the argument in question must be an item at the expression level. For the item serving as implicit argument, we introduce a new type of empty category called 'Referential phrase' (**RP**), which counts as the *formal* category of the implicit argument; the notion 'implicit argument' itself is treated as a *function* category. Given the thematic role realized by the implicit argument, this is thus a syntactic argument formally of the same type as the other syntactic arguments. In Hellan 1988, it is proposed that implicit arguments emerge only at a special level of representation, *P-structure*, which is distinct from the ordinary syntactic levels (see 2.1.6). The introduction of such a level - if adopted - will belong in the parsing mechanism, and a 'level-confined' status of implicit arguments will in case have to be marked in the lexical description. Here we give no such marking, pending the issue of whether such a level should be assumed.

Both the suppressed roles and those represented by implicit arguments are supposed to be *central*, relative to the head word in question.

Implicit arguments typically alternate with phonologically full expressions. These alternation possibilities lead us back to the notion 'obligatoriness': in some constructions where the head word allows a syntactically fully visible category to be omitted, this will formally, in the lexical description of the word, be represented as an *alternation* between various realizations of the role in question, including the realization as an implicit argument. Thus, what might seem like an *optional* occurrence of the 'visible' item XP may really be the *obligatory disjunction* of the occurrence of XP and that of an implicit argument. For verbs, these are the cases falling under the phenomena 'DO-deletion' and passive, described in appendix B, both constructed as *derivations* in the present system. Implicit arguments in nominalizations will also arise via derivational processes. Whether there are also implicit arguments which must be posited in *un*derived SAFs, obligatorily or optionally, is not clear in general, and in the entries for *verbs* considered in this note, implicit arguments will arise exclusively via derivational processes.

It should be noted that the question of deciding whether an implicit argument obtains or not remains to be fully clarified. The syntactic criterion, apart from alternation with a phonologically visible item, is that the presence of an implicit argument in a string is analytically required by its capacity as a binder of an anaphor or some other syntactic process: for such cases, the SAF should at least supply the *possibility* that the implicit argument appears. The semantically necessary criterion is that the role is understood, but this is not a sufficient criterion, as it does not distinguish between suppressed arguments and implicit arguments. Note that if an implicit argument is treated as optional and does *not* appear in a given SAF, the *thematic role* otherwise realized by this implicit argument is still in what we call the *Conceptual structure* of the verb in question; see 4.1.2.

2.1.6. Levels of representation and the grammar as an accepting device.

From a parsing perspective, it is interesting to model the rules and principles of a grammar as *acceptance* mechanisms (see Hellan 1988 for a general exposition of such a model of grammar). In the case of lexical information, such a strategy can be implemented by construing the templates of words as sets of syntactic and semantic conditions on what occurs in the environment of the words in input strings. Thus, if the environment of a word w in a given string S satisfies none of the templates of w, S is ungrammatical, and if S does satisfy some template T for w, then T determines (partly or completely) the syntactic structure and semantic interpretation assigned to S.

Such an acceptance strategy can also be made apply to non-lexical phenomena, such as anaphora and trace-binding. In general, one wants the parsing mechanism, to as large an extent as possible, to apply to sentences in their 'surface' form. We want to keep open the possibility that reconstruction of an underlying level like **NP-structure** of van Riemsdijk and Williams1981 must be available, but no lexical frame is defined for any putative level 'deeper' than that. Simplifying considerably, we thus assume that for parsing purposes, a 'syntactic derivation' of a sentence s starts with s presenting itself in its S-structure configuration, whereupon it produces a version of s with its NP-structure configuration if items remain unaccepted in the S-structure version. (P-structure, if used, comes into play as an expansion of NP-structure in turn; cf. 2.1.5.)

2.1.7. The universality of templates.

The conditions couched in the SAF of a template will contain predicates of varying degree of abstractness - ranging e.g. from linear precedence predicates, via predicates for syntactic categories and syntactic functions, to predicates expressing semantic roles. The higher the level of abstractness of the predicates, the higher is - most likely - the level of universality of the conditions containing them. Abstracting away lower level conditions concerning such factors as linear precedence, one may thus define partial templates of highly cross linguistic relevance. In the templates to be outlined below, this is what we do. For each language, general rules must then be added, stating e.g. where a governee is located relative to its governor, whether external argument status is marked by place in a structure or by case marking, etc. Such language specific rules are needed whether the grammar is conceived as an abstract system or implemented as a parsing system.

2.2. Predicates for an abstract parsing application.

In 2.2.1 we give samples of predicates in a language we call the *parsing language;* the full list is given in appendix 1a. The language is not part of any actual parsing program, but its predicates correspond to most of the concepts introduced above, in a form amenable to use in a prolog parsing program. 2.2.2 provides some illustrations of how the predicates of the language, or really the clauses they support, can be used to define a lexical template. The clauses of the parsing language have the following content:

Given a word or morph **w** defined by an entry **E**, a special variable **z**, called the **entry variable**, will represent **w** in the specification of templates in **E**. Each constituent in the SAF of **w** is represented by a specific **constituent variable** $\mathbf{x_i}$. The representation of a template of **w** consists of a sequence of clauses, each composed of a relation expression and one or more arguments, usually **z** and/or $\mathbf{x_i}$.

In the present connection, the parsing language has as its main function to provide a supplementary perspective on the system LexScript; in some respects it may be seen as an interpretation of it. For most of the predicates used, more linguistic documentation is of course called for, but as that would exceed the limits of this note, we mainly provide just illustrating examples.

2.2.1. Parsing language predicates.

In what follows, the parsing language predicates are given in a clausal context, where 'z' and 'x' mark the positions of the entry variable **z** and the constituent variable $\mathbf{x_i}$, respectively, relative to the predicate, and 'u' and 'v' hold the place of variables distinct from **z** and $\mathbf{x_i}$ in the relevant sequence. In the illustrating examples, the constituent corresponding to the first variable ('x' or 'u') is given in boldface, the constituent corresponding to the second variable ('z' or 'v') in italics.

The grouping of predicates under 'Thematic roles', 'Semantic predicates', 'Syntactic category' and 'Syntactic function' does not play any formal role in this language, but facilitates seeing the connections between this interpretive language and the corresponding elements in LexScript.

Expression	Reading and example
Thematic roles	
ag(x,z)	x realizes the agent role associated with z (**John** *ate* the cake)
exp(x,z)	x realizes the experiencer role associated with z (John *worries* **Bill**)
th(x,z)	x realizes the theme role associated with z (John *ate* **the cake**)
ben(x,z)	x realizes the benefactive role associated with z (John *gave* **Bill** a book)
dir(x,z)	x realizes the directionality role connected with z (John *sent* the book **to me**)
loc(x,z)	x realizes the locative role associated with z (John *lives* **in London**)
pred(x,z)	x is a predicate in the syntactic frame of z (John *made* Bill **sick**)

deg_pred(x,z)	x is a degenerate predicate in the syntactic frame of z (Jon *dummet* seg **ut** 'Jon made a fool of himself')
no_role(x,z)	x has no semantic role with regard to z, but possibly with regard to other elements (John *made* **Bill** sick)
no_role(x)	x has no semantic function (**it** rains)
cog_obj(x,z)	x is a 'cognate object' of z (John *died* **a dreadful death**)
inher_obj(x,z)	x is an inherent object of z (Jon *harket* **slim**)

Semantic predicates

inst(y,z)	y is the instantiator of z
rep(x,y)	x is a representative of y
situation(y)	y is a situation
agent_of(x,y)	the representative of x carries the agent-of relation to the situation y

Syntactic functions

ea(x,z)	x is external argument of z (**John** *read* the book)
gov(x,z)	x is governed by z (*read* **the book**, *on* **the floor**)
io(x,z)	x is indirect object of z (John *gave* **Bill** a book)
pp_arg(x,z)	x is a PP expressing a central role associated with z (John *talked* **with Mary**)
predic(x,z)	x functions as a predicative in the frame of z, either as a full predicate ('pred') or as a degenerate predicate ('degpred') (John *made* Bill **sick**)
impl_arg(x,z)	x is an implicit argument of z (the **x** *attack*, John was *killed* **x**)
adverbial(x,z)	x has adverbial function with regard to z (*go* **quickly**, *sing* **in Budapest**)

gen(x,z)	x functions as genitive in relation to z (**John's** *book*, *vennen* **til Jon**)
refl(x,z)	x is a 'de-argumentized' reflexive associated with the verb z (Jon *skammer* **seg**, Jon *vasker* **seg**, Jon *gikk* **seg** en tur)
incorp(x,z)	x is incorporated as sister of z in a word (stem) where z is head (**hus***bygging*)
attrib(x,z)	x is attribute of z (a **yellow** *house*)
lightv_float(x,z)	x is subject for a 'light verb' which has z as complement (**John** committed *murder*)

Category predicates

max_proj(u,v)	u is the maximal projection of v
P(x)	x is a preposition
PP(x)	x is a prepositional phrase
Adv(x)	x is an adverb
AdvP(x)	x is an adverb phrase
AP(x)	x is an adjective phrase
NP(x)	x is a noun phrase
RP(x)	x is a 'referential phrase' (the carrier of the function 'implicit argument')
[seg]NP(x)	x is a seg-reflexive
[ut]Adv(x)	x is the adverb <u>ut</u>
[[ut]Adv]AdvP(x)	x is an adverb phrase consisting only of the head <u>ut</u>
[_[ut]Adv_]AdvP(x)	x is an adverb phrase with the head <u>ut</u>, and possibly other items
[...[ut]Adv...]AdvP(x)	x is an adverb phrase with the head <u>ut</u>, and necessarily also other items
[e]NP(x)	x is a trace of category noun phrase

In order to become operative in automatic parsing, many of these notions need explicit definitions or characterizations in terms of syntactic structure, linear precedence, etc. Moreover, the level of analysis represented by these clauses is one which is relevant for a range of languages, including ones with a less configurational status than Norwegian and English; this is in accordance with our remark in 2.1.7. The addition of structurally more explicit definitions and language particular parameter values will not be entered into here.

2.2.2. Templates defined in parsing-level predicates.

The following exemplifies the use of some of the clauses listed above. The example shows part of that template for the Norwegian verb <u>bruke</u> 'use' which defines the possible environment where the verb has a subject, a direct object and a prepositional phrase expressing an application, corresponding to the English expression 'use...for...'. Other environments, such as the one without a PP, are not covered by this template. (1) is the information in this template corresponding to what we have called SAF, (2) is essentially what constitutes the LLF going with this SAF. As the clauses in (1) and (2) are all part of the same conjunction, and no ordering applies inside conjunctions, the representation in this language does not formally bring out SAF and LLF as distinct objects - this is however done in LexScript, from which representations containing sequences such as (1) and (2) are derived. 'z' is still the entry variable, which is unified with the identifier (on this notion, see section 7) of <u>bruke</u> (and thus neither the graphemic, phonological nor any other special representation associated with the word), and x_i is a constituent variable.

(1) ag(x_1,z) & NP(x_1) & ea(x_1,z) & th(x_2,z) & NP(x_2) & gov(x_2,z) & applic(x_3,z) & NP(x_3) & P(<u>til</u>) & gov(x_3,<u>til</u>) & pp_arg(<u>til</u>,z)

The first three conjuncts in (1) characterize the subject, the next three the object, and the remaining part characterizes the <u>til</u>-constituent expressing the application. (2) is then the LLF part:

(2) inst(y,z) & sit(y) & y=bruke & agent_of(rep(x_1),y) & theme_of(rep(x_2),y) & applic_of(rep(x_3),y)

'rep(x)' is read 'the representative of x'; see 2.1.1. The way the roles are entered in (2) comes close to the first conception of theta-roles mentioned in 2.1.3, to be made explicit in section 4.

In syntactic parsing application of (1) & (2), the idea is that putative sentence strings to be parsed are represented on the same format as that in (1) & (2), and acceptance takes place when the string can be **unified** with (1) & (2). A full string is accepted when all of its representative clauses have been unified with clauses residing in lexical templates (constituents like time- and place adverbials are usually not listed in the SAF of lexical templates; we leave their treatment in syntactic parsing open here).

3. Syntactic frames.

This section addresses the subattribute 'Template:SAF', where 'SAF' means 'syntactic argument frame'. In 3.1 we describe the general content of SAF, in 3.2. we address the phenomenon of *derivation* with regard to SAF.

3.1. The representation of syntactic argument frames.

The SAF of an entry is a list of the syntactic *arguments* that the word represented takes. Words of all major word classes take arguments, and we first describe how arguments are generally represented.

3.1.1. The constituency of **arguments.**

As stated in 2.1.4, each argument in a lexical frame is characterized with respect to *thematic role*, *syntactic category* and *syntactic function*. More formally speaking, the argument is an ordered triple of elements belonging to these categories, abstractly thus of the form <R,K,F>. The predicates entered under role and function are understood as being *relations* between the constituent in question and the entry identifier, whereas categories are one place predicates. The list below shows sample LexScript predicates in these three functions, together with their interpretation in the terms of the parsing language outlined in 2.2; a full list is given in appendix 1b. As we did in 2.2 and appendix 1a, we enter the interpretative predicates in their clausal context, to indicate their valence; as before, 'x' is the constituent variable and 'z' the entry variable. It will be noted that most of the LexScript predicates have atomic counterparts in the interpretation language, but not all. We comment later on the status of these exceptions - see 3.1.2.

LEXSCRIPT PREDICATES

Predicate	Comment	Interpretation in the parsing level language
Role (R):		
ag		ag(x,z)
exp		exp(x,z)
th		th(x,z)
ben		ben(x,z)
dir		dir(x,z)

loc		loc(x,z)
prd		pred(x,z)
no		no_role(x,z)
inherob		inher_obj(x,z)
norole		no_role(x)
cogob		cog_obj(x,z)
degprd		deg_pred(x,z)
wholepart		whole_part(x,z)
scsu	'small clause subj', with intr. verb	λuλv[pred(w,v) & ea(u,w)] (x,z) {'w' is shared with the argument with function 'predic'}
tvscsu	'small clause subj', with trans. verb	λuλv[pred(w,v) & th(u,v)&ea(u,w)](x,z) {'w' is shared with the argument with function 'predic'}

Category (K):

np		NP(x)
at_S		at_clause(x)
seg		[seg]NP(x)
pp		PP(x)
ap		AP(x)
rp		RP(x)
adv		Adv(x)
advp		AdvP(x)
adv_advp		[Adv]AdvP(x)
word_ut_adv	this instantiates a format used for all adverbs, and in turn all categories; likewise for the following 3 entries	[ut]Adv(x)
word_ut_advp		[_[ut]Adv_]AdvP(x)
word_ut_advp_excl		[[ut]Adv]AdvP(x)

word_ut_advp_add		[...[ut]Adv...]AdvP(x)
e		[e]NP(x)

Function (F):

ea		ea(x,z)
gov		gov(x,z)
io		io(x,z)
pgov	x is governed by a preposition	$\lambda u\lambda v[\exists y[P(y) \ \& \ gov(u,y) \ \& \ pp\_arg(y,v)]] \ (x,z)$
adv		adverbial(x,z)
gen		gen(x,z)
incorp		incorp(x,z)
attrib		attrib(x,z)
lightv		lightv_float(x,z)
på	x is governed by the prep. på, which itself heads a pp_arg of z (this predicate is used only in the context <X,np,_>, where X is not prd)	$\lambda u\lambda v[P(\underline{på}) \ \& \ gov(u,\underline{på}) \ \& \ pp\_arg(\underline{på},v)] \ (x,z)$
pp_arg		pp_arg(x,z)
predic		predic(x,z)
implarg		impl_arg(x,z)
refl		refl(x,z)

In the LexScript terms, an agent subject of verb **w** is thus identified as the ordered triple <ag,np,ea>, in the prolog-readible formalism written

rkf(ag,np,ea),

where 'rkf' stands for 'role, category, function'. It is generally assumed that *no two arguments inside a single SAF can have the same rkf-characterization.*

3.1.2. The constituency of a **SAF**, and illustration of the LexScript predicates.

(1a) is the LexScript SAF for the use of bruke mentioned in 2.2.2, i.e., the frame 'SU bruke DO *til* NP' ('SU use DO for NP'). (1b) illustrates how the various predicates in (1a) get connected with translations in the parsing language, yielding (1c), a repeated version of (1) in section 2.

(1) a. SAF: {rkf(ag,np,ea), rkf(th,np,gov), rkf(applic,np,til)}

b. rkf(ag,np,ea):

R: ag $\Rightarrow$ ag(x_1,z)
K: np $\Rightarrow$ NP(x_1)
F: ea $\Rightarrow$ ea(x_1,z)

rkf(th,np,gov):

R: th $\Rightarrow$ th(x_2,z)
K: np $\Rightarrow$ NP(x_2)
F: gov $\Rightarrow$ gov(x_2,z)

rkf(applic,np,til):

R: applic $\Rightarrow$ applic(x_3,z)
K: np $\Rightarrow$ NP(x_3)
F: til $\Rightarrow$ $\lambda u \lambda v$[P($\underline{\text{til}}$) & gov(u,$\underline{\text{til}}$) & pp_arg($\underline{\text{til}}$,v)](x_3,z)

c. ag(x_1,z) & NP(x_1) & ea(x_1,z) & th(x_2,z) & NP(x_2) & gov(x_2,z) & applic(x_3,z) & NP(x_3) & P($\underline{\text{til}}$) & gov(x_3,$\underline{\text{til}}$) & pp_arg($\underline{\text{til}}$,z)

As we have already said in 2.2, 'z' will unify with the entry identifier, whereas each x_i unifies with the constituent of the SAF-attribute where it emerges. The example illustrates the possibility of a LexScript predicate having a complex translation in the parsing language, viz. the item 'til', a preposition whose appearance in the F-slot signifies 'the NP is governed by the preposition, which in turn bears the PP-argument relation to the (entry) verb'. More LexScript predicates than those listed above may turn out to have complex translations.

We stress again that these translations serve only as an abstract illustration of how we envisage a possible connection between LexScript and a parsing application; thus, in section 7, no formal apparatus for the translation procedure, including the variable instantiation mentioned, will be presented. In particular, we abstain here from giving an exact account of how the variable sharing mentioned in connection with the LexScript predicates 'scsu' and 'tvscsu' (and likewise 'part' and 'whole' in appendix 1b) is to be implemented - our assumption is that this will involve no further apparatus than that of 'til', just slightly more complex lambda-expressions than those listed as translations for these predicates.

3.2. Derivational effects at SAF.

As stated in the Introduction, derivational rules, or D-rules, apply with respect to all aspects of an entry. We here describe the effects of D-rules at SAF, later sections treat their effects at the other levels.

At SAF, two types of D-rules have an effect: *cross lexeme rules* and *frame alternation rules*.. As an example of the latter, the Norwegian verb <u>spise</u> 'eat' has a multiplicity of distinct SAFs, including those matching the syntactic frames in examples such as (2), represented in SAF-form in (3):

(2)

a. <u>Jon spiser fisken</u> 'Jon eats the fish'

b. <u>Jon spiser</u> 'Jon eats

c. <u>Jon spiste tennene i stykker</u> 'Jon ate the teeth to pieces' (in the sense that the teeth were not directly eaten)

(3)

a. rkf(ag,np,ea), rkf(th,np,gov)

b. rkf(ag,np,ea), rkf(th,rp,implarg)

c. rkf(ag,np,ea), rkf(th,rp,implarg), rkf(scsu,np,gov), rkf(prd,advp,predic)

We assume that the entry containing (3b) is derived from the entry containing (3a) by the frame alternation rule 'DO-deletion', whose SAF part is stated in (4a); likewise the entry containing (3c) is derived from the one containing (3b) by the frame alternation rule 'IV-smallcl_AdvP', whose SAF operation is stated (in a simplified version) in (4b); it will be observed that DO-deletion effects a replacement of the 'visible' governee from the base form by an implicit argument, and that IV-smallcl_AdvP introduces a new governee and a predicative:

(4)

a. rkf(X,np,gov) ->> rkf(X,rp,implarg)

b. 0 ->> rkf(scsu,np,gov), 0 ->> rkf(prd,advp,predic)

Quite often, the specification of the frame of a verb will require listing of specific words capable of occurring there; this holds of prepositions and directional adverbs in particular. Such listing may be due to various factors. One is that the choice of word is completely idiosyncratic, predictable from nothing else. Another is that the construction (optionally or obligatorily) has a more or less non-compositional meaning with the word in question, so that, to make an appropriate LLF available, a separate template must be defined for this choice of word. A third circumstance leading to word specification is that even when there may well be a regular connection between, say, the choice of preposition and the theta-role signalled by the PP, it is incumbent on the lexicon that it provides the *facts* from which such a possible generalization may be drawn. Also in such cases there should then be specification of individual words in the SAF. All of these cases call for *parametrized* rules (cf. 1.3), the parameter in question being the choice of word.

The SAF operations of cross-lexeme D-rules are of exactly the same type as those for intra-lexeme rules - the differences between these types of rules reside in other attributes, namely 'Cat' (since cross-lexeme rules often change the category) and 'Segmental' (since cross-lexeme rules often involve affixation or vowel change).

One type of processes which have features of both frame alternation rules and cross lexeme rules is what we call *Incorporation*, in Norwegian exemplified by alternations like <u>tale om</u> - <u>omtale</u>, and <u>male rød</u> - <u>rødmale</u>: like many cross lexeme rules, they effect affixation, but the affixed items are not drawn from a fixed list, but are rather items which can occur as independent items in the SAF of the stem of the affixation. For this reason, we treat incorporation as a kind of frame alternation, with the particular feature that the new piece of 'frame' introduced has a syntactic function that we call 'incorp'. As these processes are particularly sensitive to the choice of word/morpheme incorporated, the D-rules representing them call for parametrization. As the incorporation phenomenon has not been treated in full in TROLL yet, however, the rules for incorporation stated in appendix 3 are so far left non-parametrized.

A near complete list of the intra-lexeme D-rules assumed for verbs in Norwegian is given in appendix 3a, and a range of cross-lexeme rules are stated in appendix 3b, both in their prolog readable form. For the intra-lexeme rules, there is also an informal presentation in appendix 7. In general, establishing a set of D-rules entails having a set of *basic* entries, and the list of the SAFs of such entries for Norwegian verbs is given in appendix 2. The full list of possible derivational sequences for each type of basic entry is given, for intra-lexeme verbal families in Norwegian, in appendix 6a, and for cross-lexeme families in appendix 6b. Parts of these specifications are provided also for German and Dutch, see appendices 8 and 9.

(Intra-lexeme) frame alternation D-rules are usually labelled according to their SAF part, as this is their most salient part (the main exception being the rules called 'Causativization', which have their name from the LLF operation), whereas cross-lexeme rules mostly get their name from the LLF operation; inflectional rules are largely labelled according to their 'Minor_cat' effect.

3.3. D-sequences in LexEdit.

For the LexEdit choice of D-sequences of frame-alternation rules, an extra feature is introduced in the menues of D-sequences. To render more perspicuous what type of construction a sequence represents, we enter an actual sentence or VP of the language described as an atomic label for the sequence, as illustrated in (5):

(5)

```
    Per_spiser_maten:        [ base]
    Per_spiser:              [ base | DO-deletion]
    Per_spiser_maten_opp:    [ base | DO-deletion |IVsmallcl_AdvP]
```

A full list of such convenience labels is provided in the appendices 6a,b. Note that the sentence functions as an atomic name, with no internal structure, with the sole purpose of facilitating the choices in the template menu. The way arrays like (6) now come into play in LexEdit is as follows.

Upon encountering the verb, the lexicon worker first assigns it a basic template, which amounts to making a choice from the list of around twenty basic templates (cf. appendix 2). The choice is marked by an identifying mark, such as 'TV' . Once such a marking has been made, a menu of *the largest possible number of templates for this type of verb* presents itself on the screen. From this menu, those templates which actually obtain for the verb in question are then marked, by a clicking procedure. From an interface menu processed in this way, only those AV-pairs are transferred to LexScript whose representative attribute in the interface menu has been marked.

The following points may be noted. Where there are multiple alternative specifications, the line in question gets repeated as many times as necessary. If further rules 'branch' from the specification point, whether the specification is consequential or not, a certain amount of combinatorial explosion may result, but there is no way of avoiding this situation.

In some cases, there may be more than one possible way of analyzing a construction in terms of derivational history. For instance, whether a construction like shoot bullets is analyzed by a TV basis template or by a template derived from an IV basis by Inherent object-insertion, may be difficult to decide at the moment where a menu is processed. In that case, as far as the specification of the end template is concerned, the choice is immaterial, both options leading to a correct accepting template. There is no need to avoid such situations, even though they may seem to involve some redundancy, since, for 'robustness' purposes, it may be desirable to have more than one way leading to the right goal.

These remarks have been concerned with the SAF part of templates. Remarks about LLF and Segmental properties follow below.

4. Semantic components of TROLL

4.1. Introduction

The semantic part of TROLL involves three data structure components: **LLF** (Lexical Logical Form), **CS** (Conceptual Structure), and the role part of SAF. These together cover *role* and situation structure aspects of meaning. In addition there is an aspect of meaning which we will refer to as **SemProp** (Semantic properties), whose representation is distributed over various of these components. Outside the semantics proper is a further component we call **SemRel** (Semantic relations), which is a unit both as an aspect of meaning and in the data structure. This section considers, for each component and aspect except SemRel, its linguistic content and how it should be represented in the data structures. As the definitive approach to semantics has not yet been settled upon, the section will contain more discussion of proposals than the previous one. In the Introduction, we first make some remarks about the representation of situations, then give brief initial characterizations of the four components.

4.1.1. The representation of situations.

The notion 'situation' is neutral with regard to aspectual properties, subsuming both 'event' and 'state'. A situation is generally thought of as involving some kind of 'core' or 'head' on the one hand, and a certain number of arguments on the other, carrying roles such as agent, theme, location etc. With this as a very rough starting point, there are various ways of conceiving the *representation* of the situation structure of a given word. Common to all is presumably that it involves a *head*, or predicate, corresponding uniquely to the identifier of the entry in question. Apart from that, there are at least two principally different views. The first we have in mind is what we may call the *saturational* view, according to which the head is represented as having a certain syntactic valence in the semantic representation, each valence slot getting saturated by an argument, and necessarily so if wellformedness is to obtain. This view inessence subsumes the position of Montague's logical representations. The other view, which we may call the *relational* view, holds that the head as such generally lacks syntactic valence in the semantic representation, and thus constitutes a wellformed situation representation by itself. The argument roles are construed as *relations* between the situation and various entities. What corresponds to the status of an argument as valence-bound in this format is that the role-relation in question is required to obtain, by a word specific postulate. Thus, where the saturational type of representation might have a representation for a situation involving the head 'like' like (roughly)

sit =like(exp, th),

the relational type will have (roughly)

sit = like & exp-of(x,sit) & th-of(y,sit),

and, if we take 'like' to have obligatorily two semantic arguments, a semantic account of the illformedness of *John likes on the saturational approach will crucially involve a syntactically illformed representation 'like(john)', whereas on the relational approach, there will be a syntactically wellformed situation representation 'sit=like & exp-of(john,sit)', but the extra requirement '∃x[th-of(x,sit)]' defined for 'like' is not satisfied by this representation.

As this contrast is one of formal representation, it is unclear whether it necessarily couches distinct views of what situations are as such; see 2.1.3 above for remarks tying these views to types of linguistic constructions. We here adopt the relational approach, and illustrate it in more detail as we proceed, but postpone any in depth discussion of its over-all advantages. A first extension of it may be mentioned already at this stage, necessitated by some cases of *derived* templates: in such templates it may happen that operator-like items (like 'cause' in the causative version of a verb like roll) are introduced, and in such cases, the operand is entered as a complement of the head (see 4.2.2 below). This is the only circumstance where constellations resembling those used in the saturational approach will appear. A first illustration of this 'operator-operand' constellation is given in (5) below.

4.1.2. The components.

Both the LLF and the CS of a word concern the structure of the instantiator of the word. The following is the essential difference between them. The LLF of a verb represents a situation structure of the verb containing only as much as is expressed in a sentence where the verb occurs, by phonologically realized items or empty categories of the recognized types. That is, the situation structure represented in LLF corresponds to a specific syntactic frame (SAF) of the verb. The CS of a verb, in contrast, provides all roles - central as well as marginal - that the verb could conceivably be associated with, or in other words, all roles that a situation instantiating the verb could conceivably 'contain'. Their formal roles differ accordingly: LLF is a tool in the construction of a semantic representation of a sentence where the verb actually occurs; CS acts as a filter on candidate semantic representations for any sentence where the verb occurs.

The following illustrates how the CS of a verb V will be expressed, with the representation of situation structure we assume. Suppose that V can be associated with a set G of roles, G= {{agent, theme, benefactive}, {time, place, ...}}, where the first subset is the central roles, the other the marginal roles. The CS of V then has the form (1), in the terms of the 'parsing application' language; 'z' is the 'entry variable', S_1, S_2,.. are central roles, and $\{R_1, R_2,...\}$ is the complement set to G:

(1)
$inst(y,z) \rightarrow (\exists x[S_1(x,y) \ \& \ \exists x[S_2(x,y) \& ... \& \ {-}\exists x[R_1(x,y)] \ \& \ {-}\exists x[R_2(x,y)] \ \& \ ...)$

In words, if y is the situation instantiating z, then each of the roles S_1, S_2, ... must be expressed (whether phonologically or not), and no x can carry any of the roles R_1, R_2,... to y ('x' here ranges over entities of the referential world, not (just) linguistic expressions). In this way, CS provides the full specification of central (or 'semantically obligatory'), marginal ('semantically possible') and semantically impossible roles, for any word. (Syntactic obligatoriness is accounted for in SAF.)

In the data structure (LexScript) and in the data loading specification (LexEdit), the CS of a word consists simply of lists of roles, viz. the central roles and the marginal ones.

For the description of the functioning of LLF, see the next section.

SemProp is an assembly of properties, partly of the instantiator of the word in question, partly properties of the representatives of the NPs covered in the SAF of the word (the type of information called 'Selection features' in Chomsky 65). A presentation of SemProp is given in section 4.3 below. Some interconnections exist between the role a given NP can have and its possible SemProp-features: for instance, in order for an NP to have the role 'experiencer', it presumably has to have the feature '+animate'; and in order for a verb to have an instrument argument, it must have an animate agent (the role 'agent' itself being neutral - in our terminology - with regard to animacy). The latter dependency can be stated as in (2), at the parsing language level:

(2) $instrument\text{-}of(x,y) \rightarrow \exists z[agent\text{-}of(z,y) \ \& \ animate(z)]$

This is a condition which interlinks SemProp and CS. We return in section 3 to how SemProp features are organized in LexScript and in LexEdit.

SemRel is an assembly of relations in which the instantiator of a word can stand to other objects. For instance, for a word like leg, SemRel enters objects (by the words instantiated by the objects) that legs are typically *part of*, among others. Little will be said about SemRel in this section, as it seems the (grammatically) more peripheral of the components, but an outline is given in appendix 10.

4.2. LLF

The LLF of a word w represents the structure of the instantiator of w. This structure has the following aspects: 1) the nature of the instantiator (situation or thing), 2) the head of the instantiator, or its operator-operand structure, 3) the role relations, and 4) the linking of the role-bearers to arguments of the syntactic frame of the word. It is the latter which makes possible the identification of role bearers in an actual sentence. In the data structures, these aspects are organized roughly as follows:

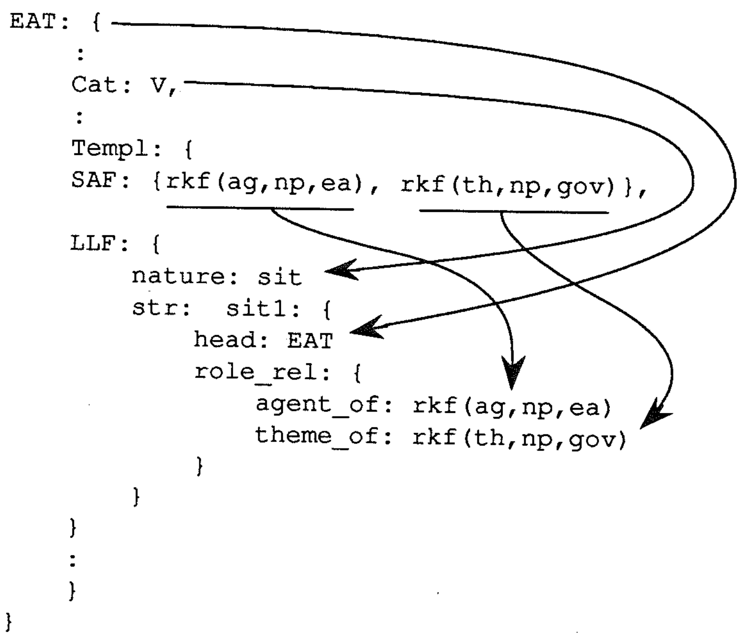
```
nature of the instantiator: (sit or thing)
structure of the instantiator: name of the instantiator:
                                    head of the inst.
                                    complement (if any)
                                    role relations
```

Illustrations follow as we proceed.

4.2.1. LLF of non-derived verbs.

In a non-derived template, the LLF is a direct projection of the SAF, the entry identifier and the category of the word taken together, by a putative algorithm we may call *LLF-transfer*. As an illustration, consider the entry for the verb eat, whose entry identifier we render as 'EAT', short for the complex number serving as the identifier. The relevant parts of the entry are as follows, with lines indicating specifications provided by LLF-transfer:

(3)



```
EAT: {
   :
   Cat: V,
   :
   Templ: {
   SAF: {rkf(ag,np,ea), rkf(th,np,gov)},

   LLF: {
       nature: sit
       str:  sit1: {
           head: EAT
           role_rel: {
               agent_of: rkf(ag,np,ea)
               theme_of: rkf(th,np,gov)
           }
       }
   }
   :
   }
}
```

The LLF specification is here interpreted as follows: The instantiator of EAT is a situation (sit_1), whose head is EAT (i.e., the identifier or an expression corresponding uniquely to the identifier) and whose role relations are 'agent_of' and 'theme_of', 'agent_of' borne to the instantiator by the representative of the NP characterized by 'rkf(ag,np,ea)' and 'theme_of' borne to the instantiator by the representative of the NP characterized by 'rkf(th,np,gov)'. The algorithm LLF-transfer functions as follows: The nature specification 'sit' (as opposed to 'thing') follows from the category 'V' (further specification of aspectual nature will have to be provided 'by hand'); the head is identical (or corresponding uniquely) to the entry identifier, and the components of the role relations are projected from the rkf-triples in SAF. The role relations are deducible from the role part of these rkf-triples ('ag' yielding 'agent_of', etc.). The name of the instantiator derives from the nature value, by subscripting '1' (the purpose of indexing is discussed in connection with (5) below, where more than one situation is involved in a structure). In a clear sense, therefore, the LLF is computable from other parts of the entry graph. As we will see below, a similar algorithm is definable for non-derived nouns.

4.2.2. LLF of derived verbs.

Turning to derived templates, we first consider verbs. In the derivation of a verbal template from a verbal template, either of two things may happen, possibly in combination:

1) an operator is introduced, taking the original situation as operand (a change which conforms with the Persistence Principle stated in (3) in 1.3);

2) the linking between roles and syntactic functions is altered.

If only the latter takes place, as in Passive or Promotion to EA, the LLF of the derived template could be constructed from the SAF of this template by LLF-transfer directly. Things however get more complicated if an operator is introduced. Often, such additions are not accompanied by the introduction of new morphemes or new head words in the syntactic structure, whereas they typically *are* accompanied by the addition of one or more new syntactic *arguments*. Given the possibility that the original argument(s) may change its/their syntactic function, there is then no guaranteed way of predicting, from the resulting SAF and the resulting operator-operand-structure alone, which of the syntactic arguments are members of which semantic role-relations. Non-ambiguity of the linking is obtained only if we have access to the previous stage in the derivation and the derivational rule itself.

To illustrate the point, consider the causativization rule 'Cause', which produces templates for sentences like John rolled the stone from templates for sentences like the stone rolled, i.e. a derivational rule for ergative verbs like roll. It turns the SAF

{rkf(th,np,gov)}

into the SAF

{rkf(ag,np,ea),rkf(th,np,gov)},

and the LLF with the components

'sit_1:head:ROLL' and 'theme_of: rkf(th,np,gov)'

into the LLF with the components (for specific comments on the status of the italicized form '*cause*', see below)

'sit_2:head:*cause*(sit_1)' and

'agent_of: rkf(ag,np,ea)', this relation being borne to sit_1, and

'theme_of:rkf(th,np,gov)', this relation being borne to sit_2.

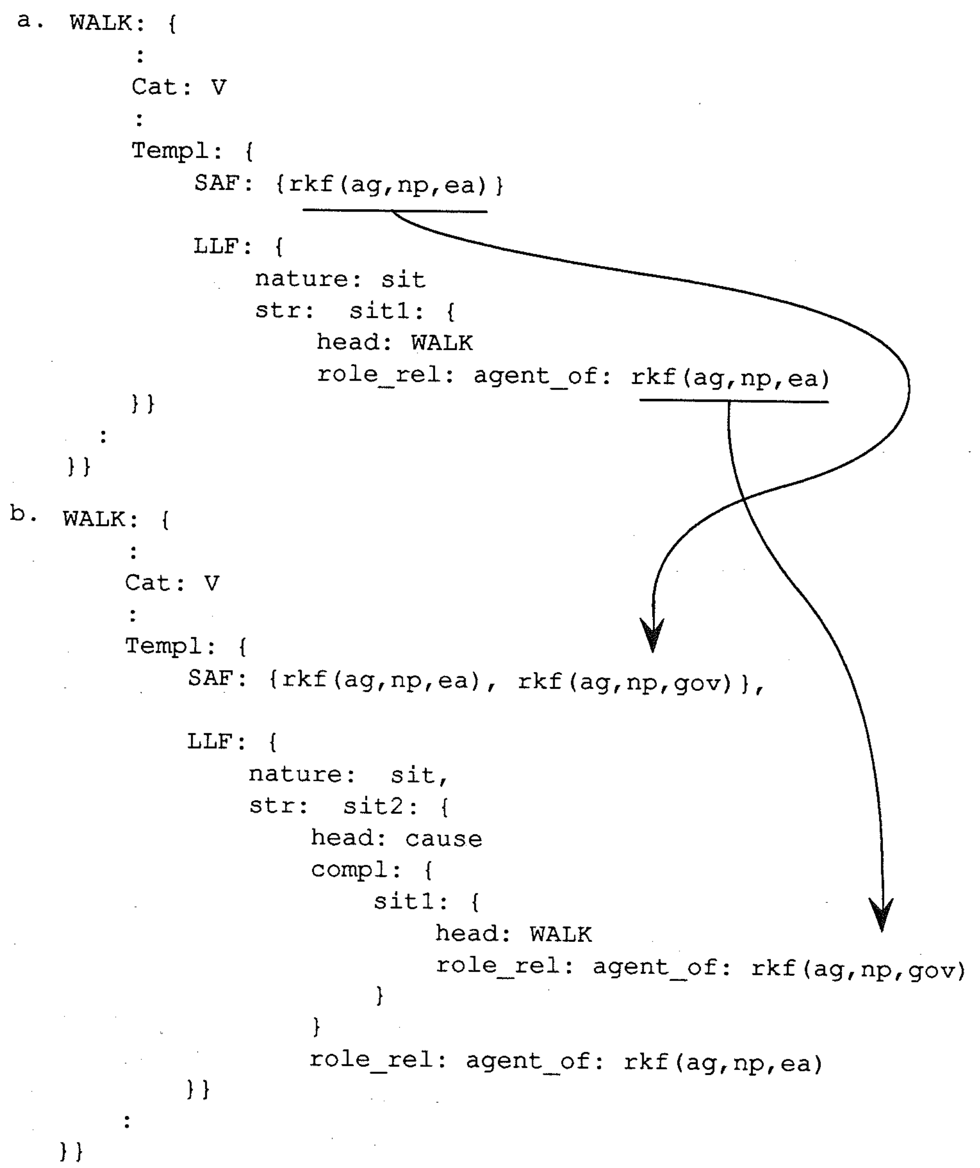
The question is how to automatically determine which of these relations involves sit_1 and which one involves sit_2. Given that links in principle *can* change, there is no guarantee that the link from the input template is preserved. This will always be the situation when a new operator-like predicate is added to LLF, without any accompanying isomorphic structure arising in the syntax: the role information provided in SAF then does not specify which element of the semantic operator-operand structure the role is related to. Hence the mechanism of LLF-transfer is insufficient for such cases. By retracing the lexical derivation, however, one would reconstruct which arguments are tied to sit_1 (i.e. the operand), and by elimination it would thus be clear which arguments link to sit_2.

The following may be one procedure for building the retracing aspect into the derivational process itself. Its main point is this: when *replacing* functional labels in SAF, then extend the replacement uniformly to LLF. Material *inserted* in SAF by the derivational process then links to material inserted by the same process in LLF. For instance, the template derivation interrelating the SAFs for the two uses of <u>walk</u> in <u>the horse walked</u> and <u>John walked the horse</u> (i.e., the causativization rule 'Ea_caus') will have the specification (4):

(4) saf: rkf(ag,np,ea) ->> rkf(ag,np,gov), 0 ->> rkf(ag,np,ea)

The template of the input will be (5a), and by extending the replacement defined in (4) from SAF to the LLF form, and linking the inserted rkf-element to the predicate inserted into LLF, we obtain the linking in (5b), as desired; the arrows indicate the replacements:

(5)



```
a. WALK: {
        :
        Cat: V
        :
        Templ: {
            SAF: {rkf(ag,np,ea)}

            LLF: {
                nature: sit
                str:  sit1: {
                    head: WALK
                    role_rel: agent_of: rkf(ag,np,ea)
        }}
      :
   }}
b. WALK: {
        :
        Cat: V
        :
        Templ: {
            SAF: {rkf(ag,np,ea), rkf(ag,np,gov)},

            LLF: {
                nature:  sit,
                str:  sit2: {
                    head: cause
                    compl: {
                        sit1: {
                            head: WALK
                            role_rel: agent_of: rkf(ag,np,gov)
                        }
                    }
                    role_rel: agent_of: rkf(ag,np,ea)
            }}
        :
   }}
```

In general, the closest dominating instantiator in the LLF-structure is interpreted as the instantiator to which the 'agent_of'-relation is borne. The exact formulation of LLF-operations is otherwise still a bit open, but the impact of the LLF counterpart to (4) is at least that the part (6a) of the input LLF be replaced by the part (6b), the remaining alterations being filled in from the rkf-operation, by the convention mentioned:

(6)	a.	sit_1	
	b.	sit_2:	head: *cause*
			compl: sit_1

The approach clearly guarantees that theta-roles and syntactic functions are paired with each other in the way desired. We note that as a consequence, the process of LLF-transfer must be restricted to base templates.

Reflection is needed also concerning the type of items here called operators, such as '*cause*'. The italicized form, as opposed to capital letters, is used to indicate that this item is not inside the template in virtue of matching a morphologically realized item in constructions accepted by the template. It might be tempting to say that it represents a 'semantic feature'; however, it is possible that such an item may in fact exert certain syntactic effects otherwise associated with the relevant morphological item - the operator *possible* in the representation of (Norwegian) -bar is a case in point, the choice of preposition (for) being the same with an adjective ending in -bar as with the adjective mulig 'possible' (see appendix 3b). This being so, it is possible that such an italicized form should really represent an entry identifier, distinguished from those marked by capital letters only in the negative manner just mentioned (viz., that this item is not inside the template in virtue of matching a morphologically realized item in constructions accepted by the template). If that turns out to be the case, we may in turn probably use the identifier directly, its lack of morphological realization being clear anyway from its absence from SAF. In the last resort, therefore, the use of italics here is just a placeholder for a question not yet resolved.

4.2.3. LLF of non-derived nouns.

Our discussion of LLF has so far been confined to templates of verbs, basic and derived. Turning next to nouns, their templates too may be basic or derived; but in addition, the instantiator in both cases can be either a thing or a situation. This section deals with basic templates. Typical LLFs will be as in (7), being part of the templates for the nouns house and event:

(7) a. LLF: nature: thing
str: $thing_1$: head: HOUSE

b. LLF: nature: sit
str: sit_1: head: EVENT

The noun event is instantiated by situations, but is otherwise quite different from nouns like run, attack, etc., which are also instantiated by situations. In LLF, the difference resides in the latter nouns having a *role-structure*, event not. (Features in SemProp might also be used to distinguish the types.) Event on its side can take complements or appositions expressing a situational content, as exemplified in the event of John coming, or Norwegian hendelsen at Jon kom. We leave this type of constructions aside for the moment.

Not only situations, but also things may have a role structure; book, for instance, may have the full LLF (8), where X and Y unify with functional labels of the types relevant with nouns:

(8)

```
LLF: {
     nature: thing
     str: {
          thing1: {
               head: BOOK
               role_rel: {
                    creator_of: X
                    matter_of: Y
               }
          }
     }
}
```

Both the choice of BOOK as head and the instantiation of X and Y will follow exactly the lines of LLF-transfer sketched above, since nouns have a SAF formally of the same type as verbs, only with partly different functional and categorial categories. The connections between SAF and LLF are shown in (9):

(9)

```
SAF: {
     rkf(matter,np,about),
     rkf(creator,np,gen),
     rkf(ident,np,ea)
}

LLF: {
     nature: thing
     str: {
          thing1: {
               head: BOOK
               role_rel: {
                    creator_of: rkf(creator,np,gen)
                    matter_of: rkf(matter,np,about)
               }
          }
     }
}
```

An element in the characterization of nouns which distinguishes them from verbs concerns the function ea: we assume that the ea of a noun is the NP dominating it, that is, the NP whose representative is the entity of which the noun may be said to be *predicated*, i.e., the property also characterizing ea's of verbs. The circumstance that this argument dominates the head of which it is an argument, sets it a bit apart from arguments in general, but not in such a way as to exclude it from being counted as part of the SAF of the head. Its theta-role, moreover, is the most degenerate conceivable, namely 'identity'. Recall that theta-roles are construed as *relations* between the instantiator of the word in question and the representative of the NP in question. In the case of the ea of a noun, these are one and the same entity, hence the only role relation obtaining is 'is identical to'. Hence we here use simply the label 'ident'. (The peculiar role status of the ea of a noun has lead some authors to assign it an undefined role 'R'; this may well be a partial anticipation of our analysis.)

Notice that an NP will be annotated in the syntax for its theta-role relative to the head of the construction; if it is also to be marked for a theta-role relative to its own head, there will seem to be a double theta-marking for all NPs. From that viewpoint, it may seem welcome if no real theta-role need to be marked for NPs relative to their heads. However, we assume that in the annotation relevant for syntactic parsing, it is marked explicitly to which head an NP bears the theta-role in question, hence there would be no inconsistencies. As a noun like attacker, as will be seen shortly, does assign a 'substantive' role to its ea, it is important that we are guaranteed against inconsistencies in this way.

As far as LLF-transfer is concerned, the marking of 'thing' under the 'nature' attribute in LLF cannot be deduced from the category 'N', as this category may also correspond to 'situation' (in the case of verbs, there is a unique correspondence). Otherwise, in a base template, the label of the instantiator itself will be the nature label indexed '1', higher indices being introduced explicitly by rules for embedding instantiators inside instantiators, analogous to the treatment of 'sit'.

4.2.4. LLF of derived nouns.

We here restrict ourselves to templates derived from verb templates. There are three interesting types: 1) those nouns whose instantiator is the same situation as the one instantiating the verbal stem, like the noun attack; 2) those nouns whose instantiator is a *thing*, carrying one of the theta-roles defined for the verbal stem, relatative to a situation of the type instantiating the verbal stem, like attacker; 3) those nouns whose instantiator is a *thing*, only morphologically related to the verbal stem, like building in the sense 'house'. We here consider the first two cases, first the one where the instantiator is a situation. An example is the template accepting the noun attack in John's attack on the fish. The input template will be essentially like (3), represented in (10) (the subscripted 'V' of the verb identifier ATTACK$_V$ serving only for the contrast with the *noun* attack, which will have 'N' correspondingly):

(10)
```
ATTACKV: {
     :
     Cat: V,
     :
     Templ: {
          SAF: {rkf(ag,np,ea), rkf(th,np,gov)}
          LLF: {
               nature: sit
               str: {
                    sit1: {
                         head: ATTACKV
                         role_rel: {
                              agent_of: rkf(ag,np,ea)
                              theme_of: rkf(th,np,gov)
                         }
                    }
          }}
     :
}}
```

The instantiator of the noun <u>attack</u> is exactly the same situation as sit_1 in (10), the only differences residing in the linking of semantic roles to syntactic functions. This is induced by the first two clauses of the rkf-derivation (11), which yields most aspects of the resulting template (12):

(11)

SAF: rkf(ag,np,ea)->> rkf(ag,np,gen),
rkf(th,np,gov)->> rkf(th,np,on),
0 ->> rkf(sit,np,ea)

(12)
```
ATTACKN: {
     :
     Cat: N
     :
     Templ: {
          SAF: {rkf(ag,np,gen), rkf(th,np,on), rkf(ident,np,ea)}
          LLF: {
               nature: sit,
               str:  {
                    sit1: {
                         head: ATTACKV
                         role_rel: {
                              agent_of: rkf(ag,np,gen),
                              theme_of: rkf(th,np,on)
                         }
                    }
          }}
     :
}}
```

The occurrence of 'sit' as nature value in the LLF is obtained since the derivational rule in question derives the LLF of the derived template simply as an identity mapping, modulo the linking between role and function induced by (11). Notice that the procedure of LLF-transfer is unwanted at this derived stage: that procedure would put the identifier of the actual word, the *noun* $ATTACK_N$, as head of sit_1, which is clearly not what we want. This point corroborates our conclusion from above in the discussion of the Caus-derivation, viz. that LLF-transfer should apply only in non-derived templates.

The second type of derived noun can be illustrated by the noun attacker, as in the attacker of the fish. Here, the instantiator has the agent role relative to $ATTACK_V$, hence the rkf-conversion is confined to (13), producing the SAF of (14):

(13) SAF: rkf(th,np,gov) ->> rkf(th,np,of)

(14)
```
ATTACKERN:    {
          :
          Cat: N
          :
          Templ: {
               SAF: {rkf(ag,np,ea), rkf(th,np,of)}
               LLF: {
                    nature: thing
                    str: {
                         abstr: ag,
                         body: {
                                sit1: {
                                       head: ATTACKV
                                       role_rel: {
                                          agent_of: rkf(ag,np,ea)
                                          theme_of: rkf(th,np,of)
                                       }
                                }
                         }
               }}
          :
}}
```

The LLF of this template is to be read as something like 'x such that x is the agent of sit_1, where sit_1 has the structure ...'. The subattributes of 'str' mark the abstraction constellation, spelled out as such in the 'parsing language'. This structure of LLF, as well as the nature attribute, are imposed directly by the derivational rule. In its operation on LLF, this rule, albeit more complex than the preceding ones, still obeys the Persistence principle, in leaving the internal structure of sit_1 intact.

In attacker, the choice of abstracted item is the agent. In attackee, it is the theme, which is equally well expressible in this formalism. Further uses may include the manner reading of John's running, with abstraction being made over a manner argument which will have to be introduced beforehand into the verbal template; likewise, length in the length of the table will involve abstraction over a degree argument associated with the adjective long. Hence this second type of derived rule has a fairly wide applicability. (For further comments on this type of nouns, see 4.3.1.1.)

4.2.5. Conclusion.

The principal issues in these remarks about LLF have been the general structure of LLF and its links to SAF, and more specifically the extent to which the LLF specification can be made to follow automatically from the SAF specification. Although the discussion has been kept only at a semi-formal level, it seems that to a large extent, such an automatization of LLF can be obtained. In basic templates, this automatization involves an algorithm *LLF-transfer*, in derived templates, it resides in the generalization of rkf-operations to comprise not only SAF but also LLF (in addition, of course, come the specific rules whose operation on LLF is the identity mapping). This automatization by no means renders LLF superfluous (e.g., lexical decomposition can only be represented there), but it gives the SAF operations the status as the motor of the lexicon, at least at the syntactic-semantic level.

4.3. SemProp

'SemProp' is a preliminary label for a potentially large set of factors which need to be represented in the Lexicon. Its use as a headline here does not imply that there is a specific component in the data structures carrying this name. The formal incorporation of these factors in the lexicon is considered as we proceed.

For any word W, there are two types of features falling within the scope of these considerations: those characterizing the *instantiator* of W, and those characterizing *arguments* defined in the SAF of W. The latter features are called *selection features* of W, the former *instantiator features* of W. There will be one set of selection features of W for each argument defined in the SAF of W, and this set of features acts as *conditions* on this argument, supplementing the conditions specified in the SAF. In the parsing of an actual sentence, each argument expression is formally marked for the same parameters of features, acceptance of the sentence requiring matching values for all the parameters for all arguments.

Two things are really in question in this section: on the one hand, a set of salient properties of arguments and instantiators which we want to capture, on the other hand, the technically optimal set of distinctive features or predicates for classifying these properties. For the purposes of this exposition, both tasks serve mainly for illustration of activities integral to the construction of the lexicon as a going concern; still, to serve as good illustrations, what we propose here should preferably qualify as possible points of departure for these activities.

In general when considering a domain of notions to be analyzed in terms of distinctive features, *binary* features are the usual preference, but the existence of fields whose arity of oppositions is greater than two is always a challenge to the construction of a uniformly binary system. What is at stake, though, is more a matter of aesthetics than of consistency. What we principally want to establish are predicates such that whenever, for the predicate P, a statement 'P(x)' is asserted, its entailments are immediately given. Minimally, '-(-P(x))' is entailed by 'P(x)', and if that exhausts the class of entailments related to the field to which P belongs, P corresponds to a binary feature. However, when the arity of oppositions is greater than two, entailment rules must be stated explicitly, to the effect that also '-R(x)', '-Q(x)', etc., are asserted, given that P, Q, R, ... are members of the same field. This is straightforward enough, but given the formal simplicity residing in the use of binary features, one naturally sets out looking for classifications done exclusively in terms of such features, abandoning them only when too many arbitrary binary predicates need to be introduced to circumvent the 'more-than-bi'-narity of a field. This is a kind of balancing act that persists throughout the construction of a classificatory system.

Apart from these considerations, a major purpose of a feature system is of course that all interesting similarities between various phenomena be represented by the occurrence of the same specified feature. This is a concern orthogonal to both the idea of just sorting a set of items by as few features as possible, and that of arriving at a binary system.

In 4.3.1, we outline the system of selection features, and in 4.3.2 we discuss aspects of the system of instantiator features. Finally, we address questions concerning redundancy of feature specification, underspecification and entailment rules (4.3.3).

4.3.1. Selection features.

As was said above, the selection features for a word W act as *conditions* on the arguments of W, supplementing the conditions specified in the SAF of W. The selection feature parameters are largely the same for all word classes. For convenience, we here exemplify mainly with verbs. The selection features fall into two main classes, in terms of the relation between the verb V and a given argument A:

i) V makes requirements concerning the *qualitative nature* of the representative of A.

ii) V makes requirements concerning *quantitative* aspects of the representative of A, more precisely, on whether the representative is a single entity or a set of entities (applicable only when the representative is countable, of course).

We first consider the most important of each of these types of features, and then discuss the formal place of representation of selection features.

4.3.1.1. The main selection features.

We first consider type i) of selectional features.

The basic classificatory concepts used so far pertaining to the qualitative nature of representatives are 'thing' and 'situation'. A further concept in this area is 'mass', a non-countable counterpart to 'thing'. Since situations must be reckoned as countable, the following binary feature classification suggests itself:

(15)	situation:	+sit	+count
	thing:	-sit	+count
	mass:	-sit	-count

Situations come in at least two varieties: events and states. The latter notions may - if there are only these two - be characterized as follows:

(16)	event:	+sit	+count	-stative
	state:	+sit	+count	+stative

Things are generally subclassified into '+animate' and '-animate', and once something is '+animate', a further distinction to be made is '+/-human'. As these notions all seem linguistically important, they are adopted here.

A further potential distinction to introduce is 'abstract' vs. 'concrete'. It is unclear, though, exactly what this distinction amounts to, and which concepts it subclassifies. Things presumably are subject to this classification, but are masses too? And are situations? Pinning down criteria clearly exceeds the scope of this note; and if criteria are not immediately clear, it is a question how fruitful the factor will be for general use anyway.

The features considered so far all pertain to representatives of NPs. Also <u>that</u>-clauses, question-clauses and infinitival clauses function as arguments of verbs. It is possible that they should be characterized as '+sit', but this depends on one's view about the 'denotation' of such constructions. Non-arbitrary features to distinguish between them are also required. It may be noted that these constructions can readily be given a *syntactic* characterization which probably is as fine as any semantic characterization. What we may want, though, are features which also capture similarities with some of the NPs in semantic respects, and not just a classification.

In a further classification of complements like that-clauses, features expressing factivity and implicativeness suggest themselves. Likewise, as the issue of *de dicto - de re* is partly lexically dependent, 'opacity' features may be considered. Such a system of features will of course not represent any analysis of these issues, only a labelling of them, neutral with regard to analysis. As usual, this is itself a balancing act, since a classification of issues is hardly independent of what theories one has about them. Still, there ought to be a compromise point where the putative feature system is both reasonably theory neutral and at the same time useful for classification purposes.

Turning then to type ii) of features, what we have in mind are features distinguishing verbs like gather, as in the men gathered or the group gathered, from verbs like run. We propose that gather in these uses has the SAF of an intransitive verb, with the subject characterized by the feature '+group'. This feature belongs to an NP if either its head noun is of the type group, assembly, herd, etc. which we call *collective nouns*, or if it has plural form. A collective head necessarily induces the feature '+group', a plural form only optionally (see (18) below).

'+group' entails '+count', both when the NP is plural with a non-collective noun, and when the head noun is collective. 'Thing' being defined (in features) as '-sit, +count', it follows that groups of things themselves are *things* in this defined sense. As a result, we will not have distinct features for, e.g., a group of men and a group of groups of men, as would be necessary if the group features were to reflect a type hierarchy. Crucially, no verbs seem to select differently among these putative type-theoretically different group-NPs.

Notice that by the entailment '+group' → '+count', also NPs with situations as their representative can have the group-feature. The relevant distinction between thing and situation is made by other features of the NP.

We may finally mention a type of feature potentially to be adopted. It pertains to logical form, more precisely, operations of the kind exemplified in (13) above, involving abstraction over arguments. A verb like exceed, for instance, may be taken to select NPs expressing *degrees* as possible subjects, as in the length of the table exceeds its height. Be elegant, correspondingly, may be said to take *manners* as possible subjects, like in John's running is elegant. As exemplified in (13), similar abstraction (we may call it *iota-abstraction*, as it combines lambda-abstraction and a definiteness operation) may take place over agent and theme arguments, but the NPs resulting behave like ordinary 'thing' arguments selectionwise. However, verbs like exceed and be elegant can also have plain 'thing' subjects (as in John is elegant and John exceeds Bill (with regard to height) - what they cannot have as subjects are clauses and NPs with situation interpretation. Hence, it seems that the 'manner' and 'degree'-NPs under consideration may well be classified simply as *things* as far as selection is concerned. This will be our assumption for the present, but further investigation may show that special degree- and manner-features are called for in selection.

4.3.1.2. Formal representation of selection features.

Consider now the formal place of selection features in the data structures we are operating with. As they pertain to properties of instantiators, they are clearly 'semantic' in nature, and as such, the most similar slot of specification so far is the 'role' slot of SAF. Let us consider the possibility of really including selectional features in SAF.

A salient feature of SAF as envisaged so far is its very simple nature - a triple of atomic predicates. In principle, this atomicity is something we should be prepared to give up - there appear to be good reasons for substituting at least thematic roles and major syntactic categories by feature combinations: for thematic roles, cf. Dowty, Engdahl and others, for major syntactic categories, cf. e.g., Chomsky, Bresnan and others (for case, see correspondingly Neidle). If so, the addition of selectional features under the role slot is not so much of a formal change - the most important difference between theta-roles and selection features is that thematic roles, or at least some of their decomposing features, are *relations* between the head word and the argument, whereas selection features are properties only of the argument. This, however, is no decisive obstacle to an incorporation of both types of features under the same slot.

With this move, we similarly open for more complex (read: feature-based) information to be given under the category and function slots in SAF. For the purpose of practical classification, it is helpful if a small set of parameters can be used for a gross first classification, and the triple so far chosen for SAF seems to serve exactly as such a partitionally effective set. At the LexEdit level, therefore, the original SAF is significant as a simple structure. Hence, at that level, the specification of further features may belong to a distinct module. At the level of the *parsing* language, there is of course no difference between the predicates introduced as values in (the classical) SAF and those introduced in the extra module, and our suggestion is that also in LexScript, the fullfledged SAF has the form of a *set* of values, one of the values being the one entered in the SAF-slot in question in LexEdit, the others being those from the additional module. For instance, the 'role' slot of the SAF defining the subject of a verb like gather may look something like the following in the LexScript data structure (see 4.3.3 on the notion 'family' and the use of '<-'):

(17)
```
Semantic properties:   {
          theta-role: agent
          family I: +count ←+animate
          family II: +count ← +group
          :
    }
```

Coming from the LexEdit SAF is 'agent', whereas the attributes 'family I' and 'family II' come from the extra module of LexEdit, this module being essentially what we have so far called 'SemProp'. That is, SemProp, at least in its selectional aspect, may be viewed as a module on its own only in LexEdit, not in LexScript.

4.3.2. Instantiator features.

In the LLF of a word, the *nature* of its instantiator is marked by a specific attribute, as seen throughout section 2. The 'nature' values used there are 'thing' and 'sit'. As these labels are also used as abbreviations for feature combinations, as suggested in (15) and (16), it is reasonable to interpret them the same way when occurring as values of the nature attribute. That is, we want to say that for any word W, its *instantiator* features are those which occur as values of the attribute *nature* in the LLF of W. Thus, just as we considered selection features as possibly belonging to the SAF specification, instantiator features belong to LLF. Both parts of what we have provisionally called 'SemProp' thereby become formally part of independently established components in LexScript.

It follows that the array of possible values under 'nature' is somewhat larger than envisaged so far, but not principally different. We now address some of these values, first for nouns, then for verbs, and then return to the formal representation of these features.

4.3.2.1. Main instantiator features.

The instantiator features of nouns are largely the same features as those which characterize NPs. In X-bar terms, this follows since NPs in general inherit their features from their head noun. In semantic terms, it follows since the instantiator of a noun is identical to the representative of the NP headed by the noun.

There is only one apparent exception to this pattern, namely the use of the feature '+group', which occurs only at NP level, reflecting either plural form or the characterization '+coll(ective)' of the head noun. In the latter case, '+group' and '+coll' amount pretty much to the same thing, so the exceptionality resides only in the plural case. Formally, these correspondences are regulated by the following entailment rules applicable at NP-level:

(18) +group → (+coll v +plur)
+coll → +group

Semantically, the discrepancy is understood as follows: in <u>five men</u>, the instantiator of <u>men</u> is a singular man on both the group and the non-group (also called distributive) reading. On the latter reading, this is also the representative of the NP. If the NP has a group reading, it is this group which is the representative of the NP. The instantiator of <u>men</u> then only provides the 'content' of the group; but this, after all, is a significant part of its characterization, and no really unmotivated violation of the general pattern of identity between N-features and NP-features.

Needless to say, a noun has both instantiator features and selection features, the latter being the characterizations of the arguments defined in the SAF of the noun.

Turning then to verbs, what act as their instantiator features are essentially what are commonly called *aspectual* features. Such features will reflect properties like 'stative','dynamic', 'punctual', 'durative', and the like. This is not the place for proposing a full set of aspectual features, so we leave that task open, noting only that - like for noun instantiator features - they will be represented under the *nature* attribute of LLF, the feature '+sit' thereby being supplemented by the aspectual features.

In the treatment of derived nouns, the Persistence principle for LLF will predict that the nature values be carried over from the verb to the noun, aspectual properties of a situation-denoting noun thus reflecting those of the verb it is derived from. It remains to be investigated to what extent such correspondences really obtain, and in case, whether they are so reliable that they can serve as basis for default mechanisms for generating lexical specifications.

4.3.2.2. Location of representation of instantiator features.

Let us now turn back to the issue of *where* the instantiator features be represented, assuming that the *nature* attribute of LLF is such a place. For nouns - as we have mentioned - the instantiator is identical to the representative of the ea, hence whatever feature characterization is supplied under LLF:nature should also appear in the semantic part of the specification of ea in SAF, given the extension of SAF proposed in 4.3.1.2. (For the purpose of a reference to be made shortly, let us refer to this duplication as *ea-duplication.*) For verbs, the instantiator is not similarly represented in SAF, hence no such duplication will obtain for verbs. For both categories, however, there is another possible duplication of the instantiator features to be considered: for various purposes, it may be desireable that the word is syntactically *annotated* for the features in question. For verbs, it may be that aspectual features play a syntactic role and hence ought to be represented in the structure assigned to a sentence in connection with parsing. For nouns, our proposed treatment of selection features presupposes that NPs are somehow annotated for selectionally relevant properties, and as these properties largely reflect properties inherent to (the instantiator of) the head noun of the NPs, it would seem reasonable that the semantic features of an NP are induced by percolation from the head. But if so, the noun has to have these features annotated on it in the first place.

For the latter case, an alternative to syntactic percolation might be sought in the *ea-duplication* mentioned above: once a noun N functions as head in an NP M, the feature characterization of M would be induced as a *selectional* requirement on the part of N. Even on such an approach, though, the phrase-head relation has to be marked in the syntax, and given general assumptions about percolation along the head projection, the features in question would eventually end up as annotated on the noun itself anyway.

On any account, then, it seems that words must be annotated for various features, identical to those occurring as value of LLF:nature. Where in our data structures would such an annotation fit in? The most plausible place is presumably 'Cat(egory)', whose atomic value posited so far will yield to sets of features. More precisely, this attribute should have various subattributes, e.g. as follows:

```
(19) Cat:    {
       major syntactic features: ...
       minor syntactic features: ...
       instantiator features: ...
     }
```

The split into major and minor syntactic features is motivated independently, but in view of that refinement, the introduction of semantic features is not as drastic an innovation as might seem first. (The situation is analogous to what we argued for the category information in SAF.) Notice that this sharing of instantiator features for a verb complies with what we suggested in section 2 about LLF-transfer, the only difference being that we may now say that the triggering of the nature value 'sit' by the major category features is something happening under the Cat attribute. In this respect, the contrast with nouns is the same ('N' not predicting whetherit should be 'thing' or 'sit').

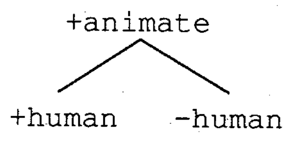
The next question is then: if the same information is to appear in three separate places in the entry for a noun, and two for a verb, is any of these places to count as *prior* in any sense? Given that this duplication is induced by value-sharing, there is no real issue here - in LexEdit, there should be a module simply called 'Instantiator features', and the specification made here is supplied automatically in all the relevant places.

4.3.3. Entailment and relevance relations, and underspecification.

In this section we reflect on questions concerning 1) entailment and relevance relations between features, and 2) the circumstance that some words may be unspecified with regard to certain properties whereas others are not.

The feature '+human' entails '+animate', whereas '-human' does not: one may well characterize e.g. a mountain as being '-human'. Still, the feature '-human' does not seem *relevant* unless the object to be characterized is +animate, and we may want to restrict its use in classification accordingly. To visualize, we may contruct 'relevance trees', where the root of each subtree limits the relevance of the daughter nodes. Thus, the features under consideration would come out in the configuration (20):

(20)



The specification of items in terms of features may vary from one item to another; for instance, whereas think requires of its agent that it be animate, destroy does not (in our sense of 'agent'). So, it will seem reasonable to leave destroy unspecified with regard to the feature '+/-animate', as far as the subject is concerned. The noun occurring in the subject of destroy may be more specific in this respect than the verb: stone, for instance, is '-animate', beaver is +animate. When words of different specificity combine in a sentence, what happens? Clearly, both the stone destroyed the car and the beaver destroyed the car are possible. One thing that we clearly want to avoid is that the absence of a requirement '+animate' on the subject in the frame of destroy should result in the feature '-animate' being interpolated and thus blocking the sentence the beaver destroyed the car. One way of avoiding this would be to - after all - give destroy many alternating complete specifications. Doing this from scratch would be cumbersome and perhaps unintuitive. A better option would be to have the lexical specification itself underspecified, as envisaged, but have an expansion procedure for underspecified matrices which would follow relevance trees. In that way, negative features would not come in in an uncontrolled manner, but only in accordance with the relevance hierarchies.

For the parsing and interpretation of the sentences mentioned, one expansion of the subject matrix for destroy would then yield '+animate' as an accepted feature, another '-animate', both sentences thus becoming accepted. In connection with the general entailment (2) mentioned above, repeated,

(2) instrument-of(x,y) $\rightarrow \exists z$[agent-of(z,y) & animate(z)]

we would in turn predict the illformedness of *the stone destroyed the car with its weight (in an instrumental sense), as opposed to the wellformedness of the beaver destroyed the car with its tail: in the former case, (2) requires the expansion where the subject of destroy has the feature '+animate', contradicting the inherent specification of stone; in the case with beaver, no contradiction arises.

It is of course essential that an entailment like '-human → +animate' is restricted to hold only for the expansion of lexical feature matrices - it is clearly not true in general.

A different type of completion rule can be illustrated with a noun like stone, marked '-animate'. No relevance tree descends from this feature, but there still should be a default value for the feature '+/-human' which will be '-human', and hence the application of an entailment '-animate → -human' (the exact opposite of the (contraposition of) the above relevance rule). Let us call this a *negative completion* rule, as opposed to the *relevance completion* rules considered first. A negative completion rule could apply either in the parsing representation or in the lexicon, in the latter case *after* the relevance completion rules.

The representation of underspecified feature-matrices in LexScript and LexEdit can now be partially envisaged as follows:
Features are clustered together in various *families* of relevance relations, where a family consists of all nodes in a tree built up of subtrees like the one in (18). Given the varying need for underspecification, and the desire to keep the dimension of specification within the relevance dimension, one way of specifying words with regard to features is to specify *paths* in relevance trees, from a shared root and as far down as the word in question extends its specification. For the subjects of the verbs think and destroy, thus, the specifications for the feature families in question could be as in (21a,b) respectively; for the instantiator specification of stone, correspondingly, the characterizing path would be (21c); 'A<-B' means 'B is a relevance daughter of A':

(21)	a. think:	+count ← +animate
	b. destroy:	
	c. stone:	+count ← -animate

In the parsing language, the line (21a) corresponds to 'count(x) & animate(x)'.

In LexEdit, the feature specification would now take the form of making choices from a menu of relevance paths like those in (21). (Suitable attribute names could be introduced for these paths.)

Each path is then automatically expanded for all available relevance paths, by relevance completion rules. Subsequently, negative completion rules apply. To illustrate, the paths in (21) are expanded to those in (22), products of negative expansion being indicated by italics:

(22) a. think:	+count ← +animate ← +human
	+count ← +animate ← -human
b. destroy:	+count ← +animate ← +human
	+count ← +animate ← -human
	+count ← -animate ← *-human*
	-count ← *-animate* ← *-human*
c. stone:	+count ← -animate ← *-human*

4.3.4. Conclusion.

The main conclusion about SemProp as a type of information is that it is expressed essentially in terms of binary features, and that it is found under various subattributes of existing components in the data structures: the selection features of a word W occur as a value of a subattribute of the *role* attribute inside each rkf-triple of the SAF of W; the instantiator features are expressed as the value of the *nature* subattribute of LLF, and also as the value of a 'semantic' subattribute of *Cat*. The exact inventory of features, even for the purposes of a first illstrative version, remains to be established, although some of the more important ones have been considered.

4.4. Summary.

This section summarizes all of the preceding proposals as far as the organization in LexScriptis concerned, and makes also proposals on how the various types of information be treated at the LexEdit level.

4.4.1. Organization of semantic information in an entry in LexScript.

(23) is a survey of the organization of the entry of an intransitive verb, with the elements of semantics discussed above indicated in italics. Shared values are indicated by boldface variables (assume that the template is a base template):

(23)
```
Id: {
     Cat: {
          Major_cat: V
          Minor_cat:
          Inst_feat: x
     }
     Segmental: ...
     Template: {
          SAF: {
               ea:  {
                    R: {
                         role: y
                         select_feat:
                    }
                    K:
                    F:
               }
          }
          LLF:  {
               nature: x
               str: {
                    z: {
                         head:
                         compl:
                         role_rel: y(ea,z)
                    }
               }
          }
          CS:  {
               centr_roles:
               marg_roles:
          }
     }
     Sem_rel:
}
```

For an ordinary, non-derived noun (like stone), the schema is as in (24), differing from (23) only in the value-sharing between the ea of SAF and *nature*, and in the lack of role relations tied to the ea (as noted in 2.3, the only relation here is the degenerate one of 'being identical to', holding between the instantiator of the noun and the representative of the ea):

(24)

```
Id: {
     Cat: {
          Major_cat: N
          Minor_cat:
          Inst_feat: x
     }
     Segmental: ...
     Template: {
          SAF: {
               ea: {
                    R: {
                         role: none
                         select_feat: x
                    }
                    K:
                    F:
               }
          }
          LLF: {
               nature: x
               str: {
                    id: {
                         head:
                         compl:
                         role_rel: none
                    }
               }
          }
          CS: {
               centr_roles:
               marg_roles:
          }
     }
     Sem_rel:
}
```

4.4.2. Semantic information in LexEdit.

In general, the only templates where semantic information is to be encoded are *basic* templates: most of this information is persistent through derivation, and whenever derivational processes do involve a change in the semantics, this is stated in the derivational rule itself.

As for the specification to be done in basic templates, we argued in section 4.2 that most aspects of LLF are deducible from information present in SAF and Cat:major_synt - from section 4.3, it is clear that the only exception is the full feature specification of LLF:nature (for verbs, the feature '+sit' follows from 'V', but not the aspectual features), i.e., the *instantiator* features. Likewise, *selectional* features hardly follow from the rkf-specifications defining basic templates. For any word W, therefore, once its basic template has been identified (in the LexEdit process), there ought to be one menu for the instantiator features for W and one selection features menu for each argument defined in the SAF. The instantiator feature menu may be tied to the path *Cat: inst_feat:*, for the selection of values, which are then shared with the other attributes receiving these values (cf. (23) and (24)). To specify the selection features, one has to go into each rkf-triple in SAF, calling the menu at the value point of *R: sel_feat:*. Since the options to be defined in the menu are partly implicatively related to the K value and R:role, it would be advantageous if these entailments could be made induce an automatic choice among possible menus.

In the design for SAF specification as conceived so far, there are two possible procedures: choice among a finite set of basic templates, and specification of a small number of parameters defined for each template, most typically allowing for choice of preposition in the F-slot of some argument. It is possible that the specification of selection features could be brought on this latter form as well, each basic template thereby having one selection feature parameter for each argument. If so, the lexicon worker will not have to represent the template as such on the screen. On the other hand, the number of parameters for each template may then grow a bit big to handle.

5. Segmental properties.

This section focuses on the information appearing under 'Segmental', in interplay with 'Cat:Minor_cat'. Like the preceding section, this section presents proposals more than decisions made and executed; in particular, phonology is so far poorly developed. We recall the over-all organization of entries of verbs, nouns and adjectives as shown in the schema (1):

(1)
```
Id: {
     Cat:      {
          Major_cat:
          Minor_cat:
          Inst_feat:
     }
     Segmental: {
          Morphon:
          Morphgraph:
     }
     Template: {
          SAF: ...
          LLF:  ...
          CS: ...
     }
     Sem_rel:
}
```

The attribute 'Segmental' (for 'Segmental properties') assembles those properties which pertain to the segmental manifestation of the entry. 'Morphon' gives morphophonological information, and 'Morphgraph' displays the ortographic counterpart of the latter. In the following, we discuss Morphon, and to some extent Morphgraph, in their interplay with Cat, in particular Cat:Minor_cat.

Typical examples of the kind of values that sort under Minor_cat are values of parameters like gender, number, person, etc. For the moment we leave open whether such property parameters are to be described in terms of binary features or something else: for short, we use the term *smallest specification unit* (SSU) for the fixation of a value within such a parameter, whatever form it may take. Thus, a specification like the standard '[3rd person, sg., past]' of an English verb may be seen as an informal instantiation of a Minor_cat specification of the form $\{SSU_{person}, SSU_{number}, SSU_{tense}\}$, where the SSU in each case fixes a value.

Given the principle that there is one entry for each type of occurrence of a word, and that entries for inflected forms are *derived* from entries for forms unspecified for the properties in question, there is in principle *one D-rule for each SSU of a word*, with one exception to be addressed shortly.

Words of a language may vary as to whether a given SSU is manifested *segmentally* or not (be this by affixation, vowel change or whatever). SSUs can thus be classified as *segmental* and *non-segmental*, and the D-rules inducing the SSUs may correspondingly be classified as segmental or non-segmental.

Moreover, within a given language, it may happen that one segmental factor (an affix, or a vowel change or whatever) encodes more than one SSU. Such a factor we call a *multiple segmental factor,* as opposed to a *singular segmental factor*, which encodes only one SSU. Rules inducing these factors may correspondingly be called 'multiple segmental D-rules' and 'singular segmental D-rules', respectively.

Segmental factors with a certain constance both with regard to *place* within words and quality will be called *morphemes*. Morphemes split into two classes (according with traditional views): *root* morphemes and *function* morphemes, the latter comprising encodings not only of 'inflection' SSUs, but also of 'derivational' (in the traditional sense) SSUs. All morphemes have an *identifier*, which is to say that also function morphemes are identified by a set of attributes, and not, e.g., by phonological form alone. In accordance with tradition, we allow a morpheme to have various segmental forms, called *allomorphs*. These are listed under 'Morphon' of the morpheme as values of the subattributes 'Alloform1', 'Alloform2', etc. A special type of allomorphs are those with zero segmental form, or manifestation through vowel change or the like. These are borderline cases of allomorphs, since they are not of a type that would count as establishing a morpheme by themselves: they count as allomorphs of morpheme M only because other segments qualify as establishing M, and once a given SSU (or assembly of SSUs) is reckoned as being realized by M, we take this SSU always to be realized by M, even when this amounts to assigning M a 'zero-allomorph'.

As alloforms we do *not* include variants determined by *phonological* processes: defining the set of alloforms of an item therefore presupposes that the range of phonological processes of the language is already defined.

To discuss the operation of D-rules on Minor_cat and Morphon, some further terminology is useful. In the operation on Morphon, let the segmental representative of the input entry be called the *host* element (H), and the morpheme added, if any, be called the *guest* element (G). (This distinction cuts across the distinction between 'head' and 'attribute': a guest morpheme representing a cross lexeme derivation is commonly counted as head of the derived word, whereas an inflectional G morpheme is not. On the other hand, 'host' and 'stem' seem to be equivalent notions.) The following main types of operation on Morphon and Minor_cat can now be distinguished. Assume an operation O, which induces the SSU (or set of SSUs, if O is a multiple segmental rule) F in Minor_cat. The result R of the part of the operation O which applies to Morphon can then be as follows:

(2)

a. R does not involve a guest morpheme. There are two subcases: 1. R equals H; that is, O has no segmental effect. 2. H undergoes a vowel change or some other (non-phonologically conditioned) effect which does not represent a morpheme by itself.

b. R does involve a guest morpheme G. This means that specified allomorphs of H and G concatenate (with whatever phonologically conditioned effects that may have). A special subcase is where the guest morpheme G has a zero or vowel- alternation alloform (cf. the remarks above): the segmental operation on H in such a case is of one of the types that arise under a, and not concatenation.

At the graphemic level, the same points apply, except that no phonological processes can be invoked to account for variation. Thereby the number of alloforms will be larger at this level. (Reductions may be possible to the extent that spelling mirrors phonological representation; the availability of such mirroring will vary from language to language.)

At the Morphon level, H and G are represented in *phonological* form. Given the Cat information going with each of them and with R, a morphologically annotated phonological representation is then constructible for R, to which phonological processes and/or principles of the language will apply. Call this representation *MorphonRep*. From MorphonRep, a *phonetic* representation of R is thereby derivable.

Presumably, MorphonRep should have the possibility of being invoked as part of Morphon itself, since certain information which refers to morphological (i.e., Minor_cat) structure is word idiosyncratic, and thus requires to be present in Morphon. Examples are likely to obtain in the area of stress placement and choice of tone in Norwegian. This means that Morphon possibly should contain not only information about H and G and a label for the type of process combining them, but also MorphonRep itself, as input to specification of stress and tone when these are not predictable. As a placeholder for this possibility, we include a subattribute 'MorphonRep' under Morphon, without going into the question of how it is to be operated on to yield the suprasegmentals of R. This subattribute will serve also in the characterization of root morphemes.

To illustrate these suggestions, we show, somewhat schematically, the derivation of the neuter sg. inflected Norwegian adjective blått 'blue' from its base form blå, where we assume the neuter sg. affix t to be a morpheme on its own, i.e., a morpheme encoding two SSUs. (3) is the entry of the base form blå, (4) the entry of the affix, (5) the entry of the derived form blått, and (6) the D-rule responsible for the change, here called 'Neuter_sg'. Some comments follow below.

a vowel change or some other (non-phonologically conditioned) effect which does not represent a morpheme by itself.

b. R does involve a guest morpheme G. This means that specified allomorphs of H and G concatenate (with whatever phonologically conditioned effects that may have). A special subcase is where the guest morpheme G has a zero or vowel- alternation alloform (cf. the remarks above): the segmental operation on H in such a case is of one of the types that arise under a, and not concatenation.

At the graphemic level, the same points apply, except that no phonological processes can be invoked to account for variation. Thereby the number of alloforms will be larger at this level. (Reductions may be possible to the extent that spelling mirrors phonological representation; the availability of such mirroring will vary from language to language.)

At the Morphon level, H and G are represented in *phonological* form. Given the Cat information going with each of them and with R, a morphologically annotated phonological representation is then constructible for R, to which phonological processes and/or principles of the language will apply. Call this representation *MorphonRep*. From MorphonRep, a *phonetic* representation of R is thereby derivable.

Presumably, MorphonRep should have the possibility of being invoked as part of Morphon itself, since certain information which refers to morphological (i.e., Minor_cat) structure is word idiosyncratic, and thus requires to be present in Morphon. Examples are likely to obtain in the area of stress placement and choice of tone in Norwegian. This means that Morphon possibly should contain not only information about H and G and a label for the type of process combining them, but also MorphonRep itself, as input to specification of stress and tone when these are not predictable. As a placeholder for this possibility, we include a subattribute 'MorphonRep' under Morphon, without going into the question of how it is to be operated on to yield the suprasegmentals of R. This subattribute will serve also in the characterization of root morphemes.

To illustrate these suggestions, we show, somewhat schematically, the derivation of the neuter sg. inflected Norwegian adjective blått 'blue' from its base form blå, where we assume the neuter sg. affix t to be a morpheme on its own, i.e., a morpheme encoding two SSUs. (3) is the entry of the base form blå, (4) the entry of the affix, (5) the entry of the derived form blått, and (6) the D-rule responsible for the change, here called 'Neuter_sg'. Some comments follow below.

(3)

```
blå318:  {
     Cat:      {
          Major_cat: A
          Minor_cat:
          Inst_feat: x
     }
     Segmental: {
          Morphon: MorphonRep: /blo:/
          Morphgraph: blå
     }
     Template: ...
     Sem_rel: ...
}
```

(4)

```
t6013:   {
     Cat:      {
          Minor_cat: neuter,sg
     }
     Segmental: {
          Morphon: MorphonRep: /t/
          Morphgraph: {
               Alloform1: t
               Alloform2: tt
          }
     }
}
```

(5)

```
blått1194:   {
     Cat:      {
          Major_cat: A
          Minor_cat: neuter,sg
          Inst_feat: x
     }
     Segmental: {
          Morphon: {
               Concat: {
                    Host: blå318:Segmental:Morphon
                    Guest: t6013:Segmental:Morphon
               }
               MorphonRep: ...
          Morphgraph: Concat: {
                    Host: blå318:Segmental:Morphgraph
                    Guest: t6013:Segmental:Morphgraph:Alloform2
          }
     }
     Template:    ...
     Sem_rel: ...
}
```

(6) Neut_sg:
- a. Cat:Minor_cat: Ø → neuter, sg.
- b. Deriv:Hist: Ø→ ++Neuter_sg
- c. Segmental:Morphon:Concat:Guest:Ø →
  +t6013:Segmental:Morphon:MorphonRep
- d. Segmental:Morphgraph:Concat:Guest: Ø →
  +t6013:Segmental:Morphgraph:Alloform1
  Condition: Host does not end in vowel or dental.
- e. Segmental:Morphgraph:Concat:Guest: Ø →
  +t6013:Segmental:Morphgraph:Alloform2
  Condition: Host ends in vowel or dental.
- f. Template: Identity

Phonologically, blått has a short vowel and geminate consonant, facts we take to be induced by phonological rules. In this respect, the graphemic form might seem derivable from the derived phonological form; however, there is no principle that spelling reflects derived phonological form. For instance, the neuter sg. form written svart 'black' ends phonologically with a (derived) retroflex, which is not reflected in spelling.

The phonological representations are given with classical phoneme notation, which is merely a matter of choice for ease of the exposition. The real choice of phonological formalism is open for the present.

For the treatment of inflection, two further steps are taken relative to what is shown above. First, classes of words can be distinguished on the basis of which *alloforms* they choose for the various inflectional rules. The most efficient way of marking choice of alloforms is therefore, in the data structure, to use alloforms as highest attribute of the rules and code classes of rules by the alloforms, codes which in turn are used to mark the class of inflectional variants chosen by the word in question.

Second, since all words of a given frame alternation family take the same inflections, except for passive forms, the most economical way of inducing the right inflections is by forming cross-products of frame alternation families and inflection paradigms.

Both of these points are reflected in appendix 4, which gives the treatment of inflection for Norwegian verbs, nouns and adjectives.

6. LexScript

6.1. Introduction

LexScript is an independent component of TROLL, both as a project and as a system (cf. 1.1). The purpose of the project is to construct a language capable of holding the information onewants to put into a linguistic electronic lexicon. The language - LexScript - will be specifically geared to its domain of application, and is not to be regarded as a general programming language for specification of general algorithms. The construction and the model of execution of the language put key importance on the way a linguist will want to describe grammatical relations.

Any programming language consists functionally of components of a certain type put together in some control structure. For standard languages like Pascal and C, the components are instructions which are performed sequentially. The programmer decides himself when and where a clause is to be executed. For languages like Prolog, the components are clauses which are tested (executed) in a control structure defined in advance. The aim of LexScript as a language can be illustrated by the following comparisons:

For the *standard languages* the domains are algorithms, and the languages are designed such that algorithms can be easily expressed (e.g., Sedgewick 1983).

For *Prolog* the domain is logical deductions, and the language is made such that one can easily express logical connections and have them evaluated. The general mechanism which is 'divided out' is the resolution mechanism, which is a general method for proving statements in 1st order predicate logic (e.g., Lloyd 1984). Resolution is what holds the components in Prolog together and gives them meaning.

For *LexScript* the domain is lexical information. The aim is to 'divide out' what is general about lexical connections, i.e., to find a general procedure by which components transfer information between each other, and use this procedure as a control structure. A central notion in this connection is *lexical derivations*. A first approach to this notion is made in the present section, especially in the parts addressing the hierarchy operator '++', viz. 6.3-6.5 below.

In addition to the functional part of LexScript comes its syntax, i.e., the way the components are written or specified. This is also a topic of the present section. It should be noted, however, that many of the symbols actually used (such as parentheses and operators) are dictated by the implementation context for LexScript, viz. Prolog.

LexScript is designed not only to describe a full lexicon, but also to take part in the development of lexica, i.e., it is a *developmental tool* as well. As such, one type of concern is its relation to the actual implementation environment - usually this environment is independent of the specification of programming languages (one exception being SmallTalk, which puts heavy restrictions on both machine and system, e.g., graphic screen and mouse control), and this holds for LexScript as well so far, even though TROLL is developed on machines with highly developed graphic interface (Apple-Macintosh).

At the content side, an important factor for the role of LexScript as a developmental tool are the possibilities of specifying choices. LexScript will contain a mechanism for specification of parametrized objects, where the parameters are chosen during the construction of a given lexicon. For instance, an object x is to be associated with a complex property P parametrized over the variable Q. The following situation is then typical: for a large part of objects with the property P, Q will have the value q, but in many cases it can take the values q_1, ..., q_n. It is a requirement on LexScript that it be possible to describe parametrized properties containing this information. For examples of an approach, see appendices 3a and 3b.

6.2. Basic structures.

A lexicon L can informally be described as a tuple <D,S,I>, where D is the description of L, S is the structures of L, and I is the linguistic interpretation of L. D, S and I are related so that S is dependent on D and I is dependent on S. That is, the level D describes the structures in S, and the elements of S determine the linguistic interpretation I.

It follows from this that the linguistic content of the lexicon is solely determined by S; the way the structures are described (=D) has no direct effect on the interpretation. However, D, being the level where the operations on L are defined and implemented, is the level of intentionality. It contains the information about how the lexicon will develop over time. If one wants to change something within S, the change is actually made to D.

The level S is a collection of attribute value graphs. Graphs together with the operation of unification are well suited for specifying a database:

- They make it possible to prepare separate parts of the database for later assembly. A graph can be ripped apart along its attributes and reassembled with unification.
- Database entries can be of varying size.

- They have a natural question-answer structure, viz. <attribute, value> ≈ <question,answer>.

How attribute value graphs are implemented and put to use in the lexicon is described in the following sections. Here we say something about the system within which the lexicon is embedded.

The lexicon is implemented in prolog in such a way that all the text specifying structures are in a prolog readable form; that is the expressions are parsed by the prolog system itself and not by some program written in prolog. Even though LexScript is supposed to be self contained with respect to operations and datatypes it is impossible (and impractical) to have a full fledged system to work with in the development of the lexicon. The embedding of LexScript in a prolog environment enables us to have parts of it fully implemented (i.e., interpreted by prolog clauses) while other parts are defined in prolog where their functionality can be tested.

In the following, keep in mind that LexScript is not fully developed, and some effects of particular mechanisms may be effects of other mechanisms in the future.

6.3. Structuring the Lexicon

The lexicon is conceptually organized around *entries, derivations,* and *other objects with their operations*. The collection of entries is relatively large in number, numbering in thousands, while the derivations are comparatively small, in the hundreds. The entries are built up using derivations and other objects.

Given the relatively large number of entries, it has been a goal to try to keep their structure simple. In the case of Troll, simplicity means that the information flow between the constituents in an entry is determined by the way they are put together. The constituents themselves have very little "knowledge" of the context they are in. That is, they have little or no oppurtunities for explicitly referring to remote places in the entry where they live, they must act on what is given to them, so to speak. Constructing the lexicon with this goal in mind creates entries which are easy to edit and maintain. The formal structure of entries is described in section 6.4 below. In the following subsections the basic constituents and mechanisms for building entries are described.

The lexicon is formally built up around the following objects:

atoms, numbers, strings, variables, and *sets* with the important special case *graphs*.

These objects are described, in turn, below, together with their associated operations and variants.

6.3.1. Atoms

Atoms are used as references and for attributes; they are specified the same way prolog atoms are, so that `an_atom` and `'ATOM'` are atoms. Typically, they will either stand for themselves or be interpeted as symbols referring to something else.

6.3.2. Numbers

Their main purpose in the lexicon is to uniquely identify entries; i.e., they serve as a kind of "social security" number for them. A number is sequence of digits ie. 10, 1001, 314 are numbers.

6.3.3. Strings

Strings are specified within double quotes so "a string" is a string. They are used to describe words, and combinations of words. Strings have certain operations defined. These include:

- *concatenation* indicated with a '+', e.g

  `"a "+"string" = "a string"`
- *subtraction* indicated with a '-', e.g.

  `"a string" - "ing" = "a str"`
- *access n^{th} character*; from end: :<[n], from start: :>[n], e.g.

  `"a string":<[3] = "i",` and

  `"a string":>[3] = "s".`
- *access substring* ranging from n'th character for m characters; from end: :<[n,m], from start: :>[n,m]. e.g.

  `"a string":<[3,2] = "ir",` and

  `"a string":>[3,2] = "st".`

These operations are defined in the lexicon but not within LexScript. The operators that encode the operations are executed using prolog clauses.

6.3.4. Variables

Variables are used for structure sharing and for passing parameters to functions and operations. Given that LexScript lives inside prolog, prolog variables are LexScript variables, at least at the present stage of development. So variables are indicated by identifiers with an initial capital letter or underscore. I.e. `T`, `_`, `Var`, `_var`, `_Var` are variables.

6.3.5. Sets and Graphs

A set is a prolog list `[E1,...,En]` such that that $E_i \neq E_j$ whenever $i \neq j$. A (feature) graph is a set of attribute value pairs `[AV1, ..., AVn]`. The AV-pairs are constructed with the prolog operator ":"; so a sample graph may look like this, `[a:b, m:[a:c, f:g], f:h]`. While the attributes are restricted to atoms, the values can be almost anything, e.g., strings, derivations, paths, operations, functions etc..

To distinguish between different uses of prolog lists, they are equipped with a type tag (or qualifier), which will tell the LexScript interpreter how to treat the list. The type tag is glued onto the list with the symbol '#', so if `a_type` is a type specifier and `G` a list, then `a_type#G` is an object of type `a_type`.

Even though graphs are a special case of sets, an untyped list will by default be treated as a feature graph, and sets are tagged with the type specifier `set`.

In association with graphs, there is the important notion of a *path*. Paths are constructed with the right associative binary operator '->', giving rise to structures like

$$p_1\text{->}p_2\text{->}\ldots\text{->}p_n,$$

where the p_i are attributes.

Graphs are evaluated on paths in the usual way by iterated application. That is, if G is a graph and $p_1\text{->}p_2\text{->}\ldots\text{->}p_n$ is a path, then G evaluated with respect to the path is

$$G(p_1\text{->}p_2\text{->}\ldots\text{->}p_n) = (\ldots((G(p_1))(p_2))\ldots)(p_n).$$

Evaluation is defined so that $G(p_i)=V_i$, if $p_i:V_i \in G$.

Paths can be used in many ways. One interesting use is to let them express shared values within graphs. However, this is probably done more easily by variables, but here is how paths have been put to that task in the lexicon. If a path P = $p_1\text{->}\ldots\text{->}p_n$ occurs inside an expression V in a graph G, e.g., as a value or as an argument to a function, P is evaluated with respect to the closest dominating subgraph D of G which has p_1 in its domain.

Example: given the following graph, with `P`, `D` and `G` indicated.

```
id := [
    a: [
        b: [d: "val"],
        a: [b: "value"],  D    G
        c : a->b
    ]
].          P
```

The reference for `P` in G is `"value"`.

This definition is not the only one. One might want to consider the whole path P, i.e., move upwards and see if there is a path matching P all the way "down", instead of just searching for a graph matching the head of P. A definition like this would make a difference if we, in the above example,replaced a->b, with a->b->c. With the former definition the result would be undefined, while with the latter the path would evaluate to "val".

6.4. The hierarchy constructor '++'

This section describes how graphs are put together in what we call a *message hierarchy*. A message hierarchy is defined with the | (bar) operation, which for implementation reasons is encoded in the lexicon as the left associative operator '++'. A different way of constructing such hierarchies is given in the section on entries. The hierarchy defines how information flows between graphs.

The one operation on graphs that makes them so useful is unification, which combines compatible pieces of information. However, it is sometimes desirable to be able to depart from this. For example, one may want to remove something from a structure, or replace it with something else. The *bar*-operation used in the coding of derivations is designed to accomplish this. It can be viewed as a binary operation on graphs which behaves exactly like unification except that it gives precedence to the right, i.e., if A is combined with B, and there are incompatibilities, the information in B "wins". The '++' is our version of a priority union.

The idea behind '++' is to view graphs as message handlers in the sense of OOP (=Object Oriented Programming). A function applied to an argument returning a value, and a message handler responding to a message is almost the same thing. The difference is that message handlers occur in hierarchies, so that if one handler is unable to respond to a message it can be passed further up in the hierarchy for processing. Constructing such a hierarchy is the task we assign to the '++'-operator. Given the way messages travel, this will implement '++' as a priority union.

Before we give the definition of how a hierarchy is interpreted, there are two extensions left to make for the graphs. The first is the addition of pairs constructed with the operator '::'; they are of the form `att::P` where `att` is an attribute and `P` a path. The intuitive idea is that a graph receiving the attribute `att` will respond with the hierarchy's value on the path `P`. The second is the introduction of the variable ←, the back arrow. The back arrow when used in a graph gets its value from the graph's predecessor in the hierarchy. How this works will become clear in the following definitions and examples.

***Definition**: Message Hierarchy (MH)*

a. Any graph is a MH.

b. If H is a MH and G is a graph then H++G is a MH.

It should be possible to view a MH as a graph, that is, if H is a MH, then H should be able to evaluate on paths as graphs do. Evaluation of MHs is defined as follows.

Definition

Let H = A++B be a MH and p an attribute. H is evaluated according to the following cases:

(1) if p::R is an element of B then
  H(p) = A(R)

(2) else, if p immediately dominates the back arrow in B, (no attribute between p and ←.)
  H(p) = B(p)[A(p)/←] (f[x/y] means bind y to x in f)

(3) else, if B(p) or A(p) is atomic, then
  H(p) = B(p)

(4) else if B is atomic
  H(p) = null (null is the universally no unifiable element)

(5) else if p $\notin$ B
  H(p) = A(p)

(6) else
  H(p) = A(p)|B(p)

This definition ensures that if A and B are unifiable graphs and there are no p::R or ← in B or A and G is the unification, then G(P)=(A++B)(P) for all paths P.

Example 1. Changing the form and category of a word:

```
A = [string:"eat", category:[main:verb, gender:masc]]
B = [string: ←+"ing", category:noun]
```

here,

```
A++B(string) = (←+"ing")["eat"/←] by (2)
             = "eat"+"ing"
             = "eating"
```

and in general,

```
A++B =[string: "eating", category:noun]
```

Example 2. Permute a graph:

```
A = [subj:"John", pred: "kill", object: "the bear"]
Permute = [subj::obj, object :: subj ]
```

Here,

```
A++Permute(subj) = A(obj) = "the bear"       by (1)
A++Permute(obj)  = A(subj) = "John"          by (1)
A++Permute(pred) = A(pred) = "kill"          by (6)
```

This section concludes with a discussion of set valued attributes, and how they fit into the scheme of things. A set value will, in the hierarchies, be treated as a kind of 'or'-graphs, i.e. the pair `attr:set#[a,b,c]` in a graph means that any of `attr:a, attr:b, attr:c` is possible. The ++ operator is extended to deal with sets simply by distributing over. That is:

```
[att:set#[a,b]]++[att:c]
```
corresponds to
```
[att:set#[a++c, b++c]]
```
and
```
[att:c]++[att:set#[a,b]]
```
corresponds to
```
[att:set#[c++a, c++b]]
```

The set valued attributes introduce an indeterminacy in the evaluation of paths. In the lexicon their use is kept to a minimum; they are used only where it seems natural to regard the value of an attribute as a set of possibilities.

6.5. The Structure of Entries

Entries are basically a collection of hierarchies; but not any collection. All the hierarchies have a common root of which they represent variations. The form of entries exploits this in the following way. All entries have an attribute BASE which encodes the basic properties of the entry; then there are a variable number of continuations (or variations) of the BASE. The continuations are attribute value pairs, where the value is classified with the type specifier *continuation*. The structure of an entry repeats itself inside each continuation. The information represented by this kind of structure is computed as follows. Let G be a graph with the structure:

```
[
 BASE: BVal,
 VARIATION: continuation # [
    BASE: SubBVal,
    variation: continuation # [
       BASE: SubSubBVal
    ]
 ],
 VARIATION2: continuation # [
    BASE: Sub2BVal
 ]
]
```

The information set expressed by G is then

`{BVal, BVal++SubBVal, BVal++SubBVal++SubSubVal, BVal++Sub2BVal,}.`

The above set is a collection of derivations.

6.6. An Example

The main purpose of the lexicon is to encode properties of words. It relates wordforms to their lexical, syntactical properties. However, this does not mean that every wordform is explicitly represented in the lexicon as such, e.g., as a list of correspondences, although the lexicon can be viewed in that way. The properties of each word are shared by other words, or are closely related to some others. How are these correspondences encoded?

There are certain problem as to what should be stated in the lexicon. Take a stem W of category C, and a derivational morpheme M mapping words of C onto category D. The question is this: should the form W#M be listed in the lexicon? The approach taken in TROLL is to list it, if W#M is an a word in common use. There are many reasons for this. First, The morph M may represent many different changes, depending on W, changes which are not predictable from the internal structure of W. Second, not all words of C can be affixed with M. Third, the lexicon will represent a complete list of the actual words that are in use in the language.

A lexical entry in TROLL encodes a set of related wordforms. The forms and their properties are related by derivational sequences. For example, the following list of words (partially representing the entry for the verb 'spise' *(eat)*), are represented within an entry:

```
spis  stem
spis#e   infinitiv
spis#er  present tense
spis#t   participle
spis#te  past tense

spis#e#s s-passive present tense
spis#te#s   s-passive past tense

spis#elig   +able (adjective)
spis#elig#het  +able+ness (noun)
```

The first group exhibits the inflectional paradigm of the verb, the second, two passive forms, and the third, two possible continuations. Each of these forms corresponds to distinct information groups as explained in section 1. The forms differ in some parts of the informational graph. The items in the inflectional group share the same category and SAF, while they have different LLFs. The items in the passive group also share between them the same SAFs, but they are different from the ones for the inflectional group, although the former is derived from latter. The last group, comprising the case of derivational morphology, have different SAFs, LLFs, and category.
The entry for 'spise' will be worked out so that one can see how this entry could be realized. The actual identifiers employed here may vary from the ones used in Troll itself. The information that goes into the entry is also simplified, in that here, only the forms and the syntactic templates are represented.
First, the entry is given a unique identifier say 102314, and the BASE information is added:

```
102314 := [
     BASE: [
          form: "spis",
          category: verb,
          SAF: transitive_verb
     ]
]

transitive_verb :=  [<ea,NP,agent>,<gov,NP,theme>]
```

This entry encodes the stem, its category and base template. The base template is typically not defined within the entry itself; there are some 20+ basic templates. The tuples `<x,y,z>` is shorthand for `x:[function:x, cat:y, role:z]`. The entry can now be expanded to accommodate the derived forms. A derived form is introduced with an attribute value pair where the value is of type *continuation*. This is to distinguish it from other pieces of information one may want to put into the entry. For the #lig type derivation, the attribute is lig_adj; so the following pair is entered at the same level as `102314 ->BASE`:

```
lig_adj : continuation # [
 BASE: [
    form: ← + "elig",
    category: adjective,
    SAF: lig1
 ]
]

lig1 derivation [...].
```

The derivation that changes the SAF is kept in a *derivation definition,* they are specified as prolog ground clauses with the binary operator `derivation;` the left argument is an identifier and the right is the actual definition used in computing the informational value.
The #het derivation can now be added to the continuation in the same way that #elig was added to the entry:

```
het_noun : continuation # [
 BASE: [
    form: ← + "het",
    category: noun,
    SAF: het1
 ]
]

het1 derivation [...].
```

The full structure of the entry for 'spise' is so far:

```
102314 := [
   BASE: [
      form: "spis",
      category: verb,
      SAF: transitive_verb
   ]
   lig_adj: continuation # [
      BASE: [
         form: ← + "elig",
         category: adjective,
         SAF: lig1
      ]
      het_noun: continuation # [
         BASE: [
            form: ← + "het",
            category: noun,
            SAF: het1
         ]
      ]
   ]
].


transitive_verb :=  [<ea,NP,agent>,<gov,NP,theme>].
lig1 derivation [...].
het1 derivation [...].
```

The entry 102314 tells us that there is a transitive verb with a stem 'spis', and that it has an adjectival form, and, based on that, a noun form. It doesn't say anything about how the verb is inflected and how the inflections relate to the syntactic properties.
Inflectional paradigms are introduced as sets. For our verb, make this definition:

```
regular_infl := set # [
  [form: ← + "e",  infl:infitival],
  [form: ← + "er",  infl:present_tense],
  [form: ← + "te",  infl:past_tense],
  [form: ← + "t",  infl:participle]
].
```

This is added to the entry at the same level as `lig_adj`, under the attribute `verbs`, i.e.:

```
verbs: continuation#[
  BASE: regular_infl
]
```

Note that the inflectional paradigm does not affect the SAF, only the shape of the word is changed. Using the rules of distribution described above, the addition of this subgraph has the effect of saying that the inflected forms of 'spis', can have the same SAF. This is a general fact about verbs, the selectional restrictions do not change if the inflection changes. Variations in the SAF are common to all the forms in the inflectional paradigm. This is accomodated by letting all the verb-related derivational sequences be continuations of the subgraph that introduces the inflectional paradigm. For example, 'spise' has in intransitive use. The detransitivaztion is effectuated by changing the gov-element in SAF into an inplicit argument, give this derivation the name `de_trans`, and add it :

```
verbs: continuation#[
  BASE: regular_infl,
  intrans: continuation#[
     BASE: de_trans
  ]
]
```

Again, the rules of distribution will link the detransitivization process to all the inflected forms of the verb.

The full structure of the entry together with derivations and symbols takes the following form.

The entry:

```
102314 := [
    BASE: [
        form: "spis",
        category: verb,
        SAF: transitive_verb
    ]
    lig_adj : continuation # [
        BASE: [
            form: ← + "elig",
            category: adjective,
            SAF: lig1
        ]
        het_noun : continuation # [
            BASE: [
                form: ← + "het",
                category: noun,
                SAF: het1
            ]
        ]
    ]
    verbs: continuation#[
        BASE: regular_infl,
        intrans: continuation#[
            BASE: de_trans
        ]
    ]
]
```

The definitions:

```
transitive_verb :=  [<ea,NP,agent>,<gov,NP,theme>]
regular_infl := set # [
    [form: ← + "e",  infl:infitival],
    [form: ← + "er",  infl:present_tense],
    [form: ← + "te",  infl:past_tense],
    [form: ← + "t",  infl:participle]
].

lig1 derivation [...definition body of lig1...].
het1 derivation [...definition body of het1...].
```

7. Final remarks.

This presentation of TROLL has had a few references to what has so far (by October 1, 1989) been done in the project. At the end, we may be slightly more explicit on this point.

1. The language which has so far been worked on is largely Norwegian, by the group in Trondheim (presently Lars Johnsen, Anneliese Pitz, Lars Hellan, and until recently, Tor Åfarli), but also on German (by Annelise Pitz) and Dutch (by Hanneke van Hoof). A French version is being initiated by Carol Neidle. The hardware available for the Trondheim group is a number of Macintoshes (including one Mac IIcx and one Mac II).

2. The emphasis has been laid mainly on principled aspects of the system: quantitative progress (in terms of entries processed etc.) obviously cannot be made until the framework has been laid down. Still, around 2000 Norwegian verbs have been partially treated for syntax, and exhaustively for inflectional paradigm.

The concentration has been mainly on verbs so far, but work on other word classes is in progress. As for the main components, syntax and morphology are those which have been developed farthest, but work on semantics, pragmatics and phonology is in good progress.

3. The project in Trondheim is financed partly by the Norwegian Research Council for Science and the Humanities (NAVF), partly by the University of Trondheim; at this point, it is clear that partial support by NAVF will continue till summer 1991. As the perspectives of TROLL go far beyond what can be accomplished within such a period, we hope to keep the project alive further: in part through renewed applications to NAVF, in part through collaboration with other groups, in part through contract research.

References

Barwise,J. and R.Cooper, 1981: "Generalized Quantifiers", Linguistics and Philosophy 5.

Chomsky,N. *Aspects of the Theory of Syntax*, MIT Press, Cambridge, Mass.

Grimshaw,J. 1988: "Adjuncts and Argument Structure", Lexicon Project Working Paper #21, Center for Cognitive Science, MIT, Cambridge, Mass.

Hellan,L., 1988: *Anaphora in Norwegian and the Theory of Grammar*, Foris Publications, Dordrecht.

Lloyd, J.W. 1984: *Foundations of Logic Programming,* Springer.

Montague,R., 1974, "The Proper Treatment of Quantification in English," in Thomason,R. (ed) *Formal Philosophy*, Yale University Press, New Haven.

Riemsdijk,H. van, and E. Williams: 'NP-structure', *The Linguistic Review* 1.

Roberts, I., 1987: The Representation of Implicit and Dethematized Subjects, Foris Publications, Dordrecht.

Sedgewick, R. 1983: *Algoritms,* Addison-Wesley.

Shieber, S. *An Introduction to Unification-Based Approaches to Grammar*, CSLI Lecture Notes, Stanford

Williams, E., 1985: "PRO and Subject of NP", Natural Language and Linguistic Theory, 3, 297-316.

Zubizarreta, M.-L., 1987: *Levels of Interpretation in the Lexicon and in the Syntax*, Foris Publications, Dordrecht.

App. 1

App. 1a.

18.10.89

PARSING LANGUAGE PREDICATES, SYNTAX AND SEMANTICS

Expression	Reading and example
Thematic roles	
ag(x,z)	x realizes the agent role associated with z (**John** *ate* the cake)
exp(x,z)	x realizes the experiencer role associated with z (John *worries* **Bill**)
th(x,z)	x realizes the theme role associated with z (John *ate* **the cake**)
ben(x,z)	x realizes the benefactive role associated with z (John *gave* **Bill** a book)
dir(x,z)	x realizes the directionality role connected with z (John *sent* the book **to me**)
loc(x,z)	x realizes the locative role associated with z (John *lives* **in London**)
target(x,z)	x realizes the target role of z (John *shot* **at the house**)
path(x,z)	x realizes the path role of z (John *went* **down the street**)
instr(x,z)	x realizes the instrument role relative to z (John *cut* the grass **with a knife**)
applic(x,z)	x realizes the 'application' role associated with z (John *used* the rope **for climbing**)
unspec_role(x,z)	x's role with regard to z is unspecified (in TROLL) (**John** is *sick*)
pred(x,z)	x is a predicate in the syntactic frame of z (John *made* Bill **sick**)

deg_pred(x,z)	x is a degenerate predicate in the syntactic frame of z (Jon *dummet* seg **ut** 'Jon made a fool of himself')
no_role(x,z)	x has no semantic role with regard to z, but possibly with regard to other elements (John *made* **Bill** sick)
no_role(x)	x has no semantic function (**it** rains)
cog_obj(x,z)	x is a 'cognate object' of z (John *died* **a dreadful death**)
inher_obj(x,z)	x is an inherent object of z (Jon *harket* **slim**)
measure(x,z)	x expresses a measure argument of ~~x~~ Z (the stone *weighs* **5kg**)
manner(x,z)	x expresses a manner role relative to z (John *lives* **well**)
repr(x,z)	x expresses the role of the represented relative to z (the picture *represents* **John**)

Semantic predicates

inst(y,z)	y is the instantiator of z
rep(x,y)	x is a representative of y
situation(y,x)	y is a situation with the internal structure x
state(y,x)	y is a state with the internal structure x
whole_part(x,z)	z's instantiator typically or necessarily forms part of a whole, and x expresses this whole (**John**'s *leg*)
rel(x,z)	z expresses a relation, and x expresses the 'relatum' (**John**'s *friend*, a *friend* of **John**)

prop_arg(x,z)	x is a propositional argument to z
den(x)	the denotation of x
part_of(u,v)	the representative of u is part of the representative of v
affected(u,v)	the representative of u is affected by the representative of v
result(u,y)	u is a result of y
possible(u,v)	v is possible for u
durative(z)	z has durative content
stative(z)	z has static content
punctual(z)	z has punctual content

Syntactic functions

ea(x,z)	x is external argument of z (**John** *read* the book)
gov(x,z)	x is governed by z (*read* **the book**, *on* **the floor**)
io(x,z)	x is indirect object of z (John *gave* **Bill** a book)
pp_arg(x,z)	x is a PP expressing a central role associated with z (John *talked* **with Mary**)
predic(x,z)	x functions as a predicative in the frame of z, either as a full predicate ('pred') or as a degenerate predicate ('degpred') (John *made* Bill **sick**)
preposed_predic(x,z)	like predic, except for occurring to the left of DO instead of to the right (Jon *malte* **rødt** huset)
impl_arg(x,z)	x is an implicit argument of z (the **x** *attack*, John was *killed* **x**)
adverbial(x,z)	x has adverbial function with regard to z (*go* **quickly**, *sing* **in Budapest**)
compl(x,z)	x is a propositional complement to z (det *virker* som **Jon er syk**)

subj_contr_compl(x,z)	x is a subject-controlled complement in the frame of z (Jon *virker* **som om han er syk**)
gen(x,z)	x functions as genitive in relation to z (**John's** *book*, *vennen* **til Jon**)
refl(x,z)	x is a 'de-argumentized' reflexive associated with the verb z (Jon *skammer* **seg**, Jon *vasker* **seg**, Jon *gikk* **seg** en tur)
incorp(x,z)	x is incorporated as sister of z in a word (stem) where z is head (**hus***bygging*)
attrib(x,z)	x is attribute of z (a **yellow** *house*)
lightv_float(x,z)	x is subject for a 'light verb' which has z as complement (**John** committed *murder*)
unspec_function(x,z)	x's function relative to z is unspecified (in TROLL)

Category predicates

max_proj(u,v)	u is the maximal projection of v
P(x)	x is a preposition
PP(x)	x is a prepositional phrase
Adv(x)	x is an adverb
AdvP(x)	x is an adverb phrase
AP(x)	x is an adjective phrase
NP(x)	x is a noun phrase
RP(x)	x is a 'referential phrase' (the carrier ofthe function 'implicit argument')
[seg]NP(x)	x is a seg-reflexive

[Adv]AdvP(x)	x is an adverb phrase consisting only of a head
[ut]Adv(x)	x is the adverb ut
[[ut]Adv]AdvP(x)	x is an adverb phrase consisting only of the head ut
[_[ut]Adv_]AdvP(x)	x is an adverb phrase with the head ut, and possibly other items
[...[ut]Adv...]AdvP(x)	x is an adverb phrase with the head ut, and necessarily also other items
[e]NP(x)	x is a trace of category noun phrase
at_clause(x)	x is an at-clause
wh_clause(x)	x is a wh-clause
å_inf(x)	x is an å-infinitive clause
bare_inf(x)	x is a 'bare' infinitive clause
å_inf_regcontr(x)	x is an å-infinitive with regular control, i.e. control by the closest NP
å_inf_markcontr(x)	x is an å-infinitive with 'marked' control', i.e. control by a remote NP
å_inf_arbcontr(x)	x is an å-infinitive with 'arbitrary' control
unspec_cat(x)	x's category is unspecified (in TROLL)
om_S(x)	x is an om-clause
som_om_S(x)	x is a som om-clause
det_there(x)	x is the expletive corresponding to there
det_it(x)	x is the expletive corresponding to it

App. 16

8.10.89 LEXSCRIPT PREDICATES, SYNTAX AND SEMANTICS

Predicate	Comment	Interpretation in the parsing level language
Role (R):		
ag		ag(x,z)
exp		exp(x,z)
th		th(x,z)
ben		ben(x,z)
dir		dir(x,z)
loc		loc(x,z)
prd		pred(x,z)
no		no_role(x,z)
unsp		unspec_role(x,z)
inherob		inher_obj(x,z)
target		target(x,z)
path		path(x,z)
instr		instr(x,z)
applic		applic(x,z)
norole		no_role(x)
cogob		cog_obj(x,z)
degprd		deg_pred(x,z)
wholepart		whole_part(x,z)
rel		rel(x,z)
prop_arg		prop_arg(x,z)
measure		measure(x,z)
manner		manner(x,z)
repr		repr(x,z)
scsu	'small clause subj', with intr. verb	u^v^[pred(w,v) & ea(u,w)] (x,z) {'w' is shared with the argument with function 'predic'}
tvscsu	'small clause subj', with trans. verb	u^v^[pred(w,v) & th(u,v) &ea(u,w)] (x,z) {'w' is shared with the argument with function 'predic'}

part	the const. is a PP whose NP denotes something which is *part* of the object's denotation ("shoot John in *his back*")	u^k^[Ew[P(w) & gov(u,w) & target(u,k) & part_of(u,y)]](x,z) {'y' is shared with DO}
whole	the const. denotes something which is a whole relative to the denotation of the succeding PP	u^k^[affected(u,k) & part_of(v,u)](x,z) {'v' is shared with the succeding PP}

Category (K):

np		NP(x)
at_S		at_clause(x)
hv_S		hv_clause(x)
å_inf		å_inf(x)
seg		[seg]NP(x)
pp		PP(x)
ap		AP(x)
unsp		unspec_cat(x)
rp		RP(x)
p		P(x)
adv		Adv(x)
advp		AdvP(x)
adv_advp		[Adv]AdvP(x)
word_ut_adv	this instantiates a format used for all adverbs, and in turn all categories; likewise for the following 3 entries	[ut]Adv(x)
word_ut_advp		[_[ut]Adv_]AdvP(x)
word_ut_advp_excl		[[ut]Adv]AdvP(x)
word_ut_advp_add		[...[ut]Adv...]AdvP(x)
e		[e]NP(x)
inf		bare_inf(x)
å_inf_regcontr		å_inf_regcontr(x)
å_inf_markcontr		å_inf_markcontr(x)
å_inf_arbcontr		å_inf_arbcontr(x)
om_S		om_S(x)

som_om_S		som_om_S(x)
det_there		det_there(x)
det_it		det_it(x)

Function (F):

ea		ea(x,z)
gov		gov(x,z)
io		io(x,z)
pgov	x is governed by a preposition	u^v^[Ey[P(y) & gov(u,y) & pp_arg(y,v)]] (x,z)
adv		adverbial(x,z)
gen		gen(x,z)
incorp		incorp(x,z)
attrib		attrib(x,z)
unsp		unspec_function(x,z)
lightv		lightv_float(x,z)
på	x is governed by the prep. <u>på</u>, which itself heads a pp_arg of z (this predicate is used only in the context <X,np,_>, where X is not prd)	u^v^[P(<u>på</u>) & gov(u,<u>på</u>) & pp_arg(<u>på</u>,v)] (x,z)
pgov2	exactly like pgov	
pgov3		
:		
på2	exactly like på	
på3		
:		
som	x is governed by <u>som</u>, and acts as a predic	u^v^[P(<u>som</u>) & gov(u,<u>som</u>) & predic(u,v)](x,z)

	(this predicate is used only in the context <prd,X,_>)	
for	x is governed by for, and acts as a predic (this predicate is used only in the context <prd,X,_>)	u^v^[P(for) & gov(u,for) & predic(u,v)](x,z)
pp_arg		pp_arg(x,z)
predic		predic(x,z)
implarg		impl_arg(x,z)
refl		refl(x,z)
preposed_predic		preposed_predic(x,z)
compl		compl(x,z)
subj_contr_compl		subj_contr_compl(x)

App. 10

SemRel

SemRel is a package of information relating to relations between instantiators of words, and is not strictly part of the grammar. The initiative to this part, as an independent enterprise, was taken by Margaret Nizhnikov and the Circle group,under the name "In Other Words" (IOW), whereas the connection to the TROLL format and the definitions have been worked out by the TROLL group. Throughout, SemRel is also referred to as 'Relations', the label used in the original IOW.

1. General remarks.

1.1. Format.
The information under Relations is given at two levels, the LexScript level (henceforth 'Script') and, as a definitional part, the level of the parsing application languageel, hencefort 'Def'. At Script level, the general format of representation for the whole package is attribute-value (AV-) graphs of the form (1),

```
(1)    entryword        {
                    Attribute 1: value
                    Attribute 2: value
                          :
                          }
```

where 'entryword' is the word for which a lexical entry is defined, and 'value' may in turn be an attribute with a value. A value is always unique, but nothing precludes it from being a *set* of entities. As 'entryword' functions an *identifier*, for present purposes written simply as a graphemic form. 'Relations' appears as one of the attributes in (1), and its value is the set of specific relations constituting Relations; schematically as in (2):

```
(2)    ID-entryword:    {
              Cat: ...
              Segmental: ...
              Template: ...
              SemRel:     {
                          Antonym: value
                          Classifier: value
                          Examples: value
                                .
                                .
```

```
        }
    }
```

What is entered as 'value' of a Relation-attribute is uniformly a *word* (or set of words), to be referred to as the *valueword* of a given relation.

1.2. Basic semantics.

The relation attributes in 'Relations' are classified according to the way entryword and valueword, either as words or via their contents, relate to each other. Some notions are crucial to making precise the possible relations in terms of content. In accordance with assumptions expressed elsewhere, we assume that a word relates to 'reality' in either of the two ways: **instantiation** and **denotation**. For a *common noun*, its *instantiator* is the (type of) *thing* (or piece of a mass) to which it correctly applies; its *denotation* is the concept it expresses, or what in extensional logic comes out as the *set* of objects of which it is true. For a *verb*, its *instantiator* is the (type of) *situation* of which it is correctly used; its *denotation* is the concept it expresses. Likewise for an *adjective*. For a *proper name*, its *denotation* is the thing it is used of; this coincides with its instantiator. The following notational conventions apply:

(3)

Expression in Def	Reading
den(x)	the denotation of x
inst(x,y)	x instantiates y

1.3. The classification of 'Relations'.

The attribute 'Relations' covers those non-grammatical properties of an entryword which somehow or other involve a *relation*, the relatum of which can be indicated by means of another *word*, the valueword. Here follows a gross classification of these relations; detailed definitions and examples follow in section 2.

A first classifying feature is whether the relation obtains between the *two words as such*, or between *what the words stand for*. Examples of the former are the compounding relation (**Compounding**) and the idiomatic meaning concatenation relation **Idiom** (i.e., that the combination entryword followed by valueword has a non-compositional meaning). Let us call such relations *Linguistic relations*, as opposed to *Non-linguistic relations*. The linguistic relations constitute GROUP G below.

Non-linguistic relations are the majority. A first classifying feature for them is whether the relation is *purely conceptual* or not. The former case comprises relations such as **Antonym, Classifier, Examples, Is, Like, Semantic Field, Synonym.** This is GROUP A. A notion common to the definitions of most of these relations is that of 'semantic field', or the derivative notion 'exemplify' . In each case, both entryword and valueword represent *concepts*.

A *not* purely conceptual relation can be either a *prototypical* relation or a *factual* relation. These options are not organizationally distinguished here, but in the formal definitions they are. At least one of the two words then represents its *instantiator*. The main types are as follows.

First, both words may represent *things*, and the relation is some prototypicality or factual relation between these things. This is GROUP B. One subgroup of this group is *constituency* relations, such as **Has, PartOf, MadeOf, Number, Organization** (GROUP Ba). Another (Bb) is relations which represent some kind of *cooccurrence*, such as **Cooccurrence, Manifestation, Sign, Snowball** and **Symbol.** A third, less homogeneous subgroup (Bc) comprises the **SourceOf** relation, the relations listed under **Prepositions,** and **Unit.**

Secondly, entryword represents its instantiating *thing*, and valueword a prototypical *property* of that thing. Examples are **Adj, Color, Shape, Texture.** This is GROUP C.

Third, entryword represents its instantiating *thing*, valueword expresses a *situation* in which this thing plays a certain role, specified by the attribute. The main cases are **Action, How, Instrument, Intensifier, Purpose 1, Result 1, Where**, together constituting GROUP Da. In addition come **Field** and **When**, GROUP Db, where valueword expresses a 'field' or type of situations.

Fourth, both words represent their instantiating *situations.*, and the attribute specifies a relation between the situations. Examples include **Purpose 2** and **Result 2.** This is GROUP E.

A further group - F - is constituted by relations belonging to a higher order level, making comparison between relations or properties (**Compare, Homologous, Metaphor**; GROUP Fa), or ascribing numbers to them (**Cardinality, Ordinality**; GROUP Fb).

2. The Relations attributes.

Below, we go through the Relations attributes according to the grouping just suggested. For each relation is provided (a) a short verbal definition; (b) examples, in the form of an entryword paired with the valueword picked out by the relation in question; (c) a formal representation of one or more of the examples, phrased in terms of the Def-notation and concepts, including those in (3). In these formal definitions, it may be noted that when a word is entered with *underline*, that expression is short for an *identifier* of the word intended (the spelled representation of which is the spelling used); this represents a *mention* of the word, as opposed to a *use* of the word when no underlining occurs. For convenience, we refer to the thing instantiating the entryword as *entrything*, and to the thing instantiating the valueword as *valuething* (similarly for situations, when that is relevant); in the same vein, the concept expressed by entryword and valueword is called *entryconcept* and *valueconcept*, respectively. Further comments are made in connection with the particular attributes, and the first attribute is given a more complete explanation than the later ones.

2.1. GROUP A.

This group consists of relations which crucially relate to the *semantic fields* of the concepts expressed by entryword and valueword.

Antonym 1

Examples:

boy:girl
right:left

Antonym 2

Example:

tall:short

In both cases of antonymy, the notion of *semantic field* is crucially involved. In the case 'boy-girl', the field is *binary*, i.e., it has two members only. (This holds even for 'right/left', namely the 'field' of directions relative to a front.) In the case 'tall-short' the field has many members, namely the values along the height dimension (whether the values are *discrete* or not does not seem to affect the present point), and the antonyms are the extreme values at each end. The Def definition (4) represents the first case, (5) the second:

(4) EP[exemplify(den(<u>boy</u>),P) & exemplify(den(<u>girl</u>),P) &
K{x|exemplify(x,P)}=2]

(5) ES,x,y[scale(S) & extremes-of(x,y,S) & x=den(<u>tall</u>) & y=den(<u>short</u>)]

('E' is the existential quantifier, 'K{...}' means 'cardinality of'. 'P' and 'S' are the semantic fields involved. 'exemplify(x,y)' is the relation between the concept x and the semantic field y to which it belongs.)

To be precise, let us point out how Script- and Def-format interact. The Script representation of the pair 'boy:girl' is (6):

```
(6)       boy       {
               Attribute x: ...
               Attribute y: ...
                    .
                    .
               Relations:   {
                                  .
                                  .
                            Antonym 1: girl
                                  .
                                  .
                                  }
                    }
```

The particular path from entryword via Relations and Antonym 1 to valueword has the more general representation (7), where '^' is the lambda-operator; the first argument to this function is the entryword, the second the valueword:

(7) ^z^yEP[exemplify(den(z),P) & exemplify(den(y),P) &
K{x|exemplify(x,P)}=2]

What has now been illustrated is the general procedure for linking Script-representations in Relations to the Def format: for each pair entryword:valueword, the path of which they are the extremes has a general translation in Def as a function involving lambda-operators, and entryword and valueword serve as arguments of this function. (There are only three exceptions throughout, under the attributes **Part of** (2.2.1), **Preposition** (2.2.3) and **Action** (2.4.1).) In the following, this function is never entered by itself, only after it has been applied to one of the examples of entryword-valueword.

Classifier

Entrything has a property which exemplifies valueconcept (in MN's words: "the type of entity that something comes in").

Examples:

car: make
man: race

Def representation:

(8) inst(x, man) -> EP[P(x) & exemplify(P, den(race))]

In words: if x instantiates man, then x has some property P which exemplifies the property denoted by race. So, P can be 'white', 'black', etc.. The Def predicate 'exemplify' is the hyponym relation (see **Examples**).

Examples

Valueword is a hyponym of entryword.

Examples:

race: black, white, oriental

Def representation:

(9) exemplify(den(black), den(race))

'Exemplify' is the Def predicate for the hyponym relation.

Is

Entryword is a hyponym to valueword.

Examples:

arm: limb

student: person

Def representation:

(10) exemplify(den(arm), den(limb))

Like

Entryconcept and valueconcept belong to the same semantic field, and are *close* to each other inside that field.

Examples:

white: light, beige

tree: bush

Def representation:

(11) EP[exemplify(den(tree), P) & exemplify(den(bush), P) & close(den(tree), den(bush))]

In words: there is a property, or 'semantic field' P, which is exemplified both by 'tree' and 'bush', and these are close to each other within this semantic field.

Semantic field

Entryconcept and valueconcept belong to the same semantic field.

Examples:

white: black, oriental, ...
north: east,south,west

Def representation:

(12) same-sem-field(den(north),den(east))

(12) is equivalent to (13), which is to say that 'same-sem-field' can be defined in terms of the relation 'exemplify':

(13) EP[exemplify(den(north),P) & exemplify(den(east),P)]

Synonym:

Example:

arm: weapon

Def representation:

(14) same(den(arm),(den(weapon))

2.2. GROUP B.
This group consists of relations between *things* instantiating entryword and valueword.

2.2.1. Subgroup Ba. Constituency relations.

Has

Valuething is an essential constituent of entrything, the Def predicate for this relation being 'part-of'.

Examples:

car: wheels, steering wheel
body: arms, legs, head

Def representation:

(15) inst(x,arm) -> Ey[inst(y,body) & part-of(x,y)]

(See also **Part of.**)

Made of

Valuething is the substance of which entrything is prototypically made. The Def predicate for this relation is 'made-of'.

Example:

fork: silver

Def representation:

(16) inst(x,<u>fork</u>) ->> Ey[inst(y,<u>silver</u>) & made-of(x,y)]

Organization

Valuething is the type of organizational unit to which entrything prototypically belongs.

Examples:

student: school, university
book: library, bookstore

Def representation:

(17) inst(x,<u>student</u>) ->> Ey[inst(y,<u>university</u>) & belong-to(x,y)]

Part of

Entrything is an essential part of valuething. The preposition indicates how the two are locatively related, reflecting the linguistic usage.

Examples:

finger: hand(on)
Paris: France(in)

Def representation:

(18) inst(x,<u>finger</u>) ->> Ey[inst(y,<u>hand</u>) & part-of(x,y) & on(x,y)]

Unlike the schema illustrated in (6) and (7), the present cases involve three parameters rather than two, viz. entryword, the preposition entered in parenthesis, and valueword; a counterpart to (7) with three lambda-operators rather than two will then be used for translation from Script to Def.

(It may be noted that as a piece of subcategorization specification, the information that <u>on</u> is the preposition used with <u>finger</u> relative to what it is part of, appears in the SAF attribute in LexScript, describing the realization of the 'part of' role associated with finger. This information would then not need to be replicated under SemRel.)

2.2.2. Subgroup Bb. Cooccurrence relations of various types.

Cooccurrences

Entrything and valuething typically cooccur.

Examples:

magnet: iron

rain: wind

Def representation:

(19) inst(x,magnet) $->>_W$ Ey,z[inst(y,iron) & sit(z) & cooccur-in(x,y,z)])

This reads: if x is a magnet, then there is prototypically (in a weak sense) a situation z such that x cooccurs with (some amount of) iron in z.

Manifestation

Valuething is a manifestation of entrything.

Example:

water: snow, steam, ice

Def representation:

(20) manifests(den(snow), den(water))

(Counting this as a 'cooccurrence' may be stretching the notion a little - it is certainly the most borderline case. The alternative of grouping it under Ba, as constituency, is not to the point either, however.)

Sign

Entrything is a natural sign of valuething.

Example:

smoke:fire

Def representation:

(21) to be filled in

Snowball

Entrything comes standardly with or without valuething.

("Snowball" alludes to the circumstance of a snowball rolling, with more material adjoined to it as it rolls on, while it is the same snowball all the time.)

Examples:

paper: lines, margins
hamburger: ketchup, relish, mustard

Def representation:

(22) inst(x,<u>paper</u>) ->>$_W$ inst(y,<u>margin</u>) & adjoined-to(y,x)

One may note the contrast to **Part of**, which covers essential parts of a thing; those here in question may be called non-essential.

Symbol

Entrything is culturally a symbol of valuething.

Example:

13: bad luck

Def representation:

(23) to be filled in

2.2.3. Subgroup Bc. Miscellaneous.

Prepositions

For each preposition P, entrything has the relation expressed by P to valuething - either prototypically or factually.

Example:

finger: {on: hand}
epilog: {after: story}
: {in: book}
Jesus: {from: Nasareth}

Def representations:

(24)a. inst(x,<u>finger</u>) ->> Ey[inst(y,<u>hand</u>) & on(x,y)]

b. from(den(<u>Jesus</u>),den(<u>Nasareth</u>))

In general, this relation is relevant only for *nouns*, and only when the preposition has a clear locative or temporal sense, thus excluding cases like on in an attack on Peter, or of in the destruction of Rome. (Other relations providing 'prepositional' relations include **Snowball**, **Organization** and **Part of**; see especially comments to the latter.) Although these relations are 'real world' relations and not idiosyncratically related to specific words, no independent characterizations are so far available for these relations, and in the absence of such characterizations, it seems safest to use the prepositions of the language to which the nouns to be classified belong.

What induces the form in (24b) ought to be the status of the words as proper names, a feature formally marked elsewhere in the lexicon. Unlike the schema illustrated in (6) and (7), the present cases involve three parameters rather than two, viz. entryword, the specific preposition and valueword; a counterpart to (7) with three lambda-operators rather than two will then be used for translation from Script to Def.

Source of

Entrything is the place, person, organization etc. where valuething was first created or came into existence.

Examples:

Italy: the mafia, chianti, fascism

Def representation:

(25) source-of(den(Italy),den(chianti))

Unit

Entrything standardly appears in the type of package or container expressed by valueword.

Example:

beer:pitcher

Def representation:

(26) measured-in(den(beer), den(pitcher))

2.3. GROUP C.

Valuewords in this group represent what may rather loosely be called 'properties' of the instantiator of the entryword.

Adjective

Valueword is an adjective and expresses a prototypical property of entrything.

Example:

knife: sharp

Def representation:

(27) inst(x,knife) ->> den(sharp)(x)

Color

Valueword expresses the prototypical color of entrything.

Example:

grass: green

Def representation:

(28) inst(x, grass) ->> den(green)(x)

This is a statement about any particular amount of grass.

Texture

Valueword expresses the prototypical texture of entrything.

Example:

fur: fuzz

Def representation:

(29) inst(x,fur) ->> has-the-texture-of(x, den(fuzz))

This entry is a bit similar to **MadeOf** in subgroup Ba, the group for *constituency*. Texture is still not constituency.

2.4. GROUP D.

This group contains relations between things and situations.

2.4.1. Subgroup Da. Relations between role-bearers and individual situations.

Action

Entrything prototypically plays a role x with regard to (action) type of situations instantiating valueword.

Examples (x indicated in parenthesis):

wind: blow (inher)
hose: spray (instr)
author: write (agent)

Def representation:

(30) inst(x,hose) ->> Ey[inst(y,spray) & Instr-of(x,y)]

This reads: if x instantiates hose, then prototypically there is an instantiation y of spray which is a situation, and x has the instrument-of relation to this situation (which is tantamount to saying that x has the instrument 'role' *in* that situation).

Entryword is always a noun, valueword a verb. Like in the case of **Part of** and **Preposition**, there are three parameters to be filled into the general form, in this case those italicized in the path schema (31):

```
(31) entryword {
                    .
                    .
               SemRel      {
                                .
                                .
                           Action: valueword (role)
                                .
                                .
                                }
                    }
```

This is thus one more exception to the general pattern illustrated in (6) and (7). The function in the present case looks as in (32),

(32) ^z^u^v[inst(x,z) ->> Ey[inst(y,u) & v(x,y)]]

with the parameters marked in (31) to be selected from left to right.

How

Valueword expresses the manner in which the action instantiating entryword is done.

Example:
elbow: forcefully

Def representation:

(33) inst(y,elbow) ->> Ex[inst(x,forcefully) & Manner-of(x,y)]

It is not quite clear what kind of entity should be taken as instantiator of a word like forcefully; all we have said here is that it can bear the manner relation to situations. Notice that unlike **Action**, the name of the relation is here used as attribute name, thus only two parameters are filled in in this case. The same holds of the other attributes in this subgroup. If economy in the Script-format were a goal, it would have sufficed with either the latter strategy *or* that of the **Action** strategy exclusively; in the Def representation, there is no difference. A possible reason for having the redundancy may be that it maximizes completeness in the data loading.

Instrument

Valuething prototypically serves as instrument of the action instantiating entryword.

Example:
write: pen

Def representation:

(34) inst(y,write) ->> Ex[inst(x,pen) & Instr-of(x,y)]

This is the same relation as the instrumental relation under **Action**, only accessed in the opposite direction in Script.

Intensifier

Valueword expresses the manner in which the action instantiating entryword is done, where the manner is some kind of intensifier.

Examples:
hurt: deeply
hit: hard
run: fast

Both in Script and Def notation, this is the same as **How**.

Purpose 1.

Entrything has been made with the purpose that it serve as instrument for the action expressed by valueword.

Example:
bed: sleep

(35) inst(x,bed) -> purpose-of(x, Ey[inst(y,sleep) & instr-of(x,y)])

This is very close to the instrumental relation under **Action**. It is possible that the role interpretation could be wider, the crucial part being 'purpose of'.

Result

Valuething belongs to that subset of instruments which are called 'intermediary agents' (Marantz 1984) with regard to the action or action-type which is expressed by entryword.

Example:

music: instrument

Def representation (without specific marking of intermediary agenthood):

(36) inst(x,music) ->> Ez[inst(z,instrument) & Instr-of(z,x)]

Where

Entrything is a typical location for the action expressed by valueword.

Example:

armpit: shave

Def representation:

(37) inst(x,armpit) ->> Ey[inst(y,shave) & location-of(x,y)]

2.4.2. Subgroup Db. Relations between things and 'fields' of situations.

Field 1

Entrything belongs in the field expressed by valueword.
Example: carpentry

Def representation:

(38) belongs-in-the-field(den(hammer), den(carpentry))

Field 2

Entryword belongs in the field expressed by valueword.

abrakadabra: magic

Def representation:

(39) belongs-in-the-field(abrakadabra, den(magic))

When

The situation type expressed by entryword takes place at the time expressed by valueword.

Example:

harvest:fall

(40) time-of(den(harvest), den(fall))

2.5. GROUP E.

In this group, both entryword and valueword are instantiated by situations, called 'entrysituation' and 'valuesituation', respectively.

Purpose 2

Entrysituation has as its purpose to bring about valuesituation.

Examples:

wash: clean
cook: eat

Def representation:

(41) inst(x,wash) ->> Ey[inst(y,clean) & purpose-of(x,y)]

It is possible that in the interesting cases of this type, one and the same entity plays a role in both situations; thus, a pair like "kill:afraid" (x kills y in order to make z afraid, the terrorist strategy) does not come too naturally, as opposed to "kill:scare" (x kills y in order that x scares z). If so, (41) might be expanded to (42):

(42) inst(x,wash) ->> Ey[inst(y,clean) & purpose-of(x,y) &
Ez[plays-a-role-in(z,x) & plays-a-role-in(z,y)]]

A question by itself is what the criterion for 'being more interesting' is in this case, and by what criteria we decide this. The set of relations constructed ihere is supposed to be those which are 'natural' to the human conception. Moreover, the example pairs used throughout are all easily obtained through plain association. Should we then suppose that ease of association reflects what are the more 'natural' relations, or does this only serve as a preliminary criterion?

2.6. GROUP F.

The relations in this group may all be said to be higher order, obtaining between relations or between properties.

2.6.1. Subgroup Fa. Non-numerical higher order relations.

Compare

to be filled in

Homologous.

Entrything bears the same relation to some entity z as valuething does to some entity w.

Example:

elbow: knee

Def representation:

(44) inst(x,<u>elbow</u>) & inst(y,<u>knee</u>) ->> ER,z,w[R(x,z) & R(y,w)]

Metaphor:

to be filled in

2.6.2. Subgroup Fb. Numerical higher order relations.

Cardinality

Valueword expresses the cardinality of those assemblies in which entrything typically occurs.

Examples:

finger:	2x5
states:	50
lips:	2

Def representation:

(46) inst(x,<u>lip</u>) ->> Ey[part-of(x,y) & K{z| lip(z) & part-of(z,y)}=2]

'K{z|...}' means 'cardinality of the set of z such that z ...'.

For <u>finger</u>, this format yields '5'; to get '10', we need access to the whole of which the wholes relative to fingers - i.e., hands - are part. To get this number as '2x5', we also must keep track of the number of hands. We will not enter into these complexities here (which are not confined to <u>fingers</u>; <u>toes</u> too has '2x5). Other problems arise when the 'part-of' relation is irrelevant: thus <u>soccer</u> has '11' or '2x11', without there being any part of a soccer team which instantiates the noun soccer. It may be necessary to specify various subrelations here.

Ordinality

Valueword expresses the ordinality of entrything with regard to that ordered set with which entrything is most naturally associated.

Example:

December: 12

2.7. GROUP G.

This group comprises those relations obtaining between linguistic items which are not part of grammar proper.

Compounds (preliminary remarks)

The relation of *word compounding* is of particular interest in languages like Norwegian and German. The constituents of a compound word by themselves form a linguistic relation, which however expresses some real-world relation. Which one differs in unpredictable ways from case to case, but it is usually one which points to some interesting properties of (the meaning of) the constituents involved.

In practice, each word will have two entries with regard to compounding - one for those compounds where it is the *head*, and one for those where it is the non-head, in LexScript called *Incorp(orated)*. They may be called **CompoundStem** and **CompoundIncorp**, respectively. In each case, we list all those compounds where the word occurs, and specify, for each compound, the meaning of that compound. (In some cases this meaning is predictable: when the compound performs a type of *saturation* specified in SAF, and those cases where the world naturally allows for a certain interpretation.)

Example (Norwegian):

pels

CompoundStem of: skinnpels, revepels

CompoundIncorp of: pelstyv

Def representation:

to be filled in

Idiom

to be filled in

2.8. GROUP H. MISCELLANEOUS.

This group is for relations which do not fit any of the above groups, and do not yield any natural new group.

Appendix

The following attributes may belong to Semantics of TROLL, as a special type of selection restrictions:

Limited object

Entryword is a verb whose theme-argument can denote only entities instantiating words from the value-set.

Example:

brew: {coffee,tea}

Def representation:

(1) inst(x,brew) -> Ay[theme-of(y,x) <-> (inst(y,coffee) v inst(y,tea))]

By phrasing this in terms of 'theme' rather than 'object', we immediately cover e.g. passive constructions with the verb.

Limited subject:

Entryword is a verb whose agent-argument can denote only entities instantiating words from the value-set.

Example:

roar:lion

Def representation:

(2) inst(x,roar) -> Ay[agent-of(y,x) <-> inst(y,lion)]

'Ay' means 'for all y'. Instead of 'subject', as occurs in the attribute name, the definition uses 'agent'; this immediately covers constructions such as "There roared a lion" (which is good in Norwegian). Notice that this representation allows any *expression* to carry the role in question, as long as the entity denoted is one which instantiates lion; thus, also "the king of the animals" is predicted to be able to cooccur with roar, as is correct. Correspondingly in the case of 'limited objects' above.

App. 2

10.89

Basic templates

(1) **intransitive verb**
SAF: <ag.np,ea>
Statement: **iv**
[Ex. Jon hopper]

(2) **ergative**
SAF: <th,np,gov>
Statement: **erg**
[Ex. _ ruller stenen]

(3) **experiencer intransitive verb**
SAF: <exp,np,ea>
Statement: **exp_iv**
[Ex. Jon fryser]

(4) **transitive verb**
SAF: <ag,np,ea>,<th,np,gov>
Statement: **tv**
[Ex. Jon sparker ballen]

(5) **theme transitive verb**
SAF: <th,np,ea>,<th,np,gov>
Statement: **th_tv**
[Ex. papiret absorberer vannet]

(6) **experiencer transitive verb**
SAF: <exp,np,ea>,<th,np,gov>
Statement: **exp_tv**
[Ex. Jon liker gåselever]

(7) **ditransitive verb**
SAF: <ag,np,ea>,<ben,np,io>,<th,np,gov>
Statement: **ditv**
[Ex. Jon gir Per suppe]

(8) **reflexive verb**
SAF: <exp,np,ea>,<norole,seg,refl>
Statement: **refl**
[Ex. Jon skammer seg]

(9) **psych verb**
SAF: <exp,np,io>,<th,np,gov>
Statement: **psych**
[Ex. _ irriterer Jon katten]

(10) **raising verb 1**
SAF: <norole,seg,refl>,<th,at_S,gov>
Statement: **raisv1**
[Ex. _ viser seg at Jon er syk]

(11) **raising verb 2**
SAF: <prop_arg, cat, compl>
Statement: **raisv2**
[Ex. _ virker som om Per er syk]

(12) **raising verb 3**
SAF: <exp,np,io>,<th,at_S,gov>
Statement: **raisv3**
[Ex. _ synes meg at Per er syk]

(13) **raising verb 4**
SAF: <th,at_S,*function*>
Statement: **raisv4**(arg)
where arg is gov (default), pgov or a preposition
[Ex. _ later til at Jon er syk]

(14) **depictive**
SAF: <ag,np,ea>,<norole,seg,refl>,<th,np,gov>
Statement: **depict**
[Ex. Jon forestiller seg Per]

(15) **small clause**
SAF: <ag,np,ea>,<scsu,np,gov>,<prd,ap,som>
Statement: **smcl**
[Ex. Jon anser Per som syk]

(16) **intransitive verb locative**
SAF: <*X*,np,ea>,<loc,*Y*,adv>
Statement: **iv_loc**(arg1,arg2)
where arg1 is ag (default) or th, and arg2 is pp (default) or advp
[Ex. Jon bor i Guatemala]

(17) **reflexive verb locative**
SAF: <*X*,np,ea>,<norole,seg,refl>,<loc,*Y*,adv>
Statement: **refl_loc**(arg1,arg2)
where arg1 is ag (default) or th, and arg2 is pp (default) or advp
[Ex. Jon oppholder seg i Guatemala]

(18) **transitive verb locative**
SAF: <ag,np,ea>,<th,np,gov>,<loc,np,*function*>
Statement: **tv_loc**(arg)
where arg is pgov (default) or a preposition
[Ex. Jon setter vasen på bordet]

(19) **indirect argument**
SAF: <ag,np,ea>,<th,np,*function*>
Statement: **ind_arg**(arg)
where arg is pgov (default) or a preposition
[Ex. Jon stoler på Per]

(20 **indirect argument reflexive**
SAF: <ag,np,ea>,<norole,seg,refl>,<th,np,*function*>
Statement: **ind_arg_refl**(arg)
where arg is pgov (default) or a preposition
[Ex. Jon forvisser seg om utfallet]

(21) **weather verb**
SAF: Ø
Statement: **weather**
[Ex. _snør]

(22) **measure verb**
SAF: <th,np,ea>,<measure,np,predic>
Statement: **measure**
[Ex. stenen veier 3 kg]

(23) **representative**
SAF: <th,np,ea>,<repr,np,gov>
Statement: **repr**
[Ex. bildet forestiller Jon]

(24) **appellative 1**
SAF: <ag,np,ea>,<scsu,np,gov>,<prd,np,*function*>
Statement: **appellat1**(arg)
where arg is pgov (default) or a preposition
[Ex. Jon utnevner Per til konge]

(25) **appellative 2**
SAF: <ag, np,ea>, <scsu, np,gov>,<prd,*Y*,predic>
Statement: **appellat2**(arg)
where arg is ap (default) or np
[Ex. Jon kaller Per Y]

(26) **reflexive verb manner**
SAF: <*X*,np,ea>,<norole,seg,refl>,<manner,*Y*,adv>
Statement: **refl_manner**(arg1,arg2)
where arg1 is ag (default) or th, and arg2 is advp (default) or pp
[Ex. Ola oppførte seg bra]

(27) **ergative ditransitive**
SAF: <ben,np,io>,<th,np,gov>
Statement: **erg_ditv**
[Ex. _ venter Jon en ulykke]

Arg uniformly replaces an italicized item. When there are two arg's (entred '(arg1,arg2)'), arg1 replaces the leftmost italicized item in the SAF string.

App. 3a

APP. 3

```
/* derivations */

'DO_del' derivation [
    saf : rkf(X,np,gov) ->> rkf(X,rp,implarg),
    cond : ~ rkf(X,np,io),
    llf:nature: durative,
    ex:"spise fisken"->>"spise"
].


'DO_to_PP'('Prep') derivation [
    saf : rkf(X,np,gov) ->> rkf(X,np,'Prep'),
    cond :[
        ~ rkf(X,np,io),
        default('Prep',pgov),
        range('Prep',[pgov,all_prepositions])
    ],
    llf: nature: durative,
    ex: "spise fisken"->>"spise på fisken"
].

'IO_del' derivation [
    saf : rkf(X,np,io) ->> rkf(X,rp,implarg),
    ex:"gi Jon penger"->>"gi penger"
].

'Free_IO_ins' derivation [
    saf : 0 ->> rkf(ben,np,io),
    cond : rkf(X,np,gov),
    ex:"slakte en sau"->>"slakte Esau en sau"
].

'IV_smallcl_AP'('Pred','Cat') derivation [
    saf : [
        0 ->> rkf(scsu,np,gov),
        0 ->> rkf('Pred','Cat',predic)
    ],
    cond : [
        ~ rkf(X,np,gov),
        default('Pred',prd),
        range('Pred',[degprd,prd]),
        default('Cat',ap),
        range('Cat',[word_x_ap,word_x_ap_excl,ap])
    ],
          %x is to be specified by the user.
    llf:,
    ex:"skyte"->>"skyte magasinet tomt"
].

'IV_smallcl_AdvP'('Pred','Cat') derivation [
    saf : [
        0 ->> rkf(scsu,np,gov),
        0 ->> rkf('Pred','Cat',predic)
    ],
    cond : [
        ~ rkf(X,np,gov),
        default('Pred',prd),
        range('Pred',[degprd,prd]),
        default('Cat',advp),
        range('Cat',[word_x_advp,word_x_advp_excl,advp])
    ],
    llf:,
].

'IV_smallcl_PP'('Pred','Cat') derivation [
    saf : [
        0 ->> rkf(scsu,np,gov),
        0 ->> rkf('Pred','Cat',predic)
    ],
    cond : [
        ~ rkf(X,np,gov),
        default('Pred',prd),
        range('Pred',[degprd,prd]),
        default('Cat',pp),
        range('Cat',[word_x_pp,pp])
    ],
    llf:,
].

'TV_smallcl_AP'('Pred','Cat') derivation [
    saf : [
        rkf(X,np,gov) ->> rkf(tvscsu,np,gov),
        0 ->> rkf('Pred','Cat',predic)
    ],
    cond:[
        default('Pred',prd),
        range('Pred',[degprd,prd]),
        default('Cat',ap),
        range('Cat',[word_x_ap,word_x_ap_excl,ap])
    ],
    llf:,
    ex:"sparke ballen"->>"sparke ballen flat"
].

'TV_smallcl_Adv_P'('Pred','Cat') derivation [
    saf : [
        rkf(X,np,gov) ->> rkf(tvscsu,np,gov),
        0 ->> rkf('Pred','Cat',predic)
    ],
    cond:[
        default('Pred',prd),range('Pred',[prd,degprd]),
        default('Cat',advp),
        range('Cat',[word_x_advp,word_x_advp_excl,advp])
    ],
    llf:,
].

'TV_smallcl_PP'('Pred','Cat') derivation [
    saf : [
        rkf(X,np,gov) ->> rkf(tvscsu,np,gov),
        0 ->> rkf('Pred','Cat',predic)
    ],
    cond:[
        default('Pred',prd),
        range('Pred',[prd,degprd]),
        default('Cat',pp),
        range('Cat',[word_x_pp,pp])
    ],
    llf:,
].

'Depict'('Cat') derivation [
    saf:[
        rkf(X,np,gov)->>rkf(scsu,np,gov),
        0->>rkf(prd,'Cat',predic)
    ],
    cond:[
        default('Cat',ap),
        range('Cat',[ap,pp])
    ],
    llf:,
    ex:"like kaffen"->>"like kaffen varm"
].

'AcI' derivation [
    saf:[
        rkf(th,at_S,gov)->>rfk(scsu,np,gov),
        0->>(prd,inf,predic)
    ],
    llf:,
    ex:"se at Jon kommer"->>"se Jon komme"
].

'ObjQual'('Cat') derivation [
    saf:0->>rkf(prd,'Cat',adjct),
    cond:[
        rkf(X,np,gov),
        default('Cat',ap),
        range('Cat',[ap,pp])
    ],
    llf:,
    ex:"drikke kaffen"->>"drikke kaffen varm"
].

'CogObj' derivation [
    saf : 0 ->> rkf(cogob,np,gov),
    cond : ~ rkf(X,np,gov),
    llf:,
    ex:"dø"->>"dø en behagelig død"
].

'InhObj' derivation [
    saf : 0 ->> rkf(inherob,np,gov),
    cond : ~ rkf(X,np,gov),
    llf:,
    ex:[
        "spytte"->>"spytte spytt",
        "skyte"->>"skyte plastikkuler"
    ]
].

load_alt derivation [
    saf : [
        rkf(inherob,np,gov) ->> rkf(inherob,np,med),
        rkf(target,np,pgov) ->> rkf(target,np,gov)
    ],
    llf:nature: resultative,
    ex:"laste h y på vogna"->>"laste vogna med h y"
].


'Part_Wh_to_DO' derivation [
    saf : [
        rkf(target,pp,pp_arg) ->> rkf(part,pp,pp_arg),
        0 ->> rkf(whole,np,gov)
    ],
    cond : ~ rkf(X,np,gov),
    ex:"spytte i ansiktet til Jon"->>"spytte Jon i ansiktet"
].

'Part_Wh_to_IO' derivation [
    saf :  [
        rkf(target,pp,pp_arg) ->> rkf(part,pp,pp_arg),
        0 ->> rkf(whole,np,io)
    ],
    cond : [
        ~ rkf(X,np,io),
        rkf(Y,np,io)
    ],
    ex:"kaste n tter i hodet på Jon"->>"kaste Jon n tter i hodet"
].

'Adv_to_DO' derivation [
    saf : rkf(path,X,Y) ->> rkf(path,np,gov),
    cond : ~ rkf(A,B,gov),
    ex:["gå veien ","jump the fence"]].

'Caus' derivation [
    saf : 0 ->> rkf(agent,np,ea),
    cond : [~rkf(A,B,ea),rkf(C,D,gov)],
    llf:[nature:sit,
         str: sit2: [head:cause
                     compl:sit1
                     role_rel:ag_of((ag,np,ea),sit2)]
    ],
    ex:"....koker vannet"->>"Jon koker vannet"
].

'Ea_caus' derivation [
    saf :
        [rkf(X,np,ea) ->> rkf(X,np,gov),0 ->> rkf(agent,np,ea)],
    cond : ~rkf(A,B,gov),
    llf:[nature:sit,
         str: sit2:[head: cause,
                    compl:sit1,
                    role_rel: ag_of(ag,np,ea),sit2)]
    ],
    ex:"the horse walked"->>"Jon walked the horse"
].

'DO_refl' derivation [
    saf : rkf(X,np,gov) ->> rkf(norole,seg,refl),
    llf:,
    ex:"Jon vasker NP"->>"Jon vasker seg"
].

'IO_refl' derivation [
    saf : rkf(X,np,io) ->> rkf(norole,seg,refl),
    llf:,
    ex:"Jon kj pte NP en frakk"->>"Jon kj pte seg en frakk"
].

'Pass_sh' derivation [
    saf : rkf(X,np,ea) ->> rkf(X,rp,implarg),
    ex:"Jon leser boken"->>"...blir lest boken"].

'Pass_lo' derivation [
    saf : rkf(X,np,ea) ->> rkf(X,np,av),
    cond: ~rkf(A,B,ea),
    ex:"Jon leser boken"->>"blir lest boken av Jon"].

'Prom_to_ea' derivation [
    saf : rkf(th,np,gov) ->> rkf(th,np,ea),
    cond : ~rkf(A,B,ea),
    ex:"koker vannet"->>"vannet koker"].

'Dem_from_ea' derivation [
    saf : rkf(X,np,ea) ->> rkf(X,np,gov),
    cond : ~ rkf(Y,Z,gov),
    ex:"en katt satt i trappen"->>"satt en katt i trappen"].

det_ins derivation [
    saf : 0 ->> rkf(norole,det_there,ea),
    cond : [~ rkf(X,Y,ea),rkf(X,at_S,gov);rkf(Y,Z,compl)],
    ex:"...satt en katt i trappen"->>"det satt en katt i trappen"].

det_ins derivation [ %lager her et nytt oppslag for det andre tilfellet.
    saf : 0 ->> rkf(norole,det_it,ea),
    cond : ~ rkf(X,Y,ea),
    ex:"...satt en katt i trappen"->>"det satt en katt i trappen"].
```

```
'SuPredAd_AP'('Pred','Cat') derivation [
    saf : 0 ->> rkf('Pred','Cat',predic),
    cond :[
        ~ rkf(X,np,gov),
        default('Pred',prd),
        range('Pred',[degprd,prd]),
        default('Cat',ap),
        range('Cat',[word_x_ap,word_x_ap_excl,ap])
    ],
    ex:"gå tom"].

'SuPredAd_PP'('Pred','Cat') derivation [
    saf : 0 ->> rkf('Pred','Cat',predic),
    cond :[
        ~ rkf(X,np,gov),
        default('Pred',dir),
        range('Pred',[dir,degprd]),
        default('Cat',pp),
        range('Cat',[word_x_pp,pp])
    ],
    llf:,
    ex:"fly i v ret"].

'SuPredAd_AdvP'('Pred','Cat') derivation [
    saf : 0 ->> rkf('Pred','Cat',predic),
    cond :[
        ~ rkf(X,np,gov),
        default('Pred',dir),
        range('Pred',[dir,degprd]),
        default('Cat',advp),
        range('Cat',[word_x_advp,word_x_advp_excl,advp])
    ],
    llf:,
    ex:"sovne"->>"sovne inn"
].

'PP_ad'('Role','Prep') derivation [
    saf : 0 ->> rkf('Role',np,'Prep'),
    cond:[
        default('Role',unsp),
        range('Role',[unsp,all_roles]),
        default('Prep',pgov),
        range('Prep',[pgov,all_prepositions])
    ],
    llf.,
    ex:[
        "snakke"->>"snakke med Ola",
        "snakke"->>"snakke med Ola om Marit"
    ]
].

 'Incorp_predic_del(G)' derivation [
    saf:[rkf(X,Y,predic)->>0,
        0->>rkf(X,Y,incorp)],
    cond:range(G, [a_ap,adv_advp],
    ex:["sparke ballen bort"->>"bortsparke ballen",
      "male huset r dt"->>"r dmale huset"]].

'Incorp_adv_add' derivation [
    saf:[rkf(X,advp,predic)->>rkf(X,pp,pp_arg),
         0->>rkf(X,adv_advp,incorp)],
    ex:"Hente Jon inn fra slummen"->>"Innhente Jon fra slummen"
].

'Incorp_p_do' derivation [
    saf:[rkf(X,pp,pp_arg)->>rkf(X,np,gov),
         0->>rkf(norole,p__pp,incorp)],
    ex:"tale om Jon"->>"omtale Jon"
```

```
].

'Incorp_p_io' derivation [
    saf:[rkf(X,pp,pp_arg)->>rkf(X,np,io),
         o->>rkf(norole,p_pp,incorp)],
    cond:[rkf(Y,np,gov),~rkf(Z,np,io)],
    ex:"sende penger til Jon"->>"tilsende Jon penger"
].

'PredicMvt'('Role','Cat') derivation [
    saf : rkf('Role',Y,predic) ->> rkf('Role',Y,preposed_predic),
    cond :[
        rkf(A,B,gov),
        default('Role',unsp),
        range('Role',[unsp,all_roles]),
        default('Cat',a_ap),
        range('Cat',[adv_advp,a_ap])
    ],
    ex:"sparke ballen ut"->>"sparke ut ballen"
].

'atS'('Funk') derivation [
    saf : rkf(th,np,'Funk') ->> rkf(th,at_S,'Funk'),
    cond: [
        default('Funk',gov),
        range('Funk',[gov,ea,pgov,all_predpositions])
    ],
    ex:"Per sa sannheten"->>"Per sa at han var syk"].

'inf_regcontr'('Funk') derivation [
    saf : rkf(th,np,'Funk') ->> rkf(th,å_inf_regcontr,'Funk'),
    cond:[
        default('Funk',gov),
        range('Funk',[gov,ea,pgov,all_prepositions])
    ],
    ex:"Per pr vde skoene"->>"Per pr vde å gå"].

'inf_markcontr'('Funk') derivation [
    saf : rkf(th,np,'Funk') ->> rkf(th,å_inf_markcontr,'Funk'),
    cond:[
        default('Funk',gov),
        range('Funk',[gov,ea,pgov,all_prepositions])
    ],
    ex:"Jon lovet Marit å gå"].

'inf_arbcontr'('Funk') derivation [
    saf : rkf(th,np,'Funk') ->> rkf(th,å_inf_arbcontr,'Funk'),
    cond:[
        default('Funk',gov),
        range('Funk',[gov,ea,pgov,all_prepositions])
    ],
    ex:"vi snakket om det å sublimere seg"].

 'hvS'('Funk') derivation [
    saf:rkf(th,np,'Funk')->>rkf(th,hv_S,'Funk'),
    cond:[
        default('Funk',gov),
        range('Funk',[gov,pgov,all_prepositions])
    ],
    llf:,
    ex:"si sannheten"->>"si hvem som hadde gjort det"].

 'omS'('Funk') derivation [
    saf:rkf(th,np,'Funk')->>rkf(th,om_S,'Funk'),
    cond:[default('Funk',gov),range('Funk',[gov,pgov,all_prepositions])],
    ex:"lure på svaret"->>"lure på om du hadde gjort det"].

 'SubjRais'('Funk') derivation [
    saf:[rkf(scsu,np,'Funk')->>rkf(scsu,e,'Funk'),0->>rkf(no,np,ea)],
    cond:[default('Funk',gov),range('Funk',[gov,pgov])],
    ex:"later til Per å v re syk"->>"Per later til [e] å v re syk"].

 'Oinf_to Predic_AP' derivation [
    saf:rkf(prd,o_inf,predic)->>rkf(prd,ap,predic),
    ex:"synes Jon å v re syk"->>"synes Jon syk"].

 'atS_to_Cl_inf'('Funk') derivation [
    saf: [rkf(th,at_S,'Funk')->>rkf(scsu,np,'Funk'),0->>rkf(prd,o_inf,predic)],
    cond: [default('Funk',gov),range('Funk',[gov,pgov])],
    ex:"later til at Jon er syk"->>"later til Jon å v re syk"].

 'Prop_compl_to_SC'('Som') derivation [
    saf:[rkf(prop_arg;'Som',compl)->>rkf(scsu,np,gov),0->>rkf(prd,ap,predic)],
    cond:[default('Som','som_om_S'),range('Som',['som_om_S','som_S'])],
    ex:"virker som om Jon er syk"->>"virker Jon syk"].

 'RaisCop'('Cat') derivation [
    saf: [rkf(prop_arg,'Cat',compl)->>rkf(prop_arg,'Cat',subj_contr_compl),
     0->>rkf(no,np,ea)],
    cond: ~rkf(X,Y,ea),
    ex:"virker som (om) Jon er syk"->>"Jon virker som (om) han er syk"].

 'Anti_exp'('Funk') derivation [
    saf: rkf(th,at_S,'Funk')->>rkf(th,at_S,ea),
    cond: [default('Funk',gov),range('Funk',[gov,pgov])],
    ex:"bekymrer meg at Jon kom"->>"at Jon kom bekymrer meg"].

 'ExpMvt' derivation [
    saf:[rkf(exp,np,io)->>rkf(exp,np,ea),
         rkf(th,np,gov)->>rkf(th,np,over),
         0->>rkf(norole,seg,refl)],
    ex:"irriterer meg katten"->>"jeg irriterer meg over katten"].

'for_oinf_to_predic_som'('Cat') derivation [
    saf: rkf(prd,o_inf,for)->>rkf(prd,'Cat',som),
    cond:[default('Cat',ap),range('Cat',[ap,np])],
    ex:"anse Jon for å v re skyldig"->>"anse Jon som skyldig"].

 'Erg_refl' derivation [
    saf: 0->>rkf(norole,seg,refl),
    cond: [~rkf(A,B,ea),rkf(th,np,gov)],
    ex:"åpner en d r"->>"åpner seg en d r"].

 'PP'('Arg') derivation [
    saf: rkf(prd,np,predic)->>rkf(prd,np,'Arg'),
    cond: [default('Arg',pgov),range('Arg',[pgov,all_prepositions])],
    ex: "kalle Jon et geni"->>"kalle Jon for et geni"].

'atS_to_SC' derivation [
    saf :rkf(th,at_S,gov)->>rkf(scsu,np,gov),0->>rkf(prd,ap,predic),
    llf:,
    ex:"forestille seg at Per er frisk"->>"forestille seg Per frisk"].

 'atS_to_prop_compl' derivation [
    saf: rkf(th,at_S,gov)->>rkf(prop_arg,som_om_S,compl),
    llf:,
    ex:"det synes meg at Per er syk" ->>"det synes meg som om Per er syk"].

 'Exp_del' derivation [
    saf: rkf(exp,np,io)->>rkf(exp,rp,implarg),
    ex:"irriterer Per vinden"->>"irriterer vinden"].
```

```
'Process1'('Prep','Aff','Comb') derivation [
    major_cat:N,
    %segmental(informal): Comb(H,Aff)
        cond : range ('Aff', [ing1,else1,0]),
    template:[
        saf : [
            rkf(ag,np,ea) ->> rkf(ag,np,gen) or rkf(ag,rp,implarg) or rkf(ag,np,av) or rkf(ag,a_ap,attr),
            rkf(th,np,gov) ->> `rkf(th,np,'Prep') or rkf(th,rp,implarg) or rkf(th,n,incorp)],
            0->>(ident,np,ea)
        ],
        cond : [
            default('Prep',av),
            range('Prep',[av,all_prepositions]),
            %rkf(th,np,'Prep') only in case there is no rkf(ag,np,av)
        ],
        llf:[ nature:process
             str:sit1]
    ],
    ex:["Per bygger et hus" ->>"Pers bygging av huset" or "husbygging av Per" or
        "bygging av huset" or "bygging av Per"or "byggingen"
        ]
].

'Process2'('Aff','Comb') derivation [
    major_cat:N,
    %segmental(informal): Comb(H,Aff).
        cond: range('Aff',[0,]),
    template:[
        saf : [rkf(X,np,ea) ->> rkf(X,np,gen) or rkf(X,rp,implarg) or rkf(X,n,incorp),
               0->>(ident,np,ea)]
        llf:nature:process
        ],
    ex:"Kirken brenner"->>"kirkens brann" or "brannen" or"kirkebrann"
].

'Result3'('Aff','Comb') derivation [
    major_cat:N,
    %segmental(informal): Comb(H,Aff)
        cond:range ('Aff',[st3,ning3,else3,ling3,0]),
    template:[
        saf :[ rkf(ag,np,ea) ->>0,
               rkf(th,np,gov)->>rkf(th,np,ea)
        ],
        llf:[ nature:thing,result
              str:[ abstr: th,
                    body:sit:
                              role_rel:th_of:rkf(th,np,ea)]
    ],
    ex:["Per bygger et hus" ->> "bygningen",
        "Per fanger fluer"->>"fangst"]
].

'Event4'('Prep','Aff','Comb') derivation [
    major_cat:N,
    %segmental(informal): Comb(H,Aff)
        cond:range ('Aff',[st4,0,]),
    template:[
        saf : [
            rkf(X,np,ea) ->> rkf(X,np,gen) or rkf(X,rp,implarg) or rkf(X,a_ap,attr),
            rkf(th,np,gov)-> rkf(th,np,'Prep') or rkf(th,rp,implarg) rkf(th,n,incorp),
            0->>rkf(ident,np,ea)
        ],
        cond : [
            default('Prep',av),
            range('Prep',[av,all_prepositions])
        ],
        llf:nature: event
    ],
    ex:["Kongen ankommer"->>"kongens ankomst" or "ankomsten" or "kongelig ankomst" or "ankomsten av kongen"
       "han myrder kongen"->>"hans mord på kongen" or "kongemord" or "mord på kongen"
       "han bryter avtalen"->> "hans brudd på avtalen" or "brudd på avtalen" or "avtalebrudd"]
].
%Comment: not as protracted as Process1 or2

'Event5'('Aff','Comb') derivation   [
    major_cat:N,
    %segmental(informal): comb(H,Aff)
        cond:range ('Aff',[0,]),
    template:[
        saf : [ rkf(X,np,ea) ->> 0,
                0->>(event,np,ea)]
        llf:nature:event
    ],
    ex:"Alle løper"->>"søndagens løp"
].

'Agent6'('Aff','Comb') derivation [
    major_cat: N,
    %segmental(informal): Comb(H,Aff)
        cond: [
                default: ('Aff',er6),
                range ('Aff',[er6,ing6,ator6, ent6])
        ],
    template:[
        saf :
            rkf(th,np,gov) ->>rkf(th,np,av) or rkf(th,rp,implarg) or rkf(th,n,incorp),
        cond: [
                default('X',ag),
                range ('X',[ag,ben])
        ],
        llf:[nature:thing
            str:abstr:X,
                body:sit:
                        role_rel:('of(X)'):rkf(X,np,ea),
                                th_of:rkf(th,np,av)
    ],
    ex:"han beundrer kunst"->>"beunderer av kunst" or "beunderer" or "kunstbeunderer",
       "han mottar et brev"->>"mottakeren av brevet" or.....
].

'Agent7'('Aff','Comb') derivation [
    major_cat:N
    %segmental(informal): Comb(H,Aff)
        cond :[
              default: ('Aff',er7),
              range ('Aff',[er7,ing7,ist7,ent7, ling7])
        ],
    template:[
        saf :rkf(th,np,gov)->>0
        ],
        llf:[nature:thing,
             str:[abstr:ag
                  body:sit:
                        role_rel:ag_of:rkf(ag,np,ea)]
             ]
    ],
    ex:["han baker brød"->>"baker",
        "han l rer et yrke"->>l rling"]
].
%Will be parameterized so as to include intermediate agents, i.e. promoted Instr.

'Emotion8'('X','Func','Aff','Comb') derivation [
    major_cat:N,
    %segmental(informal): Comb(H,Aff)
    cond :[
              default: ('Aff',else8),
              range ('Aff',[else8,sjon8,ing8,0])
      . ],
    template:
        saf: [rkf(X,np,ea) ->>rkf(X,np,gen) or rkf(X,rp,implarg) or rkf(X,a_ap,attr),
              rkf(th,np,'Func') ->>rkf(th,np,'Func') or rkf(th,n,incorp) or rkf(th,implarg)
              rkf(norole,seg,refl) ->>0
              0->>rkf(ident,np,ea)
        ],
        cond:[default(X,exp),
              range(X,all_roles),
              default('Func',gov),
              range('Func',[gov,pgov,all_prepositions])],
        llf:nature:emotion/state
    ],
    ex:["per føler avmakt" ->>"pers følelse av avmakt"or "pers avmaktsfølelse" or "nasjonal følelse av skam",
        "per hater folket"->>"pers hat mot folket"or"pers folkehat",
        "per tror på Gud"->>"pers tro på Gud","Pers Gudstro","Pers tro",
        "per forelsker seg ->> "pers forelskelse",
        "han beundrer kongen"->>"hans beundring for kongen"or "beundringen av Per for kongen"]
        "Jon irriterer seg over naboen" ->>Jons irritasjon over naboen" or "norsk irritasjon over naboen",
        "Jon skammer seg over sin tabbe" ->>"Jons skam over sin tabbe"
        "Jon er forbauset over v ret" ->>"Jons forbauselse over v ret"]
].

'ConcreteN_with_qualityA9'('Aff','Comb') derivation [
    major_cat:N,
    %segmental(informal): Comb(H,Aff)
        cond :[
              default: ('Aff',dom9),
              range ('Aff',[dom,ling9,ing9,0])
        ],
    template:[
        llf:[nature:thing
             str:[abstr: (X,np,ea)
                  body:sit:[ head:,
                             compl:,
                             role_rel:('of(X)'):rkf(X,np,ea)
                        ]
                 ]
    ],
    ex:["han er svak" ->>"svekling",
        "han er ung"->>"ungdom",
        "dette er hellig"->>"helligdom"]
].

'Quality10' ('Aff','Comb') derivation [
    major_cat:N,
    %segmental(informal): Comb(H,Aff)
    cond :[
              default: ('Aff',het10),
              range ('Aff',[het10,itet10])
        ],
    template:[
        saf : [rkf(X,np,ea) ->>rkf(X,np,gen) or rkf(X,np,av),
               0->>(ident,np,ea)
        ],
        llf:nature:quality
    ],
    ex:"steinen er ekte"->>"steinens ekthet" or "ektheten av steinen"
].

'State11' ('Aff','Comb') derivation [
    major_cat:N,
    %segmental(informal): Comb(H,Aff)
        cond : range ('Aff',[dom11,skap11,0])
    template:[
        saf : rkf(X,np,ea) ->>rkf(X,np,gen)
              0->>rkf(Y,np,ea)
        llf:nature:state
    ],
    ex:["rik" ->>"rikdom",
        "ung"->>"ungdom",
        "fangen"->>"fangenskap"]
].

'Setof12' ('Aff','Comb') derivation [
    major_cat:N,
    %segmental(informal): Comb(H,Aff)
        cond:range ('Aff',[het12,skap12,0]),
    template:[
        llf:[nature:group,
             str:member]
    ],
    ex:["menneske" ->>"menneskehet"
        "bror"->>"brorskap",
        "prest"->>"presteskap"]
```

```
].

'Totalityof13' ('Aff','Comb') derivation [
    major_cat:N,
    %segmental(informal): Comb(H,Aff)
      cond: range ('Aff',[dom13,skap13,0]),
    template:[
        llf: nature:totalityof
    ],
    ex:"morskap", "brorskap",
].

'Manner14' ('Aff','Comb') derivation [
    major_cat:N,
    %segmental(informal): Comb(H,Aff)
        cond: ('Aff',[sel14,ing14]),
    template:[
        saf:[rkf(ag,np,ea)->>rkf(ag,np,gen) or rkf(ag,n,incorp),
             rkf(norole,seg,refl) ->>0,
             rkf(manner,rp,implarg)->>rkf(manner,np,ea)
        ],
        llf:[nature:manner,
             str:[abstr:manner
                  body: sit:
                            role_rel:manner_of(manner,np,ea)
             ]
        ]
    ],
    ex:[ "han oppfører seg bra"->>"hans oppførsel",
        "han løper elegant"->>"hans løping"]
].

'TP15'('Func','Prep','Aff','Comb') derivation [
    major_cat:N,
    %segmental(informal): Comb(H,Aff)
    cond: range('Aff',[else15,sjon15,0,ing15,ning15]),
    template:[
        saf: [rkf(ag,np,ea) ->> rkf(cre,np,gen) or rkf(cre,np,ved) or rkf(cre,adj,pred),
              rkf(th,np,'Func') ->> rkf(matter,np,'Prep') or rkf(matter,n,incorp) or rkf(matter,rp,implarg)
                                    or rkf(matter,np,gen),
              0->>rkf(X,np,ea)
        ],
        cond:[default('Func',gov),
              range(gov,pgov,[all_prepositions]),
              default('Prep',pgov),
              range('Prep',[pgov,all_prepositions])
          ],
          llf:nature:thing
     ],
     ex:"beskrive" ->>"hans beskrivelse av saken" or "saksbeskrivelsen" or saksbeskrivelsen ved rektoren".
].

'Event16'('Prep','Aff','Comb') derivation [
     major_cat:N,
     %segmental(informal): Comb(H,Aff)
     template:[
          saf:[rkf(ag,np,ea) ->> rkf(ag,n,incorp) or rkf(ag,np,av) or rkf(ag,rp,implarg),
               rkf(th,np,gov)->>rkf(th,np,Prep) or rkf(th,rp,implarg),
               0->>rkf(ident,np,ea)
          ],
          cond: [default('Prep',på),
                 range('Prep',[på,all_prepositions])
          ],
          llf:nature:event
     ],
     ex:"katten angriper naboen"->>kattens angrep på naboen","katteangrep", "angrep på naboen"
].
%Comment: has AG-incorp, no TH-incorp.

'Loc17' ('Aff','Comb') derivation [
major_cat:N,
     %segmental(informal): Comb(H,Aff)
          cond:range ('Aff',[eri17,0]),
     template:[
          saf:[rkf(ag,np,ea) ->> 0,
               rkf(th,np,gov)->>0,
               rkf(loc,rp,implarg)->>rkf(loc,np,ea)
          ],
          llf:[nature:location,
               str: abstr:loc
                    body:sit:
                         role_rel:loc_of(loc,np,ea)]
          ]
     ],
     ex:["bake"->>"bakeri",
          "bryte"->>"steinbrudd"]
].


'Setof19' ('Aff','Comb') derivation [
    major_cat: N,
    %segmental(informal): Comb(H,Aff)
        cond:range ('Aff',[het19,0]),
    template:[
        llf:[nature:group,
    .        str: state: [head: setof
                          compl:state]
        ]
    ],
    ex:["almen"->>"almenhet",
        "offentlig"->>"offentlighet",
        "myndig"->>"myndighetene"]
].
% Is probably included in 'Setof12'.

'Possibility20'('Aff','Comb') derivation [
    major_cat:A,
    %segmental(informal): Comb(H,Aff)
        cond :range (('Aff',[bar20,lig21]),
    template:[
        saf :[ rkf(ag,np,ea) ->>0,
               rkf(th,np,gov) ->>rkf(th,np,ea),
               0->rkf(exp,np,for),
        % exp is identical to ag
        ],
        llf:[nature:possibility,
             str: sit2: [head:possible,
                         compl:sit1,
                         role_rel:exp_of((exp,np,for),sit2)]
        ]
    ],
    ex:"han bruker hammeren"->>"hammeren er brukbar "
].

'Potentiality21' ('Aff','Comb') derivation [
    major_cat:A,
    %segmental(informal): Comb(H,Aff)
        cond :range ('Aff',[bar21,lig21]),
    template:[
        llf:[nature:potentiality,
             str:[head:potential,
                 compl:sit]
        ]
    ],
    ex:"materialet brenner"->>"brennbar "
```

```
].

'Potentiality22' ('Aff','Comb') derivation [
    major_cat:A,
    %segmental(informal): Comb(H,Aff)
    .   cond :range ('Aff',[lig22]),
    template:[
        saf :[rkf(X,np,'Func')->>rkf(X,np,ea),
              rkf(exp,np,'Func')->>rkf(exp,np,for),
        cond:[
              default('Func',gov),
              range('Func',[gov,pgov, all_prepositions])
              %'Func' has to be different on the 2 occurrences
        ],
        llf:[nature:potentiality,
             str:[head:potential to evoke
                compl:sit1]
    ],
    ex:"Jeg ergrer meg over dette"->> "dette er ergerlig"
].

'Potentiality23' ('Aff','Comb') derivation [
    major_cat:A,
    %segmental(informal): Comb(H,Aff)
        cond :range ('Aff',[abel23]),
    template:[
        saf :[rkf(X,np,'Func')->>rkf(X,np,ea),
        cond: [default('Func',gov),
              range('Func',[gov,io,pgov, all_prepositions])
        ],
        llf:[nature:potentiality,
             str:[head:potential to be(pass),
                  compl:sit]
    ],
    ex:["Jeg irriterer meg over dette"->> "jeg er irritabel",
        "Man kan diskutere saken"->>"saken er diskutabel"]
].


'TypProperty24' ('Aff','Comb') derivation [
    major_cat:A,
    %segmental(informal): Comb(H,Aff)
        cond :range ('Aff',[som24,lig24]),
    template:
        saf : rkf(X,np,gov)->>rkf(X,np,'Prep'),
        cond:[default('Prep',på),
              range ('Prep',all_prepositions)],
        llf:nature:property
    ],
    ex:["han prater mye "->>"han er pratsom",
        "han misunner meg"->> "han er misunnelig på meg",
        "han står på sitt"->>"han er påståelig"]
].


'Neg26'('Aff','Comb') derivation [
    major_cat: id,
    %segmental(informal): Comb(H,Aff)
        cond :range ('Aff',[u26,in26]),
    template:[
        saf :id,
        llf:[nature:Property
             str:property2: [head:negation
                             compl: property1]
        ]
    ]
    ex:"leselig"->> "uleselig"
].

'Totalityof27' ('Aff','Comb') derivation [
    major_cat:N,
    %segmental(informal): Comb(H,Aff)
        cond: range ('Aff',[dom27,0]),
    template:[
        saf:rkf(ag,np,ea)->>0,
        llf:[ nature:totalityof
              str:head:
                 role_rel: th_of(th,np,ea)]
    ],
    ex:"l rdom"
].
%Idem Totalityof13' if base category is irrelevant or parameterized.

'be_alt28' derivation [
    major_cat: id,
    %segmental(informal): Comb(H,'Aff'),
        cond: range('Aff',be28)
    template:[
        saf : [
            rkf(target,np,pgov) ->> rkf(target,np,gov)
        ],
        llf:[nature:sit,
            str:[sit:
                    head:,
                    role_rel:,
                    ]
            ]
    ],
    ex:"skyte plastikkuler på Jon"->>"beskyte Jon med plastikkuler"
].
% Add intensionality feature. (To ea or to sit?).

'ExpAdj29'('Prep','Func','Aff','Comb') derivation [
    major_cat: A,
    %segmental(informal): Comb(H,'Aff'),
        cond: range('Aff',t29)
    template:[
        saf:[rkf(exp,np,'Func')->>rkf(exp,np,ea),
            rkf(th,np,gov)->>rkf(th,np,'Prep')
        ],
        cond:[default('Prep',over),
              range ('Prep',all_prepositions)
              default('Func',io)
              range('Func',all_functions],
        llf:[nature: state
             str: [head:result,
                   compl:sit1]
            ]
        ],
    ex:"irriterer meg katten"->>"jeg er irritert over katten",
        "jeg ergrer meg over dette"->>"jeg er ergerlig"].

'be_alt30' derivation [
    major_cat:V,
    %segmental(informal): Comb(H,'Aff'),
        cond: range('Aff',be30)
    template:[
        saf : [
            0->>rkf(ag,np,ea),
            rkf(th,np,ea) ->> 0,
            0->> rkf(th,np,gov)
        ],
        llf:[nature:sit,
             str:sit1:[head:cause,
                    compl:state,
                    role-rel:ag_of((ag,np,ea),sit2)],
        ],
    ex:["rik"->>"berike livet med lesning",
       "fri"->>"befri noen fra lenker",
       "rolig"->>berolige noen"]
  ].
```

10.81

Adjektiver

```
a1 :=  set # [
       [form :  (<-),infl : m_f_sg,'LLF' : log_m_f_sg],
       [form :  (<-) + "t",infl : n_sg,'LLF' : log_n_sg],
       [form :  (<-) + "e",infl : pl,'LLF' : log_pl]
].

a2 :=  set # [
       [form :  (<-),infl : m_f_sg,'LLF' : log_m_f_sg],
       [form :  (<-),infl : n_sg,'LLF' : log_n_sg],
       [form :  (<-) + "e",infl : pl,'LLF' : log_pl]
].

a3 :=  set # [
       [form :  (<-),infl : m_f_sg,'LLF' : log_m_f_sg],
       [form :  (<-),infl : n_sg,'LLF' : log_n_sg],
       [form :  (<-),infl : pl,'LLF' : log_pl]
].

a4 :=  set # [
       [form :  (<-),infl : m_f_sg,'LLF' : log_m_f_sg],
       [form :  (<-),infl : n_sg,'LLF' : log_n_sg],
       [form :  (<-) + "e",infl : pl,'LLF' : log_pl]
].

a5 :=  set # [
       [form :  (<-),infl : m_f_sg,'LLF' : log_m_f_sg],
       [form :  (<-) + "t",infl : n_sg,'LLF' : log_n_sg],
       [form :  (<-) -  (<-) :< [1,2] +  (<-) :< 1 + "e",infl : pl,
     'LLF' : log_pl]
].
```

Nomener

```
n1 :=  set # [
       [form :  (<-),infl : sg_ind,'LLF' : log_sg_ind],
       [form :  (<-) + "et",infl : sg_d,'LLF' : log_sg_d],
       [form :  (<-),infl : pl_ind,'LLF' : log_pl_ind],
       [form :  (<-) + "ene",infl : pl_d,'LLF' : log_pl_d]
].

n2 :=  set # [
       [form :  (<-),infl : sg_ind,'LLF' : log_sg_ind],
       [form :  (<-) + "t",infl : sg_d,'LLF' : log_sg_d],
       [form :  (<-) + "r",infl : pl_ind,'LLF' : log_pl_ind],
       [form :  (<-) + "ne",infl : pl_d,'LLF' : log_pl_d]
].

n3 :=  set # [
       [form :  (<-),infl : sg_ind,'LLF' : log_sg_ind],
       [form :  (<-) + "et",infl : sg_d,'LLF' : log_sg_d],
       [form :  (<-) + "er",infl : pl_ind,'LLF' : log_pl_ind],
       [form :  (<-) + "ene",infl : pl_d,'LLF' : log_pl_d]
].
```

```
n4 :=  set # [
       [form :  (<-),infl : sg_ind,'LLF' : log_sg_ind],
       [form :  (<-) + "et",infl : sg_d,'LLF' : log_sg_d],
       [form :  (<-) + "er",infl : pl_ind,'LLF' : log_pl_ind],
       [form :  (<-) + "ene",infl : pl_d,'LLF' : log_pl_d]
].

f1 :=  set # [
       [form :  (<-),infl : sg_ind,'LLF' : log_sg_ind],
       [form :  (<-) + "a",infl : sg_d,'LLF' : log_sg_d],
       [form :  (<-) + "er",infl : pl_ind,'LLF' : log_pl_ind],
       [form :  (<-) + "ene",infl : pl_d,'LLF' : log_pl_d]
].

f2 :=  set # [
       [form :  (<-),infl : sg_ind,'LLF' : log_sg_ind],
       [form :  (<-) - "e" + "a",infl : sg_d,'LLF' : log_sg_d],
       [form :  (<-) + "r",infl : pl_ind,'LLF' : log_pl_ind],
       [form :  (<-) + "ne",infl : pl_d,'LLF' : log_pl_d]
].

m1 :=  set # [
       [form :  (<-),infl : sg_ind,'LLF' : log_sg_ind],
       [form :  (<-) + "en",infl : sg_d,'LLF' : log_sg_d],
       [form :  (<-) + "er",infl : pl_ind,'LLF' : log_pl_ind],
       [form :  (<-) + "ene",infl : pl_d,'LLF' : log_pl_d]
].

m2 :=  set # [
       [form :  (<-),infl : sg_ind,'LLF' : log_sg_ind],
       [form :  (<-) + "n",infl : sg_d,'LLF' : log_sg_d],
       [form :  (<-) + "r",infl : pl_ind,'LLF' : log_pl_ind],
       [form :  (<-) + "ne",infl : pl_d,'LLF' : log_pl_d]
].

m3 :=  set # [
       [form :  (<-),infl : sg_ind,'LLF' : log_sg_ind],
       [form :  (<-) + "en",infl : sg_d,'LLF' : log_sg_d],
       [form :  (<-) + "e",infl : pl_ind,'LLF' : log_pl_ind],
       [form :  (<-) + "ne",infl : pl_d,'LLF' : log_pl_d]
].
```

Verber

```
v1 :=  set # [
       [form :  (<-),infl : inf,'LLF' : log_inf],
       [form :  (<-) + "r",infl : pres,'LLF' : log_pres],
       [form :  (<-) + "t",infl : past,'LLF' : log_past],
       [form :  (<-) + "t",infl : past_p,'LLF' : log_past_p],
       [form :  (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].

v2 :=  set # [
       [form :  (<-),infl : inf,'LLF' : log_inf],
       [form :  (<-) + "r",infl : pres,'LLF' : log_pres],
       [form :  (<-) - "e" + "te",infl : past,'LLF' : log_past],
       [form :  (<-) - "e" + "t",infl : past_p,'LLF' : log_past_p],
       [form :  (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].
```

```
v3 :=  set # [
      [form :  (<-),infl : inf,'LLF' : log_inf],
      [form :  (<-) + "r",infl : pres,'LLF' : log_pres],
      [form :  (<-) - "e" + "de",infl : past,'LLF' : log_past],
      [form :  (<-) - "e" + "d",infl : past_p,'LLF' : log_past_p],
      [form :  (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].

v4 :=  set # [
      [form :  (<-),infl : inf,'LLF' : log_inf],
      [form :  (<-) + "r",infl : pres,'LLF' : log_pres],
      [form :  (<-) + "dde",infl : past,'LLF' : log_past],
      [form :  (<-) + "dd",infl : past_p,'LLF' : log_past_p],
      [form :  (<-) + "ende",infl : pres_p,'LLF' : log_pres_p]
].

v5 :=  set # [
      [form :  (<-),infl : inf,'LLF' : log_inf],
      [form :  (<-) -  (<-) :< [1,2],infl : pres,'LLF' : log_pres],
      [form : omlyd( (<-),"u") -  (<-) :< [1,2] + "te",infl : past,
    'LLF' : log_past],
      [form : omlyd( (<-),"u") -  (<-) :< [1,2] + "t",infl : past_p,
    'LLF' : log_past_p],
      [form :  (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].

v6 :=  set # [
      [form :  (<-),infl : inf,'LLF' : log_inf],
      [form :  (<-) + "r",infl : pres,'LLF' : log_pres],
      [form : omlyd( (<-),"e") - "e",infl : past,'LLF' : log_past],
      [form : omlyd( (<-),"e") + "t",infl : past_p,
    'LLF' : log_past_p],
      [form :  (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].

v7 :=  set # [
      [form :  (<-),infl : inf,'LLF' : log_inf],
      [form :  (<-) + "r",infl : pres,'LLF' : log_pres],
      [form : omlyd( (<-),"e") - "e",infl : past,'LLF' : log_past],
      [form :  (<-) - "e" + "t",infl : past_p,'LLF' : log_past_p],
      [form :  (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].

v8 :=  set # [
      [form :  (<-),infl : inf,'LLF' : log_inf],
      [form :  (<-) + "r",infl : pres,'LLF' : log_pres],
      [form : omlyd( (<-),"ei"),infl : past,'LLF' : log_past],
      [form :  (<-) + "dd",infl : past_p,'LLF' : log_past_p],
      [form :  (<-) + "dende",infl : pres_p,'LLF' : log_pres_p]
].

v9 :=  set # [
      [form :  (<-),infl : inf,'LLF' : log_inf],
      [form :  (<-) + "r",infl : pres,'LLF' : log_pres],
      [form : omlyd( (<-),"\277") - "e",infl : past,
    'LLF' : log_past],
      [form : omlyd( (<-),"\277") + "t",infl : past_p,
    'LLF' : log_past_p],
      [form :  (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].
```

```
v10 :=  set # [
      [form :   (<-),infl : inf,'LLF' : log_inf],
      [form :   (<-) + "r",infl : pres,'LLF' : log_pres],
      [form : omlyd( (<-),"\277") - "e",infl : past,
    'LLF' : log_past],
      [form :   (<-) - "e" + "t",infl : past_p,'LLF' : log_past_p],
      [form :   (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].

v11 :=  set # [
      [form :   (<-),infl : inf,'LLF' : log_inf],
      [form :   (<-) + "r",infl : pres,'LLF' : log_pres],
      [form : omlyd( (<-),"\277") + "d",infl : past,
    'LLF' : log_past],
      [form :   (<-) + "dd",infl : past_p,'LLF' : log_past_p],
      [form :   (<-) + "dende",infl : pres_p,'LLF' : log_pres_p]
].

v12 :=  set # [
      [form :   (<-),infl : inf,'LLF' : log_inf],
      [form :   (<-) + "r",infl : pres,'LLF' : log_pres],
      [form : omlyd( (<-),"a") - "e",infl : past,'LLF' : log_past],
      [form : omlyd( (<-),"u") + "t",infl : past_p,
    'LLF' : log_past_p],
      [form :   (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].

v13 :=  set # [
      [form :   (<-),infl : inf,'LLF' : log_inf],
      [form :   (<-) + "r",infl : pres,'LLF' : log_pres],
      [form : omlyd( (<-),"a") - "e",infl : past,'LLF' : log_past],
      [form : omlyd( (<-),"\214") + "t",infl : past_p,
    'LLF' : log_past_p],
      [form :   (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].

v14 :=  set # [
      [form :   (<-),infl : inf,'LLF' : log_inf],
      [form :   (<-) + "r",infl : pres,'LLF' : log_pres],
      [form : omlyd( (<-),"a") - "e",infl : past,'LLF' : log_past],
      [form :   (<-) + "t",infl : past_p,'LLF' : log_past_p],
      [form :   (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].

v15 :=  set # [
      [form :   (<-),infl : inf,'LLF' : log_inf],
      [form :   (<-) + "r",infl : pres,'LLF' : log_pres],
      [form : omlyd( (<-),"a") - "e",infl : past,'LLF' : log_past],
      [form :   (<-) - "e" + "t",infl : past_p,'LLF' : log_past_p],
      [form :   (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].

v16 :=  set # [
      [form :   (<-),infl : inf,'LLF' : log_inf],
      [form :   (<-) + "r",infl : pres,'LLF' : log_pres],
      [form : omlyd( (<-),"i") + "kk",infl : past,'LLF' : log_past],
      [form :   (<-) + "tt",infl : past_p,'LLF' : log_past_p],
      [form :   (<-) + "ende",infl : pres_p,'LLF' : log_pres_p]
].
```

```
v17 :=  set # [
      [form :   (<-),infl : inf,'LLF' : log_inf],
      [form :   (<-) + "r",infl : pres,'LLF' : log_pres],
      [form :   (<-) - "e",infl : past,'LLF' : log_past],
      [form :   (<-) + "t",infl : past_p,'LLF' : log_past_p],
      [form :   (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].

v18 :=  set # [
      [form :   (<-),infl : inf,'LLF' : log_inf],
      [form :   (<-) + "r",infl : pres,'LLF' : log_pres],
      [form : omlyd( (<-),"a") - "e" + "t",infl : past,
    'LLF' : log_past],
      [form : omlyd( (<-),"u") + "t",infl : past_p,
    'LLF' : log_past_p],
      [form :   (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].

v19 :=  set # [
      [form :   (<-),infl : inf,'LLF' : log_inf],
      [form :   (<-) + "r",infl : pres,'LLF' : log_pres],
      [form : omlyd( (<-),"a") - "e" + "t",infl : past,
    'LLF' : log_past],
      [form :   (<-) - "e" + "t",infl : past_p,'LLF' : log_past_p],
      [form :   (<-) + "nde",infl : pres_p,'LLF' : log_pres_p]
].
```

App.5

Preliminary list of affixes

```
ing1 : spising, ødeleggelse, bedervelse
else1: ødeleggelse, bedervelse
0:Process2:brann
st3:   fangst,hogst, bakst
ning3: bygning,
ling3: forsamling, samling
else3: ødeleggelsene, forstyrrelsene
0:Result3: funn
st4:  ankomst
0:Event4:mord
er6 : beunderer av kunst,
ing6: arving
ator6: initiator
ent6: dirigent
er7 : baker
ent7:dirigent
ist7:
ing7:
else8: forbauselse,
sjon8:irritasjon
ing8: forundring
ing9:luring,raping
ling9: svekling
dom9: ungdom, helligdom
het10: ekthet
dom11: ungdom
skap11: fangenskap
het 12: menneskehet
skap12: presteskap
skap13: morskap,forfatterskap
dom13: alderdom, barndom
sel14: oppførsel
ing14: løping
ing15: fortelling
ning15: forelesning
else15: beskrivelse
else17: forelskelse
sjon17:
ing17: beundring
ling18:l rling
het19: myndighetene
lig20: spiselig
bar20: brukbar,kjørbar
lig21:
bar21: brennbar
lig22: skammelig,ergerlig
abel23: diskutabel,irritabel
som24: pratsom
lig25: ergerlig
u26: uleselig
in26: inabsorberbar
```

APP. 6

```
xtemplate(1,iv,
['Per skyter' : [basic : base],
 'det g\214r en mann' :
  [detins : base ++ 'Dem_from_ea' ++ det_ins],
 'Per synger en sang' : ['CogObj' : base ++ 'CogObj'],
 'ble sunget en sang ' :
  ['CogObj_Pass' : base ++ 'CogObj' ++ 'Pass'],
 'Per sang henne en sang' :
  ['FreeIOins' : base ++ 'CogObj' ++ 'Free_IO_ins'],
 'Per g\214r seç en tur' :
  ['IOrefl' : base ++ 'CogObj' ++ 'Free_IO_ins' ++ 'IO_refl'],
 'Per skyter kuler' : ['InhObj' : base ++ 'InhObj'],
 'ble skutt kuler' :
  ['InhObj_Pass' : base ++ 'InhObj' ++ 'Pass'],
 'Per skyter kulene bort' :
  ['TVsmallclAdvP' : base ++ 'InhObj' ++ 'TV_smallcl_AdvP'],
 'ble skutt kulene bort' :
  ['TVsmallclAdvP_Pass' :
    base ++ 'InhObj' ++ 'TV_smallcl_AdvP' ++ 'Pass'],
 'Per skyter bort kulene' :
  ['TVsmallclAdvP_PredicMvt' :
    base ++ 'InhObj' ++ 'TV_smallcl_AdvP' ++
     'PredicMvt'(_217,_218)],
 'ble skutt bort kulene' :
  ['TVsmallclAdvP_PredicMvt_Pass' :
    base ++ 'InhObj' ++ 'TV_smallcl_AdvP' ++
     'PredicMvt'(_355,_356) ++ 'Pass'],
 'Per bortskyter kuler' :
  ['TVsmallclAdvP_IncorpAdv' :
    base ++ 'InhObj' ++ 'TV_smallcl_AdvP' ++
     'Incorp_Adv'(_487,_488)],
 'ble bortskutt kulene' :
  ['TVsmallclAdvP_IncorpAdv' :
    base ++ 'InhObj' ++ 'TV_smallcl_AdvP' ++
     'Incorp_Adv'(_622,_623) ++ 'Pass'],
 'Per skyter kulene varme' :
  ['TVsmallclAP' : base ++ 'InhObj' ++ 'TV_smallcl_AP'],
 'ble skutt kulene varme' :
  ['TVsmallclAP_Pass' :
    base ++ 'InhObj' ++ 'TV_smallcl_AP' ++ 'Pass'],
 'Per skyter varm kulene' :
  ['TVsmallclAP_PredicMvt' :
    base ++ 'InhObj' ++ 'TV_smallcl_AP' ++
     'PredicMvt'(_792,_793)],
 'blir skutt varm kulene' :
  ['TVsmallclAP_PredicMvt' :
    base ++ 'InhObj' ++ 'TV_smallcl_AP' ++
     'PredicMvt'(_930,_931) ++ 'Pass'],
 'Per varmskyter kuler' :
  ['TVsmallclAP_IncorpA' :
    base ++ 'InhObj' ++ 'TV_smallcl_AP' ++
     'Incorp_A'(_1057,_1058)],
 'ble varmskutt kulene' :
  ['TVsmallclAP_IncorpA_Pass' :
    base ++ 'InhObj' ++ 'TV_smallcl_AP' ++
     'Incorp_A'(_1181,_1182) ++ 'Pass'],
 'Per skyter kulene i lufta' :
  ['TVsmallclPP' : base ++ 'InhObj' ++ 'TV_smallcl_PP'],
 'ble skutt kulene i lufta' :
  ['TVsmallclPP_Pass' :
    base ++ 'InhObj' ++ 'TV_smallcl_PP' ++ 'Pass'],
 'Per skyter kuler p\214 Jon' :
  ['InhObj_PPad' :
    base ++ 'InhObj' ++ 'PP_ad'(_1337,_1338)],
 'blir skutt kuler p\214 Jon' :
  ['InhObj_PPad_Pass' :
    base ++ 'InhObj' ++ 'PP_ad'(_1454,_1455) ++ 'Pass'],
 'Per maler veggen med' :
  ['InhObj_PPad_loadalt' :
    base ++ 'InhObj' ++ 'PP_ad'(_1929,_1930) ++ load_alt],
 'blir malt veggen med' :
  ['InhObj_PPad_loadalt_Pass' :
    base ++ 'InhObj' ++ 'PP_ad'(_2049,_2050) ++ load_alt ++ 'Pass'],
 'Per maler seg med' :
  ['InhObj_PPad_loadalt_DOrefl' :
    base ++ 'InhObj' ++ 'PP_ad'(_2170,_2171) ++ load_alt ++
     'DO_refl'],
 'Per synger julen inn' :
  ['IVsmallclAdvP' : base ++ 'IV_smallcl_AdvP'],
 'ble sunget julen inn' :
  ['IVsmallclAdvP_Pass' : base ++ 'IV_smallcl_AdvP' ++ 'Pass'],
 'Per synger inn julen' :
  ['IVsmallclAdvP_PredicMvt' :
    base ++ 'IV_smallcl_AdvP' ++ 'PredicMvt'(_2324,_2325)],
 'blir sunget inn julen' :
  ['IVsmallclAdvP_PredicMvt_Pass' :
    base ++ 'IV_smallcl_AdvP' ++ 'PredicMvt'(_2458,_2459) ++ 'Pass'],
 'innsynge julen' :
  ['IVsmallclAdvP_IncorpAdv' :
    base ++ 'IV_smallcl_AdvP' ++ 'Incorp_Adv'(_2587,_2588)],
 'ble innsunget julen' :
  ['IVsmallclAdvP_IncorpAdv_Pass' :
    base ++ 'IV_smallcl_AdvP' ++ 'Incorp_Adv'(_2718,_2719) ++ 'Pass'],
 'Per g\214r seg bort' :
  ['IVsmallclAdvP_DOrefl' : base ++ 'IV_smallcl_AdvP' ++ 'DO_refl'],
 'Per g\214r skoene skeive' :
  ['IVsmallclAP' : base ++ 'IV_smallcl_AP'],
 'blir g\214tt skoene skeive' :
  ['IVsmallclAP_Pass' : base ++ 'IV_smallcl_AP' ++ 'Pass'],
 'Per sitter flat putene' :
  ['IVsmallclAP_PredicMvt' :
    base ++ 'IV_smallcl_AP' ++ 'PredicMvt'(_2895,_2896)],
 'ble sittet flat putene' :
  ['IVsmallclAP_PredicMvt_Pass' :
    base ++ 'IV_smallcl_AP' ++ 'PredicMvt'(_3030,_3031) ++ 'Pass'],
 'flatsitte puten' :
  ['IVsmallclAP_IncorpA' :
    base ++ 'IV_smallcl_AP' ++ 'Incorp_A'(_3153,_3154)],
 'ble flatsittet puten' :
  ['IVsmallclAP_IncorpA_Pass' :
    base ++ 'IV_smallcl_AP' ++ 'Incorp_A'(_3273,_3274) ++ 'Pass'],
 'Per g\214r seg glad' :
  ['IVsmallclAP_DOrefl' : base ++ 'IV_smallcl_AP' ++ 'DO_refl'],
 'Per synger Jon i godt hum\277r' :
  ['IVsmallclPP' : base ++ 'IV_smallcl_PP'],
 'blir sunget i godt hum\277r' :
  ['IVsmallclPP_Pass' : base ++ 'IV_smallcl_PP' ++ 'Pass'],
 'p\2141\277pe familien en sykdom' :
  ['IVsmallclPP_IncorpP' :
    base ++ 'IV_smallcl_PP' ++ 'Incorp_P'(_3440,_3441)],
 'ble p\2141\277pt en sykdom' :
  ['IVsmallclPP_IncorpP_Pass' :
    base ++ 'IV_smallcl_PP' ++ 'Incorp_P'(_3564,_3565) ++ 'Pass'],
 'synge seg i godt hum\277r' :
  ['IVsmallclPP_DOrefl' : base ++ 'IV_smallcl_PP' ++ 'DO_refl'],
 'Per g\214r ut' : ['SuPredAdAdvP' : base ++ 'SuPredAd_AdvP'],
 'ble g\214tt ut' :
  ['SuPredAdAdvP_Pass' : base ++ 'SuPredAd_AdvP' ++ 'Pass'],
 'Fristen utg\214r' :
  ['SuPredAdAdvP_IncorpAdv' :
    base ++ 'SuPredAd_AdvP' ++ 'Incorp_Adv'(_3736,_3737)],
 'Motoren g\214r tom' : ['SuPredAdAP' : base ++ 'SuPredAd_AP'],
 'Per g\214r fra b\214ten' :
  ['SuPredAdPP' : base ++ 'SuPredAd_PP'],
 'Per snakker med Jon' :
  ['PPad' : base ++ 'PP_ad'(_3882,_3883)],
 'ble snakket med Jon' :
  ['PPad_Pass' : base ++ 'PP_ad'(_3996,_3997) ++ 'Pass'],
 'Per taler med Jon om mat' :
  ['PPad_PPad' :
    base ++ 'PP_ad'(_4206,_4207) ++ 'PP_ad'(_4106,_4107)],
 'ble talt med Jon om mat' :
  ['PPad_PPad_Pass' :
    base ++ 'PP_ad'(_4419,_4420) ++ 'PP_ad'(_4319,_4320) ++ 'Pass'],
 'Per omtaler v\276ret' :
  ['PPad_IncorpP' :
    base ++ 'PP_ad'(_4636,_4637) ++ 'Incorp_P'(_4533,_4534)],
 'blir omtalt v\276ret' :
  ['PPad_IncorpP_Pass' :
    base ++ 'PP_ad'(_4857,_4858) ++ 'Incorp_P'(_4754,_4755) ++
     'Pass'],
 'Per maler veggen' :
  ['PPad_loadalt' :
    base ++ 'PP_ad'(_5318,_5319) ++ load_alt],
 'blir malt veggen' :
  ['PPad_loadalt_Pass' :
    base ++ 'PP_ad'(_5435,_5436) ++ load_alt ++ 'Pass'],
 'Per maler seg' :
  ['PPad_loadalt_DOrefl' :
    base ++ 'PP_ad'(_5552,_5553) ++ load_alt ++ 'DO_refl'],
 'Per skyter Jon i hodet' :
  ['PartWhtoDO' :
    base ++ 'PP_ad'(_5665,_5666) ++ 'Part_Wh_to_DO'],
 'blir skutt i hodet Jon' :
  ['PartWhtoDO_Pass' :
    base ++ 'PP_ad'(_5782,_5783) ++ 'Part_Wh_to_DO' ++ 'Pass'],
 'Per skyter seg i foten' :
  ['PartWhtoDO_DOrefl' :
    base ++ 'PP_ad'(_5899,_5900) ++ 'Part_Wh_to_DO' ++ 'DO_refl'],
 'Per g\214r veien' : ['AdvtoDO' : base ++ 'Adv_to_DO'],
 'blir g\214tt veien' :
  ['AdvtoDO_Pass' : base ++ 'Adv_to_DO' ++ 'Pass']]).

xtemplate(2,erg,
['koker vannet' : [basic : base],
 'det koker vann' : [detins : base ++ det_ins],
 'Vannet koker' : ['Promtoea' : base ++ 'Prom_to_ea'],
 'Per koker vannet' : ['Caus' : base ++ 'Caus'],
 'Per koker vannet bort' :
  ['TVsmallclAdvP' : base ++ 'Caus' ++ 'TV_smallcl_AdvP'],
 'ble kokt vannet bort' :
  ['TVsmallclAdvP.Pass' :
    base ++ 'Caus' ++ 'TV_smallcl_AdvP' ++ 'Pass'],
 'Per koker bort vannet' :
  ['TVsmallclAdvP_PredicMvt' :
    base ++ 'Caus' ++ 'TV_smallcl_AdvP' ++
     'PredicMvt'(_142,_143)],
 'ble kokt bort vannet' :
  ['TVsmallclAdvP_PredicMvt_Pass' :
    base ++ 'Caus' ++ 'TV_smallcl_AdvP' ++
     'PredicMvt'(_280,_281) ++ 'Pass'],
 'Per bortkoker vannet' :
  ['TVsmallclAdvP_IncorpAdv' :
    base ++ 'Caus' ++ 'TV_smallcl_AdvP' ++
     'Incorp_Adv'(_412,_413)],
 'ble bortkokt vannet' :
  ['TVsmallclAdvP_IncorpAdv_Pass' :
    base ++ 'Caus' ++ 'TV_smallcl_AdvP' ++
     'Incorp_Adv'(_547,_548) ++ 'Pass'],
 'Per koker kluten ren' :
  ['TVsmallclAP' : base ++ 'Caus' ++ 'TV_smallcl_AP'],
 'ble kokt kluten ren' :
  ['TVsmallclAP_Pass' :
    base ++ 'Caus' ++ 'TV_smallcl_AP' ++ 'Pass'],
'Per koker ren kluten' :
 ['TVsmallclAP_PredicMvt' :
   base ++ 'Caus' ++ 'TV_smallcl_AP' ++
    'PredicMvt'(_717,_718)],
'ble kokt ren kluten' :
 ['TVsmallclAP_PredicMvt_Pass' :
   base ++ 'Caus' ++ 'TV_smallcl_AP' ++
    'PredicMvt'(_855,_856) ++ 'Pass'],
'Per renkoker kluten' :
 ['TVsmallclAP_IncorpA' :
   base ++ 'Caus' ++ 'TV_smallcl_AP' ++ 'Incorp_A'(_982,_983)],
'ble renkokt kluten' :
 ['TVsmallclAP_IncorpA_Pass' :
   base ++ 'Caus' ++.'TV_smallcl_AP' ++ 'Incorp_A'(_1106,_1107) ++
    'Pass'],
'Per koker potetene til gr\277t' :
 ['TVsmallclPP' : base ++ 'Caus' ++ 'TV_smallcl_PP'],
'ble kokt potetene til gr\277t' :
 ['TVsmallclPP_Pass' :
   base ++ 'Caus' ++ 'TV_smallcl_PP' ++ 'Pass'],
'Per koker kjelen inn' :
 ['IVsmallclAdvP' :
   base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AdvP'],
'ble kokt kjelen inn' :
 ['IVsmallclAdvP_Pass' :
   base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AdvP' ++ 'Pass'],
'Per koker inn kjelen' :
 ['IVsmallclAdvP_PredicMvt' :
   base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AdvP' ++
    'PredicMvt'(_1316,_1317)],
'ble kokt inn kjelen' :
 ['IVsmallclAdvP_PredicMvt_Pass' :
   base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AdvP' ++
    'PredicMvt'(_1458,_1459) ++ 'Pass'],
'innkoke kjelen' :
 ['IVsmallclAdvP_IncorpAdv' :
   base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AdvP' ++
    'Incorp_Adv'(_1594,_1595)],
'    ble innkokt kjelen' :
 ['IVsmallclAdvP_IncorpAdv_Pass' :
   base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AdvP' ++
    'Incorp_Adv'(_1732,_1733) ++ 'Pass'],
'Per koker kjelen svart' :
 ['IVsmallclAP' : base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AP'],
'ble kokt kjelen svart' :
 ['IVsmallclAP_Pass' :
   base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AP' ++ 'Pass'],
'Per koker svart kjelen' :
 ['IVsmallclAP_PredicMvt' :
   base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AP' ++
    'PredicMvt'(_1913,_1914)],
'ble kokt svart kjelen' :
 ['IVsmallclAP_PredicMvt_Pass' :
   base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AP' ++
    'PredicMvt'(_2054,_2055) ++ 'Pass'],
'svartkoke kjelen' :
 ['IVsmallclAP_IncorpA' :
   base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AP' ++
    'Incorp_A'(_2184,_2185)],
'ble svartkokt kjelen' :
 ['IVsmallclAP_IncorpA_Pass' :
   base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AP' ++
    'Incorp_A'(_2312,_2313) ++ 'Pass'],
'Per koker seg trett' :
 ['IVsmallclAP_DOrefl' :
   base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AP' ++ 'DO_refl'],
'Per koker hull i taket' :
 ['IVsmallclPP' : base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_PP'],
  'ble kokt hull i taket' :
   ['IVsmallclPP_Pass' :
     base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_PP' ++ 'Pass'],
  'Per koker seg i godt hum\277r' :
   ['IVsmallclPP_DOrefl' :
     base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_PP' ++ 'DO_refl'],
  'Vannet koker bort' :
   ['TVsmallclAdvP_Promtoea' :
     base ++ 'TV_smallcl_AdvP' ++ 'Prom_to_ea'],
  'Vannet er bortkokt' :
   ['TVsmallclAdvP_IncorpAdv_Promtoea' :
     base ++ 'TV_smallcl_AdvP' ++ 'Incorp_Adv'(_2555,_2556) ++
       'Prom_to_ea'],
  'Potetene koker m\277re' :
   ['TVsmallclAP_Promtoea' : base ++ 'TV_smallcl_AP' ++ 'Prom_to_ea'],
  'Potetene m\277rkoker' :
   ['TVsmallclAP_IncorA_Promtoea' :
     base ++ 'TV_smallcl_AP' ++ 'Incorp_A'(_2697,_2698) ++
       'Prom_to_ea'],
  'Potetene koker til gr\277t' :
   ['TVsmallclPP_Promtoea' : base ++ 'TV_smallcl_PP' ++ 'Prom_to_ea'],
  'han reiser seg' :
   ['Caus_DOrefl' : base ++ 'Caus' ++ 'DO_refl'],
  'et t\214rn reiser seg' :
   ['Ergrefl_Promtoea' : base ++ 'Erg_refl' ++ 'Prom_to_ea'],
  'det reiser seg et t\214rn' :
   ['Ergrefl_detins' : base ++ 'Erg_refl' ++ det_ins]]).

xtemplate(3,exp_iv,
['Jon fryser' : [basic : base],
 'Jon fryser til d\277de' : ['SuPredAdPP' : base ++ 'SuPredAd_PP']]).

xtemplate(4,tv,
[basic : [basic : base],
 'blir spist maten' : ['Pass' : base ++ 'Pass'],
 'Per kj\277per henne en hatt' :
  ['FreeIOins' : base ++ 'Free_IO_ins'],
 'Per kj\277per seg en hatt' :
  ['IOrefl' : base ++ 'Free_IO_ins' ++ 'IO_refl'],
 'ble kj\277pt henne en hatt' :
  ['FreeIOins_Pass' : base ++ 'Free_IO_ins' ++ 'Pass'],
 'Per vasker seg' : ['DOrefl' : base ++ 'DO_refl'],
 'Per spiser maten opp' :
  ['TVsmallclAdvP' : base ++ 'TV_smallcl_AdvP'],
 'blir kastet fangen ut' :
  ['TVsmallclAdvP_Pass' : base ++ 'TV_smallcl_AdvP' ++ 'Pass'],
 'Per kaster seg ut' :
  ['TVsmallclAdvP_DOrefl' : base ++ 'TV_smallcl_AdvP' ++ 'DO_refl'],
 'Per spiser opp maten' :
  ['TVsmallclAdvP_PredicMvt' :
    base ++ 'TV_smallcl_AdvP' ++ 'PredicMvt'(_186,_187)],
 'blir spist opp maten' :
  ['TVsmallclAdvP_PredicMvt_Pass' :
    base ++ 'TV_smallcl_AdvP' ++ 'PredicMvt'(_320,_321) ++ 'Pass'],
 'oppspise maten' :
  ['TVsmallclAdvP_IncorpAdv' :
    base ++ 'TV_smallcl_AdvP' ++ 'Incorp_Adv'(_449,_450)],
 'ble oppspist maten' :
  ['TVsmallclAdvP_IncorpAdv_Pass' :
    base ++ 'TV_smallcl_AdvP' ++ 'Incorp_Adv'(_580,_581) ++ 'Pass'],
 'Per vasker huset rent' : ['TVsmallclAP' : base ++ 'TV_smallcl_AP'],
 'blir vasket huset rent' :
  ['TVsmallclAP_Pass' : base ++ 'TV_smallcl_AP' ++ 'Pass'],
 'Per vasker seg ren' :
  ['TVsmallclAP_DOrefl' : base ++ 'TV_smallcl_AP' ++ 'DO_refl'],
 'Per vasker ren huset' :
  ['TVsmallclAP_PredicMvt' :
    base ++ 'TV_smallcl_AP' ++ 'PredicMvt'(_757,_758)],
'blir renspist tallerkenen' :
 ['IVsmallclAP_IncorpA_Pass' :
   base ++ 'DO_del' ++ 'IV_smallcl_AP' ++
    'Incorp_A'(_2832,_2833) ++ 'Pass'],
'han spiser seg mett' :
 ['IVsmallclAP_DOrefl' :
   base ++ 'DO_del' ++ 'IV_smallcl_AP' ++ 'DO_refl'],
'Per drikker Jon under bordet' :
 ['IVsmallclPP' : base ++ 'DO_del' ++ 'IV_smallcl_PP'],
'blir drukket Jon under bordet' :
 ['IVsmallclPP_Pass' :
   base ++ 'DO_del' ++ 'IV_smallcl_PP' ++ 'Pass'],
'han spiser seg i godt hum\277r' :
 ['IVsmallclPP_DOrefl' :
   base ++ 'DO_del' ++ 'IV_smallcl_PP' ++ 'DO_refl'],
'Per spiser p\214 br\277det' :
 ['DOtoPP' : base ++ 'DO_to_PP'(_3008)],
'blir spist p\214 br\277det' :
 ['DOtoPP_Pass' : base ++ 'DO_to_PP'(_3079) ++ 'Pass'],
'Per drikker kaffen varm' :
 ['ObjQual' : base ++ 'ObjQual'(_3160)],
'blir drukket kaffen varm' :
 ['ObjQual_Pass' : base ++ 'ObjQual'(_3258) ++ 'Pass'],
'Per sier noe til Jon' :
 ['PPad' : base ++ 'PP_ad'(_3360,_3361)],
'ble sagt noe til Jon' :
 ['PPad_Pass' : base ++ 'PP_ad'(_3474,_3475) ++ 'Pass'],
'Per sier noe til Jon fra M' :
 ['PPad_PPad' :
   base ++ 'PP_ad'(_3684,_3685) ++ 'PP_ad'(_3584,_3585)],
'tilskrive Jon et brev' :
 ['PPad_IncorpP' :
   base ++ 'PP_ad'(_3901,_3902) ++ 'Incorp_P'(_3798,_3799)],
'ble tilskrevet Per et brev' :
 ['PPad_IncorpP_Pass' :
   base ++ 'PP_ad'(_4121,_4122) ++ 'Incorp_P'(_4018,_4019) ++
    'Pass'],
'Per fyller b\277tta med vann' :
 ['PPad_loadalt' :
   base ++ 'PP_ad'(_4465,_4466) ++ load_alt],
'ble fylt b\277tta med vann' :
 ['PPad_loadalt_Pass' :
   base ++ 'PP_ad'(_4582,_4583) ++ load_alt ++ 'Pass'],
'Per fyller seg med Whisky' :
 ['PPad_loadalt_DOrefl' :
   base ++ 'PP_ad'(_4699,_4700) ++ load_alt ++ 'DO_refl'],
'kaste Jon n\277tter i hodet' :
 ['PartWhtoIO' :
   base ++ 'PP_ad'(_4813,_4814) ++ 'Part_Wh_to_IO'],
'ble kastet Jon n\277tter i hodet' :
 ['PartWhtoIO_Pass' :
   base ++ 'PP_ad'(_4930,_4931) ++ 'Part_Wh_to_IO' ++ 'Pass'],
'kaste seg n\277tter i hodet' :
 ['PartWhtoIO_IOrefl' :
   base ++ 'PP_ad'(_5047,_5048) ++ 'Part_Wh_to_IO' ++ 'IO_refl'],
'kaste seg Jon rundt halsen' :
 ['PartWhtoIO_DOrefl' :
   base ++ 'PP_ad'(_5164,_5165) ++ 'Part_Wh_to_IO' ++ 'DO_refl'],
'Per s\214 en elg komme' : ['AcI' : base ++ 'AcI'],
'ble sett en elg komme' :
 ['AcI_Pass' : base ++ 'AcI' ++ 'Pass'],
'Per s\214 seg gli' :
 ['AcI_DOrefl' : base ++ 'AcI' ++ 'DO_refl'],
'Per sa atS' : [atS : base ++ atS(_5333)],
'Per sa Marit atS' :
 [atS_FreeIOins : base ++ atS(_5467) ++ 'Free_IO_ins'],
'ble sagt Marit atS' :
 [atS_FreeIOins_Pass :
     base ++ atS(_5604) ++ 'Free_IO_ins' ++ 'Pass'],
  'ble sagt atS' :
   [atS_Pass : base ++ atS(_5737) ++ 'Pass'],
  'Per sa atS til Jon' :
   [atS_PPad :
     base ++ atS(_5967) ++ 'PP_ad'(_5855,_5856)],
  'ble sagt atS til Jon' :
   [atS_PPad_Pass :
     base ++ atS(_6201) ++ 'PP_ad'(_6089,_6090) ++ 'Pass'],
  'Per sa hvS' : [hvS : base ++ hvS(_6335)],
  'Per sa Marit hvS' :
   [hvS_FreeIOins : base ++ hvS(_6482) ++ 'Free_IO_ins'],
  'ble sagt Marit hvS' :
   [hvS_FreeIOins_Pass :
     base ++ hvS(_6633) ++ 'Free_IO_ins' ++ 'Pass'],
  'ble sagt hvS' :
   [hvS_Pass : base ++ hvS(_6781) ++ 'Pass'],
  'Per sa hvS til Jon' :
   [hvS_PPad :
     base ++ hvS(_7025) ++ 'PP_ad'(_6909,_6910)],
  'ble sagt hvS til Jon' :
   [hvS_PPad_Pass :
     base ++ hvS(_7272) ++ 'PP_ad'(_7156,_7157) ++ 'Pass'],
  'Per pr\277vde \214inf' : [o_inf : base ++ o_inf],
  'ble pr\277vd \214inf' :
   [o_inf_Pass : base ++ o_inf ++ 'Pass']]).

xtemplate(5,th_tv,
['papiret absorberer vannet' : [basic : base],
 'blir absorbert vannet' : ['Pass' : base ++ 'Pass']]).

xtemplate(6,exp_tv,
['Per liker kaffe' : [basic : base],
 'Per liker kaffen varm' :
  ['Depict' : base ++ 'Depict'(_46)]]).

xtemplate(7,ditv,
['Per gir Jon penger' : [basic : base],
 'ble gitt Jon penger' : ['Pass' : base ++ 'Pass'],
 'Per gir penger' : ['IOdel' : base ++ 'IO_del'],
 'ble gitt penger' :
  ['IOdel_Pass' : base ++ 'IO_del' ++ 'Pass'],
 'Per gir penger til de fattige' :
  ['IOdel_PPad' :
    base ++ 'IO_del' ++ 'PP_ad'(_98,_99)],
 'ble gitt penger til Jon' :
  ['IOdel_PPad_Pass' :
    base ++ 'IO_del' ++ 'PP_ad'(_215,_216) ++ 'Pass'],
 'Per gir seg' :
  ['IOdel_DOrefl' : base ++ 'IO_del' ++ 'DO_refl'],
 'han gir' : ['IOdel_DOdel' : base ++ 'IO_del' ++ 'DO_del'],
 'blir gitt penger' :
  ['IOdel_DOdel_Pass' :
    base ++ 'IO_del' ++ 'DO_del' ++ 'Pass'],
 'Per gir til de fattige' :
  ['IOdel_DOdel_PPad' :
    base ++ 'IO_del' ++ 'DO_del' ++ 'PP_ad'(_383,_384)],
 'Per gir boksen full' :
  ['IVsmallclAP' : base ++ 'IO_del' ++ 'DO_del' ++ 'IV_smallcl_AP'],
 'ble gitt boksen full' :
  ['IVsmallclAP_Pass' :
    base ++ 'IO_del' ++ 'DO_del' ++ 'IV_smallcl_AP' ++ 'Pass'],
 'Per gir familien ut av huset' :
  ['IVsmallclPP' : base ++ 'IO_del' ++ 'DO_del' ++ 'IV_smallcl_PP'],
 'ble gitt familien ut av huset' :
  ['IVsmallclPP_Pass' :
    base ++ 'IO_del' ++ 'DO_del' ++ 'IV_smallcl_PP' ++ 'Pass'],
 'Per gir opp' :
  ['IOdel_DOdel_SuPredAdAdvP' :
    base ++ 'IO_del' ++ 'DO_del' ++ 'SuPredAd_AdvP'],
 'Per gir Jon et slag i hodet' :
  ['PartWhtoIO' : base ++ 'IO_del' ++ 'Part_Wh_to_IO'],
 'ble gitt Jon et slag i hodet' :
  ['PartWhtoIO_Pass' :
    base ++ 'IO_del' ++ 'Part_Wh_to_IO' ++ 'Pass'],
 'Per gir pengene bort' :
  ['TVsmallclAdvP' : base ++ 'IO_del' ++ 'TV_smallcl_AdvP'],
 'ble gitt penger bort' :
  ['TVsmallclAdvP_Pass' :
    base ++ 'IO_del' ++ 'TV_smallcl_AdvP' ++ 'Pass'],
 'Per gir bort penger' :
  ['TVsmallclAdvP_PredicMvt' :
    base ++ 'IO_del' ++ 'TV_smallcl_AdvP' ++
     'PredicMvt'(_694,_695)],
 'ble gitt bort penger' :
  ['TVsmallclAdvP_PredicMvt_Pass' :
    base ++ 'IO_del' ++ 'TV_smallcl_AdvP' ++
     'PredicMvt'(_832,_833) ++ 'Pass'],
 'Per bortgir penger' :
  ['TVsmallclAdvP_IncorpAdv' :
    base ++ 'IO_del' ++ 'TV_smallcl_AdvP' ++
     'Incorp_Adv'(_965,_966)],
 'ble bortgitt penger' :
  ['TVsmallclAdvP_IncorpAdv_Pass' :
    base ++ 'IO_del' ++ 'TV_smallcl_AdvP' ++
     'Incorp_Adv'(_1099,_1100) ++ 'Pass'],
 'Per gir penger ut av landet' :
  ['TVsmallclPP' : base ++ 'IO_del' ++ 'TV_smallcl_PP'],
 'ble gitt gull ut av landet' :
  ['TVsmallclPP_Pass' :
    base ++ 'IO_del' ++ 'TV_smallcl_PP' ++ 'Pass'],
 'Per gir seg god tid' : ['IOrefl' : base ++ 'IO_refl']]).

xtemplate(8,refl,
['Per skammer seg' : [basic : base],
 'Per skammer seg over bilen sin' :
  ['PPad' : base ++ 'PP_ad'(_54,_55)]]).

xtemplate(9,psych,
['irriterer Per vinden' : [basic : base],
 'Vinden irriterer Per' : ['Promtoea' : base ++ 'Prom_to_ea'],
 'irriteres Per av vinden' : ['Pass' : base ++ 'Pass'],
 'Per irriterer seg' :
  ['DOdel_ExpMvt' : base ++ 'DO_del' ++ 'ExpMvt'],
 'Per irriterer seg over vinden' :
  ['ExpMvt_DOtoPP' : base ++ 'ExpMvt' ++ 'DO_to_PP'(_77)],
 'Jon irriterer Per med snakk' :
  ['DOtoPP_Caus' : base ++ 'DO_to_PP'(_151) ++ 'Caus'],
 'Jon irriterer Per' :
  ['DOdel_Caus' : base ++ 'DO_del' ++ 'Caus'],
 'irriteres Per av Jon' :
  ['DOdel_Caus_Pass' : base ++ 'DO_del' ++ 'Caus' ++ 'Pass'],
 'Vinden irriterer' :
  ['Expdel_Promtoea' : base ++ 'Exp_del' ++ 'Prom_to_ea'],
 'irriterer Per atS' : [atS : base ++ atS(_393)],
 'atS irriterer Per' :
  [atS_Antiexp : base ++ atS(_526) ++ 'Anti_exp'],
 'Per irriterer seg over atS' :
  [atS_ExpMvt_DOtoPP :
    base ++ atS(_663) ++ 'ExpMvt' ++ 'DO_to_PP'],
 'fryktes atS' :
  [atS_Pass : base ++ atS(_797) ++ 'Pass'],
 'interesserer Per hvS' : [hvS : base ++ hvS(_931)],
 'hvS interesserer Per' :
  [hvS_Antiexp : base ++ hvS(_1078) ++ 'Anti_exp'],
 'Per interesserte seg for hvS' :
   [hvS_ExpMvt_DOtoPP :
     base ++ hvS(_1229) ++ 'ExpMvt' ++ 'DO_to_PP'],
  'fryktes hvS' :
   [hvS_Pass : base ++ hvS(_1377) ++ 'Pass'],
  'irriterer Per \214inf' : [o_inf : base ++ o_inf],
  '\214inf irriterer Per' :
   [o_inf_Antiexp : base ++ o_inf ++ 'Anti_exp'],
  'Per irriterer seg over \214inf' :
   [o_inf_ExpMvt_DOtoPP :
     base ++ o_inf ++ 'ExpMvt' ++ 'DO_to_PP'],
  'mislikes \214inf' : [o_inf_Pass : base ++ o_inf ++ 'Pass']]).

xtemplate(10,raisv1,
['viser seg at Jon er kompetent' : [basic : base],
 'det viser seg at Jon er syk' : [detins : base ++ det_ins],
 'Jon viser seg \214 v\276re syk' :
  [atStoClinf_SubjRais :
    base ++ atS_to_Cl_inf(_227) ++ 'SubjRais'(_85)],
 'Jon viser seg syk' :
  [atStoClinf_SubjRais_o_inftopredicAP :
    base ++ atS_to_Cl_inf(_526) ++ 'SubjRais'(_384) ++
     o_inf_to_predic_AP]]).

xtemplate(11,raisv2,
['virker som om Per er syk' : [basic : base],
 'det virker som om Per er syk' : [detins : base ++ det_ins],
 'Per virker som om han er syk' :
  ['RaisCop' : base ++ 'RaisCop'],
 'Per virker syk' :
  ['PropcompltoSC_SubjRais' :
    base ++ 'Prop_compl_to_SC'(_242) ++ 'SubjRais'(_99)]]).

xtemplate(12,raisv3,
['synes meg at Per er syk' : [basic : base],
 'det synes meg at Per er syk' : [detins : base ++ det_ins],
 'det synes at Per er syk' :
  ['IOdel_detins' : base ++ 'IO_del' ++ det_ins],
 'det synes meg som om Per er syk' :
  [atStopropcompl_detins :
    base ++ atS_to_prop_compl ++ det_ins],
 'det synes som om Per er syk' :
  ['IOdel_atStopropcompl_detins' :
    base ++ 'IO_del' ++ atS_to_prop_compl ++ det_ins],
 'Per synes meg som om han er syk' :
  [atStopropcompl_RaisCop :
    base ++ atS_to_prop_compl ++ 'RaisCop'],
 'Per synes som om han er syk' :
  ['IOdel_atStopropcompl_RaisCop' :
    base ++ 'IO_del' ++ atS_to_prop_compl ++ 'RaisCop'],
 'Per synes meg \214 v\276re syk' :
  [atStoClinf_SubjRais :
    base ++ atS_to_Cl_inf(_319) ++ 'SubjRais'(_177)],
 'Per synes \214 v\276re syk' :
  ['IOdel_atStoClinf_SubjRais' :
    base ++ 'IO_del' ++ atS_to_Cl_inf(_615) ++
     'SubjRais'(_473)],
 'Per synes meg syk' :
  [atStoClinf_SubjRais_o_inftopredicAP :
    base ++ atS_to_Cl_inf(_917) ++ 'SubjRais'(_775) ++
     o_inf_to_predic_AP],
 'Per synes syk' :
  ['IOdel_atStoClinf_SubjRais_o_inftopredicAP' :
    base ++ 'IO_del' ++ atS_to_Cl_inf(_1216) ++
     'SubjRais'(_1074) ++ o_inf_to_predic_AP]]).

xtemplate(13,raisv4,
['late til at Per er syk' : [basic : base],
 'det later til at Per er syk' : [detins : base ++ det_ins],
 'Per later til \214 v\276re syk' :
  [atStoCLinf_SubjRais :
    base ++ atS_to_Cl_inf(_227) ++ 'SubjRais'(_85)]]).

xtemplate(14,depict,
['forestille seg Per' : [basic : base],
 'Jon forestiller seg at Per er frisk' :
  [atS : base ++ atS(_66)],
 'Jon forestiller seg Per frisk' :
  [atStoSC : base ++ atS(_199) ++ atS_to_SC]]).

xtemplate(15,smallcl,
['man anser Jon for \214inf' : [basic : base],
 'blir ansett Jon for \214inf' : ['Pass' : base ++ 'Pass'],
 'man anser Jon som skyldig' :
  [foro_inftopredicsom : base ++ for_oinf_to_predic_som],
 'blir ansett Jon som skyldig' :
  [foro_inftopredicsom_Pass :
    base ++ for_oinf_to_predic_som ++ 'Pass']]).

xtemplate(16,iv_loc,
['Per bor i byen' : [basic : base]
 'ble bodd i byen': [['Pass': base ++ Pass']]).

xtemplate(17,refl_loc,
['Per oppholder seg her' : [basic : base]]).

xtemplate(18,tv_loc,
['Per setter vasen p\214 bordet' : [basic : base],
 'blir satt vasen p\214 bordet' : ['Pass' : base ++ 'Pass']]).

xtemplate(19,ind_arg,
['Jon stoler p\214 Per' : [basic : base],
 'blir stolt p\214 Per' : ['Pass' : base ++ 'Pass'],
 'Per lurer p\214 omS' : [omS : base ++ omS(_84)]]).

xtemplate(20,ind_arg_refl,
['Per forvisser seg om' : [basic : base]]).

xtemplate(21,weather,
['sn\277r' : [basic : base],
 'det sn\277r' : [detins : base ++ det_ins],
 'det regner store dr\214per' :
  ['InhObj_detins' : base ++ 'InhObj' ++ det_ins]]).

xtemplate(22,measure,
['steinen veier 3 kg' : [basic : base]]).

xtemplate(23,repr,
['bildet forestiller Per' : [basic : base]]).

xtemplate(24,appellat1,
['Jon utnevner Per til sjef' : [basic : base],
 'blir utnevnt Per til sjef' : ['Pass' : base ++ 'Pass'],
 'Per utnevner seg til sjef' : ['DOrefl' : base ++ 'DO_refl']]).

xtemplate(25,appellat2,
['Per kaller Jon et geni' : [basic : base],
 'blir kalt Jon et geni' : ['Pass' : base ++ 'Pass'],
 'Per kaller seg et geni' : ['DO_refl' : base ++ 'DO_refl'],
 'Per kaller Jon for et geni' :
  ['PP' : base ++ 'PP'(_105)],
 'blir kalt Jon for et geni' :
  ['PP_Pass' : base ++ 'PP'(_280) ++ 'Pass'],
 'Per kaller seg for et geni' :
  ['PP_DO_refl' : base ++ 'PP'(_456) ++ 'DO_refl']]).

xtemplate(26,refl_manner,
['Per oppf\277rer seg bra' : [basic : base]]).


xtemplate(27,erg_ditv,
['venter jon en ulykke' : [basic : base],
 'det venter jon en ulykke' : [detins : base ++ det_ins],
 'en ulykke venter jon' : ['Promtoea' : base ++ 'Prom_to_ea']]).
```

```
xtemplate(1,iv,
['Per skyter' : [basic : base],
 'skytbar':
 ['InhObj_Possibility20' : base ++ 'InhObj' ++ 'Possibility20'],
 'gangbar':
 ['AdvtoDO_Possibility20' : base ++ 'Adv_to_DO' ++ 'Possibility20'],
 'gangbarhet' :
 ['AdvtoDO_Possibility20_Quality10 : base ++ 'Adv_to_DO' ++ 'Possibility20'
    ++ 'Quality10'(het10,'Comb')],
 'ugangbar':
 ['AdvtoDO_Possibility20_Neg26' : base ++ 'Adv_to_DO' ++ 'Possibility20'++ 'Neg26' ],
 'ugangbarhet' :
 ['AdvtoDO_Possibility20_Neg26_Quality10 : base ++ 'Adv_to_DO' ++ 'Possibility20' ++ 'Neg26'
    ++ 'Quality10']
 'pratsom' :
 ['TypProperty24' : base ++ 'TypProperty24'],
 'pratsomhet' :
 ['TypProperty24_Quality10: base ++ 'TypProperty24' ++ 'Quality10'],
  'beskyte':['bealt28': base ++ 'be_alt28'],
  'beskytning': ['bealt28_Process1: base ++ 'be_alt28'++ 'Process1'],
 'bedervelse':
 ['Process1': base ++ 'Process1'],
 'løp':
 ['Event4': base ++ 'Event4'],
 'løp':
 ['Event5': base ++ 'Event5'],
 'løper':
 'Agent6': base ++ 'Agent6'],
 'arbeider':
 [Agent7': base ++ 'Agent7'],
 'løping':
 ['Process1': base ++ 'Process1'],
 'løping':
 ['Manner14': base ++ 'Manner14'],
 'arbeid':['Result3': base ++ 'Result3']]).

 xtemplate(2,erg,
['koker vannet' : [basic : base],
 'kokbar' :
 ['Possibility20' : base ++ 'Caus' ++ 'Possibility20'],
 'kokbarhet':
 ['Possibility20_Quality10': base ++ 'Caus' ++ 'Possibility20' ++ 'Quality10'],
 'brennbar' :
 ['Potentiality21' : base ++ 'Potentiality21'],
 'brennbarhet':
 ['Potentiality21_Quality10': base ++ 'Potentiality' ++ 'Quality10'],
 'koking':
 ['Process1': base ++ 'Process1']
 'brann':
 ['Process2': base ++ 'Process2'],
 'kokk':['Agent7': base ++ 'Caus' ++ 'Agent7']]).

 xtemplate(3,exp_iv,
['Jon fryser':[basic : base],
 'frysning':['Result3': base ++ 'Result3']]).

 xtemplate(4,tv,
[basic : [basic : base],
 'spiselig':
 [Possibility20 : base ++ 'Possibility20'],
 'spiselighet' :
 ['Possibility20_Quality10': base ++ 'Possibility20' ++ 'Quality10'],
 'uspiselig':
 ['Possibility20_Neg(26)': base ++ 'Possibility20' ++ 'Neg26'],
 'uspiselighet':
 ['Possibility20_Neg(26)_Quality10 : base ++ 'Possibility20'++ 'Neg26'
    ++ 'Quality10'],
 'lesbar' :
 ['Possibility20' : base ++ 'Possibility20'],
 'lesbarhet' :
 ['Possibility20_Quality10': base ++ 'Possibility20' ++ 'Quality10'],
 'ulesbar':
 ['Possibility20_Neg26' :   base ++  'Possibility20'++ 'Neg26'],
 'ulesbarhet' :
 ['Possibility20_Neg26_Quality10' : base ++  'Possibility20' ++ 'Neg26' ++ 'Quality10'],
 'bygning' :
 [Result3' : base ++ 'Result3'],
 'bebygge': ['bealt': base ++ 'be_alt28'],
 'bebyggelse':
 ['PPad_bealt_Result3' : base ++ 'PP_ad' ++ be_alt ++ 'Result3'],
 'bebyggelse':[''PPad_bealt_Process1: base ++'PP_ad'++ be_alt ++ 'Process1'],
 'bygging':
 ['Process1': base ++ 'Process1'],
 'forfatter': ['Agent6': base ++ 'Agent6'],
 'forfatterskap':['Totalityof13': base ++ 'Totalityof13'],
 'fortelling': ['TP15': base ++ 'TP15'],
 'l rdom': ['Totalityof27': base ++ 'Totalityof27']]).
 xtemplate(5,th_tv,
['papiret absorberer vannet' : [basic : base],
 'absorberbar' :
 ['Potentiality21' : base ++ 'Potentiality21'],
 'inabsorberbar':
 ['Potentiality21_Neg26': base ++ 'Potentiality21' ++ 'Neg26'],
 'absorberbarhet':
 ['Potentiality21_Quality10': base ++ 'Potentiality21 ++ 'Quality10'],
 'inabsorberbarhet':
 ['Potentiality21_Neg26_Quality10': base ++ 'Potentiality21' ++ 'Neg26' ++ 'Quality10'],
 'absorbering':
 ['Process1': base ++ 'Process1']]).


xtemplate(6,exp_tv,
['Per hater musikk' : [basic : base],
 'hat':['Emotion17': base ++ 'Emotion17']]).
 'hatsk':['Emotion17_sk31':


xtemplate(7,ditv,
['Per gir Jon penger' : [basic : base],
 'gave' :
 ['Result3': base ++ 'Result3']]).


xtemplate(8,refl,
['Per skammer seg' : [basic : base],
'skam':
['Emotion8': base ++ ''Emotion8'],
'skammelig':
['Potentiality22': base ++ 'PP_ad'++ 'Potentiality22']]).


xtemplate(9,psych,
['irriterer Per vinden' : [basic : base],
'irritabel':
['ExpMvt_Potentiality23': base ++ 'ExpMvt' ++ 'Potentiality23'],
'irritasjon':
['ExpMvt_DOtoPP_Emotion8': base ++ 'ExpMvt' ++ 'DO_to_PP' ++ 'Emotion8'],
'irritert':['ExpAdj29':base ++ 'DO_del' ++ 'ExpAdj29'],
'irritert over':['ExpAdj29_DOtoPP': base ++ 'ExpAdj29' ++ 'DO_to_PP'],
'ergerlig':
['Promtoea_Potentiality22': base ++ 'Prom_to_ea' ++ 'Potentiality22'],
'ergerlig':
```

```
['ExpMvt_ExpAdj29':base ++ 'ExpMvt' ++ 'ExpAdj29']]).

xtemplate(16,iv_loc,
['Per bor i byen' : [basic : base],
 'bebo':['bealt28': base ++ 'be_alt28'],
 'beboelig':
 ['bealt_Possibility20'': base ++ be_alt ++ 'Possibility20'],
 'beboelse':
 ['bealt_Result3: base ++ be_alt ++ 'Result3']]).


xtemplate(17,refl_loc,
['Per oppholder seg her' : [basic : base],
 'opphold':
 ['Process1': base ++ 'Process1']]).


xtemplate(26,refl_manner,
['Per oppf\277rer seg bra' : [basic : base]]).
 'oppførsel':
 ['Manner14': base ++ 'Manner14']]).
```

App. 7

1.6.89

I. The derivational rules.

For each process, we first provide an informal, theory neutral description, then a semi-formal description in the chosen GB-associated terms, then the way in which the rule will be entered in a template menu, then an example of sentences or expressions accepted by the input and output templates, and finally whatever comments are in order. In the semi-formal statement, we include only as much detail as is necessary. Italicized expressions are used as placeholders for lambda-bound variables, indicating what type of item will occur there.

DO(direct object)-deletion:
Informal description:
Remove *direct object* from the frame, replacing it with an implicit argument. A semantic effect is that Aspect, whenever *resultative* in the input, changes to *durative*.
Semiformal description:
In SAF, replace <X,np,gov> with <X,rp,implarg>.
In SemProp->Aspect, replace 'resultative' with 'durative'. Condition: there is no <X,np,io>.

Rule statement: DO_del
[Ex. spise fisken -> spise]

Object-to-PP:
Informal description:
Remove *direct object* from the frame, replacing it with a PP. A semantic effect is that Aspect, whenever *resultative* in the input, changes to *durative*.
Semiformal description:
In SAF, replace <x,np,gov' with <X,np,pgov/*prep*>. Condition: there is no <X,np,io>.
In SemProp->Aspect, replace 'resultative' with 'durative'.
[Ex. spise brødet -> spise på brødet]

Rule statement: DO_to_PP(arg)
    where arg is either a preposition or the default 'pgov'

IO-deletion:
Informal description:
Remove IO from the frame, replacing it with an implicit argument.
Semiformal description:
In SAF, replace <X,np,io> with <X,rp,implarg>.

Rule statement: IO_del

[Ex. gi Jon penger -> gi penger]

Free IO-insertion:
Informal description:
Introduce a 'free' IO, i.e. an IO whose role is not central to the verb in question.
Semiformal description:
In SAF, provided there is a <X,np,gov>, insert <ben,np,io>.

Rule statement: Free_IO_ins
[Ex. slakte en sau-> slakte Esau en sau]

IV-SmallClause-formation:
(covers **IV-smallcl_AP, IV-smallcl_AdvP** and **IV-smallcl_PP**)
Informal description:
In the frame of an intransitive verb, add a direct object and an AP, AdvP or PP predicated of the object. In LLF, if the input construction is p and the added predication relation is q, the new construction is 'p cause q'.
Semiformal description:
In SAF, provided there is no <X,np,gov>, insert <scsu,np,gov> and <*pred*, *cat*, predic>, where *pred* is either of the values 'degprd' or (as the default) 'prd', and *cat* is a specified head or, as the default, the category 'ap', 'advp' or 'pp' according to whether the rule is IV_smallcl_AP, IV_smallcl_AdvP or IV_smallcl_PP.
In LLF, ...[to be supplied]

Rule statement: IV_smallcl_AP(arg1,arg2)
where arg1 is 'degprd' or 'prd' and arg2 is *adj*_a or ap.
IV_smallcl_AdvP(arg1,arg2)
where arg1 is 'degprd'or 'prd' and arg2 is *adv*_adv or advp.
IV_smallcl_PP(arg1,arg2)
where arg1 is 'degprd' or 'prd' and arg2 is *prep*_p or pp.

In each case the default is the option mentioned last.
[Ex. skyte -> skyte magasinet tomt, spise -> spise tallerkenen tom]

Comment: When a single word is entered as category, this is interpreted in Interpretation as a *phrase* headed by the word in question.

TV-SmallClause-formation:
(covers **TV-smallcl_AP, TV-smallcl_AdvP** and **TV-smallcl_PP**)
Informal description:
In the frame of a transitive verb, add an AP, AdvP or PP predicated of the object. In the LLF, if the input construction is p and the added predication relation is q, the new construction is 'p cause q'.
Semiformal description:
In SAF, change <X,np,gov> into <tvscsu,np,gov>, and add <*pred*, *cat*, predic>, where *pred* is either of the values 'degprd' or (as the default) 'prd', and *cat* is a

specified head or, as the default, the category 'ap', 'advp' or 'pp' according to whether the rule is TV-smallcl_AP, TV-smallcl_AdvP or TV-smallcl_PP.
In LLF, ...[to be supplied; it is assumed that whatever role is covered by X in the original DO is preserved in LLF]

Rule statement: TV_smallcl_AP(arg1,arg2)
where arg1 is 'degprd' or 'prd' and arg2 is *adj*_a or ap.
TV_smallcl_AdvP(arg1,arg2)
where arg1 is 'degprd' or 'prd' and arg2 is *adv*_adv or advp.
TV_smallcl_PP(arg1,arg2)
where arg1 is 'degprd' or 'prd' and arg2 is *prep*_p or pp.
In each case the default value is the last one.
[Ex. sparke ballen - sparke ballen flat]

Comment: When a single word is entered as category, this is interpreted in Interpretation as a *phrase* headed by the word in question.

Depictive small clause:
Informal description:
Add a predicative AP or PP to a DO, with the interpretation that the verb takes 'DO+predic' as a propositional argument, with no causative interpretation.
Semiformal description:
Turn <X,np,gov> into <scsu,np,gov>, and add <prd,*cat*,predic>.
In LLF, 'DO+predic' is interpreted as a propositional argument, with no causative interpretation.

Rule statement: Depict(arg)
where arg is the category of predic, being either pp or (default) ap.
[Ex. like kaffen -> like kaffen varm]

Accusative with infinitive:
Informal description:
Semiformal description:
Turn <th,at_S, gov> into <scsu,np,gov>, and add <prd,inf,predic>.
In LLF, there is no change.

Rule statement: AcI
[Ex.: se at Jon kommer-> se Jon komme]

Object qualification:
Informal description:
An AP or PP is added as a qualification of a DO, while the DO retains its role relative to the verb. There is no causative interpretation.
Semiformal description:
If there is a <X,np,gov>, add <prd,*cat*,adjct>.

Rule statement: ObjQual(arg)
where arg is the category (pp or (default) ap).
[Ex.: drikke kaffen -> drikke kaffen varm]

Comment:
The frame-independent character of the added constituent is marked through the functional label 'adjct' (=adjunct). It's conceivable that this is a purely syntactic process.

Cognate object:
Informal description:
In the frame of an intransitive verb, add a direct object which expresses the type of act which generally instantiates the verb.
Semiformal description:
In a SAF with no <X,np,gov>, add an argument '<cogob,np,gov>'.
In LLF, ...[to be supplied].

Rule statement: CogObj
[Ex. dø -> dø en behagelig død]

Inherent object:
Informal description:
In the frame of an intransitive verb, add a direct object which expresses the type of material which goes into the act instantiating the verb.
Semiformal description:
In a SAF with no <X,np,gov>, add an argument '<inherob,np,gov>'.

Rule statement: InhObj
[Ex. spytte spytt, skyte plastikkuler, male maling, harke slim]

load-alternation:
Informal description:
In a frame with an inherent object and a locational type of PP, turn the NP of that PP into a direct object, and the inherent object into a PP. Semantically, Aspect now becomes resultative.
Semiformal description:
In SAF, turn <inherob,np,gov> into <inherob,np,med> and <target, np,pgov> into <target,np, gov>.
In the semantics, specify SemPrep-> Aspect -> Resultative.

Rule statement: load_alt
[Ex. laste høy på vogna -> laste vogna med høy; fylle vann i bøtta -> fylle bøtta med vann]

Comment:

The process resembles be-alternation in the operation on the frame, a differénce being that the latter does not yield resultative aspect.
The term 'pgov' in the structural description is met by 'pgov' as well as any specific preposition (what count as prepositions being provided in a separate list).
If no <inherob,np,gov> is present in the input template, the rule still applies with regard to the second argument, yielding alternations like
[Ex. laste på vogna -> laste vogna; fylle i bøtta -> fylle bøtta].

be-alternation:
Informal description:
If the frame contains an inherent object and a directional or locative PP, turn the PP into an NP and the inherent object into a PP, and affix be to the verb.
Semiformal description:
In SAF, turn <inherob,np,gov> into <inherob,np,med> and <target, np, pgov> into <target,np, gov>, and prefix be to the verb.

Rule statement: be_alt
[Ex. skyte plastikkuler på Jon -> beskyte Jon med plastikkuler.]

Comment:
This rule derives items belonging to new lexemes.
The process closely resembles Load-alternation, but it has no aspectual part like Load-alternation.
The term 'pgov' in the structural description is met by 'pgov' as well as any specific preposition (what count as prepositions being provided in a separate list).
If no <inherob,np,gov> is present in the input template, the rule still applies with regard to the second argument, yielding alternations like
[Ex. synge om Trondheim -> besynge Trondheim]

Part-whole-to-DO:
Informal description:
If an intransitive verb V has a frame where it takes a PP with 'target' role, then it also has a frame where it takes a DO preceding the PP, where the DO expresses a *whole* relative to which the PP denotes a part.
Semiformal description:
In a SAF with no <X,np,gov>, but with an argument <target,pp,pp_arg>, turn this argument into 'part,pp,pp_arg> and *add* an argument <whole,np,gov>.

Rule statement: Part_Wh_to_DO
[Ex. spytte i ansiktet til Jon -> spytte Jon i ansiktet.]

Part-whole-to-IO:
Informal description:
If a transitive verb V has a frame where it takes a PP with 'target' role, then it also has a frame where it takes an IO expressing a *whole* relative to which the PP denotes a part.

Semiformal description:
In a SAF with no <X,np,io>, but with an argument <Y,np,io> and <target,pp,pp_arg>, turn the latter argument into 'part,pp,pp_arg> and *add* an argument <whole,np,io>.

Rule statement: Part_Wh_to_IO
[Ex. kaste nøtter i hodet på Jon -> kaste Jon nøtter i hodet.]

Adv-to-DO:
Informal description:
An adverbial becomes a DO.
Semiformal description:
In a SAF with no 'gov', and with an argument whose role is 'path', specify this argument as '<path,np,gov>'.

Rule statement: Adv_to_DO
[Ex. gå veien, jump the fence]

Causativization with non-ea:
Informal description:
If a verb has a single, non-agentive argument, then add an agentive argument interpreted as causer of the activity expressed by the original construction.
Semiformal description:
In a SAF with no ea and with a gov, add an argument <agent,np,ea>.
In LFF, if the representation of the input is 'p' and the introduced agent is represented as 'a', then the new SAF is represented as 'a cause p'.

Rule statement: Caus
[Ex. ... koker vannet -> Jon koker vannet]

Comment:
Both individuals and events can act as causes; cf. the LLF of small clauses.

Causativization with ea:
Informal description:
If a verb has a single argument, then add an agentive argument interpreted as causer of the activity expressed by the original construction.
Semiformal description:
In a SAF with no gov, turn <X,np,ea> into <X,np,gov>, and add an argument '<agent,np,ea>'.
In LFF, if the representation of the input is 'p' and the introduced agent is represented as 'a', then the new SAF is represented as 'a cause p'.

Rule statement: Ea_caus
[Ex. the horse walked -> John walked the horse]

Comment:
Both individuals and events can act as causes; cf. the LLF of small clauses.

DO-reflexivization:
Informal description:
A DO is reduced to the short' reflexive form seg, and the construction means that the referent of the subject performs the act expressed by the verb upon itself.
Semiformal description:
In SAF, turn <X,np,gov> into '<norole,seg,refl>.
In LLF, specify the referent of the initial gov as identical to the referent of ea.

Rule statement: DO_refl
[Ex. Jon vasker NP -> Jon vasker seg]

Comment:
In these constructions, the 'object' role is still understood, a circumstance which is expressed in LLF. They differ from the type skamme seg, where no object role is understood.

IO-reflexivization:
Informal description:
An IO is reduced to the short' reflexive form seg, and the construction means that the referent of the subject performs the act expressed by the verb with itself as bene- or malefactive part.
Semiformal description:
In SAF, turn <X,np,io> into '<norole,seg,refl>.
In LLF, specify the referent of the initial io as identical to the referent of ea.

Rule statement: IO_refl
[Ex. Jon kjøpte NP en frakk -> Jon kjøpte seg en frakk; Jon unte NP en ferie -> Jon unte seg en ferie]

Comment:
Like in the DO-reflexivization constructions, the 'indirect object' role is still understood, a circumstance which is expressed in LLF. The IOs which undergo this process can be either 'bound' or 'free'.

Passive-short:
Informal description:
The external argument (subject) in the input template is 'demoted' to an implicit argument, and the verb appears in passive form.
Semiformal description:
Turn <X,np,ea> into <X,rp,implarg>.

Rule statement: Pass_sh
[Ex. Jon leser boken -> ... blir lest boken]

Passive-long:
Informal description:
The external argument (subject) in the input template is 'demoted' to a PP with av as preposition, and the verb appears in passive form.
Semiformal description:
Turn <X,np,ea> into <X,np,av>.

Rule statement: Pass_lo
[Ex. Jon leser boken -> blir lest boken av Jon]

Promotion to EA:
Informal description:
A direct object becomes subject.
Semiformal description:
In a SAF with no ea, turn <th,np,gov> into <th,np,ea>.

Rule statement: Prom_to_ea
[Ex. ... koker vännet -> vannet koker]

Demotion from EA:
Informal description:
A subject becomes direct object, as in agentive presentational constructions.
Semiformal description:
In a SAF with no gov, turn <X,np,ea> into <X,np,gov>.

Rule statement: Dem_from_ea
[Ex.: En katt satt i trappen -> satt en katt i trappen.]

Comment:
This process is necessarily followed by Det-insertion. It never reverses the effect of Promotion to EA, or vice versa.

Det-insertion:
Informal description:
If the subject position is empty, insert the expletive det. There are two types of det, one which can alternate with der in certain dialects and corresponds to der in Danish and there in English, and one which corresponds to it in English.
Semiformal description:
In a SAF with no ea, if there is a <X,at_S,gov> or <Y,Z,compl>, insert <norole,det_it,ea>, otherwise insert <norole,det_there,ea>.

Rule statement: det_ins
[Ex. forekommer meg at Jon er syk -> det forekommer meg at Jon er syk, ... satt en katt i trappen -> det satt en katt i trappen]

Comment:
The presence of <u>det</u> requires that <X,np,gov> be indefinite. We leave open whether such a requirement should be built into this rule.
The 'it'-version is required also with weather-verbs and possibly in other cases; since the difference is not critical in Norwegian, the details are left for a language where they matter.

Subject-predicate adjunction_AP/PP/AdvP:
Informal description:
Add an AP, PP or AdvP interpreted as a predicative of the subject. The meaning is either a fullfledged predicate or a degenerate predicate.
Semiformal description:
In SAF, provided there is no <X,np,gov>, insert <*pred*, *cat*, predic>, where *cat* is a specified head or, as the default, the category 'ap', 'advp' or 'pp' according to whether the rule is Subject-predicate adjunction_AP, Subject-predicate adjunction_PP or Subject-predicate adjunction_AdvP, and *pred* is either of the values 'degprd' or (as the default for '_AP') 'prd' or (as the default for the others) 'dir':

In LLF, ...[to be supplied]

Rule statement: SuPredAd_AP(arg1,arg2)
where arg1 is 'degprd' or 'prd' and arg2 is *adj*_a or ap.
SuPredAd_AdvP(arg1,arg2)
where arg1 is 'degprd' or 'dir' and arg2 is *adv*_adv or advp.
SuPredAd_PP(arg1,arg2)
where arg1 is 'degprd' or 'dir' and arg2 is *prep*_p or pp.

In each case the default is the option mentioned last.
[Ex. gå -> gå ut (i gården), sovne -> sovne inn (i en evig søvn), gå tom]

Comment: When a single word is entered as category, this is interpreted in Interpretation as a *phrase* headed by the word in question.
In LLF, when role is 'direction', the effect is that the new item (b) is interpreted as a locative predicate of the subject (a); given that the input frame is represented as 'p', the derived frame is represented as 'p cause b(a)'. The same schema applies if role is 'prd'. If role is 'degpred', the LLF must be decided from case to case.

PP-adjunction:
Informal description:
Add an 'indirect argument' PP in the frame of a verb, where the choice of P may be verb dependent, either as a function of the roles associated with the verb, or as word-idiosyncratic selection. This rule can be reapplied, in principle indefinitely.
Semiformal description:
In SAF, add an argument <*role*,np,pgov/*prep*>, where *role* is among those that count as central relative to the verb. 'pgov' is the default value in the function-slot.

Role being central, it is among those roles which have one unique instantiatión in each frame.
For technical reasons, on each new application of the rule, it is technically a different rule, distinguished from the previous one by the addition of a cipher to the function-value, one larger than in the previous rule. For instance, the first application introduces <*role*,np,pgov1>, the second <*role*,np,pgov2>, etc. (we let absence of cipher be equal to '1'); or <*role*,np,med1> followed by <*role*,np,med2>. (The ciphers are ignored for interpretive purposes.)

Rule statement: PP_ad(arg1,arg2)
where arg1 is a specified role or the default value 'unsp',
and arg2 is a specified preposition or the default value
'pgov'.
PP_ad2(arg1,arg2)
where arg1 is a specified role or the default value 'unsp',
and arg2 is a specified preposition with cipher '2' attached
or the default value 'pgov2'.
PP_ad3(arg1,arg2)
where arg1 is a specified role or the default value 'unsp',
and arg2 is a specified preposition with cipher '3' attached
or the default value 'pgov3'.
etc.

[Ex. snakke -> snakke med Ola -> snakke med Ola om Marit]

Incorporation
(covers **Incorp_A**, **Incorp_P** and **Incorp_Adv**):
Informal description:
The preposition of an indirect argument, or an adjective constituting a predicative AP, or the adverb head of an AdvP with predicative function, is prefixed to the verb.
Semiformal description:
Incorp_A: In a frame {<scsu,np,gov>,<prd/degprd,ap,predic>}, delete the predic argument and incorporate a specified adjective into the verb. The LLF remains unchanged.
Rule statement: Incorp_A(arg)
where arg is a specified adjective.
[Ex. male huset rødt -> rødmale huset]

Incorp_P: If there is no gov in the frame, turn <X,np,pgov/*prep*> into <X,np,gov>; if there is a gov but no io, the result is <X,np,io>. In either case, a specified preposition is incorporated into the verb. The LLF remains unchanged.
Rule statement: Incorp_P(arg)
where arg is a specified preposition.
[Ex. tale om Ola -> omtale Ola; sende penger til Ola -> tilsende Ola penger]

Incorp_Adv: Given an argument <dir/degprd,advp,predic> (whether there is a governee or not), either delete this argument or turn it into <dir,pp,pp_arg>. In either case, incorporate a specified adverb into the verb. The LLF remains unchanged.
Rule statement: Incorp_Adv(arg)
where arg is a specified adverb.
[Eks. sparke ballen bort -> bortsparke ballen; lyse stillingen ut -> utlyse stillingen; sovne inn -> innsovne; vise Per bort fra banen -> bortvise Per fra banen]

Comment:
This rule creates new lexemes. Often the result has a different meaning than the input construction, but this generally seems due to a 'drift' effect which has to be specified word by word. For instance, ødelegge means something different than legge øde, but the 'new' meaning can be understood as a drift applied to the meaning of the original construction. Such cases seem consistent with the persistence principle, the 'drift' counting as a factor on top of the constructional ones in the construction of the meaning.

There still exist cases of apparent Incorporation whose meaning has nothing to do with any conceivable input frame; such cases are not derived by this rule.

Predic-movement:
Informal description:
An adjective or adverb serving as a predicative of the DO is moved to the left of DO.
Semiformal description:
In a SAF with gov, turn <X,Y,predic> into <X,Y,preposed_predic>.

Rule statement: PredicMvt
[Ex.: sparke ballen ut -> sparke ut ballen]

Comment: 'preposed_predic' is a functional label, meaning that the item characterized occurs before the sc_subj. As a functional label, it is a cheater for information which is otherwise tied to the realization level, namely information about precedence relations.

Substitution NP-atS:
Informal description: Replace an NP with an at-clause.
Semiformal description: Turn <th,np,*function*> into <th,at_S,*function*>.

Rule statement: atS(arg)
where arg provides the function (ea, gov, pgov or a specified preposition as p-governor).
[Ex.: (atS(gov): Per sa sannheten -> Per sa at han var syk]

Substitution NP-å-inf:
Informal description: Replace an NP with an å-infinitive. With regard to control properties, the infinitive has three options: being *regularly* controlled (i.e., being

controlled by the closest c-commanding NP), having *marked* control (i.e., being controlled by a subject across an object), and having arbitrary control. This gives three different types of å-infinitives, and thus three sub-rules.
Semiformal description:
åinf_regcontr: Turn <th,np,*function*> into <th,å_inf_regcontr,*function*>.
(Correspondingly for the other types.)
In LLF, the infinitive is treated as a proposition; i.e., this infinitive may be syntactically construed as 'PRO VP'.

Rule statement: åinf_regcontr(arg)
åinf_markcontr(arg)
åinf_arbcontr(arg)
where arg in each case provides the function (ea, gov, pgov or a specified preposition as p-governor).

[Ex.: (åinf_regcontr(gov) Per prøvde skoene -> Per prøvde å gå]

Substitution NP - hvS:
Informal description:
Replace NP with a wh-clause.
Semiformal description:
Turn <th,np,*function*> into <th,hv_S,*function*>.
Rule statement: hvS(arg)
where arg is the function of the replaced NP (being gov, pgov or governed by some specified preposition).
[Ex.: si sannheten -> si hvem som hadde gjort det]

Substitution NP - omS:
Informal description:
Replace NP with an om-clause.
Semiformal description:
Turn <th,np,*function*> into <th,om_S,*function*>.
Rule statement: omS(arg)
where arg is the function of the replaced NP (being gov, pgov or governed by some specified preposition).
[Ex.: lure på svaret -> lure på om du hadde gjort det]

Subject Raising:
Informal description: A small clause subject or a clausal infinitive subject is 'raised' to subject position of the matrix verb.
Semiformal description:
Turn <scsu,np,*function*> into <scsu,e,*function*>, and add <no,np,ea>.

Rule statement: SubjRais(arg)
where arg is the function of the NP 'raised' (i.e., gov or pgov)
[Ex.: late til Per å være syk -> Per later til [e] å være syk;

synes Per å være syk -> Per synes [e] å være syk;
synes Per syk -> Per synes [e] syk]

Comment: We here assume that traces are left by the movement. The alternative of doing without traces also seemsopen, however.

Substitution of å-inf with predic-AP :
Informal description:
An å-inf serving as object predicative is replaced by an AP with the same function.
Semiformal description:
Turn <prd,å_inf,predic> into <prd,ap,predic> .
In LLF, this infinitive behaves as a predicate.

Rule statement: Åinf_to_Predic_AP
[Ex.: synes Jon å være syk -> synes Jon syk]

Substitution of atS with clausal infinitive:
Informal description:
Replace an at-S by a clausal infinitive.
Semiformal description:
Replace the argument <th,at_S,*function*> by <scsu,np,*function*> and <prd,å_inf, predic>.

Rule statement: atS_to_Cl_inf(arg)
where arg is the function of the NP (gov or pgov).
[Ex.: later til at Jon er syk -> later til Jon å være syk;
forekommer meg atJon er syk -> forekommer meg Jon å være syk]

Substitution of propositional complement with small clause structure:
Informal description:
Replace a som- or som om-clause with an NP plus a predicative AP.
Semiformal description:
Replace the argument <prop_arg,som_om_S,compl>, or <prop_arg,som_S,compl> by <scsu,np,gov> and <prd,ap,predic> .
In LLF, the propositional argument is retained.

Rule statement: Prop_compl(arg)_to_SC
where arg is the category label, 'som_om_S' or 'som_S'.
[Ex.: virker som om Jon er syk (or) virker som Jon er syk-> virker Jon syk]

Raising copying
Informal description:
From a propositional argument structure with som or som om, make the subject of the clause into subject of the matrix verb, and leave a pronominal copy in the original position.
Semiformal description:

In a SAF with no ea and a <prop_arg,*cat*,compl>, turn the latter into <prop_arg,*cat*,subj_contr_compl>, and add <no,np,ea>.

Rule statement: RaisCop(arg)
where arg is the category label of the complement, 'som_om_S' or 'som_S'.
[Ex.: virker som (om) Jon er syk -> Jon virker som (om) han er syk]

Anti-extraposition
Informal description:
Move a clausal complement to subject position.
Semiformal description:
Turn <th,at_S,*function*> into <th,at_S,ea>.

Rule statement: Anti_exp(arg)
where arg is the function of the complement, default being gov.
[Ex.: bekymrer meg at Jon kom -> at Jon kom bekymrer meg]

Experiencer movement
Informal description:
Semiformal description:
Turn <exp,np,io> into <exp,np,ea>, and <th,np,gov> into <th,np,over>, and add <norole,seg,refl>.

Rule statement: ExpMvt
[Ex.: irriterer meg katten -> jeg irriterer meg over katten]

Comment:
If the rule applies after DO-del, the rule still applies, deriving alternations like
[Ex.: irriterer meg -> jeg irriterer meg]

Experiencer adjectival construction:
Informal description:
Semiformal description:
Turn <exp,np,io> into <exp,np,ea>, and <th,np,gov> into <th,np,over>, and turn the verb into a participial adjective.

Rule statement: ExpAdj
[Ex.: irriterer meg katten -> jeg er irritert over katten]

Comment:
If the rule applies after DO-del, the rule still applies, deriving alternations like
[Ex.: irriterer meg -> jeg er irritert]

Substitution of for å-inf with predic som :
Informal description:
Semiformal description:

Turn <prd,å_inf,for> into <prd,*cat*,som>, where *cat* is ap or np.
In LLF there is no change.

Rule statement: for_åinf_to_predic_som(arg)
where arg is the category of the input predicative.
[Ex.: anse Jon for å være skyldig -> anse Jon som skyldig ; anse Jon for å være (en) tyv -> anse Jon som (en) tyv]

Ergative seg insertion:
Informal description:
Semiformal description:
In a SAF with <th, np,gov> but without ea, insert a reflexive seg.
Rule statement: Erg_refl
[Ex.: åpner en dør -> åpner seg en dør ; reiser et tårn -> reiser seg et tårn]

Substitution PredicNP - PredicPP:
Informal description:
Replace NP with a PP.
Semiformal description:
Turn <prd,NP,predic> into <prd, PP,predic>
Rule statement: PP(arg)
where arg is the function of the replaced NP.
[Ex.: kalle Jon et geni -> kalle Jon for et geni]

PP 8a APP. 8

German: Basic templates

```
(1) intransitive verb
SAF: <ag,np,ea>
Statement:iv
[Ex:er singt]
'He sings'

(2) ergative verb
SAF: <th,np,gov>
Statement:erg
[Ex:_kocht das Wasser]
'_boils the water'

(3) experiencer intransitive verb
SAF: <exp,np,ea>
Statement:exp_iv
[Ex:er friert]
'He freezes'

(4) transitive verb
SAF: <ag,np,ea>,<th,np,gov>
Statement:tv
[Ex:er kauft einen Hut]
'He buys a hat'

(5) theme transitive verb
SAF: <th,np,ea>,<th,np,gov>
Statement:th_tv
[Ex:das Papier absorbiert die Tinte]
'The paper absorbs the ink'

(6) experiencer transitive verb
SAF: <exp,np,ea>,<th,np,gov>
Statement:exp_tv
[Ex:er mag Kaffee]
'He likes coffee'

(7) ditransitive verb
SAF: <ag,np,ea>,<ben,np,io>,<th,np,gov>
Statement:ditv
[Ex:er gibt ihm Geld]
'He gives him money'

(8) reflexive verb
SAF: <exp,np,ea>,<norole,sich,refl>
Statement:refl
[Ex:er schämt sich]
'He shames refl'

(9) psych1
SAF: <exp,np,io>,<th,np,gov>
Statement:psych1
[Ex:_langweilt ihn das Buch]
'_bores him the book'

(10) psych2
SAF: <exp,np,io>,<th,np,function>
Statement:psych2
[Ex:_gelüstet ihn nach Abenteuern]
'he hungers for adventures'

(11) raisv1
SAF: <norole,sich,refl>,<th,daß_S,gov>
Statement:raisv1
```

```
[Ex:_zeigt sich daß Tom kompetent ist]
'_shows refl that Tom is competent'

(12) raisv2
SAF: <exp,np,io>,<prop_arg,cat,compl>
Statement:raisv2
[Ex:_kommt mir so vor als ob]
'_appears to me as if'

(13) raisv3
SAF: <exp,np,io>,<th,daß_S,gov>
Statement:raisv3
[Ex:_scheint mir daß Tom krank ist]
'_seems to me that Tom is ill'

(14) depictive verb
SAF: <ag,np,ea>,<norole,sich,refl>,<th,np,gov>
Statement:depict
[Ex:er stellt sich Tom vor]
'he imaginesrefl Tom'

(15) small clause
SAF: <ag,np,ea>,<norole,sich,refl>,<th,np,gov>
Statement:smallcl
[Ex:man sieht ihn für einen Idioten an]
'one regards him as an idiot'

(16) intransitive verb locative
SAF: <X,np,ea>,<loc,Y,gov>
Statement:iv_loc(arg1,arg2)
        where arg1 is ag(default) or th, and arg2 is pp(default) or advp
[Ex:er wohnt in der Stadt]
'he lives in town'

(17) reflexive verb locative
SAF: <X,np,ea>,<norole,sich,refl>,<loc,Y,gov>
Statement:refl_loc(arg1,arg2)
        where arg1 is ag(default) or th, and arg2 is pp(default) or advp
[Ex:er hält sich hier in der Stadt]
'he stays here'

(18) transitive verb locative
SAF: <ag,np,ea>,<th,np,gov>,<loc,np,function>
Statement:tv_loc(arg)
        where arg is pgov(default)
[Ex:er stellt die Vase auf den Tisch]
'he puts the vase on the table'

(19) indirect argument
SAF:<th,np,ea>, <X,np,function>
Statement:ind_arg(arg)
    where arg is pgov or case
[Ex: seine Ansicht beruht auf einem Irrtum]
'his opinion rests on an error'

(20) indirect argument reflexive
SAF: <ag,np,ea>,<norole,sich,refl>,<th,np,function>
Statement:ind_arg_refl(arg)
    where arg is pgov or case
[Ex: er vergewissert sich seines Daseins]
'he assures himself of his existence'

(21) weather verb
SAF:0
Statement:weather
[Ex: _schneit]
'_snows'
```

```
(22) measure verb
SAF:<th,np,ea>,<measure,np,predic>
Statement:measure
[Ex: der Stein wiegt 1 kg]
'the stone weighs 1 kg'

(23) representative verb
SAF:<th,np,ea>,<repr,np,gov>
Statement:repr
[Ex: das Bild stellt Tom dar]

(24) appellative 1
SAF:<ag,np,ea>,<scsu,np,gov>,<prd,np,function>
Statement:appellat1(arg)
    where arg is pgov(default)
[Ex: er ernennt ihn zum König]
'he nominates him king'

(25) appellative 2
SAF:<ag,np,ea>,<scsu,np,gov>,<prd,Y,predic>
Statement:appellat1(arg)
    where arg is ap(default) or np
[Ex: er nennt ihn ein Genie]
'he calls him a genius'

(26) reflexive manner
SAF:<th,np,ea>,<repr,np,gov>
Statement:refl_manner(arg1,arg2)
    where arg1 is ag(default) or th, and arg 2 is advp(default) or pp
[Ex: er benahm sich vorbildlich]
'he behaved himself well'

(27) ergative ditransitive
SAF:<ben,np,gov>,<th,np,gov>
Statement:erg_ditr
[Ex: _erwartet ihn eine Katastrophe]
'_awaited him a catastrophe'

(28) dat verb
SAF:<ag,np,ea>,<ben,np,io>
Statement:dat
[Ex: er winkt ihm, er vertraut ihm, er begegnet ihm]
'he trusts him'

(29) double acc verb
SAF:<ag,np,ea>,<ben,np,gov>,<th,np,gov>
Statement:double_acc
[Ex: er lehrt ihn Schwimmen, ]
'he teaches him swimming'

(30) ergative experiencer verb
SAF:<ben,np,io>,<th,np,gov>
Statement:erg_exp
[Ex: passiert ihm ein Unglück]
'happens him an accident'
```

86

```
xtemplate(1,iv,
['Er schießt' : [basic : base],
 'es singt sich gut hier':
  [IVmiddle' : base ++ IV_middle'],
 'Er singt ein Lied' : ['CogObj' : base ++ 'CogObj'],
 'wurde ein Lied gesungen ' :
  ['CogObj_Pass' : base ++ 'CogObj' ++ 'Pass'],
 'Das Lied singt sich gut':
  ['TVmiddle' : base ++ 'CogObj' ++ 'TV_middle'],
 'Er sang ihr ein Lied' :
  ['FreeIOins' : base ++ 'CogObj' ++ 'Free_IO_ins'],
 'Er singt sich ein Lied' :
  ['IOrefl' : base ++ 'CogObj' ++ 'Free_IO_ins' ++ 'IO_refl'],
 'Er schießt Schrot' : ['InhObj' : base ++ 'InhObj'],
 'wurde Schrot geschossen' :
  ['InhObj_Pass' : base ++ 'InhObj' ++ 'Pass'],
 'Er schießt die Kugeln weg' :
  ['TVsmallclAdvP' : base ++ 'InhObj' ++ 'TV_smallcl_AdvP'],
 'die Kugeln wegschießen' :
  ['TVsmallclAdvP_IncorpAdv' :
    base ++ 'InhObj' ++ 'TV_smallcl_AdvP' ++
     'Incorp_Adv'(_487,_488)],
 'wurden die Kugeln weggeschossen' :
  ['TVsmallclAdvP_IncorpAdv_Pass' :
    base ++ 'InhObj' ++ 'TV_smallcl_AdvP' ++'Incorp_Adv'() ++ 'Pass'],
 'Er schießt die Kugeln warm' :
  ['TVsmallclAP' : base ++ 'InhObj' ++ 'TV_smallcl_AP'],
 'die Kugeln warmschießen' :
  ['TVsmallclAP_IncorpA' :
    base ++ 'InhObj' ++ 'TV_smallcl_AP' ++
     'Incorp_A'(_1057,_1058)],
 'wurden warmgeschossen die Kugeln' :
  ['TVsmallclAP_IncorpA_Pass' :
    base ++ 'InhObj' ++ 'TV_smallcl_AP' ++
     'Incorp_A'(_1181,_1182) ++ 'Pass'],
 'Er schießt die Kugeln in die Luft' :
  ['TVsmallclPP' : base ++ 'InhObj' ++ 'TV_smallcl_PP'],
 'wurden die Kugeln in die Luft geschossen' :
  ['TVsmallclPP_Pass' :
    base ++ 'InhObj' ++ 'TV_smallcl_PP' ++ 'Pass'],
 'Er schießt Kugeln auf Jon' :
  ['InhObj_PPad' :
    base ++ 'InhObj' ++ 'PP_ad'(_1337,_1338)],
 'wurden Kugeln auf Jon geschossen' :
  ['InhObj_PPad_Pass' :
    base ++ 'InhObj' ++ 'PP_ad'(_1454,_1455) ++ 'Pass'],
 'Er schießt Jon mit Kugeln' :
  ['InhObj_PPad_loadalt' :
    base ++ 'InhObj' ++ 'PP_ad'(_1929,_1930) ++ load_alt],
 'wird gemalt die Wand mit' :
  ['InhObj_PPad_loadalt_Pass' :
    base ++ 'InhObj' ++ 'PP_ad'(_2049,_2050) ++ load_alt ++ 'Pass'],
 'Er malt sich mit' :
  ['InhObj_PPad_loadalt_DOrefl' :
    base ++ 'InhObj' ++ 'PP_ad'(_2170,_2171) ++ load_alt ++
     'DO_refl'],
 'Die Glocken läuten die Feiertage ein' :
  ['IVsmallclAdvP' : base ++ 'IV_smallcl_AdvP'],
 'die Feiertage einläuten' :
  ['IVsmallclAdvP_IncorpAdv' :
    base ++ 'IV_smallcl_AdvP' ++ 'Incorp_Adv'(_2587,_2588)],
 'wurde eingeläutet die Feiertage' :
  ['IVsmallclAdvP_IncorpAdv_Pass' :
    base ++ 'IV_smallcl_AdvP' ++ 'Incorp_Adv'(_2718,_2719) ++ 'Pass'],
 'Er träumt sich weg' :
  ['IVsmallclAdvP_DOrefl' : base ++ 'IV_smallcl_AdvP' ++ 'DO_refl'],
 'Er läuft die Schuhe schief' :
  ['IVsmallclAP' : base ++ 'IV_smallcl_AP'],
'werden schiefgelaufen die Schuhe' :
 ['IVsmallclAP_IncorpA_Pass' :
   base ++ 'IV_smallcl_AP' ++ 'Incorp_A'(_3273,_3274) ++ 'Pass'],
'die Sohlen schieflaufen' :
 ['IVsmallclAP_IncorpA' :
   base ++ 'IV_smallcl_AP' ++ 'Incorp_A'(_3153,_3154)],
'Er singt sich froh' :
 ['IVsmallclAP_DOrefl' : base ++ 'IV_smallcl_AP' ++ 'DO_refl'],
'Er singt alle in gute Laune' :
 ['IVsmallclPP' : base ++ 'IV_smallcl_PP'],
'wurden alle in gute Laune gesungen ' :
 ['IVsmallclPP_Pass' : base ++ 'IV_smallcl_PP' ++ 'Pass'],
'Er singt sich in Stimmung' :
 ['IVsmallclPP_DOrefl' : base ++ 'IV_smallcl_PP' ++ 'DO_refl'],
'Er geht aus' : ['SuPredAdAdvP' : base ++ 'SuPredAd_AdvP'],
'wurde ausgegangen' :
 ['SuPredAdAdvP_Pass' : base ++ 'SuPredAd_AdvP' ++'Incorp_Adv'()++ 'Pass'],
'ausgehen' :
 ['SuPredAdAdvP_IncorpAdv' :
   base ++ 'SuPredAd_AdvP' ++ 'Incorp_Adv'(_3736,_3737)],
'Der Motor läuft leer' : ['SuPredAdAP' : base ++ 'SuPredAd_AP'],
'Er geht an Land' :
 ['SuPredAdPP' : base ++ 'SuPredAd_PP'],
'Er spricht mit Tom' :
 ['PPad' : base ++ 'PP_ad'(_3882,_3883)],
'wurde mit Tom gesprochen' :
 ['PPad_Pass' : base ++ 'PP_ad'(_3996,_3997) ++ 'Pass'],
'Er spricht mit Tom übers Essen' :
 ['PPad_PPad' :
   base ++ 'PP_ad'(_4206,_4207) ++ 'PP_ad'(_4106,_4107)],
'wurde mit Tom übers Essen gesprochen' :
 ['PPad_PPad_Pass' :
   base ++ 'PP_ad'(_4419,_4420) ++ 'PP_ad'(_4319,_4320) ++ 'Pass'],
'den Chef sprechen':
 ['PPad_PPtoDO':
   base ++ 'PP_ad'(_4419,_4420) ++ 'PP_to_DO'],
'Er überspricht die Sache' :
 ['PPad_IncorpP' :
   base ++ 'PP_ad'(_4636,_4637) ++ 'Incorp_P'(_4533,_4534)],
'wurde übersprochen die Sache' :
 ['PPad_IncorpP_Pass' :
   base ++ 'PP_ad'(_4857,_4858) ++ 'Incorp_P'(_4754,_4755) ++
    'Pass'],
'Er malt die Wand' :
 ['PPad_loadalt' :
   base ++ 'PP_ad'(_5318,_5319) ++ load_alt],
'wird gemalt die Wand' :
 ['PPad_loadalt_Pass' :
   base ++ 'PP_ad'(_5435,_5436) ++ load_alt ++ 'Pass'],
'Er malt sich' :
 ['PPad_loadalt_DOrefl' :
   base ++ 'PP_ad'(_5552,_5553) ++ load_alt ++ 'DO_refl'],
'Er schießt ihn ins Bein' :
 ['PartWhtoDO' :
   base ++ 'PP_ad'(_5665,_5666) ++ 'Part_Wh_to_DO'],
'Er schießt ihm ins Bein' :
 ['PartWhtoIO' :
  base ++ 'PP_ad'(_5665,_5666) ++ 'Part_Wh_to_IO'],
'wird geschossen Tom ins Bein' :
 ['PartWHtoIO_Pass' :
   base ++ 'PP_ad'(_5782,_5783) ++ 'Part_Wh_to_IO' ++ 'Pass'],
'Er schießt sich ins Bein' :
 ['PartWHtoDO_DOrefl' :
   base ++ 'PP_ad'(_5899,_5900) ++ 'Part_Wh_to_DO' ++ 'DO_refl'],
'Er geht den langen Weg' : ['AdvtoDO' : base ++ 'Adv_to_DO'],
'wird gegangen der Weg' :
 ['AdvtoDO_Pass' : base ++ 'Adv_to_DO' ++ 'Pass']]).
```

```
xtemplate(2,erg,
['kocht das Wasser' : [basic : base],
 'Das Wasser kocht' : ['Promtoea' : base ++ 'Prom_to_ea'],
 'Er kocht Wasser' : ['Caus' : base ++ 'Caus'],
 'Er kocht das Wasser weg' :
  ['TVsmallclAdvP' : base ++ 'Caus' ++ 'TV_smallcl_AdvP'],
 'das Wasser wegkochen' :
  ['TVsmallclAdvP_IncorpAdv' :
    base ++ 'Caus' ++ 'TV_smallcl_AdvP' ++
     'Incorp_Adv'(_412,_413)],
 'wurde weggekocht das Wasser' :
  ['TVsmallclAdvP_IncorpAdv_Pass' :
    base ++ 'Caus' ++ 'TV_smallcl_AdvP' ++
     'Incorp_Adv'(_547,_548) ++ 'Pass'],
 'Er kocht die Eier hart' :
  ['TVsmallclAP' : base ++ 'Caus' ++ 'TV_smallcl_AP'],
 'die Eier hartkochen' :
  ['TVsmallclAP_IncorpA' :
    base ++ 'Caus' ++ 'TV_smallcl_AP' ++ 'Incorp_A'(_982,_983)],
 'wurden hartgekocht die Eier' :
  ['TVsmallclAP_IncorpA_Pass' :
    base ++ 'Caus' ++ 'TV_smallcl_AP' ++ 'Incorp_A'(_1106,_1107) ++
     'Pass'],
 'Er kocht die Kartoffel zu Mus' :
  ['TVsmallclPP' : base ++ 'Caus' ++ 'TV_smallcl_PP'],
 'wurden die Kartoffel zu Mus gekocht ' :
  ['TVsmallclPP_Pass' :
    base ++ 'Caus' ++ 'TV_smallcl_PP' ++ 'Pass'],
 'Es kocht sich gut hier':
  ['Caus_Dodel_IVmiddle': base ++ 'Caus' ++ 'IV_middle'],
 'Er kocht die Sachen aus' :
  ['IVsmallclAdvP' :
    base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AdvP'],
 'die Sachen auskochen' :
  ['IVsmallclAdvP_IncorpAdv' :
    base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AdvP' ++
     'Incorp_Adv'(_1594,_1595)],
 ' wurden ausgekocht die Sachen' :
  ['IVsmallclAdvP_IncorpAdv_Pass' :
    base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AdvP' ++
     'Incorp_Adv'(_1732,_1733) ++ 'Pass'],
 'Er kocht den Kessel schwarz' :
  ['IVsmallclAP' : base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AP'],
 'den Kessel schwarzkochen' :
  ['IVsmallclAP_IncorpA' :
    base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AP' ++
     'Incorp_A'(_2184,_2185)],
 'wurde schwarzgekocht der Kessel' :
  ['IVsmallclAP_IncorpA_Pass' :
    base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AP' ++
     'Incorp_A'(_2312,_2313) ++ 'Pass'],
 'Er kocht sich müde' :
  ['IVsmallclAP_DOrefl' :
    base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_AP' ++ 'DO_refl'],
 'Er kocht ein Loch in den Kessel' :
  ['IVsmallclPP' : base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_PP'],
 'wurde gekocht ein Loch in den Kessel' :
  ['IVsmallclPP_Pass' :
    base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_PP' ++ 'Pass'],
 'Er kocht sich in gute Laune' :
  ['IVsmallclPP_DOrefl' :
    base ++ 'Caus' ++ 'DO_del' ++ 'IV_smallcl_PP' ++ 'DO_refl'],
 'Das Wasser kocht weg' :
  ['TVsmallclAdvP_Promtoea' :
    base ++ 'TV_smallcl_AdvP' ++ 'Prom_to_ea'],
 'Das Wasser ist weggekocht' :
  ['TVsmallclAdvP_IncorpAdv_Promtoea' :
    base ++ 'TV_smallcl_AdvP' ++ 'Incorp_Adv'(_2555,_2556) ++
      'Prom_to_ea'],
 'Die Kartoffeln kochen trocken' :
  ['TVsmallclAP_Promtoea' : base ++ 'TV_smallcl_AP' ++ 'Prom_to_ea'],
 'Die Kartoffeln kochen zu Brei' :
  ['TVsmallclPP_Promtoea' : base ++ 'TV_smallcl_PP' ++ 'Prom_to_ea'],
 'Er erhebt sich' :
  ['Caus_DOrefl' : base ++ 'Caus' ++ 'DO_refl'],
 'Eine Stimme erhebt sich' :
  ['Ergrefl_Promtoea' : base ++ 'Erg_refl' ++ 'Prom_to_ea'],
 'Es erhebt sich eine Stimme' :
  ['Ergrefl_detins' : base ++ 'Erg_refl' ++ det_ins]]).

xtemplate(3,exp_iv,
['Er friert' : [basic : base],
 'Er friert zu Tode' : ['SuPredAdPP' : base ++ 'SuPredAd_PP']]).

xtemplate(4,tv,
['Er ißt das Essen : [basic : base],
 'wird gegessen das Essen' : ['Pass' : base ++ 'Pass'],
 'Er kauft ihr einen Hut' :
  ['FreeIOins' : base ++ 'Free_IO_ins'],
 'Er kauft sich einen Hut' :
  ['IOrefl' : base ++ 'Free_IO_ins' ++ 'IO_refl'],
 'wurde ihr ein Hut gekauft' :
  ['FreeIOins_Pass' : base ++ 'Free_IO_ins' ++ 'Pass'],
 'Bücher kaufen sich leicht':
  ['TVmiddle': base ++ 'TV_middle'],
 'Er wäscht sich' : ['DOrefl' : base ++ 'DO_refl'],
 'Er ißt das Essen auf' :
  ['TVsmallclAdvP' : base ++ 'TV_smallcl_AdvP'],
 'das Essen aufessen' :
  ['TVsmallclAdvP_IncorpAdv' :
    base ++ 'TV_smallcl_AdvP' ++ 'Incorp_Adv'(_449,_450)],
 'wird aufgegessen' :
  ['TVsmallclAdvP_Pass' : base ++ 'TV_smallcl_AdvP' ++'Incorp_Adv'()++ 'Pass'],
 'Er wirft sich hinab' :
  ['TVsmallclAdvP_DOrefl' : base ++ 'TV_smallcl_AdvP' ++ 'DO_refl'],
 'Er wäscht die Kleider sauber' : ['TVsmallclAP' : base ++ 'TV_smallcl_AP'],
 'Er wäscht sich sauber' :
  ['TVsmallclAP_DOrefl' : base ++ 'TV_smallcl_AP' ++ 'DO_refl'],
 'die Kleider sauberwaschen' :
  ['TVsmallclAP_IncorpA' :
    base ++ 'TV_smallcl_AP' ++ 'Incorp_A'(_1015,_1016)],
 'werden die Kleider saubergewaschen' :
  ['TVsmallclAP_IncorpA_Pass' :
    base ++ 'TV_smallcl_AP' ++ 'Incorp_A'(_1135,_1136) ++ 'Pass'],
 'Er führt Energie zum Körper' :
  ['TVsmallclPP' : base ++ 'TV_smallcl_PP'],
 'wird Energie dem Körper zugeführt' :
  ['TVsmallclPP_Pass' : base ++ 'TV_smallcl_PP' ++'Incorp_Adv'(X,Y)++ 'Pass'],
 'dem Köper Energie zuführen' :
  ['TVsmallclPP_IncorpP' :
    base ++ 'TV_smallcl_PP' ++ 'Incorp_P'(_1405,_1406)],
 'Er führt sich Energie zu' :
  ['TVsmallclPP_IncorpP_DOrefl' :
    base ++ 'TV_smallcl_PP' ++ 'DO_refl'],
 'sich Energie zuführen':
  ['TVsmallclPP_IncorpP_DOrefl' :
    base ++ 'TV_smallcl_PP' ++ 'Incorp_P'(_1653,_1654) ++ 'DO_refl'],
 'Er ißt' : ['DOdel' : base ++ 'DO_del'],
 'wurde gegessen' : ['DOdel_Pass' : base ++ 'DO_del' ++ 'Pass'],
 'Er trinkt die Sorgen hinweg' :
  ['IVsmallclAdvP' : base ++ 'DO_del' ++ 'IV_smallcl_AdvP'],
  'die Sorgen hinwegtrinken' :
  ['IVsmallclAdvP_IncorpAdv' :
    base ++ 'DO_del' ++ 'IV_smallcl_AdvP' ++
      'Incorp_Adv'(_2118,_2119)],
 'wurden hinweggetrunken' :
```

```
 ['IVsmallclAdvP_IncorpAdv_Pass' :
   base ++ 'DO_del' ++ 'IV_smallcl_AdvP' ++
    'Incorp_Adv'(_2253,_2254) ++ 'Pass'],
'er trinkt sich weg' :
 ['IVsmallclAdvP_DOrefl' :
   base ++ 'DO_del' ++ 'IV_smallcl_AdvP' ++ 'DO_refl'],
'Er ißt den Teller leer' :
 ['IVsmallclAP' : base ++ 'DO_del' ++ 'IV_smallcl_AP'],
 'den Teller leeressen' :
 ['IVsmallclAP_IncorpA' :
   base ++ 'DO_del' ++ 'IV_smallcl_AP' ++
    'Incorp_A'(_2708,_2709)],
'wird der Teller leergegessen' :
 ['IVsmallclAP_IncorpA_Pass' :
   base ++ 'DO_del' ++ 'IV_smallcl_AP' ++
    'Incorp_A'(_2832,_2833) ++ 'Pass'],
'Er ißt sich satt' :
 ['IVsmallclAP_DOrefl' :
   base ++ 'DO_del' ++ 'IV_smallcl_AP' ++ 'DO_refl'],
'Er trinkt Tom unter den Tisch' :
 ['IVsmallclPP' : base ++ 'DO_del' ++ 'IV_smallcl_PP'],
'wird unter den Tisch getrunken' :
 ['IVsmallclPP_Pass' :
   base ++ 'DO_del' ++ 'IV_smallcl_PP' ++ 'Pass'],
'Er ißt sich in gute Laune' :
 ['IVsmallclPP_DOrefl' :
   base ++ 'DO_del' ++ 'IV_smallcl_PP' ++ 'DO_refl'],
'Er ißt vom Brot' :
 ['DOtoPP' : base ++ 'DO_to_PP'(_3008)],
'wird gegessen vom Brot' :
 ['DOtoPP_Pass' : base ++ 'DO_to_PP'(_3079) ++ 'Pass'],
'Er trinkt den Kaffee heiß' :
 ['ObjQual' : base ++ 'ObjQual'(_3160)],
'wird getrunken der Kaffee heiß' :
 ['ObjQual_Pass' : base ++ 'ObjQual'(_3258) ++ 'Pass'],
'Er sagt etwas zu Tom' :
 ['PPad' : base ++ 'PP_ad'(_3360,_3361)],
'wird gesagt etwas zu Tom' :
 ['PPad_Pass' : base ++ 'PP_ad'(_3474,_3475) ++ 'Pass'],
'Er sagt etwas zu Tom über Marit' :
 ['PPad_PPad' :
   base ++ 'PP_ad'(_3684,_3685) ++ 'PP_ad'(_3584,_3585)],
'Er füllt den Eimer mit Wasser' :
 ['PPad_loadalt' :
   base ++ 'PP_ad'(_4465,_4466) ++ load_alt],
'wird gefüllt der Eimer mit Wasser' :
 ['PPad_loadalt_Pass' :
   base ++ 'PP_ad'(_4582,_4583) ++ load_alt ++ 'Pass'],
'Er füllt sich mit Gin' :
 ['PPad_loadalt_DOrefl' :
   base ++ 'PP_ad'(_4699,_4700) ++ load_alt ++ 'DO_refl'],
'Er wirft Tom ein Buch an den Kopf' :
 ['PartWhtoIO' :
   base ++ 'PP_ad'(_4813,_4814) ++ 'Part_Wh_to_IO'],
'wurden geworfen Tom ein Buch an den Kopf' :
 ['PartWhtoIO_Pass' :
   base ++ 'PP_ad'(_4930,_4931) ++ 'Part_Wh_to_IO' ++ 'Pass'],
'er wirft sich ein Buch an den Kopf' :
 ['PartWhtoIO_IOrefl' :
   base ++ 'PP_ad'(_5047,_5048) ++ 'Part_Wh_to_IO' ++ 'IO_refl'],
'er wirft sich Tom an den Hals' :
 ['PartWhtoIO_DOrefl' :
   base ++ 'PP_ad'(_5164,_5165) ++ 'Part_Wh_to_IO' ++ 'DO_refl'],
'Er sieht einen Elch kommen' : ['AcI' : base ++ 'AcI'],
'wurde ein Elch kommen gesehen' :
 ['AcI_Pass' : base ++ 'AcI' ++ 'Pass'],
'Er sah sich fallen' :
 ['AcI_DOrefl' : base ++ 'AcI' ++ 'DO_refl'],
  'Er sagt daßS' : [daßS : base ++ daßS(_5333)],
  'Er sagte Maria daßS' :
   [daßS_FreeIOins : base ++ daßS(_5467) ++ 'Free_IO_ins'],
  'wurde gesagt Maria daßS' :
   [daßS_FreeIOins_Pass :
     base ++ daßS(_5604) ++ 'Free_IO_ins' ++ 'Pass'],
  'wurde gesagt daßS' :
   [daßS_Pass : base ++ daßS(_5737) ++ 'Pass'],
  'Er sagt zu Tom daßS' :
   [daßS_PPad :
     base ++ daßS(_5967) ++ 'PP_ad'(_5855,_5856)],
  'wurde gesagt zu Tom daßS' :
   [daßS_PPad_Pass :
     base ++ daßS(_6201) ++ 'PP_ad'(_6089,_6090) ++ 'Pass'],
  'Er sagt weshalb' : [whS : base ++ whS(_6335)],
  'Er sagte Maria weshalb' :
   [whS_FreeIOins : base ++ whS(_6482) ++ 'Free_IO_ins'],
  'wurde gesagt Maria weshalb' :
   [whS_FreeIOins_Pass :
     base ++ whS(_6633) ++ 'Free_IO_ins' ++ 'Pass'],
  'wurde gesagt weshalb' :
   [whS_Pass : base ++ whS(_6781) ++ 'Pass'],
  'Er sagt zu Tom weshalb' :
   [whS_PPad :
     base ++ whS(_7025) ++ 'PP_ad'(_6909,_6910)],
  'wurde gesagt zu Tom weshalb' :
   [whS_PPad_Pass :
     base ++ whS(_7272) ++ 'PP_ad'(_7156,_7157) ++ 'Pass'],
  'wurde versucht zu' :
   [zu_inf_Pass : base ++ zu_inf ++ 'Pass']]).

xtemplate(5,th_tv,
['Löschpapier absorbiert Tinte' : [basic : base],
 'wird absorbiert Tinte' : ['Pass' : base ++ 'Pass']]).

xtemplate(6,exp_tv,
['Er mag Kaffee' : [basic : base],
 'Er mag seinen Kaffee schwarz' :
  ['Depict' : base ++ 'Depict'(_46)]]).

xtemplate(7,ditv,
['Er gibt Tom Geld' : [basic : base],
 'wurde Tom Geld gegeben' : ['Pass' : base ++ 'Pass'],
 'Er gibt Geld' : ['IOdel' : base ++ 'IO_del'],
 'wurde Geld gegeben' :
  ['IOdel_Pass' : base ++ 'IO_del' ++ 'Pass'],
 'Er gibt Geld an die Armen' :
  ['IOdel_PPad' :
    base ++ 'IO_del' ++ 'PP_ad'(_98,_99)],
 'wurde Geld an die Armen gegeben' :
  ['IOdel_PPad_Pass' :
    base ++ 'IO_del' ++ 'PP_ad'(_215,_216) ++ 'Pass'],
 'Er gibt sich ganz' :
  ['IOdel_DOrefl' : base ++ 'IO_del' ++ 'DO_refl'],
 'Er gibt' : ['IOdel_DOdel' : base ++ 'IO_del' ++ 'DO_del'],
 'wurde gegeben' :
  ['IOdel_DOdel_Pass' :
    base ++ 'IO_del' ++ 'DO_del' ++ 'Pass'],
 'Er gibt den Armen' :
  ['IOdel_DOdel_PPad' :
    base ++ 'IO_del' ++ 'DO_del' ++ 'PP_ad'(_383,_384)],
 'Er gibt die Büchse voll' :
  ['IVsmallclAP' : base ++ 'IO_del' ++ 'DO_del' ++ 'IV_smallcl_AP'],
 'wurde die Büchse vollgegeben' :
  ['IVsmallclAP_IncorpA_Pass' :
    base ++ 'IO_del' ++ 'DO_del' ++ 'IV_smallcl_AP' ++'Incorp_A'()++ 'Pass'],
 'Er gibt die Familie aus dem Haus' :
  ['IVsmallclPP' : base ++ 'IO_del' ++ 'DO_del' ++ 'IV_smallcl_PP'],
  'wurde die Familie aus dem Haus gegeben' :
   ['IVsmallclPP_Pass' :
     base ++ 'IO_del' ++ 'DO_del' ++ 'IV_smallcl_PP' ++ 'Pass'],
  'Er gibt den Plan auf' :
   ['IVsmallclAdvP' :
     base ++ 'IO_del' ++ 'DO_del' ++ 'IV_smallcl_AdvP'],
  'den Plan aufgeben' :
   ['IVsmallclAdvP_IncorpAdv' :
     base ++ 'IO_del' ++ 'DO_del' ++ 'IV_smallcl_AdvP' ++'Incorp_Adv'()],
  'wurde der Plan aufgegeben' :
   ['IVsmallclAdvP_IncorpAdv_Pass' :
     base ++ 'IO_del' ++ 'DO_del' ++ 'IV_smallcl_AdvP'
     ++'Incorp_Adv'() ++'Pass'],,
  'Er gibt sich auf',:
  ['IVsmallclAdvP_DOrefl':
     base ++ 'IO_del' ++ 'DO_del' ++ 'IV_smallcl_AdvP'
     ++'DO_refl'],
  ['sich aufgeben' :
     base ++ 'IO_del' ++ 'DO_del' ++ 'IV_smallcl_AdvP'
      ++'Incorp_Adv'() ++'DO_refl'],
  'Er gibt auf' :
    ['IOdel_DOdel_SuPredAdAdvP' :
     base ++ 'IO_del' ++ 'DO_del' ++ 'SuPredAd_AdvP'],
  'Er gibt Tom einen Stoß in den Rücken' :
   ['PartWHtoIO' : base ++ 'IO_del' ++ 'Part_Wh_to_IO'],
  'wurde Tom ein Stoß in den Rücken gegeben' :
   ['PartWHtoIO_Pass' :
     base ++ 'IO_del' ++ 'Part_Wh_to_IO' ++ 'Pass'],
  'Er gibt das Geld weg' :
   ['TVsmallclAdvP' : base ++ 'IO_del' ++ 'TV_smallcl_AdvP'],
  'das Geld weggeben' :
  ['TVsmallclAdvP' :
     base ++ 'IO_del' ++ 'TV_smallcl_AdvP' ++'Incorp_Adv'()],
  'wurde das Geld weggegeben' :
   ['TVsmallclAdvP_IncorpAdv_Pass' :
     base ++ 'IO_del' ++ 'TV_smallcl_AdvP' ++'Incorp_Adv'()++ 'Pass'],
  'Er gibt das Geld aus dem Lande' :
   ['TVsmallclPP' : base ++ 'IO_del' ++ 'TV_smallcl_PP'],
  'wurde das Gold aus dem Lande gegeben' :
   ['TVsmallclPP_Pass' :
     base ++ 'IO_del' ++ 'TV_smallcl_PP' ++ 'Pass'],
  'Per gir seg god tid' : ['IOrefl' : base ++ 'IO_refl']]).

xtemplate(8,refl,
['Er schämt sich' : [basic : base],
 'Er schämt sich wegen seines Autos' :
  ['PPad' : base ++ 'PP_ad'(_54,_55)]]).

xtemplate(9,psych,
['langweilt ihn das Buch' : [basic : base],
 'Das Buch langweilt ihn' : ['Promtoea' : base ++ 'Prom_to_ea'],
 'Er langweilt sich' :
  ['DOdel_ExpMvt' : base ++ 'DO_del' ++ 'ExpMvt'],
 'Er interessiert sich für das Buch' :
  ['ExpMvt_DOtoPP' : base ++ 'ExpMvt' ++ 'DO_to_PP'(_77)],
 'Er langweilt Tom mit Gerede' :
  ['DOtoPP_Caus' : base ++ 'DO_to_PP'(_151) ++ 'Caus'],
 'Er langweilt Tom' :
  ['DOdel_Caus' : base ++ 'DO_del' ++ 'Caus'],
 'Er ist gelanweilt' :
  ['DOdel_ExpAdj' : base ++ 'DO_del' ++ 'ExpAdj'],
 'Er ist interessiert am Buch' :
  ['ExpAdj_DOtoPP' : base ++ 'ExpAdj' ++ 'DO_to_PP'(_272)],
 'Das Buch begeistert' :
  ['Expdel_Promtoea' : base ++ 'Exp_del' ++ 'Prom_to_ea'],
 'interessiert Tom daßS : [daßS : base ++ daßS(_393)],
 'daßS interessiert Tom' :
  [daßS_Antiexp : base ++ daßS(_526) ++ 'Anti_exp'],
 'Er ärgert sich darüber daßS' :
  [daßS_ExpMvt_DOtoPP :
    base ++ daßS(_663) ++ 'ExpMvt' ++ 'DO_to_PP'],
 'interessiert Tom whS' : [whS : base ++ whS(_931)],
 'whS interessiert Tom' :
  [whS_Antiexp : base ++ whS(_1078) ++ 'Anti_exp'],
 'Er interessierte seg dafür whS' :
  [whS_ExpMvt_DOtoPP :
    base ++ whS(_1229) ++ 'ExpMvt' ++ 'DO_to_PP'],
 'interessiert Tom zu' : [zu_inf : base ++ zu_inf],
 'zu interessiert Tom' :
  [zu_inf_Antiexp : base ++ zu_inf ++ 'Anti_exp'],
 'Er interessiert sich dafür zu' :
  [zu_inf_ExpMvt_DOtoPP :
    base ++ zu_inf ++ 'ExpMvt' ++ 'DO_to_PP']]).

xtemplate(10,psych2,
 ['gelüstet ihn nach Abenteuern' : [basic : base]
  'es gelüstet ihn nach Abenteuern' : [esins :base ++ esins]]).

xtemplate(11,raisv1,
['zeigt sich daß Tom kompetent ist' : [basic : base],
 'es zeigte sich daß Tom' : [esins : base ++ es_ins],
 'Jon erweist sich krank zu sein' :
  [daßStoClinf_SubjRais :
    base ++ daßS_to_Cl_inf(_227) ++ 'SubjRais'(_85)],
 'Jon erweist sich krank' :
  [daßStoClinf_SubjRais_zu_inftopredicAP :
    base ++ daßS_to_Cl_inf(_526) ++ 'SubjRais'(_384) ++
     zu_inf_to_predic_AP]]).

xtemplate(12,raisv2,
['kommt mir vor als ob' : [basic : base],
 'es kommt mir vor als ob ' : [esins : base ++ es_ins],
 'er kommt mir vor als ob' :
  ['RaisCop' : base ++ 'RaisCop'],
 'er kommt mir krank vor' :
  ['PropcompltoSC_SubjRais' :
    base ++ 'Prop_compl_to_SC'(_242) ++ 'SubjRais'(_99)]]).

xtemplate(13,raisv3,
['scheint mir daß Tom krank ist' : [basic : base],
 'es scheint mir daß Tom krank ist' : [esins : base ++ es_ins],
 'es scheint daß' :
  ['IOdel_detins' : base ++ 'IO_del' ++ es_ins],
 'es scheint mir als ob ' :
  [daßStopropcompl_esins :
    base ++ daßS_to_prop_compl ++ es_ins],
 'es scheint als ob' :
  ['IOdel_daßStopropcompl_detins' :
    base ++ 'IO_del' ++ daßS_to_prop_compl ++ es_ins],
 'Tom scheint mir als ob er krank ist??' :
  [atStopropcompl_RaisCopy :
    base ++ daßS_to_prop_compl ++ 'RaisCop'],
 'Tom scheint als ob er ..' :
  ['IOdel_daßStopropcompl_Raiscopy' :
    base ++ 'IO_del' ++ daßS_to_prop_compl ++ 'RaisCop'],
 'Tom scheint mir krank zu sein :
  [atStoClinf_SubjRais :
    base ++ daßS_to_Cl_inf(_319) ++ 'SubjRais'(_177)],
 'Tom scheint krank zu sein' :
  ['IOdel_daßStoClinf_SubjRais' :
    base ++ 'IO_del' ++ daßS_to_Cl_inf(_615) ++
     'SubjRais'(_473)],
 'Er scheint mir krank' :
  [daßStoClinf_SubjRais_z_inftopredicAP :
    base ++ daßS_to_Cl_inf(_917) ++ 'SubjRais'(_775) ++
     zu_inf_to_predic_AP],
 'Er scheint krank' :
  ['IOdel_daßStoClinf_SubjRais_zuinftopredicAP' :
    base ++ 'IO_del' ++ daßS_to_Cl_inf(_1216) ++
     'SubjRais'(_1074) ++ zu_inf_to_predic_AP]]).

xtemplate(14,depict,
['Er stellt sich Tom vor' : [basic : base],
 'Er stellt sich vor daß' :
  [daßS : base ++ daßS(_66)],
 'Er stellt sich Tom gesund vor' :
  [daßStoSC : base ++ daßS(_199) ++ daßS_to_SC]]).

xtemplate(15,smallcl,
['man sieht ihn für einen Idioten an' : [basic : base],
 'wird für ein Idiot gehalten Tom ' : ['Pass' : base ++ 'Pass'],
 'er hält sich für ein Genie' : ['DOrefl' : base ++ 'DO-refl'],
 'man sieht ihn als schuldig an' :
  [predicals : base ++ predic_für_to_predic_als],
 'wird als schuldig angesehen Tom' :
  [predicals_Pass : base ++ predic_als ++ 'Pass']
 'er sieht sich als schuldig an' :  ['predicals_DOrefl' : base ++
    'predic_für_to_predic_als' ++ 'DO-refl']]).

xtemplate(16,iv_loc,
['Er wohnt in der Stadt' : [basic : base],
 'wird gewohnt in dieser Hütte' :[ base ++ 'Pass'],
 'es wohnt sich gut in dieser Hütte':
 ['IVmiddle': base ++ 'IV_middle']]).

xtemplate(17,refl_loc,
['Er hält sich hier auf' : [basic : base]]).

xtemplate(18,tv_loc,
['Er setzt die Blumen auf den Tisch' : [basic : base],
 'wurde die Vase auf den Tisch gesetzt' : ['Pass' : base ++ 'Pass']]).

xtemplate(19,ind_arg,
['Seine Ansicht beruht auf einem Irrtum' : [basic : base]]).

xtemplate(20,ind_arg_refl,
['Er vergewissert sich seines Daseins' : [basic : base],
 'Er vergewissert sich daßS': ['daßS' :base ++ 'daßS'()],
 'Er vergewissert sich obS' : [ 'obS' : base ++ 'obS'()]]).

xtemplate(21,weather,
['schneit' : [basic : base],
 'es schneit' : [esins : base ++ es_ins],
 'es schneit dicke Flocken' :
  ['InhObj_esins' : base ++ 'InhObj' ++ es_ins]]).

xtemplate(22,measure,
['der Stein wiegt 3 kg' : [basic : base]]).

xtemplate(23,repr,
['das Bild stellt Tom dar' : [basic : base],
 'Tom darstellen' : ['IncorpAdv': base ++ Incorp_Adv(X,Y)]]).

xtemplate(24,appellat1,
['Er ernennt ihn zum Chef' : [basic : base],
 'wurde Tom zum Chéf ernannt' : ['Pass' : base ++ 'Pass'],
 'Tom ernennt sich zum Chef' : ['DOrefl' : base ++ 'DO_refl']]).

xtemplate(25,appellat2,
['Er nennt Tom ein Genie' : [basic : base],
 'wird genannt Tom ein Genie' : ['Pass' : base ++ 'Pass'],
 'Er nennt sich ein Genie' : ['DO_refl' : base ++ 'DO_refl']]).

xtemplate(26,refl_manner,
['Er benimmt sich vorbildlich' : [basic : base]
 'Er benimmt sich wie ein Idiot': ['Predicwie': base ++ 'Predic_wie']]).

xtemplate(27,erg_ditv,
['erwartet Tom ein Unglück' : [basic : base],
 'es erwartet Tom eine Katastrophe' : [esins : base ++ es_ins],
 'eine Katastrophe erwartet Tom' : ['Promtoea' : base ++ 'Prom_to_ea']]).

xtemplate(28, dat_verb,
['er winkt den Leuten' : [basic : base],
 'wurde gewunken den Leuten' :'Pass' : base ++ 'Pass']]).

xtemplate(29, double_acc,
['er lehrt ihn das Schwimmen' : [basic : base],
 'wurde gelehrt ihn das Schwimmen' :'Pass' : base ++ 'Pass']
 'er lehrte den Kindern das Schwimmen':['Datalt': base ++ 'Dat_alt']
 'ihnen wurde das Schwimmen gelehrt': 'Datalt_Pass': basé ++ 'Dat_alt'++ 'Pass'
 'er lehrte ihn die Maschine zu bedienen':['zuinf': base ++ zu_inf]]).

xtemplate(30, erg_exp,
['passiert mir ein Unglück': [basic : base],
'ein Unglück passiert': [IOdel':base ++ 'IO_del']]).
```

App. 9a

APP. 9

Basic Templates

(1) **intransitive verb**
SAF: <ag,np,ea>
statement: **iv**
[Ex. Jan danst]
'John dances'

(2) **ergative**
SAF: <th,np,gov>
Statement: **erg**
[Ex. _rollen stenen]
'roll stones'

(3) **experiencer intransitive verb**
SAF: <exp,np,ea>,<scsu,np,gov>,<prd,X,predic>
Statement: **exp_iv**(arg)
where arg is AP (default) or AdvP
[Ex. Jan is het gezeur beu]
'John is tired of harping'

(4) **transitive verb**
SAF: <ag,np,ea>,<th,np,gov>
Statement: **tv**
[Ex. Jan gooit een steen]
'John throws a stone'

(5) **theme transitive verb**
SAF: <th,np,ea>,<th,np,gov>
Statement: **th_tv**
[Ex. papier absorbeert water]
'paper absorbs water'

(6) **experiencer transitive verb**
SAF: <exp,np,ea>,<th,np,gov>
Statement: **exp_tv**
[Ex. Jan haat huiswerk]
'John hates homework'

(7) **ditransitive verb**
SAF: <ag,np,ea>,<ben,np,io>,<th,np,gov>
Statement: **ditv**
[Ex. Jan geeft Piet een boek]
'John gives Peter a book'

(8) **reflexive verb**
SAF: <exp,np,ea>,<norole,zich,refl>
Statement: **refl**
[Jan schaamt zich]
'John shames himself'

(9) **psych verb**
SAF: <exp,np,io>,<th,np,gov>
Statement: **psych**
[Ex. _irriteren Jan de kat]
'_irritate John the cat'

(10) **raising verb 1**
SAF: <th,dat_S,gov>
Statement: **raisverb1**
[Ex. _blijken dat Jan ziek is]
'_appears John to be ill'

(11) **raising verb 2**
SAF: <exp,np,io>,<th,dat_S,gov>
Statement: **raisv3**
[Ex. _lijken mij dat Jan ziek is]
'_seem to me that John is ill'

(12) **raising verb 3**
SAF: <th,dat_S,*function*>
Statement: **raisv3**(arg)
where arg is gov (default), pgov or a preposition
[Ex. _lijken op dat Jan ziek is]
'_look like John to be ill'

(13) **depictive**
SAF: <ag,np,ea>,<norole,zich,refl>,<th,np,gov>
Statement **depict**
[Ex. Jan stelt zich Piet voor]
'John imagines (himself) Peter'

(14) **small clause**
SAF: <ag,np,ea>,<scsu,np,gov>,<prd,X,*function*>
Statement: **smcl**(arg1,arg2)
where arg1 is ap (default) or np, and arg2 is pgov or a preposition
[Ex. Jan beschouwt Piet als ziek]
'John considers Peter as ill'

(15) **intransitive verb locative**
SAF: <X,np,ea>,<loc,Y,adv>
Statement: **iv_loc**(arg1,arg2)
where arg1 is ag (default) or th, and arg2 is pp (default) or advp
[Ex. Jan woont in Noorwegen]
'John lives in Norway'

(16) **reflexive verb locative**
SAF: <X,np,ea>,<norole,zich,refl>,<loc,Y,adv>
Statement: **refl_loc**(arg1,arg2)
where arg1 is ag (default) or th, and arg2 is pp (default) or advp
[Ex. Jan houdt zich op in Noorwegen]
'John stays in Norway'

(17) **transitive verb locative**
SAF: <ag,np,ea>,<th,np,gov>,<loc,np,*function*>
Statement: **tv_loc**(arg)
where arg is pgov (default) or a preposition
[Ex. Jan zet de vaas op de tafel]
'John puts the vase on the table'

(18) **indirect argument**
SAF: <X,np,ea>,<th,np,*function*>
Statement: **ind_arg**(arg1,arg2)
where arg1 is ag (default), exp or th, and arg2 is pgov (default) or a preposition
[Ex. Jan verlangt naar haar]
'John is longing for her'

(19) **di-indirect argument**
SAF: <ag,np,ea>,<th,np,*function*>,<ben,np,*function*>
Statement: **di_ind_arg**(arg1,arg2)
where arg1 is pgov or gov, and arg2 is pgov or gov
[Ex. Jan dringt bij Marie op iets aan]
'John presses Mary for something'

(20) **indirect argument reflexive**
SAF: <ag,np,ea>,<norole,zich,refl>,<th,np,*function*>
Statement: **ind_arg_refl**(arg)
where arg is pgov (default) or a preposition
{Ex. Jan verdiept zich in de krant]
'John pores over the newspaper'

(21) **weather verb**
SAF: Ø
Statement: **weather**
[Ex. _regenen]
'_rain'

(22) **measure verb**
SAF: <th,np,ea>,<measure,np,predic>
Statement: **measure**
[Ex. de stenen wegen 3 kg]
'the stones weigh 3 kg'

(23) **representative**
SAF: <th,np,ea>,<repr,np,,gov>
Statement: **repr**
[Ex. het schilderij stelt een vrouw voor'
'the painting represents a woman'

(24) **appellative 1**
SAF: <ag,np,ea>,<scsu,np,gov>,<prd,np,*function*>
Statement: **appellat1**(arg)
where arg is pgov (default) or a preposition
[Ex. Jan benoemt Piet tot koning]
'John nominates Peter king'

(25) **appellative 2**
SAF: <ag,np,ea>,<scsu,np,gov>,<prd,Y,predic>
Statement: **appellat2**(arg)
where arg is ap or np
[Ex. Jan noemt Piet Y]
'John calls Peter Y'

(26) **reflexive verb manner**
SAF: <X,np,ea>,<norole,zich,refl>,<manner,Y,adv>
Statement: **refl_manner**(arg1,arg2)
where arg1 is ag (default) or th, and arg2 is advp (default) or pp
[Ex. Jan gedraagt zich goed]
'John behaves himself well'

(27) **ergative ditransitive**
SAF: <ben,np,io>,<th,np,gov>
Statement: **erg_ditv**
[Ex. _wachten Jan een ongeluk]
'_wait John an accident'

(28) **benefactive transitive**
SAF: <ag,np,ea>,<ben,np,gov>
Statement: **ben_tv**
[Ex. Jan betaalt haar]
'John pays her'

(29) **copula**
SAF: <scsu,np,gov>,<prd,Y,predic>
Statement: **cop**(arg)
where arg is ap (default), np or pp
[Ex. _zijn_Jan_ziek]
'_be John ill'

(30a) **laten1**
SAF: <ag,np,ea>,<ben,np,io>,<th,Inf_compl,gov>
statement: **laten1**
[Ex. Jan laat Piet de eendjes voeren]
'John allows Peter to feed the ducks'

(30b) **laten2**
SAF: <X,np,ea>,<scsu,np,gov>,<prd,Y,predic>
statement: **laten2**(arg1,arg2)
where arg1 is ag (default) or th, and arg2 is Inf-compl (default), ap or pp
[Ex. Jan laat mij koud]
'John leaves me cold'

(30c) **laten3**
SAF: <th,Inf_compl,gov>
statement: **laten3**
[Ex. _laten een oplossing vinden]
'let a solution to be found'

App. 96

The derivational rules[1]

DO(direct object)-deletion	DO_del
Object-to-PP	DO_to_PP
IO-deletion	IO_del
IV-SmallClause-formation AP / PP / AdvP	IV_smcl_*[2]
TV-SmallClause-formation AP / PP / AdvP	TV_smcl_*
Depictive small clause	Depict
Accusative with infinitive	AcI
Object Qualification	ObjQual
Cognate Object	CogObj
load-alternation	load_alt
Part-whole-to-DO	Part_Wh_to_DO
Part-whole-to-IO	Part_Wh_to_IO
Adv-to-DO	Adv_to_DO
Causativization with non-ea	Caus
Causativization with ea	Ea_caus
DO-reflexivization	DO_refl
IO-reflexivization	IO_refl
Passive	Pass[3]
Promotion to EA	Prom_to_ea
Subject-predicate adjunction AP / PP / AdvP	SuPred_ad_*
PP-adjunction	PP_ad
Incorporation A / P[4] / Adv	Incorp_*

[1]This list contains the derivational rules that are used to describe derivations of Dutch verbs. The first part of this list is an enumeration of the rules that are overtaken from the derivational rules for Norwegian and German. The last part consists of new rules. I'll give an informal description of the syntactic change they effectuate, and where possible the same is done for the change in the semantics.
[2]The '*' can be substituted by AP, AdvP or PP.
[3]The Passive rule consists of two types: Short Passive and Long Passive. In derivations of the verbs these differentiations are both denominated by 'Pass'. At the end of this list is stated under which circumstances the different types occur.
[4] Only postpositions incorporate in Dutch.

Substitution NP-datS NP_to_datS[1]

Substitution NP-te-Inf NP_to_te-Inf

Substitution NP-wS NP_to_wS

Substitution NP-ofS NP_to_ofS

Subject Raising SubjRais

Substitution of te-Inf with predic-AP te-Inf_to_pred_AP

Substitution of datS with clausal infinitive datS_to_CL_inf

Anti-extraposition Anti_exp

Experiencer movement ExpMvt

Experiencer adjectival construction ExpAdj

Ergative zich insertion Erg_refl

Indirect object insertion[2] IO_ins

Inherent object InhObj

Existential er insertion
Syntactic change:
If there is an indefinite ea, then:
-move this ea to internal argument position, insert expletive er in the empty ea position.
Else, if there's no ea and there's an indefinite direct object, then:
-insert expletive er in the empty ea position.
Semantic change:
Adds an 'observational' aspect to the meaning of the event or state that is expressed by the verb and its complements.
Rule statement: Exist_er_ins
[Ex. iemand zingt -> er zingt iemand]

VP-Qualification
Syntactic change:
If there's an agentive ea:
-delete the ea. Add an advP that qualifies the VP.
Comment:
The agent probably becomes an implicit argument.
Rule statement: VP_Qual
[Ex. zitten -> zitten lekker]

het insertion
Syntactic change:

[1] A complement clause (sentence or infinitive) is always extraposed to the right in Dutch. Extraposition can be adjunction to the top S node or adjunction to VP. In main clauses extraposition remains hidden for the eye in most cases, because of V2. But in subordinate clauses the phenomenon shows up, since in subordinate clauses the verb is basically in final position and only extraposed clausal complements can follow the verb. For consistency extraposition must apply both in main clauses and subordinate clauses. A possibility would be to incorporate the extraposition rule in every 'Substitution NP-Clause' rule.

[2] This rule is not very productive in Dutch. It only occurs in archaic sounding expressions.

If there is no ea or a small clause subject is empty, then:
-insert het in ea position.
Rule statement: het_ins
[Ex. _regent -> het regent]

Middle alternation
Syntactic change:
If the VP is qualified by an advP and the ea position is empty, then:
-if there's a direct object, move the direct object to the ea position.
-if there's a PP-adjunct, delete the preposition and move the NP to ea position.
Rule statement: Middle_alt
[Ex. _zit lekker op deze stoel -> deze stoel zit lekker]

Swarm alternation
Syntactic change:
If the meaning of the SAF of an intransitive verb implies that there must be a 'swarm' of objects, denoted by the ea, then:
-PP-adjungate the ea to the VP with van as head of the PP.
Semantic change:
Change the 'swarm' connotation into the main meaning of the proposition:
'it is full of....'.
Rule statement: Swarm_alt
[Ex. de mensen stikken hier -> het stikt hier van de mensen]

Adv-to-Ea
Syntactic change:
If there's a locative advP and the ea position is empty, then:
-move the NP to the ea position.
Rule statement: Adv_to_DO
[Ex. _krioelt van de mieren in de keuken -> de keuken krioelt van de mieren]

PP-to-Direct Object
Syntactic change:
If there's a PP-adjunct that denotes the goal of the action that is denoted by the verb, then:
-remove the P and move the remaining NP to direct object position.
Rule statement: PP_to_DO
[Ex. hij schiet op fazanten -> hij schiet fazanten]

PP-to-Indirect Object
Syntactic change:
If there's a benefactive PP-adjunct, then:
-remove the P and move the remaining NP to indirect object position.
Rule statement: PP_to_IO
[Ex. Jan zegt iets tegen Tom -> Jan zegt Tom iets]

Preposition incorporation
Syntactic change:
If there's a PP-adjunct, then:
-incorporate the P in the beginning of the verb and move the remaining NP to direct object position.
Comment:
Often a verb with incorporated P becomes a new lexeme with its own meaning, so that the meaning can't be predicted from the compositional parts. (e.g. 'doorlopen' (= 'to walk through') which can mean something like 'to walk fast'.) But in this case the stress swithes to the preposition part. So, probably a lot of cases must be dealt with in the cross derivational rules.
Rule statement: Incorp_prep
[Ex. hij denkt over de zaak -> hij overdenkt de zaak]

be-alternation
Syntactic change:
If there's a PP-adjunct that denotes 'aboutness', goal or location, then:
-remove the P, turn the remaining NP into a direct object and prefix be to the verb.
Rule statement: be_alt
[Ex. Jan zingt over de liefde -> Jan bezingt de liefde]

Partitive object
Syntactic change:
If there's a quantified direct object NP, then:
-turn this NP into a partitive object, which means: take the quantifier apart and substitute er van with the NP, so that er precedes the quantifier and van follows the quantifier.
Comment:
er refers to the object that is quantified. This rule turns a quantified NP into a quantified PP.
Rule statement: PartObj
[Ex. Jan eet veel appels -> Jan eet er veel van]

Substitution NP - om-Inf
Syntactic change:
Replace an NP with an om-infinitive and extrapose the om-infinitival clause
Rule statement: NP_to_om-Inf
[Ex. Jan probeert iets -> Jan probeert om te komen]

Indirect object-to-PP
Syntactic change:
Turn an indirect object into a PP adjunct
Rule statement: IO_to_PP
[Ex. Jan geeft zijn moeder een kado -> Jan geeft een kado aan zijn moeder]

Indirect Object Passive
Syntactic change:
If there's an indirect object, then:
-change the agentive external argument into an implicit argument and passivize the verb.
-move the indirect object into ea position.
Semantic change:
This passive rule adds, as all passive rules do, an objective aspect to the meaning of the SAF.
Comment::
This rule is not very productive in Dutch and occurs in the standard language only in some frozen expressions.
Rule statement: IO_Pass
[Ex. de agent verzoekt ons door te lopen -> wij worden verzocht door te lopen]

er insertion
Syntactic change:
If a preposition doesn't have an NP complement, then:
insert er before the preposition
Rule statement: er_ins
[Ex. Jan denkt aan -> Jan denkt er aan]

Substitution of datS with propositional complement
Syntactic change:
If there's a datS with a predicative (?) function, then:
-replace the datS with an alsof clause.
Rule statement: datS_to_prop_compl
[Ex. Het lijkt dat Jan ziek is -> Het lijkt alsof Jan ziek is]

Propositional complement predication
Syntactic change:
If there's a manner adverb, then:
-replace it with a propositional complement, i.e. als-NP or alsof clause.
Rule statement: Pred_prop_compl
[Ex. Jan gedraagt zich slecht -> Jan gedraagt zich als een idioot]

Ea load alternation
Syntactic change:
If, in a frame with a theme ea, there is an intransitive verb small clause with a locative object, then:
-move the direct object to the ea position and turn the ea into a PP (which is headed by the P met)
Rule statement: ea_load_alt
[ex. het water stroomt het gat vol -> het gat stroomt vol met water]

Experiencer insertion
Syntactic change:
If a there's an agent in the SAF (explicit or implicit), then:
-modify theVP or the direct object with an adverbial phrase that denotes a 'too much' degree: 'te-Adv'
-insert an NP with the experiencer role in indirect object position.
Rule statement: Exp_ins
[Ex. hij leest boeken -> hij leest me te veel boeken]

Infinitive passive raising
Syntactic change:
If there's a SAF of laten3, then:
-raise the indirect object of the infinitive complement, if present, to the ea position of laten3, else
-raise the direct object of the infinitive complement to the ea position of laten3.
Comment:
laten3 has a passivizing function. It seems to be the only verb with this characteristic.
Rule statement: Inf_Pass_Rais
[Ex. _laten zich de oplossing raden -> de oplossing laat zich raden]

Noun Incorporation
Syntactic change:
If there's an inherent object, then:
-prefix the bare noun singular to the verb.
Rule statement: Incorp_N
[Ex. Hij speelt piano -> pianospelen]

Passive-short
Syntactic change:
If the frame contains an external argument that causes (active as an agent or passive being a theme) the action or activity that is instantiated by the verb, then:
-demote the external argument to an implicit argument
-put the verb in passive form

Passive-long
Syntactic change:
If the frame contains an external argument that causes the action or activity that is instantiated by the verb and contains no direct object reflexive, then:
-demote the external argument to a PP, headed by the preposition van
-put the verb in passive form

—

App. 9c

Derivations of Verbs[1]

(1) iv

hij_zingt:	base
wordt_gezongen:	base | Pass
er_zingt_iemand:	base | Exist_er_ins
het_zingt_lekker:	base | VP_Qual | het_ins
het_zit_lekker_op_deze_stoel:	base | PP_ad | VP_Qual | het_ins
deze_stoel_zit_lekker:	base | PP_ad | VP_Qual | Middle_alt
hij_rookt_me_teveel:	base | Exp_ins
hij_zingt_een_lied:	base | CogObj
wordt_een_lied_gezongen:	base | CogObj | Pass
het_lied_zingt_lekker:	base | CogObj | VP_Qual | Middle_alt
hij_schrijft_me_teveel_boeken:	base | CogObj | Exp_ins
hij_zingt_het_lied_uit:	base | CogObj | TV_smcl_AdvP
het_lied_uitzingen:	base | CogObj | TV_smcl_AdvP | Incorp_Adv
wordt_het_lied_uitgezongen:	base | CogObj | TV_smcl_AdvP | Incorp_Adv | Pass
de_klokken_luiden_de_feestdag_in:	base | IV_smcl_AdvP
de_feestdag_inluiden:	base | IV_smcl_AdvP | IncorpAdv
wordt_ingeluid_de_feestdag:	base | IV_smcl_AdvP | IncorpAdv | Pass
hij_zingt_zich_te_pletter:	base | IV_smcl_AdvP | DO_refl
zich_rotrennen:	base | IV_smcl_AdvP | DO_refl | Incorp_Adv

[1] I'm not sure of the grammaticality or derivation of the examples marked with "/".

TROLL: derivations of Dutch Vs

hij_loopt_de_schoenen_scheef:	base | IV_smcl_AP
de_schoenen_scheeflopen:	base | IV_smcl_AP | Incorp_A
worden_scheefgelopen_de_schoenen:	base | IV_smcl_AP | Incorp_A | Pass
hij_lacht_zich_dood:	base | IV_smcl_AP | DO_refl
zich_doodlachen:	base | IV_smcl_AP | DO_refl | Incorp_AP
hij_zingt_de_buren_het_huis_uit:	base | IV_smcl_PP
de_buren_het_huis_uitzingen:	base | IV_smcl_PP | Incorp_P
wordt_iemand_het_huis_uitgezongen:	base | IV_smcl_PP | Incorp_P | Pass
wordt_iemand_onder_de_voet_gelopen:	base | IV_smcl_PP | Pass
hij_zingt_zich_in_de_zevende_hemel:	base | IV_smcl_PP | DO_refl
ze_luiden_de_klok:	base | Ea_Caus
het_krioelt_van_de_mieren:	base | Swarm_alt | het_ins
de_keuken_krioelt_van_de_mieren:	base | Swarm_alt | Adv_to_ea
hij_gaat_weg:	base | SuPredad_advP
weggaan:	base | SuPredad_advP | Incorp_Adv
wordt_weggegaan:	base | SuPredad_advP | Incorp_Adv | Pass
hij_gaat_aan_land:	base | SuPredad_PP
hij_spreekt_met_Tom:	base | PP_ad
wordt_met_Tom_gesproken:	base | PP_ad | Pass
hij_spreekt_met_Tom_over_het_eten:	base | PP_ad | PP_ad
wordt_met_Tom_over_het_eten_gesproken:	base | PP_ad | PP_ad | Pass
hij_spreekt_Tom:	base | PP_ad | PP_to_DO
hij_spreekt_Tom_over_het_eten:	base | PP_ad | PP_to_DO | PP_ad
hij_schiet_fazanten:	base | PP_ad | PP_to_DO
worden_fazanten_geschoten:	base | PP_ad | PP_to_DO | Pass
hij_overdenkt_de_zaak:	base | PP_ad | Incorp_prep
hij_bezingt_de_liefde:	base | PP_ad | be_alt

TROLL: derivations of Dutch Vs

wordt_de_liefde_bezongen:	base | PP_ad | be_alt | Pass
hij_beschildert_zich:	base | PP_ad | be_alt | DO-refl
hij_speelt_viool:	base | InhObj
wordt_viool_gespeeld:	base | InhObj | Pass
pianospelen:	base | InhObj | Incorp_N
hij_schiet_kogels_op_Jan:	base | InhObj | PP_ad
worden_kogels_geschoten_op_Jan:	base | InhObj | PP_ad | Pass
boogschieten_op_de vijand:	base | InhObj | PP_ad | Incorp_N
hij_schiet_Tom_in_het_been:	base | PP_ad | Part_Wh_to_DO
wordt_geschoten_Tom_in_het_been:	base | PP_ad | Part_Wh_to_DO | Pass
hij_schiet_zich_door_het_hoofd:	base | PP_ad | Part_Wh_to_DO | DO_refl
ze_vliegt_Tom_om_de_hals:	base | PP_ad | Part_Wh_to_IO
wordt_Tom_om_de_hals_gevlogen:	base | PP_ad | Part_Wh_to_IO | Pass
hij_loopt_een_rondje:	base | Adv_to_DO
wordt_een_rondje_gelopen:	base | Adv_to_DO | Pass

(2) erg

kookt_het_water:	base
er_kookt_water:	base | Exist_er_ins
het_water_kookt:	base | Prom_to_ea
het_water_stroomt_de_kuil_vol:	base | Prom_to_ea | IV_smcl_AP
de_kuil_volstromen:	base | Prom_to_ea | IV_smcl_AP | Incorp_A
de_kuil_stroomt_vol_met_water:	base | Prom_to_ea | IV_smcl_AP | Ea_load_alt
een_stem_verheft_zich:	base | Erg_refl | Prom_to_ea
er_verheft_zich_een_stem:	base | Erg_refl | Exist_er_ins
het_gat_vult_zich_met_water:	base | Erg_refl | load_alt | Prom_to_ea

TROLL: derivations of Dutch Vs

hij_kookt_water:	base | Caus
hij_verheft_zich:	base | Caus | DO_refl
hij_vult_de_handen_met_snoep:	base | Caus | PP_ad | load_alt
hij_vult_haar_de_handen_met_snoep:	base | Caus | PP_ad | load_alt | IO_ins
hij_vult_zich_de_handen_met_snoep:	base | Caus | PP_ad | load_alt | IO_ins | IO_refl
hij_smelt_het_ijs_weg:	base | Caus | TV_smcl_AdvP
het_ijs_wegsmelten:	base | Caus | TV_smcl_AdvP | Incorp_Adv
wordt_weggesmolten_het_ijs:	base | Caus | TV_smcl_AdvP | Incorp_Adv | Pass
hij_kookt_de_eieren_hard:	base | Caus | TV_smcl_AP
de_eieren_hardkoken:	base | Caus | TV_smcl_AP | Incorp_A
worden_hardgekookt_de_eieren:	base | Caus | TV_smcl_AP | Incorp_A | Pass
hij_kookt_de_appels_tot_moes:	base | Caus | TV_smcl_PP
worden_de_appels_tot_moes_gekookt:	base | Caus | TV_smcl_PP | Pass
/hij_kookt_de_pan_door:	base | Caus | DO_del | IV_smcl_AdvP
/de_pan_doorkoken:	base | Caus | DO_del | IV_smcl_AdvP | Incorp_Adv
/wordt_doorgekookt_de_pan:	base | Caus | DO_del | IV_smcl_AdvP | Incorp_Adv | Pass
hij_kookt_de_pan_zwart:	base | Caus | DO_del | IV_smcl_AP
de_pan_zwartkoken:	base | Caus | DO_del | IV_smcl_AP | Incorp_A
wordt_zwartgekookt_de_pan:	base | Caus | DO_del | IV_smcl_AP | Incorp_A | Pass
hij_kookt_zich_ziek:	base | Caus | DO_del | IV_smcl_AP | DO_refl
hij_kookt_een_gat_in_de_pan:	base | Caus | DO_del | IV_smcl_PP
wordt_een_gat_in_de_pan_gekookt:	base | Caus | DO_del | IV_smcl_PP | Pass
hij_kookt_zich_het_huis_uit:	base | Caus | DO_del | IV_smcl_PP | DO_refl
het_ijs_smelt_weg:	base | TV_smcl_AdvP | Prom_to_ea
het_ijs_is_weggesmolten:	base | TV_smcl_AdvP | Incorp_Adv | Prom_to_ea
de_aardappels_koken_droog:	base | TV_smcl_AP | Prom_to_ea
de_aardappels_zijn_drooggekookt:	base | TV_smcl_AP | Incorp_A | Prom_to_ea

TROLL: derivations of Dutch Vs

de_appels_koken_tot_moes:	base | TV_smcl_PP | Prom_to_ea
/datS_komt_voor:	base | TV_smcl_AdvP | NP_to_datS | Anti_exp
/het_komt_voor_datS:	base | TV_smcl_AdvP | NP_to_datS | het_ins
/het_doet_zich_voor_datS:	base | TV_smcl_AdvP | NP_to_datS | Erg_refl | het_ins
het_ligt_eraan_ofS:	base | PP_ad | NP_to_ofS | er_ins | het_ins

(3) exp_iv

Jan_is_het_gezeur_beu:	base
Jan_heeft_het_warm:	base | del_DO | het_ins

(4) tv

hij_slaat_de_hond:	base
wordt_geslagen_de_hond:	base | Pass
er_slaat_iemand_een_hond:	base | Exist_er_ins
hij_wast_zich:	base | DO_refl
hij_eet_me_teveel_worst:	base | Exp_ins
hij_ziet_een_stier_komen:	base | AcI
de_bal_gooit_lekker:	base | VP_Qual | Middle_alt
ze_maakt_het_bed_op:	base | TV_smcl_AdvP
het_bed_opmaken:	base | TV_smcl_AdvP | Incorp_Adv
wordt_opgemaakt_het_bed:	base | TV_smcl_AdvP | Incorp_Adv | Pass
zij_maakt_zich_op:	base | TV_smcl_AdvP | DO_refl
zich_opmaken:	base | TV_smcl_AdvP | DO_refl | Incorp_Adv

TROLL: derivations of Dutch Vs

hij_wast_de_kleren_schoon:	base | TV_smcl_AP
de_kleren_schoonwassen:	base | TV_smcl_AP | Incorp_Adv
worden_de_kleren_schoongewassen:	base | TV_smcl_AP | Incorp_Adv | Pass
hij_voegt_water_bij_de_wijn:	base | TV_smcl_PP
wordt_water_bij_de_wijn_gedaan:	base | TV_smcl_PP | Pass
de_kogels_de_lucht_inschieten:	base | TV_smcl_PP | Incorp_P
worden_de_kogels_de_lucht_ingeschoten:	base | TV_smcl_PP | Incorp_P | Pass
hij_voegt_zich_bij_een_gezelschap:	base | TV_smcl_PP | DO_refl
ze_maakt_me_te_vaak_de_boel_schoon:	base | TV_smcl_* | Exp_ins
hij_drinkt:	base | DO_del
wordt_gedronken:	base | DO_del | Pass
er_eet_iemand:	base | DO_del | Exist_er_ins
hij_drinkt_me_teveel:	base | DO_del | Exp_ins
hij_eet_er_veel_van:	base | PartObj
het_drinkt_lekker_uit_dit_glas:	base | DO_del | PP_ad | VP_Qual | het_ins
dit_glas_drinkt_lekker:	base | DO_del | PP_ad | VP_Qual | Middle_alt
zij_doet_hem_tekort:	base | DO_del | IV_smcl_AdvP
iemand_tekortdoen:	base | DO_del | IV_smcl_AdvP | Incorp_Adv
zij_doet_zich_tekort:	base | DO_del | IV_smcl_AdvP | DO_refl | Incorp_Adv
wordt_hij_tekort_gedaan:	base | DO_del | IV_smcl_AdvP | Pass
worden_de_zorgen_weggedronken:	base | DO_del | IV_smcl_AdvP | Incorp_Adv | Pass
hij_eet_het_bord_leeg:	base | DO_del | IV_smcl_AP
het_bord_leegeten:	base | DO_del | IV_smcl_AP | Incorp_A
wordt_het_bord_leeggegeten:	base | DO_del | IV_smcl_AP | Incorp_A | Pass
Jan_eet_zich_dik:	base | DO_del | IV_smcl_AP | DO_refl
zich_diketen:	base | DO_del | IV_smcl_AP | Incorp_A | DO_refl

TROLL: derivations of Dutch Vs

hij_drinkt_Tom_onder_de_tafel:	base | DO_del | IV_smcl_PP
wordt_Tom_onder_de_tafel_gedronken:	base | DO_del | IV_smcl_PP | Pass
hij_drinkt_zich_onder_de_tafel:	base | DO_del | IV_smcl_PP | DO_refl
hij_drinkt_me_zich_te_vaak_onder_tafel:	base | DO_del | IV_smcl_* | Exp_ins
hij_eet_van_het_brood:	base | DO_to_PP
wordt_gegeten_van_het_brood:	base | DO_to_PP | Pass
hij_drinkt_de_koffie_zwart:	base | ObjQual
wordt_de_koffie_zwart_gedronken:	base | ObjQual | Pass
hij_zegt_iets_tegen_Tom:	base | PP_ad
wordt_iets_tegen_Tom_gezegd:	base | PP_ad | Pass
hij_zegt_iets_tegen_Tom_over_mij:	base | PP_ad | PP_ad
wordt_gezegd_iets_tegen_Tom_over_mij:	base | PP_ad | PP_ad | Pass
hij_zegt_Tom_iets:	base | PP_ad | PP_to_IO
wordt_Tom_iets_gezegd:	base | PP_ad | PP_to_IO | Pass
hij_zegt_Tom_iets_over_mij:	base | PP_ad | PP_to_IO | PP_ad
wordt_Tom_iets_over_mij_gezegd:	base | PP_ad | PP_to_IO | PP_ad | Pass
hij_laadt_de_wagen_met_hooi:	base | PP_ad | load_alt
wordt_de_wagen_met_hooi_geladen:	base | PP_ad | load_alt | Pass
hij_laadt_de_wagen:	base | PP_ad | load_alt | del_PP
wordt_de_wagen_geladen:	base | PP_ad | load_alt | del_PP | Pass
hij_gooit_Tom_een_boek_naar_het_hoofd:	base | PP_ad | Part_WH_to_IO
wordt_Tom_een_boek_naar_het_hoofd_gegooid:	base | PP_ad | Part_WH_to_IO | Pass
hij_gooit_zich_een_boek_naar_het_hoofd:	base | PP_ad | Part_WH_to_IO | IO_refl
hij_zegt_datS:	base | NP_to_datS
wordt_gezegd_datS:	base | NP_to_datS | Pass
hij_zegt_tegen_haar_datS:	base | NP_to_datS | PP_ad
wordt_tegen_haar_gezegd_datS:	base | NP_to_datS | PP_ad | Pass

TROLL: derivations of Dutch Vs

hij_zegt_Maria_datS:	base | NP_to_datS | PP_ad | PP_to_IO
wordt_Maria_gezegd_datS:	base | NP_to_datS | PP_ad | PP_to_IO | Pass
hij_zegt_wS:	base | NP_to_wS
wordt_gezegd_wS:	base | NP_to_wS | Pass
hij_zegt_tegen_Tom_wS:	base | NP_to_wS | PP_ad |
wordt_tegen_Tom_gezegd_wS:	base | NP_to_wS | PP_ad | Pass
hij_zegt_Maria_wS:	base | NP_to_wS | PP_ad | PP_to_IO
wordt_Maria_gezegd_wS:	base | NP_to_wS | PP_ad | PP_to_IO | Pass
hij_probeert_om-Inf:	base | NP_to_om-Inf
wordt_geprobeerd_om-Inf:	base | NP_to_om-Inf | Pass
hij_probeert_te-Inf:	base | NP_to_te-Inf
wordt_geprobeerd_te:	base | NP_to_te-Inf | Pass
hij_zegt_te-Inf:	base | NP_to_te-Inf
hij_zegt_tegen_Maria_te-Inf:	base | NP_to_te-Inf | PP_ad
hij_zegt_Maria_te-Inf:	base | NP_to_te-Inf | PP_ad | Pass

(5) th_tv

vloeipapier_absorbeert_inkt:	base
wordt_de_inkt_geabsorbeerd:	base | Pass

(6) exp_tv

hij_houdt_van_koffie:	base
hij_lust_de_koffie_zwart:	base | Depict

TROLL: derivations of Dutch Vs

hij_lust_er_veel_van:	base | PartObj

(7) ditr

Hij_geeft_Tom_geld:	base
wordt_Tom_geld_gegeven:	base | Pass
er_geeft_iemand_een_boek_aan_haar:	base | Exist_er_ins
hij_geeft_geld:	base | IO_del
wordt_geld_gegeven:	base | IO_del | Pass
er_geeft_iemand_een_boek:	base | IO_del | Exist_er_ins
hij_geeft_geld_aan_de_armen:	base | IO_to_PP
wordt_geld_aan_de_armen_gegeven:	base | IO_to_PP | Pass
hij_geeft_aan_de_armen:	base | IO_to_PP | DO_del
hij_geeft_zich_helemaal:	base | IO_del | DO_refl
hij_geeft:	base | IO_del | DO_del
wordt_gegeven:	base | IO_del | DO_del | Pass
hij_geeft_me_teveel_kado's:	base | IO_del | Exp_ins
hij_geeft_een_emmer_vol:	base | IO_del | DO_del | IV_smcl_AP
een_emmer_volgeven:	base | IO_del | DO_del | IV_smcl_AP | Incorp_A
wordt_een_emmer_volgegeven:	base | IO_del | DO_del | IV_smcl_AP | Incorp_A | Pass
hij_geeft_het_plan_op:	base | IO_del | DO_del | IV_smcl_AdvP
het_plan_opgeven:	base | IO_del | DO_del | IV_smcl_AdvP | Incorp_Adv
wordt_het_plan_opgegeven:	base | IO_del | DO_del | IV_smcl_AdvP | Incorp_Adv | Pass
/hij_stuurt_alles_in_de_war:	base | IO_del | DO_del | IV_smcl_PP
/wordt_alles_in_de_war_gestuurd:	base | IO_del | DO_del | IV_smcl_PP | Pass

TROLL: derivations of Dutch Vs

hij_geeft_er_veel_van_aan_Piet:	base | PartObj
er_wordt_veel_van_aan_Piet_gegeven:	base | PartObj | Pass
hij_geeft_er_veel_van:	base | IO_del | PartObj
er_wordt_veel_van_gegeven:	base | IO_del | PartObj
hij_geeft_het_geld_weg:	base | IO_del | TV_smcl_AdvP
het_geld_weggeven:	base | IO_del | TV_smcl_AdvP | Incorp_Adv
wordt_het_geld_weggegeven:	base | IO_del | TV_smcl_AdvP | Incorp_Adv | Pass
Peter_geeft_zich_over:	base | IO_del | TV_smcl_AdvP | DO_refl
zich_overgeven:	base | IO_del | TV_smcl_AdvP | Incorp_Adv | DO_refl
hij_stuurt_het_geld_het_land_uit:	base | IO_del | TV_smcl_PP
het_geld_het_land_uitsturen:	base | IO_del | TV_smcl_PP | Incorp_P
wordt_het_geld_het_land_uitgestuurd:	base | IO_del | TV_smcl_PP | Incorp_P | Pass
hij_geeft_Tom_een_stomp_in_de_rug:	base | IO_del | Part_Wh_to_IO
wordt_Tom_een_stomp_in_de_rug_gegeven:	base | IO_del | Part_Wh_to_IO | Pass
Peter_gunt_zich_tijd:	base | IO_refl
hij_belooft_haar_datS:	base | NP_to_datS
wordt_haar_beloofd_datS:	base | NP_to_datS | Pass
hij_beloofd_datS:	base | NP_to_datS | IO_del
wordt_beloofd_datS:	base | NP_to_datS | IO_del | Pass
hij_belooft_aan_haar_datS:	base | NP_to_datS | IO_to_PP
wordt_aan_haar_beloofd_datS:	base | NP_to_datS | IO_to_PP | Pass
hij_vraagt_Piet_ofS:	base | NP_to_ofS
wordt_Piet_gevraagd_ofS:	base | NP_to_ofS | Pass
hij_vraagt_aan_Piet_ofS:	base | NP_to_ofS | IO_to_PP
wordt_aan_Piet_gevraagd_ofS:	base | NP_to_ofS | IO_to_PP | Pass
hij_vraagt_of:	base | NP_to_ofS | IO_del
wordt_gevraagd_of:	base | NP_to_ofS | IO_del | Pass
zich_afvragen_of:	base | NP_to_ofS | IO_refl
hij_belooft_haar_om-Inf:	base | NP_to_om-Inf

TROLL: derivations of Dutch Vs

hij_belooft_om-Inf:	base I NP_to_om-Inf I IO_del
hij_belooft_haar_te-Inf:	base I NP_to_te-Inf
hij_belooft_te-Inf:	base I NP_to_te-Inf I IO_del
worden_de_krakers_verzocht_te-Inf:	base I IO_Pass

(8) refl

hij_schaamt_zich:	base
hij_schaamt_zich_voor_zijn_auto:	base I PP_ad
hij_schaamt_zich_ervoor_datS:	base I PP_ad I NP_to_datS I er_ins
hij_schaamt_zich_ervoor_om-Inf:	base I PP_ad I NP_to_om-Inf I er_ins
hij_schaamt_zich_ervoor_te-Inf:	base I PP_ad I NP_to_te-In I er_ins

(9) psych

verveelt_hem_het_boek:	base
het_boek_verveelt_hem:	base I Prom_to_ea
hij_verveelt_zich:	base I DO_del I ExpMvt
hij_interesseert_zich_voor_het_boek:	base I ExpMvt I DO_to_PP
/hij_is_verveeld:	base I DO_del I ExpAdj
/hij_is_geinteresseerd_in_het_boek:	base I ExpAdj I DO_to_PP
het_boek_verveelt:	base I Exp_del I Prom_to_ea
het_interesseert_Tom_datS:	base I NP_to_datS I het_ins

TROLL: derivations of Dutch Vs

datS_interesseert_Tom:	base | NP_to_datS | Anti_exp
het_interesseert_Tom_wS:	base | NP_to_wS | het_ins
wS_interesseert_Tom:	base | NP_to_wS | Anti_exp
het_interesseert_Tom_om-Inf:	base | NP_to_om-Inf | het_ins
het_verbaast_Tom_te:	base | NP_to_te-Inf | het_ins
te-Inf_bevalt_Tom:	base | NP_to_te-Inf | Anti-ext
hij_ergert_zich_erover_datS:	base | ExpMvt | NP_to_datS | er_ins
hij_interesseert_zich_ervoor_wS:	base | ExpMvt | NP_to_wS | er_ins
hij_interesseert_zich_ervoor_om-Inf:	base | ExpMvt | NP_to_om-Inf | er_ins
hij_verbaast_zich_erover_te-Inf:	base | ExpMvt | NP_to_ te-Inf | er_ins

(10) raisv1

blijken_datS:	base
het_blijkt_datS	base | het_ins
alles_blijkt_fout_te_zijn:	base | datS_to_Cl_inf | SubjRais
alles_blijkt_fout:	base | datS_to_Cl_inf | SubjRais | te-Inf_to_predic_AP

(11) raisv2

lijken_mij_datS:	base
het_lijkt_mij_datS	base | het_ins
het_lijkt_datS:	base | het_ins | IO_del
het_lijkt_me_alsof:	base | het_ins | datS_to_prop_compl
het_lijkt_alsof:	base | het_ins | datS_to_prop_compl | IO_del
Tom_lijkt_mij_ziek_te_zijn:	base | datS_to_CL_inf | SubjRais
Tom_lijkt_ziek_te_zijn:	base | datS_to_CL_inf | SubjRais | IO_del
Tom_lijkt_mij_ziek:	base | datS_to_CL_inf | SubjRais | te_inf_to_predic_AP
Tom_lijkt_ziek:	base | datS_to_CL_inf | SubjRais | te_inf_to_predic_AP | IO_del

TROLL: derivations of Dutch Vs

(12) raisv3

lijken_op_iets:	base
het_lijkt_op_iets:	base | het_ins
het_lijkt_erop_datS:	base | het_ins | NP_to_datS | er_ins

(13) depict

dat_hij_zich_Tom_voorstelt:	base
er_stelt_zich_iemand_Tom_voor:	base | Exist_er_ins
hij_stelt_zich_voor_datS:	base | NP_to_datS
hij_stelt_zich_Tom_gezond_voor:	base | NP_to_datS | datS_to_SC

(14) smcl

men beschouwt_Tom_als_een_genie:	base
wordt_Tom_als_genie_beschouwd:	base | Pass
er_beschouwt_iemand_Tom_als_geniaal:	base | Exist_er_ins
hij_beschouwt_zich_als_een_genie:	base | DO_refl

(15) iv_loc

hij_woont_in_de_stad:	base
wordt_in_dat_krot_gewoond:	base | Pass

TROLL: derivations of Dutch Vs

er_woont_iemand_in_dat_krot:	base | Exist_er_ins
het_woont_lekker_in_de_stad:	base | PP_ad | VP_Qual | het_ins
Tilburg_winkelt_afschuwelijk:	base | PP_ad | VP_Qual | Middle_alt
(16) refl_loc	
dat_hij_zich_ophoudt_in_Noorwegen:	base
er_houdt_zich_iemand_op_in_Noorwegen:	base | Exist_er_ins
(17) tv_loc	
hij_zet_de_bloemen_op_de_tafel:	base
worden_de_bloemen_op_tafel_gezet:	base | Pass
(18) ind_arg	
hij_verlangt_naar_haar:	base
wordt_naar_haar_verlangd:	base | Pass
er_verlangt_iemand_naar_haar:	base | Exist_er_ins
(19) di_ind_arg	
dat_hij_bij_Marie_op_iets_aandringt:	base
wordt_zijn_leven_aan_de_studie_gewijd:	base | Pass
er_wijdt_iemand_zijn_leven_aan_de_studie:	base | Exist_er_ins

TROLL: derivations of Dutch Vs

Peter_wijdt_zich_aan_de_studie:	base I DO_refl
Peter_verzekert_zich_van_de_overwinning:	base I IO_refl
(20) ind_arg_refl	
ze_verdiepen_zich_in_het_spoorboekje:	base
er_verdiept_zich_iemand_in_de_krant:	base I Exist_er_ins
(21) weather	
sneeuwt:	base
het_stinkt:	base I het_ins
het_hagelt_grote_korrels:	base I het_ins I CogObj
het_hagelt_grote_korrels_Vens:	base I het_ins I CogObj I ObjQual
(22) measure	
de_steen_weegt_3_kg:	base
(23) repr	
dat_de_foto_Tom_voorstelt:	base
er_stelt_iemand_de_koning_voor:	base I Exist_er_ins
(24) appellat1	

TROLL: derivations of Dutch Vs

hij_benoemt_hem_tot_opperhoofd:	base
wordt_benoemd_Jan_tot_opperhoofd:	base I Pass
er_benoemt_iemand_Jan_tot_chef:	base I Exist_er_ins
Tom_benoemt_zich_tot_president:	base I DO_refl
Tom_benoemt_Piet:	base I del_PP
wordt_Piet_benoemd:	base I del_PP I Pass

(25) appellat2

hij_noemt_Jan_een_genie:	base
wordt_genoemd_Tom_een_genie:	base I Pass
er_noemt_iemand_Jan_een_genie:	base I Exist_er_ins
hij_noemt_zich_geniaal:	base I DO_refl

(26) refl_manner

hij_gedraagt_zich_uitstekend:	base
er_gedraagt_zich_iemand_uitstekend:	base I Exist_er_ins
hij_gedraagt_zich_als_een_idioot:	base I Predic_als
hij_gedraagt_zich_alsof_hij_idioot_is:	base I Predic_Prop_compl

(27) erg_ditv

wacht_Tom_een_ongeluk:	base

er_wacht_Tom_een_ongeluk:	base | Exist_er_ins
een_ongeluk_wacht_Tom:	base | Prom_to_ea
(28) ben_tv	
hij_betaalt_haar:	base
wordt_zij_betaald:	base | Pass
er_betaalt_iemand_de_werkster:	base | Exist_er_ins
hij_betaalt_aan_haar:	base | IO_to_PP
hij_betaalt_haar_voor_het_werk:	base | PP_ad
wordt_voor_het_werk_betaald:	base | PP_ad | Pass
hij_betaalt:	base | IO_del
wordt_betaald:	base | IO_del | Pass
het_betaalt_gemakkelijk_met_checks:	base | IO_del | PP_ad | VP_Qual | het_ins
checks_betalen_gemakkelijk:	base | IO_del | PP_ad | VP_Qual | Middle_alt
(29) cop	
zijn_ik_ziek:	base
ik_ben_ziek:	base | Prom_to_ea
er_is_iemand_ziek:	base | Exist_er_ins
dat_het_gaat_vriezen_is_zeker:	base | NP_to_datS | Anti_exp
het_is_zeker_dat_het_gaat_vriezen:	base | NP_to_datS | het_ins

TROLL: derivations of Dutch Vs

wie_er_komt_is_bekend:	base | NP_to_wS | Anti_exp
het_is_bekend_wie_er_komt:	base | NP_to_wS | het_ins
het_is_goed_te_trainen:	base | NP_to_te-Inf | het_ins

(30a) laten1

hij_laat_Piet_spaghetti_maken:	base
er_laat_iemand_Piet_buiten_spelen:	base | Exist_er_ins
hij_laat_haar_ombrengen:	base | IO_del
hij_laat_zich_pesten:	base | IO_del | DO_refl

(30b) laten2

hij_laat_de_jas_hangen:	base
er_laat_iemand_een_boek_op_tafel_liggen:	base | Exist_er_ins
hij_laat_zich_gaan:	base | DO_refl

(30c) laten3

laten_een oplossing_vinden:	base
laten_zich_een_oplossing_vinden:	base | Erg_refl
een_oplossing_laat_zich_vinden:	base | Erg_refl | Inf_Pass_Rais
er_laat_zich_een_oplossing_vinden:	base | Erg_refl | Inf_Pass_Rais | Exist_er_ins