A grammar design accommodating packed argument frame information on verbs

Petter Haugereid

Division of Linguistics and Multilingal Studies, Nanyang Technological University,
14 Nanyang Drive, Singapore 637332, Singapore
petterha@ntu.edu.sg

Abstract. This paper presents a comparison of two designs for implementing argument frame information on verbs. In the first design, alternating verbs will be represented with one lexical entry pr. possible argument frame. In the other design, each verb form will be associated with only one lexical entry containing a packed representation of the possible argument frames. The first design represents how valence alternations are treated in lexicalist grammars, while the second shows how valence alternations can be handled in a constructionalist grammar. The comparison is done with an implemented "deep" grammar of Norwegian.

Keywords: HPSG, grammar engineering, lexical representations, under-specification.

1 Introduction

In most "deep" grammar implementations, information about the argument frame of a verb is specified in the lexicon. This can be observed in grammatical frameworks such as Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994), Lexical Functional Grammar (LFG) (Bresnan, 2001), and Combinatory Categorial Grammar (CCG) (Steedman, 2000). Deep grammars need to be as precise as possible since they are not only expected to parse grammatical sentences, but also *not* to parse ungrammatical sentences. So, in order to avoid overgeneration, the grammar needs, among numerous other things, to contain information about what syntactic frames a verb is expected to appear in. The most natural place to put this information is in the lexicon.

In this paper I will discuss one problem associated with the way argument frame information is specified in a lexicalist grammar, namely the use of multiple lexical entries for one verb form in cases where the verb may appear in more than one argument frame. Multiple lexical entries for one form leads to an increased processing effort for the parser. A possible solution to this problem is to pack the information from the different entries into one. I will present a constructionalist grammar design, where each verb form is assigned a single lexical entry with a packed representation of the possible argument frames of the verb, using a type hierarchy of argument frame types to account for argument frame alternations. I will show how the grammar design compares to HPSG, which is the framework mostly used for deep grammar implementations, and compare two versions of the implemented grammar, one with an expanded lexicon and one with a packed lexicon, with regard to competence and performance.

2 Valence in HPSG

In HPSG, a transitive verb has the information in Figure 1. The figure illustrates how the valence requirements of a verb is represented by means of lists and also how the linking to the semantics is accounted for.

$$\begin{bmatrix} \text{PHON} & \left\langle \textit{admire} \right\rangle \\ \text{CAT} & \begin{bmatrix} \text{HEAD} & \textit{verb} \\ \text{VAL} & \begin{bmatrix} \text{SUBJ} & \left\langle \text{NP}_{\boxed{1}} \right\rangle \\ \text{COMPS} & \left\langle \text{NP}_{\boxed{2}} \right\rangle \end{bmatrix} \end{bmatrix} \\ \text{CONT}\,|\,\text{RESTR} & \left\langle \begin{bmatrix} \text{PRED} & \textbf{\_admire\_v\_rel} \\ \text{ARG1} & \boxed{1} \\ \text{ARG2} & \boxed{2} \end{bmatrix} \right\rangle \end{bmatrix}$$

Figure 1: Lexical entry for the verb *admire*

For each argument realized by the valence rules (Head-Complement Rule and Head-Subject Rule), an element on the valence lists will be unified with it and checked off. A verb projection is accepted as a sentence when both the SUBJ list and the COMPS list are empty.

Implemented HPSG grammars like the English Resource Grammar, ERG (Flickinger, 2000), use a binary Head-Complement Rule to realize the complements (see Figure 2.) This rule realizes one complement at a time (the first) and links the rest of the list in the mother ($\boxed{3}$) (see Sag *et al.* (2003, 97)). If the complement list contains more than one element, the Head-Complement Rule will work repeatedly until the COMPS list is empty.[1]

$$\begin{bmatrix} \textit{phrase} & \\ \text{CAT} & \begin{bmatrix} \text{HEAD} & \boxed{1} \\ \text{VAL}\,|\,\text{COMPS} & \boxed{3} \end{bmatrix} \end{bmatrix} \Rightarrow \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \boxed{1} \\ \text{VAL}\,|\,\text{COMPS} & \left\langle \boxed{2} \right\rangle \oplus \boxed{3} \end{bmatrix} \end{bmatrix}, \boxed{2}$$

Figure 2: Binary Head-Complement Rule

The subject of a clause is realized by the Head-Subject Rule (see Figure 3). This rule has as its head daughter a word or phrase that has an empty COMPS list and an element on the SUBJ list ($\boxed{2}$). The element on the SUBJ list is realized as the non-head daughter, and the SUBJ list of the mother is empty.

$$\begin{bmatrix} \textit{phrase} & \\ \text{CAT} & \begin{bmatrix} \text{HEAD} & \boxed{1} \\ \text{VAL} & \begin{bmatrix} \text{SUBJ} & \langle\rangle \\ \text{COMPS} & \langle\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \Rightarrow \boxed{2}, \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \boxed{1} \\ \text{VAL} & \begin{bmatrix} \text{SUBJ} & \left\langle \boxed{2} \right\rangle \\ \text{COMPS} & \langle\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 3: Head-Subject Rule

The fact that linking is accounted for in the lexicon means that a verb needs to have access to its syntactic arguments in the lexicon. It is therefore difficult to account for verbs with more than one argument frame with only one lexical entry. One approach to this problem is described in Flickinger (2000), which implements a special type of lists in order to account for optional arguments. A list is accepted as empty if all elements on it are marked as OPT +. This approach works well for alternations like the intransitive/transitive alternation, but does not completely eliminate the need for multiple lexical entries (or non-inflectional lexical rules).[2] There are also methods for

[1] By assuming such binary structures, rather than the flat Head-Complement Rule from (Pollard and Sag, 1994, 362–363), the account of adjuncts intervening the complements becomes more straightforward, since a Head-Modifier Rule can be allowed to apply in between two instances of the Head-Complement Rules.

[2] The treatment of ditransitive verbs in combination with passive becomes more challenging, since the passive lexical

making parsing more efficient where local ambiguities are packed while parsing is going on, as shown in Oepen and Carroll (2000), where the packing is done by the parser.

3 Norsyg

Norsyg is a typed feature structure grammar for Norwegian, developed with resources from the open-source repository of the Deep Linguistic Processing with HPSG Initiative (DELPH-IN),[3] in particular the LKB system (Copestake, 2002), which is a software for parsing and grammar development, and the HPSG Grammar Matrix (version 0.8) (Bender *et al.*, 2002), which the grammar Norsyg orignially is based on. Evaluation is done with the [incr tsdb()] system (Oepen and Flickinger, 1998).

Even though much of the feature geometry of the Grammar Matrix has been kept, the design is radically different from that of a standard HPSG grammar. It is now more a constructionalist grammar, rather than lexicalist, inspired by the approach in Borer (2005a,b) and Åfarli (2007).

Norsyg (version 2011-07-28) is a grammar with 4,454 types, 133 features, 144,679 lexical entries (of which 1,436 are hand-built), 52 grammar rules, and 51 lexical rules. In comparison, the ERG (version 2010-10-01) has 7,682 types, 202 features, 35,473 lexical entries, 175 grammar rules, and 73 lexical rules, and JACY (Jacy Japanese Grammar, version 2009-07-05) (Siegel and Bender, 2002) has 2524 types, 183 features, 56,944 lexical items, 51 grammar rules, and 69 lexical rules.

3.1 Linking types

In the approach taken in Norsyg, linking is done in the syntax rather than in the lexical types. Instead of assuming that a lexical entry has detailed information about a certain syntactic frame, which is crucial in an approach that does linking in the lexicon (see Figure 1), it is assumed that a lexical entry by default has little information about its syntactic environment. The syntactic frames are not projections of the lexicon. They are rather constructions made up of *functional signs*, that is inflections, closed class lexical items, and syntactic rules. These signs do the linking of the arguments of the open class lexical items that enter the syntactic frames, realizing what I refer to as subconstructions as they serve as part of a larger, overall construction. In order to avoid overgeneration, the open class lexical items are provided with information that restricts the number of argument frames they can enter.

3.2 Four valence features

In the implementation of a grammar that does linking by means of functional signs realizing subconstructions, I make use of four valence features (ARG1, ARG2, ARG3 and ARG4), corresponding to what in GB is referred to as the "external argument" (ARG1), the "direct object internal argument" (ARG2), the "indirect object internal argument" (ARG3), and "goal/locative oblique" (ARG4). The four features have *synsem* as value.[4] The type *synsem* is given the feature LINK. The value of the LINK feature is the type *link*. In addition, there is a feature ARGFRAME with the value *link*. It is via this feature that a lexeme may put restrictions on what types of constructions it can enter. There is also a feature PART which allows a lexeme to select for particles. The type *valence* now has the definition in Figure 5, rather than the definition with the SUBJ and COMPS lists as presented in Figure 4.

rule looks for the first element on the COMPS list to promote it to subject, and will not be able to find the second element in case the first is not realized.

[3] http://www.delph-in.net.

[4] The type *synsem* is a supertype of *phr-synsem* and *lex-synsem*. This makes it compatible with both words and phrases.

$$\begin{bmatrix} \textit{valence} & \\ \text{SUBJ} & \textit{list} \\ \text{SPR} & \textit{list} \\ \text{COMPS} & \textit{list} \\ \text{SPEC} & \textit{list} \end{bmatrix}$$

Figure 4: The type *valence* in the Grammar Matrix

$$\begin{bmatrix} \textit{valence} & \\ \text{ARGFRAME} & \textit{link} \\ \text{ARG1} & \textit{synsem}\begin{bmatrix}\text{LINK} & \textit{link}\end{bmatrix} \\ \text{ARG2} & \textit{synsem}\begin{bmatrix}\text{LINK} & \textit{link}\end{bmatrix} \\ \text{ARG3} & \textit{synsem}\begin{bmatrix}\text{LINK} & \textit{link}\end{bmatrix} \\ \text{ARG4} & \textit{synsem}\begin{bmatrix}\text{LINK} & \textit{link}\end{bmatrix} \\ \text{PART|SAT} & \textit{bool} \end{bmatrix}$$

Figure 5: The type *valence* in Norsyg

3.3 A hierarchy of linking types

The type *link* has a hierarchy below it. Directly under *link*, there are eight types, one positive and negative type for each of the valence features in Figure 5 (see Figure 6).[5] So there is one *arg1+*, one *arg1–*, one *arg2+*, one *arg2–* and so on.

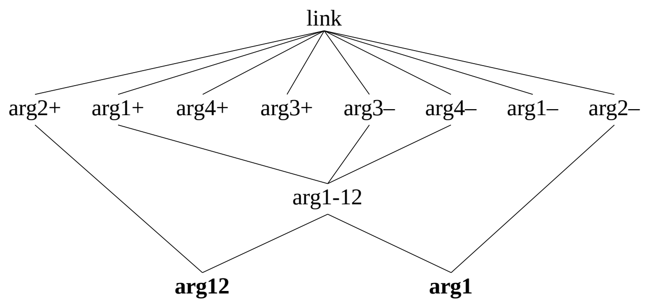

Figure 6: The *link* hierarchy

Each of the types in the bottom of the hierarchy inherits from four of the top types. These types represent different argument frames. For instance, the type *arg12* represents an arg12-construction, which is the frame type for transitive verbs like *devour* in *John devoured the pizza*. The type *arg1* is the type for unergative intransitive verbs like *smile* in *John smiled*. If we study the hierarchy above the bottom types, we see that *arg12* is a subtype of *arg1+*, *arg2+*, *arg3–*, and *arg4–*, and that *arg1* is a subtype of *arg1+*, *arg2–*, *arg3–*, and *arg4–*.

3.4 *Packing* of argument frames

The intermediate types in the hierarchy are inserted in order to allow something that can be thought of as *packing* of argument frames.[6] These types have two or more bottom types as subtypes. So a verb that is specified in the lexicon with an intermediate link type will be compatible with all the frames that correspond to the subtypes of the intermediate link type.

The verb *eat* can occur with two valence frames, as illustrated in (1).[7]

(1) a. John eats.
    b. John eats an apple.

[5] The hierarchy in Figure 6 is very simplified. Most of the intermediate and bottom types are left out in order to keep the illustration as simple as possible.

[6] The term *packing* was suggested to me by Lars Hellan.

[7] Passive and presentational variants of the examples I am using are not assumed to alter the argument frame, so I do not mention them here.

In (1a) *eat* has an arg1-frame, and in (1b) an arg12-frame. In order to allow the verb to enter both frames, it is given the ARGFRAME value *arg1-12* in the lexicon. *arg1-12* inherits from *arg1+*, *arg3-*, and *arg4-*, but is underspecified with regard to arg2. It has two subtypes, namely *arg1* and *arg12*, which means that *eat* can enter the relevant argument frames.

A verb like *break* can enter the frames illustrated in (2).

(2) a. John broke the cup.
 b. John broke the cup to pieces.
 c. The cup broke.
 d. The cup broke to pieces.

(2a) has a transitive frame (arg12-construction), (2b) has a transitive + resultative frame (arg124-construction), (2c) has an unaccusative frame (arg2-construction) and (2d) has an unaccusative + resultative frame (arg24-construction). In order to allow *break* in all these frames, it is specified with the intermediate link-type *arg12-124-2-24*, which has the four subtypes *arg12*, *arg124*, *arg2* and *arg24* (not displayed in Figure 6). In all, the hierarchy below the type *link* consists of 126 types. The most frequent argument frame types with linguistic definitions are given in Table 1. It shows that the most common alternation grouping among the verbs is the transitive/intransitive alternation, with 1815 occurrences. In comparison, the number of verbs alternating between (only) transitive and ditransitive is 56.

Table 1: The 16 most frequent argument frame types in Norsyg

Frequency	**Argument frame type**	**Alternation grouping**
3335	arg12	Transitive
1815	arg1-12	Transitive/intransitive
627	arg12-124	Transitive with optional delimiter
513	arg1	Unergative intransitive
341	arg1-14	Intransitive with optional PP complement
338	arg2	Unaccusative intransitive
139	arg12-14	Transitive/intransitive with PP complement (if intransitive)
124	arg1-12-14	Transitive/intransitive with optional delimiter (if intransitive)
114	arg14	Intransitive with PP complement
105	arg12-2	Transitive/unaccusative intransitive (causative/inchoative)
81	arg12-124-14	Intransitive/transitive with optional PP complement
76	arg124	Transitive with PP complement
75	arg1-2	Unergative/unaccusative intransitive
74	arg123	Ditransitive
59	arg12-123-124	Transitive/ditransitive with optional delimiter (if transitive)
56	arg12-123	Transitive/ditransitive

3.5 The composition of subconstructions

Figure 7 gives a simplified illustration of how the information about realized subconstructions in the syntax and argument structure information specified on the main verb is represented.[8] As the figure shows, each valence rule switches a negative LINK value in the mother to a positive

[8] This tree does not reflect the fact that syntactic structures produced by Norsyg actually are left-branching, with the initial constituent at the bottom-left.

LINK value in the daughter. The top node has only negative LINK values. In this way, the LINK values in the bottom of the tree reflect what subconstructions are realized higher up in the tree. The argument structure information specified on the main verb is given as value of the feature ARGFRAME (*arg1-12)*. The tree also illustrates how linking is done by subconstructions, rather than in the lexicon. The semantic representation composed in Figure 7 is a decomposed version of the semantic relation in Figure 1. The relations introduced by the subconstructions in Figure 7 can be seen as Parsoninan sub-events.

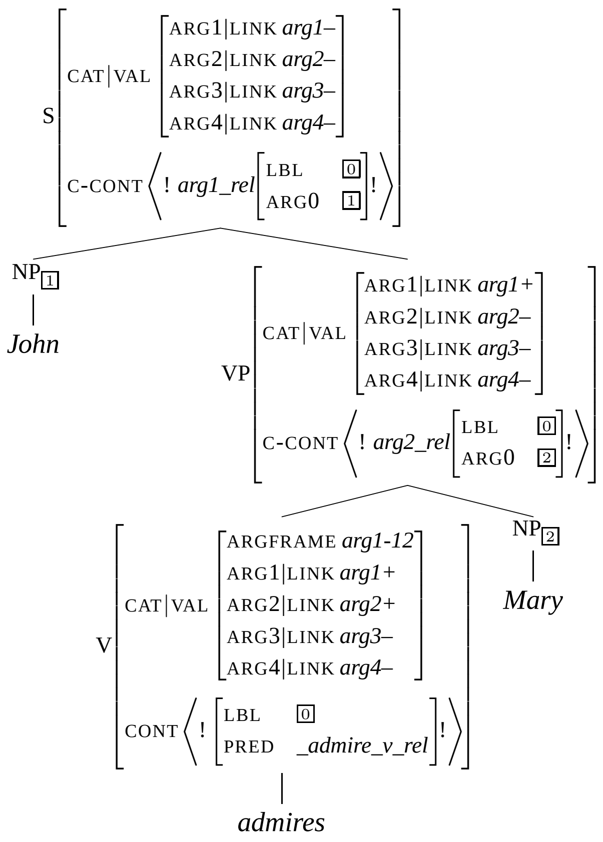

Figure 7: Information about realized subconstructions.

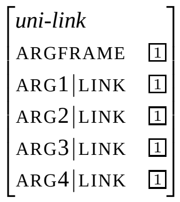

Figure 8: Unification of LINK values and ARGFRAME value.

The type *uni-link* (see Figure 8) unifies the LINK values with the argument structure information specified on the main verb (the value of ARGFRAME). This type applies to constituents at the bottom of the tree where the linking information is available.[9] In the analysis of a transitive sentence like that in Figure 7, the types *arg1+*, *arg2+*, *arg3–*, *arg4–*, and *arg1-12* will be unified. This gives the type *arg12* (see Figure 6).

3.6 Lexical types in Norsyg

There are 100 handwritten and 126 automatically derived lexical entry types for verbs in Norsyg. The lexical type for a transitive verb with an optional NP object, like *eat*, is presented in Figure 9. The feature ARGFRAME is given the value *arg1-12*, which means that the verb is compatible with both the unergative intransitive frame (arg1-construction) and the transitive frame (arg12-construction). The HEAD value of the (optional) ARG2 of the verb is specified to be *nominal*. Since optionality is expressed by means of the argument frame type, there is no need for the

[9] This unification is left out in Figure 7 in order to show how the linking types end up at the bottom of the tree.

feature OPT on syntactic arguments. The PART|SAT value is *plus*, which means that the verb is not a particle verb.

$$\begin{bmatrix} \textit{arg1-12\_np\_le} & \\ \textsc{cat}|\textsc{val} & \begin{bmatrix} \textsc{argframe}\ \textit{arg1-12} \\ \textsc{arg2}|\textsc{cat}|\textsc{head}\ \textit{nominal} \\ \textsc{part}|\textsc{sat}\ + \end{bmatrix} \end{bmatrix}$$

Figure 9: The *arg1-12_np_le* type

The lexical type for verbs like *paint*, which can be both intransitive, transitive and transitive resultative, is given in Figure 10. The ARGFRAME value is specified as *arg1-12-124*, which means that it can enter an unergative frame, a transitive frame, and a transitive frame with a delimiter. The HEAD value of ARG2 is specified to be *nominal*, and the HEAD value of ARG4 is specified to be *adj*. This ensures that the internal argument is an NP, and that the delimiter is an adjective.

$$\begin{bmatrix} \textit{arg1-12-124\_np\_ap\_le} & \\ \textsc{cat}|\textsc{val} & \begin{bmatrix} \textsc{argframe}\ \textit{arg1-12-124} \\ \textsc{arg2}|\textsc{cat}|\textsc{head}\ \textit{nominal} \\ \textsc{arg4}|\textsc{cat}|\textsc{head}\ \textit{adj} \end{bmatrix} \end{bmatrix}$$

Figure 10: The *arg1-12-124_np_ap_le* type

Given the means I have described for restricting the syntactic environment of verbs in Norsyg, the ARGFRAME values, the HEAD values of the ARG2 and ARG4 arguments, the KEY value of the ARG4 argument, and the PRED value of the particles, one is free to give very specific constraints, only allowing one particular argument frame, or one can let the constraints be less specific, so that the verb can enter more frames.

3.7 Adaptation of Norsk Ordbank

Norsyg is adapted to Norsk Ordbank,[10] which is a full-form lexicon for Norwegian with 1,179,549 entries (148,141 different lemmas). The verbs in Norsk Ordbank are annotated with the argument frame information from the NorKompLeks project.[11] A program *convlex* converts the lexicon into a format compatible with the Norsyg grammar (143,263 uninflected lexical entries, of which 8,647 are verbs). It gathers the argument frame information about each verb and creates the corresponding type if this type does not exist already. This is often necessary if a verb can enter many argument frames. The lexical types for verbs have five kinds of information. First, they specify what kind of constructions the verb can enter. If the verb can enter the arg1-construction, the arg12-construction, and the arg124-construction, it is assigned the ARGFRAME value *arg1-12-124*. Second, they specify the HEAD value of the ARG2 argument (if applicable). If the ARG2 is either an NP or a subordinate clause, the new verb lexical entry type inherits from the type *arg2_cp-np*. Third, the ARG3 value is specified to be a reflexive (if applicable). Forth, the new verb lexical types specify the ARG4 value (if applicable). If the ARG4 value is a PP, the type inherits from the type *arg4_pp*. Fifth, the new verb lexical entry type specifies whether the verb is a particle verb. If it is a particle verb, it inherits from the type *part-verb*, and if not, it inherits from *non-part-verb*. Other information, like the PRED values of selected particles and prepositions, is specified on each

[10] http://www.edd.uio.no/prosjekt/ordbanken/

[11] NorKompLeks (NKL) is a Norwegian computational lexicon developed at NTNU, Trondheim, Norway. It contains information about inflectional patterns and phonological representations as well as argument structure frames for verbs. There are 105 different codes for argument structure frames in NKL, and each verb is provided with a list of codes showing the possible argument structure frames.

individual lexical entry. Based on the argument frame information specified on verbs in Norsk Ordbank, the lexicon conversion program builds 126 new types for verb lexical entries in addition to the 100 lexical entry types for verbs that already exist. An example of an automatically created verb lexical type is given in (3).

(3)
```
arg12-124-2_part_np_pp_le := arg2_np & arg4_pp & part-verb &
  [ SYNSEM.LOCAL.CAT.VAL.ARGFRAME arg12-124-2 ].
```

The type in (3) is the type for the verbs *etse* ('corrode'), *helle* ('pour'/'slope'), *hive* ('throw'), *kippe* ('flip up'), and *knalle* ('crack'). What these verbs have in common, is that they can enter the arg12-construction, the arg124-construction, and the arg2-construction, hence the ARGFRAME value *arg12-124-2*. The verbs are particle verbs, so the type inherits from *part-verb*. The verbs require an NP as value of ARG2 and a PP as value of ARG4 (if applicable), so the type inherits from *arg2_np* and *arg4_pp*.

The entry of the infinitival form of *helle* in Norsk Ordbank is given in (4), where the fields in angle brackets show what argument frames the verb can enter, `<intrans2>`, `<adv6>`, and `<part1/ut>`.

(4)
```
27112 helle helle verb inf <intrans2> <adv6> <part1/ut> 021 1
```

These argument frame specifications are translated into the type in (3) according to a table distributed with the Norsyg grammar ('nkl2lkb.txt'). When appearing alone, `<intrans2>` translates into the type *arg2_np_le* (the type for intransitive unaccusative verbs), `<adv6>` translates into the type *arg124_np_pp_le* (the type for transitive verbs with PP complements), and `<part1/ut>` translates into the type *arg12_part_np_le* (the type for transitive particle verbs (the PRED value of the particle *ut* ('out') is specified on the lexical entry)). When these three argument frames appear on the same lexical entry, the type *arg12-124-2_part_np_pp_le* is created, as shown above. It accommodates all the frames just mentioned.The lexical entry of *helle* in the Norsyg grammar is given in (5).

(5)
```
helle-v := arg12-124-2_part_np_pp_le &
[ STEM <"helle">,
  INFLECTION v1,
  SYNSEM.LKEYS.ALTKEYREL.PRED _ut_p_rel,
  SYNSEM.LKEYS.KEYREL.PRED "_helle_v_rel" ].
```

4 Comparison of 'packed' vs. expanded lexicon

In order to check the impact of a lexicon with packed argument frame representations as described in the previous section, I used the *convlex* program to generate two versions of the lexicon. In the first version, all verbs were given packed representations, and in the other, each argument frame version of a verb was spelled out as a separate lexical entry.

This means that a verb that has the type *arg1-12_np_le* (see Figure 9) in the packed lexicon, in the expanded version is given two lexical entries, one of the type *arg1_le* and one of the type *arg12_np_le* (one for each of the argument structure codes assigned by the original Norsk Ordbank lexicon). 5,329 of the verbs from the Norsk Ordbank lexicon are listed with only one frame, and are therefore given only one lexical entry in the expanded lexicon, while 3,318 verbs are listed with more than one argument frame and are given the corresponding number of lexical entries. This gave me an expanded lexicon with 12,213 lexical entries for verbs, rather than the 8,647 lexical entries for verbs in the packed lexicon, an increase of 3,566.

The data used for the comparison are taken from a 37 million word Norwegian Wikipedia corpus (2,252,972 sentences). I selected 8,271 sentences containing 5-10 words where all the words were covered by the dictionary of the grammar. This set is referred to as *All items* in the next section. The grammar had a coverage of 5,105 sentences with the packed lexicon and 5,078 sentences with the expanded lexicon. Of the sentences that were parsed with both version of the lexicon, 3,446 sentences were given the same number of analyses with both versions. This set is referred to as *Equal coverage* in the next section. I also created a third set of sentences where I excluded the sentences containing the copula verb '*å være*' 'to be' from the *Equal coverage* set, since they are likely to be overrepresented in the data. (Typical short sentences in the Wikipedia data are sentences like *Lesotho er et land i Afrika* 'Leshoto is a country in Africa'.) This set is referred to as *No copula* in the next section and has 1,902 items.

5 Results

I let the grammar loaded with the two different lexicons (*packed* and *expanded*) parse the three sets of sentences described in the previous section (*All items*, *Equal coverage*, and *No copula*) and compared the results. Table 2 shows that the two versions of the grammar, as already noted, have similar coverage on the *All items* set (61.4% and 61.7%), and, as expected, the same coverage (100%) on *Equal coverage*, and *No copula*. The two versions produce slightly more analyses on average with the expanded lexicon (23.69) than the packed lexicon (20.58) for *All items*. For the *Equal coverage*, and *No copula* sets, the two versions (as expected) produce the same number of analyses (14.39 and 13.66, respectively). More importantly, the table also illustrates the difference in lexical ambiguity of the two grammars. The difference in lexical ambiguity of expanded vs. packed is 4.20 vs. 3.41 on *All items*, 3.73 vs. 3.20 on *Equal coverage*, and 3.81 vs. 3.22 on *No copula*. This means that on average more lexical items are entered into the parse chart with the expanded lexicon than with the packed lexicon.

Table 2: Comparison of competence

		Expanded lexicon			'Packed' lexicon		
Data	items	lexical	analyses	coverage (%)	lexical	analyses	coverage (%)
All items	8,271	4.20	23.69	61.4	3.41	20.58	61.7
Equal coverage	3,443	3.73	14.39	100.0	3.20	14.39	100.0
No copula	1,902	3.81	13.66	100.0	3.22	13.66	100.0

Table 3 shows how the use of packed argument frames affects the performance of the parser, compared to the use of the expanded lexicon. When applied to the *Equal coverage* set, the number of tasks is reduced by 10.7%, and the use of space is reduced with 13.3%. Similarly, the number of tasks is reduced by 12.8%, and the use of space is reduced with 13.2%, when applied to *No copula*. The numbers showing reductions for *All data* are less reliable, since the set includes sentences where the two versions of the grammar produce different numbers of analyses.

Table 3: Comparison of performance

		Expanded lexicon		'Packed' lexicon		Reduction (%)	
Data	items	tasks	space	tasks	space	tasks	space
All items	8,271	3,265	208,324	2,480	139,430	24.0	33.1
Equal coverage	3,443	2,022	102,964	1,805	69,540	10.7	13.3
No copula	1,902	1,341	59,080	1,170	51,294	12.8	13.2

One possible objection to this test would be that a grammar without the packing of argument structure information could be implemented in a different way, that would make parsing more efficient. However, this comparison is only done for testing the impact of the packing of argument structure information in a grammar that is implemented similar to Norsyg.

6 Conclusion

I have compared two versions of a lexicon for a grammar implementation. One, where valence alternations are accounted for by means of multiple lexical entries, and one, where valence alternations are accounted for by means of packed type constraints. It has been demonstrated that the packed version of the lexicon introduces fewer lexical items in the parse chart. The use of the packed version of the lexicon also leads to a reduction of tasks performed and space used by the parser.

References

Bender, E. M., Flickinger, D., and Oepen, S. (2002). The grammar matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In J. Carroll, N. Oostdijk, and R. Sutcliffe, editors, *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, pages 8–14, Taipei, Taiwan.

Borer, H. (2005a). *Structuring Sense. An Exo-Skeletal Triology. Volume I. In Name Only*. Oxford University Press.

Borer, H. (2005b). *Structuring Sense. An Exo-Skeletal Triology. Volume II. The Normal Course of Events*. Oxford University Press.

Bresnan, J. (2001). *Lexical-Functional Syntax*. Blackwell Publishers.

Copestake, A. (2002). *Implementing Typed Feature Structure Grammars*. CSLI publications.

Flickinger, D. P. (2000). On building a more efficient grammar by exploiting types. *Natural Language Engineering*, **6**(1), 15–28.

Oepen, S. and Carroll, J. (2000). Ambiguity packing in constraint-based parsing. Practical results. In *Proceedings of the 1st Conference of the North American Chapter of the ACL*, pages 162–169, Seattle, WA.

Oepen, S. and Flickinger, D. (1998). Towards systematic grammar profiling. test suite technology ten years after. *Journal of Computer Speech and Language: Special Issue on Evaluation*, **12 (4)**, 441–437.

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

Sag, I. A., Wasow, T., and Bender, E. M. (2003). *Syntactic Theory: A Formal Introduction*. CSLI Publications, Stanford, 2 edition.

Siegel, M. and Bender, E. M. (2002). Efficient deep processing of Japanese. In *COLING:02*, pages 1–8, Taipei, Taiwan.

Steedman, M. (2000). *The Syntactic Process*. Cambridge, MA and London: The MIT Press.

Åfarli, T. A. (2007). Do verbs have argument structure? In E. Reuland, T. Bhattacharya, and G. Spathas, editors, *Argument Structure*, pages 1–16. Amsterdam: John Benjamins.