Metagrammar Engineering: Towards systematic exploration of implemented grammars

Antske Fokkens
Department of Computational Linguistics, Saarland University &
German Research Center for Artificial Intelligence (DFKI) Project Office Berlin
Alt-Moabit 91c, 10559 Berlin, Germany
afokkens@coli.uni-saarland.de

Abstract

When designing grammars of natural language, typically, more than one formal analysis can account for a given phenomenon. Moreover, because analyses interact, the choices made by the engineer influence the possibilities available in further grammar development. The order in which phenomena are treated may therefore have a major impact on the resulting grammar. This paper proposes to tackle this problem by using metagrammar development as a methodology for grammar engineering. I argue that metagrammar engineering as an approach facilitates the systematic exploration of grammars through comparison of competing analyses. The idea is illustrated through a comparative study of auxiliary structures in HPSG-based grammars for German and Dutch. Auxiliaries form a central phenomenon of German and Dutch and are likely to influence many components of the grammar. This study shows that a special auxiliary+verb construction significantly improves efficiency compared to the standard argument-composition analysis for both parsing and generation.

1 Introduction

One of the challenges in designing grammars of natural language is that, typically, more than one formal analysis can account for a given phenomenon. The criteria for choosing between competing analyses are fairly clear (observational adequacy, analytical clarity, efficiency), but given that analyses of different phenomena interact, actually evaluating analyses on those criteria in a systematic manner is far from straightforward. The standard methodology involves either picking one analysis, and seeing how it goes, then backing out if it does not work out, or laboriously adapting a grammar to two versions supporting different analyses (Bender, 2010). The former approach is not in any way systematic, increasing the risk that the grammar is far from optimal in terms of efficiency. The latter approach potentially causes the grammar engineer an amount of work that will not scale for considering many different phenomena.

This paper proposes a more systematic and tractable alternative to grammar development: metagrammar engineering. I use "metagrammar" as a generic term to refer to a system that can generate implemented grammars. The key idea is that the grammar engineer adds alternative plausable analyses for linguistic phenomena to a metagrammar. This metagrammar can generate all possible combinations of these analyses automatically, creating different versions of a grammar that cover the same phenomena. The engineer can test directly how competing analyses for different phenomena interact, and determine which combinations are possible (after minor adaptations) and which analyses are incompatible.

The idea of metagrammar engineering is illustrated here through a case study of word order and auxiliaries in Germanic languages, which forms the second goal of this paper. Auxiliaries form a central phenomenon of German and Dutch and are likely to influence many components of the grammar. The results show that the analysis of auxiliary+verb structures presented in Bender (2010) significantly improves efficiency of the grammar compared to the standard argument-composition analysis within the range of phenomena studied. Because future research is needed to determine whether the auxiliary+verb alternative can interact properly with additional phenomena and still lead to more efficient results than argument-composition, it is particularly useful to have a grammar generator that can automatically create grammars with either of the two analyses.

The remainder of this paper starts with the case study. Section 2 provides a description of the context of the study. The relevant linguistic properties and alternative analyses are described in Sections 3 and 4. After evaluating and discussing the case study's results, I return to the general approach of metagrammar engineering. Section 6 presents related work on metagrammars. It is followed by a conclusion and discussion on using metagrammars as a methodology for grammar engineering.

2 A metagrammar for Germanic Languages

2.1 The LinGO Grammar Matrix

The LinGO Grammar Matrix (Bender et al., 2002; Bender et al., 2010) provides the main context for the experiments described in this paper. To begin with, its further development plays a significant role for the motivation of the present study. More importantly, the Germanic metagrammar is implemented as a special branch of the LinGO Grammar Matrix and uses a significant amount of its code.

The Grammar Matrix customization system allows users to derive a starter grammar for a particular language from a common multi-lingual resource by specifying linguistic properties through a web-based questionnaire. The grammars are intended for parsing and generation with the LKB (Copestake, 2002) using Minimal Recursion Semantics (Copestake et al., 2005, MRS) as parsing output and generation input. After the starter grammar has been created, its development continues independently: engineers can thus make modifications to their grammar without affecting the multi-lingual resource.

Internally, the customization system works as follows: The web-based questionnaire registers linguistic properties in a file called "choices" (henceforth choices file). The customization system takes this choices file as input to create grammar fragments, using so-called "libraries" that contain implementations of cross-linguistically variable phenomena. Depending on the definitions provided in the choices file, different analyses are retrieved from the customization system's libraries. The language specific implementations inherit from a core grammar which handles basic phrase types, semantic compositionality and general infrastructure, such as feature geometry (Bender et al., 2002).

The present study is part of a larger effort to improve the customization library for auxiliary structures in free word order and verb second languages. It examines whether Bender's observations concerning an improved analysis for auxiliaries in Wambaya (Bender, 2010) also hold for Germanic languages. A more elaborate study of German and Dutch (including both Flemish and (Northern) Dutch, which have slightly different word order constraints) is informative, because these languages are well-described and known to have distinctly challenging word order behavior.

2.2 Germanic branch

In order to create grammars for Germanic languages, a specialized branch of the Grammar Matrix customization system was developed. This Germanic grammars generator uses the Grammar Matrix's facilities to generate types in type description language (tdl). At present, the generator uses the Grammar Matrix analyses for agreement and case marking as well as basics from its morphotactics, coordination and lexicon implementations.

In the first stage, the word order library and auxiliary implementation were extended to cover two alternative analyses for Germanic word order (see Section 4). The coordination library was adapted to ensure correct interactions with the new word order analyses and agreement. The morphotactics library was extended to cover Dutch and Flemish interactions between word order and morphology. Finally, the lexicon and verbal case pattern implementations were extended to cover ditransitive verbs.

Both versions of word order analyses can be tweaked to include or exclude a rarely occurring variant of partial VP fronting (see Section 4.3) resulting in four distinct grammars for each of the

Vorfeld	**LB**	Mittelfeld	**RB**	Nachfeld
Der Mann	**hat**	den Jungen	**gesehen**	nach der Party
The man.nom	*has*	*the boy.acc*	*seen*	*after the party*
Der Mann	**hat**	den Jungen nach der Party	**gesehen**	
Den Jungen	**hat**	der Mann	**gesehen**	nach der Party
Nach der Party	**hat**	der Mann den Jungen	**gesehen**	
Den Jungen **gesehen**	**hat**	der Mann nach der Party		
Gesehen	**hat**	der Mann den Jungen nach der Party		
The man saw the boy after the party				

Table 1: Basic structure of German word order (not exhaustive)

languages under investigation. These 12 grammars cover Dutch, Flemish and German main clauses with up to three core arguments.[1]

3 Germanic word order

3.1 German word order

Topological fields (Erdmann, 1886; Drach, 1937) form the easiest way to describe German word order. The sentence structure for declarative main clauses, consists of five topological fields: Vorfeld (“pre-field”), Left Bracket (LB), Mittelfeld (“middle field”), Right Bracket (RB) and the Nachfeld (“after field”). A subset of permissible alternations in German are provided in Table 1. The last two sentences present an example of partial VP fronting.

The fields are defined with regard to verbal forms, which are placed in the Left and Right Brackets. Each topological field has word order restrictions of its own. The Vorfeld must contain exactly one constituent in an affirmative main clause. The Left Bracket contains the finite verb and no other elements. Other verbal forms (if not fronted to the Vorfeld) must be placed in the Right Bracket. Most non-verbal elements are placed in the Mittelfeld. When main verbs are placed in the Vorfeld, their object(s) may stay in the Mittelfeld. This kind of partial VP fronting is illustrated by the last example in Table 1. The Nachfeld typically contains subordinate clauses and sometimes adverbial phrases.

In German, the respective order between the verbs in the Right Bracket is head-final, i.e. auxiliaries follow their complements. The only exception is the auxiliary flip: under certain conditions in subordinate clauses, the finite verb precedes all other verbal forms.

3.2 Dutch word order

Dutch word order reveals the same topological fields as German. There are two main differences between the languages where word order is concerned. First, whereas the order of arguments in the German Mittelfeld allows some flexibility depending on information structure, Dutch argument order is fixed, except for the possibility of placing any argument in the Vorfeld. A related aspect is that Dutch is less flexible as to what partial VPs can be placed in the Vorfeld.

The second difference is the word order in the Right Bracket. The order of auxiliaries and their complements is less rigid in Dutch and typically auxiliary-complement, the inverse of German order. Most Dutch auxiliaries can occur in both orders, but this may be restricted according to their verb form. Four groups of auxiliary verbs can be distinguished that have different syntactic restrictions.

1. Verbs selecting for participles which may appear on either side of their complement (e.g. *hebben* (“have”), *zijn* (“be”)).

2. Verbs selecting for participles which prefer to follow their complement and must do so if they are in participle form themselves (e.g. *blijven* (“remain”), *krijgen* (“get”)).

3. Modals selecting for infinitives which prefer to precede their complement and must do so if they appear in infinitive form themselves.

[1]The grammar generation system also creates Danish grammars. Danish results are not presented, because the language does not pose the challenges explained in Section 4.

VF	LB	MF	RB
De man	zou	haar	kunnen hebben gezien
the man	*would*	*her.acc*	*can have seen*
De man	zou	haar	gezien kunnen hebben
%De man	zou	haar	kunnen gezien hebben
The man should have been able to see her			

Table 2: Variations of Dutch auxiliary order

4. Verbs selecting for "to infinitives" which must precede their complement.

While there is some variation among speakers, the generalizations above are robust. The permitted variations assuming a verb of the 3rd and 1st category in the right bracket are presented in Table 2.[2]

The variant %*De man zou haar kunnen gezien hebben* is typical of speakers from Belgium (Haeseryn, 1997); speakers from the Netherlands tend to regard such structures as ungrammatical. Our system can both generate a Flemish grammar accepting all of the above and a (Northern) Dutch grammar, rejecting the third variant.

4 Alternative auxiliary approaches

This section presents the alternative analyses for auxiliary-verb structures in Germanic languages compared in this study. For reasons of space, I limit my description to an explanation of the differences and relevance of the compared analyses.[3]

4.1 Argument-composition

The standard analysis for German and Dutch auxiliaries in HPSG is a so-called "argument-composition" analysis (Hinrichs and Nakazawa, 1994), which I will explain through the following Dutch example:[4]

(1)	Ik	zou	het	boek	willen	lezen.
	I	would	the	book	want	read.
	"I would like to read the book."					

In the sentence above, the auxiliary *willen* "want" separates the verb *lezen* "read" from its object *het boek* "the book". A parser respecting surface order can thus not combine *lezen* and *het boek* before combining *willen* and *lezen*.

The argument-composition analysis was introduced to make sure that *het boek* can be picked up as the object of the embedded verb *lezen*. The subcategorization of an auxiliary under this analysis is presented in Figure 1. The subject of the auxiliary is identical to the subject of the auxiliary's complement. Its complement list consists of the concatenation of the verbal complement and any complement this verbal complement may select for. In the sentence above, *willen* will add the subject and the object of *lezen* to its own subcatorization lists.[5] This standard solution for auxiliary-verb structures is (with minor differences) also what is provided by the Matrix customization system.

$$\left[\text{VAL}\ \left[\begin{array}{ll}\text{SUBJ} & \boxed{1}\\ \text{COMPS} & \left\langle \left[\begin{array}{ll}\text{HEAD} & \textit{verb}\\ \text{VAL} & \left[\begin{array}{ll}\text{SUBJ} & \boxed{1}\\ \text{COMPS} & \boxed{2}\end{array}\right]\end{array}\right], \boxed{2} \right\rangle\end{array}\right]\right]$$

Figure 1: Standard Auxiliary Subcategorization

Argument-composition can capture the grammatical behavior of auxiliaries in German and Dutch. However, grammaticality and coverage is not all that matters for grammars of natural language. Efficiency remains an important factor, and argument-composition has some undesirable properties on this level. The problem lies in the fact that lexical entries of auxiliaries have underspecified elements on their subcategorization lists. With the current chart parsing and chart generation algorithms (Carroll and Oepen, 2005), an auxiliary in a language with flexible word order will speculatively add edges to the chart for potential analyses with the adjacent constituent as subject or complement. Because the length of the lists are underspecified as well, it can continue wrongly combining with all elements in the string. In the worse case scenario, the number of edges created by an auxiliary grows exponentially in the number of words and constituents in the string. The efficiency problem is even worse for generation: while the parser is restricted by the surface order of

[2]Note that the same orders as in the Right Brackets may also occur in the Vorfeld (with or without the object).

[3]Details of the implementations can be found by using the metagrammar, which can be found on my homepage.

[4]Hinrichs and Nakazawa (1994) present an analysis for the German auxiliary flip. The relevant observations are the same.

[5]In the semantic representation, both arguments will be directly related to the main verb exclusively.

$$(i)\begin{bmatrix} \text{VAL} \begin{bmatrix} \text{SUBJ} & \langle\rangle \\ \text{COMPS} & \left\langle\begin{bmatrix}\text{HEAD} & \textit{verb}\end{bmatrix}\right\rangle \end{bmatrix} \end{bmatrix}$$

$$(ii)\begin{bmatrix} \text{VAL} \begin{bmatrix} \text{SUBJ} & \boxed{1} \\ \text{COMPS} & \boxed{2} \end{bmatrix} \\ \text{HEAD-DTR}|\text{VAL}|\ \text{COMPS}\ \boxed{3} \\ \text{NON-HEAD-DTR}\ \boxed{3} \begin{bmatrix} \text{VAL} \begin{bmatrix} \text{SUBJ} & \boxed{1} \\ \text{COMPS} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 2: Auxiliary lexical type (*i*) and Auxiliary+verb construction (*ii*) under alternative analysis

the string, the generator will attempt to combine all lexical items suggested by the input semantics, as well as lexical items with empty semantics, in random order.

4.2 Aux+verb construction

Bender (Bender, 2010)[6] presents an alternative approach to auxiliary-verb structures for the Australian language Wambaya. The analysis introduces auxiliaries that only subcategorize for one verbal complement, not raising any of the complement's arguments or its subject. Auxiliaries combine with their complement using a special auxiliary+verb rule. Figure 2 presents this alternative solution. In principle, the new analysis uses the same technique as argument composition. The difference is that the auxiliary now starts out with only one element in its subcategorization lists and can only combine with potential verbal complements that are appropriately constrained. The structure that combines the auxiliary with its complement places the remaining elements on the complement's SUBJ and COMPS lists on the respective lists of the newly formed phrase, as can be seen in Figure 2 (*ii*). The constraints on raised arguments are known when the construction applies. The efficiency problem sketched above is thus avoided.

4.3 A small wrinkle: partial VP fronting

In its basic form, the auxiliary+verb structure cannot handle partial VP fronting where the main verb is placed in first position leaving one or more verbal forms in the verbal cluster, as illustrated in (2) for Dutch:

(2)	Gezien	zou	de	man	haar	kunnen	hebben.
	Seen	should	the	man	her	can	have

“The man should have been able to **see** her.”

The problem is that *hebben* “have” cannot combine with *gezien* “seen”, because they are separated by the head of the clause. Because the verb *hebben* cannot combine with its complement, it cannot raise its complement's arguments either: the auxiliary+verb analysis only permits raising when auxiliary and complement combine.

This shortcoming is no reason to immediately dismiss the proposal. Structures such as (2) are extremely rare. The difference in coverage of a parser that can and a parser that cannot handle such structures is likely to be tiny, if present at all, nor is it vital for a sentence generator to be able to produce them. However, a correct grammar should be able to analyze and produce all grammatical structures.

I implemented an additional version of the auxiliary+verb construction using two rather complex rules that capture examples such as (2). Because the structure in (2) also presented difficulties for the argument-composition analysis in Dutch, I tested both of the analyses with and without the inclusion of these structures. In the ideal case, the full coverage version will remain efficient enough as the grammar grows. But if this turns out not to be the case, the decision can be made to exclude the additional rule from the grammar or to use it as a robustness rule that is only called when regular rules fail. Given the metagrammar engineering approach, it will be straightforward to decide at a later point to exclude the special rule, if corpus studies reveal this is favourable.

5 Grammars and evaluation

5.1 Experimental set-up

As described above, the Germanic metagrammar is a branch of the customization system. As such, it takes a choices file as input to create a grammar. The basic choices files for Dutch and German were created through the LinGO Grammar Matrix web inter-

[6]Bender credits the key idea behind this analysis to Dan Flickinger (Bender, 2010).

	Complete Set		Reduced Set		
	Positive	Total	Positive	Total	Av.
	s	s	s	s	w/s
Du	177	14654	138	14591	6.61
Fl	195	14654	156	14606	6.61
Ge	116	6926	84	6914	6.65

Table 3: Number of test examples (s) used in evaluation and average words per sentence (w/s)

face.[7] The choices files defined artificial grammars with a dummy vocabulary. The system can produce real fragments of the languages, but strings representing syntactic properties through dummy vocabulary were used to give better control over ambiguity facilitating the evaluation of coverage and overgeneration of the grammars. The grammars have a lexicon of 9-10 unambiguous dummy words.

The created choices files were extended offline to define those properties that the Germanic metagrammar captures, but are not incorporated in the Matrix customization system. This included word order of the auxiliary and complement, fixed or free argument order, influence of inflection on word order, a more elaborate case hierarchy, ditransitive verbs, and the choice of auxiliary/verb analysis. Four choices files with different combinations of analyses were created for each language, resulting in 12 choices files in total.

A basic test suite was developed that covers intransitive, transitive and ditransitive main clauses with up to three auxiliaries. The German set was based on a description provided by Kathol (2000), Dutch and Flemish were based on Haeseryn (1997). For each verb and auxiliary combination, all permissible word orders were defined based on descriptive resources. In order to make sure the grammars do not reveal unexpected forms of overgeneration, all possible ungrammatical orders were automatically generated. Table 3 provides the sizes of the test suites. Each language has both a complete set for the 6 grammars that provide full coverage, and a reduced set for the 6 grammars that can not handle split verbal clusters (see Section 4.3 for the motivation to test grammars that do not have full coverage).

[7]http://www.delph-in.net/matrix/customize/

Each grammar was created using the metagrammar, ensuring that all components except the competing analyses were held constant among compared grammars. The [incr tsdb()] competence and performance profiling environment (Oepen, 2001) was used in combination with the LKB to evaluate parsing performance of the individual grammars on the test suites. For each grammar, the number of required parsing tasks, memory (space) and CPU time per sentence, as well as the number of passive edges created during an average parse were compared. Performance on language generation was evaluated using the LKB.

5.2 Parsing results

Table 4 presents the results from the parsing experiment. Note that all directly compared grammars have the same empirical coverage (100% coverage and 0% overgeneration on the phenomena included in the test suites). The comparison therefore addresses the effect on efficiency of the alternative analyses. Three tests per grammar were carried out: one on positive data, one on negative data and one on the complete dataset. Results were similar for all three sets, with slightly larger differences in efficiency for negative examples. For reasons of space, only the results on positive examples are presented, which are more relevant for most applications involving parsing. The results show that the auxiliary+verb (aux+v) leads to a more efficient grammar according to all measures used. There is an average reduction of 73.2% in performed tasks, 56.3% in produced passive edges and 32.9% in memory when parsing grammatical examples using the auxiliary+verb structure compared to argument-composition. CPU-time per sentence also improved significantly, but, due to the short average sentence length (5-10 words) the value is too small for exact comparison with [incr tsdb()].

5.3 Sentence generation evaluation

The complete coverage versions of Dutch and German were used to create the exhaustive set of sentences with an intransitive, transitive and ditransitive verb combined with none, one or two auxiliaries but rapidly loses ground when one or more auxiliaries[8]

[8]All auxiliaries in the grammars contribute an ep.

Average Performed Tasks				
	Compl. Cov. Gram.		No Split Cl. Gram.	
	arg-comp	**aux+v**	**arg-comp**	**aux+v**
Du	524	149	480	134
Fl	529	150	483	137
Ge	684	148	486	136
Average Created Edges				
	Compl. Cov. Gram.		No Split Cl. Gram.	
	arg-comp	**aux+v**	**arg-comp**	**aux+v**
Du	58	25	52	25
Fl	58	26	52	25
Ge	67	23	52	24
Average Memory Use (kb)				
	Compl. Cov. Gram.		No Split Cl. Gram.	
	arg-comp	**aux+v**	**arg-comp**	**aux+v**
Du	9691	6692	8944	6455
Fl	9716	6717	8989	6504
Ge	10289	5675	8315	5468
Average CPU Time (s)				
	Compl. Cov. Gram.		No Split Cl. Gram.	
	arg-comp	**aux+v**	**arg-comp**	**aux+v**
Du	0.04	0.02	0.03	0.01
Fl	0.04	0.02	0.03	0.01
Ge	0.06	0.01	0.04	0.01

Table 4: Parsing results positive examples

	Required edges					
Du	No Aux		1 Aux		2 Aux	
	arg-c	**aux+v**	**arg-c**	**aux+v**	**arg-c**	**aux+v**
iv	54	57	221	99	792	248
tv	124	141	1311	211	7455	500
dv	212	230	14968	378	–	910
Ge	No Aux		1 Aux		2 Aux	
	arg-c	**aux+v**	**arg-c**	**aux+v**	**arg-c**	**aux+v**
iv	54	57	295	84	1082	165
tv	130	142	4001	212	18473	422
dv	306	351	–	608	–	1379

Table 5: Performance on Sentence Generation

from a total of 18 MRSs. The input MRSs were obtained by parsing a sentence with canonical word order. Both versions provide the same set of sentences as output, confirming their identical empirical coverage. Table 5 presents the number of edges required by the generator to produce the full set of generated sentences from a given MRS. The cells with no number represent conditions under which the LKB generator reaches the maximum limit of edges, set at 40,000, without completing its exhaustive search.

The grammar using argument-composition is slightly more efficient when there are no auxiliaries, are added, in particular when sentence length increases: For ditransitive verbs (dv), the Dutch argument-composition grammar maxes out the 40,000 edge limit with two auxiliaries, whereas the auxiliary+verb grammar creates 910 edges, a manageable number. Due to the more liberal order of arguments, results are even worse for German: the argument-composition grammar reaches its limit with the first auxiliary for ditransitive verbs. These results indicate that the auxiliary+verb analysis is strongly preferable where natural language generation is concerned.

5.4 In summary

The results of the experiment presented above show that avoiding underspecified subcategorization lists, as found in the standard argument-composition analysis, significantly increases the efficiency of the grammar for both parsing and generation. On average, they show a reduction of 73.2% in performed tasks, 56.3% in produced passive edges and 32.9% in memory for parsing. In generation experiments, results are even more impressive: the reduction of edges for German sentences with one auxiliary and a ditransitve verb is at least 98.5%. These results show that the auxiliary+verb alternative should be considered seriously as an alternative to the HPSG standard analysis of argument-composition, though further investigation in a larger context is needed before final conclusions can be drawn.

Future work will focus on increasing the coverage of the grammars, as well as the number of alternative options explored. In particular, both approaches for auxiliaries should be compared using alternative analyses for verb-second word order found in other HPSG-based grammars, such as the GG (Müller and Kasper, 2000; Crysmann, 2005), Grammix (Müller, 2009; Müller, 2008) and Cheetah (Cramer and Zhang, 2009) for German, and Alpino (Bouma et al., 2001) for Dutch. These grammars may use approaches that somewhat reduce the problem of argument-composition, leading to less significant differences between the auxiliary+verb and argument-composition analyses. On the other hand, planned extensions that cover modification and subordinate clauses will increase local ambiguities. The advantage of the auxiliary+verb analysis is likely to become more important as a result.

In addition to providing a clearer picture of auxiliary structures, these extensions will also lead to a better insight into efforts involved in using grammar generation to explore alternative versions of a grammar over time. In particular, it should provide an indication of the feasibility of maintaining a higher number of competing analyses as the grammar grows. After providing background on related metagrammar projects and their goals, I will elaborate on the importance of systematic exploration of grammars in the discussion.

6 Related work

Metagrammars (or grammar generators) have been established in the field for over a decade. This section provides an overview of the goals and set-up of some of the most notable projects.

The MetaGrammar project (Candito, 1998; de la Clergerie, 2005; Kinyon et al., 2006) started as an effort to encode syntactic knowledge in an abstract class hierarchy. The hierarchy can contain cross-linguistically invariable properties and syntactic properties that hold across frameworks (Kinyon et al., 2006). The factorized descriptions of MetaGrammar support Tree-Adjoining Grammars (Joshi et al., 1975, TAG) as well as Lexical Functional Grammars (Bresnan, 2001, LFG). The eXtensible MetaGrammar (Crabbé, 2005, XMG) defines its MetaGrammar as classes that are part of a multiple inheritance hierarchy. Kinyon et al. (Kinyon et al., 2006) use XMG to perform a cross-linguistic comparison of verb-second structures. Their study focuses on code-sharing between the languages, but does not address the problem of competing analyses investigated in this paper.

The GF Resource Grammar Library (Ranta, 2009) is a multi-lingual linguistic resource that contains a set of syntactic analyses implemented in GF (Grammatical Framework). The purpose of the library is to allow engineers working on NLP applications to write simple grammar rules that can call more complex syntactic implementations from the grammar library. The grammar library is written by researchers with linguistic expertise. It makes extensive use of code sharing: general categories and constructions that are used by all languages are implemented in a core syntax grammar. Each language[9] has its own lexicon and morphology, as well as a set of language specific syntactic structures. Code sharing also takes place between the subset of languages explored, in particular by means of common modules for Romance languages and for Scandanavian languages.

PAWS creates PC-PATR (McConnel, 1995) grammars based on field linguists' input. The main purpose of PAWS lies in descriptive grammar writing and "computer-assisted related language adaptation", where the grammar is used to map words from a text in a source language to a target language. PAWS differs from the other projects discussed here, because grammar engineering or syntactic research are not the main focus of the project.

The LinGO Grammar Matrix, described in Section 2.1, is most closely related to the work presented in this paper. Like the other projects reviewed here, the Grammar Matrix does not offer alternative analyses for the same phenomenon. Moreover, starter grammars created by the Grammar Matrix are developed manually and individually after their creation. The approach taken in this paper differs from the original goal of the Grammar Matrix in that it continues the development of new grammars within the system, introducing a novel application for metagrammars. By using a metagrammar to store alternative analyses, grammars can be explored systematically over time. As such, the paper introduces a novel methodology for grammar engineering. The discussion and conclusion will elaborate on the advantages of the approach.

7 Discussion and conclusion

7.1 The challenge of choosing the right analysis

As mentioned in the introduction, most phenomena in natural languages can be accounted for by more than one formal analysis. An engineer may implement alternative solutions and test the impact on the grammar concerning interaction with other phenomena (Bierwisch, 1963; Müller, 1999; Bender, 2008; Bender et al., 2011) and efficiency to decide between analyses.

[9]Ranta (Ranta, 2009) reports that GF is developed for fourteen languages, and more are under development.

However, it is not feasible to carry out comparative tests by manually creating different versions of a grammar every time a decision about an implementation is made. Moreover, even if such a study were carried out at each stage, only the interaction with the current state of the grammar would be tested. This has two undesirable consequences. First, options may be rejected that would have worked perfectly well if different decisions had been made in the past. Second, because each decision is only based on the current state of the grammar, the resulting grammar is partially (or even largely) a product of the order in which phenomena are treated.[10]

For grammar engineers with practical applications in mind, this is undesirable because the resulting grammar may end up far from optimal. For grammar writers that use engineering to find valid linguistic analyses, the problem is even more serious: if there is a truth in a declarative grammar, surely, this should not depend on the order in which phenomena are treated.

7.2 Metagrammar engineering

This paper proposes to systematically explore analyses throughout the development of a grammar by writing a metagrammar (or grammar generator), rather than directly implementing the grammar. A metagrammar can contain several different analyses for the same phenomenon. After adding a new phenomenon to the metagrammar, the engineer can automatically generate versions of the grammar containing different combinations of previous analyses. As a result, the engineer can not only systematically explore how alternative analyses interact with the current grammar, but also continue to explore interactions with phenomena added in the future. Especially for alternative approaches to basic properties of the language, such as the auxiliary-verb structures examined in this study, parallel analyses may prevent the cumbersome scenario of changing a deeply embedded property of a large grammar.

An additional advantage is that the engineer can use the methodology to make different versions of the grammar depending on its intended application. For instance, it is possible to develop a highly restricted version for grammar checking that provides detailed feedback on detected errors (Bender et al., 2004), next to a version with fewer constraints to parse open text.

As far as finding optimal solutions is concerned, it must be noted that this approach does not guarantee a perfect result, partially because there is no guarantee the grammar engineer will think of the perfect solution for each phenomenon, but mainly because it is not maintainable to implement all possible alternatives for each phenomenon and make them interact correctly with all other variations in the grammar. The grammar engineer still needs to decide which alternatives are the most promising and therefore the most important to implement and maintain. The resulting grammar therefore partially remains a result of the order in which phenomena are implemented. Nevertheless, the grammar engineer can keep and try out solutions in parallel for a longer time, increasing the possibility of exploring more alternative versions of the grammar. These additional investigations allow for better informed decisions to stop exploring certain analyses. In addition, by breaking up analyses into possible alternatives, chances are that the resulting metagrammar will be more modular than a directly written grammar would have been, which facilitates exploring alternatives further.

In sum, even though metagrammar engineering does not completely solve the challenge of complete explorations of a grammar's possibilities, it does facilitate this process so that finding optimal solutions becomes more likely, leading to better supported choices among alternatives and a more scientific approach to grammar development.

Acknowledgments.

The work described in this paper has been supported by the project TAKE (Technologies for Advanced Knowledge Extraction), funded under contract 01IW08003 by the German Federal Ministry of Education and Research. Emily M. Bender, Laurie Poulson, Christoph Zwirello, Bart Cramer, Kim Gerdes and three anonymous reviewers provided valuable feedback that resulted in significant improvement of the paper. Naturally, all remaining errors are my own.

[10]It is, of course, possible to go back and change old analyses based on new evidence. In practice, the large effort involved will only be undertaken if the advantages are apparent beforehand.

References

Emily M. Bender, Dan Flickinger, and Stephan Oepen. 2002. The grammar matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In John Carroll, Nelleke Oostdijk, and Richard Sutcliffe, editors, *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, pages 8–14, Taipei, Taiwan.

Emily M. Bender, Dan Flickinger, Stephan Oepen, Annemarie Walsh, and Tim Baldwin. 2004. Arboretum: Using a precision grammar for grammar checking in call. In *Proceedings of the InSTIL/ICAL Symposium: NLP and Speech Technologies in Advance Language Learning Systems*, Venice, Italy.

Emily M. Bender, Scott Drellishak, Antske Fokkens, Laurie Poulson, and Safiyyah Saleem. 2010. Grammar customization. *Research on Language & Computation*, 8(1):23–72.

Emily M. Bender, Dan Flickinger, and Stephan Oepen. 2011. Grammar engineering and linguistic hypothesis testing. In Emily M. Bender and Jennifer E. Arnold, editors, *Language from a Cognitive Perspective: Grammar, Usage and Processing*, pages 5–29. Stanford: CSLI Publications, Palo Alto, USA.

Emily M. Bender. 2008. Grammar engineering for linguistic hypothesis testing. In Nicholas Gaylord, Alexis Palmer, and Elias Ponvert, editors, *Proceedings of the Texas Linguistics Society X Conference: Computational Linguistics for Less-Studied Languages*, pages 16–36, Stanford. CSLI Publications.

Emily M. Bender. 2010. Reweaving a grammar for Wambaya: A case study in grammar engineering for linguistic hypothesis testing. *Linguistic Issues in Language Technology*, 3(3):1–34.

Manfred Bierwisch. 1963. *Grammatik des deutschen Verbs*, volume II of *Studia Grammatica*. Akademie Verlag.

Gosse Bouma, Gertjan van Noord, and Robert Malouf. 2001. Alpino: Wide coverage computational analysis of Dutch. In *Computational Linguistics in the Netherlands CLIN 2000*.

Joan Bresnan. 2001. *Lexical Functional Syntax*. Blackwell Publishers, Oxford.

Marie-Helene Candito. 1998. Building parallel LTAG for French and Italian. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pages 211–217, Montreal, Quebec, Canada. Association for Computational Linguistics.

John Carroll and Stephan Oepen. 2005. High efficiency realization for a wide-coverage unification grammar. In *IJCNLP*, Jeju Island. Springer-Verlag LNCS.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan Sag. 2005. Minimal recursion semantics. an introduction. *Journal of Research on Language and Computation*, 3(2–3):281 – 332.

Ann Copestake. 2002. *Implementing Typed Feature Structure Grammars*. CSLI Publications, Stanford, CA.

Benoît Crabbé. 2005. *Représentation modulaire et paramétrable de grammaires électroniques lexicalisées*. Ph.D. thesis, Université de Paris 7.

Bart Cramer and Yi Zhang. 2009. Constructon of a German HPSG grammar from a detailed treebank. In *Proceedings of the ACL 2009 Grammar Engineering across Frameworks workshop*, pages 37–45, Singapore, Singapore.

Berthold Crysmann. 2005. Relative clause extraposition in German: An efficient and portable implementation. *Research on Language and Computation*, 3(1):61–82.

Éric Villemonte de la Clergerie. 2005. From metagrammars to factorized TAG/TIG parsers. In *Proceedings of IWPT'05*, pages 190–191.

Erich Drach. 1937. *Grundgedanken der Deutschen Satzlehre*. Diesterweg, Frankfurt am Main, Germany.

Oskar Erdmann. 1886. *Grundzüge der deutschen Syntax nach ihrer geschichtlichen Entwicklung dargestellt. Erste Abteilung*. Verlag der Cotta'schen Buchhandlung, Stuttgart, Germany.

Walter Haeseryn. 1997. De gebruikswaarde van de ans voor tekstschrijvers, taaltrainers en taaladviseurs. *Tekst[blad]*, 3.

Erhard Hinrichs and Tsuneko Nakazawa. 1994. Linearizing auxs in German verbal complexes. In John Nerbonne, Klaus Netter, and Carl Pollard, editors, *German in HPSG*. CSLI, Stanford, USA.

Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163.

Andreas Kathol. 2000. *Linear Syntax*. Oxford Press.

Alexandra Kinyon, Owen Rambow, Tatjana Scheffler, SinWon Yoon, and Aravind K. Joshi. 2006. The metagrammar goes multilingual: A cross-linguistic look at the V2-phenomenon. In *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms*, pages 17–24, Sydney, Australia. Association for Computational Linguistics.

Stephen McConnel. 1995. PC-PATR reference manual.

Stefan Müller and Walter Kasper. 2000. HPSG analysis for German. In Wolfgang Wahlster, editor, *Verbmobil: Foundations of Speech-to-Speech translation*, pages 238 – 253, Berlin, Germany. Springer.

Stefan Müller. 1999. *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche*. Max Niemeyer Verlag, Tübingen.

Stefan Müller. 2008. Depictive secondary predicates in german and english. In Christoph Schroeder, Gerd Hentschel, and Winfried Boeder, editors, *Secondary Predicates in Eastern European Languages and Beyond*, number 16 in Studia Slavica Oldenburgensia, pages 255–273, Oldenburg, Germany. BIS-Verlag.

Stefan Müller. 2009. On predication. In Stefan Müller, editor, *Proceedings of the 16th International Conference on Head-Driven Phrase Structure Grammar*, Stanford, USA. CSLI Publications.

Stephan Oepen. 2001. [incr tsdb()] — competence and performance laboratory. Technical report, DFKI, Saarbrücken, Germany.

Aarne Ranta. 2009. The GF resource grammar library. *Linguistic Issues in Language Technology*, 2(2).