CLIMB grammars: Three projects using metagrammar engineering

Antske Fokkens, Tania Avgustinova, Yi Zhang

Department of Computational Linguistics, LT-Lab DFKI GmbH
Saarland University, Saarbrücken Germany
{afokkens,avgustinova}@coli.uni-saarland.de,yizhang@dfki.de

Abstract

This paper introduces the CLIMB (Comparative Libraries of Implementations with Matrix Basis) methodology and grammars. The basic idea behind CLIMB is to use code generation as a general methodology for grammar development in order to create a more systematic approach to grammar development. The particular method used in this paper is closely related to the LinGO Grammar Matrix. Like the Grammar Matrix, resulting grammars are HPSG grammars that can map bidirectionally between strings and MRS representations. The main purpose of this paper is to provide insight into the process of using CLIMB for grammar development. In addition, we describe three projects that make use of this methodology or have concrete plans to adapt CLIMB in the future: CLIMB for Germanic languages, CLIMB for Slavic languages and CLIMB to combine two grammars of Mandarin Chinese. We present the first results that indicate feasibility and development time improvements for creating a medium to large coverage precision grammar.

Keywords: Grammar engineering, methodology, code sharing

1. Introduction

This paper introduces the CLIMB (Comparitive Libraries of Implementations with Matrix Basis) methodology for grammar engineering together with two projects that use the methodology and one that has concrete plans to do so. CLIMB is based on the basic idea expressed in Fokkens (2011) to use metagrammar engineering as a methodology for implementing linguistically motivated precision grammars. The term *metagrammar* refers here to software that can generate implemented grammars.

The main goal of this paper is to provide a detailed description of the set-up and workflow of metagrammar engineering using technology from the LinGO Grammar Matrix (Bender et al., 2010). Second, it introduces three projects that use this technology, each with a different (main) purpose. Germanic CLIMB grammars (Fokkens, 2011) use CLIMB for systematic grammar development and empirical research on alternative analyses. SlaviCore (Avgustinova and Zhang, 2009) uses CLIMB mainly for sharing implementations across languages. Finally, two independently developed grammars for Mandarian Chinese, the MCG (Zhang et al., 2011) and ManGO[1] will be combined with CLIMB. We will provide a brief motivation for the approach in this introduction.

Grammar engineers are confronted with the problem that, often, more than one analysis can account for a syntactic phenomenon. Because phenomena interact, the choices made at early stages of grammar development can have a major impact on the possibilities that remain for phenomena that are treated in the future. When using metagrammar engineering as a methodology, all additions to a grammar are added to syntactic libraries. These libraries contain syntactic analyses as well as implementations to make sure analyses interact properly. The grammar developer can store alternative analyses for the same phenomenon in such libraries and keep on creating and testing the impact of these alternatives as the grammar grows.

To our knowledge, Fokkens (2011) is the first approach that suggests complete grammar development through code generation with the purpose of comparing analyses.[2] However, the grammars implemented and tested in Fokkens (2011) are small. The successor of this implementation, Germanic CLIMB, covers the development set of Cheetah (Cramer and Zhang, 2009), a 105 positive example set[3] that was used to develop phenomena occurring in the TiGerTreebank (Brants et al., 2002). This shows that the technique can be applied to develop large-scale grammars.

Investigating the practical side of using metagrammar engineering as a methodology for grammar development is one of the main questions the projects presented in this paper address. As part of this question, we also investigate the possibility to adapt grammars that have been implemented manually for an extended period of time to CLIMB. We measure the time invested in the structure of this process precisely to get insight into the investment needed to adapt the approach at a later stage of development. The current time investment of 36 hours covers approximately 84% of the SlaviCore and 70% of the Russian Resource Grammar (Avgustinova and Zhang, 2010, RRG). As will be discussed below, further refinement is needed to fully benefit from the methodology, but it seems that the time needed to adapt this grammar to the new methodology will be around two to three weeks of fulltime work for one person.

The rest of this paper is structured as follows. We start with a description of the LinGO Grammar Matrix, explain who CLIMB extends its use and compare the Grammar Matrix standard approach to CLIMB. The main advantages of using CLIMB are explained, before we describe the individual projects. After presenting related work, we discuss advantages and shortcomings of the approach and provide an overview of future work to improve it.

[1] `moin.delph-in.net/MandarinGrammarOnline`

[2] Metagrammars and similar techniques have been used in grammar development for over a decade. We will discuss these approaches in Section 7.

[3] One example of the original 106 set was rated ungrammatical by a native speaker.

```
nom-acc-transitive-verb-lex :=
       transitive-verb-lex &
  [ ARG-ST
     < [ LOCAL.CAT.HEAD noun &
                    [ CASE nom ] ],
       [ LOCAL.CAT.HEAD noun &
                    [ CASE acc ] ] > ].
```

Figure 1: Example of type definition defined in TDL.

2. The CLIMB Methodology

CLIMB emerged from the LinGO Grammar Matrix (Bender et al., 2010) and uses parts of its software. We will therefore start the description of CLIMB with an overview of the LinGO Grammar Matrix. After presenting how CLIMB uses the Grammar Matrix architecture, we will discuss the difference between the two approaches.

2.1. LinGO Grammar Matrix

The Grammar Matrix is a multi-lingual resource that creates starter grammars based on specified linguistic properties provided through a web-based questionnaire.[4] The Grammar Matrix and its derived grammars are situated within DELPH-IN[5], an international research consortium that develops open-source software for NLP applications using linguistically motivated grammars. In line with DELPH-IN, the Grammar Matrix uses the HPSG (Pollard and Sag, 1994) framework and can map bidirectionally between surface strings and semantic representations in the format of Minimal Recursion Semantics (Copestake et al., 2005, MRS). They can be used to parse and generate sentences with the LKB (Copestake, 2002) and to parse with PET (Callmeier, 2000).

The basic workflow of using the Grammar Matrix is to fill out the Matrix questionnaire and click on "create grammar". An archive containing a grammar based on the filled-out questionnaire can be downloaded. The grammar is specified in DELPH-IN's reference formalism TDL (Krieger and Schäfer, 1994; Copestake, 2002) and can be extended manually. Figure 1 presents an example of a type definition in TDL. It defines a type called *nom-acc-transitive-verb-lex* which receives basic properties for a transitive verb from its supertype *transitive-verb-lex*. Further constraints indicate that its first argument must be a noun in nominative case, and its second a noun in accusative case.

The source code of the Grammar Matrix can be obtained under the MIT-license.[6] Internally, the system is organized into "libraries" that by and large correspond to the individual pages of the web-interface. The customization system that generates language specific implementations takes a file called "choices" (henceforth *choices file*) as input containing options and definitions provided by the user through the questionnaire. The options in the choices file can parameterize the analyses that are combined as well as specific properties on individual items. The functions in the libraries can create types and add properties to types introduced elsewhere in the customization system.

The parts developing lexicon and morphotactics are particularly interesting from a developer's point of view. We will highlight some properties that are relevant to the CLIMB method. First, the customization system revises user defined type hierarchies based on underspecified forms in the lexicon or in morphotactics of the language. Second, the morphotactics library allows the user to define a complex system with interacting stems and morphemes. Finally, it contains code to help define linguistic properties that often vary on lexical items or inflection. Currently, these aspects include syntactic head features (e.g. case and verbal forms) as well as semantic features (e.g. index, tense). These features can be marked on the lexical item, on its subject or object for verbs and on the specified noun for determiners. These three properties permit users to create large chunks of the grammar using relatively simple definitions. When used cleverly, they can lead to a significant speed up in the grammar development process.

2.2. The CLIMB Workflow

CLIMB can be used to create any grammar written in TDL. However, all three projects described in this paper are Matrix-based grammars, i.e. they make use of the Matrix core, as well as implementations provided by the standard customization system of the Grammar Matrix. In our description below, we will assume this is the typical case for using CLIMB.

When using CLIMB as a methodology, the grammar engineer creates a Matrix starter grammar through the web-based questionnaire and obtains the Grammar Matrix source code. The starter grammar contains the choices file described above, which defines phenomena and properties that are generated using the customization system's *libraries*. Choices are directly linked to implementations in the libraries and their parameters. Engineers can start by extending the intial grammar manually as before. Each time an implementation is completed or improved, the grammar engineer adds it to a linguistic library and associates it with a particular definition in the choices file. After updating the libraries and choices file, a grammar including the extensions can be created for further development.

More advanced users can extend the libraries and choices files directly. This can typically be done for new lexical categories and morphotactic properties. As described above, these libraries contain many general functions that can combine complex properties. Simple changes to the source code of CLIMB and a set of definitions in the choices file can give a major boost in grammar coverage. In this case, the CLIMB method can be faster than traditional grammar engineering.[7]

Implementing syntactic libraries requires programming skills in an other language than TDL for the grammar engineer. The Grammar Matrix customization system is written in Python, which is therefore also used in CLIMB. Figures

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

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

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

[7] German adjectives (including inflection depending on determiner, number, gender and case) were implemented within 6 hours.

```
section=word-order                  section=lexicon
word-order=v2                       noun1_name=1st-pron-nom
has-dets=yes                          noun1_feat1_name=person
noun-det-order=det-noun               noun1_feat1_value=1st
has-aux=no                            noun1_feat2_name=case
                                      noun1_feat2_value=nom
section=case                        noun1_det=obl
case-marking=nom-acc
nom-acc-nom-case-name=nominative    verb1_name=trans
nom-acc-acc-case-name =accusative   verb1_valence=nom-acc
```

Figure 2: Small extract of a choices file

```
  wo = ch.get('word-order')

  if wo == 'v2':
    mylang.add('head-initial-head-nexus := head-initial & \
                [ SYNSEM.LOCAL.CAT.MC na & #mc, \
                  HEAD-DTR.SYNSEM.LOCAL.CAT.MC #mc ].')
    mylang.add('head-final-head-nexus := head-final & \
                [ SYNSEM.LOCAL.CAT.MC bool, \
                  HEAD-DTR.SYNSEM.LOCAL.CAT.MC na ].')

#rules shared among free and v2

  if wo == 'free' or wo == 'v2':
    mylang.add('head-subj-phrase := decl-head-subj-phrase & head-initial-head-nexus.')
    mylang.add('subj-head-phrase := decl-head-subj-phrase & head-final-head-nexus.')
```

Figure 3: Sample code from word order library: implementations triggered by word-order=v2 in choices

2 and 3 provide samples of a choices file and Python code found in a syntactic library, respectively. The code presented in Figure 3 will be called because of the definition *word-order=v2* in the choices file. Functions that must be written to use CLIMB are typically if-then-else conditions, as in Figure 3, or iterators. The most complex implementations of the system are the parts that interpret and combine type definitions, but these are provided by the Grammar Matrix customization system. We therefore believe that basic programming skills suffice to adapt the methodology.

2.3. Methodological comparisons

The description of CLIMB has shown that it is tightly linked to the Grammar Matrix. However, both the philosophy behind the projects as well as the practice of applying them highly differ. The Grammar Matrix aims at lowering the hurdle starting a new grammar. It is therefore essential that the system be easy to use and cover a wide typological range. CLIMB, on the other hand, is a methodology that particularly pays off on long term projects. Users of CLIMB can be expected to be experts for whom it pays off to invest in the architecture of their system and techniques that may facilitate modularity. We will elaborate on the consequences of these differences below.

The standard approach when using the Grammar Matrix is to create a grammar through the web interface and extend this grammar manually. Most of the linguistic properties defined through the questionnaire do not reveal a direct link to the customization system or even to HPSG theory. The only exception is the possibility to create hierarchies of supertypes and subtypes, which points to HPSG's formalism: typed feature structures. The Grammar Matrix basic approach thus emphasizes the central control of "hidden" logic behind the scenes of the customization process. The user can explore provided analyses in TDL, but no direct insight into how the customization system came to the resulting grammar can be obtained while merely using the web-interface. CLIMB takes a radically different approach by placing the customization source code under control of the grammar engineer, so that different levels of parameterization can be achieved in individual grammar development projects. Users are encouraged to explore the possibilities of the customization system and expand it for their language specific needs.

Another difference between the Grammar Matrix and CLIMB is the wide typological variations aspired by the Grammar Matrix. Sharing implementations across languages is one of the main purposes of the Grammar Matrix. Even though CLIMB takes advantage of this possibility, the method was originally introduced to examine different ways of achieving the same goal. One could say that the Grammar Matrix explores an analysis that can be used in different manifestations of a phenomenon, whereas CLIMB explores a specific manifestation of a phenomenon that may use different analyses for its implementation. In the end, the Grammar Matrix and CLIMB complement each other. The former makes grammar engineering accessable to a wider public and provides a starting point for new grammars. The latter can be used to improve grammar development on long

term projects. The next section will give a more elaborate explanation of this property and other advantages of using CLIMB.

3. Advantages of CLIMB

The main advantage of the CLIMB approach is the increased flexibility as compared to traditional grammar engineering. This advantage is seen in the Germanic CLIMB grammars where libraries store alternative analyses that capture the same phenomenon. The extensions developed for one version can be used to automatically extend other versions of the grammar (though minor adaptations may be needed due to interactions between phenomena). The Chinese CLIMB project will investigate whether the approach works if alternative analyses come from independently developed grammars. CLIMB's flexibility also allows researchers to choose which phenomena to include or exclude in the grammar. This can be an advantage when an old analysis needs to be revised: a new grammar can be generated without the older analysis which does include analyses added at a later stage.

The second advantage is the speed-up through support by Grammar Matrix tools. In particular, when grammars for related languages are developed in parallel. Grammar Matrix techniques for implementing phenomena that vary cross-linguistically can be used to integrate extensions made for one language into the grammar for another. This advantage is mainly explored in the Slavic and Germanic grammar projects. However, speed-up can also occur within an individual language. Its exact impact is one of the research questions addressed by the Germanic CLIMB project, which compares development time of the CLIMB grammars to that of the German grammar Cheetah (Cramer and Zhang, 2009).

Finally, the approach can be used to evaluate the Grammar Matrix. The Grammar Matrix provides basic implementations for a broad range of languages. The CLIMB grammars expand the Grammar Matrix approach in another direction, covering more phenomena and more language specific depth. This may provide insights into futher extensions of the Grammar Matrix. In addition, CLIMB developement has already led to increased feedback to the Matrix. The original idea behind the Grammar Matrix (Bender et al., 2002) was that derived grammars would in turn provide feedback on the cross-linguistic applicability of the Matrix. In practice, changes made in the core for an individual language are seldom reported. For customized analyses, the feedback loop is even harder to maintain: it is not straight-forward to see which language specific changes are extensions and which are corrections from the original analysis. The Germanic system reveals exactly where revisions to previous analyses were introduced and where additions for new analyses were made. Moreover, it is kept in sync with the original Grammar Matrix so that changes to the core must be discussed with other Matrix developers. The Germanic CLIMB grammars have already lead to revisions for adjectives, modification, wh-questions, long-distance dependencies, relative clauses and adpositions: the biggest revision of existing analyses since the Matrix was launched (Bender, p.c.).

4. Germanic CLIMB

The Germanic CLIMB resource focuses on German. Variations are introduced for Dutch, and to a lesser extend for Danish. The development of this resource has several goals. First, we want to examine whether the CLIMB approach can be used in a large scale grammar development project. Second, we examine the impact of alternative analyses for verb-second languages and auxiliary structures on coverage, overgeneration and efficiency of the grammar. Third, in future work, we would like to gain insight into the impact of using analyses from the Grammar Matrix on the rest of the grammar. Finally, we use the CLIMB resource to adapt analyses based on the grammars' intended use. The first and third goal and its current achievements will be elaborated below.

4.1. CLIMB in large scale projects

Part of our investigation addresses the impact of using CLIMB on development speed. On the one hand, CLIMB requires additional implementations of syntactic libraries which may be time consuming. Moreover, the decision to keep alternative analyses may slow development down due to additional adaptations to assure correct interaction with other phenomena. The same holds for the efforts to also cover certain phenomena in Dutch and Danish. On the other hand, CLIMB's facilities for code generation can be time saving. The increased flexibility created by linking each part of an implementation to an explicit phenomenon or analysis may facilitate changing the grammar, which may provide an additional gain in development time. Grammar development using CLIMB could in principle either be slower or faster than implementing TDL code directly.

In order to get insight into the impact on development time using CLIMB, we compare the development time of a CLIMB grammar for German to the development time of Cheetah (Cramer, 2011). Note that a comparison between two development processes can never give conclusive evidence on the benefit of a given approach: no two grammar engineers are alike. Experience has a major impact on development speed, so measuring the difference in development of grammars developed by the same engineer is not an option either. However, conditions between our grammar and Cheetah are similar enough to give an indication of the impact of CLIMB: Both grammars were grammars of German written by a graduate student (having comparable experience) with Dutch as a native language. Both had access to the same resources: implementations of Cheetah were not looked at during the development of Germanic CLIMB, literature on HPSG analyses of German was consulted by both engineers.

The development time for a German grammar that gets (at least) the same coverage as Cheetah on Cheetah's development set was approximately six months,[8] compared to one person year reported in Cramer (2011). If all influencing

[8] About one month of this time was spend in revising the architecture of the first metagrammar implementation. We expect this is mainly due to the fact that Germanic CLIMB is the first CLIMB grammar and could possibly be avoided in future projects.

factors had been stable, this would have meant that CLIMB reduced development time by half. Even considering the fact that the engineers were different, we believe that it is unlikely that this could lead to such big a difference. When taking into account that time spent on alternative languages as well keeping Germanic CLIMB in sync with the Grammar Matrix are not obligatory in using CLIMB, this result provides a strong indication that CLIMB in general has a positive influence on development time.

4.2. Future work: The impact of the Grammar Matrix

An important question related to the Grammar Matrix is what impact it has on the resulting grammar to use the Matrix's basic implementations. This research question was one of the reasons to develop a grammar for German. Two comparative resources using the same formalism exist for this language, namely GG (Müller and Kasper, 2000; Crysmann, 2003) and Cheetah. The two differ in the sense that GG has stayed reasonably close to standard analyses for German in HPSG. Though coverage and efficiency have certainly been taken into account, linguistic precision has always been a major aim in GG. Cheetah, on the other hand, aimed to get high coverage on newspaper data fast, preferring to exclude phenomena that are infrequent and lead to inefficiencies when covered. Germanic CLIMB can be placed in between these two: it contains both implementations that reflect HPSG theory, as well as variations that primarily aim at efficiency.

Table 1 presents results for running GG, Cheetah and the best performing settings of Germanic CLIMB on Cheetah's (Coverage Original) and CLIMB's extended (Coverage Extended) development set. CLIMB's development set is an extension from Cheetah's 105 positive development examples, containing 504 positive and 632 negative examples. Given that CLIMB was developed on this set, results on coverage and overgeneration do not evaluate the quality of the grammars. The experiment was used for initial observations. Cheetah covered, as expected, the basic phenomena in the set, but not all linguistic variations or combinations of phenomena. GG, on the other hand, did not cover all phenomena (e.g. comparatives), but variations and combinations were generally handled correctly for those phenomena it did cover. Overgeneration was mainly due to flexible morphology for both GG and Cheetah. CPU time was measured on 319 positive examples parsed by all grammars.

	Coverage Original	Coverage Extended	Over-generation	CPU (s per sentence)
Cheetah	88.6%	79.0%	17.9%	0.46
GG	86.7%	80.4%	8.7%	1.56
CLIMB	94.3%	99.8%	1.1%	0.60

Table 1: Results on CLIMB development data

To evaluate the performance of individual grammars on independent data, we plan to import Cheetah's learned lexicon into Germanic CLIMB and GG and compare their performance on the TiGer Treebank. The main focus will be comparing Germanic CLIMB and Cheetah. It is hard to predict a priori which system will do better: Cheetah covers less linguistic variation, but may cover more long sentences due to its efficiency. Depending on the results, we will import alternative analyses from Cheetah and GG into CLIMB and measure the impact of such changes.

5. SlaviCore

SlaviCore is a resource that contains basic analyses known to occur cross-linguistically within the Slavic language family. The main idea behind this project was inspired by the Grammar Matrix and some of the issues it addresses are in line with those addressed by CLIMB. Like the Grammar Matrix, it aims to share knowledge and implementations for grammar development to help new projects get started. Like CLIMB, the resource primarily addresses advanced users with a large scale project of grammar development in mind.

A multilingual resource restricted to one language family allows us to tackle some of the challenges faced by the Grammar Matrix's ambition to cover a wide typological range of languages. A Slavic specific core can cover more phenomena and model them in greater detail than a core grammar that aims to be useful for all natural languages. Another aspect of multilingual grammar engineering the Slavic grammars hopes to improve is the feedback loop between developers of the core grammar and engineers working on individual languages. Current implementation projects in SlaviCore are mainly run by three researchers, Avgustinova and Zhang working on Russian and Osenova working on Bulgarian, who collaborate to establish the core grammar. The resulting SlaviCore in itself can provide important feedback to the Grammar Matrix. The Slavic language family namely does not only reveal many common properties, but also exhibits a wide range of typological variation.

5.1. SlaviCore strategy

The Slavic grammar uses a strategy designed to be compatible with the current Grammar Matrix program: the customization system is used to quickly build small grammars for individual languages; shared analyses are put into a SlaviCore; when the next language is added, the SlaviCore helps to more efficiently build the new grammar, simultaneously receiving a cross-Slavic validation. As related languages share a much wider range of linguistic information than typically assumed in standard multilingual grammar architectures, a distinctive feature of this approach to Slavic grammatical resources is that grammar engineering for each individual language takes place in a common Slavic setting. This in particular means that if, for example, two possibilities are conceivable of how to model a particular phenomenon observed in a certain Slavic language, the option that would potentially be consistent with what is found in the other grammars will be strongly preferred.

The ultimate goal is a SlaviCore module in the format of a phenomenon based library designed for maximum reusability, lifting out the elements that can and should be common across individual resource grammars. The the Russian Resource Grammar (Avgustinova and Zhang, 2010, RRG) is

the main focus of the project, both in terms of the end product and as a large-scale experimental set-up for hypothesis testing. The integration of an existing Bulgarian grammar (Osenova, 2010, BURGER) into the Slavic matrix architecture forms the second central activety of the project. Finally, a grammar prototype for Polish provides the possibility to test the approach on a new Slavic language.

5.2. SlaviCore and CLIMB

The CLIMB approach is primarily intended to facilitate the integration of the independently developed BURGER, as well as to extend the static SlaviCore with more dynamic methods for sharing analyses across languages.

As a first step, we have started to reproduce the RRG through CLIMB. This can be tackled in two ways: either we include the Slavic core as a stable file of definitions, just like the basic types of the Matrix core, or we add analyses found in the SlaviCore to CLIMB libraries and generate the core just like the language individual files. The SlaviCore in its current form is based on extensive research on formal analyses for Slavic phenomena (Avgustinova, 2007). This means we can expect the core to remain relatively stable. This stability can, however, also be achieved by a stable set of 'family choices' at the top of choices files that can serve as a basis for new Slavic grammars. We decided to include general Slavic analyses in CLIMB libraries, because this has the additional advantage that it facilities research on the true applicability across Slavic languages for implementations in the SlaviCore. We will elaborate on this advantage in the next subsection, which describes the current state of Slavic CLIMB and observations made in the creation process.

5.3. Current stage and observations

Currently, 36 work hours have been invested in getting analyses of the RRG integrated in Slavic CLIMB, resulting in 84% coverage of definitions found in the SlaviCore and 70% of the RRG as a whole. Based on the present experience and phenomena left to integrate into Slavic CLIMB, we estimate needed time for completing the process to be approximately two work weeks in total. The currently produced fragment contains most of the syntactic rules and complete hierarchy of verbs and nouns defined in the RRG. In fact, the fragment already has full (treebanked) coverage over the 280 examples provided in RRG's basic testset.

Based on these initial efforts, we can report some observations on SlaviCore and using CLIMB. First, most effort went into extending and changing the choices file. Several of these extensions did not require any additions to the CLIMB generation code. This indicates that the Matrix customization system could have provided more initial implementations than the original developers of the RRG made use of. It may therefore be worthwhile for Grammar Matrix users to download the customization system and experiment with extended choices files, even if they do not intend to adapt the CLIMB approach.

Our second observation concern the implementations in SlaviCore themselves: many, as well as a large part of Russian specific implementations, consist of large type hierarchies based on theoretical research reported in (among other sources) Avgustinova (2007). Our present implentation for Slavic CLIMB reproduces these exact hierarchies, including the original name in most cases. As it turns out, the advantages of code generation are relatively limited for creating these implementations. Extensions to code generation software may have been small, but declaring type definitions in a choices file is as labour intensive as directly writing them in TDL. Only minor advantages could be found in generating cross-classifications automatically. It is, however, in this area were some interesting research questions can be addressed.

If we relax the effort to exactly reproduce SlaviCore and allow its exact form to interact with language specific definitions, we can create empirical tests of Avgustinova (2007)'s model. In particular, the customization system's capability to introduce underspecified supertypes based on surface forms defined for the language can lead to such insights. It would, for instance, be interesting to see whether case hierarchies created for a set of languages with a flexible customization system correspond to the hierarchy proposed by Avgustinova (2007) based on theoretical research. We plan to address these questions in future research.

6. Mandarin Chinese

Two HPSG-based grammars for Mandarin Chinese, MCG (Zhang et al., 2011) and ManGO, have been under independent development at different institutes for over a year. Both grammars started with the Grammar Matrix, and achieved moderate coverage over basic phenomena, including numeral-classifier phrases, light-verb constructions, relative clauses with DE, various aspect markers, topic-comment constructions, inter alios.

Although the two grammars have significant overlap on covered phenomena, MCG aims for broader coverage of more frequent phenomena, while ManGO specializes in challenging but less frequent phenomena. Aiming to improve the long-term maintainability, developers have decided to merge the two grammars. The grammar merging task is non-trivial, and involves detailed comparison of different linguistic treatments. Following the CLIMB approach helps us isolate implementations for each individual phenomenon, making the comparison easier and more focused.

Preliminary analysis shows that the V-not-V construction (for interrogative sentences in Mandarin Chinese) implementation in ManGO can be straight-forwardly carried over to the MCG. The treatment of BA(causative) and BEI(passive) constructions in MCG involves systematic changes on the valency list, and can be easily applied to ManGO. Meanwhile, the analysis for the numeral-classifier phrases in MCG is more consistent than ManGO (in various cases of ellipsis). The difference in the treatment for DE relative clauses is another interesting case. The MCG further distinguishes three subtypes of relative clauses with differences on semantic argument binding. ManGO, on the other hand, delivers only one type analysis for DE relative clauses with underspecified semantics, due to the lack of syntactic cue for further disambiguation in most of the cases. Both treatments are interesting, and with CLIMB we

can leave this as a toggle option to produce either coarse-grained or fine-grained analyses.

7. Related work

Several grammar engineering projects make use of code sharing or metagrammars. This section provides a brief overview of the goals and set-up of some notable projects.
The MetaGrammar project (Candito, 1998; Villemonte de la Clergerie, 2005) was originally set up as a hierarchy that encodes syntactic knowledge. 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. Within TAG grammars, the metagrammar plays a role that is comparable to the upper part of the type hierarchy in HPSG: it allows to share parts of structures between individual elementary trees.
The GF Resource Grammar Library (Ranta, 2009) is a linguistic resource consisting of crosslinguistically applicable syntactic analyses implemented in GF (Grammatical Framework). Engineers can write basic grammar rules that inherit complex syntactic structures (written by linguistic experts) from the GF library. Code sharing is used extensively: a core syntax grammar contains general categories and constructions that are used in individual grammars. 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.
The main difference between CLIMB and the projects mentioned above is that CLIMB promotes complete grammar development through a metagrammar, including language specific properties. In MetaGrammar, GF or the typical methodology applied with the Grammar Matrix described in Section 2.1., users build up language specific analysis using implementation from the metagrammar or libraries. All projects described above, including CLIMB, aim at improving the architecture of implemented grammars, development speed and consistency across grammars through code sharing. However, CLIMB introduces, to our knowledge, new applications of metagrammars and syntactic libraries in comparing individual analyses in a consistent manner over time and in using the technique to combine individually developed analyses.

8. Conclusion and Future Work

8.1. Conclusion

This paper introduced CLIMB, a methodology for grammar engineering based on an idea originally introduced by Fokkens (2011). We have provided a detailed description of how the methodology works and how it relates to the Grammar Matrix. Three projects that use CLIMB or have direct plans to do so have been introduced. The first project presented was Germanic CLIMB that investigates the impact of CLIMB on a large scale projects as well as the influence of individual analyses on overall performance of the grammar. Second, Slavic CLIMB uses the methodology to integrate shared analyses from two individually developed grammars for Russian and Bulgarian. Third, a project planning to use CLIMB to combine analyses for individually developed grammars of Mandarin Chinese.
The results from the first two projects aim to provide an indication in the impact of CLIMB on development time. Germanic CLIMB was developed twice as fast as the comparable resource Cheetah. Efforts into adapting the RRG to CLIMB indicate that the transfer will cost approximately two work weeks. We believe that this effort is worthwhile given the expected speed in future development. Based on these experience, we believe that it is both feasible and advantageous to adapt CLIMB, if needed expertise is present.

8.2. Discussion and future work

Time spent on developing generation software and testing alternative analyses has emerged as criticism on CLIMB. Regarding this concern, it should be noted that, even though we believe that it makes sense to stick to CLIMB once one is set-up, this is not strictly necessary. If working on syntactic libraries should become too cumbersome at a certain time, it is always possible to create the latest or best version of the grammar and continue with manual development only. Concerning time spent on testing alternative versions; they can be run in parallel, so this does not necessarily cost more time than testing only one grammar. It is also possible to focus on the most promising version of the grammar, and run tests on the others in the background while continuing development. Changes needed to keep other versions performing correctly can be added at a later stage.
It should also be noted that, even though all existing CLIMB projects create Grammar Matrix-based grammars, CLIMB can also be used without using the linguistic properties of the Matrix. The basic functions of the customization system can combine any properties written in well-formed TDL. The grammar engineer can thus choose to use CLIMB and write a grammar that is not related to the LinGO Grammar Matrix by removing linguistic analyses from the source code. This also means that, technically, any grammar written in TDL can be reproduced by CLIMB. It would afterall be possible (though not very useful) to add all types individually to the grammar through a generation definition.
Because CLIMB can, in principle, be used to write any grammar in TDL, impact on coverage and performance of the grammars was not the main focus of our evaluation. Coverage and performance depend, in principal, on the analyses of the grammar and not on the methodology. Nevertheless, we have started comparative research between two grammars for German to Germanic CLIMB in this paper. This research may provide some indication into the impact of using the Grammar Matrix and CLIMB. The outcome of this research remains a mere indication: initial analyses provided by the Grammar Matrix are likely to influence future development of the grammar and CLIMB may influence the choice of analyses. It is, however, impossible to determine how much is influenced by the Grammar Matrix or CLIMB and how much is the result of insights and decisions of the engineer writing the grammar.

Finally, required programming skills can be a hindrance for a grammar engineer to start using CLIMB. We have argued above that basic skills in a scripting language suffice, but nevertheless, it remains a drawback of the approach. We plan to address this in future work by developping supporting software for CLIMB. For instance, Fokkens et al. (2011) present an algorithm that helps engineers to identify parts of the grammar that do not influence the performance. This algorithm is integrated into the TDL processing implementations of CLIMB. We plan to extend these implementations so that they can compare an extended grammar to the grammar originally produced by its choices file. Through this comparison, it will provide feedback to the engineer concerning the additions that need to be made to CLIMB.

9. Acknowledgements

We would like to thank Emily Bender and anonymous reviewers for their feedback, which helped to improve this paper. The third author thanks the Deependence project funded by the BMBF (01IW1103) for its support of the work. All errors are our own.

10. References

Tania Avgustinova and Yi Zhang. 2009. Parallel grammar engineering for Slavic languages. In *Proceedings of GEAF*, Singapore.

Tania Avgustinova and Yi Zhang. 2010. Conversion of a Russian dependency treebank into HPSG derivations. In *Proceedings of TLT'9*.

Tania Avgustinova. 2007. *Language Family Oriented Perspective in Multilingual Grammar Design*. Linguistik International: Band 17. Peter Lang - Eurpopäischer Verlag der Wissenschaft, Frankfurt am Main, Germany.

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 GEE Workshop*, pages 8–14, Taipei, Taiwan.

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

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The TIGER treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories*, Sozopol, Bulgaria.

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

Ulrich Callmeier. 2000. PET - a platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering*, 6(1).

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.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A. Sag. 2005. Minimal recursion semantics: An introduction. *Research on Language & Computation*, 3(4):281–332.

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

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

Bart Cramer and Yi Zhang. 2009. Constructon of a German HPSG grammar from a detailed treebank. In *Proceedings of GEAF*, pages 37–45, Singapore.

Bart Cramer. 2011. *Improving the feasibility of precision-oriented HPSG parsing*. Ph.D. thesis, Saarland University.

Berhold Crysmann. 2003. On the efficient implementation of german verb placement in hpsg. In *Proceedings of RANLP*.

Antske Fokkens, Yi Zhang, and Emily M. Bender. 2011. Spring cleaning and grammar compression: Two techniques for detection of redundancy in HPSG grammars. In *Proceedings of the 25th PACLIC*, Singapore, Singapore.

Antske Fokkens. 2011. Metagrammar engineering: Towards systematic exploration of implemented grammars. In *Proceedings of ACL:HLT*, Portland, Oregon, USA.

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

Hans-Ulrich Krieger and Ulrich Schäfer. 1994. TDL - A Type Description Language for constraint-based grammars. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 893–899, Kyoto, Japan.

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.

Petya Osenova. 2010. BUlgarian Resource Grammar Efcient and Realistic (BURGER).

Carl Pollard and Ivan Sag. 1994. *Head-Driven Phrase Structure Grammar*. UCP, Chicago, USA.

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

Éric Villemonte de la Clergerie. 2005. From metagrammars to factorized TAG/TIG parsers. In *Proceedings of IWPT05*, pages 190–191.

Yi Zhang, Rui Wang, and Yu Chen. 2011. Engineering a deep hpsg for mandarin chinese. In *Proceedings of ALR*, Chiang Mai, Thailand.