Enhancing Empirical Research for Linguistically Motivated Precision Grammars
* Thesis Summary *

Antske Fokkens

1 Introduction

Grammars, both those built by linguists and the natural objects they are intended to model, are complex objects (Bierwisch, 1963; Bender, 2008; Bender et al., 2011).[1] Syntactic phenomena and therefore their analyses interact. Furthermore, often more than one formal analysis can account for a given phenomenon. These two properties make it notoriously difficult to make strong assertions about analyses for grammars of natural language. Grammar engineering improves this situation over pen-and-paper syntax allowing researchers to build and manipulate models of much greater complexity. In fact, it is extremely difficult to verify if all interactions between phenomena in a language model behave correctly without implementing them and checking them with a computer (Bierwisch, 1963; Müller, 1999).

Implemented grammars thus have the advantage that their correctness can be verified, but the exact influence of choices made when designing a grammar remains difficult to foresee. Furthermore, even if the grammar's behaviour can be tested empirically, it remains difficult to investigate the exact impact of individual analyses. The objective of this thesis is to enhance the possibilities of empirical research on linguistic precision grammars by proposing a new methodology that provides a more systematic approach to grammar development. I will briefly elaborate on the main idea in this introduction.

In any given grammar engineering project, analyses are implemented in some temporal order. But analyses of phenomena interact and the analytical choices already made influence the relative attractiveness of alternatives analyses for phenomena encountered at a later choice point. The order in which phenomena are added thus affects the resulting grammar (Fokkens, 2011). In this thesis, I argue that better grammars can result if grammar engineers can break free of the temporal sequence of implementation, and that metagrammar engineering is an effective way to do so.

Challenges related to deeply embedded analyses are familiar to most grammar engineers working on large scale grammars. Francis Bond (p.c.) reports that it is often hard to identify parts of the grammar which relate to obsolete analyses in the Japanese grammar Jacy (Siegel and Bender, 2002). Montserrat Marimon (p.c.) reports that there are analyses in her Spanish grammar (Marimon, 2010) for clitics and word order that need revisions, but it would be an elaborate undertaking to make these changes. Tracy King (p.c.) reports a discussion within ParGram (King et al., 2005) whether adjectives have subjects. The English LFG grammar (Riezler et al., 2002) was changed a few times, but this was so time consuming that King decided to call the last change final.

The work presented in this thesis is, to my knowledge, the first to make a fundamental problem in grammar engineering and syntactic research explicit, namely, the challenge of dealing with inconclusive evidence for analyses and the unforeseeable interaction with phenomena added in the future. It is also the first work I am aware of that proposes a solution that addresses this problem. The key idea is that alternative plausible analyses for linguistic phenomena are added to a metagrammar, where *metagrammar* refers to a system that can generate computational grammars. This meta-

[1] Some parts of this summary have been taken from the full thesis and previously published work, notably Fokkens (2011) and Fokkens and Bender (2013).

grammar can generate all possible combinations of these analyses automatically, creating different versions of a grammar that cover the same phenomena. The grammar engineer can test directly how competing analyses for different phenomena interact, and determine which combinations are possible (after minor adaptations) and which are incompatible. Alternative analyses can be maintained while new analyses are developed, thus reducing the influence of the order in which phenomena are investigated. The interaction between alternative analyses with phenomena can be tested empirically after new analyses have been added to the grammar. As such, metagrammar engineering provides a more systematic method for grammar development that enhances empirical experiments on the grammar's behaviour.

The enhancement of empirical research by using a metagrammar forms the main contribution of this thesis. In addition, the implemented approach provides several contributions to developing HPSG grammars.[2] These contributions include increased modularity, multilingual grammar development, maintaining alternative analyses for different applications or different dialects, a phenomenon-based organisation of the grammar, facilitating grammar development by multiple engineers and the possibility of including or excluding rare phenomena. The method and related tools have been evaluated through the implementation of a large scale metagrammar for German that can generate alternative analyses. These studies have led to new (though preliminary) insights in grammars for German. The thesis furthermore includes studies that explore multilingual aspects of the approach confirming the increased modularity and possibility of sharing analyses (partially) across languages. This shows how the approach can be used to increase linguistic insight in multilingual studies. Finally, the thesis provides an extensive study of how a common language independent core has been used in largely independently developed grammars using an algorithm that can identify which parts of the grammar are actually used and which are obsolete.

I will elaborate on these contributions in the rest of this summary which is structured as follows. Section 2 describes CLIMB,[3] the metagrammar engineering approach that I developed as part of my research. This is followed by a description of the evaluation of the approach through the development of gCLIMB, CLIMB for Germanic languages in Section 3. In Section 4, I provide an overview of the multilingual studies carried out as part of this thesis. Related approaches and reflections on how the idea behind CLIMB could be adopted in other grammar engineering projects are described in Section 5. Section 6 provides concluding remarks.

2 CLIMB

This section describes the CLIMB approach for metagrammar engineering. I will first describe how CLIMB emerged from the Grammar Matrix (Bender et al., 2010). This is followed by a description of three variations of CLIMB implemented as part of the thesis.

2.1 The Grammar Matrix and CLIMB

The Grammar Matrix customisation system allows users to derive a starter grammar for a particular language from a common multilingual resource by specifying linguistic properties through a web-based questionnaire. The resulting HPSG 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 multilingual resource. Internally, the customisation system works as follows: The web-based questionnaire registers linguistic properties in a file called "choices" (henceforth *choices file*). The customisation 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 customisation system's libraries. The language specific implementations inherit from a core grammar which handles basic

[2] Head-driven Phrase Structure Grammars (Pollard and Sag, 1994)

[3] Comparative Libraries of Implementations with a (grammar) Matrix Basis (Fokkens, 2011; Fokkens et al., 2012; Fokkens and Bender, 2013)

phrase types, semantic compositionality and general infrastructure, such as feature geometry (Bender et al., 2002).[4]

Climb shares the overall setup of the Grammar Matrix customisation system of using choices files that define linguistic phenomena and Python libraries that generate HPSG grammars in TDL[5] (Copestake, 2002), the DELPH-IN[6] joint reference formalism, based on the phenomena defined in the choices file. However, the two approaches serve radically different purposes and therefore make different use of grammar customisation.

The customisation system remains a black box for its users who are (potentially novice) grammar engineers working with any language. The users provide linguistic definitions and obtain a grammar that they can alter and extend manually. The project emphasizes typological coverage. It can only add phenomena slowly, as each one must be grounded in thorough typological research. Climb, on the other hand, generalises the idea of grammar customisation to metagrammar engineering, placing the customisation source code under control of the (usually expert) grammar engineer, so that different levels of parameterisation can be achieved in individual grammar development projects. Users are encouraged to explore the possibilities of the customisation system and expand it for their language-specific needs. Using grammar customisation in the development of language-specific resources frees the grammar engineer to focus on phenomena as they manifest in the language(s) at hand, without concern for the full range of typological variation. This allows the grammar engineer to include more phenomena and more detailed analyses of them. The next subsection will describe metagrammar engineering with the original implementation of CLIMB.

2.2 Procedural CLIMB

The original version of CLIMB (procedural CLIMB) builds directly on the Grammar Matrix, by simply taking the customisation system (minus the web front-end), and allowing grammar engineers to extend it for particular languages. A CLIMB metagrammar takes a choices file as its input and produces a grammar in TDL. The choices file specifies phenomena and properties that are generated using the metagrammar's libraries, which mainly consist of procedural functions that add type definitions to the grammar based on definitions in the choices file.

The metagrammar code contains control statements which check for linguistic properties and selected analyses in the choices file. Based on these properties it triggers statements which output TDL accordingly. The control statements can verify whether analyses are compatible with other choices or whether additional constraints are required for correct interaction. Fig. 1 in the appendix gives a sample analysis illustrating how this is implemented in procedural CLIMB.

Improving the metagrammar in this approach consists of the following steps. The first steps are the same as in regular grammar engineering: the grammar writer identifies a phenomenon that (s)he would like to capture (better), creates test data to verify the analysis and designs an analysis (or multiple analyses). However, (s)he does not implement this directly in the grammar, but writes a function in the metagrammar that can create the analysis and associates it with a definition in the choices file. Then grammars with the new analysis are generated and tested on the relevant test data. Corrections are made to the metagrammar based on the outcome of the tests. In some cases, alternative implementations for the same analysis need to be added to ensure the new analysis interacts appropriately with all alternative analyses present in the metagrammar. Some combinations of analyses may turn out to be incompatible.

Procedural CLIMB can support alternative analyses in grammars in a highly flexible manner. A draw-back of this approach is that it requires implementing procedural functions in Python. Even for grammar engineers proficient in programming, it can be a nuisance to switch back and forth between declarative definitions of constraints in the grammar and procedural code invoking constraints based on choices in the metagrammar.

[4]HPSG uses typed feature structures (Carpenter, 1992) to model natural language grammars. Chapter 2 of the thesis provides an introduction to HPSG and how its formal properties relate to CLIMB.

[5]Type Description Language

[6]DELPH-IN (`http://www.delph-in.net`) is an international initiative working on HPSG grammars for NLP applications.

2.3 Declarative CLIMB

Declarative CLIMB avoids flipping back and forth between declarative and procedural programming. The definitions in declarative CLIMB are written directly in TDL, where paths in type definitions may optionally be abbreviated. A small set of additional declarative statements is used to identify where analysis-specific parts of an implementation start. The definitions in the choices file indicate which analyses in the metagrammar should not be selected for the generated grammar. Procedural Python implementations are still used to interpret the choices file and to merge type definitions from different locations in the metagrammar, but grammar engineers do not need to write these functions in order to use declarative CLIMB. Fig. 2 in the appendix illustrates an implementation in declarative CLIMB.

In addition to procedural and declarative CLIMB, which both assume that the grammar is built within CLIMB from the ground up, I developed SHORT-CLIMB.[7] This method provides support for maintaining alternative analyses for grammars that have been developed the traditional way (i.e. not using a metagrammar) so that existing large-scale grammars can also make use of the approach.

This section described the origin of CLIMB and introduced the three variations of CLIMB that were developed as part of the thesis. The original form of CLIMB, procedural CLIMB, is the most powerful of the three and was used in the evaluation research described in the next sections.

3 Evaluating CLIMB

The evaluation of CLIMB addressed two aspects can determine the methodology's merit. First, the main question, namely, whether CLIMB can be used in long-term grammar development projects was investigated through the development of gCLIMB. Second, I carried out studies using the grammars produced by gCLIMB that would be impossible or very difficult to carry out when using a traditional approach to grammar engineering.

[7]Starting High On a Roof Top CLIMB

3.1 Large-scale CLIMB grammars

Evaluating the impact of using CLIMB on long-term and large-scale grammar development is a non-trivial task. There are many aspects that influence the quality of a grammar and the speed in which it can be developed. The grammar engineer, their experience and knowledge about the language are among the biggest influential factor. It is thus not possible to carry out a controlled experiment where CLIMB and traditional grammar development are compared and all influential factors are maintained equal. Possibly a set of such experiments would provide a solid indication of its exact influence, but given that the main effect can only be observed on large grammars this is not feasible. A comparative study can nevertheless provide insight into the methodology's influence on grammar development.

I evaluated CLIMB through the development of gCLIMB and compared it to the development of Cheetah (Cramer, 2011), another grammar for German that was developed by a Dutch PhD student in the same formalism in one person year. The goal was to develop a metagrammar that can create German grammars with alternative analyses covering at least all phenomena that the Cheetah core covers in Cheetah's test set, or at least, to discover how close I could get in one person year.

The alternative analyses that were included in the grammar covered word order and auxiliaries: two phenomena that occur in almost every sentence and highly interact with each other as well as other phenomena in the language. A detailed overview of the included phenomena with basic indications of how their treatment differed from Cheetah is provided in Section 5.1.3 of the thesis. This overview gives an indication of the complexity of the analyses in gCLIMB.

After six person months, gCLIMB covered slightly more phenomena than Cheetah. The grammars were also evaluated on a portion of the TiGer corpus, where Cheetah outperformed gCLIMB due to reasons of efficiency, and the independently developed Babel corpus where gCLIMB outperformed Cheetah handling more linguistic phenomena correctly. Furthermore, gCLIMB provides more in-depth semantic analyses and, in the first weeks of its development, time was spent to include alternative analyses for Dutch and German. Overall, the evaluation indicated that the two grammars are comparable in scale and complexity.

In addition to the influential factors already mentioned above, it is clear that no definite conclusions on development speed can be drawn by comparing the development of gCLIMB and Cheetah. However, it was shown that gCLIMB covers a wide range of linguistic phenomena indicating that the approach scales and is suitable for large-scale grammar development. Furthermore, gCLIMB was developed in less than half the time of Cheetah's core grammar while partially addressing crosslinguistic variation and adapting a more challenging standard for its semantic output. This strongly indicates that if the method were to have a negative impact on development time, this stays within an acceptable range.

3.2 Enhancing empirical research

As mentioned in the previous section, gCLIMB contains alternative analyses for word order and auxiliaries. The grammar supports both the "standard" HPSG analysis for verb second order (based on Uszkoreit (1987)'s GPSG analysis) and auxiliaries (from Hinrichs and Nakazawa (1994)) for German as well as the analyses Bender developed for Wambaya (Bender, 2010). The second word order analysis for Wambaya is more basic than the analysis for German and has also been included in the Grammar Matrix customisation system (Fokkens, 2010). The alternative analysis for auxiliaries was developed for Wambaya because of advantages in efficiency.[8]

The large-scale implementation where these alternative analyses are maintained provide the first illustration of how metagrammar can enhance empirical research. Grammar engineering can be used to verify syntactic analyses (Bender et al., 2011). GCLIMB provides the means to verify the competence of the alternative analyses (i.e. can they provide the correct semantic interpretation of all well-formed expressions and do they reject ill-formed expressions?) in a larger context. Because all analyses are automatically generated from the same metagrammar, we can make sure that all other potentially influential factors remain equal (and even verify the correct interaction between various alternative accounts). This would be highly challenging, if not to say impossible, to achieve in a traditional grammar development approach: developing multiple grammars in parallel is not only time consuming, but also increases the risk that an improvement made in one grammar is forgotten in the other. If an alternative analysis is integrated in a relatively large grammar at a later stage, the original analysis has most likely already had a significant impact on the grammar.

In addition to the grammar's competence, I carried out two experiments addressing efficiency of the alternative analyses included in gCLIMB in parsing and language generation. The first experiment compared parsing results showing that the alternative analysis for Wambaya remains more efficient as the grammar grows, but that this difference can be greatly reduced by constraining the non-head-daughter in subject and complement rules in the grammar. If no such constraints are assigned, the difference in efficiency of the two analyses increases significantly as the grammars grow, just as predicted by Fokkens (2011).

Similar observations are made in the second experiment investigating natural language generation. This experiment furthermore examined how specific linguistic properties of a language interact with different analyses. Grammars with slightly different word order properties for auxiliaries and their verbal complements were created and their performance was compared. An interesting aspect of this experiment is that CLIMB truly facilitated it. The different grammars were created by changing a few lines in choices files. It would be much harder to create all these versions of a grammar by manually adapting and maintaining them. Furthermore, the metagrammar (again) ensures that properties that are not related to the changes under investigation remain stable. As such, this experiment shows that CLIMB facilitates research that would be extremely difficult to carry out without the tools provided by CLIMB.

4 Multilingual Studies

In this section, I will describe the thesis's main multilingual studies concerning CLIMB and the Grammar Matrix. I will first elaborate on multilingual approaches using CLIMB. This is followed by a de-

[8]A detailed description of these analyses can be found in Chapter 4 of the thesis, Sections 4.3 and 4.4, where Section 4.4 illustrates why the alternative analysis for Wambaya boosts efficiency.

scription of insights gained into the way the Grammar Matrix is used in various sources.

4.1 Multilingual aspects of CLIMB

Given the close relation with the Grammar Matrix, it is not surprising that CLIMB supports parallel grammar development. Multilingual grammar development with CLIMB is indeed similar to that already offered by the Grammar Matrix customisation system. The main difference is that when CLIMB is used, analyses need not be limited to the basic properties of linguistic phenomena. The resource can include detailed analyses for individual languages, because only a closed group of languages is considered in this approach. If the languages in question are well-documented, the full range of variation for a phenomenon is known. Furthermore, parallel grammar development will often be opted in case of related languages or languages that are known to share linguistic phenomena. The differences will therefore typically be less than the full typological variation found in all natural languages.

This idea is explored in the context of the project PaGES[9] (Avgustinova and Zhang, 2009). Section 6.3 of the thesis describes SlaviCLIMB, a dynamic component supporting the development of *Slavi-Core*, a static core capturing typical Slavic phenomena. The thesis outlines how a dynamic component can be used to empirically verify Avgustinova (2007)'s theoretical model for cross-Slavic grammar development combining linguistic insights of Slavicists with CLIMB's technology. SlaviCLIMB can currently generate the SlaviCore and Russian Grammar RRG described in Avgustinova and Zhang (2009) and Avgustinova and Zhang (2010).[10]

I furthermore investigated the possibilities of creating grammars for Germanic languages other than German with gCLIMB. Grammars for Dutch, Danish and Northern Frisian were created and evaluated. The earliest versions of gCLIMB contained variations for Dutch and Danish, but these were not maintained as the metagrammar was further developed to cover the Cheetah test set. Dutch was evaluated on a test set based on the original set created for German. For Danish, the test set accompanying DanGram (Müller and Ørsnes, to appear) was used.

[9]Parallel Grammar Engineering for Slavic languages

[10]See also Fokkens and Avgustinova (2013) for an explanation of SlaviCLIMB.

Neither of the experiments led to perfect coverage of the test data, but the Dutch results clearly outperformed those for Danish, with Dutch leading to coverage of 97.1%-100% and overgeneration of 0.2%-1.0% and Danish resulting in 57.6% coverage and 20.7% overgeneration. The difference in results can be explained by the fact that the Dutch test set largely includes the same phenomena as the test set used to develop gCLIMB and that German syntax is closer to Dutch syntax than to Danish. It is furthermore possible that properties of Dutch have influenced the implementations in gCLIMB, because it is my native language.

Despite the similarities between Dutch and German, however, a few fundamental revisions would be required to correct the errors in the Dutch grammars currently created with gCLIMB. The fact that Dutch word order was handled correctly in earlier versions of gCLIMB shows how important it can be to make use of CLIMB's possibility of maintaining old analyses. If I had kept the original word order analysis in parallel with the revised one, I could have "time travelled" back to grammars that could handle word order in Dutch and possibly avoided fundamental revisions in a complex grammar.[11] This observation has led to the conclusion that a multilingual resource can only support multiple languages correctly if the variations found in these languages are taken into consideration at all stages of development. On the other hand, the Dutch gCLIMB grammars handle a large part of the phenomena correctly and even the Danish grammars revealed correct behaviour for several phenomena. It is unlikely that a similar result could have been obtained by using only the Grammar Matrix customisation system and extending the grammars manually. The customisation system covers significantly less phenomena and implementing all phenomena covered by gCLIMB from scratch would probably take several weeks. Adapting a traditionally developed DELPH-IN grammar that contains these analyses such as German grammar GG (Müller and Kasper, 2000) for Dutch and Danish is a highly complex task and would most likely not lead to a similar result in such a short time.

The results on Northern Frisian, which was not

[11]See Fokkens and Bender (2013) or Chapter 1 of the thesis for the time travel reference.

considered at any time during the development of gCLIMB, confirm this outcome. The final result using gCLIMB revealed coverage of 94% and no overgeneration on a linguistically motivated test set. It was obtained in one day of work. Kilmer and Packard's developed a grammar with the same test set over several weeks as part of the *Knowledge and Engineering course for NLP*, reached 70.6% of coverage and 1.8% overgeneration on the same test set. Again, conclusions should be drawn with care. The grammar was developed by different engineers (though Packard is an expert) and Kilmer and Packard also needed to develop the test set. Nevertheless, the significant difference in time and coverage indicate that gCLIMB provided a major boost. This result and the outcome of the Dutch and Danish evaluations show that gCLIMB indeed offers an increased level of modularity that makes it easier to share its implementations across languages.

Finally, the thesis examines how the CLIMB method may be used to create alternative grammars for second language learners in Section 6.4. A prototype to learn German adjective endings has been implemented and an approach for making use of parallel grammar development to capture typical mistakes of speakers of a related language is outlined.

4.2 The Grammar Matrix

The development of gCLIMB led to the most extensive revisions of the Grammar Matrix core since its release. Several revisions were bug fixes and some removed English specific properties. Because it seemed unlikely that none of the other grammars using the Matrix core ran into similar issues, I investigated how individual languages use the Matrix core. The Grammar Matrix core provides general (mostly) language independent implementations for HPSG grammars. The first step in this investigation was to apply the spring cleaning algorithm (Fokkens et al., 2011) to all grammars. This algorithm identifies types that do not have an impact on the competence of the grammar and removes them.[12] This leads to a cleaned up version of the grammar, where types that do not have an impact on the grammar are removed, but the structure of the grammar is preserved. The impact of the Grammar Matrix was investigated by looking at changes that were made to the Grammar Matrix core and the types that were removed by spring cleaning the grammar.

The most important observation out of this investigation is that several grammar engineers made changes to the Grammar Matrix core that form a general improvement to the Grammar Matrix. The current setup of the Grammar Matrix, where grammar writers typically go their own way after using the Grammar Matrix for a jump start, provides complete flexibility to grammar writers. They can make any change that suits them without needing to explain their motivation to fellow grammar writers. The disadvantage is that there is no need for grammar writers to discuss the changes they make with fellow grammar writers and often changes remain unnoticed by developers of other grammars and of the Grammar Matrix. Valuable information on how the Grammar Matrix core works is thus lost and grammar engineers need to reinvent the wheel when it comes to dealing with shortcomings of the resource. The ParGram program (Butt et al., 2002), where grammar engineers implementing LFG grammars meet twice a year to discuss their analyses, avoids this problem. The Grammar Matrix and individual grammars could greatly benefit from such an approach, but it is currently not feasible to carry this out.

The study described in this section is, to my knowledge, the first overview of the Matrix core types that are used and changes made to the core in different Matrix-based grammars. The spring cleaning algorithm played an important role in this investigation revealing that this algorithm can also be used to support empirical investigation for linguistic precision grammars.

5 Related Work

This thesis proposed to adopt metagrammar engineering as a general methodology for grammar development, so that alternative analyses can be tested systematically at different stages of the development process. Though my work is, to my knowledge, the first to tackle this problem, it is not the first work that proposes the use of a meta-

[12]The competence of the grammar refers to its capability of providing the correct bidirectional mapping between semantic representations and surface strings.

grammar. Moreover, different grammar engineering frameworks have proposed a variety of ways to provide the other benefits for their approach that CLIMB provides for HPSG grammars, including improving maintainability and consistency, increasing modularity and supporting knowledge and code-sharing across grammars. Chapter 7 of the thesis provides an extensive overview of various grammar engineering approaches, the solutions they provide and how these solutions relate to the syntactic theory the approach is based on and its main goals. For this summary, I will restrict myself to the approaches that are closest related to CLIMB and to the general contributions of this thesis.

5.1 (X)MG

Metagrammars for TAG were developed to provide a level of linguistically motivated generalisation and thereby improve the maintainability of the grammar. They also contribute to the overall structure of the grammar and have a multilingual ambition (Candito, 1999).

The (X)MG approach clearly has a lot in common with the approach proposed in this thesis. First, they both use libraries (or dimensions) of basic implementations in combination with code generation to develop grammars. Second, they share the goal of improving maintainability, modularity and the systematic structure of the grammar. Finally, both approaches support multilingual grammar development, even though MGs and XMGs for TAG are generally monolingual (Kallmeyer, p.c.).

On the other hand, the differences between the approaches are also significant. The main purpose of MG and XMG is to allow the grammar engineer to organise the grammar in a manner that is generally done by the type hierarchy and multiple inheritance in HPSG grammars. Code generation thus plays different roles in (X)MG and CLIMB. Whereas (X)MG uses code generation to capture generalisations, CLIMB uses code generation to include alternative analyses for the same phenomenon (within a language or crosslinguistically).

Furthermore, there is little overlap between XMG's organisation in dimensions and the organisation in linguistic libraries in the Grammar Matrix or CLIMB. On the one hand, CLIMB represents basic subcategorisation and surface constraints together at several places in the metagrammar. On the other hand, XMG does not, to my knowledge, use the more fine grained architecture found in CLIMB libraries to distinguish between different morphosyntactic properties, such as case assignment or person-number-gender agreement. With CLIMB, linguists can define observations they make on the surface (e.g. ambiguous morphological markings) and automatically generate a grammar containing a corresponding type hierarchy.[13] To my knowledge, (X)MG does not provide such support to grammar writers.

5.2 Modular typed unification grammars

Sygal and Wintner (2011) present foundations of a modular construction for unification grammars using typed feature structures. This research is part of a project on mathematical and computational infrastructure for grammar engineering.

When comparing their modular typed unification grammars to CLIMB, similarities in setup and overall ideas are immediately apparent, especially for declarative CLIMB. As a matter of fact, Sygal and Wintner (2011) address the LinGO Grammar Matrix and, in particular the customisation system, in their related work as a similar approach that improves modularity. They point out that the approach is different from theirs, because only prewritten code is divided over different modules and the grammar writer has no control over the customisation system. This is exactly the point in which CLIMB differs from the Grammar Matrix. As explained in Section 2, the grammar engineers using CLIMB directly manipulate the libraries in the customisation system, and are thus fully in control of the language specific customisation system they are creating. Both approaches allow the grammar writer to provide partial definitions of a type hierarchy in separate modules which can be combined to form a full grammar. CLIMB can, in principle, fulfil all nine desiderata for modularity set by Sygal and Wintner (2011) and fulfils five of them for the same reasons (see Section 7.5.2 of the thesis for a detailed comparison).

The main difference between the two approaches probably is that improving modularity is the pri-

[13] Section 6.3 of the thesis explains how this property of CLIMB can be used to verify linguistic hypotheses.

mary goal of Sygal and Wintner (2011)'s approach. CLIMB, on the other hand, is the result of a practical solution to a theoretical problem. Modularity improves through CLIMB because it is necessary for sharing grammar components and maintaining alternative analyses at the same time (regardless of whether alternatives are meant to support syntactic research or to capture crosslinguistic differences). For Sygal and Wintner (2011), modularity is a goal in itself, with a predefined set of desiderata. This makes their approach more principled (there are no formal restrictions on CLIMB modules), but it remains to be seen whether this will be advantageous in practice.

5.3 CoreGram

CoreGram is the common core of a set of HPSG grammars developed at the Freie Universität Berlin (Müller, 2013). Despite its HPSG background, interest in linguistic precision and facilitation of sharing implementations, CoreGram is a rather different approach from CLIMB. The organisation of parts of the grammars across files that may be included or excluded from the grammar is similar to declarative CLIMB, but the absence of code generation leaves out the option of both working on the level of phenomena (which may entail partial definitions of several types) and types (mostly complete type definitions with constraints related to several phenomena) at the same time. CoreGram does not (to my knowledge) contain an equivalent of the dynamic component of CLIMB that can speed up grammar development and capture similarities across languages where individual feature values differ by generating lexical items or rules.

5.4 The CLIMB idea for other projects

The proposal made in this thesis is theory and formalism independent. It can thus in principle be adopted for each of the projects above as well as the other projects outlined in my thesis, namely GF (Ranta, 2009), ParGram (Butt et al., 2002) and PAWS[14] (Black, 2004; Black and Black, 2009). However, the amount of effort that would be involved as well as the potential interest in adopting metagrammar engineering differs from one project to another. Feasibility is directly related to the method used for sharing implementations.

As far as implementation methods and feasibility are concerned, two manners of sharing code can be distinguished: static code sharing and dynamic code sharing. ParGram, GF and CoreGram develop sets of definitions that can be used by individual grammars. They share code statically. (X)MG and PAWS use code generation in their systems, just like CLIMB. Their method is more dynamic. Even though it is in principle possible to systematically compare implementations using different versions of static code, code generation truly provides a basic methodology that facilitates and stimulates this. (X)MG would not require many changes to its architecture to allow its users to carry out systematic comparative research.

Another question is whether systematic comparison may be of interest to the projects. As far as they address the question of alternative possibilities, the aforementioned projects tend to go for restricting possibilities based on crosslinguistic research. This is specifically mentioned in Butt et al. (2002). Ranta (2009) also aims to share as much as possible between grammars. In many ways, this attitude towards alternatives makes sense. If data cannot provide clear evidence for a particular analysis, any other indications to help and choose are welcome. But in the end, it seems clear that testing more analyses integrated into a large grammar gives a more reliable indication than merely discussing alternatives or only trying things out at the moment they first come up. In principle, all approaches that are interested in optimising their grammars could therefore benefit from a methodology that can help to compare analyses systematically.

The GF Resource Library may make use of the approach for optimisation reasons. Systematic exploration can lead to more efficient grammars, which is generally interesting for application-oriented grammar engineering. Moreover, it might be possible to get more out of sharing analyses between different application grammars if code generation were introduced. For now, it seems that most of the sharing is done through the core grammar in GF.

Despite the occasional influence of practical as-

[14]Parser And Writer for Syntax

pects, both (X)MG and ParGram aim for representing theoretical assumptions from TAG and, respectively, LFG correctly. Systematic exploration would make the grammars more flexible to changes in the theory and increase the support the respective theories can get from implemented grammars. Tracy King (p.c.) confirmed that an approach like CLIMB would have been very useful for ParGram when dealing with the question whether adjectives have subjects, mentioned in the introduction.

Of all projects mentioned in this section, the proposal made in this thesis is in my opinion most relevant for CoreGram. The grammars in this project are all intended to provide correct HPSG models of language. To my knowledge, they are not application driven and will therefore not make practical decisions to improve grammar performance if this is not in line with HPSG theory. Even though the CoreGram developers are probably not interested in experiments comparing computational efficiency, they seem to care more than the other approaches about finding correct linguistic analyses. Theoretical syntax suffers as much from inclusive evidence supporting multiple alternative analyses and potential interactions as grammar engineering does, if not more. Adapting an approach similar to that of CLIMB for the CoreGram grammars would therefore, in my opinion, mean a significant improvement in their methodology.

6 Discussion and Conclusion

This summary has introduced the idea of using metagrammar engineering as a general approach for grammar development. I have argued that this proposal addresses a fundamental challenge in grammar engineering and syntactic theory. The implementation of this idea CLIMB and several studies revealing its potential have been outlined. Finally, I discussed the most closely related work. In this section, I will provide a brief discussion of the main open issues, which is followed by concluding remarks.

The main challenge in adopting CLIMB as a general method for writing HPSG grammars is the learning curve grammar engineers may experience when starting to work with CLIMB. As mentioned in Section 2, switching back and forth between procedural and declarative programming can be a hurdle even for proficient coders in both Python and TDL. The main focus of future work on CLIMB will therefore be to design a declarative form of CLIMB that supports the full flexibility of procedural CLIMB. Another obvious direction of future work is to expand the studies in German syntax. The experiments with gCLIMB confirmed earlier hypotheses about the efficiency of alternative analyses, but their overall validity would need to confirmed by testing their interaction with analyses in other large HPSG grammars for German.

The thesis did present several results that show the potential of CLIMB. It is easier to adapt a grammar written in CLIMB to cover phenomena that behave slightly differently in a related language. Three alternative analyses for German auxiliaries and word order were compared in grammars covering a wide range of phenomena. Their efficiency was compared and a study involving the interaction between linguistic properties, alternative analyses and efficiency in natural language generation was conducted using CLIMB. These results show that CLIMB can be used to conduct empirical research that would be significantly less feasible (if not virtually impossible) in traditional grammar engineering. The spring cleaning algorithm developed to support CLIMB also led to new insights into linguistic precision grammars. The algorithm was applied to a number of grammars using the Grammar Matrix core resulting in a unique study on the way the Matrix core is used in individual grammars.

Overall, the methodology proposed in this thesis and the software developed to support it provide a platform for research on linguistic precision grammars that could not be carried out before, or only with great difficulty. This thesis thus succeeded in its objective to enhance empirical research for linguistically motivated precision grammars. This work presented several indicative results which suggest topics to investigate in future work, which is exactly the outcome one would want when introducing a new methodology. I believe that a new method for carrying out research should first and foremost offer new directions of investigation. It is my hope that this work will inspire fellow researchers to take the tools offered by this thesis, explore and learn more about grammars of natural language.

References

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

Avgustinova, Tania and Zhang, Yi. 2009. Parallel Grammar Engineering for Slavic Languages. In *Proceedings of GEAF*, Singapore.

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

Bender, Emily M. 2008. Evaluating a Crosslinguistic Grammar Resource: A Case Study of Wambaya. In *Proceedings of ACL-08: HLT*, pages 977–985, Columbus, Ohio: Association for Computational Linguistics.

Bender, Emily M. 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.

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

Bender, Emily M., Flickinger, Dan and Oepen, Stephan. 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 (eds.), *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, pages 8–14, Taipei, Taiwan.

Bender, Emily M., Flickinger, Dan and Oepen, Stephan. 2011. Grammar Engineering and Linguistic Hypothesis Testing: Computational Support for Complexity in Syntactic Analysis. In *Language from a Cognitive Perspective: Grammar, Usage and Processing*, pages 5–29, Stanford, USA: CSLI Publications.

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

Black, Cheryl A. 2004. Parser And Writer for Syntax, paper presented at the International Conference on Translation with Computer-Assisted Technology: Changes in Research, Teaching, Evaluation, and Practice, University of Rome "La Sapienza", April 2004.

Black, Cheryl A. and Black, H. Andrew. 2009. PAWS: Parser And Writer for Syntax: Drafting Syntactic Grammars in the Third Wave. In *SIL Forum for Language Fieldwork*, volume 2.

Butt, Miriam, Dyvik, Helge, King, Tracy Holloway, Masuichi, Hiroshi and Rohrer, Christian. 2002. The Parallel Grammar Project. In John Carroll, Nelleke Oostdijk and Richard Sutcliffe (eds.), *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, pages 1–7.

Candito, Marie-Hélène. 1999. *Organisation modulaire et paramétrable de grammaires électroniques lexicalisées. Application au francais et á l'italien*. Ph. D.thesis, l'université Paris 7.

Carpenter, Bob. 1992. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theorertical Computer Science, No. 32, New York, USA: Cambridge University Press.

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

Copestake, Ann, Flickinger, Dan, Sag, Ivan and Pollard, Carl. 2005. Minimal Recursion Semantics. An Introduction. *Journal of Research on Language and Computation* 3(2–3), 281 – 332.

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

Fokkens, Antske. 2011. Metagrammar engineering: Towards systematic exploration of implemented grammars. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1066–1076, Portland, Oregon, USA: Association for Computational Linguistics.

Fokkens, Antske and Avgustinova, Tania. 2013. SlaviCLIMB: Combining exp ertise for Slavic grammar development using a metagrammar. In Denys Duchier and Yannick Parmentier (eds.), *Proceedings of the Workshop on High-Level Methodologies for Grammar Engineering at ESSLLI 2013*, Düsseldorf, Germany.

Fokkens, Antske, Avgustinova, Tania and Zhang, Yi. 2012. CLIMB grammars: three projects using metagrammar engineering. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Jan Odijk and Stelios Piperidis (eds.), *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 2012)*, Instanbul, Turkey: European Language Resources Association (ELRA).

Fokkens, Antske and Bender, Emily M. 2013. Time Travel in Grammar Engineering. Using a Metagrammar to Broaden the Search Space. In Denys Duchier and Yannick Parmentier (eds.), *Proceedings of the Workshop on High-Level Methodologies for Grammar Engineering at ESSLLI 2013*, Düsseldorf, Germany.

Fokkens, Antske, Zhang, Yi and Bender, Emily. 2011. Spring Cleaning and Grammar Compression: Two Techniques for Detection of Redundancy in HPSG grammars. In *Proceedings of the 25th PACLIC*, Singapore, Singapore.

Fokkens, Antske S. 2010. Documentation for the Grammar Matrix word order library. Technical Report, Saarland University.

Hinrichs, Erhard and Nakazawa, Tsuneko. 1994. Linearizing AUXs in German Verbal Complexes. In John Nerbonne, Klaus Netter and Carl Pollard (eds.), *German in HPSG*, Chapter 1, Stanford, USA: CSLI.

King, Tracy Holloway, Forst, Martin, Kuhn, Jonas and Butt, Miriam. 2005. The Feature Space in Parallel Grammar Writing. *Research on Language and Computation, Special Issue on Shared Representations in Multilingual Grammar Engineering* 3(2), 139–163.

Marimon, Montserrat. 2010. The Spanish Resource Grammar. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner and Daniel Tapias (eds.), *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC 2010)*, Valetta, Malta: European Language Resources Association (ELRA).

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

Müller, Stefan. 2013. The CoreGram Project: A Brief Overview and Motivation. In Denys Duchier and Yannick Parmentier (Eds) (eds.), *Proceedings of the Workshop on High-level Methodologies for Grammar Engineering (HMGE 2013)*, Düsseldorf, Germany.

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

Müller, Stefan and Ørsnes, Bjarne. to appear. Danish in Head-Driven Phrase Structure Grammar. Ms, Freie Universität Berlin.

Pollard, Carl and Sag, Ivan. 1994. *Head-Driven Phrase Structure Grammar*. Chicago, USA: University of Chicago Press.

Ranta, Aarne. 2009. The GF Resource Grammar Library. *Linguistic Issues in Language Technology* 2(2).

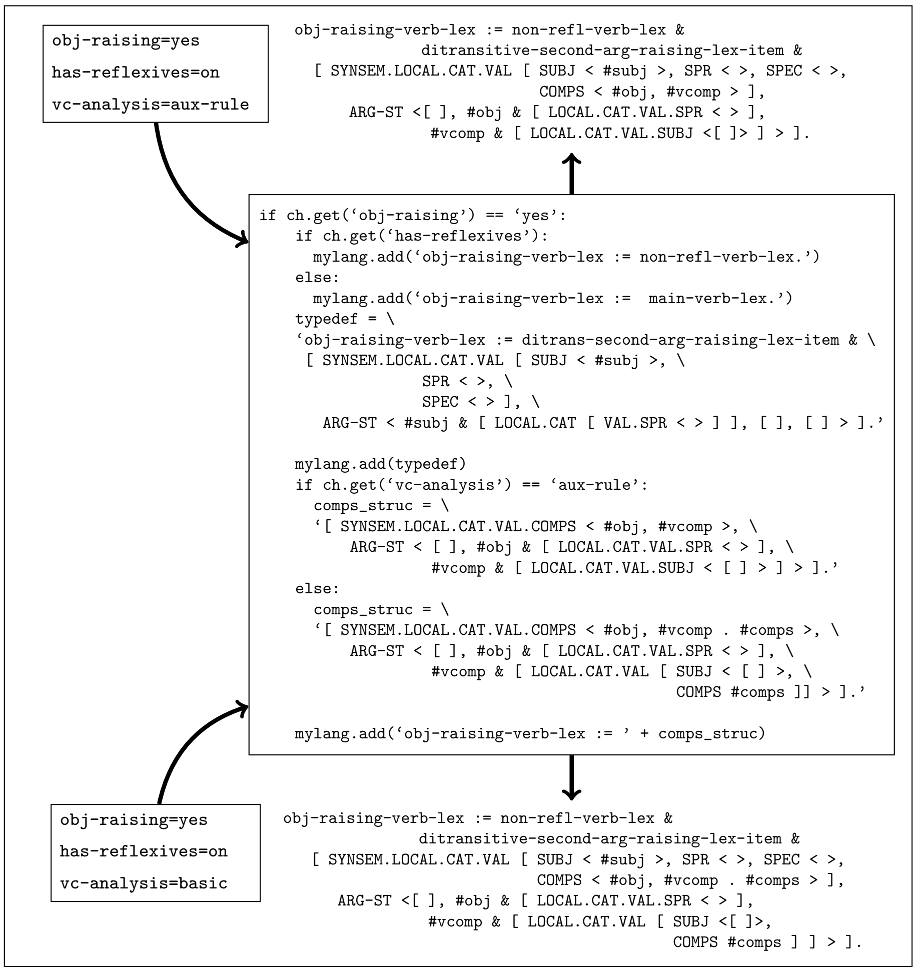
Riezler, Stefan, King, Tracy Holloway, Kaplan, Ronald M., Crouch, Richard, III, John T. Maxwell and Johnson, Mark. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of ACL*.

Siegel, Melanie and Bender, Emily M. 2002. Efficient Deep Processing of Japanese. In *Proceedings of the 3rd Workshop on Asian Language Resources and International Standardization at the 19th International Conference on Computational Linguistics*, Taipei, Taiwan.

Sygal, Yael and Wintner, Shuly. 2011. Towards Modular Development of Typed Unification Grammars. *Computational Linguistics* 37(1), 29–74.

Uszkoreit, Hans. 1987. *Word Order and Constituent Structure in German*. Stanford, USA: CSLI Publications.

A Appendix



```
obj-raising=yes
has-reflexives=on
vc-analysis=aux-rule
```

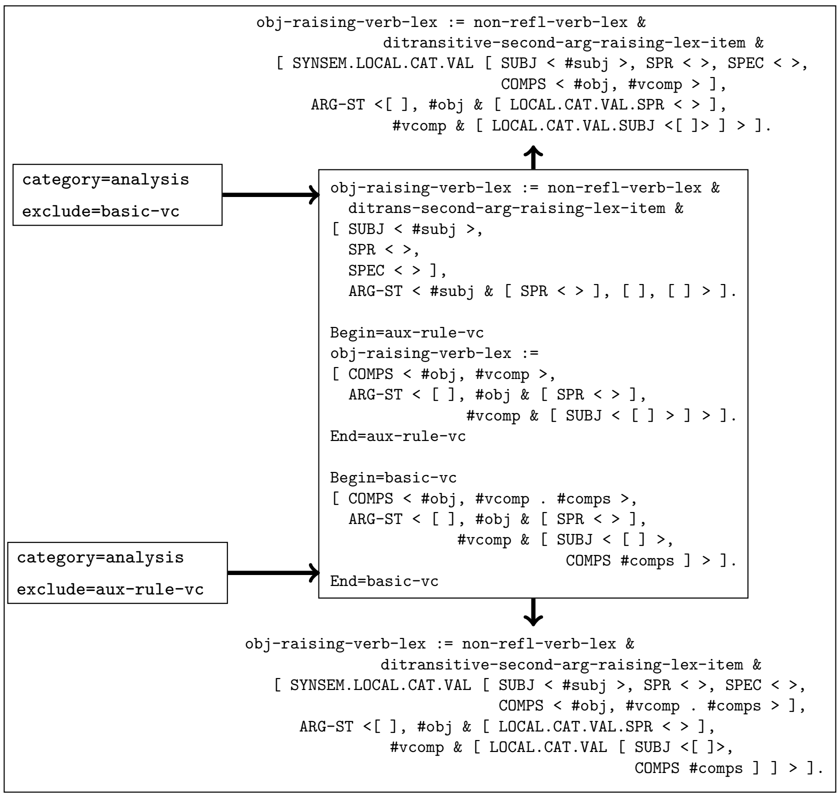
```
obj-raising-verb-lex := non-refl-verb-lex &
              ditransitive-second-arg-raising-lex-item &
  [ SYNSEM.LOCAL.CAT.VAL [ SUBJ < #subj >, SPR < >, SPEC < >,
                           COMPS < #obj, #vcomp > ],
      ARG-ST <[ ], #obj & [ LOCAL.CAT.VAL.SPR < > ],
               #vcomp & [ LOCAL.CAT.VAL.SUBJ <[ ]> ] > ].
```

```
if ch.get('obj-raising') == 'yes':
    if ch.get('has-reflexives'):
      mylang.add('obj-raising-verb-lex := non-refl-verb-lex.')
    else:
      mylang.add('obj-raising-verb-lex :=  main-verb-lex.')
    typedef = \
    'obj-raising-verb-lex := ditrans-second-arg-raising-lex-item & \
     [ SYNSEM.LOCAL.CAT.VAL [ SUBJ < #subj >, \
                 SPR < >, \
                 SPEC < > ], \
       ARG-ST < #subj & [ LOCAL.CAT [ VAL.SPR < > ] ], [ ], [ ] > ].'

    mylang.add(typedef)
    if ch.get('vc-analysis') == 'aux-rule':
      comps_struc = \
      '[ SYNSEM.LOCAL.CAT.VAL.COMPS < #obj, #vcomp >, \
          ARG-ST < [ ], #obj & [ LOCAL.CAT.VAL.SPR < > ], \
                   #vcomp & [ LOCAL.CAT.VAL.SUBJ < [ ] > ] > ].'
    else:
      comps_struc = \
      '[ SYNSEM.LOCAL.CAT.VAL.COMPS < #obj, #vcomp . #comps >, \
          ARG-ST < [ ], #obj & [ LOCAL.CAT.VAL.SPR < > ], \
                   #vcomp & [ LOCAL.CAT.VAL [ SUBJ < [ ] >, \
                                         COMPS #comps ]] > ].'

    mylang.add('obj-raising-verb-lex := ' + comps_struc)
```

```
obj-raising=yes
has-reflexives=on
vc-analysis=basic
```

```
obj-raising-verb-lex := non-refl-verb-lex &
              ditransitive-second-arg-raising-lex-item &
  [ SYNSEM.LOCAL.CAT.VAL [ SUBJ < #subj >, SPR < >, SPEC < >,
                           COMPS < #obj, #vcomp . #comps > ],
     ARG-ST <[ ], #obj & [ LOCAL.CAT.VAL.SPR < > ],
               #vcomp & [ LOCAL.CAT.VAL [ SUBJ <[ ]>,
                                          COMPS #comps ] ] > ].
```

Figure 1: Snippet of procedural CLIMB code with alternative choices and their output. The small boxes to the left of the figure are partial choices files, while the (declarative) TDL code at the top and bottom shows the system output according to those choices. This example illustrates variations of object raising verbs depending on the presence or absence of reflexives and chosen analysis for verbal clusters.



```
obj-raising-verb-lex := non-refl-verb-lex &
               ditransitive-second-arg-raising-lex-item &
  [ SYNSEM.LOCAL.CAT.VAL [ SUBJ < #subj >, SPR < >, SPEC < >,
                           COMPS < #obj, #vcomp > ],
      ARG-ST <[ ], #obj & [ LOCAL.CAT.VAL.SPR < > ],
               #vcomp & [ LOCAL.CAT.VAL.SUBJ <[ ]> ] > ].
```

```
category=analysis
exclude=basic-vc
```

```
obj-raising-verb-lex := non-refl-verb-lex &
  ditrans-second-arg-raising-lex-item &
[ SUBJ < #subj >,
  SPR < >,
  SPEC < > ],
  ARG-ST < #subj & [ SPR < > ], [ ], [ ] > ].

Begin=aux-rule-vc
obj-raising-verb-lex :=
[ COMPS < #obj, #vcomp >,
  ARG-ST < [ ], #obj & [ SPR < > ],
               #vcomp & [ SUBJ < [ ] > ] > ].
End=aux-rule-vc

Begin=basic-vc
[ COMPS < #obj, #vcomp . #comps >,
  ARG-ST < [ ], #obj & [ SPR < > ],
              #vcomp & [ SUBJ < [ ] >,
                         COMPS #comps ] > ].
End=basic-vc
```

```
category=analysis
exclude=aux-rule-vc
```

```
obj-raising-verb-lex := non-refl-verb-lex &
               ditransitive-second-arg-raising-lex-item &
  [ SYNSEM.LOCAL.CAT.VAL [ SUBJ < #subj >, SPR < >, SPEC < >,
                           COMPS < #obj, #vcomp . #comps > ],
      ARG-ST <[ ], #obj & [ LOCAL.CAT.VAL.SPR < > ],
               #vcomp & [ LOCAL.CAT.VAL [ SUBJ <[ ]>,
                                          COMPS #comps ] ] > ].
```

Figure 2: Snippet of declarative CLIMB code with alternative choices and their output. It shows the implementation of a basic type for object raising in declarative CLIMB. The example includes alternative additions made to the basic type depending on the analysis chosen for verbal clusters (like in Fig. 1)