# Time Travel in Grammar Engineering: Using a Metagrammar to Broaden the Search Space*

Antske Fokkens[1] and Emily M. Bender[2]

[1] The Network Institute, VU University Amsterdam,
De Boelaan 1105, 1081 HV Amsterdam, The Netherlands
antske.fokkens@vu.nl,
[2] Department of Linguistics, University Washington,
Box 354340, Seattle WA 98195-4340, USA
ebender@uw.edu

**Abstract.** In syntactic research, often more than one analysis can be found to account for available data. Interaction with analyses of other phenomena may help choose among them, but only phenomena analyzed in the past are taken into account. As a result, syntactic research is influenced by the order in which phenomena are treated. This paper presents a metagrammar-based methodology for maintaining alternative analyses over time. This way, we can use evidence from phenomena to be analyzed in the future to choose between analyses. We describe CLIMB, a methodology that implements this idea for HPSG grammars in TDL.

**Keywords:** Linguistic hypothesis testing, metagrammars

## 1 Introduction

Grammars, both those built by linguists and the natural objects they are intended to model, are complex objects (Bierwisch, 1963, Bender 2008a, Bender et al. 2011). A key benefit of grammar engineering as an approach to syntactic research is that it allows 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).

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 at each later choice point and thus the order in which phenomena are added affects the result (Fokkens, 2011). We argue (with Fokkens (2011)) 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

* The authors thank Ned Letcher and the anonymous reviewers for their feedback which helped improve this paper. All remaining errors are our own.

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, due to interactions with other phenomena, and they have been put on hold for now. Tracy King (p.c.) reports an ongoing discussion within ParGram (Butt et al. 2002) on whether adjectives have subjects or not. 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.

This paper presents the workflow of a metagrammar engineering approach that helps avoid such problems. §2 discusses the theoretical question of what makes an analysis correct. In §3, we discuss the role a metagrammar can play when using grammar engineering for hypothesis testing. §4 presents the workflow of a specific approach for using this idea for building grammars in HPSG (Pollard and Sag, 1994). We compare our approach to related work in §6.

## 2 Correct analyses

We take it as uncontroversial that grammar engineers, whether working towards practical ends or building grammars to test linguistic hypotheses, strive to develop correct analyses. But what does it mean for a linguistic analysis to be correct? In order to answer that question, we first have to pin down our modeling domain. Many linguists (e.g. Chomsky (1986)) take the modeling domain to be internalized knowledge of language. That is, the rules and principles proposed in the grammatical model should stand in a transparent relation to the rules and principles that speakers encode in 'wet-ware'. However, we do not have direct access to speaker knowledge. Instead, our primary data include attested utterances (written and spoken) as well as speaker intuitions about the meanings of those utterances (entailments, paraphrase relations) and their acceptability.

The first test of a model of grammar, then, is that it should be able to account for the available data: Chomsky's (1965) 'descriptive adequacy' is a prerequisite for any kind of 'explanatory adequacy'. However, the available data typically vastly underdetermine the analysis (Wasow, 1978; Soames, 1984; Bender (2010): Even with the large data sets that implemented grammars can work with, and even fixing the general grammatical framework, there are always multiple ways to approach the analysis of any given phenomenon. Possible sources of information constraining the choice between alternative analyses include psycholinguistic evidence (Sag and Wasow, 2011) and cross-linguistic comparability. That is, if an analysis works for comparable phenomena across different languages, it can be considered a point in favor of the analysis, though the weight given to such evidence varies depending on the theoretical status of universals in a framework.

Here, however, we will focus on evidence which comes from the rest of the grammar. A correct analysis should not only work correctly in isolation, but also work in the larger context of the grammar, making correct predictions when it interacts with other analyses. This is part of descriptive adequacy and therefore

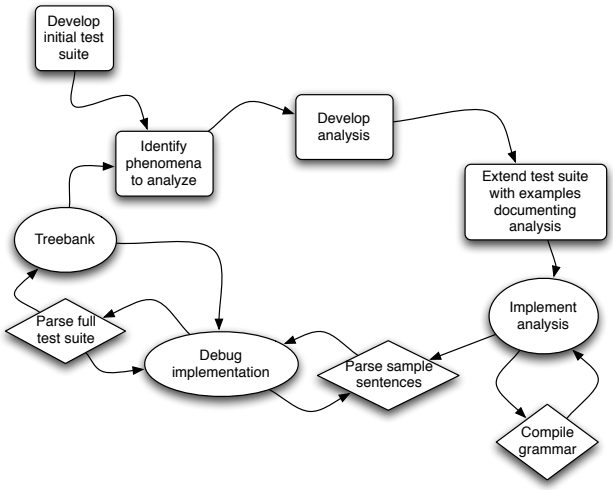


**Fig. 1.** Grammar engineering workflow, from 2011, p. 10

uncontroversial. Care should be taken however with its application, as we need to distinguish between *phenomena* and their *analyses*. If an analysis leads to incorrect predictions regarding the interaction with another phenomenon, then the analysis can be shown to be in error. However, if the incorrect predictions come from the interaction of the analyses of two phenomena, then it is not clear a priori which of the analyses is incorrect.

## 3 Metagrammar engineering for hypothesis testing

Because grammars are such complex objects, testing hypotheses about grammatical phenomena generally requires the development of complex models, with many interacting parts. Bender et al. (2011) illustrate how grammar engineering, together with regression testing, can help linguists manage the complexity of their models. This involves both testing the interactions between analyses and testing analyses against much larger collections of data than is feasible by hand. Bender et al. illustrate the process with the diagram in Fig. 1. Grammar engineering is a powerful tool for linguistic hypothesis testing because humans and computers are good at different things: Whereas humans are (still!) better suited to the task of developing linguistic analyses, computers are better at systematic checking and can verify the interaction of analyses on large amounts of data.

### 3.1 Multiple analyses

The process depicted in Fig. 1 is cyclic, with new phenomena being added on each successive pass. The testsuites document the analyses that have been implemented and allow the linguist to ensure that later work does not break what

has gone before. This is key to the ability of grammar engineering to facilitate linguistic hypothesis testing. However, when we view the process of grammar engineering in this light, it also becomes apparent that phenomena considered earlier in the development of a grammar and their analyses have an asymmetrical influence on analyses of phenomena developed later (see also Bender (2008b)).

This asymmetrical influence is unfortunate: It is fairly common for a key phenomenon constraining the choice of analysis of another phenomenon to be only addressed after several further passes through the cycle. In the meantime, whichever analysis was chosen of the phenomenon implemented earlier may become deeply embedded in the growing grammar. This has several unfortunate consequences: First, over time, it is easy to forget what alternative analyses were available. Second, the longer an analysis has been part of a grammar, the more other analyses are likely to depend on it in some way. As noted in the introduction, this leads to scenarios where it becomes cumbersome or impractical to change an analysis, even when it is discovered to be suboptimal. Finally, even if a grammar engineer is inspired to revise a deeply-embedded analysis, it is simply not possible to explore all the paths-not-taken, that is, all the alternative analyses of the various interacting phenomena that might have been just slightly more desirable had the revised analysis been the one chosen in the first place.

In order to escape from this asymmetrical influence problem, what is required is the ability to explore multiple development paths. As described in the next section, this is exactly what metagrammar engineering provides.

### 3.2 A Many-Model interpretation

We will illustrate how a metagrammar may help improve this situation by drawing a parallel to the many-world interpretation of quantum mechanics (Everett 1957, DeWitt 1972). Quantum mechanics can predict the probability of a location of a photon and this prediction forms a wave function. However, as soon as the photon is observed, the probability function is reduced to one point and, according to the alternative Copenhagen Interpretation, the wavefunction collapses. The many-world interpretation rejects this idea and maintains that the alternative worlds in which the photon is at another location than the one observed are real, implying that alternative histories and futures are real. The analogue we imagine for grammar engineering is a many-model interpretation, where each model considers different analyses to be correct. Each implementation decision we make places us in a given model. While making a decision in grammar engineering, the grammar engineers sets off on a specific road and the route that is taken (the order in which phenomena are considered, the choices made concerning their implementations) influences the final destination (the resulting grammar). However, unlike real life, where we are stuck in a specific world and cannot explore alternatives, we can look at alternative models for grammars. The only problem is that it is not feasible to have a clear picture of all consequences of a specific decision while following the traditional grammar engineering approach. But what if we could monitor more than one model at the same time? What if, instead of making a decision to commit to a specific model,

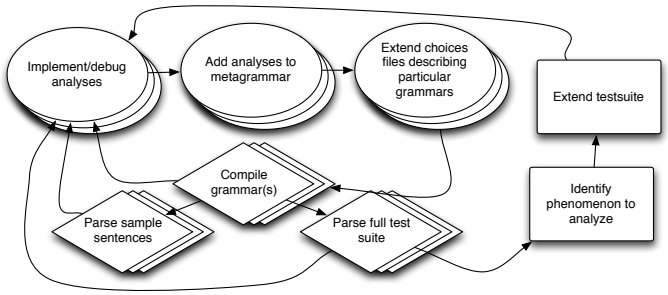


**Fig. 2.** Metagrammar engineering workflow

we follow a couple of models for a while, and test them with new analyses until we have gathered enough evidence to feel comfortable about a decision?

The methodology of metagrammar engineering can achieve this. The metagrammar can contain alternative analyses, including the alterations needed to make them interact correctly with analyses for other phenomena. Alternative models can be generated by selecting different options in the metagrammar. In §4, we describe a specific approach that applies this idea to HPSG grammars (Pollard and Sag, 1994) written in the TDL (Copestake, 2000).[3] It should be noted, however, that the idea of using a metagrammar and code generation to monitor alternative grammar models is both theory- and software-independent.

Fig. 2 schematizes the general workflow with a metagrammar approach. The stacked ovals represent manual processes that manipulate sets of alternative analyses, and the stacked diamonds automatic processes involving alternative grammars compiled out of those resources. The grammar engineer has the option of adding multiple alternative analyses for any phenomenon and, maintains a set of so-called 'choices' files specifying grammars built out of particular sets of choices among the analyses. The testing cycle involves compiling all of these grammars and parsing the testsuite with all of them. Using the many worlds metaphor, the resources encoding the alternative analyses and grammars are what define the space of models that the grammar engineer can monitor, the testsuite is the lens through which the monitoring is done, and the automated processing steps constitute the observations of each of the many 'worlds'.

## 4 The CLIMB approach

This section describes the CLIMB[4] approach for metagrammar engineering. We first describe how CLIMB emerged from the Grammar Matrix (Bender et al. 2010), and then describe the three variations of CLIMB that currently exist: the traditional procedural CLIMB, declarative CLIMB and SHORT-CLIMB.

[3] TDL is the DELPH-IN (www.delph-in.net) joint reference formalism.

[4] Comparative Libraries of Implementations with a Matrix Basis; See moin.delph-in.net/ClimbTop for more project information and software download.

### 4.1 The Grammar Matrix and CLIMB

CLIMB (Fokkens 2011, Fokkens et al. 2012) builds on the LinGO Grammar Matrix (Bender et al. 2010), sharing with it the key idea of grammar customization. Grammar customization is the practice of generating grammar code from stored analyses on the basis of some input. There are, however, important differences, stemming from the goals of the projects: the Grammar Matrix aims to help (possibly novice) grammar engineers start new grammars, where the goal of CLIMB is to support long-term grammar development for experienced grammar engineers.

The Grammar Matrix applies grammar customization to the problem of rapid development of precision grammars. It stores a core grammar and libraries of analyses of cross-linguistically varying phenomena, which users can select in order to create a grammar fragment for (ideally) any human language. The project emphasizes typological coverage, and therefore can only add phenomena slowly, as each one must be grounded in thorough typological research. CLIMB generalizes the idea of grammar customization to metagrammar engineering, 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. This entails further differences between the projects: Matrix users access the system through a web-based questionnaire that hides the underlying HPSG analyses, while CLIMB users are actively developing HPSG analyses (as implemented in TDL). Also, using grammar customization 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.

### 4.2 Procedural CLIMB

The original version of CLIMB (procedural CLIMB) builds directly on the Grammar Matrix, by simply taking the customization 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.

Fig. 3 gives a sample analysis as implemented in procedural CLIMB. The metagrammar code contains control statements which check for certain properties in the choices file and statements which output TDL accordingly. 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 the ways in which the analysis of object raising verbs depends on the presence or absence of reflexives in the grammar specification, on the one hand, and the chosen analysis of verbal clusters on the other.

Procedural CLIMB can capture alternative analyses and thus achieves the goal of maintaining analyses in parallel. It has some additional advantages common

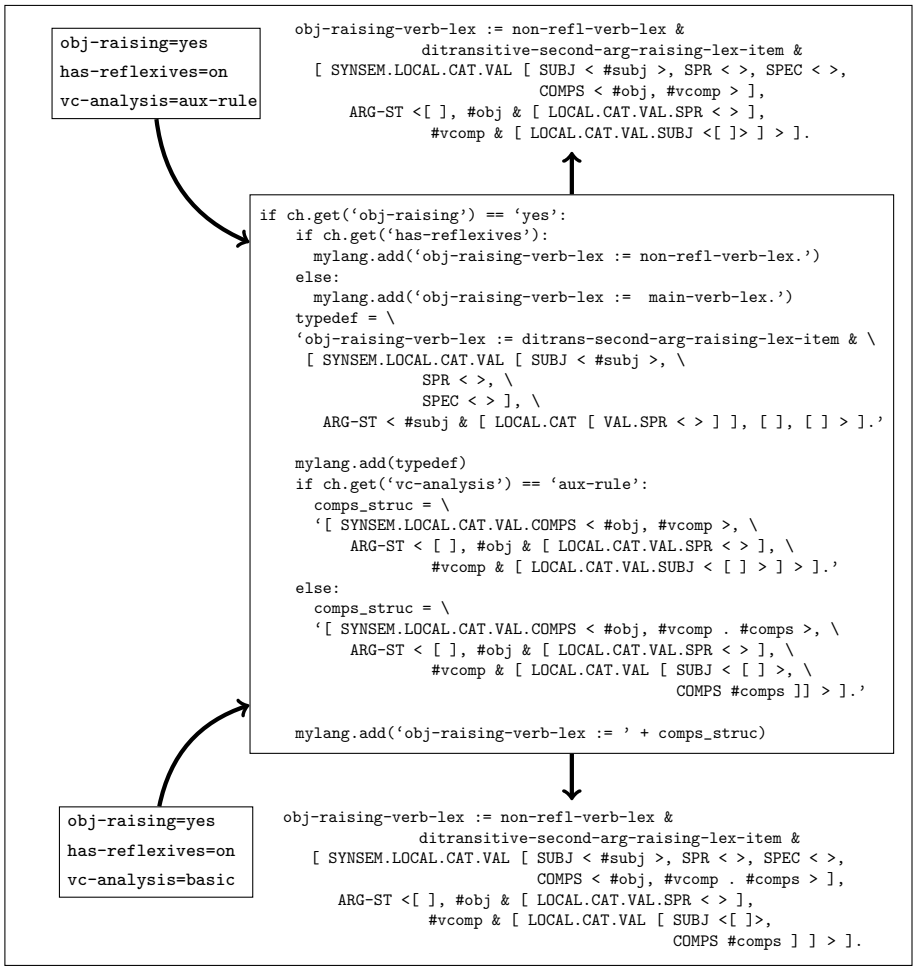


**Fig. 3.** Snippet of procedural CLIMB code with alternative choices and their output

to metagrammar approaches, including increased modularity, more phenomena-oriented organization, facilitation of multilingual grammar development and application dependent variations in grammars. A draw-back of this approach, however, 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.

### 4.3 Declarative CLIMB

Declarative CLIMB addresses this drawback of procedural CLIMB. Fig. 4 shows the implementation of a basic type for object raising in declarative CLIMB. 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 example includes alternative additions made to the basic type depending on the analysis chosen for verbal clusters (like in Fig. 3). The choices file indicates which analyses in the metagrammar should not be selected for the generated grammar. Procedural Python implementations are still used to interpret a choices file and merge type definitions from different locations in the metagrammar, but grammar engineers can treat these like a black box.

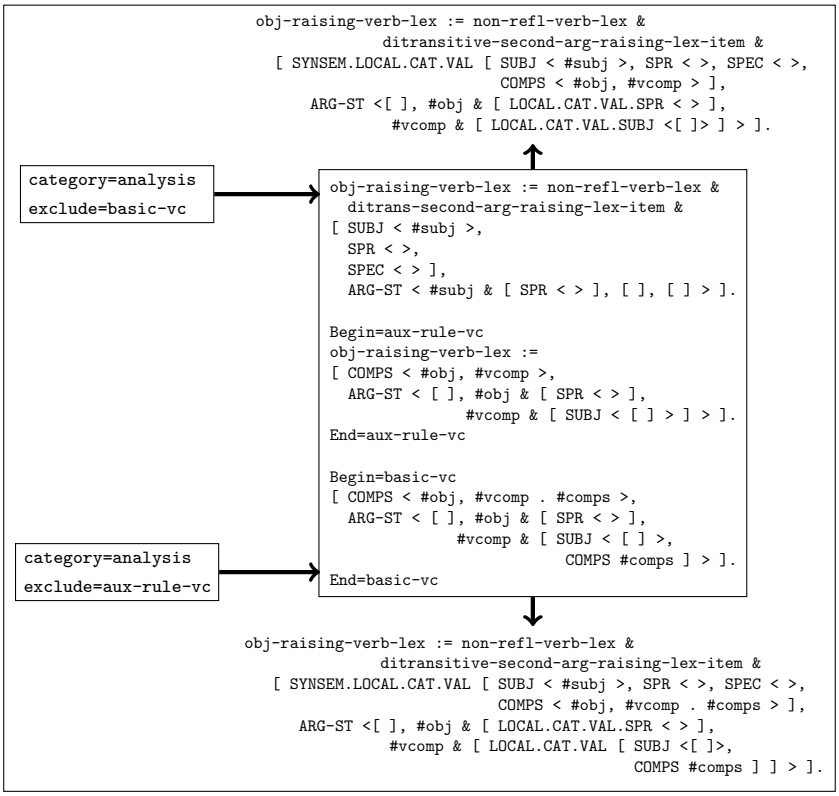


**Fig. 4.** Snippet of declarative CLIMB code with alternative choices and their output

### 4.4 SHORT-CLIMB

Both procedural and declarative CLIMB assume that the grammar is built within a CLIMB approach from the ground up. But mature grammars can also benefit from the ability to explore alternative analyses (even if most of what has been implemented before adopting CLIMB is kept constant). It is not practical to take a very large grammar and re-implement it from the ground up just to get this benefit. Instead, SHORT-CLIMB[5] provides support for maintaining alter-

[5] Starting High On a Roof Top CLIMB

native analyses for grammars that have been developed the traditional way. An alternative analysis can be included by writing declarative statements on types and (partial) type definitions to be added to the grammar, types or constraints to be removed from the grammar and types that need to be modified.

The SHORT-CLIMB processing code takes a grammar written in TDL and a set of modifications as input and produces a revised grammar. It can optionally also provide a SHORT-CLIMB definition file to convert the revised grammar into the original grammar. Finally, it can produce a reduced grammar that only contains the common properties of both analyses, together with two files that can either create the original or the revised analysis using SHORT-CLIMB software. As such, SHORT-CLIMB can be used to convert a traditionally written grammar to declarative CLIMB step by step.

## 5 Scalability

CLIMB has been used to develop gCLIMB, a metagrammar for Germanic languages, developed with the goals of exploring the efficiency of alternative analyses and the applicability of the CLIMB methodology at the scale of resource grammars (Fokkens, in progress). The results in Fokkens (in progress) indicate that CLIMB scales well for longterm projects. gCLIMB has been developed for two and a half years and in spite of it having been a side project for most of that time, it already covers a wide range of phenomena for German. It contains alternative analyses for word order and auxiliaries, two phenomena that interact heavily with other phenomena in the grammar together resulting in five variations. In addition, the system provides for different approaches to the lexicon, including fully inflected or abstract forms of morphemes as well as a variation that integrates the lexicon and morphology of Cheetah (Cramer, 2011). In total, there are 25 different choices files specifying grammars representing different combinations of these analytical possibilities. As far as quantity is concerned, gCLIMB was built on top of a version of the Grammar Matrix that had 3,651 lines of code in the linguistic libraries. The metagrammar's libraries now consist of 13,032 lines of code, compared to 6,250 lines of code in the current version of the Grammar Matrix customization system. The choices files for these grammars have 1,970 lines of definitions (for small grammars used for testing) up to 3,657 lines (for creating the complete Cheetah core grammar and incorporating its lexicon). A customized grammar consists of approximately 7,300 lines of code.

## 6 Related Work

Metagrammars have been in use for over a decade, with notable long term efforts such as the MetaGrammar project (Candito 1998, Clément and Kinyon 2003) and GF (Ranta 2004, Ranta 2011). They have supported code sharing (Clément and Kinyon 2003, Ranta 2011), increased modularity, provided means for generalizations (Candito, 1998), combined expertise from linguists and engineers (Ranta, 2004) and facilitated multilingual grammar development (Ranta, 2011).

The two projects that most closely resemble declarative CLIMB in their architecture are Sygal and Wintner's (2011) approach for modular development of typed unification grammars and the CoreGram Project (Müller, 2013). Like the work mentioned above, neither of these has addressed systematic comparison of alternative analyses, instead focusing on increasing modularity (Sygal and Wintner, 2011) and cross-linguistic grammar sharing (CoreGram). These difference in goals are reflected in differences in architecture.

Sygal and Wintner (2011) propose a structure that allows engineers to implement modules defining a part of a type hierarchy. These modules can be combined to form a complete grammar by specially defined functions. They point out the similarity between their approach and the Grammar Matrix customization system, where the main difference is that the grammar engineer does not have control over customization. This is exactly where CLIMB differs from the Grammar Matrix. Where Sygal and Wintner improve modularity by developing a mathematically well-founded model, CLIMB resulted from a practical approach to implementing alternative analyses, which requires increased modularity. This difference is also reflected in the verification of the approach. The former has been tested in a proof-of-concept implementation of the small hierarchy from in the appendix of Pollard and Sag (1994). CLIMB includes a grammar for German covering phenomena including subordinate clauses, complex predicates and extraposed comparatives. It seems promising that the theoretically well-founded approach and the practical broad-coverage approach are similar.

The CoreGram Project (Müller, 2013) is a bottom-up approach for cross-linguistic sharing of analyses between HPSG grammars running in TRALE (Meurers et al., 2002). Implementations that can be used for all grammars form a common core. When properties are observed that are shared by a subset of languages, they can be placed in subcores. Definitions belonging to the core, a subcore or language specific properties are defined in different files, and complete grammars are created by selecting the appropriate files. This approach partially provides the same functionality as declarative CLIMB. In fact, the CoreGram approach can be used to monitor alternative analyses, but this is not (yet) a goal of the CoreGram project. Declarative CLIMB differs from the CoreGram in providing a few additional features to facilitate the accommodation of alternative analyses. First, CLIMB allows the metagrammar engineer to consolidate all constraints associated with one alternative in a single location, even if in the compiled grammar they affect many different types, allowing the grammar engineer two views: one based on analyses and one based on complete types. Declarative CLIMB also supports the possibility of defining abbreviated paths which to a certain extent supports the exploration of alternative feature geometries. These differences can all be traced back to the different goals of CoreGram and CLIMB, respectively.

To our knowledge, the only work that suggests that a metagrammar can be used to test alternative analyses is Fokkens (2011), but that work focuses on efficiency of implemented grammars. As such, we believe this work is the first to address the role of a metagrammar in hypothesis testing for syntactic research.

## 7 Conclusion

This paper outlined the role metagrammars can play in linguistic hypothesis testing. We described CLIMB: a metagrammar engineering approach that provides the means to create HPSG grammars written in TDL. Three versions of CLIMB have been presented: procedural CLIMB following Fokkens (2011), declarative CLIMB and SHORT-CLIMB. Declarative CLIMB allows grammar engineers to define their metagrammar libraries declaratively in TDL and has been developed to make the approach more accessible to grammar engineers who are not trained in procedural programming. SHORT-CLIMB can be used to create libraries that apply changes to large grammars that have been developed using the traditional (non-metagrammar) way. It thus provides the possibility of testing alternative analyses systematically in a large scale grammar.

The advantages of implementing grammars for hypothesis testing are well-known (Bierwisch 1963, Müller 1999, Bender 2008b). Metagrammar engineering adds an additional advantage in that allows the syntactician to systematically explore alternative analyses over time. In traditional grammar engineering, analyses may be so deeply embedded in the grammar it becomes too cumbersome to go back and change them. Metagrammar engineering makes it possible to monitor multiple models of a grammar. If we include two alternative analyses in a metagrammar and evidence for a particular analysis is found in the far future, we have the complete paths of both analyses. There is thus no need to dive into past decisions instead we can simply follow the path of the right choice as if we time traveled and corrected our past decision.

## References


Bender, E.M.: Evaluating a crosslinguistic grammar resource: A case study of Wambaya. In: Proceedings of ACL-08: HLT, Columbus, Ohio (2008) 977–985

Bender, E.M.: Grammar engineering for linguistic hypothesis testing. In: Proceedings of the Texas Linguistics Society X Conference: Computational Linguistics for Less-Studied Languages, Stanford, CSLI Publications (2008) 16–36

Bender, E.M.: Reweaving a grammar for Wambaya: A case study in grammar engineering for linguistic hypothesis testing. LILT **3** (2010) 1–34

Bender, E.M., Drellishak, S., Fokkens, A., Poulson, L., Saleem, S.: Grammar customization. Research on Language & Computation **8**(1) (2010) 23–72

Bender, E.M., Flickinger, D., Oepen, S.: Grammar engineering and linguistic hypothesis testing: Computational support for complexity in syntactic analysis. In Bender, E., J.E., A., eds.: Language from a Cognitive Perspective: Grammar, Usage and Processing. Stanford: CSLI Publications. (2011) 5–29

Bierwisch, M.: Grammatik des deutschen Verbs. Volume II of Studia Grammatica. Akademie Verlag (1963)

Butt, M., Dyvik, H., King, T.H., Masuichi, H., Rohrer, C.: The parallel grammar project. In: Proceedings of the 2002 workshop on Grammar engineering and evaluation-Volume 15, Association for Computational Linguistics (2002) 1–7

Candito, M.: Building parallel LTAG for French and Italian. In: Proceedings of ACL and ICCL, Montreal, Canada (1998) 211–217

Chomsky, N.: Aspects of the Theory of Syntax. MIT Press, Cambridge, MA (1965)
Chomsky, N.: Knowledge of Language: Its Nature, Origin, and Use. Praeger, New York (1986)
Clément, L., Kinyon, A.: Generating parallel multilingual LFG-TAG grammars with a MetaGrammar. In: Proceedings of ACL, Sapporo, Japan (2003)
Copestake, A.: Appendix: Definitions of typed feature structures. Natural Language Engineering **6** (2000) 109–112
Cramer, B.: Improving the feasibility of precision-oriented HPSG parsing. PhD thesis, Saarland University (2011)
DeWitt, B.S.: The many-universes interpretation of quantum mechanics. In: Proceedings of the International School of Physics "Enrico Fermi" Course IL: Foundations of Quantum Mechanics. (1972)
Everett, H.: "Relative State" Formulation of Quantum Mechanics. Reviews of Modern Physics **29** (1957) 454–462
Fokkens, A.: Metagrammar engineering: Towards systematic exploration of implemented grammars. In: Proceedings of ACL, Portland, Oregon, USA (2011)
Fokkens, A., Avgustinova, T., Zhang, Y.: CLIMB grammars: three projects using metagrammar engineering. In: Proceedings of LREC 2012, Instanbul, Turkey (2012)
Fokkens, A.: Enhancing Empirical Research for Linguistically Motivated Precision Grammars. PhD thesis, Saarland University (in progress)
Marimon, M.: The Spanish resource grammar. In: Proceedings of LREC, European Language Resources Association (2010)
Meurers, W.D., Penn, G., Richter, F.: A web-based instructional platform for constraint-based grammar formalisms and parsing. In: Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics. (2002) 19–26
Müller, S.: Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche. Max Niemeyer Verlag, Tübingen (1999)
Müller, S.: The CoreGram project: Theoretical linguistics, theory development and verification. In: Proceedings of the workshop on High-level Methodologies for Grammar Engineering. (2013)
Pollard, C., Sag, I.: Head-Driven Phrase Structure Grammar. University of Chicago Press, Chicago, USA (1994)
Ranta, A.: Grammatical Framework: A type-theoretical grammar formalism. Journal of Functional Programming **14**(2) (2004) 145–189
Ranta, A.: Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford, USA (2011)
Riezler, S., King, T.H., Kaplan, R.M., Crouch, R., III, J.T.M., Johnson, M.: Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In: Proceedings of ACL. (2002)
Sag, I.A., Wasow, T.: Performance-compatible competence grammar. In Borsley, R., Borjars, K., eds.: Non-Transformational Syntax: Formal and Explicit Models of Grammar. Wiley-Blackwell (2011)
Siegel, M., Bender, E.M.: Efficient deep processing of Japanese. In: Proceedings of the 3rd Workshop on Asian Language Resources and International Standardization, Taipei, Taiwan (2002)
Soames, S.: Linguistics and psychology. Linguistics and Philosophy **7** (1984) 155–179
Sygal, Y., Wintner, S.: Towards modular development of typed unification grammars. Computational Linguistics **37**(1) (2011) 29–74
Wasow, T.: On constraining the class of transformational languages. Synthese **39** (1978)