Chapter 29

Computational linguistics and grammar engineering

Emily M. Bender
University of Washington

Guy Emerson
University of Cambridge

We discuss the relevance of HPSG for computational linguistics, and the relevance of computational linguistics for HPSG.

1 Introduction

From the inception of HPSG in the 1980s, there has been a close integration between theoretical and computational work (for an overview, see Flickinger, Pollard & Wasow 2018, Chapter 2 of this volume). In this chapter, we discuss computational work in HPSG, starting with the infrastructure that supports it (both theoretical and practical) in Section 2. Next we describe several existing large-scale projects which build HPSG or HPSG-inspired grammars (see Section 3) and the deployment of such grammars in applications including both those within linguistic research and otherwise (see Section 4). Finally, we turn to linguistic insights gleaned from broad-coverage grammar development (see Section 5).

2 Infrastructure

2.1 Theoretical considerations

There are several properties of HPSG as a theory that make it well-suited to computational implementation. First, the theory is kept separate from the formalism:

Emily M. Bender & Guy Emerson. 2018. Computational linguistics and grammar engineering. In Stefan Müller, Anne Abeillé, Robert D. Borsley & Jean-Pierre Koenig (eds.), *Head-Driven Phrase Structure Grammar: The handbook*, 555–575. Berlin: Language Science Press. DOI:??

the formalism is expressive enough to encode a wide variety of possible theories. While some theoretical work does argue for or against the necessity of particular formal devices (e.g. the shuffle operator (**FIXME-Reape**)), much of it proceeds within shared assumptions about the formalism. This is in contrast to work in the context of the Minimalist Program (Chomsky 1993), where theoretical results are typically couched in terms of modifications to the formalism itself. From a computational point of view, the benefit of differentiating between theory and formalism is that it means that the formalism is relatively stable. That in turns enables the development and maintenance of software systems that target the formalism, for parsing, generation, and grammar exploration (see Section **??** below for some examples).[1]

A second important property of HPSG that supports a strong connection between theoretical and computational work is an interest in both so-called 'core' and so-called 'peripheral' phenomena. Most implemented grammars are built with the goal of handling naturally occurring text.[2] This means that they will need to handle a wide variety of linguistic phenomena not always treated in theoretical syntactic work (**FIXME-Baldwin-et-al-Beauty**). A syntactic framework that excludes research on 'peripheral' phenomena as uninteresting provides less support for implementational work than does one, like HPSG or Construction Grammar, that values such topics (for a comparison of HPSG and Construction Grammar, see Müller 2018, Chapter 37 of this volume).

Finally, the type hierarchy characteristic of HPSG lends itself well to developing broad-coverage grammars which are maintainable over time (**FIXME-find-cite?**). The use of the type hierarchy to manage complexity at scale comes out of the work of Flickinger (1987) and others at HP labs in the project where HPSG was originally developed. The core idea is that any given constraint is (ideally) expressed only once on types which serve as supertypes to all entities that bear that constraint.[3] Such constraints might represent broad generalizations that apply to many entities or relatively narrow, indiosyncratic properties. By isolating any given constraint on one type (as opposed to repeating it in mutiple places), we build grammars that are easier to update and adapt in light of new data that

[1] There are implementations of Minimalism, notably **FIXME-Stabler** and **FIXME-Indianadiss** However, writing an implementation requires fixing the formalism, and so these are unlikely to be useful for testing theoretical ideas as the theory moves on.

[2] It is possible, but less common, to do implementation work strictly against testsuites of sentences constructed specifically to focus on phenomena of interest.

[3] Originally this only applied to lexical entries in Flickinger's work. Now it also applies phrase structure rules, lexical rules, and types below the level of the sign which are used in the definition of all of these.

require refinements to constraints. Having a single locus for each constraint also makes the types a very useful target for documentation (**FIXME:LTDB**) and grammar exploration (**FIXME:typediff**).

2.2 Practical considerations

The formalism of HPSG allows practical implementations, since feature structures are well-defined data structures. Furthermore, because HPSG is defined to be bi-directional, an implemented grammar can be used for both parsing and generation. In this section, we discuss how HPSG allows tractable algorithms, which enables linguists to empirically test hypotheses, and which also enables HPSG grammars to be used in a range of applications, as we will see in Sections 4.1 and 4.2, respectively.

2.2.1 Computational complexity

One way to measure how easy or difficult it is to use a syntactic theory in practical computational applications is to consider the *computational complexity* of parsing and generation algorithms (Gazdar & Pullum 1985). For example, we can consider how much computational time a parsing algorithm needs to process a particular sentence. For longer sentences, we would expect the amount of time to increase, but the more complex the algorithm is, the more quickly the amount of time increases. If we consider sentences containing n tokens, we can find the average amount of time taken, or the longest amount of time taken. We can then increase n, and see how the amount of time changes, both in the average case, and in the worst case.

At first sight, analysing computational complexity would seem to paint HPSG in a bad light, because the formalism allows us to write grammars which can be arbitrarily complex; in technical terminology, the formalism is *Turing-complete* (Johnson 1988: Section 3.4). However, as discussed in the previous section, there is a clear distinction between theory and formalism. Although the feature-structure formalism rules out the possibility of efficient algorithms that could cope with any possible feature-structure grammar, a particular theory (or a particular grammar) might well allow efficient algorithms.

The difference between theory and formalism becomes clear when comparing HPSG to other computationally-friendly frameworks, such as Combinatory Categorial Grammar (CCG),[4] or Tree Adjoining Grammar (TAG; Joshi 1987; Schabes

[4]For an introduction, see Steedman & Baldridge (2011). For a comparison with HPSG, see Kubota (2018), Chapter 34 of this volume.

et al. 1988)). The formalisms of CCG and TAG inherently limit computational complexity: for both of them, as the sentence length n increases, worst-case parsing time is proportional to n^6 (Kasami et al. 1989). This is a deliberate feature of these formalisms, which aim to be just expressive enough to capture human language, and not any more expressive. Building this kind of constraint into the formalism itself highlights a different school of thought from HPSG. Indeed, Müller (2015) explicitly argues in favor of developing linguistic analyses first, and improving processing efficiency second. As discussed above in Section 2.1, separating the formalism from the theory means that the formalism is stable, even as the theory develops.

It would be beyond the scope of this chapter to give a full review of parsing algorithms, but it is instructive to give an example. For grammars that have a context-free backbone (we can express every derivation as a phrase-structure tree plus constraints between mother and daughter nodes), it is possible to adapt the standard chart-parsing algorithm for context-free grammars. The basic idea is to parse "bottom-up", starting by finding analyses for each token in the input, and then finding analyses for increasingly longer sequences of tokens, until we reach the entire sentence. The resulting algorithm is more computationally complex than for a context-free grammar, because we are dealing with feature structures, rather than nonterminal symbols. While a context-free grammar allows a finite number of nonterminals, a feature-structure grammar may allow a infinite number of possible feature structures. For an HPSG grammar without recursive unary rules, this algorithm has a worst-case complexity of exponential time. This is less complex than for an arbitrary grammar (which means that this class of grammars is *not* Turing-complete), but more complex than for CCG or TAG. However, when parsing real corpora, it turns out that the average-case complexity is much better than we might expect (Carroll 1994). On the one hand, grammatical constructions do not generally combine in the worst-case way, and on the other hand, when a grammar writer is confronted with multiple possible analyses for a particular construction, they may opt for the analysis that is more efficient for a particular parsing algorithm. Oepen & Flickinger (1998) describe the use of test-suites for profiling the time efficiency of a grammar and parsing algorithm, when running on a representative sample of text.

2.2.2 Parse ranking

Ambituity in modifier attachment, part of speech assignment, and the like are well-known in linguistics, such that examples like (1) are stock in trade:

(1) a. I saw the kid with the telescope.
b. Visiting relatives can be annoying.

A well-constructed grammar should be expected to return multiple parses for each ambiguous sentence.

However, people are naturally very good at resolving ambiguity, which means most ambiguity is not apparent, even to linguists. It is only with the development of large-scale grammars that the sheer scale of ambiguity has become clear. For example, (2) might seem unambiguous, but there is a second reading, where *my favorite* is the topicalized object of *speak*, which would mean that town criers generally speak the speaker's favorite thing (perhaps a language) clearly. There is also a third, even more implausible reading, where *my favorite town* is the topicalized object. Such implausible readings don't easily come to mind, and in fact, the 2018 version of the English Resource Grammar (Flickinger 2000; 2011) gives a total of 21 readings for this sentence. With increasingly long sentences, such ambiguities stack up very quickly. For (3), the first line of a newspaper article,[5] the ERG gives 35,094 readings.

(2) My favorite town criers speak clearly.

(3) A small piece of bone found in a cave in Siberia has been identified as the remnant of a child whose mother was a Neanderthal and father was a Denisovan, a mysterious human ancestor that lived in the region.

While exploring ambiguity can be interesting for a linguist, typical practical applications require just one parse per input sentence and specifically the parse the best reflects the intended meaning (or only the top few parses, in case the one put forward as 'best' might be wrong). Thus, what is required is a *ranking* of the parses, so that the application can only use the most highly-ranked parse, or the top N parses.

Parse ranking is not usually determined by the grammar itself, because of the difficulty of manually writing disambiguation rules. Typically, a statistical system is used (Toutanova et al. 2002; 2005). First, a corpus is *treebanked*: for each sentence in the corpus, an annotator (often the grammar writer) chooses the best parse, out of all parses produced by the grammar. The set of all parses for a sentence is often referred to as the *parse forest*, and the selected best parse is often referred to as the *gold standard*. Given the gold parses for the whole corpus, a statistical system is trained to predict the gold parse from a parse forest, based

[5] https://www.theguardian.com/science/2018/aug/22/offspring-of-neanderthal-and-denisovan-identified-for-first-time

on many features[6] of the parse. From the example in (2), a number of a different features all influence the preferred interpretation: the likelihood of a construction (such as topicalization), the likelihood of a valence frame (such as transitive *speak*), the likelihood of a collocation (such as *town crier*), the likelihood of a semantic relation (such as speaking a town), and so on.

Because of the large number of possible parses, it can be helpful to *prune* the search space: rather than ranking the full set of parses, we can restrict attention to a smaller set of parses, which hopefully includes the correct parse. By carefully choosing how to restrict the parser's attention, we can drastically reduce processing time without hurting parsing accuracy. One method, called *supertagging*,[7] exploits the fact that HPSG is a lexicalized theory: choosing the correct lexical entry brings in rich information that can be exploited to rule out many possible parses. Thus if the correct lexical entry can be chosen prior to parsing (e.g. on the basis of the prior and following words), the range of possible analyses the parser must consider is drastically reduced. Although there is a chance that the supertagger will predict the wrong lexical entry, using a supertagger can often improve parsing accuracy, by ruling out parses that the parse-ranking model might incorrectly rank too high. Supertagging was first applied to HPSG by Matsuzaki et al. (2007), building on previous work for TAG (Bangalore & Joshi 1999) and CCG (Clark & Curran 2004). To allow multi-word expressions (such as *by and large*), where the grammar assigns a single lexical entry to multiple tokens, Dridan (2013) has proposed an extension of supertagging, called *ubertagging*, which jointly predicts both a segmentation of the input and supertags for those segments. Dridan manages to increase parsing speed by a factor of four, while also improving parsing accuracy.

Finally, in order to train these statistical systems, we need to first annotate a treebank. When there are many parses for a sentence, it can be time-consuming to select the best parse. To efficiently use an annotator's time, it can be helpful to use *discriminants*, properties which hold for some parses but not for others (Carter 1997). For example, discriminants might include whether to analyse an ambiguous token as a noun or a verb, or where to attach a prepositional phrase. This approach to treebanking also means that annotations can be re-used when the grammar is updated (Oepen et al. 2004; Flickinger et al. 2017).

[6] In the machine-learning sense of "feature", not the feature-structure sense.

[7] The name refers to *part-of-speech tagging*, which predicts a part-of-speech for each input token, from a relatively small set of part-of-speech tags. Supertagging is "super", in that it predicts detailed lexical entries, rather than simple tags.

2.2.3 Semantic dependencies

In practical applications of HPSG grammars, the full derivation trees and the full feature structures are often unwieldy, containing far more information than necessary for the task at hand. It is therefore often desirable to extract a concise semantic representation.

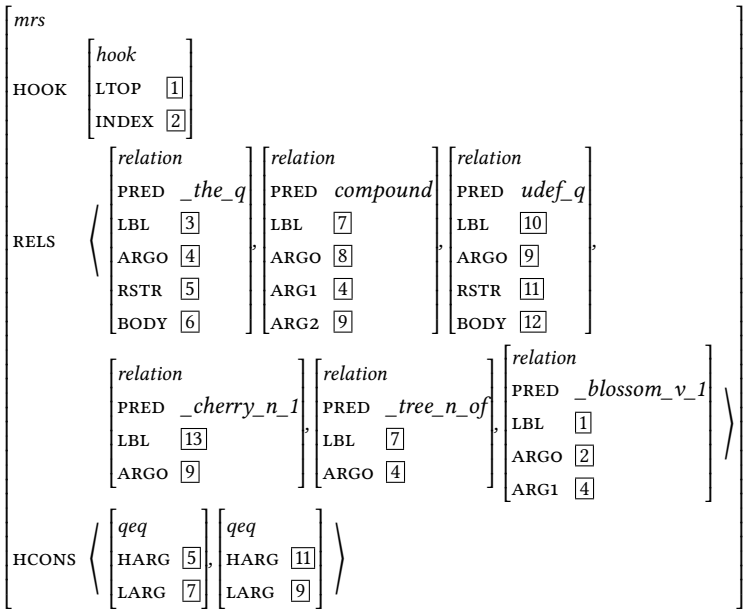
In computational linguistics, a popular approach to semantics is to represent the meaning of a sentence as a *dependency graph*, as this enables the use of graph-based algorithms.[8] Several types of dependency graph have been proposed based on Minimal Recursion Semantics (MRS; Copestake et al. 2005), with varying levels of simplification. The most expressive is Dependency Minimal Recursion Semantics (DMRS; Copestake 2009), which is fully interconvertible with MRS.[9] In contrast, Elementary Dependency Structures (EDS; Oepen & Lønning 2006) lose some scope information, which, for many applications, is less important than predicate-argument structure. Finally, DELPH-IN MRS Dependencies (DM; Ivanova et al. 2012) express predicate-argument structure purely in terms of the surface tokens, without introducing any abstract predicates. A comparison of MRS, DMRS, and DM is given in Figures 1 and 2. The existence of such dependency graph formalisms, as well as software packages to manipulate such graphs (Copestake et al. 2016), has made it easier to use HPSG grammars in a number of practical tasks, as we will discuss in Section 4.2.

3 Development of HPSG resources

In this section we describe various projects that have developed computational resources on the basis of or inspired by HPSG. As we'll discuss in Section 4 below, such resources can be used both in linguistic hypothesis testing as well as in various practical applications. The intended purpose of the resources influences the form that they take. CoreGram (§4.1.1) is primarily targeted as linguistic hypothesis testing, the Enju and Alpino parsers desdribed in Section 3.2 primarily at practical applications, and the resources produced by the DELPH-IN Consortium (§3.3) attempt to balance the two.

[8] In this section, we are concerned with *semantic* dependencies. For *syntactic* dependencies, see Hudson (2018), Chapter 36 of this volume.

[9] More precisely, there is a one-to-one correspondence between DMRS and MRS structures, if every predicate as a unique *intrinsic variable*. As observed by Oepen & Lønning (2006), this allows a variable-free semantic representation, by replacing each reference to a variable with a reference to the corresponding predicate.

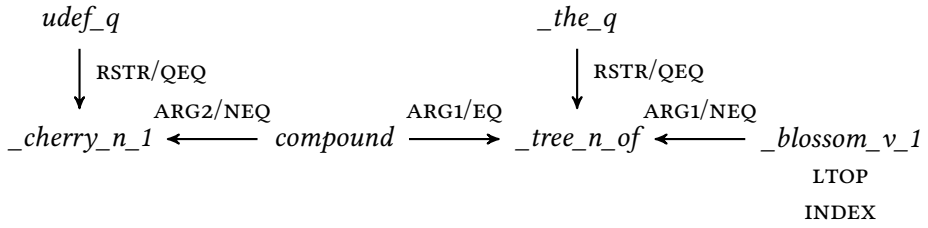


(a) MRS, as a feature structure.

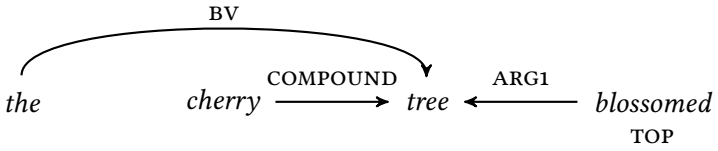
$$\begin{aligned} &\text{INDEX} : e_1 \\ &l_1 : \_the\_q(x_1, h_1, h_2), h_1 \text{ QEQ } l_4 \\ &l_2 : udef\_q(x_2, h_3, h_4), h_3 \text{ QEQ } l_3 \\ &l_3 : \_cherry\_n\_1(x_2) \\ &l_4 : \_tree\_n\_of(x_1), compound(e_2, x_1, x_2) \\ \text{LTOP} \quad &l_5 : \_blossom\_v\_1(e_1, x_1) \end{aligned}$$

(b) MRS, abstractly.

Figure 1: Minimal Recursion Semantics (MRS) representation for the sentence *The cherry tree blossomed*, as produced by the English Resource Grammar (ERG). For simplicity, we have omitted some details, including features such as number and tense, individual constraints (ICONS), and the use of difference lists.



(a) Dependency Minimal Recursion Semantics (DMRS) is as expressive as MRS. Predicates are represented as nodes, while semantic roles and scopal constraints are represented as links.



(b) DELPH-IN MRS Dependencies (DM) are a simplified version of MRS. Nodes are the tokens in the sentence (rather than abstract predicates), and some scopal information is dropped.

Figure 2: MRS-based dependency graphs for the sentence *The cherry tree blossomed*, based on the MRS given in Figure 1.

3.1 CoreGram

3.2 HPSG-inspired broad-coverage parsing

3.3 The DELPH-IN Consortium

The DELPH-IN[10] Consortium was established in 2001 to facilitate the development of large-scale, linguisticaly motivated HPSG grammars for multiple languages in tandem with the software required for developing them and deploying them in pratical applications. At the time that DELPH-IN was founded, the English Resource Grammar (ERG; Flickinger 2000; 2011) had been under development already for 8 years and the Verbmobil project (Wahlster 2000) had also spurred the development of grammars for German (GG; Müller & Kasper 2000; Crysmann 2003) and Japanese (Jacy; Siegel et al. 2016). Project DeepThought (Callmeier et al. 2004) was exploring methodologies for combining deep and shallow processing in pratical applications across multiple languages. This inspired the development of the LinGO Grammar Matrix (Bender et al. 2002), which began as a core

[10] This acronym stands for DEep Linguistic Processing in HPSG INitiative; see www.delph-in.net

grammar, consisting of constraints hypothesized to be cross-linguistically useful, abstracted out of the ERG with reference to Jacy and GG. The goal of the Grammar Matrix is to serve as a starting point for the development of new grammars making it easy to reuse what has been learned in the development of existing grammars. In the years since, it has been extended to include 'libraries' of analyses of cross-linguistically variable phenomena (e.g., Drellishak 2009; Bender et al. 2010).

DELPH-IN provides infrastructure (version control repositories, mailing lists, annual meetings) and an emphasis on open-source distribution of resources. Both of these support the collaboration of a global network of researchers working on interoperable components. These include repositories of linguistic knowledge, that is, both grammars and meta-grammars (including the Matrix and CLIMB (Fokkens 2014)); processing engines that apply that knowledge for parsing and generation (discussed further below); software for supporting the development of grammar documentation (e.g., Hashimoto et al. 2008), software for creating treebanks (Oepen et al. 2004; Packard 2015: see also §4.1.4 below), and parse ranking models trained on them (Toutanova et al. 2005: see also §2.2.2 above), and software for robust processing, i. e. using the knowledge encoded in the grammars to return analyses for sentences even if the grammar deems the ungrammatical (Zhang & Krieger 2011; Buys & Blunsom 2017; Chen et al. 2018).

A key accomplishment of the DELPH-IN Consortium is the standardization of a formalism for the declaration of grammars (Copestake 2002a), a formalism for the semantic representations (**FIXME**), and file formats for the storage and interchannge of grammar outputs (e.g., the forest that results from parsing a forest, as well as the results of treebanking (Oepen 2001; Oepen et al. 2004)). These standards facilitate the development of multiple different parsing and generation engines which can all process the same grammars (including, so far, the LKB (Copestake 2002b), PET (Callmeier 2000), ACE,[11] and Agree (Slayden 2012)), of multiple software systems for processing bulk grammar output (**FIXME-art**; **FIXME-pyDelphin**; Oepen 2001), and of multilingual downstream systems which can be adapted to additional languages by plugging in different grammars.

The DELPH-IN community maintains research interests in both linguistics an practical applications. The focus on linguistics means that DELPH-IN grammarians strive to create grammars which capture linguistic generalizations and model grammaticality. This, in turn, leads to grammars with lower ambiguity than one finds with treebank-trained grammars and, importantly, grammars which

[11] http://sweaglesw.org/linguistics/ace/

produce well-formed strings in generation. The focus on practical applications leads to several kinds of research goals: (1) robustness measures such as techniques for handling unknown words and extra-grammatical mark-up in text (e.g., Wikipedia pages) and strategies for processing inputs that are ungrammatical (at least according to the grammar); (2) performance innovations, such as supertagging or ubertagging to speech up processing times and pre-processors that provide partial information about constituent boundaries to narrow the search space; (3) the integration of external components such as morphological analyzers, named entity recognizers; (4) parse and realization ranking; and (5) work towards making sure that the semantic representations can serve as a suitable interface for external components. From a linguistic point of view, these efforts are also valuable. First, the broader/more robust the coverage of a grammar is, the more linguistic phenomena it can be used to explore. Second, external constraints on the form of semantic representations provide useful guide points in the development of semantic analyses.

4 Deployment of HPSG resources

4.1 Language documentation and linguistic hypothesis testing

Deployment for linguistic goals.

4.1.1 CoreGram

4.1.2 Grammar Matrix

As noted in Section 3.3, the LinGO Grammar Matrix Bender et al. (2002; 2010) was initially developed in the context of Project DeepThought with the goal of speeding up the development of DELPH-IN-style grammars for additional languages. It consists of a shared core grammar and a series of 'libraries' of analyses for cross-lingusitically variable phenomena. Both of these constitute linguistic hypotheses: The constraints in the constraints in the core grammar are hypothesized to be cross-linguistically useful. However, in the course of developing grammars based on the Matrix for specific languages, it is not uncommon to find reasons to refine the core grammar. The libraries, in turn, are intended to cover the attested range of variation for the phenomena they model. Languages that are not covered by the analyses in the libraries provide evidence that they need to be extended to refined.

Grammar Matrix grammar development is less tightly coordinated than that of CoreGram (see §4.1.1): in the typical use case, grammar developers start from the Grammar Matrix, but with their own independent copy of the Matrix core grammar. This impedes somewhat the ability of the Matrix to adapt to the needs of various languages (unless grammar developers report back to the Matrix developers). On the other, hand the Matrix libraries represent an additional kind of linguistic hypothesis testing: each library on its own represents one linguistic phenomenon, but the libraries must be interoperable with each other. This is cross-linguistic analogue of how monolingual implemented grammars allow linguists to ensure that analyses of different phenomena are interoperable (Müller 1999; Bender 2008): the Grammar Matrix customization system allows its developers to test cross-linguistic libraries of analyses for interactions with other phenomena (Bender et al. 2011; Bender 2016). Without computational support — i. e. a computer keeping track of the constraints that make up each analysis, compiling them into specific grammars, and testing those grammars against test suites — this problem space would be too complex for exploration.

4.1.3 AGGREGATION

In many ways, the most urgent need for computational support for linguistic hypothesis testing is the description of endangered languages. Implemented grammars can be used to process transcribed but unglossed text in order to find relevant examples more quickly, both of phenomena that have already been analyzed and of phenomena that are as yet not well-understood.[12] Furthemore, treebanks constructed from implemented grammars can be tremendously valuable additions to language documentation (see §4.1.4 below). However, the process of building an implemented grammar is time-consuming, even with the start provided by a multilingual grammar engineering project like CoreGram or the Grammar Matrix.

This is the motivation for the AGGREGATION project. AGGREGATION starts from two observations: (1) Descriptive linguists produce extremely rich annotations on data in the form of interlinear glossed text (IGT); and (2) the Grammar Matrix's libraries are accessed through a customization system which elicits a grammar specification in the form of a series of choices describing either high-level typological properties or specific constraints on lexical classes and lexical rules. The goal of AGGREGATION is to automatically produce such grammar

[12] This methodology of using an implemented grammar as a sieve to sift the interesting examples out of corpora is demonstrated for English by Baldwin et al. (2004).

specifications on the basis of information encoded in IGT, to be used by the Grammar Matrix customization system to produce language-particular grammars.

* Morphotactics * High-level properties (case system, word order) * End-to-end prototype for one language merging those all

4.1.4 Derived resources: Redwoods-style treebanks

Treebank search

Language documentation

4.2 Downstream applications

In this section, we discuss the use of HPSG grammars for practical tasks. There is a large number of applications, and we focus on several important applications here. Within the DELPH-IN community, a regularly updated list of applications is available online.[13]

4.2.1 Education

Precise syntatic analyses can be useful in language teaching, in order to automatically identify errors and give feedback to the student. In order to model common mistakes, a grammar can be extended with so-called "mal-rules". A mal-rule is like a normal rule, in that it licenses a construction, and can be treated the same during parsing – however, given a parse, the presence of mal-rules indicates that the student needs to be given feedback (Bender et al. 2004; Flickinger & Yu 2013; Morgado da Costa et al. 2016). A large scale system implementing this kind of computer-aided teaching has been developed by the Education Program for Gifted Youth at Stanford University, using the ERG (Suppes et al. 2014). This system has reached tens of thousands of elementary and middle school children, and has been found to improve the school results of underachieving children.

Another way to use a precision grammar is to automatically produce teaching materials. Given a semantic representation, a grammar can generate one or more sentences. Flickinger (2017) uses the ERG to produce practice exercises for a student learning first-order logic. For each exercise, the student is presented with a natural language sentence, and is supposed to write down the corresponding first-order logical form. By using a grammar, the system can produce syntactically varied questions, and automatically evaluate the student's answer.

[13] http://moin.delph-in.net/DelphinApplications

4.2.2 NLP tasks

HPSG grammars have been used in a range of tasks in Natural Language Processing (NLP).

Information extraction. Schäfer et al. (2011). Reiplinger et al. (2012).

Summarization. Fang et al. (2016)

Machine translation. Parse, transfer, generate. (Oepen et al. 2007; Bond et al. 2011). Goodman (2018). Statistical without transfer, Horvat (2017).

Distributional semantics. Emerson (2018).

4.2.3 Data for machine learning

In section 4.2.2, we described how HPSG grammars can be used to tackle a number of NLP tasks. Another use of HPSG grammars in NLP is to generate data, on which a statistical system can be trained.

Training deep learning systems – semantic parsing – skip over HPSG, go straight to semantic representations. (Oepen et al. 2014; 2015). Buys & Blunsom (2017). Chen et al. (2018).

Evaluation of deep learning systems – ShapeWorld – use a grammar to produce annotations. Kuhnle & Copestake (2018).

4.3 Other?

Alpino?

Enju?

Babel

5 Linguistic Insights

- Ambiguity
- Long-tail phenomena (raising and control?)
- Scaling up (thematic roles)
- CLIMB methodology

6 Summary

Abbreviations

Acknowledgements

References

Baldwin, Timothy, John Beavers, Emily M. Bender, Dan Flickinger, Ara Kim & Stephan Oepen. 2004. Beauty and the beast: what running a broad-coverage precision grammar over the BNC taught us about the grammar — and the corpus. Paper presented at the International Conference on Linguistic Evidence: Empirical, Theoretical, and Computational Perspectives, Tübingen, Germany.

Bangalore, Srinivas & Aravind K. Joshi. 1999. Supertagging: an approach to almost parsing. *Computational Linguistics* 25(2). 237–265. http://aclweb.org/anthology/J99-2004.

Bender, Emily M. 2008. Grammar engineering for linguistic hypothesis testing. In Nicholas Gaylord et. al. (ed.), *Proceedings of the Texas Linguistics Society X Conference*, 16–36. Stanford CA: CSLI Publications ONLINE.

Bender, Emily M. 2016. Linguistic typology in natural language processing. *Linguistic Typology* 20. 645–660.

Bender, Emily M., Scott Drellishak, Antske Fokkens, Laurie Poulson & Safiyyah Saleem. 2010. Grammar customization. *Research on Language & Computation*. 1–50. http://dx.doi.org/10.1007/s11168-010-9070-1. 10.1007/s11168-010-9070-1.

Bender, Emily M., Dan Flickinger & Stephan Oepen. 2011. Grammar engineering and linguistic hypothesis testing: Computational support for complexity in syntactic analysis. In E.M. Bender & Arnold J.E. (eds.), *Language from a cognitive perspective: Grammar, usage and processing*, 5–29. Stanford, CA: CSLI Publications.

Bender, Emily M., Dan Flickinger, Stephan Oepen, Annemarie Walsh & Timothy Baldwin. 2004. Arboretum. Using a precision grammar for grammar checking in CALL. In *Proceedings of the InSTIL Symposium on NLP and Speech Technologies in Advanced Language Learning Systems*. Venice, Italy.

Bender, Emily M., Daniel P. Flickinger & 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 & Richard Sutcliffe (eds.), *Proceedings of the Workshop on Grammar Engi-*

neering and Evaluation at the 19th International Conference on Computational Linguistics, 8–14. Taipei, Taiwan.

Bond, Francis, Stephan Oepen, Eric Nichols, Dan Flickinger, Erik Velldal & Petter Haugereid. 2011. Deep open-source machine translation. *Machine Translation* 25(2). 87.

Buys, Jan & Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. In *Proceedings of the 55th annual meeting of the association for computational linguistics (ACL), long papers*, 1215–1226. http://aclweb.org/anthology/P17-1112.

Callmeier, Ulrich. 2000. PET: A platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering* 6(1). 99–108. Special Issue on Efficient Processing with HPSG: Methods, Systems, Evaluation.

Callmeier, Ulrich, Andreas Eisele, Ulrich Schäfer & Melanie Siegel. 2004. The deepthought core architecture framework. In *Proceedings of the fourth international conference on language resources and evaluation (lrec'04)*. Lisbon, Portugal: European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2004/pdf/603.pdf.

Carroll, John. 1994. Relating complexity to practical performance in parsing with wide-coverage unification grammars. In James Pustejovsky (ed.), *32th Annual Meeting of the Association for Computational Linguistics. Proceedings of the conference*, 287–294. Las Cruses: Association for Computational Linguistics.

Carter, David. 1997. The TreeBanker. A tool for supervised training of parsed corpora. In *Proceedings of the 1997 ACL workshop on computational environments for grammar development and linguistic engineering (ENVGRAM)*, 9–15. Madrid, Spain. http://aclweb.org/anthology/W97-1502.

Chen, Yufei, Weiwei Sun & Xiaojun Wan. 2018. Accurate SHRG-based semantic parsing. In *Proceedings of the 56th annual meeting of the association for computational linguistics (ACL), long papers*, 408–418. http://aclweb.org/anthology/P18-1038.

Chomsky, Noam. 1993. A Minimalist Program for linguistic theory. In Kenneth Hale & Samuel Jay Keyser (eds.), *The view from building 20: Essays in linguistics in honor of Sylvain Bromberger* (Current Studies in Linguistics 24), 1–52. Cambridge, MA/London: MIT Press.

Clark, Stephen & James R. Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of the 20th international conference on computational linguistics (COLING)*. http://aclweb.org/anthology/C04-1041.

Copestake, Ann. 2002a. Definitions of typed feature structures. In Stephan Oepen, Dan Flickinger, Jun-ichi Tsujii & Hans Uszkoreit (eds.), *Collaborative language engineering*, 227–230. Stanford, CA: CSLI Publications.

Copestake, Ann. 2002b. *Implementing typed feature structure grammars* (CSLI Lecture Notes 110). Stanford, CA: CSLI Publications.

Copestake, Ann. 2009. Slacker semantics: why superficiality, dependency and avoidance of commitment can be the right way to go. In *Proceedings of the 12th conference of the European chapter of the Association for Computational Linguistics (EACL)*, 1–9. http://aclweb.org/anthology/E09-1001.

Copestake, Ann, Guy Emerson, Michael Wayne Goodman, Matic Horvat, Alexander Kuhnle & Ewa Muszyńska. 2016. Resources for building applications with Dependency Minimal Recursion Semantics. In *Proceedings of the 10th international conference on language resources and evaluation (LREC)*, 1240–1247. http://www.lrec-conf.org/proceedings/lrec2016/pdf/634_Paper.pdf.

Copestake, Ann, Daniel P. Flickinger, Carl J. Pollard & Ivan A. Sag. 2005. Minimal Recursion Semantics: An introduction. *Research on Language and Computation* 3(2–3). 281–332. DOI:10.1007/s11168-006-6327-9

Crysmann, Berthold. 2003. On the efficient implementation of German verb placement in HPSG. In *Proceedings of RANLP 2003*, 112–116. Borovets, Bulgaria.

Drellishak, Scott. 2009. *Widespread but not universal: Improving the typological coverage of the Grammar Matrix*. University of Washington Doctoral dissertation.

Dridan, Rebecca. 2013. Ubertagging: joint segmentation and supertagging for English. In *Proceedings of the 2013 conference on empirical methods in natural language processing (EMNLP)*, 1201–1212. http://aclweb.org/anthology/D13-1120.

Emerson, Guy. 2018. *Functional distributional semantics: learning linguistically informed representations from a precisely annotated corpus*. University of Cambridge dissertation. https://www.repository.cam.ac.uk/bitstream/handle/1810/284882/thesis.pdf.

Fang, Yimai, Haoyue Zhu, Ewa Muszyńska, Alexander Kuhnle & Simone Teufel. 2016. A proposition-based abstractive summariser. In *Proceedings of COLING 2016, the 26th international conference on computational linguistics: technical papers*, 567–578. http://aclweb.org/anthology/C16-1055.

Flickinger, Dan, Stephan Oepen & Emily M. Bender. 2017. Sustainable Development and Refinement of Complex Linguistic Annotations at Scale. In Nancy Ide & James Pustejovsky (eds.), *Handbook of Linguistic Annotation*, 353–377. Dordrecht: Springer Netherlands. http://dx.doi.org/10.1007/978-94-024-0881-2_14. DOI:10.1007/978-94-024-0881-2_14

Flickinger, Dan & Jiye Yu. 2013. Toward more precision in correction of grammatical errors. In *Proceedings of the seventeenth conference on computational natural language learning: shared task*, 68–73. http://aclweb.org/anthology/W13-3609.

Flickinger, Daniel. 2017. Generating English paraphrases from logic. In Martijn Wieling, Gosse Bouma & Gertjan van Noord (eds.), *From semantics to dialectometry: festschrift in honour of john nerbonne*, 99–108. Springer.

Flickinger, Daniel P. 1987. *Lexical rules in the hierarchical lexicon*. Stanford University dissertation.

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

Flickinger, Daniel P. 2011. Accuracy vs. robustness in grammar engineering. In Emily M. Bender & Jennifer E. Arnold (eds.), *Language from a cognitive perspective: Grammar, usage, and processing*, 31–50. Stanford, CA: CSLI Publications.

Fokkens, Antske Sibelle. 2014. *Enhancing empirical research for linguistically motivated precision grammars*. Department of Computational Linguistics, Universität des Saarlandes dissertation.

Gazdar, Gerald & Geoffrey K. Pullum. 1985. Computationally relevant properties of natural languages and their grammars. *New Generation Computing* (3). 237–306. Reprinted in Savitch, Bach, Marsh, and Safran-Naveh (eds.), *The Formal Complexity of Natural Language*.

Goodman, Michael Wayne. 2018. *Semantic operations for transfer-based machine translation* dissertation. https://digital.lib.washington.edu/researchworks/bitstream/handle/1773/42432/Goodman_washington_0250E_18405.pdf.

Hashimoto, Chikara, Francis Bond, Takaaki Tanaka & Melanie Siegel. 2008. Semi-automatic documentation of an implemented linguistic grammar augmented with a treebank. *Language Resources and Evaluation* 42(2). 117–126.

Horvat, Matic. 2017. *Hierarchical statistical semantic translation and realization*. University of Cambridge dissertation. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-913.pdf.

Ivanova, Angelina, Stephan Oepen, Lilja Øvrelid & Dan Flickinger. 2012. Who did what to whom?: A contrastive study of syntacto-semantic dependencies. In *Proceedings of the 6th linguistic annotation workshop*, 2–11. http://aclweb.org/anthology/W12-3602.

Johnson, Mark. 1988. *Attribute-value logic and the theory of grammar* (CSLI Lecture Notes 16). Stanford, CA: CSLI Publications.

Joshi, Aravind K. 1987. Introduction to Tree Adjoining Grammar. In Alexis Manaster-Ramer (ed.), *The mathematics of language*, 87–114. Amsterdam: John Benjamins Publishing Co.

Kasami, Tadao, Hiroyuki Seki & Mamoru Fujii. 1989. Generalized context-free grammars and multiple context-free grammars. *Systems and Computers in Japan* 20(7). 43–52.

Kuhnle, Alexander & Ann Copestake. 2018. Deep learning evaluation using deep linguistic processing. In *Proceedings of the workshop on generalization in the age of deep learning*, 17–23. http://aclweb.org/anthology/W18-1003.

Matsuzaki, Takuya, Yusuke Miyao & Jun'ichi Tsujii. 2007. Efficient HPSG parsing with supertagging and CFG-filtering. In *Proceedings of the 20th international joint conference on artificial intelligence*, 1671–1676. https://www.aaai.org/Papers/IJCAI/2007/IJCAI07-270.pdf.

Morgado da Costa, Luis, Francis Bond & Xiaoling He. 2016. Syntactic well-formedness diagnosis and error-based coaching in computer assisted language learning using machine translation. In *Proceedings of the 3rd workshop on natural language processing techniques for educational applications (NLPTEA2016)*, 107–116. http://aclweb.org/anthology/W16-4914.

Müller, Stefan. 1999. *Deutsche Syntax deklarativ: Head-Driven Phrase Structure Grammar für das Deutsche* (Linguistische Arbeiten 394). Tübingen: Max Niemeyer Verlag.

Müller, Stefan. 2015. The CoreGram project: Theoretical linguistics, theory development and verification. *Journal of Language Modelling* 3(1). 21–86. DOI:10.15398/jlm.v3i1.91

Müller, Stefan & Walter Kasper. 2000. HPSG analysis of German. In Wolfgang Wahlster (ed.), *Verbmobil: Foundations of speech-to-speech translation* (Artificial Intelligence), 238
bibrangessep–
*bibrangessep*253. Berlin Heidelberg New York: Springer.

Oepen, Stephan. 2001. *[incr tsdb()] — Competence and performance laboratory. User manual.* Technical Report. Saarbrücken, Germany:

Oepen, Stephan & Daniel P. Flickinger. 1998. Towards systematic grammar profiling: Test suite technology ten years after. *Journal of Computer Speech and Language* 12(4). 411–436. http://www.delph-in.net/itsdb/publications/profiling.ps.gz, accessed 2018-2-25. (Special Issue on Evaluation).

Oepen, Stephan, Daniel P. Flickinger, Kristina Toutanova & Christopher D. Manning. 2004. LinGO Redwoods: A rich and dynamic treebank for HPSG. *Research on Language and Computation* 2(4). 575–596.

Oepen, Stephan, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajic & Zdenka Uresova. 2015. SemEval 2015 task 18: broad-coverage semantic dependency parsing. In *Proceedings of the 9th international*

workshop on semantic evaluation (SemEval 2015), 915–926. http://aclweb.org/anthology/S15-2153.

Oepen, Stephan, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajic, Angelina Ivanova & Yi Zhang. 2014. SemEval 2014 task 8: broad-coverage semantic dependency parsing. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, 63–72. http://aclweb.org/anthology/S14-2008.

Oepen, Stephan & Jan Tore Lønning. 2006. Discriminant-based MRS banking. In *Proceedings of the 5th international conference on language resources and evaluation (LREC)*, 1250–1255. http://www.lrec-conf.org/proceedings/lrec2006/pdf/364_pdf.pdf.

Oepen, Stephan, Erik Velldal, Jan Tore Lønning, Paul Meurer, Victoria Rosén & Dan Flickinger. 2007. Towards hybrid quality-oriented machine translation: On linguistics and probabilities in MT. In *Proceedings of 11th conference on theoretical and methodological issues in machine translation*, 144–153. Skövde, Sweden.

Packard, Woodley. 2015. *Full forest treebanking*. University of Washington MA thesis.

Reiplinger, Melanie, Ulrich Schäfer & Magdalena Wolska. 2012. Extracting glossary sentences from scholarly articles: a comparative evaluation of pattern bootstrapping and deep analysis. In *Proceedings of the acl-2012 special workshop on rediscovering 50 years of discoveries*, 55–65. http://aclweb.org/anthology/W12-3206.

Schabes, Yves, Anne Abeillé & Aravind K. Joshi. 1988. *Parsing strategies with 'lexicalized' grammars: Application to Tree Adjoining Grammars*. Technical Report MS-CIS-88-65. University of Pennsylvania Department of Computer & Information Science.

Schäfer, Ulrich, Bernd Kiefer, Christian Spurk, Jörg Steffen & Rui Wang. 2011. The ACL anthology searchbench. In *Proceedings of the 49th annual meeting of the association for computational linguistics (ACL), systems demonstrations*, 7–13. http://aclweb.org/anthology/P11-4002.

Siegel, Melanie, Emily M. Bender & Francis Bond. 2016. *Jacy: An implemented grammar of Japanese* (CSLI Studies in Computational Linguistics). Stanford, CA: CSLI Publications.

Slayden, Glenn C. 2012. *Array TFS storage for unification grammars*. University of Washington MA thesis.

Steedman, Mark & Jason Baldridge. 2011. Combinatory Categorial Grammar. In Robert D. Borsley & Kersti Börjars (eds.), *Non-transformational syntax: Formal*

and explicit models of grammar: A guide to current models, 181–224. Oxford, UK/Cambridge, MA: Blackwell Publishers Ltd.

Suppes, Patrick, Tie Liang, Elizabeth E Macken & Daniel P Flickinger. 2014. Positive technological and negative pre-test-score effects in a four-year assessment of low socioeconomic status k-8 student learning in computer-based math and language arts courses. *Computers & Education* 71. 23–32.

Toutanova, Kristina, Christopher D. Manning, Dan Flickinger & Stephan Oepen. 2005. Stochastic HPSG parse disambiguation using the Redwoods corpus. *Research on Language & Computation* 3(1). 83–105.

Toutanova, Kristina, Christopher D. Manning, Stuart M. Shieber, Dan Flickinger & Stephan Oepen. 2002. Parse disambiguation for a rich HPSG grammar. In *Proceedings of the first workshop on treebanks and linguistic theories (TLT2002)*. Sozopol, Bulgaria. http://bultreebank.org/wp-content/uploads/2017/05/paper17.pdf.

Wahlster, Wolfgang (ed.). 2000 (Artificial Intelligence). Berlin Heidelberg New York: Springer.

Zhang, Yi & Hans-Ulrich Krieger. 2011. Large-scale corpus-driven pcfg approximation of an hpsg. In *Proceedings of the 12th international conference on parsing technologies*, 198–208.