Chapter 30

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*, 411–420. 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 2.3 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 **FIXME-Hopcroft-and-Ullman?** 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; this means that the formalism can be called *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 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" through the tree, 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. 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.

2.2.2 Parse ranking

For an ambiguous sentence, a grammar gives multiple valid parses. This is widely known, with attachment ambiguities and coordination ambiguities being particularly well-known examples. 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 am-

biguity has become clear. For example, (1) 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, but with increasingly long sentences, such ambiguities stack up very quickly. For (2), the first line of a newspaper article,[5] the 1214 version of the English Resource Grammar (Flickinger 2000; 2011) gives TODO readings.

(1) My favorite town criers speak clearly.

(2) 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.

Where 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 only uses 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 (Oepen et al. 2004). 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 on many features[6] of the parse. From the example in (1), we can see how 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 our attention, we can drastically reduce processing time

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

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

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.

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

[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.

[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.

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 Figure 1. The existence of such dependency graphs has made it easier to use HPSG grammars in a number of practical tasks, as we will discuss in Section 4.2.

...

Figure 1: Comparison of MRS, DMRS, and DM.

2.3 A brief history of HPSG grammar engineering

History: PAGE, VerbMobil, ??

Current platforms:

- LKB/ACE/PET/Agree
- Trale
- Other

3 Development of HPSG resources

- CoreGram
- DELPH-IN consortium
  - ERG
  - Other large-ish grammars
  - Grammar Matrix
- Systems inspired by HPSG:
  - Alpino
  - Enju

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

4.1.3 AGGREGATION

4.1.4 Derived resources: Redwoods-style treebanks

4.2 Downstream applications

Deployment for other tasks. Large number of applications. Focus on several important applications below. See also http://moin.delph-in.net/DelphinApplications

4.2.1 Language teaching

Redbird (McGraw-Hill)

4.2.2 NLP tasks

Information extraction
Summarisation
Machine translation

4.2.3 Data for machine learning

Unlike the previous sections, not providing analyses, but providing data. Not used at test time, only at training time.

Training deep learning systems – semantic parsing – skip over HPSG, go straight to semantic representations

Evaluation of deep learning systems – ShapeWorld – use a grammar to produce annotations

4.3 Other?

Alpino?
Enju?

5 Linguistic Insights

- Ambiguity
- Long-tail phenomena (raising and control?)

- Scaling up (thematic roles)
- CLIMB methodology

6 Summary

Abbreviations

Acknowledgements

References

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

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. 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, 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

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.

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.

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.

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.

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

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 & 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.

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.

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.