

# Implementing HPSG with modular tools for fast compiling and parsing

Graham Wilcock\*

Centre for Computational Linguistics  
University of Manchester Institute  
of Science and Technology  
PO Box 88, Manchester M60 1QD  
United Kingdom  
graham@ccl.umist.ac.uk

Yuji Matsumoto

Graduate School of Information Science  
Nara Institute of Science and Technology  
8916-5 Takayama, Ikoma, Nara 630-01  
Japan  
matsu@is.aist-nara.ac.jp

## Abstract

We describe a modular HPSG implementation, based on a set of tools rather than a single monolithic engine such as ALE. With these tools we can use techniques for much faster compiling and parsing than ALE. We use two-stage grammar compilation with partial execution and a concurrent process implementation of the chart for fast parsing. We compile HPSG lexical rules into Prolog rules used at run-time by the morphological preprocess, thus eliminating lexical rule expansion at compile-time as in ALE. This gives fast lexicon compilation, and also allows further exploitation of lexical rules to eliminate empty categories.

## 1 Introduction

Using the ALE system (Carpenter and Penn, 1994) to develop an English grammar, we found that it was possible to implement successfully almost everything in (Pollard and Sag, 1994), but development was hindered by slow compilation, especially lexicon compilation. By contrast, using the SAX parsing system (Matsumoto et al., 1994) with a DCG-based grammar, both compiling and parsing were much faster than ALE, but without an inheritance-based typed feature system we could not fully implement HPSG theory, and the DCG-based grammar had many rules, many categories, and many arguments per category. We therefore investigated how to combine fast compiling and parsing with HPSG theory.

Instead of using a monolithic engine such as ALE, we prefer a modular approach using different tools for different purposes. The tools are the SAX parsing system (a grammar translator, a morphological preprocess, and a concurrent chart parser), the SGX generation system (a grammar translator and a concurrent chart generator), the ProFIT typed feature system and SICStus Prolog.

---

\*Visiting researcher of Information Systems Product Development Laboratories, Sharp Corporation.

The ProFIT inheritance-based typed feature system (Erbach, 1995) enables us to implement HPSG theory, and has the advantage that it can be combined with any appropriate parser or generator and with SICStus Prolog. We combine ProFIT with the SAX parser and the SGX generator, and use ProFIT with SICStus Prolog to implement lexical rules as run-time inference rules.

We describe our bidirectional framework and our implementation of delayed lexical choice in (Wilcock and Matsumoto, 1996). In this paper we will contrast our implementation primarily with ALE, so we will not discuss generation.

## 2 Two-stage grammar compilation

The grammar is compiled in two stages. In the first stage, the typed feature structures are compiled by ProFIT into Prolog terms, so that relatively slow unification of feature structures is replaced by relatively fast unification of terms. Finite domains such as index agreement are compiled as boolean vectors for fast unification.

In the second stage, the grammar is compiled by the SAX translator, using partial execution to produce efficient code for bottom-up chart parsing. During parsing, instead of building the chart by asserting edges into the Prolog database, the SAX parser implements the chart by creating concurrent *processes*. The terminal and non-terminal symbols of the grammar are realized as processes which communicate via streams to build larger structures. A meta-process monitors the streams and controls the whole parsing process (Matsumoto and Sugimura, 1987). For compiled Prolog systems, this is a highly efficient form of chart parsing, even on sequential machines.

## 3 A finite sort hierarchy

In ALE, every lexical predicate must be declared as an atomic sort. This requires a massive sort hierarchy, and when a new word is added to the lexicon its atomic sort must also be added to the sort hierarchy, thereby requiring the whole system to be recompiled.

With ProFIT, atomic sorts for individual lexical predicates do not need to be declared. By using thematic roles, rather than purely lexical semantic roles as used in (Pollard and Sag, 1994), we only need a finite, extensible set of thematic role declarations in the sort hierarchy. Therefore a new word, with a new lexical predicate and an existing combination of thematic roles, can be added to the lexicon without extension of the sort hierarchy, and only the lexicon needs to be recompiled.

## 4 Lexical rule compilation

ALE applies lexical rules during lexicon compilation, to derive new lexical entries from existing ones. The new lexical entries are collected together with the existing entries, and all of them are compiled to produce a finite collection which is the lexicon. During parsing, each input word is simply looked up in this static lexicon. The problem is that lexicon compilation becomes impossibly slow, as every possible alternative sign for every possible morphological form of every possible word must be generated and added to the list.

In our system, lexical rules are kept as *rules*. HPSG lexical rules are compiled by ProFIT into Prolog inference rules. During lexical lookup, the rules are used deductively to *prove* the existence of a lexical item. We do not have a static, finite lexicon, and we do not require any lexical rule expansion process during compilation. Lexicon compilation is therefore very fast.

During parsing, lexicon lookup is performed by the SAX morphological preprocess, which builds a morpheme lattice to be passed to the SAX parser for syntactic parsing. The Prolog lexical inference rules are used during this preprocess.

## 5 Eliminating empty categories

ALE supports empty categories, always inserting every empty category at every node in the chart, whether the actual input requires them or not. SAX does not support them (though they could be implemented by a meta-process if necessary). Like (Sag, 1995), we believe that theories which posit no invisible entities are *a priori* preferable, and that the traces and null relativizers of (Pollard and Sag, 1994) are undesirable.

We eliminate traces by a Complement Extraction Lexical Rule. In ALE this would massively increase the size of the static lexicon, with a corresponding increase in lexicon compilation time. In our implementation, lexicon compilation is not affected. During parsing, where transitive substantives appear in the actual input, the SAX morphological preprocess uses the rule to infer the existence of both the slashed and unslashed signs, and inserts them both into the morpheme lattice.

We are currently investigating elimination of null relativizers as proposed by (Sag, 1995).

## 6 Unrestricted lexical rules

ALE lexical rules can only derive a new lexical entry from an existing entry. This is awkward, e.g. attributive and predicative adjectives cannot be derived from each other because there are different exceptions in both classes. Our lexical rules are not restricted in this way, so they can derive a lexical entry from data which is not part of the lexicon. They can therefore conditionally derive either alternative adjective entry, or both, from a basic list of adjectives with explicit exceptions.

We can extend lexical rules in other ways. For example, we could develop rules to access an existing source of lexical information and construct HPSG signs from whatever form of information was available.

## Acknowledgements

The first author would like to thank Mr Hitoshi Suzuki (Sharp Corporation) and Prof Jun-ichi Tsujii (UMIST) for making this work possible. We also thank Dr Kristiina Jokinen (NAIST) and the anonymous reviewers for valuable comments.

## References

- Bob Carpenter and Gerald Penn, 1994. *The Attribute Logic Engine User's Guide, Version 2.0*. Carnegie Mellon University.
- Gregor Erbach. 1995. ProFIT: Prolog with Features, Inheritance, and Templates. In *Seventh Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics.
- Yuji Matsumoto and Ryoichi Sugimura. 1987. A parsing system based on logic programming. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, volume 2, pages 671–4.
- Yuji Matsumoto, Yasuharu Den, and Takehito Utsuro, 1994. *Koubun kaiseki shisutemu SAX, shiyou setsumeisho (Parsing system SAX Manual) version 2.1*. Nara Institute of Science and Technology.
- Carl Pollard and Ivan Sag. 1994. *Head-driven Phrase Structure Grammar*. University Press, Chicago.
- Ivan Sag. 1995. English relative clause constructions. Unpublished manuscript.
- Graham Wilcock and Yuji Matsumoto. 1996. Reversible delayed lexical choice in a bidirectional framework. In *16th International Conference on Computational Linguistics*. Association for Computational Linguistics.