# ROBUSTNESS and EFFICIENCY in AGFL

**Erik Oltmans**
Dept. of Computer Science
University of Twente
Enschede, The Netherlands
oltmans@cs.utwente.nl

**Caspar Derksen, Kees Koster**
Dept. of Computer Science
University of Nijmegen
Nijmegen, The Netherlands
caspard@cs.kun.nl, kees@cs.kun.nl

## Robust Parsing

The Condorcet project is an information retrieval project carried out at the Knowledge-Based Systems Group at the University of Twente, The Netherlands, and funded by the Dutch Technology Foundation (STW). The objective of Condorcet is to build a prototype information retrieval system that will ultimately be able to handle titles + abstracts of 30,000 documents. The system is mostly concerned with *indexing* of documents within two specific domains: mechanical properties of engineering ceramics and epilepsy. For an elaborate discussion of the system, the reader is referred to the Condorcet Annual Reports, available through the Condorcet home page: `http://wwwis.cs.utwente.nl:8080/kbs/condorcet/`. Assignment of indexing concepts to documents is done by means of a syntactic parser (CONSTRUCTER) and a semantic analyzer. As the documents contain numerous misspelled words, ill-formed constructions and new words, not all input can be parsed according to the underlying grammar. Since achieving robustness by constantly adding rules to the grammar is laborious and may be even impossible, an extra robustness device is needed in order to parse all documents. However, approaches that can be found in the literature such as [1], imply in most cases adjustments of the underlying parsing strategy: problematic fragments are skipped or completely deleted. Within Condorcet however, a knowledge-based approach is preferred, as handling ill-formed sentences with robust grammars rules would allow for more flexibility and expandability. By employing a knowledge-based approach, it is possible to develop a solution to the robustness problem that is not based on ad hoc approaches, but that acknowledges the linguistic nature of the problem. It also allows for more control over the quality of the output: in cases of an unsatisfactory result, the robust rules can be easily adapted so as to tune the ultimate parser. Robustness is thus accomplished by the linguist instead of the programmer, and by means of writing specific grammar rules rather than writing computer programs. The AGFL formalism, developed at the University of Nijmegen (`http://www.cs.kun.nl/agfl`) is used because it allows for elegantly expressing a knowledge-based approach to robustness and because of its efficient implementation.

## Affix Grammars over a Finite Lattice

Affix grammars over finite lattices (AGFLs) are a restricted form of affix grammars in which Context Free production rules are extended with affixes (or features) for expressing agreement between parts of speech, [2]. Like in PROLOG with DCGs, these are passed as parameters to the rules of the grammar. Each affix expression at a parameter position denotes a set of features associated with the nonterminal in a certain context. In rewriting, all occurrences of a certain affix in a rule obtain the same value, being any subset of the (finite) domain of the affix, but not the empty set. *Feature ambiguity* is distinguished from *structural ambiguity* by decorating parse trees with *sets* of affix values, rather than with single valued affixes. As a result, multiple parses with the same structure are combined. Although Tomita or Earley are the preferred parsing algorithms for CF grammars, in parsing according to a grammar with features, the (exponential) determination of features may well overwhelm the parsing process, so that the efficiency of the parsing by itself is not the main issue. Therefore AGFL is implemented using Recursive Backup Parsing (RBP), a generalization of Recursive Descent Parsing to ambiguous grammars. The main drawback of RBP is that parsers exhibit exponential behavior, but by applying the optimizations described below, large-scale parsers show quasi-linear behavior for sentences of reasonable length; their exponential character becomes apparent for only a fraction of the sentences. Furthermore, top-down backtrack parsers are easy to extend with affixes, allow top-down filtering of values, [3], and parser

generation for RBP is a straightforward transcription from the grammar. Fast parser generation makes AGFL suitable for development and prototyping and finally, RB parsers are small and use memory efficiently.

Factoring out common left-factors of alternatives for the same nonterminal, possibly in combination with repeated unfolding of nonterminals, can be used for improving the time complexity of RBP, as it eliminates duplicate parsing effort. Top-down RBP by itself is not capable of dealing with left-recursion, but it is possible to remove left-recursion from a grammar, e.g. by *goal-corner transformation.* Applying look-ahead is another classical technique for improving the run-time behavior of parsers, although the time complexity of the parsing algorithm remains within the same complexity class.

Memoization is a technique for optimizing repeated computations of a function by remembering a table of values for certain arguments. However, since tuples of parameters do not have a total ordering, it is hard to search a table in which calls of parameterized nonterminals are memoized. Therefore, *negative memoization* is used, preventing the repetition of unsuccessful computations. In the mapping $memo : pos \times nonterminal \rightarrow bool \times bool$ the first boolean (*known*) indicates whether this nonterminal has been tried at this position, and the second boolean (*blocked*) indicates whether the nonterminal will fail at this position, and should be blocked. In applying this technique, each call of a nonterminal at a position where it has not been called before is first *generalized* to a call with all parameters generalized to their full domains; if the rule fails it is known that the rule will fail for all possible combinations of affixes and this fact is recorded for future use. If the call succeeds, the parameters are restricted to their original values. This optimization improves the efficiency of RBP dramatically.

## Robust Parsing in Practice

For achieving knowledge-based robustness without ambiguity, the most important features in AGFL are the semi-predicate `PENALTY` and the *commit operator.* They can be used to indicate a preference of clear syntactic forms over doubtful ones. By using these features, alternatives can be ordered in such a way that *graceful degradation* is ensured, providing as much of an accurate analysis as possible: instead of failing in returning a highly structured analysis, a reliable shallow analysis is returned. This is accomplished by means of an ordered set of alternatives, in which the level of underspecification in order to describe problematic parts of the input is decreasing. If the parsing process is aborted, or the parser reaches a pre-defined time limit, the current parse (which is the best parse for that particular moment) is returned. However, if the input is not problematic, only the parse according to the last alternative will be returned, because of penalties in previous alternatives. If the input is extremely complex and the time-limit is exceeded, minimal information will at least be returned. This strategy yields a rule set that is divided into a *core grammar* and a *peripheral grammar* and it allows for full control over the quality of the output, depending on the grammar writer's intentions. The linguist only needs to express ideas with respect to robustness; computational complexity, efficiency or speed are irrelevant from the grammar writer's point of view.

Within the Condorcet parser, a robustness device has been developed according to this knowledge-based strategy. This has resulted in a parser that is capable of analyzing 100% of its input, providing information on the syntactic subject, the finite verb and all noun phrases and prepositional phrases. For now, it seems that this is useful enough but if more information were needed, the rules could simply be adjusted. CONSTRUCTER is a parser for English, generated from an AGFL grammar that consists of 68 rewrite rules extended with 14 robust rules. We report on a test involving a corpus of 84 documents (30 documents on epilepsy, 30 documents on ceramic materials and a randomly selected corpus of 24 in order to test the parser on unseen material). Most sentences in the corpus are very complex: the average sentence length is 23.8 words per sentence. 60 documents were parsed in 11 seconds, which is equal to parsing 40 sentences per second or 950 words per second (on a SUN/SPARC station). The shortest parse took 0.01 seconds, the longest 0.46 seconds. 61% of the sentences were parsed according to the core grammar and 39% were parsed robustly. The parser needed relatively more robustness (37% vs. 63%) when analyzing unseen texts, which is not surprising.

# References


[1] T. Strzalkowski. Robust Text Processing in Automated Information Retrieval. In *Proceedings of the Fourth ACL Conference on Applied Natural Language Processing.* Association for Computational Linguistics, 1994.

[2] C.H.A. Koster. Affix Grammars for Natural Languages. In H. Alblas and B. Melichar, editors, *Attribute Grammars, Applications and Systems*, volume 545 of *Lecture Notes in Computer Science*, pages 469–484. Springer-Verlag, Berlin, Germany, 1992.

[3] M.J. Nederhof. *Linguistic Parsing and Program Transformations.* PhD thesis, University of Nijmegen, The Netherlands, 1994.