Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Research Report
RR-98-04

A Bag of Useful Techniques for Efficient and Robust Parsing

Bernd Kiefer, Hans-Ulrich Krieger

December 1998

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210
E-Mail: info@dfki.uni-kl.de

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341
E-Mail: info@dfki.de

WWW: http://www.dfki.de

Deutsches Forschungszentrum für Künstliche Intelligenz

DFKI GmbH

German Research Center for Artificial Intelligence

Founded in 1988, DFKI today is one of the largest nonprofit contract research institutes in the field of innovative software technology based on Artificial Intelligence (AI) methods. DFKI is focusing on the complete cycle of innovation — from world-class basic research and technology development through leading-edge demonstrators and prototypes to product functions and commercialization.

Based in Kaiserslautern and Saarbrücken, the German Research Center for Artificial Intelligence ranks among the important “Centers of Excellence” worldwide.

An important element of DFKI’s mission is to move innovations as quickly as possible from the lab into the marketplace. Only by maintaining research projects at the forefront of science can DFKI have the strength to meet its technology transfer goals.

DFKI has about 115 full-time employees, including 95 research scientists with advanced degrees. There are also around 120 part-time research assistants.

Revenues for DFKI were about 24 million DM in 1997, half from government contract work and half from commercial clients. The annual increase in contracts from commercial clients was greater than 37% during the last three years.

At DFKI, all work is organized in the form of clearly focused research or development projects with planned deliverables, various milestones, and a duration from several months up to three years.

DFKI benefits from interaction with the faculty of the Universities of Saarbrücken and Kaiserslautern and in turn provides opportunities for research and Ph.D. thesis supervision to students from these universities, which have an outstanding reputation in Computer Science.

The key directors of DFKI are Prof. Wolfgang Wahlster (CEO) and Dr. Walter Olthoff (CFO).

DFKI’s six research departments are directed by internationally recognized research scientists:

- Information Management and Document Analysis (Director: Prof. A. Dengel)
- Intelligent Visualization and Simulation Systems (Director: Prof. H. Hagen)
- Deduction and Multiagent Systems (Director: Prof. J. Siekmann)
- Programming Systems (Director: Prof. G. Smolka)
- Language Technology (Director: Prof. H. Uszkoreit)
- Intelligent User Interfaces (Director: Prof. W. Wahlster)

In this series, DFKI publishes research reports, technical memos, documents (eg. workshop proceedings), and final project reports. The aim is to make new results, ideas, and software available as quickly as possible.

Prof. Wolfgang Wahlster
Director

A Bag of Useful Techniques for Efficient and Robust Parsing

Bernd Kiefer, Hans-Ulrich Krieger

DFKI-RR-98-04

This work has been supported by a grant from The Federal Ministry of Education, Science, Research, and Technology (FKZ ITW-01 IV 701 V0).

ISSN 0946-008X

A Bag of Useful Techniques for Efficient and Robust Parsing

Bernd Kiefer and Hans-Ulrich Krieger
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
{kiefer,krieger}@dfki.de

Abstract

This paper describes new and improved techniques which help a unification-based parser to process input efficiently and robustly. We show that combining these methods leads to a speed-up in parsing time of more than an order of magnitude. The methods are correct in the sense that none of them rule out legal rule applications.

1 Introduction

This paper describes great number of useful techniques which help a unification-based parser to process input efficiently and robustly. We present several new methods and report improvements to existing techniques. The methods are correct in the sense that none of them rule out legal rule applications—neither will we investigate statistical nor approximative methods. It is worth noting that these methods are also parser-independent and neutral w.r.t. a given grammar theory/formalism.

How can we gain reasonable efficiency in parsing with wide-coverage grammars and lexicons with several thousand complex entries? Our belief is that there is no single method which achieves this goal alone. Instead, we have to develop and use a set of "cheap" filters which are correct in the above sense. As we indicate in section 10, combining these methods leads to a speed-up in parsing time (and reduction of space consumption) of more than an order of magnitude.

We have implemented our methods as extensions to a HPSG grammar development environment (Uszkoreit et al. 1994) which employs a sophisticated typed formalism (Krieger and Schäfer 1995) and an advanced agenda-based bottom-up chart parser (Kiefer and Scherf 1996). A special runtime version of this system is currently used in VERBMOBIL as the primary deep analysis component.[1]

In the next three sections, we report on transformations we have applied to the knowledge base (grammar/lexicon) and on modifications in the core formalism

[1] VERBMOBIL (Wahlster 1993) deals with the translation of spontaneously spoken dialogues, where only a minor part consists of "sentences" in a linguistic sense. Current languages are English, German, and Japanese.

(unifier, type system). In section 5–8, we describe how a given parser can be extended to quickly filter out possible rule applications before doing "expensive" unification. After that, we show how to compute best partial analyses in order to gain robustness. Finally, we present empirical results to demonstrate the efficiency gain and discuss our plans for work in the near future. Within the different sections, we refer to three corpora we have used to measure the effects of our methods. The reference corpora for English, German, and Japanese consist of 100–130 sample sentences.

2 Precompiling the Lexicon

Lexical entries in the development system are small templates that are loaded and expanded on demand by the typed feature structure system. Thereafter, all lexical rules are applied to the expanded feature structures. The results of these two computations form the input of the analysis stage.
In order to save space and time in the runtime system, the expansion and the application of lexical rules is now done offline. In addition, parts of the feature structure are deleted, since they are only needed to restrict the application of lexical rules (see also section 7 for a similar approach). For each stem, all results are stored in compact form as one compiled LISP file, which allows to access and load a requested entry rapidly with almost no restriction on the size of the lexicon. Although load time is small (see table 1), the most frequently used entries are cached in main memory, reducing effort in the lexicon stage to a minimum.
Morphological information is continued to be computed online, due to the significant increase of entries (a factor of 10 to 20 for German), which is not justifiable considering the minimal computation time for this operation (only a few msec per word).

	German	English	Japanese
number of stems	4269	3754	1875
total space (MB)	44.1	40.5	10.1
space per stem (KB)	10.3	10.8	5.4
entries per stem	6	2.2	2.1
load time p. stem (msec)	25.8	29.5	7.5

Figure 1: Space and time requirements.

3 Moving to Conjunctive Unification

It has often been argued that explicit disjunctions are inevitable for specifying grammatical information adequately. We take a liberal stance here and note that the original HPSG grammar development system mentioned in the introduction even allows for distributed (or named) disjunctions in order to link particular

choices (Dörre and Eisele 1990). The popular assumption then is that these disjunctions are not only a nice descriptive mean but also allow more efficient processing, since they obviate the need to expand to disjunctive normal form (DNF). Unfortunately, we cannot validate this in VERBMOBIL for the large English, German, and Japanese HPSG grammars written by different people. We note here that the English grammar has been converted manually into a disjunction-free form, allowing to optimize the performance of the resulting grammar.
Let us note a few things here to make our statement clear: (i) moving to DNF but using the same unification engine (usually) does not lead to faster unification; (ii) the ratio of the number of lexical entries/rules in the original grammar and the DNFed grammar clearly depends on the 'style' of the grammar author, the particular grammar theory, the number of disjunction alternatives, etc.; (iii) a lot of disjunctive information can be easily put into the type hierarchy in case one uses a typed grammar formalism; (iv) the context management for (distributed) disjunctions causes an enormous overhead when compared to simple conjunctive unification; (v) several lazy copying algorithms have been proposed for conjunctive unification which clearly outclass a naïve implementation.
Our hope was that even if we move automatically to DNF, the general amount of disjunctions which cannot be put into the type hierarchy is so small that the number of DNFed rules and lexical entries grows relatively slowly. Exactly this was true for the German and Japanese grammars: we got 1.4–3× more rules/lexical entries, but by moving to a sophisticated conjunctive unifier, we obtained an overall speed-up of 2–5.[2]

4 Precompiling Type Unification

After having changed the unification engine, we saw that the type unification part became a big factor: nearly 50% of the overall unification and copying time was now due to the computation of the greatest lower bounds (GLBs). Although we have used bit vectors in the grammar development system to efficiently compute them online, the offline computation of the GLBs is of course superior.
The feasibility of the latter method depends on the number of types $\mathcal{T}$ of a grammar. The English grammar employs 6000 types which results in 36,000,000 possible GLBs. Our experiments have shown, however, that only 0.5%–2% of type unifications were successful and only these GLBs need to be entered into the GLB table. We have used a hash table (and not a sparsely filled array) to efficiently store and access the GLBs, employing a hash key function of arity 2. Accessing an arbitrary GLB takes 0.008 msec, compared to 15 msec of 'expensive' bit vector computation à la Aït-Kaci et al. 1989 which also produces a lot of memory garbage.[3] In order to access a unique GLB, we must require that the type hierarchy is a lower semilattice (or bounded complete partial order). This is

[2]We use Rob Malouf's (of CSLI, Stanford) improved version of Hideto Tomabechi's quasi-destructive graph unification algorithm (Tomabechi 1991). Compared with our original distributed disjunction unifier, the unification time decreased by a factor of 10–20, when using conjunctive structures only.

[3]Clearly, the bit-vector computed GLBs can also be cached, exactly what we did in $\mathcal{TDL}$; see Krieger 1995.

often not the case, but this deficiency can be overcome either by computing the missing types or by making the online table lookup more complex.
A naïve implementation of the offline computation (*compute the GLBs for* $\mathcal{T} \times \mathcal{T}$) only works for small grammars. Since type unification is a commutative operation ($glb(s,t) = glb(t,s);\ s,t \in \mathcal{T}$), we can improve the algorithm by computing only $glb(s,t)$. A second improvement is due to the following fact: if the GLB of s and t is bottom, we do not have to compute the GLBs of subtypes for both s and t, since they guarantee to fail (an efficient implementation of this observation must enforce a topological ordering on $\mathcal{T}$). The final improvement separates the type hierarchy into a family of maximal subsumption-incomparable sets, guaranteeing that only types from the same set might unify. Even with these improvements, the GLB computation of a specific grammar took 42 CPU hours, due to the special 'topology' of the type hierarchy. Using the offline GLBs (together with every method described here), we obtained a parsing speed-up of 1.5, compared to the bit vector computation. By applying this method alone, the savings are clearly more substantial, since a GLB failure is the only way to forbid a rule application in the parser (we obtain a speed-up factor of more than 24 and 10 times less space consumption).

5 Precompiling Rule Filters

The aim of the methods described in this and the next section is to avoid failing unifications by applying cheap 'filters' (i.e., methods that are cheaper than unification). The first filter we want to describe is a rule application filter. We use this method for quite a while, and it has proven both efficient and easy to employ.
Our rule application filter is a function that takes two rules and an argument position and returns a boolean value that specifies if the second rule can be unified into the given argument position of the first rule.
The conjunctive grammars have between 20 and 120 unary and binary rule schemata. Since all rule schemata in our system bear a unique number, this filter can be realized as a three dimensional boolean array, thus access costs are minimized. These filters are computed offline in less than one minute and typically rule out 50% to 60% of the failing unifications during parsing, saving about 45% of the parsing time.

6 Avoiding Unification through Quick Checks

Our second filter utilizes the fact that at certain points in the feature structure, unification fails more often than at others.[4] Since all substructures are typed, type clashes can occur at every point in the feature structure. Checking the most frequent failure points first and only then unifying saves a great amount of processing time.

[4]This filter is an implementation of a method used by John Carroll in the LKB system. Thanks to Stephan Oepen who provided the hint.

The method works as follows. First, there is an offline stage, where a modified unification engine of the original grammar development system is used that does not terminate unification after the first failure, but instead records all failures in a global data structure. Using this modified system, a corpus is parsed to compute the n paths with the highest failure counts. Exactly these paths will be used later for filtering.
When an active chart item (i.e., a rule schema or a partly instantiated rule schema) and a passive chart item are combined, the feature structure of the passive item is unified into the substructure of the active item that corresponds to the argument to be filled. For every parser item, we now build a vector of n types by computing the type values of the substructures under the previously determined paths (i.e., we establish pointers to the substructures to gain constant access time). If the item is an active item, we start at the feature structure corresponding to the next argument. If the item is passive (i.e., fully instantiated), the computation starts at the root of its feature structure. The type corresponding to the most frequently failing path is put into the first position of the vector, the second-best into the second position, and so on. When an active item shall be further instantiated with a passive item, the types in the two vectors are checked for compatibility element by element. Only if this test succeeds for all vector positions, unification is executed.
Clearly, when considering the number of paths used for this method, there is a tradeoff between time savings by filtered unifications and the effort to create the vectors and comparing them. Since type unification is computationally cheap as described in section 4, the savings are substantial. At the moment, we use 13 to 22 paths for quick-check filtering. In connection with the rule application filter (cf. section 5), the filter rate varies from over 94% to over 99%, hence almost all failing unifications can be avoided. The parsing time relative to the system with only rule application filtering is reduced by approx. 75%. According to Dan Flickinger (p.c.), the filter rate in the LKB system which applies the quick-check method alone, is up to 80%, using 40 paths however.

7 Reducing Feature Structure Size via Restrictors

The 'category' information that is attached to each chart item of the parser consists of a single feature structure. Thus a rule is implemented by a feature structure where the daughters have to be unified into predetermined substructures. Although this implementation is along the lines of HPSG, it has the drawback that the tree structure that is already present in the chart items is duplicated in the feature structures.
Since HPSG requires all relevant information to be contained in the SYNSEM feature of the mother structure, the unnecessary daughters only increase the size of the overall feature structure without constraining the search space. Due to the *Locality Principle* of HPSG (Pollard and Sag 1987, p. 145ff), they can therefore be legally removed in fully instantiated items. The situation is different for active chart items since daughters can affect their siblings.

To be independent from a certain grammatical theory or implementation, we use *restrictors* similar to Shieber 1985 as a flexible and easy-to-use specification to perform this deletion. A *positive restrictor* is an automaton describing the paths in a feature structure that will remain after *restriction* (the deletion operation), whereas a *negative restrictor* specifies the parts to be deleted. Both kinds of restrictors can be used in our system.
In addition to the removal of the tree structure, the grammar writer can specify the restrictor further to remove features that are only used locally and do not play a role in further derivation. This reduction in size results in a speed-up in unification itself, but also in copying and memory management.
As already mentioned in section 2, there exists a second restrictor to get rid of unnecessary parts of the lexical entries after lexicon processing. The speed gain using the restrictors in parsing ranges from 30% for German to 45% for English.

8 Limiting the Number of Initial Chart Items

Since the number of lexical entries per stem has a direct impact on the number of parsing hypotheses (in the worst case leads to an exponential increase), it would be a good idea to have a cheap mechanism at hand that helps to limit these initial items. The technique we have implemented is based on the following observation: in order to contribute to a reading, certain items (concrete lexical entries, but also classes of entries) require the existence of other items such that the non-existence of one allows a safe deletion of the other (and vice versa). Note that such a method operates in a much larger context (in fact, the whole chart) as a local rule application filter or the quick-check method.
In German, for instance, prefix verbs require the right separable prefixes to be present in the chart, but also a potential prefix requires its prefix verb (the same holds for English verbs, requiring certain particles, as in *come along*, *come across*, etc.). The method works as follows. In a preprocessing step, we first separate the chart items which encode prefix verbs from those items which represent separable prefixes. Since both specify the morphological form of the prefix, a set-exclusive-or operation yields exactly the items which can be safely deleted from the chart.
Let us give some examples to see the usefulness of this method. In the sentence *Ich komme morgen* (*I (will) come tomorrow*), *komme* maps onto 97 lexical entries—remember, *komme* might encode prefix verbs such as *ankommen* (*arrive*), *zurückkommen* (*come back*), etc. although here, none of the prefix verb readings are valid, since the prefix is missing. Using the above method, only 8 of 97 lexical entries will remain in the chart. The sentence *Ich komme morgen an* (*I (will) arrive tomorrow*) results in 8+7 entries for *komme* (8 entries for the *come* reading and 7 entries for the *arrive* reading of *komme*) and 3 prepositional readings plus 1 prefix entry for *an*. But in *Der Mann wartet an der Tür* (*The man is waiting at the door*), only the three prepositional readings for *an* come into play, since no prefix verb *anwartet* exists.
The parsing time for the second example goes down by a factor of 2.4; overall savings w.r.t. our reference corpus is 17% of the parsing time (i.e., speed-up factor of 1.2).

9 Computing Best Partial Analyses

Not only in speech parsing, we often find deficient, ungrammatical, or spontaneous input s.t. a traditional parser will not come up with an analysis. Contrary to that, our approach focuses on partial analyses which are combined in a later stage to form total analyses without giving up the correctness of the overall deep grammar. But what is a partial analysis? Obviously a (sub)tree licensed by the grammar which covers a continuous part of the input (i.e., a passive or inactive parser edge). Not every passive edge is a good candidate since otherwise we would end up with thousands of candidates. Our approach lies between these two extremes: computing a sequence of adjacent 'best' partial analyses which span the whole input. The idea here is to view the set of passive edges as a directed graph and to compute shortest paths w.r.t. a user-defined estimation function.
Since this graph is acyclic and topologically sorted, we have chosen the DAG-shortest-path algorithm (Cormen et al. 1990) which runs in $\Theta(V + E)$. We have modified this algorithm to cope with the needs we have encountered in speech parsing: (i) one can use several start and end vertices (e.g., in case of n-best chains or wordgraphs); (ii) all best shortest paths are returned (i.e., we obtain a shortest-path subgraph); (iii) estimation and selection of the best edges is done incrementally in case of n-best chains and time-critical applications (i.e., only new passive edges entered into the chart are estimated and perhaps selected). This approach has one important property: even if certain parts of the input have not undergone any rule application, there are still lexical edges which help to form a best path through the passive edges.
Let us give an example to see what the estimation function on edges (= trees) might look like (this estimation is actually used in the German grammar):

- n-ary tree ($n > 1$) with utterance status (e.g., NPs, PPs): value 1
- lexical items: value 2
- otherwise: value ∞

This approach does not always favor paths with longest edges as the example in figure 2 shows—instead it prefers paths containing no lexical edges (where this is possible) and there might be several such paths having the same cost. Longest (sub)paths, however, can be obtained by employing an exponential estimation function.

10 Conclusion and Further Work

The collection of methods described in this paper has enabled us to unite deep linguistic analysis with speech processing. The overall speed-up compared to the grammar development system is about a factor of 10 up to 25, depending on the specific grammar. We note here that in the original system this measurement was conducted by even using the rule application filter method, the restrictor method, and a caching of bit-vector computed GLBs. Without these methods, the speed gain is even more than two orders of magnitude.

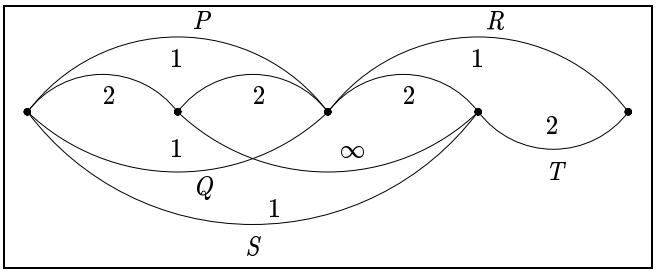

Figure 2: Computing best partial analyses. Note that the paths PR and QR are chosen, but not ST, although S is the longest edge.

We now present some absolute timings to give an idea of the current systems' performance. For a German test set of 153 sentences with an average sentence length of 7.3, we get an overall parse time of 420 cpu seconds when computing all results.[5] On average, there are 6.8 analyses, the first computed in 0.73 sec (including lexicon access and morphological analysis) and the search space is exhausted after 2.7 sec.
We believe to have shown that the combination of algorithmic methods together with some discipline in grammar writing can lead to a practical high performance analysis system even with large general grammars for different languages.
There is, however, scope for improvements. Removing unnecessary lexical items can be generalized to other cases, and also to English and Japanese phenomena. A detailed investigation of the quick-check method and its interaction with the rule application filter is also planned for the near future. Since almost all failing unifications are avoided through the use of filtering techniques, we will now focus on methods which reduce the number of (successful) chart items that do not lead to an analysis, for instance, context-free or regular approximations of the HPSG grammars (e.g., Nederhof 1997).

Acknowledgments

The research described in this paper has greatly benefited from a very fruitful collaboration with the HPSG group of CSLI at Stanford University. This cooperation is part of the deep linguistic processing effort within the BMBF project VERBMOBIL. The combination of methods achieved by the two groups has made it possible to meet the ambitious efficiency goals for the processing of spoken input in VERBMOBIL. Especially, we want to thank Rob Malouf for providing us his unification engine and integrating it into our core grammar development system. The parser filter described in section 6 is an implementation of a method devised by John Carroll for the LKB system. Special thanks are due to Stefan Müller for discussing the topic of German prefix verbs. Thanks also to Dan Flickinger who provided us with several similar English phenomena. Mark-Jan Nederhof's comments have helped us a lot. Finally, we want to thank Nicolas Nicolov for

[5]The computations were made using a 300MHz SUN Ultrasparc 2 with Solaris 2.5. The whole system is programmed in Franz Allegro Common Lisp

reading a version of this paper. This research was supported by the German Federal Ministry for Education, Science, Research and Technology under grant no. 01 IV 701 V0.

References

Aït-Kaci, H., R. Boyer, P. Lincoln, and R. Nasr. 1989. Efficient Implementation of Lattice Operations. *ACM Transactions on Programming Languages and Systems* 11(1):115–146.

Cormen, T. H., C. E. Leiserson, and R. L. Rivest. 1990. *Introduction to Algorithms.* Cambridge, MA: MIT Press.

Dörre, J., and A. Eisele. 1990. Feature Logic with Disjunctive Unification. In *Proceedings of the 13th International Conference on Computational Linguistics, COLING-90*, Vol. 3, 100–105.

Kiefer, B., and O. Scherf. 1996. Gimme more HQ Parsers. The Generic Parser Class of DISCO. Technical report, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany.

Krieger, H.-U. 1995. *TDL—A Type Description Language for Constraint-Based Grammars. Foundations, Implementation, and Applications.* PhD thesis, Universität des Saarlandes, Department of Computer Science, September.

Krieger, H.-U., and U. Schäfer. 1995. Efficient Parameterizable Type Expansion for Typed Feature Formalisms. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI-95*, 1428–1434. Also available as DFKI Research Report RR-95-18.

Nederhof, M. J. 1997. Regular Approximations of CFLs: A Grammatical View. In *Proceedings of the 5th International Workshop on Parsing Technologies, IWPT'97*, 159–170.

Pollard, C., and I. A. Sag. 1987. *Information-Based Syntax and Semantics. Vol. I: Fundamentals.* CSLI Lecture Notes, Number 13. Stanford: Center for the Study of Language and Information.

Shieber, S. M. 1985. Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics, ACL-85*, 145–152.

Tomabechi, H. 1991. Quasi-Destructive Graph Unification. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, 315–322.

Uszkoreit, H., R. Backofen, S. Busemann, A. K. Diagne, E. A. Hinkelman, W. Kasper, B. Kiefer, H.-U. Krieger, K. Netter, G. Neumann, S. Oepen, and S. P. Spackman. 1994. DISCO—An HPSG-based NLP System and its Application for Appointment Scheduling. In *Proceedings of COLING-94*, 436–440. A version of this paper is available as DFKI Research Report RR-94-38.

Wahlster, W. 1993. VERBMOBIL—Translation of Face-to-Face Dialogs. Research Report RR-93-34, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany. Also in Proc. MT Summit IV, 127–135, Kobe, Japan, July 1993.

A Bag of Useful Techniques for Efficient and Robust Parsing

Bernd Kiefer, Hans-Ulrich Krieger

RR-98-04
Research Repor