Measure For Measure: Parser Cross-Fertilization*

— Towards Increased Component Comparability and Exchange —

Stephan Oepen, Ulrich Callmeier

Computational Linguistics, Saarland University,

Postfach 15 11 50, 66041 Saarbrücken, Germany

e-mail: {oe | uc}@coli.uni-sb.de

Abstract

Over the past few years significant progress was accomplished in efficient processing with wide-coverage HPSG grammars. HPSG-based parsing systems are now available that can process medium-complexity sentences (of ten to twenty words, say) in average parse times equivalent to real (i.e. human reading) time. A large number of engineering improvements in current HPSG systems were achieved through collaboration of multiple research centers and mutual exchange of experience, encoding techniques, algorithms, and even pieces of software. This article presents an approach to grammar and system engineering, termed *competence & performance profiling*, that makes systematic experimentation and the precise empirical study of system properties a focal point in development. Adapting the profiling metaphor familiar from software engineering to constraint-based grammars and parsers, enables developers to maintain an accurate record of system evolution, identify grammar and system deficiencies quickly, and compare to earlier versions or between different systems. We discuss a number of exemplary problems that motivate the experimental approach, and apply the empirical methodology in a fairly detailed discussion of what was achieved during a development period of three years. Given the collaborative nature in setup, the empirical results we present involve research and achievements of a large group of people.

Dramatis Personæ

> [...] *we view the discovery of parsing strategies as a largely experimental process of incremental optimization.* (Erbach, 1991a)

> [...] *the study and optimisation of unification-based parsing must rely on empirical data until complexity theory can more accurately predict the practical behaviour of such parsers.* (Carroll, 1994)

Early 1994, research groups at Saarbrücken[1] (Uszkoreit et al., 1994) and CSLI Stanford[2] (Copestake, 1992; Flickinger & Sag, 1998) started to collaborate on the development of large-scale HPSG grammars, suitable grammar engineering platforms, and efficient processors. While both sites had worked on HPSG implementation before, the joint effort has greatly increased productivity, resulted in a mutual exchange of knowledge and technology, and helped building a collection of grammar development environments, several highly engineered parsers (Kiefer, Krieger, Carroll, & Malouf, 1999) and an efficient generator (Carroll, Copestake, Flickinger, & Poznanski, 1999). Recently, the HPSG group at Tokyo University[3] (Torisawa & Tsujii, 1996) has entered the stage and now supplies additional expertise on (abstract-machine-based) compilation of typed feature structures, Japanese HPSG, and grammar approximation techniques.

*This play builds heavily on results obtained through collaboration between a largish group of people at several sites; Act 5 presents a list of individuals who have contributed to the development providing the background for our current discussion. Technical details and empirical assessments of individual techniques will be documented in a forthcoming Special Issue of the *Journal of Natural Language Engineering* (Flickinger, Oepen, Uszkoreit, & Tsujii, 2000).

[1] See 'http://www.dfki.de/lt/' and 'http://www.coli.uni-sb.de/' for information on the DFKI Language Technology Laboratory and the Computational Linguistics Department at Saarland University, respectively.

[2] The 'http://lingo.stanford.edu/' web pages list HPSG-related projects and people involved at CSLI, and also provide an on-line demonstration of the LKB system and LinGO grammar.

[3] Information on the Tokyo laboratory, founded and managed by Professor Jun'ichi Tsujii, can be found at 'http://www.is.s.u-tokyo.ac.uk/'.

John Carroll (editor) 2000: International Workshop on Parsing Technology. Trento, Italy.

Although the individual systems often supply extra functionality, the groups have converged on a common descriptive formalism — a conservative blend of Carpenter (1992), Copestake (1992), and Krieger and Schäfer (1994) — that allows grammars[4] a to be processed by five different platforms. The LinGO grammar, a multi-purpose, broad-coverage grammar of English developed at CSLI and among the largest HPSG implementations currently available, serves as a common reference for all three groups (while of course the sites continue development of additional grammars for English, German, and Japanese). With one hundred thousand lines of source, roughly eight thousand types, an average feature structure size of some three hundred nodes, twenty seven lexical and thirty seven phrase structure rules, and some six thousand lexical (stem) entries, the LinGO grammar presents a fine challenge for processing systems. While scaling the systems to this rich set of constraints and improving processing and constraint resolution algorithms, the groups have regularly exchanged benchmarking results, in particular at the level of individual components, and discussed benefits and disadvantages of particular encodings and algorithms. Precise comparison was found essential in this process and has facilitated a degree of cross-fertilization that proved beneficial for all participants.

Act 1 below introduces the profiling methodology, supporting tools, and the sets of common reference data and benchmarking metrics that were used among the groups. By way of example, the profiling metaphor is then applied in Act 2 to a choice of engineering issues that (currently) can only be approached empirically. Act 3 introduces the PET platform (Callmeier, 2000) as another actor in the experimental setup; PET synthesizes a variety of techniques from the individual systems in a fresh, modular, and highly parameterizable reimplementation. Finally, on the basis of empirical data obtained with PET, Act 4 provides a detailed comparison of competence and performance profiles obtained in October 1996 with the current development status (as of October 1999).

1 Competence & Performance Profiling

In system development and optimization, subtle algorithmic and implementational decisions often have a significant impact on system performance, so monitoring system evolution very closely is crucial. Developers should be enabled to obtain a precise record of the status of the system at any given point; also, comparison with earlier results, between various parameter settings, and across platforms should be automated and integrated with the regular development cycle. System performance, however, cannot be adequately characterized merely by measurements of overall processing time (and perhaps memory usage). Properties of (i) individual modules (in a classical setup, especially the unifier, type system, and parser), (ii) the grammar being used, and (iii) the input presented to the system all interact in complex ways. In order to obtain an analytical understanding of strengths and weaknesses of a particular configuration, finer-grained records are required. By the same token, developer intuition and isolated case studies are often insufficient, since in practice, people who have worked on a particular system or grammar for years still find that an intuitive prediction of system behaviour can be incomplete or plainly wrong.

Although most grammar development environments supply facilities to batch-process a test corpus and record the results produced by the system, these are typically restricted to processing a flat, unstructured input file (listing test sentences, one per line) and outputting a small number of processing results into a log file.[5] In total, we note a striking methodological and technological deficit in the area

[4] In the HPSG universe (and accordingly our present play) the term 'grammar' is typically used holistically, referring to the linguistic system comprised of (at least) the type hierarchy, lexicon, and rule apparatus.

[5] (Meta-)Systems like PLEUK (Calder, 1993) and HDRUG (van Noord & Bouma, 1997) that facilitate the exploration of multiple descriptive formalisms and processing strategies come with slightly more sophisticated benchmarking facilities and visualization tools. However, they still largely operate on monolithic, unannotated input data sets, restrict

readings	number of complete analyses obtained (when applicable, after unpacking)
filter	percentage of parser actions predicted to fail (rule filter plus 'quick check')
etasks	number of attempts to instantiate an argument position in a rule
stasks	number of successful instantiations of argument positions in rules
aedges	number of active edges built by the parser (where appropriate)
pedges	number of passive edges built by the parser (typically in all-paths search)
unifications	number of top-level calls into the feature structure unification routine
copies	number of top-level feature structure copies made
tcpu	amount of cpu time (in milliseconds) spent in processing
space	amount of dynamic memory allocated during processing (in bytes)

Table 1: Some of the parameters making up a competence & performance profile.

of precise and systematic, let alone comparable, assessment of grammar and system behaviour.

Oepen and Flickinger (1998) propose a methodology, termed *grammar profiling*, that builds on structured and annotated collections of test and reference data (traditionally known as *test suites*). The *competence & performance profiling* approach we advocate in this play can be viewed as a generalization of this methodology — in line with the experimental paradigm suggested by, among others, Erbach (1991a) and Carroll (1994). A *competence & performance profile* is defined as a rich, precise, and structured snapshot of system behaviour at a given development point. The production, maintenance, and inspection of profiles is supported by a specialized software package (called [incr tsdb()][6]) that supplies a uniform data model, an application program interface to the grammar-based processing components, and graphical facilities for profile analysis and comparison. Profiles are stored in a relational database which accumulates a precise record of system evolution, and which serves as the basis for flexible report generation, visualization, and data analysis via basic descriptive statistics. All tables and figures used in this play were generated using [incr tsdb()].

The profiling environment defines a common set of descriptive metrics which aim both for in-depth precision and also for sufficient generality across a variety of processing systems. Most parameters are optional, though analysis potential may be restricted for partial profiles. Roughly, profile contents can be classified into information on (i) the processing *environment* (grammar, platform, versions, parameter settings and others), (ii) grammatical *coverage* (number of analyses, derivation and parse trees per reading, corresponding semantic formulae), (iii) *ambiguity* measures (lexical items retrieved, number of active and passive edges, where applicable, both globally and per result), (iv) *resource consumption* (various timings, memory allocation), and indicators of (v) *parser* and *unifier* throughput. Excluding relations and attributes that encode annotations on the input data, the current *competence & performance* database schema includes some one hundred attributes in five relations. Table 1 summarizes some of the profiling parameters as they are relevant to the drama to come.

While the current [incr tsdb()] data model has already been successfully adapted to six different parsing systems (with another two pending; see Act 5), it remains to be seen how well it scales to the description of a larger variety of processing regimes. And although absolute numbers must be viewed *cum grano salis*, the common metric has greatly increased comparability and data exchange among the groups mentioned above, and has in some cases also helped to identify unexpected sources of performance variation. For example, we have found that two Sun UltraSparc servers (at different sites) with identical hardware configuration (down to the level of cpu revision) and OS release reproducibly

accounting of system results to a small number of parameters (e.g. number of analyses, overall processing time, memory consumption, possibly the total number of chart edges), and only offer a limited, predefined choice of analysis views.

[6]See '`http://www.coli.uni-sb.de/itsdb/`' for the (draft) [incr tsdb()] user manual, pronunciation rules, and instructions on obtaining and installing the package.

Set	Aggregate	total items ♯	word string ϕ	lexical entries ϕ	total results ♯	parser analyses ϕ	passive edges ϕ	fs size ϕ
'tsnlp'	wellformed	1574	4·96	13·8	945	2·00	86	273
	illformed	2775	4·50	11·5	409	1·83	39	257
'csli'	wellformed	918	6·45	15·3	732	2·16	115	302
	illformed	375	6·11	14·9	85	2·31	84	298
'aged'	wellformed	96	8·41	23·1	72	7·00	292	315
'blend'	wellformed	1910	11·13	32·1	1008	51·39	1181	336
	illformed	142	11·05	34·2	24	20·33	611	317

Table 2: Reference data sets used in comparison and benchmarking with the LinGO grammar.

exhibit a performance difference of around ten per cent. This appears to be caused by different installed sets of vendor-supplied operating system patches. Also, average cpu load and availability of main memory have been observed to have a noticeable effect on cpu time measurements; therefore, all data reported in this play, was collected in an (artificial, in some sense) environment in which sufficient cpu and memory resources were guaranteed throughout each complete test run.

The [incr tsdb()] package includes a number of test suites and development corpora for English, German, and French (and has facilities for user-level import of additional test data). For benchmarking purposes with the LinGO grammar four test sets were chosen: (i) the English TSNLP test suite (Oepen, Netter, & Klein, 1997), (ii) the CSLI test suite derived from the original Hewlett-Packard data (Flickinger, Nerbonne, Sag, & Wasow, 1987), (iii) a small collection of transcribed scheduling dialogue utterances collected in the VerbMobil context, and (iv) a larger extract from recent VerbMobil corpora that was selected pseudo-randomly to achieve a balanced distribution of one hundred samples for each input length below twenty words. Some salient properties of these test sets are summarized in Table 2.[7] Looking at the degrees of lexical (i.e. the ratio between columns five and four), global (column seven), and local (approximated in column eight by the number of passive edges created in pure bottom-up parsing) ambiguity, the three test sets range from very short and unambiguous to mildly long and highly ambiguous. The *'blend'* test set is a good indicator of maximal input complexity that the available parsers can currently process (in plausible amounts of time and memory). Contrasting columns six and three (i.e. items accepted by the grammar vs. total numbers of well- or ill-formed items) provides a measure of grammatical coverage and overgeneration, respectively.

2 Strong Empiricism: A Few Examples

A fundamental measure in comparing two different versions or configurations of one system as well as for contrasting two distinct systems is correctness and equivalence of results. No matter what unification algorithm or parsing strategy is chosen, parameters like the numbers of lexical items retrieved per input word, total analyses found, passive edges derived (in non-predictive bottom-up parsing, at least) and others should only vary when the grammar itself is changed. Therefore, regular regression testing is required. In debugging and experimentation practice, we have found that minor divergences in results are often hard to identify; using an experimental parsing strategy, for example, over- and undergeneration can even out for the number of readings and even the accounting of passive edges. Hence, assuring an exact match in results (for a given test set) is a non-trivial task.

[7] While wellformedness and item length are properties of the test data proper, the indicators for average ambiguity and feature structure (fs) size were obtained using the current release version of the LinGO grammar, frozen in August 1999. Here and in the tables to come the symbol '♯' indicates absolute numbers, while 'ϕ' denotes average values.

The [incr tsdb()] package eases comparison of results on a per-item basis, using an approach similar to Un*x `diff(1)`, but generalized for structured data sets. By selection of a set of parameters for intersection (and optionally a comparison predicate), the user interface allows to browse the subset of items that fail to match in the selected properties. One dimension that we found especially useful in intersecting profiles is on the derivation trees (a bracketed structure labeled with rule names and identifiers of lexical items) associated with each parser analysis. Once a set of missing or extra derivations (representing under- or overgeneration, respectively) between two profiles is identified, they can be fed back into the defective parser as a request to try and reconstruct each derivation. Reconstruction of derivation trees, in a sense, amounts to fully deterministic parsing, and enables the processor to record where the failure occurs that caused undergeneration in the first place; conversely, when dealing with overgeneration, reconstruction in the correct parser can be requested to identify the missing constraint(s). While these techniques illustrate basic debugging facilities that the profiling and experimentation environment provides, the following two scenes discuss algorithmic issues in parser design and tuning that can only be addressed empirically.

2.1 Hyper-Active Parsing

The two established development platforms, the LKB (CSLI Stanford) and PAGE (DFKI Saarbrücken) systems, have undergone homogenization of approaches and even individual modules (the conjunctive PAGE unifier, for instance, was developed by Rob Malouf at CSLI Stanford) for quite a while.[8] Until recently, however, the parsing regimes deployed in the two systems were significantly different. Both parsers use quasi-destructive unification, are purely bottom-up, chart-based, perform no ambiguity packing, and can be operated in exhaustive (all paths) or agenda-driven best-first search modes; before any unification is attempted, both parsers apply the same set of pre-unification filters, viz. a test against a rule compatibility table (Kiefer et al., 1999), and the 'quick check' partial unification test (Malouf, Carroll, & Copestake, 2000). The LKB passive chart parser (in exhaustive mode) uses a breadth-first CKY-like algorithm; it processes the input string strictly from left to right, constructing all admissible complete constituents whose right vertex is at the current input position before moving on to the next lexical item. Attempts at rule application are made from right to left. All and only complete constituents found (passive edges) are entered in the chart. The active PAGE parser, on the other hand, uses a variant of the algorithm described by Erbach (1991b). It operates bidirectionally, both in processing the input string and instantiating rules; crucially, the *key* daughter (see Scene 2.2 below) of each rule is analyzed first, before the other daughter(s) are instantiated.

But while the LKB and the PAGE developers both assumed the strategy chosen in their own system was the best-suited for parsing with large feature structures (as exemplified by the LinGO grammar), the choices are motivated by conflicting desiderata. Not storing active edges (as in the passive LKB parser) reduces the amount of feature structure copying but requires frequent recomputation of partially instantiated rules, in that the unification of a daughter constituent with the rightmost argument position of a rule is performed as many times as the rule is applied to left-adjacent sequences of candidate chart edges. Creating active edges that add partial results to the chart, on the other hand, requires that more feature structure copies are made, which in turn avoids the necessity of redoing unifications. Given the effectiveness of the pre-unification filters it is likely that for some active edges no attempts to extend them with adjacent inactive edges will ever be executed, so that the copy

[8] Still, the two systems are by no means merely two instantiations of the same concept, and continue to focus on different application contexts. While the LKB is primarily used for grammar development and generation (in an AAC basic research project), recent PAGE develoment has emphasized efficient and robust parsing methods with speech recognizer output (in application to VerbMobil).

Set	Parser	filter %	etasks ϕ	stasks ϕ	unifs ϕ	copies ϕ	tcpu ϕ (s)	space ϕ (kb)
'*csli*'	*passive*	94·2	658	555	663	114	0·38	2329
	active	95·8	283	180	288	180	0·31	2432
	hyper-active	95·8	283	180	354	114	0·28	1686
'*aged*'	*passive*	94·2	1843	1604	1845	293	1·14	5692
	active	96·1	716	452	718	452	0·93	5449
	hyper-active	96·1	716	452	928	293	0·71	3830
'*blend*'	*passive*	93·6	9209	7968	9214	1074	5·87	16757
	active	96·0	2849	1580	2853	1580	3·42	13767
	hyper-active	96·0	2849	1580	4156	1074	3·31	10393

(generated by [incr tsdb()] at 3-nov-1999 (19:08 h)

Table 3: Contrasting parser performance: passive, active, and hyper-active in the LKB.

associated with the active edge was wasted effort. Profiling the two parsers individually showed that overall performance is roughly equivalent (with a minimal lead for the passive LKB parser in both time and space). While the passive parser executes far more parser tasks (i.e. unifications), it creates significantly fewer copies — as should be expected from what is known about the differences in parsing strategy. Hence, from a superficial comparison of parser throughput one could conclude that the passive parser successfully trades unifications for copies, and that both basic parsing regimes perform equally well with respect to the LinGO grammar.

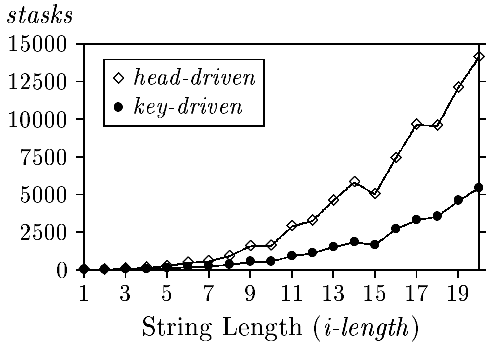
To obtain fully comparable results, the algorithm used in PAGE was imported into the LKB, which serves as the (single) experimentation environment for the remainder of this scene. The direct comparison is shown in Table 3 for three of the standard test sets. The re-implementation of the active parser in the LKB, in fact, performs slightly better than the passive version and does not allocate very much more space. On the '*aged*' test set, the active parser even achieves a modest reduction in memory consumption which most likely reflects the larger proportion of extra unifications compared to the savings in copies (columns five and six) for this test set. Having profiled the two traditional parsing strategies and dissected each empirically, it now seems natural to synthesize a new algorithm that combines the advantages of both strategies (i.e. reduced unification *and* reduced copying). The following algorithm, termed 'hyper-active' by Oepen and Carroll (2000), achieves this goal:

- use the bottom-up, bidirectional, key-driven control strategy of the active parser;
- when an 'active' edge is derived, store this partial analysis in the chart but do *not* copy the associated feature structure;[9]
- when an 'active' edge is extended (combined with a passive edge), recompute the intermediate feature structure from the original rule and already-instantiated daughter(s);
- only copy feature structures for complete passive edges; partial analyses are represented in the chart but the unification(s) that derived each partial analysis are redone on-demand.

Essentially, storing 'active' (or, in a sense, hyper-active) edges without creating expensive feature structure copies enables the parser to perform a key-driven search effectively, and at the same time avoids over-copying for partial analyses; additional unifications are traded for the copies that were avoided only where hyper-active edges are actually extended in later processing.[10]

[9]Although the intermediate feature structure is not copied, it is used to compute the 'quick-check' vector for the next argument position to be filled; as was seen already, this information is sufficient to filter the majority (i.e. up to ninety five per cent) of subsequent operations on the 'active' edge.

[10]There is an additional element — termed 'excursion' — to the algorithm proposed in Oepen and Carroll (2000)



PET (cheap)	head-driven	key-driven	Δ
filter (%)	93·2	95·5	—
etasks	8441	2995	64·5 %
stasks	4170	1465	64·9 %
tcpu (s)	1·47	0·64	56·4 %
space (kb)	7861	6472	17·7 %

Figure 1: Effects of head- vs. key-driven rule instantiation on parser work load (PET on '*blend*').

Table 3 confirms that hyper-active parsing combines the desirable properties of both basic algorithms: the number of copies made is exactly the same as for the passive parser, while the number of unifications is only moderately higher than for the active parser (due to on-demand recomputation of intermediate structures). Accordingly, average parse times are reduced by twenty six ('*csli*') and thirty seven ('*aged*') per cent, while memory consumption drops by twenty seven and thirty two per cent, respectively. Applying the three parsers to the much more challenging '*blend*' test set, reveals that the greater search space poses a severe problem for the passive parser, and limits the relative advantages of the hyper-active over the plain active strategy somewhat: while in the latter comparison the amount of copying is reduced by one third in hyper-active parsing, the number of unifications increases by thirty per cent at the same time (but see the discussion of rule instantiation below). Still, the hyper-active algorithm greatly reduces memory consumption, which by virtue of lower garbage collection times (not included in *tcpu* values) results in a significant overall speed-up. Compared to the original LKB passive parser, hyper-active parsing achieves a time and space reduction of forty three and thirty eight per cent, respectively. Thorough profiling and eclectic engineering have resulted in an improved parsing algorithm that is now used standardly in both the LKB and PAGE; for the German and Japanese VerbMobil grammars in PAGE, the observed benefits of hyper-active parsing were broadly confirmed.

2.2 Rule Instantiation Strategies

Head-driven approaches to parsing have been explored successfully with lexicalized grammars like HPSG (see van Noord, 1997, for a recent overview) because, basically, they can avoid proliferation of partial rule instantiations (i.e. active edges in a chart parser) with rules that contain very unspecific argument positions. Many authors either implicitly (Kay, 1989) or explicitly (Bouma & van Noord, 1993) assume the *linguistic head* to be the argument position that the parser should instantiate first. However, the right choice of argument position in each rule, such that it best constrains rule applicability (with respect to all categories derived by the grammar) cannot be determined analytically. Though the selection is likely to be related to the amount and specificity of information encoded for each argument, for some rules a single feature value (e.g. the [WH +] constraint on the non-head daughter in one of the instantiations of the filler – head schema used in LinGO) can be most important. For terminological clarity, PAGE introduces the term *key* daughter to refer to the argument position in each rule that is the best discriminator with respect to other categories that the grammar derives; at the same time, the notion of *key-driven* parsing emphasizes the observation that for individual rules in a particular grammar a non-(linguistic)head daughter may be a better candidate.

that aims to take advantage of the feature structure associated with an active edge while it is still valid (i.e. within the same unification generation). Put simply, the hyper-active parser is allowed to deviate slightly from the control strategy governed by the agenda, to try and combine the active edge with one suitable passive edge immediately.

Rule Name	head	key	aedges left→right	aedges right→left	pedges	ratio
HEAD – COMPLEMENT	left	left	84,396	1,404,652	264,137	3·13
SPECIFIER – HEAD	right	right	582,736	108,450	14,849	0·14
SUBJECT – HEAD	right	left	48,464	364,846	300,561	6·20
HEAD – MARKER	left	left	1,494	1,404,652	106,349	71·18
HEAD – ADJUNCT (*scopal*)	left	right	856,419	12,946	73,975	5·71
ADJUNCT – HEAD (*isective*)	right	left	34,482	1,260,660	37,343	1·08
ADJUNCT – HEAD (*scopal*)	right	left	11,177	1,260,660	119,513	10·69
FILLER – HEAD (*wh, subj*)	right	left	162	147,636	546	3·37

Table 4: Head and key positions and distribution of active vs. passive edges for selected rules.

Figure 1 compares parser performance (using the PET parser; see below) for a rule instantiation strategy that always fills the (linguistic) head daughter first (labelled '*head-driven*') with a variant that uses an idiosyncratically chosen key daughter for each rule (termed '*key-driven*'; see below for key selection). The data shows that the number of executed (*etasks*) as well as the number of successful (*stasks*) parser actions increase far more drastically with respect to input length in the head-driven setup (on the '*blend*' test suite, truncated above 20 words due to sparse data). Since parser tasks are directly correlated to overall parser performance, the key-driven strategy on average reduces parsing time by more than a factor of two. Clearly, for the LinGO grammar at least, linguistic headedness is not a good indicator for rule instantiation. Thus, the choice of good parsing keys for a particular grammar is an entirely empirical issue. Key daughters, in the current setup, are stipulated by the grammar engineer(s) as annotations to grammar rules; in choosing the key positions, the grammarian builds on knowledge about the grammar and observations from parsing test data. The [incr tsdb()] performance profiling tools can help in this choice since they allow the accounting of active and passive edges to be broken down by individual grammar rules (as they were instantiated in building edges). Inspecting the ratio of edges built per rule, for any given choice of parsing keys, can then help to identify rules that generate an unnecessary number of active edges. Thus, in the experimental approach to grammar and system optimization the effects of different key selections can be analyzed precisely and compared to earlier results.[11] Table 4 shows the head and key positions together with the differences in the number of active edges derived (in strict left to right vs. right to left rule instantiation modes) for a subset of binary grammar rules in LinGO. For the majority of head – argument structures (with the notable exception of the subject – head rule) the linguistic head corresponds to the key daughter, in adjunction and (most) filler – head constructions we see the reverse image; for some rules, choosing the head daughter as the key can result in an increase of active edges close to two orders of magnitude.

Inspecting edge proliferation by individual rules reveals another property of the particular grammar: the ratio of passive to active edges (column seven in Table 4, using the key-driven values for *aedges*) varies drastically. The specifier – head rule, for example, licenses a large number of active edges but, on average, only one out of seven active edges can be completed to yield a passive edge. The head – marker rule, on the other hand, on average generates seventy one passive edges from just one active edge. While the former should certainly benefit from hyper-active parsing, this seems very unlikely for the latter; Scene 2.1 above suggests that no more than three unifications should be traded for one copy in the LKB. Therefore, it seems plausible to apply the hyper-active parsing regime selectively to rules with a *pedges* to *aedges* ratio below a certain threshold t. We plan to explore this technique in

[11]For a given test corpus, the optimal set of key daughters can be determined (semi- or fully automatically) by comparing results for unidirectional left to right to pure right to left rule instantiation; the optimal key position for each rule is the one that generates the smallest number of active items.

a series of experiments, evaluating parser performance in relation to varied values for t.

3 PET — Synthesizing Current Best Practice

PET is a platform to build processing systems based on the descriptive formalism represented by the LinGO grammar. It aims to make experimentation with constraint-based parsers easy, including comparison of existing techniques and evaluating new approaches. Thus, flexibility and extendibility are primary design objectives. Both desiderata are achieved by a tool box approach — PET provides an extendible set of configurable building blocks, that can be combined and configured in different ways to instantiate a concrete processing system. The set of building blocks includes objects like *chart*, *agenda*, *grammar*, *type hierarchy*, and *typed feature structure*. Using the available objects, a simple bottom-up chart parser, for instance, can be realized in a few lines of code.

Alternative implementations of a certain object may be available to allow comparison of different approaches to one aspect of processing in a common context. For example, the current PET environment provides a choice of destructive, semi-destructive, and quasi-destructive implementations of the *typed feature structure* object (viz. the algorithms proposed by Wroblewski (1987), Ciortuz (2000), Tomabechi (1991), and Malouf et al. (2000); experimentation with additional algorithms and fixed-arity encodings is under development). In this setup properties of various graph unification algorithms and feature structure representations can be compared among each other and in interaction with different processing regimes.

In a parser called cheap, PET implements all relevant techniques from Kiefer et al. (1999) (i.e. conjunctive-only unification, rule filters, quick-check, restrictors), as well as techniques originally developed in other systems (e.g. key-driven parsing from PAGE, caching type unification and hyper-active parsing from the LKB, and partial expansion from CHIC). Re-implementation and strict modularization often resulted in improved representations and algorithmic refinement; since individual modules can be specialized for a particular task, the overhead often found in monolithic implementations (like slots in internal data structures, say, that are only required in a certain configuration) could be reduced.

Efficient memory management and minimizing memory consumption was another important consideration in the development of PET. Experience with Lisp-based systems suggests that memory throughput is one of the main bottlenecks when processing large grammars. In fact, one observes a close correlation between the amount of dynamically allocated memory and processing time, indicating much time is spent moving data, rather than in actual computation. Using builtin C++ memory management, allocation and release of feature structure nodes can account for up to forty per cent of total run time. Like in the WAM (Aït-Kaci, 1991), a general memory allocation scheme allowing arbitrary order of allocation and release of structures is not necessary in this context. Within a larger unit of computation, the application of a rule, say, the parser typically builds up structure monotonically; memory is only released in the case of a top-level unification failure when all partial structure built during this unification is freed. Therefore, PET employs a simple stack-based memory management strategy, acquiring memory from the operating system in large chunks which are then sub-allocated. A *mark – release* mechanism allows saving the current allocation state (the current stack position) and returning to that saved state at a later point. Thus, releasing a chunk of objects amounts to a single pointer assignment.

Also, feature structure representations are maximally compact.[12] In combination with other memory-reducing techniques (e.g. partial expansion and shrinking, substructure sharing, hyper-active

[12] The size of one dag node in the PET implementation of Tomabechi (1991) is only twenty four bytes, compared to, for example, fifty six in the Lisp-based LKB system.

Test Set	test items ♯	October 1996 lexical ϕ	parser ϕ	in %	out %	August 1999 lexical ϕ	parser ϕ	in %	out %
‘*tsnlp*’ test set	**4463**	**2·32**	**1·75**	**65·3**	**26·7**	**2·67**	**2·21**	**76·7**	**26·5**
S_Types	174	2·70	2·16	78·7	40·0	3·37	1·24	96·0	51·6
C_Agreement	123	2·59	1·33	58·8	10·0	2·27	1·28	77·9	10·0
C_Complementation	1010	2·45	2·19	62·2	12·1	2·99	1·67	83·1	10·5
C_Diathesis-Passive	220	3·58	2·87	25·3	8·1	3·52	3·52	50·5	6·3
NP_Agreement	1196	1·56	1·06	47·8	14·8	1·70	1·21	62·2	15·9
Other	1740	2·28	1·72	73·2	54·9	2·70	2·66	79·9	53·3
‘*aged*’ test set	**95**	**2·11**	**2·55**	**65·8**	—	**2·74**	**7·00**	**75·0**	—

(generated by [incr tsdb()] at 5-nov-1999 (17:11 h))

Table 5: Development of grammatical coverage and overgeneration over three years.

parsing) this results in very attractive memory consumption characteristics for the cheap parser, allowing to process the ‘*blend*’ test set with a process size of around one hundred megabytes (where Lisp- or Prolog-based implementations easily grow beyond half a gigabyte). To maximize compactness and efficiency, PET is implemented in ANSI C++, but uses traditional C representations (rather than C++ objects) for some central objects where minimal overhead is required (e.g. the basic features structure elements).

4 Quantifying Progress

The preceding acts have exemplified the benefits of competence and performance profiling in the application to isolated properties of various parsing algorithms. In this final act we take a wider perspective and use the profiling approach to give an impression of overall progress made in processing the LinGO grammar over a development period of three years. The oldest available profiles (for the ‘*tsnlp*’ and ‘*aged*’ test sets) were obtained with PAGE (version 2·0 released in May 1997) and the October 1996 version of the grammar; the current best parsing performance, to our best knowledge, is achieved in the cheap parser of PET. All data was sampled on the same Sun UltraSparc server (dual 300 Mhz; 1.2 gbytes memory; mildly patched Solaris 2.6) at Saarbrücken.

The evolution of grammatical coverage is depicted in Table 5, contrasting salient properties from the individual competence profiles (see Table 2) side by side; to illustrate the use of annotations on the test data, the table is further broken down by selected syntactic phenomena for the TSNLP data (Oepen et al., 1997, give details of the phenomenon classification). Comparison of the *lexical* and *parser* averages shows a modest increase in lexical but a dramatic increase in global ambiguity (by close to a factor of three for ‘*aged*’). Columns labeled *in* and *out* indicate coverage of items marked wellformed and overgeneration for ill-formed items, respectively. While the ‘*aged*’ test set does not include negative test items, it confirms that coverage within the VerbMobil domain has improved. However, the TSNLP test suite is far better suited to gauge development of grammatical coverage, since it was designed to systematically exercise different modules of the grammar. In fact, a net increase in coverage (from sixty five to seventy seven per cent) in conjunction with slightly reduced overgeneration confirms that the LinGO grammar engineers have steadily improved the overall quality of the linguistic resource.

The assessment of parser performance shows a more dramatic development. Average parsing times per test item have dropped by more than two orders of magnitude (a factor of one hundred and fifty on the ‘*aged*’ data), while memory consumption was reduced to about two per cent of the original values.

Version	Platform	Test Set	filter %	etasks ϕ	pedges ϕ	tcpu ϕ (s)	space ϕ (kb)
October 1996	PAGE	*'tsnlp'*	49·9	656	44	3·69	19016
		'aged'	51·3	1763	97	21·16	79093
August 1999	PET (cheap)	*'tsnlp'*	93·9	170	55	0·03	333
		'aged'	95·1	753	292	0·14	1435
		'blend'	95·5	3084	1140	0·65	10589

(generated by [incr tsdb()] at 5-nov-1999 (21:23 h)

Table 6: Development of salient performance parameters (PAGE vs. PET) over three years.

Because in the early PAGE data the 'quick check' pre-unification filter was not available, current filter rates for PET (and the other systems alike) are much better and result in a reduction of parser tasks that are actually executed. At the same time, comparing the number of passive edges licensed by the two versions of the grammar, provides a good estimate on the size of the search space processed by the two parsers. Although for the (nearly) ambiguity-free TSNLP test suite the *pedges* averages are almost stable, the '*aged*' data shows an increase by a factor of three. Asserting that the average number of passive edges is a direct measure for input complexity (with respect to a particular grammar), we extrapolate the overall speed-up in processing the LinGO grammar as a factor of roughly five hundred (again, *tcpu* values in Table 6 do *not* include garbage collection for PAGE which in turn is avoided in PET). Finally, Table 6 includes PET results on the currently most challenging '*blend*' test set (see above). Despite of greatly increased search space and ambiguity, the cheap parser achieves an average parse time of 650 milliseconds and processes almost ninety per cent of the test items in less than one second.[13]

5 Conclusion, Outlook, and Acknowledgments

Precise, in-depth comparison has enabled a large, multi-national group of developers to quantify and exchange algorithmic knowledge and benefit from each others experience. The [incr tsdb()] profiling package has been integrated with six processing environments for (HPSG-type) unification grammars so far; developers of other systems are encouraged to seek assistance in interfacing to the common metric (connections to the Alvey Tools GDE and Xerox XLE are currently under consideration) — scaling up and generalizing the set of competence and performance parameters is, once more, an empirical challenge. In parallel, the range of experimental choices in PET will be increased, aiming for import (and adaptation) of recent results, especially in the areas of fixed-arity feature structure encodings (inspired by the Tokyo LiLFeS implementation) and ambiguity packing (from the LKB).

The cathartic effect achieved in this period of close collaboration between sites over several years would not have been possible without the main characters, researchers, engineers, grammarians, and managers at Saarbrücken, Stanford, Tokyo, and Sussex University. To name a few, John Carroll, Liviu Ciortuz, Ann Copestake, Dan Flickinger, Bernd Kiefer, Hans-Ulrich Krieger, Takaki Makino, Rob Malouf, Yusuke Miyao, Stefan Müller, Mark-Jan Nederhof, Günter Neumann, Takashi Ninomiya, Kenji Nishida, Ivan Sag, Kentaro Torisawa, Jun'ichi Tsujii, and Hans Uszkoreit have all greatly contributed to the development of efficient HPSG processors as described above. Many of the individual achievements and results are reflected in the bibliographic references given throughout the play.

[13]To obtain the results on the '*blend*' test set shown in Table 6, an upper limit on the number of passive edges was imposed in the cheap parser; with a permissible maximum of twenty thousand edges, around fifty (in a sense pathological) items from the '*blend*' set cannot be processed within the limit and, accordingly, are excluded in the overall assessment. Maximal parsing times for the remaining test items range to around fourteen seconds for input strings that approximate twenty thousand edges and derive a very large number of readings.

References

Aït-Kaci, H. (1991). *Warren's Abstract Machine: A tutorial reconstruction.* Cambridge, MA: MIT Press.

Bouma, G., & van Noord, G. (1993). Head-driven parsing for lexicalist grammars. Experimental results. In *Proceedings of the 6th Conference of the EACL.* Utrecht, The Netherlands.

Calder, J. (1993). Graphical interaction with constraint-based grammars. In *Proceedings of the 3rd Pacific Rim Computational Linguistics Conference* (pp. 160 – 169). Vancouver, BC.

Callmeier, U. (2000). PET — A platform for experimentation with efficient HPSG processing techniques. In *(Flickinger et al., 2000).*

Carpenter, B. (1992). *The logic of typed feature structures.* Cambridge, UK: Cambridge University Press.

Carroll, J. (1994). Relating complexity to practical performance in parsing with wide-coverage unification grammars. In *Proceedings of the 31st meeting of the ACL* (pp. 287 – 294). Las Cruces, NM.

Carroll, J., Copestake, A., Flickinger, D., & Poznanski, V. (1999). An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the7th European Workshop on Natural Language Generation.* Toulouse, France.

Ciortuz, L. (2000). *Compiling HPSG into C.* In preparation, DFKI, Saarbrücken, Germany.

Copestake, A. (1992). The ACQUILEX LKB. Representation issues in semi-automatic acquisition of large lexicons. In *Proceedings of ANLP 1992* (pp. 88 – 96). Trento, Italy.

Erbach, G. (1991a). An environment for experimenting with parsing strategies. In J. Mylopoulos & R. Reiter (Eds.), *Proceedings of IJCAI 1991* (pp. 931 – 937). San Mateo, CA: Morgan Kaufmann Publishers.

Erbach, G. (1991b). A flexible parser for a linguistic development environment. In O. Herzog & C.-R. Rollinger (Eds.), *Text understanding in LILOG* (pp. 74 – 87). Berlin: Springer.

Flickinger, D., Nerbonne, J., Sag, I. A., & Wasow, T. (1987). *Toward evaluation of NLP systems* (Tech. Rep.). Hewlett-Packard Laboratories.

Flickinger, D., Oepen, S., Uszkoreit, H., & Tsujii, J. (Eds.). (2000). *Journal of Natural Language Engineering. Special Issue on Efficient processing with HPSG: Methods, systems, evaluation.* Cambridge, UK: Cambridge University Press. (in preparation)

Flickinger, D. P., & Sag, I. A. (1998). Linguistic Grammars Online. A multi-purpose broad-coverage computational grammar of English. In *CSLI Bulletin 1999* (pp. 64 – 68). Stanford, CA: CSLI Publications.

Kay, M. (1989). Head-driven parsing. In *Proceedings of International Parsing Technology Workshop* (pp. 52 – 62). Pittsburgh, PA.

Kiefer, B., Krieger, H.-U., Carroll, J. A., & Malouf, R. (1999). A bag of useful techniques for efficient and robust parsing. In *Proceedings of the 37th meeting of the ACL* (pp. 473 – 480). Baltimore, MD.

Krieger, H.-U., & Schäfer, U. (1994). *TDL* — A type description language for constraint-based grammars. In *Proceedings of the 15th International Conference on Computational Linguistics COLING* (pp. 893 – 899). Kyoto, Japan.

Malouf, R., Carroll, J., & Copestake, A. (2000). Efficient feature structure operations without compilation. In *(Flickinger et al., 2000).*

van Noord, G. (1997). An efficient implementation of the head-corner parser. *Computational Linguistics, 23*, 425 – 456.

van Noord, G., & Bouma, G. (1997). HDRUG. a flexible and extendible development environment for natural language processing. In *Proceedings of the EACL/ACL workshop ENVGRAM.* Madrid, Spain.

Oepen, S., & Carroll, J. (2000). Performance profiling for parser engineering. In *(Flickinger et al., 2000).*

Oepen, S., & Flickinger, D. P. (1998). Towards systematic grammar profiling. Test suite technology ten years after. *Journal of Computer Speech and Language, 12 # 4 (Special Issue on Evaluation)*, 411 – 436.

Oepen, S., Netter, K., & Klein, J. (1997). TSNLP — Test Suites for Natural Language Processing. In J. Nerbonne (Ed.), *Linguistic databases.* Stanford, CA: CSLI Publications.

Tomabechi, H. (1991). Quasi-destructive graph unification. In *Proceedings of the 29th meeting of the ACL* (pp. 315 – 322). Berkeley, CA.

Torisawa, K., & Tsujii, J. (1996). Computing phrasal signs in HPSG prior to parsing. In *Proceedings of COLING 1996* (pp. 949 – 955). Kopenhagen, Denmark.

Uszkoreit, H., Backofen, R., Busemann, S., Diagne, A. K., Hinkelman, E. A., Kasper, W., Kiefer, B., Krieger, H.-U., Netter, K., Neumann, G., Oepen, S., & Spackman, S. P. (1994). DISCO — an HPSG-based NLP system and its application for appointment scheduling. In *Proceedings COLING 1994.* Kyoto, Japan.

Wroblewski, D. A. (1987). Nondestructive graph unification. In *Proceedings of the 6th National Conference on Artificial Intelligence* (pp. 582 – 587). Seattle, WA.