Chapter 1

MEASURE FOR MEASURE: TOWARDS INCREASED COMPONENT COMPARABILITY AND EXCHANGE

Stephan Oepen
Center for the Study of Language and Information
Stanford University
Stanford, CA 94305 (USA)
oe@csli.stanford.edu

Ulrich Callmeier
Department of Computational Linguistics
Saarland University
66041 Saarbrücken (Germany)
uc@coli.uni-sb.de

Abstract Over the past few years, significant progress has been made 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 have been 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 example problems that motivate the experimental approach, and apply the empirical methodology in a fairly detailed discussion of progress made during a development period of three years.

2

Dramatis Personæ

Let there be some more test made of my metal,
Before so noble and so great a figure
Be stamp'd upon it.
(Shakespeare, 1623)

[...] *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 in 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 build a collection of grammar development environments, several highly engineered parsers (Kiefer, Krieger, Carroll, & Malouf, 1999) and an efficient generator (Carroll, Copestake, Flickinger, & Poznanski, 1999). Around 1998, the grammar formalisms and parsing group at Tokyo University[3] (Torisawa & Tsujii, 1996) made an entrance on stage and now supplies additional expertise on (abstract-machine-based) compilation of typed feature structures, Japanese HPSG, and grammar approximation techniques. More recently, people from Cambridge, Edinburgh, and Sussex Universities (UK) and from the Norwegian University of Science and Technology (in Trondheim) have also joined the cast.

Although their 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 & Schäfer (1994) – that allows grammars[4] to be processed by (at least) 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 difficult 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 re-implementation. 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 development status three years later. Finally, Act 5 applies some of the metrics introduced earlier to a multi-dimensional, cross-grammar *and* cross-platform comparison.

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 have 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

Table 1.1. Some of the parameters making up a competence & performance profile.

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)

deficit in the area of precise and systematic, let alone comparable, assessment of grammar and system behaviour.

Oepen & 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

Table 1.2. Reference data sets used in comparison and benchmarking with the LinGO grammar.

1	2	3	4	5	6	7	8	9
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

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 1.1 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 or so different parsing systems (with more underway; see Act 6), 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 with a grain of salt, 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 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 Verb*mobil* context (Wahlster, 2000), and (iv) a larger extract from later Verb*mobil* 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 1.2.[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.

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 one 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 oldest development platforms within the consortium – viz. 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

Table 1.3. Contrasting parser performance: passive, active, and hyper-active in the LKB.

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)

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 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.

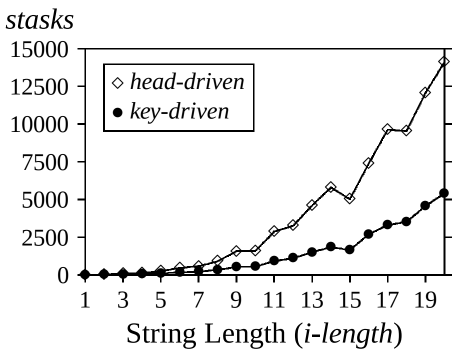
To obtain fully comparable results, the algorithm used in PAGE was imported into the LKB, which serves as the (single) experimentation environment for the remainder of this scene. The direct comparison is shown in Table 1.3 for three of the standard test sets. The re-implementation of the active parser 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 & 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]

Table 1.3 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 Verb*mobil* grammars in PAGE, the observed benefits of hyper-active parsing were broadly confirmed.

Figure 1.1. Effects of head- vs. key-driven rule instantiation on parser work load (‘*blend*’).



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

2.2 Rule Instantiation Strategies

Head-driven approaches to parsing have been explored successfully with lexicalized grammars like HPSG (see van Noord, 1997, for an 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 the most important. For terminological clarity, PAGE has coined 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.

Figure 1.1 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

Table 1.4. Head and key positions and distribution of active vs. passive edges for selected rules.

Rule Name	*head*	*key*	*aedges* *left → right*	*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

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 1.4 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; 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 1.4, 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.

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 (2001), Tomabechi (1991), and Malouf et al. (2000)). 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 DFKI 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 parsing) this results in very attractive memory consumption characteristics for the cheap parser, allowing it 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 feature structure elements).

4. Quantifying Progress

The preceding acts have exemplified the benefits of competence and performance profiling applied to isolated properties of various parsing algorithms. In this penultimate act we take a slightly 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 megahertz; 1.2 gigabytes memory; mildly patched Solaris 2.6) at Saarbrücken.

The evolution of grammatical coverage is depicted in Table 1.5, contrasting salient properties from the individual competence profiles (see Table 1.2) side by side; to illustrate the use of annotations on the test data, the table is further

Table 1.5. Development of LinGO grammatical coverage and overgeneration over three years.

	test	*October 1996*				*August 1999*			
Test Set	*items*	*lexical*	*parser*	*in*	*out*	*lexical*	*parser*	*in*	*out*
	♯	ϕ	ϕ	%	%	ϕ	ϕ	%	%
‘*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))

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 Verb*mobil* 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 (on identical hardware) have dropped by more than two orders of magnitude (a factor of two hundred and seventy on the ‘*aged*’ data), while memory consumption was reduced to about two per cent of the original values. 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

Table 1.6. Development of salient performance parameters (PAGE vs. PET) over three years.

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	36·69	79093
August 1999	PET	*'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)

number of passive edges is a direct measure of input complexity (with respect to a particular grammar), we extrapolate the overall speed-up in processing the LinGO grammar as a factor of roughly eight hundred (again, *tcpu* values in Table 1.6 do *not* include garbage collection for PAGE which in turn is avoided in PET; hence, the net speed-up is more than three orders of magnitude). Finally, Table 1.6 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. Multi-Dimensional Performance Profiling

The preceding acts have demonstrated how the [incr tsdb()] profiling approach enables comparison over time and across platforms, using the same grammar and reference input in both cases. In this final application of the framework, we draw the curtain wide open and attempt a contrastive study along several dimensions simultaneously. The basic theme of the exercise is the search for a reliable point of comparison across two distinct (though, of course, abstractly similar) systems, using different grammars (of different languages) and unrelated test data.

Table 1.7 summarizes a number of performance metrics for four different configurations that result from the cross product of applying two distinct processing environments (viz. the LKB and PET) to two distinct grammars (the LinGO grammar and the Japanese HPSG developed within Verb*mobil*; see Siegel, 2000). While of course within each row the results for both platforms were profiled against the same data set (viz. a sample of one thousand sentences randomly extracted from English and Japanese Verb*mobil* corpora, respectively), the exact details of the two test corpora will not matter for the present exercise; besides asserting a general, if rough similarity in origin and average length, nothing in the following paragraphs will hinge on inherent properties

Table 1.7. The matrix: simultaneous cross-grammar, cross-platform comparison.

	LKB		*PET*		*Speed-Up*
English	aedges	854	aedges	854	∼ *5.34*
	pedges	1850	pedges	1850	
	etasks	5946	etasks	6541	
	stasks	2695	stasks	2661	
	tcpu	2.96 s	tcpu	0.56 s	
	space	16894 kb	space	3436 kb	
Japanese	aedges	153	aedges	153	∼ *8.40*
	pedges	725	pedges	725	
	etasks	950	etasks	893	
	stasks	851	stasks	851	
	tcpu	0.56 s	tcpu	0.07 s	
	space	4053 kb	space	604 kb	
Speed-Up	∼ *5.29*		∼ *8.10*		

of the test data. As both systems implement the same common typed feature structure formalism (see Section 1 above) and obey the [incr tsdb()] application program interface, the matrix is complete and for each corresponding pair it has been confirmed that the results across systems yield an exact match (the minor diversions in task accounting are due to slightly different sets of 'quick check' paths and to a technical difference in how inflectional rules are applied by PET). Therefore, the complete symmetric matrix allows contrastive analyses both across grammars (vertically comparing within a column) and across platforms (horizontally comparing within a row). Before looking at the diagonals of the matrix – comparing across grammars and platforms simultaneously – we will use the available data to observe a number of relevant differences in the two grammars and parsing systems, respectively.

Comparing the two grammars, it seems to be the case that the Japanese grammar presents a smaller challenge to the processing system than is posed by the English grammar: while the absolute differences in the total numbers of passive edges (as a measure of global ambiguity, say) and overall parse times could in principle be a property of different test corpora (i.e. suggest that the Japanese sample on average was significantly less ambiguous and therefore easier to analyze than the English data), putting the two metrics into proportion reveals a genuine difference between the two grammars. Assuming that the average cost to build a single passive edge is relatively independent of the input data, the ratio of passive edges built per second is 625 (English) to 1295 (Japanese) for the LKB and 3304 to 10357, respectively, for PET. Further looking at the average size of a passive edge – i.e. relating the average amount of dynamically allocated memory during parsing (*space*) to the total number of passive edges built – suggests an explanation for the higher cost per edge in the English grammar: the ratio of *space* per passive edge is 9·3 kb (English) to 5·6 kb (Japanese) for the LKB (i.e. a ratio of 1·66) and 1·9 kb to 0·8 kb, respectively, for PET (i.e. a ratio of 2·38). Ignoring the somewhat surprising mismatch in exactly how much less memory is allocated per edge for the Japanese grammar in the two platforms for a moment, parsing with the Japanese grammar clearly seems to take both less time and memory.[14] The difference in average allocation per passive edge (the ratios of $\frac{9{\cdot}3}{5{\cdot}6} = 1{\cdot}66$ for the LKB constrasted with $\frac{1{\cdot}9}{0{\cdot}8} = 2{\cdot}38$ for PET, on the other hand, points to another differences between the grammars that, in turn, makes an asymmetry between the two platforms surface. Unlike the LKB, the PET grammar preprocessor deploys a technique known as *unfilling* (Götz, 1993; Gerdemann, 1995; Callmeier, 2000) – essentially a recursive elimination of information in feature structures that is implicit in the type of the structure – to reduce feature structure size at run-time. While the English LinGO grammar has been hand-tuned to achieve an effect similar to unfilling through an increased use of types (Flickinger, 2000), such a manual optimization has not been applied to the Japanese grammar. Accordingly, PET obtains a bigger bonus from the unfilling operation for structures of the Japanese grammar than it does for English (while the LKB in both cases uses the complete structures). The unfilling advantage on the Japanese grammar also explains the observed difference in the ratios of average cost per passive edge (measured as *pedges* per second: $\frac{1295}{625} = 2{\cdot}07$ for the LKB vs. $\frac{10357}{3304} = 3{\cdot}13$ for PET); again, the comparatively better performance of PET on the Japanese grammar almost exactly equals the relative ratio in edge size ($\frac{2{\cdot}07}{3{\cdot}13} = 0{\cdot}66$ vs. $\frac{1{\cdot}66}{2{\cdot}38} = 0{\cdot}69$). We can therefore conclude that the overall vertical speed-up across the two grammars (5·29 for the LKB and 8·10 for PET) accumulates three factors, viz. (i) reduced processing complexity (partly due to smaller feature structures) for the Japanese grammar, (ii) reduced test corpus complexity (to account for the additional speed-up over the factor-of-two decrease in cost per edge observed in the LKB), and (iii) increased unfilling efficiency (explaining why PET performs relatively better on the Japanese than on the English grammar).

Finally, what if we pretended that the comparison matrix was only partially available, say providing one profile of the LKB applied to the English grammar and another sample of PET processing the Japanese grammar? At first, it seems, nothing much can be concluded from the observation that PET takes 0·07 seconds to solve one problem while the LKB requires 2·96 seconds to solve a different problem. Without knowledge about the complexity of the actual problem, relating raw processing times must be completely uninformative. To arrive at a comparative assessment of relative performance for the two systems, instead, would require a derived measure of generalized (or inherent) complexity, a metric that with sufficient confidence can be expected to provide a stable predictor of processing cost independent of the grammar and type of test data. From some of the observations reviewed in the previous paragraphs, it follows that the rate of passive (or active) edges per (cpu) second will not be a good measure because the proportion of successful vs. failing unifications may vary drastically across grammars (and indeed does for the data in Table 1.7), where only succeeding unifications will be reflected as an edge in the chart. By the same token, looking at *stasks* per second would suffer from the same potential for skewing.

But what about the ratio of executed parser actions (*etasks*) per second of total parsing time? Applying this metric to the problem at hand, we obtain $\frac{12757}{2009} = 6{\cdot}35$ and $\frac{11680}{1520} = 7{\cdot}68$ for the lower right to upper left and upper right to lower left diagonals of Table 1.7, respectively. If executed tasks per second was a suitably independent metric, the diagonal comparison would thus predict that PET is between a factor of 6·35 and 7·68 more efficient that the LKB. Looking at the actual values – speed-ups of 5·34 and 8·40 on the English and Japanese grammars, respectively – the prediction is reasonably accurate; it would indeed seem that the average cost of executing a single parser task is a relatively stable indicator of overall system efficiency, at least for two platforms that despite all technical differences share a large number of basic assumptions and design. At this point, however, we can only speculate about why *etasks* per seconds appears to be a surprisingly good metric of (abstract) efficiency for the two systems considered. Firstly, total parsing times are dominated by feature structure manipulation, that is calls to the unification and copy routines; executing a parser task is the fundamental operation that (in most cases) requires exactly one unification and, for some subset of tasks, also a subsequent copy. Independent of the unification to copy cost and the unification failure to success ratios, all constraint solver activity somehow originates in a task execution. Secondly – even with moderate-size and mildly ambiguous test data – the number of executed tasks will be very large and therefore the ratio of *etasks* per cpu second has been found to be quite stable across varying test data or grammars. Thirdly, where two (closely related) systems incorporate similar approaches to parsing and reducing search, the number of parser tasks that come to be executed will correlate in some informal sense to the size of the search space (or problem complexity) that has been explored; therefore relating it to the time that a system takes to solve that problem yields a measure of efficiency. Obviously, these conclusions are necessarily preliminary and – given remaining noise in the cost per parser task across platforms and grammars – the metric proposed can only approximate relative efficiency; indeed, looking at another broad-coverage HPSG – viz. the German Verb*mobil* grammar – we found the general prediction confirmed but the variance of diagonal comparison slightly larger than with the English – Japanese pairing.

6. Conclusion – Recent Developments

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 some six processing environments for (HPSG-type) unification grammars and has thus facilitated a previously unmatched degree of cross-fertilization. Many of the parameters of variation in system design and optimization – including the choice of parsing strategy, feature structure encoding, and unification scheme – require detailed knowledge about the relative contributions of sub-tasks (feature structure unification vs. copying, for example) to the overall problem size as well as a fine-grained, accurate understanding of which aspects of the problem (as defined by the grammar and input data) are especially hard on the processor. Our integrated competence and performance profiling approach aims to make appropriate data and suitable analysis techniques available to grammar and system engineers.

The modular PET platform provides an experimentation tool box that allows developers to combine various encoding and processing techniques and rapidly assess both their strong and weak points. The attractive practical performance of the cheap parser has made it the preferred run-time system for test set processing in the development of several large-scale HPSG grammars. PET has also been successfully deployed in commercial products.

As this play was first brought to stage early in 2000, obviously there have been a number of recent developments not reflected here. Beyond what was shown in Acts 3 through 5, the range of experimental choices in PET has been increased significantly, particularly in the areas of fixed-arity feature structure encodings (in the tradition of Prolog compilation) and ambiguity packing (from the LKB). Callmeier (2002) presents an empirical study comparing the benefits of various feature structure encoding techniques. A teichoscopic view of collaborative activities among the groups is compiled by Oepen, Flickinger, Tsujii, & Uszkoreit (2002).

Acknowledgments

This play builds heavily on results obtained through collaboration between a largish group of people at several sites. The peripeteia 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.

Notes

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/`'.

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 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.

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 release version of the LinGO grammar, frozen in August 1999. Here and in the tables to come the symbol '$\sharp$' indicates absolute numbers, while 'ϕ' denotes average values. Coverage on the '*blend*' corpus is comparatively low, as it includes Verb*mobil* data (specifically vocabulary) that became available only after the grammar had been frozen for our experiments.

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, education, and generation (in an AAC basic research project), PAGE develoment since 1997 has emphasized robust parsing methods with speech recognizer output (in application to Verb*mobil*).

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 & Carroll (2000) 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.

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.

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.

13. To obtain the results on the '*blend*' test set shown in Table 1.6, 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.

14. The calculation of average allocation cost per passive edge is, of course, only approximative in that other computation – primarily dag and arc allocations during failed unification attempts and the edge structures themselves – also contributes to overall memory consumption. However, both platforms utilize a hyper-active chart parser, so that active edges do not have a feature structure associated with them; likewise, a high filter efficiency reduces the number of failed unifications, such that (the feature structures associated with) passive edges certainly account for the bulk of dynamic allocation.

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 European Chapter of the ACL* (pp. 71 – 80). Utrecht, The Netherlands.

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

Callmeier, U. (2000). PET — A platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering, 6 (1) (Special Issue on Efficient Processing with HPSG)*, 99 – 108.

Callmeier, U. (2002). Preprocessing and encoding techniques in PET. In S. Oepen, D. Flickinger, J. Tsujii, & H. Uszkoreit (Eds.), *Collaborative language engineering. A case study in efficient grammar-based processing.* Stanford, CA: CSLI Publications.

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 32nd Meeting of the Association for Computational Linguistics* (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 the 7th European Workshop on Natural Language Generation* (pp. 86 – 95). Toulouse, France.

Ciortuz, L. (2001). *LIGHT. A feature constraint language applied to parsing with large-scale HPSG grammars* (Unpublished DFKI Research Report). Saarbrücken, Germany: Deutsches Forschungszentrum für Künstliche Intelligenz GmbH.

Copestake, A. (1992). The ACQUILEX LKB. Representation issues in semi-automatic acquisition of large lexicons. In *Proceedings of the 3rd ACL Conference on Applied Natural Language Processing* (pp. 88 – 96). Trento, Italy.

Erbach, G. (1991a). An environment for experimenting with parsing strategies. In J. Mylopoulos & R. Reiter (Eds.), *Proceedings of the 12th International*

Joint Conference on Artificial Intelligence (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, Germany: Springer.

Flickinger, D. (2000). On building a more efficient grammar by exploiting types. *Natural Language Engineering, 6 (1) (Special Issue on Efficient Processing with HPSG)*, 15 – 28.

Flickinger, D., Nerbonne, J., Sag, I. A., & Wasow, T. (1987). *Toward evaluation of NLP systems* (Technical Report). Hewlett-Packard Laboratories. (Distributed at the 24th Annual Meeting of the Association for Computational Linguistics)

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

Gerdemann, D. (1995). Term encoding of typed feature structures. In *Proceedings of the 4th International Workshop on Parsing Technologies* (pp. 89 – 97). Prague, Czech Republik.

Götz, T. (1993). *A normal form for typed feature structures.* Magisterarbeit, Universität Tübingen, Tübingen, Germany.

Kay, M. (1989). Head-driven parsing. In *Proceedings of the 1st International Workshop on Parsing Technologies* (pp. 52 – 62). Pittsburgh, PA.

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

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

Malouf, R., Carroll, J., & Copestake, A. (2000). Efficient feature structure operations without compilation. *Natural Language Engineering, 6 (1) (Special Issue on Efficient Processing with HPSG)*, 29 – 46.

van Noord, G. (1997). An efficient implementation of the head-corner parser. *Computational Linguistics, 23 (3)*, 425 – 456.

van Noord, G., & Bouma, G. (1997). Hdrug. A flexible and extendible development environment for natural language processing. In *Proceedings of the Workshop on Computational Environments for Grammar Development and Linguistic Engineering* (pp. 91 – 98). Madrid, Spain.

Oepen, S., & Carroll, J. (2000). Performance profiling for parser engineering. *Natural Language Engineering, 6 (1) (Special Issue on Efficient Processing with HPSG)*, 81 – 97.

Oepen, S., Flickinger, D., Tsujii, J., & Uszkoreit, H. (Eds.). (2002). *Collaborative language engineering. A case study in efficient grammar-based processing.* Stanford, CA: CSLI Publications.

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* (pp. 13 – 36). Stanford, CA: CSLI Publications.

Shakespeare, W. (1623). *Measure for measure* (First Folio ed.). London, UK: I. Iaggard and E. Blount.

Siegel, M. (2000). HPSG analysis of Japanese. In W. Wahlster (Ed.), *Verbmobil. Foundations of speech-to-speech translation* (Artificial Intelligence ed., pp. 265 – 280). Berlin, Germany: Springer.

Tomabechi, H. (1991). Quasi-destructive graph unification. In *Proceedings of the 29th Meeting of the Association for Computational Linguistics* (pp. 315 – 322). Berkeley, CA.

Torisawa, K., & Tsujii, J. (1996). Computing phrasal signs in HPSG prior to parsing. In *Proceedings of the 16th International Conference on Computational Linguistics* (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 of the 15th International Conference on Computational Linguistics.* Kyoto, Japan.

Wahlster, W. (Ed.). (2000). *Verbmobil. Foundations of speech-to-speech translation.* Berlin, Germany: Springer.

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