Learning attribute values in typed-unification grammars: On generalised rule reduction

Liviu Ciortuz

CS Department, University of York, UK. E-mail: ciortuz@cs.york.ac.uk

Abstract: We present Generalised Reduction (GR), a learning technique for generalising attribute/feature values in typed-unification grammars. GR eliminates as many as possible of the feature constraints (FCs) from the type feature structures (FSs) while applying a criterion of preserving the parsing results on a given, training corpus. For parsing with GR-restricted rule FSs, and for checking the correctness of obtained parses on other corpora, one may use a new form FS unification which we call two-step unification to speed up parsing. We report results on a large-scale HPSG grammar for English.

1 Introduction and Related Work

Interest in typed-unification grammars in Natural Language Processing can be traced back to the seminal work on the PATR-II system [20]. Since then, different types of unification-based *grammar formalisms* were developed by the Computational Linguistics community — most notably Lexical Functional Grammars [11], Head-driven Phrase Structure Grammars (HPSG, [19]) and Categorial (unification) Grammars [23]. During the last years, such wise-coverage grammars were implemented.[1]

FS unification is by far the most time-consuming task during unification with large-scale typed-unification grammars. Therefore, making it more efficient is of great interest. Several solutions for speeding-up FS unification have been proposed until now: FS sharing [18] [22], the Quick-Check pre-unification filter [12], the hyper-active parsing [16] strategy, or parsing with ambiguity packing [17]. Compilation of parsing with typed FSs [24] [7] is another approach for speeding-up the FS unification.

This paper will show that an ILP-inspired learning technique [14] for generalising typed-unification grammars opens the way for further applying a new speed-up technique for parsing with such grammars, namely *two-step unification*.

The QC pre-unification filter, one of the most effective speed-up technique for FS unification, is very simple in itself. It considers the set of feature paths most probably leading to unification failure, and then compares the corresponding values of these paths inside the two FSs to be unified. If such a pair of values is — eventually if their root sorts are — incompatible, then certainly the two FSs don't unify. The QC filter rules out many of the unifications, and speeds up considerably the parsing — around 63% for the PET system [4], and 42% for the LIGHTcompiler [8].[2]

The effectiveness of the QC pre-unification technique resides in the fact that there is a relatively reduced number of failure paths inside rule FSs. Using the LIGHT system, we identified 148 failure paths on the CSLI test-suite, out of the total of 494 paths inside rule FSs for LinGO. Among these paths, only a small number is responsible for most of the unification failures.[3]

The *Generalised Reduction* technique will work on a larger subset of feature paths inside rule FSs, namely those paths which contribute to unification failure, by eliminating those FCs which are not decisively contribute (or only seldom contribute) to unification failure. The paths retained by GR will be called in the sequel *GR-paths*.[4]

We will present an efficient algorithm for generalised reduction of rule FSs in large-scale typed-unification grammars. Measurements done on the LinGO grammar revealed that almost 60% of the feature constraints in the expanded grammar rule FSs can be eliminated without affecting the parsing result on the CSLI test-suite. As a consequence, the

[1] For instance the HPSG for English developed at Stanford [9] called LinGO, two HPSGs for Japanese developed at Tokyo University [13], and respectively at DFKI–Saarbrücken, Germany [21], and the HPSG for German, developed also at DFKI [15]. Large-scale LFG grammars were developed by Xerox Corp., but they are not publicly available.

[2] LIGHT uses compiled unification, which without QC proved to be (40%) faster than unification in PET, the fastest interpreter running LinGO-like grammars. This is why the speed-up factor provided by QC for LIGHT is lower than that for PET.

[3] For the LinGO grammar, in the LIGHT system, we used maximum 43 QC-paths.

[4] QC and two-step unification, working on the GR-learnt grammar, will be orthogonal.

Light system registered a reduction of the unification time that sped parsing up to 23%.[5]

2 Generalised Reduction — Definitions

From the logical point of view, FSs can be viewed as positive OSF-clauses, which are finite sets of atomic OSF-constraints [2] — sort constraints, feature constraints and equation constraints. Therefore the generalisation of FSs can be logically achieved through elimination of some atomic constraints from the FS. We showed in [6] how one can improve the linguistic competence — i.e., enhancing the coverage — of a given typed-unification grammar by guided generalisation, using parsing failures.[6]

Generalised reduction (GR) is a restricted form of FS generalisation that eliminates as many as possible of the feature constraints from a type FS of the given unification grammar while applying an *evaluation criterion* for maintaining the parsing results on a large corpus/test-suite. Note that although GR explicitly eliminates only feature constraints, it may be the case that sort and/or equation constraints associated with the value of an eliminated feature constraint get implicitly eliminated through FS normalisation [2].

We make the observation that the notion of *restriction* in the HPSG literature designates the elimination of certain (few) features following the application of a parsing rule. Generalised Reduction extends this operation on arbitrary chosen features (of course, ensuring the parsing correctness on the given training corpus). We prefer the name *reduction* for this operation, since from the logical point of view, eliminating one feature constraint from a FS logically generalises that FS, while its set of attribute paths gets indeed *restricted*. However, we will denote the result of applying GR to a FS as the *GR-restricted form* or the *GR-learnt form* of that FS.

Alternatively, one may think of generalised reduction as *relative unfilling*. Indeed, *expansion* for typed-unification grammars — the procedure that propagates both top-down in the type hierarchy and locally in the typed FS the constraints associated to types in the grammar — is usually followed by *unfilling* [10], a feature constraint removal technique which is corpus-independent. Generalised Reduction may be thought as a corpus/test-suite relative unfilling technique.

GR is shown not only to improve the performance of the given grammar — since it maximally reduces the size of the (rule) FSs in grammar — but also adds an interesting improvement to the parsing system design. As measurements on parsing the CSLI test-suite have shown that in average only 8% of the items produced on the chart during parsing constitute part of the full parses, one can devise unification as a two-step operation. Parsing with *two-step unification* will *i.* use the GR-learnt form of the grammar rules to produce full parses, and *ii.* eventually complete/check the final parses using the full (or, better: complementary) form of rule FSs.

3 Generalised Reduction of Rule FSs — Algorithms

We present two algorithms for generalised reduction of types in unification grammars. The first one, called A in the sequel is a simple, non-incremental one. From it we derived a second, incremental algorithm (B) that we have optimised. Both algorithms have mainly the same kind of input and output.

Input: $\mathcal{G}$, a typed-unification grammar, Θ a test-suite i.e., a set of sentences annotated by a *parsing evaluation* function;
Output: a more/most general grammar than $\mathcal{G}$ obtained by generalised reduction of FSs, and producing the same parsing evaluation results as $\mathcal{G}$ on Θ.

The measurements provided below used the following *parsing evaluation criterion:* the number of full parses, the number of attempted unifications and the number of successful unifications must be preserved for each sentence in the test-suite, after the elimination of a (selected) feature constraint.

Notes:
1. Due to the large size of (both) the LinGO grammar and the test-suite used for running GR, this parsing evaluation criterion is a good approximation of the following "tough" criterion: the set of actual parses (up to the associated FSs) delivered for each sentence by the two grammar versions must be exactly the same. We make the remark that after applying GR on the LinGO using the CSLI test-suite,

[5] Note that the feature constraints eliminated through generalised reduction are usually on the bottom levels of rule FSs, therefore are least visited during unification.

[6] The approach we followed in developing both the generalisation and specialisation procedures for learning attribute values in typed-unification grammars is basically the one proposed by Inductive Logic Programming (ILP) [14]. We adapted/applied the main ILP ideas to the feature constraint (subset of the first-order) logic specific to such grammars [2] [5] [3].

this "tough" criterion was satisfied.
2. The parsing evaluation criteria must be actually combined with one regarding the memory/space resources exhaustion: if for a given sentence, after elimination of a feature constraint the resources initially allocated for parsing are exhausted, then that feature constraint is kept in the grammar.
3. Although in our experiments only the rule types in $\mathcal{G}$ were subject to generalised reduction, the formulation of the two GR algorithms can be extended to any FS in the input grammar. Applying GR to the type FSs used in type-checking is easy, but if parsing correctness must be ensured — i.e., elimination of over-generalised parses is required in the end —, then things get more complicated than for rule FSs. Extending GR to lexical entries is even more demanding in case lexical rules are used.

A. *A* ***simple*** *GR procedure:*

> **for** each rule $\Psi(r)$ in the grammar $\mathcal{G}$
>   **for** each feature constraint φ in $\Psi(r)$
>   **if** removing φ from $\Psi(r)$
>   preserves the parsing (evaluation) results
>   **for** each sentence in the test-suite Θ
>   **then** $\Psi(r) := \Psi(r) - \{\varphi\}$;

Remarks:

1. Obviously, the GR result is dependent on the order in which feature constraints are processed: the elimination of the constraint φ can block the elimination of the constraint φ' if φ is tried first, and vice-versa. Therefore usually there is no unique GR form for a given grammar.
2. In the actual implementation of the GR algorithms
− we first worked on the elimination of feature constraints from key arguments (in the decreasing order of their "usage" frequency on the given test-suite), then from non-key arguments (according to the same kind of frequency), and finally from rule LHS substructures;
− inside a "reduction partition" (i.e., key argument, non-key argument, LHS structure), feature constraints are tried for elimination following the bottom-up traversal order of the acyclic rooted graph representing the rule FS.
3. The first two **for**'s in the procedure A can be combined into a single **for** iterating over the set of all feature constraints $< r, \varphi >$ in the grammar's rules ($\varphi \in \Psi(r)$).

The more efficient GR algorithm B introduced below is obtained basically by:
− reversing the two **for**'s, namely the one identified above (4), and the one iterating over the sentences in the test-suite. The advantage is that those sentences which become/are correctly parsed by $\mathcal{G}_i$ — an intermediate generalised form of $\mathcal{G}$ — need not to be tried again when computing $\mathcal{G}_{i+1}$.
− getting a first GR version of the input grammar by running the simple procedure A on one or more sentences, and subsequently improving it iteratively.

B. *An* ***improved*** *GR procedure:*

> **do** 0. $\mathcal{G}_0 = \mathcal{G}$, $i = 0$;
>   1. Apply the procedure A on a sentence s from Θ;
>   let $\mathcal{G}_{i+1}$ be the result
>   2. eliminate from Θ the sentences for which $\mathcal{G}_{i+1}$
>   provides the same parsing results as $\mathcal{G}$
> **until** $\Theta = \emptyset$.

The reader should note that the set of FCs to be tried for elimination in the procedure A called at the step 1 — see the Remark 3 following the presentation of the algorithm A — is provided by $\mathcal{G} - \mathcal{G}_i$, therefore it is becoming lower and lower at each loop. Indeed, from the logical point of view, $\mathcal{G} = \mathcal{G}_0 \models \ \ldots \ \models \mathcal{G}_{n+1} \models \mathcal{G}_n \ \ldots \ \models \mathcal{G}_2 \models \mathcal{G}_1$. Viewed as set of constraints, $\mathcal{G} = \mathcal{G}_0 \supset \ \ldots \ \supset \mathcal{G}_{i+1} \supset \mathcal{G}_i \ \ldots \ \supset \mathcal{G}_2 \supset \mathcal{G}_1$ therefore $\mathcal{G} - \mathcal{G}_{n+1} \subset \mathcal{G} - \mathcal{G}_n$.

Further improvements:

1. At the step 2, the elimination of sentences from Θ is done in a procrastinated manner: initially Θ is sorted according to the number of failed unifications per sentence when using $\mathcal{G}$, — therefore the sentence s chosen at the step 1 may be considered the first sentence in Θ_i — and subsequently only the first sentences from Θ_i which are correctly parsed by $\mathcal{G}_{i+1}$ are eliminated.[7] The reason for procrastinated elimination of sentences from Θ is the significant time consumption engendered by the over-parsing and/or the exhaustion of allocated resources that may be caused — very frequently, in the beginning — for some of the the remaining sentences in Θ due to the (premature) elimination of certain constraints in precedent loops in the procedure B.

2. Only FCs from rules involved in the parsing of the sentence s at the step 1 must be checked for elimination in view of generalisation. If the sentence exhausted the allocated resources for parsing, this optimisation does not apply, because it is not possible to tell in advance which rules might have been used if exhaustion had not been reached.

3. A "preview" test which decides whether a whole rule partition is immediately learnable is highly improving the running time of the GR procedure on LinGO. This test means eliminating from

[7] Θ_{i+1} starts with the first sentence in Θ_i which causes over-parsing with $\mathcal{G}_{i+1}$. Obviously, $\Theta_0 = \Theta$.

$\mathcal{G} - \mathcal{G}_i$ in a single step all the constraints in a partition — argument, respectively LHS substructure —, for the rules previously identified as contributing to parsing the sentence s. At rule level, identifying the FCs that must be retained in $\mathcal{G}_{i+1}$ may be further speeded up through halving the FC search space.

4. As constraints in FCs in a rule FS are implicitly ordered by their position in the rooted acyclic graph representing the rule, another optimisation is possible: if all ancestors of a FC have been approved for elimination, than that FC may be eliminated immediately, assuming that its value is not coreferenced in a subsequent partition — argument of LHS —, and this fact that can be determined in the preparation of the GR application.

5. In our current implementation of the GR procedure, the *exhaustion* of resources allocated for parsing is controlled by a sentence-independent criterion. Currently, the allocated resources allow for the complete parsing of all sentences in the test-suite. But using such a criterion has the following drawback: if the procedure A causes strong over-parsing (possibly looping) for a sentence, then this fact is detected eventually only after *all* resources get exhausted. We suggest a better, punctual criterion for evaluating the consumption of allocated resources for parsing: *i.* initially, for each sentence in the given test-suite register the parsing time using $\mathcal{G}$; *ii.* if during the generalised reduction, parsing a sentence using the current generalised form of rules consumes more than n times the initially registered parsing time without finishing, then consider that that sentence is a resource exhausting one.

Incorporating all but the last improvement mentioned above, the procedure B required 61 minutes on a Pentium III PC at 933MHz running Red Hat Linux 7.1. (All measurements reported in the next section were done on that PC.)

Because the test-suite for training is processed incrementally only by the algorithm B, we will refer to it as the *incremental* GR algorithm, while the algorithm A will be called the non-incremental one.

4 Measurements and Comparisons

When running the GR algorithm A, the reduction of the number of FCs in rule argument sub-structures of LinGO was impressive: 61.38% for key/head arguments and respectively 61.85% for non-key arguments (see Figure 1). The number of feature paths in rule argument sub-structures was reduced from 494 to 234, revealing that the decisive contribution to unification failures for LinGO on the CSLI test-suite is restricted to less than half of the feature paths in the arguments. We will call those feature paths *GR-paths*.

This result complements the view provided by the QC technique: only 8.5% of the total feature paths in the arguments are responsible for most of the unification failures during parsing with LinGO. As expected, all QC-paths we are using in LIGHT for LinGO are among the GR-paths identified by the procedure A.

While the average number of FCs eliminated from LinGO by the GR algorithm B and the parsing performances on the resulted grammar version are only slightly different than those provided by the algorithm A, the number of GR-paths retained in the rule arguments is significantly higher in the B case than in the A case. This is difference is explained by the test-suite fragmentation/atomisation on which the design of the procedure B was based.

The table in Figure 2 presents a snapshot on the most needed memory resources (and the reduction percentage) when parsing with full-form rule FSs, respectively GR-restricted rule FSs. The graphic in Figure 3 presents the evolution of the average rule reduction rate for getting the LinGO GR-version using the algorithm B. Figure 4 illustrates the reduction of the parsing time for the sentences in the CSLI test-suite when running the LinGO grammar with the GR-restricted rules. One can see in this last figure — especially for the sentences requiring many unifications — that although the number of unifications is increased, the total parsing time is reduced.

The GR algorithm B may be generalised so to provide the inner loop (the call to the procedure A) not only one but several (n) sentences which are incorrectly parsed by $\mathcal{G}_{i+1}$. If so, the processing time for getting the GR-version of the given grammar will increase. However, as the table in Figure 5 shows, this is a convenient way to get (empirically) a smaller number of computed GR-paths.We expect this would be convenient for further research — compiled unification of FSs using look-up tables — using the GR-result or independently of this. [8]

[8] Note that the running time of all GR algorithms can be significantly improved if, in the beginning of the GR application, all feature constraints not used when parsing the training test-suite are eliminated from the initial grammar. (Unfortunately, the current version of the LIGHT system cannot identify these unused FCs.) However, eliminating from start these FCs will bias the set of GR-paths to be computed. In general, FCs initially not used in parsing the training test-suite may however appear in the GR-restricted grammar version.

	A	B
FC reduction rate: average	58.92%	56.64%
key, non-key arg, LHS	61.38% 61.85% 55.96%	52.21% 57.75% 60.73%
GR vs. total feat. paths in rule arg. (reduction)	234/494 (47.36%)	318/494 (35.62%)
average parsing time, in msec.		
using full-form rule FSs	21.617	
1st-step unification (reduction %)	16.662 (22.24%)	16.736 (22.07%)
emulated 2-step unification (red. %)	18.657 (13.12%)	18.427 (14.20%)

Fig. 1. Comparison between the results of applying the two GR procedures on LinGO/CSLI.

	full-form rules	GR-restricted, 1st-step-unif.	GR-restricted, emulated 2-step-unif.
heap cells	101614	38320 (62.29%)	76998 (24.23%)
feature frames	60303	30370 (49.64%)	58963 (02.22%)

Fig. 2. A snapshot view on reduction of memory usage when parsing the CSLI test-suite with GR-LinGO.

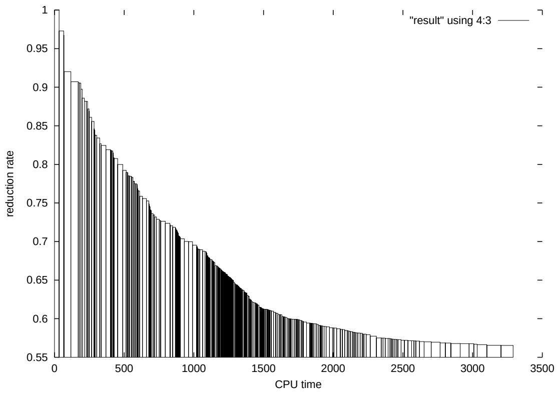

Fig. 3. Procedure B: rule reduction rate vs. CPU time consumption on LinGO/CSLI.

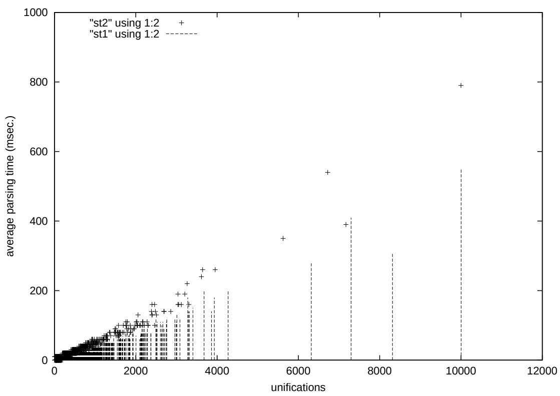

Fig. 4. Comparisons: unifications vs. parsing time on the CSLI test-suite using LIGHT on LinGO: the full version vs. the GR-restricted version using 1st-step unification.

n	running time	FCs reduction rate	GR-paths
2	1h 36min	59.38	200
4	2h 27min	59.42	187
8	4h 03min	59.39	187

Fig. 5. A comparison on running the parameterised (n) GR-procedure B on the CSLI test-suite.

We tested a GR-version of LinGO (produced when using the CSLI test-suite) on the *aged* test-suite also provided with LinGO. This test-suite requires in average 4.65 times more unifications per sentence than the CSLI test-suite. (The CSLI test-suite has 1348 sentences of average length of 6.38 tokens and the ambiguity 1.5. For the *aged* test-suite, the average length is 8.5 and the ambiguity 14.14.) Of the 96 sentences in the *aged* test-suite, 59 were correctly parsed, 3 exhausted the allocated resources, and 34 were over-parsed. The average parsing precision for the sentences non-exhausting the allocated resources was 83.79%.

On LinGO, applying the algorithm A (only) to the most complicated sentence in the CSLI test-suite resulted in a 18% reduction of the number of FCs inside rule FSs. We obtained the same reduction rate (i.e., feature constraint elimination percentage) on LinGO, running 30 iterations using the simplest sentences in the CSLI test-suite.

To produce a GR-version of LinGO using the *aged* test-suite for training, the procedure B needed 4h 44min, and the reduction rate was 65.60%. However, it took only 38min to further improve the previously obtained GR-version of LinGO (on the CSLI test-suite), using the *aged* test-suite, and the rule FS reduction rate went down only with 3.1%.

5 Further Work

A. Compiled 2-step unification vs. interpreted 2-step unification:

As already presented in Section 2, unifying a (passive item) FS with a rule argument FS may be seen as a two step/phase process: *i.* performing unification with the GR-restricted form of the rule argument FS, and if successful, *ii.* continuing unification with the GR-complementary rule argument sub-structure. The interesting thing here is that the second unification phase may be postponed. If parsing doesn't loop (and the allocated resources are not consumed), this second step is eventually needed only for full parses.[9]

Different strategies may be used to apply efficiently the unification's second phase.

The simplest one is to emulate the second unification step as (on-line) type-checking via FS unfolding.[10] (This is why this step may be called "rule checking".) Unfortunately, from the implementation point of view, rule/type checking doesn't commute with (further) rule application. This is why, when emulating the second unification step as type-checking, the parsing ambiguity engenders a lot of redundant work, namely retrieving the GR-values — i.e., the values of the GR-paths — in the FS to be checked, each time the second-unification step is applied on that FS.[11]

A simple solution that can be imagined to avoid this redundancy is to save those GR-values. But, if the GR-values have to be saved after each application of a rule — i.e., the first unification step —, then it would be too much time-costly. This cost may simply exceed the cost for the redundant work that consists in retrieving the GR-values, because only a reduced percentage of items take usually part in full parses.

[9] Otherwise, i.e., if parsing correctness is required and using the GR-restricted form of rules make the parser loop and/or the allocated resources are consumed, then we have to use the full form of rules. Subsequently, the GR-learnt grammar version may be improved.

[10] This is how we did the measurements for the bottom line showing parsing results in Figure 1.

[11] This is why on the *aged* test-suite, we found that parsing with emulated two-step unification is not gaining significant speed w.r.t. parsing with full-form rules.

Therefore we suggest the following trade-off:
i. given a certain item — eventually found on a full parse —, perform the first application of the second unification step on that item as type checking; additionally, save the identified values of the GR-paths.[12]
ii. subsequent second-unification step operations performed on that item may be performed using the GR-complementary sub-structure of the rule, starting from the (previously) stored GR-values.[13]

B. Identifying *exception FCs*, and *handling* exception FCs typed-unification grammars:

If for instance only successful unifications designate which rules must be checked during rule reduction, the number of eliminated FCs increases (≈5% of all rule FCs in LinGO), but the parsing time with the GR form of the grammar might increase because the number of parse items produced may increase.

Those FCs involved in the difference set "successful" unifications – "successful+failed" unifications) may be seen as FC *exceptions*. We suggest the following *handling* of exception FCs in a two-step unification approach: if at least one parent such an "exception" FC in the GR form of the rule was passed in a successful unification, then trigger the application of the full form of the rule FS either immediately or at a certain higher phrase level.

Exception FCs may thus constitute the object of a trade-off between over-parsing and parsing pruning, using the two-step unification strategy.

GR would benefit if its application would be preceded by a procedure for automate detection of cycles in parsing with generalised forms of the input typed-unification grammar. Finally, integrating linguistics-specific knowledge and testing GR on other grammars and test-suites will provide additional insights into the use of this learning method.

Conclusion: Generalised Reduction extends the view previously provided by the Quick-Check pre-unification technique [12] on feature paths inside rule argument FSs for large-scale typed-unification grammars. We showed that if the number of FCs in rule FSs may be highly reduced — eventually through automate learning —, then a simple technique called two-step unification can significantly

[12] In compiled unification, those GR-values may be simply retrieved via the X variables which are indexing the nodes of the compiled FSs [1].

[13] The compiled form of the second unification step is obtainable by application of abstract program transformation techniques on the (non-SRC) compiled form of the rule FS.

speed-up the parsing. We elaborated two GR algorithms for rule FSs, firstly a simple, non-incremental one, and then an optimised, incremental one. In the LIGHT system, to the 42% speed-up factor provided by QC on the LinGO grammar, GR may add a subsequent 23% reduction of the average parsing time.

Acknowledgements: The LIGHT system was developed at the Language Technology Lab of The German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany. The present work was done while the author was supported by an EPSRC ROPA grant at the University of York.

References

1. H. Aït-Kaci and R. Di Cosmo. Compiling order-sorted feature term unification. Technical report, Digital Paris Research Laboratory, 1993. PRL Technical Note 7, downloadable from http://www.isg.sfu.ca/life/.
2. H. Aït-Kaci and A. Podelski. Towards a meaning of LIFE. *Journal of Logic Programming*, 16:195–234, 1993.
3. H. Aït-Kaci, A. Podelski, and S.C. Goldstein. Order-sorted feature theory unification. *Journal of Logic, Language and Information*, 30:99–124, 1997.
4. U. Callmeier. PET — a platform for experimentation with efficient HPSG processing techniques. *Journal of Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):99–108, 2000.
5. B. Carpenter. *The Logic of Typed Feature Structures – with applications to unification grammars, logic programs and constraint resolution.* Cambridge University Press, 1992.
6. L. Ciortuz. Towards ILP-based learning of attribute path values in typed-unification grammars. 2002. (Submitted).
7. L. Ciortuz. On compilation of head-corner bottom-up chart-based parsing with unification grammars. In *Proceedings of the IWPT 2001 International Workshop on Parsing Technologies*, pages 209–212, Beijing, China, October 17–19, 2001.
8. L. Ciortuz. LIGHT — a constraint language and compiler system for typed-unification grammars. In *Proceedings of the 25th German Conference on Artificial Intelligence (KI-2002)*, Aachen, Germany, September 16–20, 2002. Springer-Verlag.
9. A. Copestake, D. Flickinger, and I. Sag. *A Grammar of English in HPSG: Design and Implementations.* Stanford: CSLI Publications, 1999.
10. D. Gerdemann. Term encoding of typed feature structures. In *Proceedings of the 4th International Workshop on Parsing Technologies*, pages 89–97, Prague, Czech Republik, 1995.
11. R. M. Kaplan and J. Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–381. The MIT Press, 1982.
12. R. Malouf, J. Carroll, and A. Copestake. Efficient feature structure operations without compilation. *Journal of Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):29–46, 2000.
13. Y. Mitsuishi, K. Torisawa, and J. Tsujii. HPSG-Style Underspecified Japanese Grammar with Wide Coverage. In *Proceedings of the 17th International Conference on Computational Linguistics: COLING-98*, pages 867–880, 1998.
14. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.
15. Stefan Müller. *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche.* Number 394 in Linguistische Arbeiten. Max Niemeyer Verlag, Tübingen, 1999.
16. S. Oepen and J. Caroll. Performance profiling for parser engineering. *Journal of Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG: Methods, Systems, Evaluation):81–97, 2000.
17. S. Oepen and J. Carroll. Ambiguity packing in HPSG — practical results. In *Proceedings of the 1st Conference of the North American Chapter of the ACL*, pages 162–169, Seattle, WA, 2000.
18. F. Pereira. A structure-sharing representation for unification-based grammar formalisms. In *Proceedings of the 23rd meeting of the Association for Computational Linguistics*, pages 137–144, Chicago, Illinois, 1985.
19. C. Pollard and I. Sag. *Head-driven Phrase Structure Grammar.* Center for the Study of Language and Information, Stanford, 1994.
20. S. M. Shieber, H. Uszkoreit, F. C. Pereira, J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In J. Bresnan, editor, *Research on Interactive Acquisition and Use of Knowledge.* SRI International, Menlo Park, Calif., 1983.
21. M. Siegel. HPSG analysis of Japanese. In *Verbmobil: Foundations of Speech-to-Speech Translation.* Springer Verlag, 2000.
22. H. Tomabechi. Quasi-destructive graph unification with structure-sharing. In *Proceedings of COLING-92*, pages 440–446, Nantes, France, 1992.
23. H. Uszkoreit. Categorial Unification Grammar. In *International Conference on Computational Linguistics (COLING'92)*, pages 498–504, Nancy, France, 1986.
24. S. Wintner and N. Francez. Efficient implementation of unification-based grammars. *Journal of Language and Computation*, 1(1):53–92, 1999.