Conférence TALN 2000, Lausanne, 16-18 octobre 2000

Robust unification for linguistics

Frederik Fouvry

Seminar für Sprachwissenschaft
Eberhard-Karls-Universität Tübingen
Kleine Wilhelmstrasse 113
D-72074 Tübingen, Germany
fouvry@sfs.nphil.uni-tuebingen.de

Abstract

Unification- and constraint-based unification formalisms have been used widely to write natural-language grammars. There have nevertheless always been problems with the coverage of the grammars, due to too restrictive an interpretation of what the grammar described. Several approaches have been proposed, relaxing different requirements. Many of these however are *ad hoc* and rely on properties of a specific parser or architecture. In this paper, I present an approach that is based on a more abstract and general level, viz. the formalism. More concretely, I use a typed attribute logic, based on (Carpenter, 1992). In a first step it is ensured that the type hierarchy has certain properties to make a meaningful robustness possible (i.e. containing information about what went wrong). The size of such a hierarchy is limited through modulation. Then, a distance measure based on the logic is introduced to distinguish between different analyses. Reporting problems and treatment of unknown words fall out naturally from the formalism, and existing grammars and linguistic theories can be implemented or re-used without any specific adaptations for robustness like in other, non-robust, formalisms.

1. Linguistic theories and implementations

The formalisation of linguistic theories, like Lexical-Functional Grammar (LFG) (Dalrymple *et al.*, 1995) and Head-Driven Phrase Structure Grammar (HPSG) (Pollard & Sag, 1987; Pollard & Sag, 1994), by means of mathematical models has greatly improved the possibility to test them by using implementations. This has not only confirmed certain predictions on the linguistic side, but it has also shown that there is a general deficiency in dealing with unexpected, *extra-grammatical* input. In a constraint-based grammatical theory like HPSG, this becomes visible in parse failures, especially when the logical setup of the formalism is consistently pursued. For this reason, computational linguists have worked with weaker assumptions to ensure that at least some result is obtained. One of these assumptions is that the implementations are mostly *unification*-based: there are no satisfying computational accounts yet for *constraint*-based formalisms of that complexity. Some implementations of versions of these formalisms are the Attribute Logic Engine (ALE) (Carpenter & Penn, 1998), the Linguistic Knowledge Base (LKB)

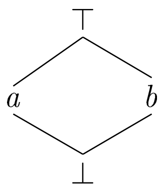


Figure 1: A simple type hierarchy

(Copestake, 1999), and ConTroll (Götz *et al.*, 1997). Another, nearly ubiquitous, technique is to use parsers that store intermediate analysis results (e.g. chart parsers). After the parse, these results are often put through a recovery module, as in (Jensen *et al.*, 1983). A problem with the partial results is that the part of the analysis where the problem was detected is not present in the final analysis (or analyses): applications that need that information, say grammar checkers, have to reproduce it. Statistical methods, a common approach to solve the problem of undergeneration (through relaxation of the rules) and overgeneration (through ranking of the results), cannot, as far as I am aware, handle arbitrary feature structures yet. In what follows, I present a method to re-interpret the typed formal framework of linguistic theories such that robustness and information about mistakes comes with the formalism, and does not require the development of specialised modules to deal with robustness.

2. Formal preliminaries

Following (Carpenter, 1992), an inheritance hierarchy or type hierarchy consists of a set of types and a relation: $\langle \mathsf{Type}, \sqsubseteq \rangle$. The relation $\sqsubseteq$, called the subsumption relation, defines a partial ordering on the types, expressing that some are more general than others. Usually there is an explicit most general type ($\bot$) and a most specific type ($\top$). If they are not present, it is easy to add them if necessary, because of their unique position: $\top$ for instance is more specific than all other elements in the hierarchy. $\top$ can be interpreted as $fail$, and $\bot$ as $true$. Given the above, it is obvious that $\bot \sqsubseteq \top$.

The unification of two types t and s, $t \sqcup s$, is defined as the most general type that is more specific than each of the two types being unified. It is important that the unification gives a unique result; many properties of the inheritance hierarchy disappear otherwise. For instance (see Figure 1), if we have a type a and the two aforementioned types $\top$ and $\bot$, $\bot \sqcup a = a$, and $\top \sqcup a = \top$. (This also shows that $\bot$ is the neutral element under $\sqcup$, and $\top$ the null element.) If we add a fourth type b, with $\bot \sqsubseteq b \sqsubseteq \top$, then $a \sqcup b = \top$. The relation between a and b is undefined: a and b are *incomparable*.

Carpenter furthermore requires that the type hierarchies should satisfy the condition that every two types that have common subtypes, have a *unique* most general common subtype (the bounded complete partial order (BCPO) condition); see Figure 2). The benefit of this is that the unification operation will only return one result. These hierarchies are also known as finite meet semi-lattices, which I define now.

Definition 1 (Lattice) *A non-empty partially ordered set* $\langle L, \sqsubseteq \rangle$ *is a lattice if*

- *for all* $a, b \in L : a \sqcup b$ *is defined;*
- *for all* $a, b \in L : a \sqcap b$ *is defined.*

Figure 2: The partially ordered set on the left is not a BCPO, because the unification of the black nodes is not unique. The partially ordered set on the right is a BCPO or meet semi-lattice. Note that the latter has one more node.

(The symbols can be replaced by others. Later I shall use $\vee$, $\wedge$, $\leq$ and $\cup$, $\cap$, $\subseteq$.) A partially ordered set that only satisfies the second condition, is called a *meet semi-lattice*, after the *meet* operation on lattices, which in this case is the generalisation operation (with a unique result). (The equivalent of unification is called *join*; join semi-lattices are partially ordered sets where only the first condition is satisfied.) The hierarchies I shall use are lattices, a proper subset of the set of the BCPOs.

3. Dealing with failure

3.1. Types

An instance of the basic appearance of the problem we have to overcome is this:

(1) $singular \sqcup plural = \top$

I.e. the unification of the types $singular$ and $plural$ fails. It would be a more interesting result when it not only tells us that there was a problem, but also *what* it was. Normally, unification failure reduces an entire feature structure to $\top$, regardless of all other values. With feature structures there are many values involved in each unification, each of which can cause a failure. In order to know what caused a failure, such *top-smashing* must be avoided: unification should always return a result different from the null element $\top$. Given that unification is defined in terms of subsumption, that relation (subsumption) must be expanded. For instance in example (1), $singular$ and $plural$ are (obviously) the offending couple. If we take another unification

(2) $third \sqcup second = \top$

then the result of this unification is the same as the result in (1), even though the unificands were completely different. The unificands cannot be recovered from the result. The information about failure that we are looking for is only described by the types that were involved in the (failing) unification.

To deal with that problem, I use lattices of sets.[1] Earlier, I have already defined lattices.

Definition 2 (Lattices of sets) *A lattice L is a* lattice of sets *if it is closed under finite unions and intersections.*

[1] I have borrowed the terminology and definitions from (Davey & Priestley, 1990).

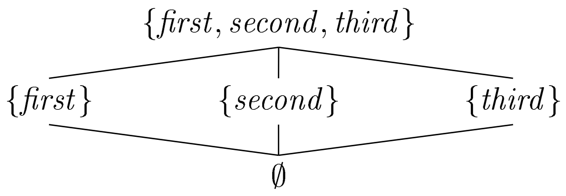


Figure 3: A lattice which is not complete

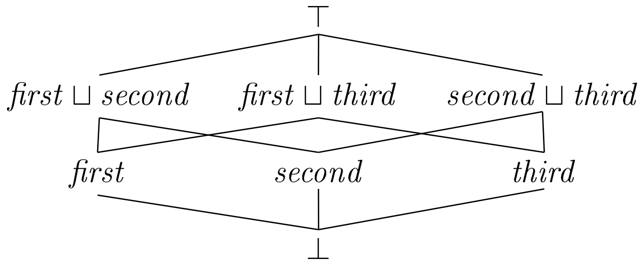


Figure 4: A lattice of sets

Since the hierarchies I work with are finite, the finiteness is not so strict a limitation.

The powerset of a set S for instance is a lattice of sets $\langle \mathcal{P}(S), \subseteq \rangle$. In Figure 3, a lattice is shown which is not a lattice of sets because $\{\mathit{first}, \mathit{second}\} \subseteq \{\mathit{first}, \mathit{second}, \mathit{third}\}$ but $\bigcup\{\mathit{first}, \mathit{second}\}$ is not a node in the lattice.

Lattices of sets are precisely the structures that wanted for this task. They require that every combination of elements in the set can be join-ed and gives a result which is again a member of the set. Because of the containment relation between the elements of the lattice, we know that an element of the set will only show up in the resulting subset when it has contributed to the result: third is not an element of $\{\mathit{first}, \mathit{second}\}$, so it does not occur in $\bigcup\{\mathit{first}, \mathit{second}\}$. It may seem obvious here, but it is a property I will use in less clear contexts.

It seems cumbersome to have to find a lattice of sets for every hierarchy one works with. However, *every* distributive lattice is isomorphic to a lattice of sets. In other words, when we know we have to do with a distributive lattice, it is certain that it has the desired properties. I define distributive lattices next.

Definition 3 (Distributive lattice) *A lattice L is distributive if the distributive laws hold over it:* $\forall a, b, c \in L : a \sqcup (b \sqcap c) = (a \sqcup b) \sqcap (a \sqcup c)$ *and* $\forall a, b, c \in L : a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$.

In fact, only one of the laws needs to be checked: if it holds, then the other one does as well. An example of a distributive lattice is shown in Figure 4. It is easy to check that it is isomorphic to a lattice of sets: the powerset of $\{\mathit{first}, \mathit{second}, \mathit{third}\}$ set out in a diagram and arranged by containment has exactly the same shape as this lattice. Going back to example (2), the result of the same unification but with the hierarchy displayed in figure 4, will give the result $\mathit{second} \sqcup \mathit{third}$, and the result is unique to that pair of types. If $\mathit{second} \sqcup \mathit{third}$ is then unified with the type first, the result is $\top$, or $\mathit{first} \sqcup \mathit{second} \sqcup \mathit{third}$.

The distributivity is not used during unification (since that is only one of the two operations, and distributivity says something the relation between both operations), but it provides information to recover from the error. That does not necessarily mean that the error can be corrected, since that involves much more, not in the least because errors are rarely as simple as the exam-

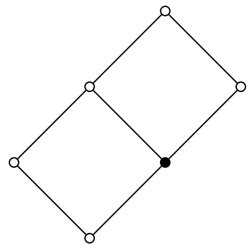


Figure 5: A robust lattice which is not Boolean

ples given here. In practice, the recovery information defines the most general analysis result that is conform *both* the grammar rules and the unchanged input phrase(s) given the lexicon. What exactly is done with the information is left to specific applications: one valid value may be chosen for correction, the generalisation of both values may be taken, ...

Some readers may wonder at this point if a Boolean lattice is not what is required, since the examples I have used always refer to Boolean lattices. A Boolean lattice is a distributive lattice with a top and a bottom in which every element a has a (unique) complement a', such that $a \sqcup a' = \top$ and $a \sqcap a' = \bot$. Complementation is however not necessary. This is clear from the hierarchy shown in Figure 5: the black node does not have a complement, but the hierarchy is distributive (and robust).

3.2. Partitioning of the type hierarchy

A type hierarchy which is expanded to a distributive lattice will grow very big. Fortunately, not all combinations of types are needed. Given a certain grammar, the type hierarchy can be divided into several partitions called *modules*. Such partitions have been discussed in among others (Aït-Kaci *et al.*, 1989) and (Fall, 1996). Types of one such partition never occur on the same attribute as a type of a different partition. (Should they do, it just means that they belonged to the same module in the first place.) A module is defined as a subset of types that has the same (subsumption) relation to the rest of the type hierarchy (often incomparable, but not necessarily so), in other words, seen from 'outside' the module, they are the equivalent. Inside the module, the types are related by the usual subsumption.

This gives a big gain in space needed to store the type information. Types that never occur together do not need to be encoded together either. The result could be that for an English grammar for instance the three values for person form a module on their own, which is clearly much smaller than a robust version of the entire type hierarchy.

With measures like these, I am not removing any robustness from the formalism. Types from different modules are never unified in a correct (albeit potentially incomplete) system. With such a setup, failure points to an implementation mistake; in a non-robust system, it might just as well have been replaced by an error. Having said this, it is possible to design a grammar where the entire hierarchy is one module, in which case modularisation is not of much use.

I have used modularisation in the implementation of the formalism. The compiler only compiles a module at a time into a Prolog term, and uses that as the value for attributes appropriate for that type (and its module).

3.3. Feature structures

The types (with attributes) are used to build up feature structures: every type may have attributes, which have typed values in turn. To define them, it is easier to think of feature structures as directed graphs. (Carpenter, 1992) defined them as a four-tuple $\langle Q, \overline{q}, \theta, \delta\rangle$, where Q is the set of nodes in the graph, $\overline{q}$ the root node (i.e. where the graphs begins), θ the function that assigns a type to a node, and δ the partial function that defines the (directed) paths (over features) from nodes to other nodes.

Just like between types, there is also a subsumption relation between feature structures.

Definition 4 (Feature structure subsumption) $F_1 = \langle Q_1, \overline{q}_1, \theta_1, \delta_1\rangle$ *subsumes* $F_2 = \langle Q_2, \overline{q}_2, \theta_2, \delta_2\rangle$ *if and only if there is a total function* $h : Q_1 \to Q_2$ *such that*

- $h(\overline{q}_1) = \overline{q}_2$
- $\forall q_1 \in Q_1 : \theta_1(q_1) \sqsubseteq \theta_2(h(q_1))$
- $h(\delta_1(f, q_1)) = \delta_2(f, h(q_1))$ *for every* $q_1 \in Q_1$ *and feature* f *such that* $\delta_1(f, q_1)$ *is defined.*

(Carpenter, 1992, 41)

This definition says that F_1 must be homomorph to a subgraph of F_2 that starts at the root $\overline{q}_2$, and that the types of the nodes in Q_1 must subsume the types of the homomorph nodes in Q_2.

Since the relaxation of the unification of types was entirely defined by using the subsumption relation, it is possible to do the same for feature structures. No additions need to be made to obtain robust feature structure unification, since the only source of failure comes from the types. (In PATR-II (Shieber *et al.*, 1983) for instance only atomic values could cause the feature structure unification to fail.)

The result of the unification in (3) is not $\top$ as before, but (4).

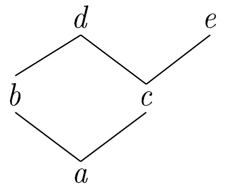
(3) $$\begin{bmatrix} \textit{arg} & \\ \text{NUMBER} & \textit{singular} \\ \text{PERSON} & \textit{third} \end{bmatrix} \sqcup \begin{bmatrix} \textit{arg} & \\ \text{NUMBER} & \textit{plural} \\ \text{PERSON} & \textit{second} \end{bmatrix}$$

(4) $$\begin{bmatrix} \textit{arg} & \\ \text{NUMBER} & \textit{singular} \sqcup \textit{plural} \\ \text{PERSON} & \textit{third} \sqcup \textit{second} \end{bmatrix}$$

The values of the two attributes cannot be simplified further because the values do not exist in the (original, non-robust) hierarchy.

4. Distance

Unification will now succeed on any input that still obeys the module distinctions. In order to allow systems like parsers to make real distinctions between different (robustly obtained) solutions, I introduce a distance measure between the types and between the feature structures. In line with the previous changes, the measure is entirely defined in terms of subsumption.



a	0				
b	1	0			
c	1	x	0		
d	3	1	1	0	
e	2	x	1	x	0
	a	b	c	d	e

Figure 6: Distance in the type hierarchy. On the lefthand side is the hierarchy, on the righthand side the table with the distances.

4.1. Types

The distance between two types is the number of nodes in the (non-robust) subsumption hierarchy that there is between the two types, i.e. the nodes subsumed by the most general one and subsuming the most specific one plus 1. There is no distance defined between incomparable types. An example is shown in Figure 6. Note the distance between d and a: d is a case of multiple inheritance.

Of course, in a lot of cases, some distance between incompatible types needs to be calculated. Then I use the *recovery distance* which is the distance between the generalisation of the type at hand and the expected type, and the type at hand. Contrary to the distance, the recovery distance is not commutative. The recovery distance for the hierarchy in Figure 6 is shown in Table 1. Because it is about recovery, it only concerns types which are incomparable. The only

a					
b			1		2
c		1			
d					1
e		1		1	
	a	b	c	d	e

Table 1: Recovery distance for hierarchy in Figure 6.

interesting case is the recovery distance between b and e, because that is an asymmetric result. This distance plays an important role because it indicates the ungrammaticality of analysis results: some feature structure is expected, and another one is to be checked. The recovery distance keeps track of how different they are. This happens throughout the analysis so that results only can get worse, or stay the same. This asymmetric situation (expected versus available) is reflected in the non-commutativity of the recovery distance.

It is finally also possible to define a correction distance, which is the sum of the recovery distances in both directions. This could be used to judge the goodness of a correction.

4.2. Feature structures

The distance definitions of types can be carried over to feature structures, with the extra requirement that for feature structures also the path length is counted. In other words, if the type value of G is not counted, and we assume the values of $\begin{bmatrix}\text{G} & a\end{bmatrix}$ and a to be the same, there is more information in (5) than in (6), just because of the extra depth.

(5) $\begin{bmatrix}\text{F} \mid \text{G} & a\end{bmatrix}$

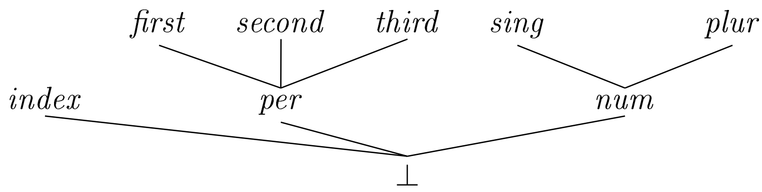


Figure 7: Type hierarchy for (7) and (8)

(6) $\begin{bmatrix} \textsc{f} & a \end{bmatrix}$

This path length difference does not need to be specified explicitly when the distance between feature structures is taken to be the distance in the subsumption hierarchy of feature structures. The distance between the types will be reflected in the feature structure subsumption hierarchy because the types are used to build up the feature structures. An example is shown now, using the hierarchy of Figure 7.

The following four feature structures describe two different situations. Between the first two, there is a recovery distance of 1, because they only differ in one value, the person. The value 1 can be derived from the type hierarchy.[2]

(7) $\begin{bmatrix} \textit{index} & \\ \textsc{per} & \textit{third} \\ \textsc{num} & \textit{sing} \end{bmatrix}$

(8) $\begin{bmatrix} \textit{index} & \\ \textsc{per} & \textit{second} \\ \textsc{num} & \textit{sing} \end{bmatrix}$

When however in the hierarchy *index* gets two subtypes, *3rdsing* and *non3rdsing*, with *3rdsing* constrained to be (9) and *non3rdsing* any other value, the recovery distance between them changes: there is now not only the distance between the PER values, but also between the two root types, which gives the total distance of 2.

(9) $\begin{bmatrix} \textit{3rdsing} & \\ \textsc{per} & \textit{third} \\ \textsc{num} & \textit{sing} \end{bmatrix}$

(10) $\begin{bmatrix} \textit{non3rdsing} & \\ \textsc{per} & \textit{second} \\ \textsc{num} & \textit{sing} \end{bmatrix}$

This shows that the distance of between feature structures can be manipulated by encoding things in the type hierarchy in a certain way. It may be considered to be a linguistically significant result: the distance between second plural and third singular in one language using the first encoding would be smaller than in another one using the second encoding. Which one is most appropriate is for the grammar writer to decide. In non-robust formalisms, there is a fairly great

[2] Strictly speaking I have to define the direction of the recovery distance, but it is the same in either direction.

freedom where new attributes are placed, and whether they are encoded as types or as features. With this distance measure, the formalism becomes more sensitive to differences. This also shows that the distance is not just any measure, but significant in the formalism.

5. Usage

In implementations of a theory like HPSG, rules (or principles) and lexical entries are all defined entirely in terms of feature structures based on the type hierarchy. When the grammar is compiled using the type hierarchy, the structures to keep track of the information required for robustness are incorporated (e.g. distance). Rule applications then consist of unifications. The feature structures involved in these unifications might be inconsistent. In all cases, assuming a correct implementation, the unification will succeed, as explained above, but it will entail a deterioration of the structure when the parser tries to unify two inconsistent feature structures, i.e. the (recovery) distance of the structure that was used (say an edge category in the chart) to the one that was expected (say the rule daughter as specified by the grammar writer) is not zero anymore. For every feature and value that is incompatible, the distance increases. The distance is also carried over between rule applications, such that it becomes monotonically larger within a parse. The parser can obviously make use of this information to choose the best candidates to proceed. For the final result, the best overall analyses are the ones with the smallest deterioration. Grammatical analyses will have no deterioration at all. Parsing will therefore always succeed, and return one or more feature structures with associated deterioration values.

The current parser is a bottom-up chart parser that uses heuristics to keep the number of solutions down. The heuristics consist of a subsumption check and a check on the deterioration value of the feature structure. Both are useful to remove results that have lost too much information. A time-out heuristic is applied for parses that take too long.

6. Other work

Approaches along similar lines of thinking have been published for artificial intelligence to deal with defaults and presuppositions (Belnap, 1977; Ginsberg, 1988; Fitting, 1989), and for linguistics (Schöter, 1995; Vogel & Cooper, 1995) under the name of *para-consistent* or *multi-valued* logic. The usage of defaults is less prominent in linguistics than in artificial intelligence. My proposal follows similar lines as that of (Vogel & Cooper, 1995)—it "extends" the type hierarchy to deal with problems—but they cannot handle certain types (for instance lists). They also mention the possibility of measuring discrepancy between feature structures, but do not propose any such concrete measure.

(Abney, 1994) and (Barg & Walther, 1998) also use techniques which are reminiscent of this, but much less extensive. Abney only works with a few features and uses bit vectors to record the values, and possible inconsistencies. Barg and Walther are mainly interested in automatic updating of a lexicon with unknown words. They need to be able to parse sentences containing new words, and then they use the resulting information to derive semantic and syntactic properties of those words.

(Bod & Kaplan, 1998) in the LFG-DOP framework focus on stochastic processing using a more complex linguistic formalism, but they also suggest to use the information distance between feature structures to deal with ill-formed sentences. Note that LFG is not a typed framework.

(Foster, 2000) is a recent approach where inconsistencies in the unification are recorded in the feature structure. Like in my proposal, unification never returns plain failure. Different is that the nodes of the feature structures are not merged if their values are not compatible. The changes are therefore not in the type hierarchy but in the feature structure unification algorithm.

7. Future work

The set of errors that can be treated in this way is limited to substitution errors. A feasible solution for word order changes would be an interesting extension.

An obvious follow-up to this work is to experiment with the weights through changes in the type hierarchies. It would also be interesting to see to what extent statistical techniques can be used and integrated in the architecture. Because of the way feature structures are composed from types, on which the distance measures are based, any distance measure between feature structures should not violate the subsumption hierarchy that exists between the feature structures.

Any effects of parsing with a distance measure might also have an effect on the shape of the inheritance hierarchy and the location of attributes. How big these changes would be remains to be seen.

8. Conclusion

The technique presented here provides the basis for a framework for robust analysis of natural language using modern linguistic theories. The unification that emerges from these changes is completely monotonic. It allows existing theories and descriptions of language to be compiled into grammars fit for 'real' use. It is a general, parser- and application-independent approach to robustness, the results of which can be used for diagnosing or even correcting errors, and at the same time offers the potential to use ungrammatical data in a positive way in linguistic argumentation.

Acknowledgements

This research was supported by a studentship from the University of Essex (United Kingdom). I also want to thank Anna Kupść for reading draft versions and two anonymous reviewers. Any remaining errors are my own.

References

ABNEY S. (1994). Partial parsing. Tutorial presented at ANLP 94, 11 October 1994, Stuttgart. On-line document: `http://www.sfs.nphil.uni-tuebingen.de/~abney/Abney_94j.2up.ps.gz` (13/11/1996).

AÏT-KACI H., BOYER R., LINCOLN P. & NASR R. (1989). Efficient implementation of lattice operations. *ACM Transactions on programming languages and systems*, **11**(1), 115–146. contains some errors according to project description of SFB340 B8.

BARG P. & WALTHER M. (1998). Processing unknown words in HPSG. In (COLING 98, 1998), p. 91–95. Also as cs.CL/9809106.

BELNAP N. (1977). A useful four-valued logic. In J. DUNN & G. EPSTEIN, Eds., *Modern uses of multiple-valued logic*, volume 2 of *Episteme*, chapter ?, p. 8–37.

BOD R. & KAPLAN R. (1998). A probabilistic corpus-driven model for lexical-functional analysis. In (COLING 98, 1998), p. 145–151. Also in (Butt & Holloway, 1998).

M. BUTT & T. HOLLOWAY, Eds. (1998). *Proceedings of the LFG 98 Conference*, Brisbane. University of Queensland. Published on-line at `http://www-csli.stanford.edu/publications` (`LFG3/lfg98.html`) (25 March 1999).

CARPENTER B. (1992). *The logic of typed feature structures. With applications to unification grammars, logic programs and constraint resolution*. Number 32 in Cambridge Tracts in Computer Science. Cambridge–New York–Melbourne.

CARPENTER B. & PENN G. (1998). ALE. The Attribute Logic Engine. User's guide. Version 3.1 Beta. Distributed as part of ALE.

COLING 98 (1998). *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Conference of the Association for Computational Linguistics*, Montreal, Canada. International Committee on Computational Linguistics.

COPESTAKE A. (1999). The (new) LKB system. `http://www-csli.stanford.edu/~aac/doc5-2.pdf`.

M. DALRYMPLE, R. M. KAPLAN, I. MAXWELL, JOHN T. & A. ZAENEN, Eds. (1995). *Formal issues in Lexical-Functional Grammar*. Number 47 in CSLI Lecture Notes. Stanford, California.

DAVEY B. A. & PRIESTLEY H. A. (1990). *Introduction to lattices and order*. Cambridge Mathematical Textbooks. Cambridge, second reprint (1992) of first edition.

FALL A. (1996). *Reasoning with Taxonomies*. PhD thesis, School of Computing Science, Simon Fraser University, Canada.

FITTING M. (1989). Negation as refutation. In R. PARIKH, Ed., *Proceedings. Fourth Annual Symposium on Logic in Computer Science*, p. 63–70: IEEE.

FOSTER J. (2000). Handling inconsistency with a new type of unification. In *ESSLLI 2000 Student session*, p.–-, Birmingham. In preparation as a technical report.

GINSBERG M. L. (1988). Multivalued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, **4**, 265–316.

GÖTZ T., MEURERS D. & GERDEMANN D. (1997). *The ConTroll Manual (ConTroll v.1.0β, XTroll v.5.0β)*. Seminar für Sprachwissenschaft, Universität Tübingen, Tübingen, Germany. Draft of September 17, 1997.

JENSEN K., HEIDORN G. E., MILLER L. A. & RAVIN Y. (1983). Parse fitting and prose fixing: getting a hold on ill-formedness. *American Journal of Computational Linguistics*, **9**(3–4), 147–160.

POLLARD C. & SAG I. A. (1987). *Information-based syntax and semantics*, volume 1: Fundamentals of *CSLI Lecture Notes*. Stanford.

POLLARD C. & SAG I. A. (1994). *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. Chicago–London–Stanford, first edition.

SCHÖTER A. (1995). Paraconsistent feature logic. In (Schöter & Vogel, 1995), chapter 1, p. 3–38.

A. SCHÖTER & C. VOGEL, Eds. (1995). *Nonclassical feature systems*, volume 10 of *Edinburgh Working Papers in Cognitive Science*. Centre for Cognitive Science, University of Edinburgh.

SHIEBER S., USZKOREIT H., PEREIRA F. C. N., ROBINSON J. J. & TYSON M. (1983). The formalism and implementation of PATR-II. In J. BRESNAN, Ed., *Research on interactive acquisition and use of knowledge. Final report [. . .], SRI Project 1894*, p. 39–79. SRI International, Artificial Intelligence Center, Menlo Park, Cal.

VOGEL C. & COOPER R. (1995). Robust chart parsing with mildly inconsistent feature structures. In (Schöter & Vogel, 1995), chapter 6, p. 197–216.