Visions for Logic-Based Natural Language Processing

Gregor Erbach
German Research Center
for Artificial Intelligence
Saarbrücken, Germany
erbach@dfki.uni-sb.de

Suresh Manandhar
Language Technology Group
Human Communication Research Centre
University of Edinburgh
skm@cogsci.ed.ac.uk

Abstract

The purpose of this paper is to outline the requirements that the field of Natural Language Processing (NLP) has for a declarative successor to Prolog. Logic programming approaches play a major role in NLP, and likewise considerations of NLP have had a strong influence on logic programming ([Colmerauer, 1978], [Pereira and Warren, 1980], [Smolka, 1992]).

We will review the state of the art in NL grammar formalisms and processing algorithms in the logic programming paradigm, from the perspective of research in grammatical theory (Computational Linguistics) and from the perspective of applied NLP (Language Engineering). The relationship to Prolog and to current developments in (C)LP will be discussed.

We will outline the requirements for a logic programming language that will both support the current practice in NLP and provide a platform which can accommodate future research results.

1 Motivation

The early development of Prolog and logic programming was strongly influenced by NLP applications (Colmerauer's Q-Systems and Metamorphosis Grammar), and DCGs have been part of Prolog from the very beginning. Today, logic programming is the most important programming paradigm for NLP, and the strong relationship between the two fields is evidenced by regular conferences (e.g., *Natural Language Understanding and Logic Programming*) and workshops (e.g., *Computational Logic and Natural Language Processing*, jointly organised by the European Network in Computational Logic and the European Network in Language and Speech).

A large part of the work in NLP is based on Prolog. A key idea is to express a grammar as a set of statements in a logic (e.g. Horn clauses), and to use a theorem prover (e.g. resolution) in order to parse or generate NL sentences. The original slogan *parsing as deduction* [Pereira and Warren, 1983] is nowadays extended to *bidirectional linguistic deduction* in which the same grammar can be employed both for parsing and generation. However, Prolog has turned out to be insufficient for the purposes of NLP for a number of reasons (inefficient search strategy, unexpressive constraint language). As a consequence, research in grammar formalisms has come up with a number of alternatives, extensions or specialisations of the design choices made in Prolog. One important point is that most grammar formalisms make a clear distinction between declarative specification (grammars) and execution (NL parsing or generation). The following section gives an overview of the state of the art in grammar formalisms, with special emphasis on those developments that are different from developments in logic programming (although there is a large overlap, where the fields have either influenced each other or the same kinds of approaches have been developed independently).[1]

[1]Due to space limitations, this paper can only give a very broad overview and references to the relevant literature. A more extensive overview of grammar formalisms is in preparation. Also, we concentrate the bibliographic references on the work in grammar formalisms since we expect the reader to be familiar with the relevant literature in logic programming.

2 State-of-the-art in grammar formalisms

Grammar formalisms are implementations which allow (largely) declarative specification of grammars together with a collection of tools such as parsers, generators, morphological processors *etc.* that make use of the grammar. Grammar formalisms provide means for detecting inconsistency within grammars at compile-time and means for debugging grammars at run-time.

For grammar formalisms there is a choice between expressing grammars in a general form as Horn clauses or in a special grammar rule notation. For instance, within the CUF formalism, grammar rules are not given any special status and are expressed as Horn clauses. On the other hand, within the ALE formalism, grammar rules are expressed as phrase-structure rules and a chart parser is employed. Usually, the former corresponds to the choice of a general deduction system, whereas the latter corresponds to the use of specialised parsers or generators (e.g. chart parser, semantic-head driven generator).

There is a convergence between these two approaches because often parsing algorithms can be generalised as deduction algorithms, and because methods from logic programming, such as constraint solving, are being imported into NL parsing and generation.

Also there is a trend in grammatical theory away from construction-specific rules towards general well-formedness conditions (grammatical principles), which can be naturally expressed as constraints that need to satisfied universally. This can be evidenced within the so called *principle-based* grammatical theories such as the *government-binding theory* [Chomsky, 1981; Lasnik, 1991] and HPSG [Pollard and Sag, 1994].

Constraint Logic Programming

In logic programming — following early work such as CHIP — and the development of the CLP scheme [Jaffar and Lassez, 1987] [Höhfeld and Smolka, 1988], Constraint Logic Programming has become an important paradigm and an area of much active research. This CLP scheme has been adopted in NLP in systems such as CUF [Dörre and Dorna, 1993]. The CLP scheme also permits the addition of new constraint solvers in addition to feature constraints and this approach is currently being experimented with in formalisms such as CL-ONE [Erbach *et al.*, 1995].

Typed feature formalisms

A lot of effort within the NLP community has been directed towards a class of grammar formalisms known as *typed feature formalisms*. Broadly speaking, typed feature formalisms are Horn-extensions over feature descriptions. They can loosely be understood as specific instantiations of the CLP(FT) paradigm where FT stands for a specific version of feature logic [Kasper and Rounds, 1986] [Smolka, 1992] with types and inheritance. Languages such as LOGIN and LIFE known within the logic programming community and systems such as ALE, CUF, TFS, ALEP known within the NLP community are examples of typed feature formalisms.

2.1 Resolution strategies

The search strategy of Prolog (top-down, left-to-right, depth-first, backtracking) has turned out to be problematic for natural language parsing and generation because of termination and efficiency problems.

As a consequence, a number of alternative deduction strategies have been developed for logic grammars, partly as genuine deduction algorithms, and partly as adaptations of existing parsing algorithms (such as *bottom-up* and *left-corner* parsing) for context-free grammars or of existing generation algorithms (such as *head-driven generation*).

For example, CUF is a formalism which follow a CLP scheme and employs the Andorra resolution strategy [Haridi and Janson, 1990], which prefers deterministic goal reduction. On the other hand, systems such as ALE employ a bottom-up chart parser.

Bottom-Up Approaches

Pure top-down resolution is often problematic for NLP because it leads to a large amount of search and to termination problems, especially for grammars which make use of discontinuous constituents. Therefore NL parsers and generators often make use of bottom-up methods, e.g., left-corner parsing and its compilation into a Prolog program (BUP) [Matsumoto *et al.*, 1983], semantic-head driven generation [Shieber *et al.*, 1990], head-corner parsing [van Noord, 1991]. The advantages of bottom-up and top-down processing are often combined by making use of a precompiled *reachability relation* [Kay, 1980].

In logic programming, bottom-up approaches are found for example in the area of deductive databases.

Compilation and program (grammar) transformation

Partial deduction (or partial execution) techniques have been used to transform a grammar (or program) into an equivalent, but more efficient, one in which some goals have been replaced by their defining clauses. Explanation-based learning is a partial deduction technique that has been used to speed up NL parsing and generation by replacing sequences of frequently occurring rule applications (inference steps) by one compiled rule [Neumann, 1994a; Samuelsson, 1994a].

Approaches derived from LR-parsing compile a grammar into a condition-action table which has the benefit that different rules (or clauses) with a common prefix don't lead to separate threads in the computation. LR-based techniques have been applied to parsing and generation [Samuelsson, 1994b; Samuelsson, 1995].

Termination problems can arise when the same program (logic grammar) is used for parsing and generation because the termination is dependent on the instantiation of arguments and the flow of information between different goals of a clause.

The *Essential Argument Algorithm* [Strzalkowski, 1991] distinguishes input and output arguments of DCGs, and uses re-ordering of goals within and across clauses in order to derive grammars that are specialised for parsing or generation, in the sense that the input arguments (the semantics in case of generation, and the surface string in case of parsing) are always guaranteed to be instantiated for the respective direction of processing.

Tabulation

Due to the amount of search involved in NLP, Prolog's backtracking search is rather inefficient, and tabulation (memoing) techniques should be used in order to store and re-use partial results.

The earliest application of tabulation in logic programming is Earley Deduction [Pereira and Warren, 1983], which adapts the basic idea of Earley's context-free parsing algorithm as a proof procedure for logic programs, and especially DCG parsing. Refinements include the restriction technique to avoid prediction loops [Shieber, 1985], the use of the algorithm for both parsing and generation through a modification of the indexing scheme [Shieber, 1988; Neumann, 1994b], extensions for the memoisation of coroutined constraints [Johnson and Dörre, 1995], and a bottom-up variant [Erbach, 1994a].

Only recently, memoisation techniques (OLDT resolution) have been incorporated into logic programming languages such as XSB Prolog [Warren, 1992]. This is an important development for NLP because traditional Prolog-based implementations of chart parsers suffer from a loss of efficiency due to the copying of structures when items are stored. Further research into more efficient memoisation schemes is probably needed.

2.2 Feature Structures

In computational linguistics, feature structures (attribute-value matrices) have for a long time been preferred over Prolog terms (e.g. PATR [Shieber *et al.*, 1983] or Lexical Functional Grammar [Kaplan and Bresnan, 1982]) because they allow a more perspicuous and compact encoding of linguistic knowledge. The encoding is more perspicuous because feature names indicate what the purpose of a particular value is, and it is more compact because values can be left unspecified by omitting features instead of introducing anonymous variables.

Feature structures can be *open* or *closed*. Open feature structures allow more features to be added to a structure, whereas closed feature structures allow only a fixed set of features. Closed feature structures are generally used in typed feature structures, where a fixed set of features is declared to be appropriate for a particular type. A good overview of feature constraint logics for NLP is given in [Smolka, 1992].

In logic programming, feature constraints have become fairly common. Following the pioneering work of Aït-Kaci [Aït-Kaci, 1984; Aït-Kaci and Nasr, 1986], they have been introduced into various logic programming languages. Efficient implementation methods exist for open and closed feature structures, and have been integrated in the Oz language [Smolka *et al.*, 1995].

2.3 Types and Typed Feature Structures

Types and inheritance between types are important devices for the hierarchical structuring of linguistic knowledge, and are widely used in influential linguistic theories such as head-driven phrase structure grammar [Pollard and Sag, 1994]. However, the notions of type (or sort) can mean different things in different grammatical formalisms, which can be a source of confusion.

Single inheritance is generally considered insufficient since it does not allow cross-classification of linguistic descriptions. Most typed feature systems generally provide some form of multiple inheritance. For instance, Carpenter's formalisation of typed feature structures [Carpenter, 1992] requires the inheritance lattice to be specified as a BCPO (bounded complete partial order). On the other hand, the ProFIT typed feature system allows either single inheritance or multi-dimensional inheritance which can be employed for cross-classification [Erbach, 1994b].

Some expressive formalisms, such as TDL [Krieger and Schäfer, 1994], use a more expressive type system with relational types, full negation, lazy type expansion etc. A system such as CUF allows type axioms to be stated using the full propositional calculus. However, CUF does not permit variables in type definitions.

The choice of an ideal type system for NLP is subject to debate and will ultimately depend on the needs of the representation of linguistic knowledge. At this stage, it would be premature to decide on one type system.

2.4 Higher-Order Unification

A standard approach for treating ellipsis in NLP makes crucial use of higher-order unification [Dalrymple *et al.*, 1991]. For example, in the sentence *Dan likes golf, and George does too* the first conjunct will have the logical form *like(dan,golf)*, and for the second, elliptical, conjunct we know that the property P predicated of Dan in the first conjunct holds of George.

$$P(dan) = like(dan,golf)$$

A possible value for P is the property $\lambda x.like(x,golf)$. Then $\lambda x.like(x,golf).george$ reduces to *like(george,golf)*.

2.5 Constraints

Different types of constraints play an important role in grammatical descriptions. Equality or co-reference constraints are most important within computational linguistics, and have led to the designation of a whole class of grammar formalisms as *Unification Grammars*, which are the forerunners of typed feature formalisms.

Inequality constraints (cf. *dif* in Prolog II or Sicstus Prolog) are for example needed to model aspects of the binding theory, e.g. in the sentence *John saw him*, the pronoun *him* cannot be coreferent with the NP *John*.

NL processing demands specialised constraint processing facilities for both high-level declarative encodings of grammars and for efficient processing. However, as opposed to specialised constraint solving in the numeric domain which appears to be the trend within constraint logic programming evidenced by formalisms such as CLP(R), CLP(Q) and CLP(FD), NL processing usually demands specialised constraint processing in the symbolic domain.

Symbolic constraints such as *set descriptions and set operations* [Manandhar, 1994], *linear precedence constraints* [Manandhar, 1995], *guarded constraints over these constraints* [Erbach *et al.*, 1995], *tree descriptions* [Vijay-Shanker, 1992] are among those that have been investigated.

Set descriptions permit multiple values for the same feature symbol in the same way as knowledge representation languages such as KL-ONE [Brachman and Schmolze, 1985][2]. Set constraints are widely used in linguistic theories to model phenomena whose handling involves storing and passing information about possibly large collections of certain linguistic objects such as subcategorisation frames, discourse referents, quantifiers, extraposed or fronted constituents etc. A set constraint language has been implemented which makes use of membership constraints, cardinality constraints, disjointness constraints, and union and intersection constraints. A sound and complete constraint solver for these constraints has been implemented [Manandhar, 1994].

Linear precedence constraints allow an underspecified representation of the order among the elements of a set [Manandhar, 1995]. Unlike lists, where the order is completely fixed, or sets, where the order is completely open, linear precedence constraints allow to specify which elements must stand in a precedence relation. These constraints are used for modelling partially free word order in languages such as German or Bulgarian, and also for underspecified representation of quantifier scoping possibilities.

With the introduction of attributed variables [Holzbaur, 1992], it has become possible to implement constraint solvers in systems such as Sicstus 3. This mechanism is very useful for NLP because it permits the experimentation with and application of new kinds of constraints solvers (e.g. the handling of grammatical principles as type constraints [Matiasek, 1994], or morphology as a constraint between surface and lexical forms [Trost and Matiasek, 1994]).

2.6 Disjunction

Disjunction handling is an important problem for NLP due to the lexical and structural ambiguity inherent in natural language [Trost, 1993]. The creation of choice points for these cases of ambiguity would lead to serious efficiency problems.

There are two approaches other than backtracking to disjunction handling in grammar formalisms: distributed disjunctions and finite domains.

Distributed Disjunction

Distributed disjunctions [Dörre and Eisele, 1990; Böttcher, 1993; Maxwell III and Kaplan, 1991] implement a full disjunction handling by avoiding expansion to disjunctive normal form by pushing disjunctions as deep into the feature structures as possible. In order to deal with the fact that the choice in one disjunct requires a corresponding choice in another disjunct, the disjuncts are given names, and the choice of one named disjunct leads to the choice of the disjunct with the same name in another part of the structure.

An example is the relationship between particular syntactic forms and semantic readings, as in German where the combination of the preposition *in* with an dative NP has a locative reading (English: *in*), whereas the combination with an accusative NP leads to a directional reading (English: *into*).

Compared to the relatively inefficient implementation of distributed disjunctions on top of Prolog, their re-implementation in C, which is used in the XEROX Language Engineering system (XLE), is very efficient.

[2] See [Manandhar, 1993] for a comparison between feature logics and terminological logics employed in knowledge representation languages such as KL-ONE.

Finite Domains

Finite domains are useful where disjunctions are not distributed over different parts of the structure. A finite domain variable is a variable which can take on a finite set of ground values. As an example, take the following finite domain which consists of the possible combination of the agreement features NUMBER and PERSON in English:

$$\{[1st,sg],[2nd,sg],[3rd,sg],[1st,pl],[2nd,pl],[3rd,pl]\}$$

A word form such as *takes* can simply be described as taking a subject with the agreement value [3rd,sg], but for the form *take*, there are five different possibilities:

$$\{[1st,sg],[2nd,sg],[1st,pl],[2nd,pl],[3rd,pl]\}$$

In this case, the use of finite domains can avoid the creation of choice points and contribute to efficient processing. In Prolog-based systems such as ProFIT [Erbach, 1995], finite domains with N possible values are implemented by compilation into a term with $N+1$ arguments [Mellish, 1988; Mellish, 1992].

Underspecification

Underspecification techniques have been used to avoid disjunctions by constructing type hierarchies or term representations, in which one term can stand for a set of values and a choice among these values takes place by further instantiation of the term. Underspecification techniques are especially important for problems in semantic representation such as the encoding of quantifier scoping possibilities [Alshawi and Crouch, 1992].

2.7 Lean Formalisms

The implementation of typed feature structures, constraints, disjunction handling, and alternative deduction strategies on top of a high-level language such as Prolog leads to a serious loss of efficiency, which hinders the development of NLP applications which make use of these formalisms. In order to overcome these problems, some recent formalisms (e.g. the Core Language Engine CLE [Alshawi, 1991], the Advanced Linguistic Engineering Platform ALEP [Alshawi *et al.*, 1991], ProFIT [Erbach, 1995]) have made use of a compilation of typed feature structures into Prolog terms, so that Prolog's term unification can be used at runtime. These systems have been successfully used for representing linguistic knowledge for NLP applications. However, these systems do not allow currently support multiple inheritance.

In order to handle multiple inheritance, the Attribute Logic Engine ALE [Carpenter and Penn, 1994] therefore takes a different approach by compiling typed feature structure unification into a set of primitive instructions [Carpenter, 1993]. This approach is currently implemented on top of Prolog by making use of open data structures and efficient indexing techniques, but a re-implementation in a lower-level language using an abstract machine approach is planned in order to speed up the system.

Lean formalisms are important for NLP systems to enable implementations of large-scale systems and for downsizing research prototypes into commercial products.

2.8 Template Notation

Template definitions provide the possibility to give names to terms, and to refer to frequently occurring terms by that name. For example the template *np* can stand for the complex term representing a noun phrase. Templates can have arguments, and an n-place template is equivalent to an $n+1$-place relation. While templates are a logically redundant notation, they are considered important by grammar writers because they permit the definition of complex terms in one place and refer to them by their name on all occurrences, which ensures transparency and facilitates the maintenance of a grammar.

2.9 Preference-driven Constraint Satisfaction

In order to deal with the ambiguous readings which occur very often with NL (for which humans seem to have no difficulty), newer processing models make use of a preference-driven best-first search strategy (where the preferences are acquired using a statistical model).

Within CLP based systems, this can be implemented through a dynamic goal selection function which for a selected atom chooses clauses with the highest preference first [Uszkoreit, 1991]. In future language processing systems such best-first selection strategies are likely to play an important role.

An equally important problem is the handling of ill-formed (ungrammatical) input as it occurs in applied systems. In order to handle this problem, where not all constraints of the grammar can be satisfied in parsing a sentence, future constraint-based linguistic processing will require dynamic and preferential constraint satisfaction. Thus research in areas such as *hierarchical constraint logic programming* [Wilson and Borning, 1993] has potential applications for such NLP tasks. HCLP can be thought as an extension of the CLP(X) paradigm to CLP(X,S) where X represents the constraint domain and S represents the *comparator* or *selection function* that can handle preferences in case of conflicts.

3 Requirements of NLP for a programming language

In this section, we outline the requirements for a declarative successor to Prolog from the perspective of the needs of Natural Language Processing. Within NLP, there are two separate kinds of activities:

1. The development of formal models of NL syntax and semantics (Computational Linguistics)
2. The implementation of NLP components for applied systems (Language Engineering)

Both kinds of activities are closely related, and make use of the same declarative core, but need different kinds of support services from a programming language:

For *Computational Linguistics*, it is important to have the best support for developing, testing, debugging grammars and for formalising and evaluating linguistic theories. The emphasis on developing linguistic descriptions, expressing generalisations about linguistic phenomena, and experimenting with new linguistic theories and new descriptive devices imposes the following requirements on a logic programming language, which can support the building of tools for these activities:

- declarativeness
- an expressive constraint language with a convenient notation
- the possibility to add new kinds of constraint solvers
- tools for debugging and visualisation (e.g. Tcl/TK interface, as it is provided in Sicstus and Oz).

For *Language Engineering*, on the other hand, computational efficiency is a very important consideration because systems must either respond with a minimum of waiting time (dialog systems) or be able to handle large amounts of data (text understanding, message extraction).

A second important consideration for Language Engineering is the need to make a selection among the large number of possible solutions of a particular parsing or generation task. The chosen solution should be the preferred (most probable) reading in the case of parsing, or the sentence that is optimal for achieving the desired communicative effect in the case of generation.

While it is not yet clear what the optimal model for ranking the possible solutions is, there is currently a strong trend towards statistical language models (which often still use very primitive models such as bigram or trigram statistics). In Computational Linguistics research, this new trend is also reflected in the shift from competence models to performance models [Uszkoreit, 1991]. A programming language for natural language must support research into the integration of declarative logical models and statistical language models.

In summary, a programming language which supports Computational Linguistics research and the development of applied NLP systems must satisfy the following requirements:

- **Declarative Core**
  1. feature constraints (in addition to terms), cf. LIFE, Oz.
  2. finite domains (not only over integers)
  3. higher-order unification and higher-order programming
  4. support for different type systems
- **Resolution Strategies**
  1. support for preference-driven constraint satisfaction
  2. efficient disjunction handling
  3. the possibility to add new types of constraints (such as sets, trees, linear precedence, regular expressions)
  4. support for different search strategies (bottom-up, head-driven, best-first etc.)
  5. tabulation support
- **Support Services**
  1. support for 'programming in the large' (e.g., modules, integration with imperative, object-oriented and functional programming)
  2. embeddability into other systems (e.g., database support for large lexicons, compilation to machine-code which can be linked with other system components)

As a whole, the programming language should provide efficient implementations of the basic services, such as feature constraints, tabulation, disjunction, or search engines, so that a grammar formalism can be built on top of it without a lot of computational overhead, and on the other hand be flexible enough to offer the possibility to pursue interesting research directions, e.g., in the combination of declarative with statistical language models, or the addition of new types of constraints.

Acknowledgements

This work was supported by research grants from the German Bildungsministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF) to the DFKI project PARADICE, FKZ ITW 9002 0 (for G. Erbach) and by the Language Technology Group, Human Communication Research Centre, University of Edinburgh (for S. Manandahar). Both authors particiapte in the Grammar Formalisms Working Group of the *Expert Advisory Group on Language Engineering Standards* (EAGLES), which is supported by the Commission of the European Communities (LRE-1 61100).

We wish to thank the anonymous reviewers of the paper, as well as Günter Neumann, for their detailed and constructive comments.

References

[Aït-Kaci and Nasr, 1986] Hassan Aït-Kaci and Roger Nasr. LOGIN: A logical programming language with built-in inheritance. In *Proceedings of the 13th ACM Symposium an Principles of Programming Languages*, pages 219–228, St. Petersburg, Florida, 1986.

[Aït-Kaci, 1984] Hassan Aït-Kaci. *A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially-Ordered Type Structures.* PhD thesis, University of Pennsylvania, 1984.

[Alshawi and Crouch, 1992] Hiyan Alshawi and Richard Crouch. Monotonic semantic interpretation. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 32 – 39, Newark, Delaware, 1992.

[Alshawi *et al.*, 1991] H. Alshawi, D. J. Arnold, R. Backofen, D. M. Carter, J. Lindop, K. Netter, J. Tsujii, and H. Uszkoreit. Eurotra 6/1: Rule formalism and virtual machine design study — Final report. Technical report, SRI International, Cambridge, 1991.

[Alshawi, 1991] Hiyan Alshawi, editor. *The Core Language Engine.* MIT Press, 1991.

[Böttcher, 1993] Martin Böttcher. Disjunctions and inheritance in the context feature structure system. In *EACL93*, pages 54 – 60, Utrecht, NL, 1993.

[Brachman and Schmolze, 1985] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Carpenter and Penn, 1994] Bob Carpenter and Gerald Penn. *The Attribute Logic Engine: User's Guide (Version 2.0.1).* Carnegie-Mellon University, Pittsburgh, December 1994.

[Carpenter, 1992] Bob Carpenter. *The logic of typed feature structures.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, 1992.

[Carpenter, 1993] Bob Carpenter. Compiling typed attribute-value logic grammars. In *Third International Workshop on Parsing Technologies*, pages 39 – 48, Tilburg, NL and Durbuy, B, 1993.

[Chomsky, 1981] Noam Chomsky. *Lectures on Government and Binding.* Studies in Generative Grammar. Foris, Dordrecht, 1981.

[Colmerauer, 1978] Alain Colmerauer. Metamorphosis grammar. In Leonard Bolc, editor, *Natural Language Communication with Computers.* Springer, Berlin, 1978.

[Dalrymple *et al.*, 1991] Mary Dalrymple, Stuart M. Shieber, and Fernando C. N. Pereira. Ellipsis and higher-order unification. *Linguistics and Philosophy*, 14(4):399–452, 1991.

[Dörre and Dorna, 1993] Jochen Dörre and Michael Dorna. CUF – A formalism for linguistic knowledge representation. In Jochen Dörre, editor, *Computational Aspects of Constraint-Based Linguistic Description. Deliverable R1.2.A.* DYANA-2 – ESPRIT Basic Research Project 6852, 1993.

[Dörre and Eisele, 1990] Jochen Dörre and Andreas Eisele. Feature logic with disjunctive unification. In Hans Karlgren, editor, *Coling 90: Papers presented to the the Thirteenth International Conference on Computational Linguistics*, pages 100–105, Helsinki, Finland, 1990. Helsinki University.

[Erbach *et al.*, 1995] G. Erbach, M. van der Kraan, S. Manandhar, H. Ruessink, W. Skut, and C. Thiersch. Extending unification formalisms. In *2nd Language Engineering Convention*, London, 1995.

[Erbach, 1994a] Gregor Erbach. Bottom-up Earley Deduction. In *COLING*, pages 796 – 802, Kyoto, 1994.

[Erbach, 1994b] Gregor Erbach. Multi-dimensional inheritance. In H. Trost, editor, *Proceedings of KONVENS '94*, pages 102 – 111, Vienna, 1994. Springer.

[Erbach, 1995] Gregor Erbach. ProFIT: Prolog with features, inheritance and templates. In *Seventh Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, Dublin, 1995.

[Haridi and Janson, 1990] Seif Haridi and Sverker Janson. Kernel Andorra Prolog and its computation model. In *International Conference in Logic Programming (ICLP)*, pages 31–46, 1990.

[Höhfeld and Smolka, 1988] Markus Höhfeld and Gert Smolka. Definite relations over constraint languages. LILOG Report 53, IBM Deutschland, Stuttgart, October 1988.

[Holzbaur, 1992] C. Holzbaur. DMCAI CLP reference manual. Technical Report TR-92-24, Österreichisches Forschungsinstitut für Artificial Intelligence, Vienna, 1992.

[Jaffar and Lassez, 1987] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proceedings 14th POPL*, 1987.

[Johnson and Dörre, 1995] Mark Johnson and Jochen Dörre. Memoization of coroutined constraints. In *Proceedings of the 32rd Annual Meeting of the Association for Computational Linguistics*, Cambridge, 1995. (Computation and Language e-print archive cmp-lg/9504028).

[Kaplan and Bresnan, 1982] Ronald Kaplan and Joan Bresnan. Lexical Functional Grammar: a formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, chapter 4, pages 173 – 281. MIT Press, Cambridge, MA, 1982.

[Kasper and Rounds, 1986] Robert Kasper and William Rounds. A logical semantics for feature structures. In *24th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York*, pages 257–265, 1986.

[Kay, 1980] Martin Kay. Algorithm schemata and data structures in syntactic processing. Technical report, Xerox PARC, Palo Alto, CA, October, 1980 1980.

[Krieger and Schäfer, 1994] Hans-Ulrich Krieger and Ulrich Schäfer. $\mathcal{TDL}$—a type description language for constraint-based grammars. In *Proceedings of the 15th International Conference on Computational Linguistics*, Kyoto, 1994.

[Lasnik, 1991] Noam Chomsky & Howard Lasnik. Principles and parameters theory. In *et al.* J. Jacobs, editor, *Syntax: an international handbook of contemporary research*. Walter de Gruyter, Berlin, (to appear) edition, Dec. 1991.

[Manandhar, 1993] Suresh Manandhar. *Relational Extensions to Feature Logic: Applications to Constraint Based Grammars*. PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1993.

[Manandhar, 1994] Suresh Manandhar. An attributive logic of set descriptions and set operations. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, pages 255 – 262, Las Cruces, NM, 1994.

[Manandhar, 1995] Suresh Manandhar. Deterministic consistency checking of LP constraints. In *Proceedings of the 7th European Conference of the Association for Computational Linguistics*, Dublin, 1995.

[Matiasek, 1994] Johannes Matiasek. *Principle-based Processing of Natural Language using CLP Techniques*. PhD thesis, Technische Universität, Vienna, October 1994.

[Matsumoto *et al.*, 1983] Yuji Matsumoto, Hozumi Tanaka, H. Hirakawa, Hideo Miyoshi, and Hideki Yasukawa. BUP: a bottom-up parser embedded in Prolog. *New Generation Computing*, 1:145–158, 1983.

[Maxwell III and Kaplan, 1991] John T. Maxwell III and Ronald M. Kaplan. A method for disjunctive constraint satisfaction. In Masaru Tomita, editor, *Current issues in parsing technology*, pages 173 – 190. Kluwer, Boston, 1991.

[Mellish, 1988] Christopher S. Mellish. Implementing systemic classification by unification. *Computational Linguistics*, 14(1):40–51, 1988.

[Mellish, 1992] Christopher S. Mellish. Term-encodable description spaces. In D. R. Brough, editor, *Logic Programming: New Frontiers*, pages 189 – 207. Intellect, Oxford, 1992.

[Neumann, 1994a] Günter Neumann. Application of explanation-based learning for efficient processing of constraint-based grammars. In *Proceedings of the Tenth IEEE Conference on Artificial Intelligence for Applications*, pages 208–215, San Antonio, Texas, March 1994.

[Neumann, 1994b] Günter Neumann. *A Uniform Computational Model for Natural Language Parsing and Generation*. PhD thesis, Universität des Saarlandes, Saarbrücken, 1994.

[Pereira and Warren, 1980] Fernando C.N. Pereira and David H.D. Warren. Definite Clause Grammars for language analysis - a survey of the formalism and a comparison with Augmented Transition Networks. *Artificial Intelligence*, 13(3):231–278, 1980.

[Pereira and Warren, 1983] Fernando C.N. Pereira and David H.D. Warren. Parsing as deduction. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 137–144, 1983.

[Pollard and Sag, 1994] Carl Pollard and Ivan Sag. *Head-driven Phrase Structure Grammar*. University of Chicago Press, Chicago, 1994.

[Samuelsson, 1994a] Christer Samuelsson. *Fast Natural-Language Parsing Using Explanation-Based Learning*. PhD thesis, The Royal Institute of Technology and Stockholm University, Stockholm, Feb 1994.

[Samuelsson, 1994b] Christer Samuelsson. Notes on LR parser design. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 386 – 390, Kyoto, 1994.

[Samuelsson, 1995] Christer Samuelsson. An efficient algorithm for surface generation. In *IJCAI 95*, Montreal, 1995.

[Shieber *et al.*, 1983] Stuart M. Shieber, Hans Uszkoreit, Fernando C. N. Pereira, Jane J. Robinson, and Mabry Tyson. The formalism and implementation of PATR-II. In Barbara J. Grosz and Mark E. Stickel, editors, *Research on Interactive Acquisition and Use of Knowledge*, pages 39–79. SRI International, Menlo Park, CA, 1983.

[Shieber *et al.*, 1990] Stuart M. Shieber, Gertjan van Noord, Fernando C. N. Pereira, and R. C. Moore. Semantic-head-driven generation. *Computational Linguistics*, 16(1):30–42, 1990.

[Shieber, 1985] Stuart M. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 145–152, 1985.

[Shieber, 1988] Stuart M. Shieber. A uniform architecture for parsing and generation. In *COLING-88*, pages 614–619, 1988.

[Smolka *et al.*, 1995] Gert Smolka, Martin Henz, and Jörg Würtz. Object-oriented concurrent constraint programming in Oz. In P. van Hentenryck and V. Saraswat, editors, *Principles and Practice of Constraint Programming*, chapter 2, pages 27–48. The MIT Press, 1995.

[Smolka, 1992] Gert Smolka. Feature constraint logics for unification grammars. *Journal of Logic Programming*, 12:51 – 87, 1992.

[Strzalkowski, 1991] Tomek Strzalkowski. A general computational method for grammar inversion. In Tomek Strzalkowski, editor, *ACL SIG workshop "Reversible Grammar in Natural Language Processing"*, pages 91 – 99, Berkeley, CA, 1991.

[Trost and Matiasek, 1994] Harald Trost and Johannes Matiasek. Morphology with a null-interface. In *Proceedings of the 15th International Conference on Computational Linguistics*, Kyoto, Japan, 1994.

[Trost, 1993] Harald Trost, editor. *Feature Formalisms and Linguistic Ambiguity.* Ellis Horwood, Chichester, 1993.

[Uszkoreit, 1991] Hans Uszkoreit. Strategies for adding control information to declarative grammars. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 237–245, Berkeley, CA, 1991.

[van Noord, 1991] Gertjan van Noord. Head corner parsing for discontinuous constituency. In *29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, 1991.

[Vijay-Shanker, 1992] K Vijay-Shanker. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, pages 519 – 530, 1992.

[Warren, 1992] David S. Warren. Memoing for logic programs. *Communications of the ACM*, 35(3):93 – 111, 1992.

[Wilson and Borning, 1993] Molly Wilson and Alan Borning. Hierarchical constraint logic programming. Technical Report 93-01-02a, Department of Computer Science and Engineering, University of Washington, May 1993.