The Logic of Typed Feature Structures

Bob Carpenter
(Carnegie Mellon University)

Cambridge, England: Cambridge
University Press (Cambridge Tracts in
Computer Science 32, edited by
C. J. van Rijsbergen), 1992, viii +
270 pp.
Hardbound, ISBN 0-521-41932-8, $34.95

Reviewed by
Fernando Pereira
AT&T Bell Laboratories

1. Introduction

For those of us who belonged to the "Bay Area (Computational) Linguistics Community," the early eighties were a heady time. Local researchers working on linguistics, computational linguistics, and logic programming were investigating notions of category, type, feature, term, and partial specification that appeared to converge to a powerful new approach for describing (linguistic) objects and their relationships by monotonic accumulation of constraints between their features. The seed notions had almost independently arisen in generalized phrase structure grammar (GPSG) (Gazdar et al. 1985), lexical-functional grammar (LFG) (Bresnan and Kaplan 1982), functional-unification grammar (FUG) (Kay 1985), logic programming (Colmerauer 1978, Pereira and Warren 1980), and terminological reasoning systems (Aït-Kaci 1984). It took, however, a lot of experimental and theoretical work to identify precisely what the core notions were, how particular systems related to the core notions, and what were the most illuminating mathematical accounts of that core. The development of the unification-based formalism PATR-II (Shieber 1984) was an early step toward the definition of the core, but its mathematical analysis, and the clarification of the connections between the various systems, are only now coming to a reasonable closure. *The Logic of Typed Feature Structures* is the first monograph that brings all the main theoretical ideas into one place where they can be related and compared in a unified setting. Carpenter's book touches most of the crucial questions of the developments during the decade, provides proofs for central results, and reaches right up to the edge of current research in the field. These contributions alone make it an indispensable compendium for the researcher or graduate student working on constraint-based grammatical formalisms, and they also make it a very useful reference work for researchers in object-oriented databases and logic programming.

Having discharged the main obligation of the reviewer of saying who should read the book under review and why, I will now survey each of the book's four parts while raising some more general questions impinging on the whole book as they arise from the discussion of each part.

2. Basics

From the beginning, Carpenter emphasizes the strong links between attribute-value formalisms in computational linguistics and in knowledge representation (KR). This

is a welcome conceptual connection. Historically, however, the two strands developed fairly independently of each other. Aït-Kaci's dissertation (1984) arose from an attempt to define a computationally tractable core of inheritance and frame-based reasoning, but its relevance to the analysis of linguistic categories was not appreciated as early as it should have been. Interestingly, systemic and dependency grammars had the lead in bringing inheritance and featural classification notions together on the linguistic side, but their influence in the particulars of the linguistic formalisms under consideration was slight, if any. Inheritance reasoning played no direct role in LFG, GPSG, PATR-II, or logic grammars, and it came into play first as a lexicon organization discipline (Flickinger, Pollard, and Wasow 1985; Sheiber 1985), not as a central part of the formalism.

The organization of the first part of the book follows naturally from the emphasis on KR ideas. Types and inheritance are discussed first, followed by feature (attribute–value) structures and the relations and operations that they inherit from the underlying type system: subsumption and join (unification). The last introductory chapter addresses in detail the crucial move of Kasper and Rounds (1986) to clarify the meaning of feature structures by viewing them as models of appropriately chosen modal logics.

2.1 Feature Structures and Feature Logics

The fruitful connection between feature structures and feature logics is pursued throughout the book, with soundness and completeness results for the basic system and all the major extensions and variations considered later. If something is missing in that comprehensive development, it might be some effort to relate feature logics to modal logics, and feature structures to modal frames. I believe that the original Kasper–Rounds logic was to some extent inspired by modal logics of concurrency, in particular the modal Hennessy–Milner logic for CCS (Hennessy and Milner 1985). It has also been argued that the connection to modal logic is an important route for easier and more general proofs of the required normal form and completeness results (Blackburn 1991).

2.2 Representations versus Algorithms

The introductory part establishes the algebraic, denotational semantics orientation of the book, and throughout the book, the more computational aspects of feature logics receive little attention. In purely conjunctive feature logics such as those arising from PATR-II, there is a simple connection between formulas and models. An almost linear satisfiability procedure, based on the UNION-FIND algorithm (Aho, Hopcroft, and Ullman 1976, Aït-Kaci 1984, Jaffar 1984), can be used to build the unique most-general feature structure satisfying a formula. There is thus relatively little to say about computational complexity (but not about practical computation costs, as will be observed below when rational unification is discussed), and the algebraic approach is direct and instructive. When we move to more-expressive feature logics, however, the situation changes radically. There are no longer unique most-general models, the algebraic approach becomes more labored, and satisfiability becomes NP-hard or worse. Computational complexity results were a central part of the development of the more-expressive feature logics by Rounds, Kasper, Moshier (Kasper and Rounds 1986, Moshier and Rounds 1987) and others, but they are barely mentioned in Carpenter's book. One feels that those results, being of a more traditional (finite) model-theoretic character, may have been left out because they do not fit the book's algebraic plan.

2.3 Feature Logic Intractability and Natural Language Processing

A more general point arises from the computational issues just discussed. The intractability of satisfiability for expressive feature logics might seem a serious roadblock in the practical application of those logics in grammatical description systems. Two escape routes, one pragmatic and the other more radical, suggest themselves. The pragmatic route involves trying to identify more tractable subcases that may cover most of the situations of interest in actual grammatical description. Such "optimistic" algorithms were already suggested in Kasper's dissertation (1987), and have been extensively investigated since (Maxwell and Kaplan 1989; Eisele and Dörre 1988).

The more radical route, which as far as I know has not been pursued vigorously, looks more closely at the search control aspects of language processing systems based on feature computations. It takes conjunctive description as the only one that can have global extent in a computation. Nonconjunctive aspects of a description are then ephemeral in that nonconjunctive connectives introduced in a derivation must be eliminated within a bounded number of steps by a committed choice operation based on some preference-based search control mechanism. Such a view can be seen as a mild generalization of the ideas of deterministic parsing (Marcus 1980), and also closely related to the flat-guard committed choice logic programming languages (Ueda 1987; Saraswat 1990). In both of those frameworks a single conjunctive constraint is constructed incrementally on the basis of local committed choices among alternatives. Search completeness is of course sacrificed, but the computational intractability arising from having to consider all the combinations of smaller alternative constraints into larger consistent constraints is bypassed. Finally, the radical route suggests a discipline of trying to replace as much as possible disjunctive or negative constraints by somewhat weaker kinds of underspecification that admit of purely conjunctive formulations. That program was already suggested in the context of deterministic parsing (d-theory) (Marcus, Hindle, and Fleck 1983), and more recently in the context of incremental monotonic semantic interpretation (Alshawi and Crouch 1992), and might also be profitably employed in the more abstract feature-logic setting.

2.4 Prerequisites

I am well aware of the difficulties in determining what should be taken as a reasonable common background for readers in an interdisciplinary topic. There is little enough commonality in the theoretical backgrounds of computer scientists trained in different schools, and the common background becomes even more difficult to find when one wants to reach also theoretical linguists and AI researchers. Still, the introductory chapters, including their historical portions, seem to assume more than is strictly necessary, and in fact sometimes seem to assume what is later explained in the text in careful detail. This kind of forward reference might confuse readers as to what the book's prerequisites are, and what they should know as a matter of course. This is especially the case with respect to concepts of domain theory such as complete partial orders, conditional completeness, and powerdomains, which the great majority of potential readers (even, I believe, many U.S.-trained theoretical computer scientists) will not be familiar with. The early mentions will thus be confusing to them, even though there is later in the book a good introduction to those prerequisites. Another instance is the repeated mentions to intensionality and extensionality before their careful discussion in Chapter 8. Those have simply too many (admittedly related) meanings for the reader who would most benefit from the book to grasp what they refer to in feature logics before the in-depth discussion.

2.5 Quibbles

2.5.1 Up or Down? Following much of the literature on feature structures, Carpenter adopts the domain-theoretic convention that places more-specific objects "higher up" in informational partial orders. This conflicts with the conventions of model theory and knowledge representation, and leads to occasionally distracting dissonances in notation, terminology, and figures—for instance, inheritance hierarchies with the most specific elements at the top.

2.5.2 Abstract Feature Structures and Path Congruences. The discussion of abstract feature structures raises a historical difficulty. While I do not dispute that the full theoretical investigation of feature structures modulo renaming is correctly attributed to Moshier, the idea of representing renaming classes by equivalence relations over paths seems an obvious variant of the representation of such classes as deductively closed sets of path equations in Pereira and Shieber's account (1984) of the semantics of PATR-II, which is further explored in Shieber's dissertation (1989).

2.5.3 Unification Tradeoffs. The discussion of the tradeoffs between acyclic and rational term unification at various points in the book might be a bit misleading. The original Prolog used a weakened pointer-based version of Robinson's (1965) unification algorithm (conceivably attributable to Boyer and Moore) without the occurs check. Removing the occurs check, which blocks the binding of a variable to a term containing the variable, from Robinson's algorithm allows cyclic unifiers to be built. This not only changes the interpretation of unification, but is also a source of potential nontermination when cyclic unifiers are applied. Nevertheless, in the early development of Prolog the occurs check was seen as too costly to be involved in the basic computational step of a programming language, and few if any examples were known in which the lack caused problems for knowledgeable Prolog programmers. The development of linear acyclic unification algorithms such as Paterson and Wegman's (1978) or Martelli and Montanari's (1982) did not change that assessment. Those algorithms require far heavier data structures and constant factors than Prolog's unification, they do not interface well with Prolog's backtracking control regime, and, most importantly, they are linear on the *sum* of the sizes of the terms involved. In contrast, for most practical purposes, Prolog's algorithm is linear on the size of the *smaller* term involved, which depends only on program size and not on the length of the computation. This was crucial for the acceptance of Prolog as a programming language, since it was felt that the cost of a procedure call in a reasonable programming language should not depend on the sizes of the actual parameters. In Prolog II, Colmerauer and his colleagues side-stepped the main weakness of Prolog's unification, nontermination, by moving to rational term unification, which also has added representational value for certain applications (although for other applications, particularly those derived from theorem proving, only acyclic unification makes sense). The best rational term unification algorithms are almost linear in all cases, and may be linear on the size of the smaller term in the same cases as Prolog's algorithm. However, the data structure complexity and constant factors are still higher than in Prolog's algorithm, and the interaction with backtracking is less straightforward.

3. Extensions

The second part of the book concerns extensions and specializations: acyclic feature structures, type constraints, inequations, extensionality, and groundedness. I found most interesting in this part the very thorough accounts of type constraints and of

inequations. With type constraints restricting what features are appropriate for a type (so, for instance, an agreement feature is only appropriate for types of phrases that are subject to agreement constraints, and must yield a value of appropriate agreement type), we move decisively beyond what was provided by all earlier formalisms with the exception of GPSG (which was limited in other ways). Type constraints support good engineering practice in writing large systems such as wide-coverage grammars. Furthermore, in certain cases type information can lead to more efficient implementation. If the set of features for each type can be determined at compile time, the normal open-ended attribute–value representation of features can be replaced by the kind of positional representation used for record structures in programming languages such as C.

Carpenter starts the discussion of inequations from the Prolog II inequation (`dif`) mechanism (Colmerauer 1986), and extends it elegantly to feature logics. The simplicity of the account shows that the earlier exposition was carefully orchestrated to allow extensions and alterations of the core framework without major upheavals.

3.1 Extensionality

I was somewhat less happy about the chapters on extensionality and groundedness. That material seems less definitive, and indeed various points of the discussion are confusing or unclearly targeted.

There are conceptual and formal reasons for taking seriously the extensionality question. Different researchers in the field started with different intuitions of feature structures, with different identity conditions. GPSG categories, for example, were seen purely extensionally as labeled trees. As the area developed, mismatches between pointer-based implementations of feature structures and conceptual choices, and failures of completeness for various feature logics, pushed for increasing intensionalization. However, Carpenter goes directly into technical aspects of extensionality without much attention to the examples and intuitions that brought the question forward in the first place. It would have been better if alternative feature-structure models, for instance various tree and domain-theoretic models, had been compared with respect to their computational and logical implications, even if they were to be ultimately discarded in favor of the now standard DFA models. As it is, the reader must turn elsewhere, for instance Shieber's (1985) monograph, for a broader comparative analysis of feature models.

As a minor problem related to the above, the discussion of feature structures as a solution for a (domain) equation over partial functions seems unclear as to whether that is the most intensional model or the most extensional one (which would seem to be the case).

The relationship between extensionality and Prolog II unification is hinted at repeatedly, but its computational implications are not discussed. The differences in extensionality of feature structures and Prolog II terms are directly reflected in the differences between the corresponding unification algorithms. Feature-structure unification requires the identification of *all* corresponding feature-structure nodes, while term unification (leaving aside issues of computational complexity and termination in certain algorithms) only needs to install pointers from leaf (variable) nodes to corresponding nodes (Jaffar 1984).

Other algorithmic connections are not noted either, such as that between feature structure collapsing and DFA minimization. Finally, issues of extensionality and individuation may be most important for object-oriented databases, but that application is not discussed.

4. Alternatives

The third part of the book, named "Alternatives," is really an introduction to technical tools needed in later applications. Variables and assignments add nothing to the power of previous systems, but are convenient when discussing grammars and in another form were historically important in Aït-Kaci's system. Feature algebras, on the other hand, simplify and generalize radically certain mathematical arguments about feature structures. In fact, they might have been introduced sooner in the text to improve conceptual unity and eliminate some repetition in proofs.

4.1 Domain Theory

The last chapter of "Alternatives" discusses infinite feature structures and their formalization through domains. While the topic is potentially important for rounding out the theory of feature structures and the sketch of domain theory is for the most part on target, one wonders again whether the uninitiated reader will not stumble on references to notions that are discussed only later or not at all. For instance, compactness is mentioned informally before its definition, without suitable intuitions being provided. Scott's information systems are mentioned, without definition, although they are quite relevant to the material at hand, particularly abstract feature structures. And some of the proofs are too sketchy for a reader who presumably is not yet familiar with typical argument patterns in domain theory. The chapter concludes with the suggestive comment that a formalization of feature structures in terms of abstract closure operators on domains would eliminate the repetitiveness of completeness proofs for feature logics. One wishes the suggestion had been tested in the book, although one might also wonder whether the full apparatus of domain theory would be needed to take advantage of the convenience of closure operators. After all, closure operators arise naturally in logic from the notions of deductive closure and of logical consequence (Tarski 1983), so one might imagine that the simpler proofs could be carried out in a model-theoretic setting short of domain theory.

5. Applications

The last part of the book applies the theory developed earlier in three important areas: the semantics of unification-based phrase structure formalisms, the semantics of feature-based definite clause programs, and the specification of recursive type constraints.

5.1 Semantics of Grammar Formalisms

Carpenter's account of the denotational semantics of unification-based phrase structure grammars benefits greatly from the extensive use of feature algebras and feature-algebra morphisms to connect derivation steps. Earlier treatments were much less perspicuous, because they were based on complex encodings of phrase-structure rules as feature structures and of derivation steps as formal manipulations on rule encodings (Pereira and Shieber 1984; Shieber 1984; Rounds and Manaster-Ramer 1987).

As a minor terminological point, the qualifier *unification-based* used here is somewhat unfortunate, because unification is just a particular constraint-solving algorithm applicable for certain kinds of constraint-based grammars. The term *constraint-based grammar* is both less biased and more appropriate to modern formalisms in which unification is only one of several constraint-solving methods. Historically, neither LFG nor GPSG were originally thought of in terms of unification. GPSG features and feature constraints were seen as abbreviatory conventions for large collections of context-

free terminal categories (Gazdar 1982). LFG F-structures were seen as the result of a congruence-closure equation-solving process after a sentence was fully analyzed into constituents (C-structures; Bresnan and Kaplan 1982). Even the term *unification* in *functional unification* grammar was chosen by Martin Kay as intuitively suggestive, and not by analogy with Robinson's notion of unification.

Constraint-based grammar formalisms would not have gained the attention they did if they did not have practical parsing and generation algorithms. As is the case for programming languages, the impetus for giving a sound denotational semantics to those formalisms arose in part from the need to prove the correctness of particular implementation methods. However, Carpenter concentrates only in giving the denotational semantics for a typical formalism, and does not show its correspondence to its operational realization. Proofs of equivalence between denotational and operational semantics are useful not only as examples of what needs to be done to show the correctness of a parsing or generation algorithm, but also for the insights they give on the connections between the semantics of constraint-based formalisms, of logic programs, and of traditional formal language representations. The reader interested in those aspects will have to turn elsewhere, especially again to Shieber's monograph (1985).

5.2 Logic Programs and Recursive Types

Carpenter's semantics of constraint-based grammars extends straightforwardly to the form of definite-clause programming in Aït-Kaci and Nasr's (1986) LOGIN language, although one might have hoped for a bit more information on the connection to constraint logic programming.

The formalization of recursive type constraints, which were first introduced in Aït-Kaci's dissertation, is more challenging. Carpenter clarifies and completes Aït-Kaci's work, and relates it nicely to the computational interpretation of the constraint-based grammatical formalism, HPSG (Pollard and Sag 1987).

6. Details

The book is remarkably free of editorial errors, which can be particularly confusing but difficult to catch in a mathematical text. Here are a few problems that seem to have slipped through and could confuse the reader momentarily. The **agr** type seems to be missing in the Conc set (13) for the example of Figure 2.11. In Definition 4.2, and in a few other places, the convention that $x = y$ is intended to mean x and y are both defined and equal seems to be used without comment, but in other places the definedness is explicitly stated. On page 130, first sentence, the reference must be to "Prolog II and Prolog III", not to "Prolog II and Prolog II." On page 170, paragraph before Lemma 12.6, the first sentence should read "Note that even for countably based domains, there may be *an uncountably infinite* number of domain objects." The term "most-general morphism" used in definition 13.14 was not defined anywhere that I could find, although there is some mention of pointwise ordering of morphisms (but are there lubs in the order?). There seems to be something wrong in Definition 15.13. I believe $G@\pi$ should be G_π, where G_π is the result of resolving F along path π. Finally, the initial point in the discussion of fan-out resolution in the limit on page 244 should be F_0, not F_i.

7. Conclusion

I believe that *The Logic of Typed Feature Structures* is essential for any practicing or prospective researcher on feature-based grammar or knowledge representation formalisms and also very useful to researchers or graduate students in the grammar formalisms area of computational linguistics. Nowhere else can one find all the main mathematical analysis tools related to each other and all the central results carefully proved. Many readers, however, will need to come equipped with the support of a careful instructor or an attentive reading of a good introduction to the mathematical theory of partial orders, for instance, Davey and Priestley's (1990) *Introduction to Lattices and Order*. And those readers interested in the complexity of decision procedures for feature logics or in implementing systems based on them will have to look elsewhere for detailed algorithmic descriptions and complexity analyses of operations on feature structures and formulas. Carpenter's book is more in the European tradition that emphasizes algebraic models for formalisms than in the American tradition of complexity analyses for deductive procedures. Both are important. *The Logic of Typed Feature Structures* is the first systematic mapping of the landscape of feature logics, but many of the underlying processes and mechanisms still await an equally adept analysis.

Acknowledgments

David Israel made several useful suggestions on content and form, and Daniel Pereira helped simplify and clarify the prose. All remaining errors, obscurities, and biases are, of course, my own.

References

Aho, A. V.; Hopcroft, J. E.; and Ullman, J. D. (1976). *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

Aït-Kaci, Hasan (1984). *A lattice theoretic approach to computation based on a calculus of partially ordered type structures*. Doctoral dissertation, University of Pennsylvania, Philadelphia, PA.

Aït-Kaci, Hasan, and Nasr, R. (1986). "LOGIN: A logic programming language with built-in inheritance." *Logic Programming*, 3(3), 185–217.

Alshawi, Hiyan, and Crouch, R. (1992). "Monotonic semantic interpretation." In *Proceedings, 30th Annual Meeting of the Association for Computational Linguistics*. Newark DE, 32–39.

Blackburn, P. (1991). "Modal logic and attribute value structures." In *Colloquium on Modal Logic*, edited by M. de Rijke. Dutch Network for Language, Logic and Information.

Bresnan, Joan, and Kaplan, Ronald (1982). "Lexical-functional grammar: A formal system for grammatical representation." In *The Mental Representation of Grammatical Relations*, edited by Joan Bresnan, 173–281. The MIT Press.

Colmerauer, Alain (1978). "Metamorphosis grammars." In *Natural Language Communication with Computers*, edited by L. Bolc. Springer-Verlag. (First appeared as "Les grammaires de metamorphose," groupe d'intelligence artificielle, Université de Marseille II, November 1975.)

Colmerauer, Alain (1986). "Theoretical model of Prolog II." In *Logic Programming and its Applications*, edited by M. van Caneghen and David H. D. Wane. Ablex Series in Artificial Intelligence, 3–31. Ablex.

Davey, B. A., and Priestley, H. A. (1990). *Introduction to Lattices and Order*. Cambridge University Press.

Eisele, A., and Dörre, J. (1988). "Unification of disjunctive feature descriptions." In *Proceedings, 26th Annual Meeting of the Association for Computational Linguistics*. Buffalo NY, 286–294.

Flickinger, Dan; Pollard, Carl; and Wasow, Thomas (1985). "Structure-sharing in lexical representation." In *Proceedings, 23rd Annual Meeting of the Association for Computational Linguistics*. Chicago IL, 262–267.

Gazdar, Gerald (1982). "Phrase structure grammar." In *The Nature of Syntactic Representation*, edited by Pauline Jacobson and Geoffrey K. Pullum, 131–186. D. Reidel.

Gazdar, Gerald; Klein, Ewan; Pullum, Geoffrey K.; and Sag, Ivan (1985).

Generalized Phrase Structure Grammar. Harvard University Press.

Hennessy, M., and Milner, R. (1985). "Algebraic laws for nondeterminism and concurrency." *Journal of the Association for Computing Machinery*, 32(1), 137–161.

Jaffar, J. (1984). "Efficient unification over infinite terms." *New Generation Computing*, 2(3), 207–219.

Kasper, Robert T. (1987). *Feature structures: A logical theory with application to language analysis*. Doctoral dissertation, University of Michigan, Ann Arbor, Michigan.

Kasper, Robert T., and Rounds, William C. (1986) "A logical semantics for feature structures." In *Proceedings, 24th Annual Meeting of the Association for Computational Linguistics*. New York, 257–266.

Kay, Martin (1985). "Parsing in functional unification grammar." In *Natural Language Parsing*, edited by David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky, 251–278. Cambridge University Press.

Marcus, Mitchell P. (1980). *A Theory of Syntactic Recognition for Natural Language*. The MIT Press.

Marcus, Mitchell P.; Hindle, Donald; and Fleck, Margaret (1983). "D-theory: Talking about talking about trees." In *Proceedings, 21st Annual Meeting of the Association for Computational Linguistics*. Cambridge MA.

Martelli, A., and Montanari, U. (1982). "An efficient unification algorithm." *ACM Transactions on Programming Languages and Systems*, 4(2), 258–282.

Maxwell, J. T. III, and Kaplan, Ronald M. (1989). "An overview of disjunctive constraint satisfaction." In *Proceedings, First International Workshop on Parsing Technology*, edited by Masaru Tomita. Pittsburgh PA.

Moshier, M. D., and Rounds, William C. (1987). "A logic for partially specified data structures." In *ACM Symposium on the Principles of Programming Languages*. Munich, Germany.

Paterson, M. S., and Wegman, M. N. (1978). "Linear unification." *Journal of Computer and Systems Sciences*, 16(2), 158–167.

Pereira, Fernando C., and Shieber, Stuart M. (1984). "The semantics of grammar formalisms seen as computer languages." In *Proceedings of 1984 International Computational Linguistics Conference*. Stanford CA, 123–129.

Pereira, Fernando C., and Warren, David H. D. (1980). "Definite clause grammars for language analysis—A survey of the formalism and a comparison with augmented transition networks." *Artificial Intelligence*, 13, 231–278.

Pollard, Carl, and Sag, Ivan (1987). *Information-Based Syntax and Semantics, Volume I: Fundamentals*, Lecture notes 13. Center for the Study of Language and Information, Stanford CA.

Robinson, J. (1965). "A machine-oriented logic based on the resolution principle." *Journal of the Association for Computational Machinery*, 12(1), 23–44.

Rounds, William C., and Manaster-Ramer, Alexis (1987). "A logical version of functional grammar." In *Proceedings, 25th Annual Meeting of the Association for Computational Linguistics*. Stanford CA, 89–96.

Saraswat, V. A. (1990). "JANUS: A step towards distributed constraint programming." In *Logic Programming: Proceedings of the 1990 North American Conference*, edited by S. Debray and M. Hermenegildo, 431–446. The MIT Press.

Shieber, Stuart M. (1984). "The design of a computer language for linguistic information." In *Proceedings of 1984 International Computational Linguistics Conference*. Stanford CA, 362–366.

Shieber, Stuart M. (1985). *An Introduction to Unification-Based Approaches to Grammar*, Lecture notes 4. Center for the Study of Language and Information, Stanford, CA.

Shieber, Stuart M. (1989). *Parsing and type inference for natural and computer languages*. Doctoral dissertation, Department of Computer Science, Stanford University, Stanford CA.

Shieber, Stuart M. (1992). *Constraint-Based Grammar Formalisms*. The MIT Press.

Tarski, A. (1983). *Logic, Semantics, Metamathematics*, Second edition. Hackett Publishing Company.

Ueda, K. (1987). "Guarded Horn clauses." In *Concurrent Prolog: Collected papers*, edited by Ehud Shapiro, 140–156. The MIT Press.

Fernando Pereira is president of the Association for Computational Linguistics. Pereira's address is: AT&T Bell Laboratories, 2D-447, 600 Mountain Avenue, PO Box 636, Murray Hill, NJ 07974-0636; e-mail: pereira@research.att.com.