Charts, interaction-free grammars, and the compact representation of ambiguity

Marc Dymetman
Rank Xerox Research Center
6, chemin de Maupertuis
38240 Meylan, France

Marc.Dymetman@grenoble.rxrc.xerox.com

Abstract

Recently researchers working in the LFG framework have proposed algorithms for taking advantage of the implicit context-free components of a unification grammar [Maxwell and Kaplan, 1996]. This paper clarifies the mathematical foundations of these techniques, provides a uniform framework in which they can be formally studied and eliminates the need for special purpose runtime data-structures recording ambiguity. The paper posits the identity: *Ambiguous Feature Structures = Grammars,* which states that (finitely) ambiguous representations are best seen as unification grammars of a certain type, here called "interaction-free" grammars, which generate in a backtrack-free way each of the feature structures subsumed by the ambiguous representation. This work extends a line of research [Billot and Lang, 1989; Lang, 1994] which stresses the connection between charts and grammars: a chart can be seen as a specialization of the reference grammar for a given input string. We show how this specialization grammar can be transformed into an interaction-free form which has the same practicality as a listing of the individual solutions, but is produced in less time and space.

1 Introduction

Chart-parsing is a well-known technique for representing compactly the multiple parses of a CF grammar. If n is the input string length, the chart can register all the parses in $0(n^3)$ space, although there may be an exponential number of them. Each parse can then be recovered in linear time from the chart.

Chart-parsing can be extended to CF-based unification grammar formalisms such as LFG or DCGs. In this case, however, the valid parses of the input string cannot be recovered so easily from the chart. Interaction between feature structures can in theory lead to np-complete complexity: printing the first valid parse may require exponential time.

Such complex interactions are however rare in natural language. There is growing agreement among researchers about the "mild context-sensitiveness" of natural language [Joshi. 1985; Vijay-Shanker and Weir, 1994]. This means that NL grammars deviate only minimally in complexity from the context-free class. Thus, although NL grammars written in a unification formalism may appear superficially to be highly context-sensitive, most of the interactions between features tend to be local, so that many of the unification constraints could in principle be replaced by fine-grain context-free rules.

Recently researchers working in the LFG framework have proposed algorithms for taking advantage of the implicit context-free components of a unification grammar. Several related algorithms have been proposed, which rely on a notion of "disjunctive lazy copy links", a form of information propagation in feature structures which is only triggered in case of possible interactions between features [Maxwell and Kaplan, 1996].

This paper clarifies the mathematical foundations of these techniques, provides a uniform framework in which they can be formally studied and eliminates the need for special purpose runtime data-structures recording ambiguity. The paper posits the identity:

Ambiguous Feature Structures = Grammars,

which states that (finitely) ambiguous representations are best seen as unification grammars of a certain type, here called "interaction-free" grammars, which generate in a backtrack-free way each of the feature structures subsumed by the ambiguous representation. This work extends a line of research [Billot and Lang, 1989; Lang, 1994] which stresses the connection between charts and grammars: a chart can be seen as a specialization of the reference grammar for a given input string. We show how this specialization grammar can be transformed into an interaction-free form which has the same practicality as a listing of the individual solutions, but is produced in less time and space.

The paper proceeds in the following way:

- Charts can be seen as grammar specializations. The context-free case is presented first, then the case of CF-based unification grammars;
- The rhs of rules can be *standardized:* the unifications explicitly appearing in the rhs can be standardized to a normal form;
- The notion of a *interaction-free,* or *IF.* grammar is introduced; A unification grammar is called IF when its standardized rules have a certain property which guarantees absence of conflict when expanding the rhs nonterminals.
- The chart corresponding to a given input string is generally a non-IF grammar. An algorithm which transforms this grammar into an equivalent IF grammar is introduced.

2 Charts

Charts as grammar specializations For a CFG in Chomsky Normal Form (binary rules), and for an input string of length n, a chart can be built in time $0(n^3)$ and space $0(n^2)$ which recognizes whether the string belongs to the language associated with the grammar. If not only *recognition,* but also *parsing* of the string is required, then it is convenient, during the bottom-up construction of the chart, to associate with each edge the collection of combinations of daughter edges from which this edge can be obtained. The augmented chart then requires $0(n^3)$ space, but then each parse tree can be recovered in a trivial way by starting from the top edge and following top-down the links between mother and daughter edges. In this way, an exponential number of parse trees can be represented in polynomial space.

It has been remarked in [Billot and Lang, 1989] that an augmented chart for a CFG G, given the input string a. can be viewed as a context-free grammar G_α, generating only a, possibly more than once. Each mother edge *A* is seen as a nonterminal, and each combination of daughter edges < *B,C* > associated with *A* is seen as the rhs *B C* of a rule whose lhs is A This rule corresponds to a specific instance of some rule of *G*. Each top-down traversal of G_α generates a parse tree for α which is also a parse tree relative to the full grammar *G*. We will call the grammar G_α a *specialization* of *G* for the given input string.[1]

[1] **Charts applied to FSAs** More generally, it is possible to directly extend chart-parsing, with the same polynomial bounds in time and space, to the situation where the input string of words is generalized to any finite-state automaton *FSA*. Chart edges are constructed in the usual way, but they now connect automaton nodes rather than positions in the input string. The chart constructed over *FSA* can then be seen as a CFG *GFSA* , a specialization of *G,* which generates the *intersection* of the regular language associated with *FSA* and the CF language associated with *G* [Lang, 1994]. Thus chart-parsing is connected to the closure of context-free languages under intersection with a regular language, and the original proof of this closure property [Bar-Hillel *et al,* 1961] can be seen as a forerunner (as well as an extension!) of chart-parsing.

Charts and unification Chart-parsing methods extend naturally to unification grammars based on a context-free backbone, such as LFG [Kaplan and Bresnan, 1982] or DCG [Pereira and Warren, 1980]. For ease of exposition, we will use a DCG-like notation, but we believe the results to be applicable to any CF-based unification grammar.

Assuming a grammar with binary branching rules, any grammar rule can be written in one of the following forms:

$$R: \quad \mathtt{a(A)} \rightarrow \mathtt{b(B)\ c(C)}\ \ U_R(\mathtt{A,B,C})$$

for a non-lexical rule *R,* and:

$$S: \quad \mathtt{a(A)} \rightarrow \mathtt{[t]}\ \ U_S(\mathtt{A})$$

for a lexical rule 5. Nonterminals are written as lowercase letters, terminals under brackets, and uppercase letters are variables representing feature structures. The notation U_R(A, B, C) is an abbreviation for the set of unification constraints relating the structures A, 8 and C that appears in rule *R*.

For such a grammar, one can construct a chart/grammar specialization for a given input string[2] in the following way:

- One constructs a chart for the CF backbone of the grammar as in the previous section; This chart can be seen as a specialization of the CF-backbone.
- Each non-lexical rule

  $$R: \quad \mathtt{a} \rightarrow \mathtt{b\ c}$$

  of the CF-backbone specialization can be seen as a specialization of a rule

  $$R': \quad \mathtt{a'} \rightarrow \mathtt{b'\ c'}$$

  of the CF-backbone. where the nonterminals a', b', c' are specializations of the nonterminals a, b, c (that is, a' is the instance of the nonterminal a covering a certain specific substring of the input).
- Each such rule *R* is augmented into:

  $$R: \quad \mathtt{a(A)} \rightarrow \mathtt{b(B)\ c(C)}\ \ U_{R'}(\mathtt{A,B,C})$$

  where A, B, C are fresh variables, and where $U_{R'}$ is the set of unifications associated with *R'* in the original grammar.
- A similar operation is performed for the lexical rules.

The unification grammar obtained by this process is an extension of the chart/grammar obtained for the CF-backbone. It is a specialization of the original unification grammar, which is equivalent to this grammar as far as the given input string is concerned. If one uses the specialization grammar in a generation mode, expanding nonterminals top-down and

[2] Or, more generally, any input FSA.

"collecting" the unification constraints, one obtains sets of constraints which collectively describe, *when they are jointly satisfiable,* feature structures associated with the initial symbol s of the grammar.

The specialization grammar accepts at most the given input string. Determining whether it *does accept* this string can however still be a difficult problem. Two cases can occur:

1. The chart/grammar for the CF-backbone contains cycles, that is, there exists some nonterminal *A (or* equivalently, edge) in this grammar which calls itself recursively. This can only happen when the CF-backbone is an infinitely ambiguous CFG, or, in other words, if the given unification grammar is not *offline-parsable* [Pereira and Warren, 1983]; Offline-parsability is guaranteed under certain conditions, such as the fact that the CF-backbone does not contain any chain production A → B or empty production A → ε [Kaplan and Bresnan, 1982].

   When there are cycles in the chart, determining whether there exists a traversal of the (unification) specialization grammar which has satisfiable constraints is in general undecidable.

2. The full grammar is offline-parsable. Then the chart/grammar for the CF-backbone is acyclic. In this case, there are only a finite number of top-down traversal of the (unification) specialization grammar. For each of the traversals, one can perform unification of the collected constraints. Each traversal for which the constraints are satisfiable gives rise to a feature structure solution for the input string (or, more precisely, to a "most general unifier" in Prolog terminology, or a "minimal feature structure" in LFG terminology).

The second case is by far the most important in grammars naturally occurring in NLP. The recognition problem is then decidable, and all the feature structure solutions can be enumerated in finite time. However, there may be an exponential number of these solutions, and it can be shown that, in general, the problem of determining whether a solution exists is np-complete in the length of the input string (it is easy to simulate a boolean satisfaction problem with a non-cyclic unification grammar). With NL grammars, however, such *intrinsic* complexity apparently does not happen: as was discussed above, NL grammars are "almost" context-free, so that unification features could in principle be "almost" dispensed with, and then, there would be no unification interactions to cope with.

In the remainder of this paper, we will explore ways to transform the specialization chart/grammar obtained for an offline-parsable grammar in such a way that each top-down traversal leads to a satisfiable set of constraints. Because of the np-completeness property noted above, such transformed chart/grammars could in theory take exponential space, but the situation seems to occur rarely, if ever, with NL grammars.

3 Standardized rules

Standardized unification sets We will use the following notation for a unification constraint:

$$[[A_1, \ldots, A_n], (l_1, B_1), \ldots, (l_m, B_m)],$$

with $1 \leq n, 0 \leq m$, and $A_i \neq A_{i'}$ for $i \neq i'$, with the following interpretation: each A_i, B_j is a variable representing a feature structure or is a constant representing an atomic feature structure (such as 'sg', 'love', etc.); $l_1, \ldots, l_m$ are attribute labels (such as 'subj', 'number', etc.); the first element $[A_1, \ldots, A_n]$ of the constraint, called its *identification set*, says that the structures $A_1, \ldots, A_n$ should be unified, the remaining elements $(l_1, B_1), \ldots, (l_m, B_m)$, called *access relations*, say that the value of the attribute l_j of A_1 (and, therefore, of any A_i) is B_j. We will also use the notation $\top$ for the "impossible" unification constraint (which is never satisfied by any structure). Two simple cases of a unification constraint are of special interest: the constraint $[[A, A']]$, which indicates unification of A and A', and the constraint $[[A], (l, B)]$, which indicates that B is accessed from A through the attribute l.

A finite set of unification constraints is said to be in *standardized form* if it is the singleton $\{\top\}$ or if it has the following properties:

- it does not contain $\top$;
- the identification sets of the constraints are disjoint;
- in any given constraint, an attribute label appears at most once;
- in any given constraint, if any of the A_i is a constant, then it is the only one, and also $m = 0$.

A functional structure, in the sense of LFG or related formalisms, can be seen as an oriented graph whose edges are labeled by attributes, in such a way that no two edges with the same label originate in the same node. If one distinguishes a certain "root" variable A in a unification constraint set, then this set can be seen as subsuming all the functional structures rooted in a node N_A which respect, in the obvious sense, the description expressed by the constraint set. If the set is in standardized form and is different from $\top$, then there exist such structures, and the "minimal" functional structure (or, in Prolog parlance, the most general unifier) which subsumes all these structures is trivially obtainable from this standardized set. One has the following property (see [Colmerauer, 1984] for a slightly different wording of the result):

> If $\mathcal{U}$ is a constraint set, then one can obtain in linear time (in function of the size of $\mathcal{U}$) an equivalent standardized set $\mathcal{U}'$, where "equivalent" means: accepting the same functional structures.

The reduction proceeds by interleaving two steps:

- If two constraints have non-disjoint identification sets, then replace them by a single constraint having as identification set the union of the two identification sets, and having as access relations the union of the access relations of the two constraints;
- If a given constraint contains two access relations with the same label (/, *B)* and (/, G), then eliminate the second of these relations, and add to the constraint set a new constraint [[B, C]] expressing the identification of *B* and G.

After a finite number of steps, no more such transformations can be done. At this point, two cases can occur: either some identification set contains two different atomic constants (which indicates unification failure) in which case one replaces the whole constraint set by the singleton T, or this is not so, in which case the unification set is in standardized form.

Standardized rules A grammar rule

$$R: \quad a(A) \rightarrow b(B)\ c(C)\ \ U_R(A,B,C)$$

is said to be standardized if the unification constraint set U_R is in standardized form. From what has just been said, it is clear that any grammar rule can be put in standardized form without altering the structures accepted by the grammar.

4 Interaction-free grammars

From any (binary branching) CF grammar G, one can obtain a related unification grammar G', which "registers" the derivations of G. If

$$a \rightarrow b\ c$$

is a non-lexical rule of G, then the corresponding rule of *G'* is:

$$a(A) \rightarrow b(B)\ c(C),\ [[A],(1,B),(r,C)],$$

where A, B, C are fresh variables, and where the constraint expresses that the left constituent of A is B, its right constituent C. Similarly, for a lexical rule:

$$a \rightarrow [t]$$

of G. the corresponding rule of *G'* is:

$$a(A) \rightarrow [t],\ [[A],(lex,t)],$$

which indicates that the lexical value of *A* is t.

The grammar G\ which we call a a *pure derivation grammar,* accepts the same strings as G, and assigns to them a functional description which is just an encoding of the derivation tree for G.

It is clear that, in any top-down traversal of G', the constraints cannot clash. In fact, if one considers the set of constraints collected during such a traversal, it is obvious that this set is a *standardized* unification set, so that no interaction exists between the different variables.

We now introduce a definition which generalizes the situation obtained with G':

> A grammar is called an *interaction-free,* or /F, grammar, if all its standardized rules are interaction-free, that is, have the following two properties:
>
> - the unification set of the rule is not {T};
> - if B is the variable associated with any rhs nonterminal b, then this variable does not appear in the identification set of any unification constraint in the rule.

It can be checked that this condition is respected by grammar G'. In an interaction-free grammar, any top-down traversal gives rise to a standardized set of unifications, so that no clash can occur between constraints.

Standardized unification sets and interaction-free rules The choice of representation for unification constraints made in section 3 was obviously not the only one possible. A more standard, but equally expressive, notation for unifications would have been:

$$(A,l,B)$$

for access constraints, and:

$$C = D$$

for equality constraints.

The interest of the notation of section 3 is that the notion of standardization can be defined in it straighforwardly, and that this last notion is directly related to the property of interaction-freeness. Because of the form of a standardized rule, it is obvious that a variable *B* that does not appear in the identication set of any unification constraint can be set arbitrarily by the "nonterminal call" b(B). without risk of clashing with variables set by other nonterminal calls in the rule.

This is to be contrasted with the situation had the more standard notation been used. Consider, in such a notation, the following rule:

$$a(A) \rightarrow b(B)\ c(C),\ (A,11,B),(A,11,D),(D,12,C).$$

In this rule there can be a conflict between the assignments given to B and to C. This is because, implicitly, the value of the 12 attribute in B is equal to C. On the other hand, using the notation of section 3, the standardized rule is written as

$$a(A) \rightarrow b(B)\ c(C),\ [[A],(11,B)],[[B,D],(12,C)]$$

which is immediately seen *not* to be interaction-free: B appears in the identification set of the second constraint.

5 Making an acyclic grammar interaction-free

Let us consider the chart/grammar specialization G_α of an offline-parsable grammar G for a string α. This grammar is acyclic, that is, no nonterminal calls itself recursively. We will now introduce an algorithm for transforming G_α into an equivalent acyclic IF grammar. We will suppose that the rules of G_α have been standardized.

The algorithm works by iteratively replacing non-IF rules of the grammar G_α by equivalent, "more IF", rules, until no non-IF rule remains in the grammar:

1. We may suppose that there exists a non-IF rule in the grammar, otherwise we are finished. Let us say that a given rule R' is *below* another (different) rule R'' if there is a sequence of rules $R_1 = R'', R_2, \ldots, R_n = R'$ in the grammar such that R_i contains in its rhs a nonterminal which appears on the lhs of R_{i+1}. Because the grammar is acyclic, there must be some non-IF rule R such that any rule below it is IF; otherwise there would be a non-IF rule which is below itself, which would imply grammar cyclicity.
2. Take R. There exists a nonterminal `b(B)` in its rhs such that `B` appears in the identification set of some constraint of R. Partially evaluate `b(B)` by replacing it with the right-hand sides of the (IF) rules which define `b`.
3. This may produce several different "copies" of R, or none, depending on the number of rules which define `b` in the grammar. In the latter case, no copy is produced, so that R has disappeared (it was unproductive, that is, could not generate anything.)
4. Is some copies of R are produced, standardize them. If, after standardization, the constraint set of the rule is $\{\top\}$, eliminate the corresponding rule (it is unproductive).
5. Go to step 1.

The algorithm terminates after a finite number of steps, because in an acyclic grammar, partial evaluation can only be performed a finite number of times.

Also, one can see, by a simple induction, that the IF grammar obtained only contains productive nonterminals, that is, nonterminals which do generate some string with satisfiable constraints.

Let us consider a simple example. Suppose that the input string is "john read here", and that the chart/grammar specialization is:

```
s(S) → np(NP) vp(VP), [[S],(l,NP),(r,VP)],
   [[NP],(n,X)], [[VP],(n,X)]
vp(VP) → v(V) a(A), [[VP],(l,V),(r,A),(n,Y)],
   [[V],(n,Y)]
v(V) → [read], [[V],(lex,read),(n,sg)]
v(V) → [read], [[V],(lex,read),(n,pl)]
np(NP) → [john], [[NP],(lex,john),(n,sg)]
a(A) → [here], [[A],(lex,here)]
```

where n refers to the attribute 'number'. All rules are IF, apart from the `s` and `vp` rules. All the rules below the `vp` rules are IF. We evaluate partially `v(V)` in this rule, which gives the two rules:

```
vp(VP) → [read] a(A), [[V],(lex,read),(n,sg)],
   [[VP],(l,V),(r,A),(n,Y)], [[V],(n,Y)]
vp(VP) → [read] a(A), [[V],(lex,read),(n,pl)],
   [[VP],(l,V),(r,A),(n,Y)], [[V],(n,Y)]
```

or, after standardization:

```
vp(VP) → [read] a(A), [[V],(lex,read),(n,sg)],
   [[VP],(l,V),(r,A),(n,Y)], [[sg,Y]]
vp(VP) → [read] a(A), [[V],(lex,read),(n,pl)],
   [[VP],(l,V),(r,A),(n,Y)], [[pl,Y]]
```

These rules are IF, because the nonterminal `a(A)` does not appear in any identification set. The only non-IF rule is now the rule for `s`, and we partially evaluate `np(NP)` in this rule, giving, after standardization:

```
s(S) → [john] vp(VP), [[NP],(lex,john),(n,sg)]
  [[S],(l,NP),(r,VP)], [[VP],(n,X)], [[X,sg]]
```

This rule is again non-IF. We partially evaluate `vp(VP)`, giving the two rules:

```
s(S) → [john] [read] a(A),
  [[V],(lex,read),(n,sg)],
  [[VP],(l,V),(r,A),(n,Y)], [[sg,Y]],
  [[NP],(lex,john),(n,sg)],
  [[S],(l,NP),(r,VP)], [[VP],(n,X)], [[X,sg]]
s(S) → [john] [read] a(A),
  [[V],(lex,read),(n,pl)],
  [[VP],(l,V),(r,A),(n,Y)], [[pl,Y]],
  [[NP],(lex,john),(n,sg)],
  [[S],(l,NP),(r,VP)], [[VP],(n,X)], [[X,sg]]
```

which, during standardization, become:

```
s(S) → [john] [read] a(A),
  [[V],(lex,read),(n,sg)],
  [[VP],(l,V),(r,A),(n,Y)],
  [[NP],(lex,john),(n,sg)],
  [[S],(l,NP),(r,VP)],
  [[Y,X,sg]],
s(S) → [john] [read] a(A),
  [[V],(lex,read),(n,pl)],
  [[VP],(l,V),(r,A),(n,Y)],
  [[NP],(lex,john),(n,sg)],
  [[S],(l,NP),(r,VP)],
  [[pl,Y,X,sg]],
```

In the second rule, the constraint `[[pl,Y,X,sg]]` reduces to $\top$, and so the rule is eliminated. As for the first rule, it is is already standardized, as well as IF. The grammar obtained is now IF.

It should be noted that, in the IF rule obtained for `s`, the adjunct `a(A)` is now "free": because `A` does not appear in any identification set of the rhs, whatever the values `A` may take, it won't interact anymore with the rule. Thus, even if there

were many analyses for the adjunct, this unique rule would take care of all of them. This is to be contrasted with the case where we would have evaluated all the nonterminals appearing in the rhs of the s rule, where each solution for the adjunct would have been explicitly represented along with the others.

The example, although simple, displays the crucial feature of the transformation into IF form: partial evaluation is performed only in case of need; as soon as a nonterminal can no longer interact with its siblings in a rule, it is not expanded further. If this nonterminal itself has a number of possible solutions, these solutions are kept "factorized" in the grammar.

Ambiguous structures seen as IF grammars Grammars are a kind of and/or representation: alternative rules for expanding the same nonterminal correspond to a disjunction, while a string of nonterminals on the rhs of a given rule corresponds to a conjunction.

An acyclic IF grammar with no unproductive nonterminals is an efficient representation of the several feature structures it generates: if one traverses this grammar top-down from its initial nonterminal *s,* then one never backtracks, because all nonterminals are productive, and because the collected constraints cannot clash. Thus one can enumerate all the structures in a direct way. Such grammar representations can be likened to finite-state representations for dictionaries, which, although they are more compact than simple lists of words, have for most purposes the same practicality as such lists.

6 Conclusion

This paper has brought together two current lines of research: (i) viewing charts as grammar specializations, and (ii) extending chart-like ambiguity packing to unification grammars. By doing so, it has clarified the principles underlying the nature of shared representations, stressed the link between grammars and representations, and opened the way for further applications in computational linguistics.

Acknowledgments Thanks for discussions and comments to Max Copperman, Lauri Kartunnen, Bernard Lang, John Maxwell and Eric Villemonte de la Clergerie.

References

[Bar-Hillel *et al,* 1961] Y. Bar-Hillel, M. Pexles, and E Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift fur Phonetik, Sprachwissenschaft und Kommunicationsforschung,* 14:142-172,1961.

[Billot and Lang. 1989] S. Billot and B. Lang. The structure of shared forests in ambiguous parsing. In *21th Meeting of the Association for Computational Linguistics.* 1989.

[Colmerauer, 1984] A. Colmerauer. Equations and Inequations on Finite and Infinite Trees. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS-84),* pages 85-99, Tokyo. Japan. November 1984. ICOT.

[Joshi, 1985] A. K. Joshi. How much context-sensitivity is required to provide reasonable structural descriptions: Tree adjoining grammars. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Parsing: Psycholinguistic, Computational and Theoretical Perspectives,* pages 206-250. Cambridge University Press, New York, 1985.

[Joshi, 1987] Aravind K. Joshi. The convergence of mildly context-sensitive grammatical formalisms. Presented at the Workshop on *Processing of Linguistic Structure,* Santa Cruz, 1987.

[Kaplan and Bresnan, 1982] R. Kaplan and J. Bresnan. Lexical functional grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The mental representation of grammatical relations.* MIT press, Cambridge, MA, 1982.

[Lang, 1994] Bernard Lang. Recognition can be harder than parsing. *Computational Intelligence,* 10,1994.

[Maxwell and Kaplan, 1990] J. T Maxwell and R. Kaplan. A Method for Disjunctive Constraint Satisfaction. Technical Report SSL-90-06. Xerox Pare. (Palo Alto Research Center), June 1990.

[Maxwell and Kaplan, 1993]
John T. Maxwell and Ronald M. Kaplan. The interface between phrasal and functional constraints. *Computational Linguistics.* 19(4):571-590, December 1993.

[Maxwell and Kaplan. 1996]
John T. Maxwell and Ronald M. Kaplan. An efficient parser for lfg. In *First LFG Conference,* Grenoble. France, August 1996. http://www-csli.Stanford.edu/user/mutt/.

[Pereira and Shieber. 1987] Fernando C. N. Pereira and Stuart M. Shieber. *Prolog and Natural-Language Analysis,* volume 10 of *CSU Lecture Notes.* Chicago University Press, Stanford. 1987.

[Pereira and Warren. 1980] Fernando C. N. Pereira and David H. D. Warren. Definite clause grammars for language analysis. *Artificial Intelligence,* 13:231-278,1980.

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

[Vijay-Shanker and Weir, 1994] K. Vijay-Shanker and D. J. Weir. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory,* 27(6):511-546. November/December 1994.

NATURAL-LANGUAGE PROCESSING AND GRAPHICAL PRESENTATION

Natural-Language Processing 4: Dialogue and Discourse