Bi-directional Grammars for Machine Translation

Koichi Takeda

In this paper we consider bi-directional grammars for natural languages. That is, a special class of grammars which can be used for both parsing and generation of sentences. We define the notion of "bi-directionality" and a class of general unification grammar and show that any instance of the grammar is bi-directional. We also discuss a subset of the grammar where more desirable property holds. We also consider an operational counterpart of the unification grammar, called pseudo-unification grammar, and show that similar results hold.

1. Introduction

Since FUG (Functional Unification Grammar) was introduced by Kay (1984) and DCG (Definte Clause Grammar) by Pereira and Warren (1980), bi-directionality (Hasida and Isizaki (1987), Shieber (1988), Wedekind (1988), Gates et al. (1989), Noord (1990) (i.e. using a single grammar for both parsing and generation of sentences) of so-called unification grammars (Shieber (1986)) has been one of the central issues on computational linguistics. Obvious reasons to explore the bi-directionality include psychological and theoretical elegance and practical importance that a single knowledge source can be shared by different processes. In particular, machine translation systems will be benefited from this architecture because, otherwise, they have to provide two different version of grammars (analysis, generation) for each language.

Because of the nature of unification, it has been predicted that a phrase-structure grammar formalism with unification as the only operation for building feature structures is indeed bi-directional (Kay (1984), Pereira &

Warren (1980), Hasida & Isizaki (1987)). Recently, procedural considerations on bi-directional grammars have emerged. Shieber (1988) recently reported a uniform architecture of using a deductive algorithm for parsing/generation and later Shieber et. al. (1990) elaborated the algorithm to incorporate a class of grammars broader than what they call *semantically monotonic* grammars. Wedekind (1988) showed that there exist two algorithms, one for analysis and the other for generation, for LFG (Lexical Functional Grammar)(Kaplan & Bresnan (1982)) such that, given an instance G of LFG, the former accepts a sentence S with a well-formed f-structure F in G iff the latter generates S from F using G.

Remaining questions, then, include

- Does any instance of a given grammar formalism satisfy the bi-directionality?
- Can we determine the bi-directionality of the instance?
- Is bi-directional grammar practical?

In this paper we give an answer to these questions when a grammar formalism is a PATR-II(Shieber (1986))-type unification grammar with no special scheme for building semantic structures. The notion of bi-directionality is defined in terms of syntactic feature structures alone so that we can discuss bi-directionality independently of any constraints imposed by a specific semantic composition scheme. However, our results can be applied to feature structures with semantic information as shown in Wedekind (1988). We show that bi-directionality theoretically holds for any instance of our grammar, but there is a practical subset of instances where more desirable properties hold. We also discuss Pseudo-Unification Grammar (Tomita & Knight (1987), Gates et al. (1989)), an extension of the Augmented Context-Free Grammar using unification, and its bi-directionality.

2. Feature Structures, Unification Grammar and Rule Graphs

A *feature structure* is a *finite*, rooted DAG(directed acyclic graph)$D=\langle V,E\rangle$, where V is a non-empty, finite set of vertices, and E is a finite subset of directed edges $V\times L\times V$. L is a (possibly infinite) set of feature names, called *labels*. A *path* is a sequence of edges $e_0=\langle v_0, a_0, v_1\rangle$, $e_1=\langle v_1, a_1, v_2\rangle$, $e_m=\langle v_m, a_m, v_{m+1}\rangle$, $(m \geq 0)$. The last node in a path, v_{m+1} above, is

called a *destination* node of the path. An element of E with no out-going edges is called a *leaf*. A leaf can be associated with an atomic feature value. Such a leaf is called an *atomic* node. A leaf which is not an atomic node is just a place-holder. There is one and only one node in D, called a *root*. By $D(p)$, where p is a path from the root, we mean A subgraph of D located by the path p from the root. E does not contain two edges with the same starting node, the same label, and different destination nodes, which guarantees the uniqueness of a path specified by a sequence of labels. Hence, we often abbreviate a path as a list of labels when a starting node is understood. We say D is *typed* if we define a set of permissible labels on edges for each node and possible atomic values for atomic nodes. Otherwise, it is *untyped*.

We assume untyped DAGs unless otherwise stated. The readers can find typed feature structures with inheritance in Pollard and Sag (1987). Our feature structures are similar to the ones in Shieber (1986). A partial ordering, called *subsumption* is defined over a set of feature structures. A feature structure D_1 is said to subsume D_2 iff (1) $D_1(p)$ is undefined, (2) if $D_2(p)$ is an atomic node, then $D_1(p)$ is also an atomic node associated with the same atomic value or just a non-atomic leaf, or (3) D_1 (p) subsumes D_2 (p), for every path p in either D_1 or D_2. *Unification* of two feature structures D_1 and D_2, if exists, is a feature structure D such that (i) D is subsumed by both D_1 and D_2 and (ii) for all D' such that D' is subsumed by D_1 and D_2, D' is also subsumed by D. Operationally, unification can be defined as a graph-merge algorithm (see Knight(1989)) for DAGs.

An instance of *unification grammar* consists of a set g of rules. Each rule has a *phrase structure part* and a set of *equations*. A phrase structure part is a context-free rule of the form $X_0 \rightarrow X_1X_2 \cdots X_n$, $(n \geq 1)$ and an equation is one of the following.

$\langle x, p \rangle = c$, or

$\langle x, p \rangle = \langle x, p \rangle$

An equation specifies that the feature structures specified in the LHS (left hand side) and the RHS (right hand side) to be unified. The symbol X_i refers to the root of a feature structure associated with the i-th non-terminal symbol in the RHS, $(i=1, ...,n)$ and X_0 denotes a root of the LHS feature structure. Symbols p and q denote paths from a root of a feature structure. That is, $\langle x, p \rangle$ means $D(p)$, where D is the feature structure of X_i.

The first equation says that $D(p)$ must be an atomic node associated with a constant c. The second equations says that two paths $\langle x_i\ p\rangle$ and $\langle x_j\ q\rangle$ reach the same feature structure. A path in an equation may be empty. In this case, the entire feature structure is referred to by the empty path. Note that we do not allow negations, disjunctions, and functions in the rules.

Lexical rules have a form $X \rightarrow$ *word*, where only allowable equations are

$\langle x_0\ p\rangle = c$, or

$\langle x_0\ p\rangle = \langle x_0\ q\rangle$

This definition of a unification grammar is similar to the basic PATR-II grammar in Shieber (1986).

Each rule r in a unification grammar G, just like feature structures, can also be represented by a rooted DAG called *rule graph* D_r as follows. Let r be $X_0 \rightarrow X_1X_2...X_n$, $(n \geq 1)$ with a set Q_r of equations e_1, ..., e_m. Then, $D_r = \langle V,E\rangle$ is also a rooted DAG, such that

1. V is a union of nodes $\{x_i \mid i=0,...,n\} \cup$ {nodes appearing in Q_r},
2. The root of D_r is X_0,
3. E is a union of edges appearing in the paths of Q_r, and
4. D_r is a unification of n pairs of DAGs ⟨LHS of e_1, RHS of e_1⟩,⋯, ⟨LHS of en, RHS of en⟩.

A rule graph for a lexical rule is defined similarly. Multiple rule graphs can be combined into one. For example, a rule graph D_1 for $X_1 \rightarrow$ *word* and a rule graph D_2 for $X_2 \rightarrow X_1$ can be combined into a rule graph D_{21} for $X_2 \rightarrow$ *word* such that D_{21} is obtained by unifying x_0 in D_1 with x_1 in D_2. Hence, the feature structure of an entire sentence is obtained by combining all the rule graphs that appear in the derivation of the sentence. Note that we may have a trivial equation $\langle x_i,p\rangle = \langle x_i,p\rangle$ or its transitive equivalent $\langle x_i,p\rangle = \langle x_j,p\rangle$ and $x_i = x_j$[1]. In a (single or combined) rule graph. the path p seems redundant but it prevents the graph from being unified with a feature structure which forces a node in p, except the destination node, to be atomic. A *non-trivial* node is either an atomic node or a node which has at least one out-going edge. A leaf node is called *trivial* if it is non-atomic and has just one in-coming edge from a non-trivial node.

A rule graph can be viewed as a relation over n constituent feature

[1] This equation does not require the existence of an atomic node in D(p) although a different interpretation is possible.

structures and one superordinate feature structure. For example, given n constituent feature structures, the rule graph defines their superordinate feature structure by unifying a node x_i in the rule graph with a feature structure of the i-th constituent for each i. Rule graphs are said to be *equivalent* if they define the same $(n + 1)$-ary relations over such feature structures. Note that a rule graph might not be *connected*. That is, not all nodes may not be reachable from the root of the graph. It is easily shown that there must be a node x_i for some i if there is a node unreachable from the root. Then the node x_i is called a *hole* since only reachable components are visible from the root and are percolated to superordinate levels. Figure 1 illustrates a feature structure, a grammar rule, and its rule graph. Two rule graphs are called *semi-identical* if they are associated with the same phrase structure part and one is obtained by just adding a set of trivial nodes to another.

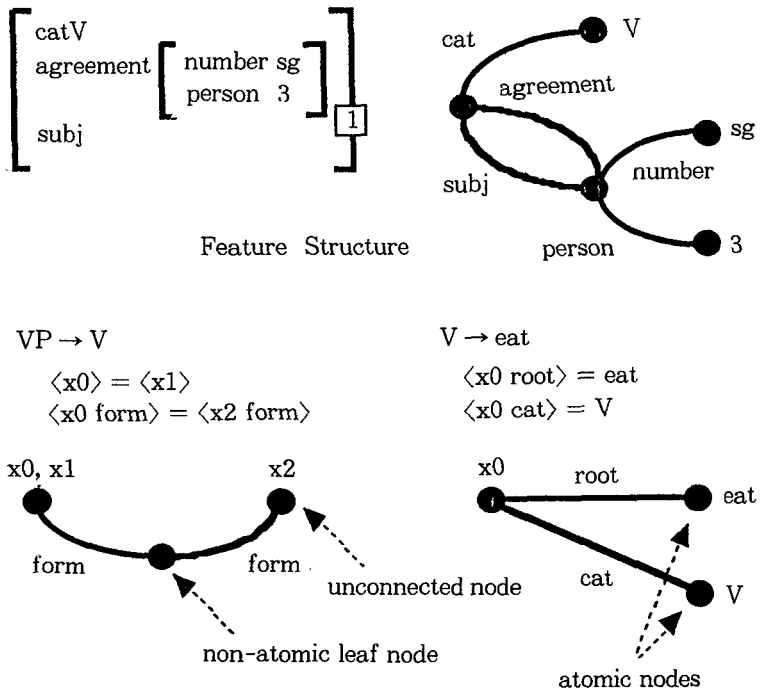

Figure 1. Feature Structure, Grammar Rules and Rule Graphs

Proposition 1 *Two rule graphs are equivalent iff they are semi-identical.*

Proof : Let the two rule graphs be D_1 and D_2. The "if" part is easily proved because any feature structures which cannot unify with D_1 also fail to unify with D_2. Successful cases could only differ in the trivial nodes but they do not make different *untyped* feature structures. The "only-if" part is proved as follows. Suppose D_1 has a node x which is not in D_2, and x is an atomic node, or a node which has multiple in-coming edges. Then, the two graphs are not equivalent because we can choose n feature structures which introduce an atomic node in the position of x and make successful unification with D_2 but not with D_1. Similarly, if x is a leaf node connected to a node y which is not non-trivial, we can find n feature structures which unifies an atomic node with y to make successful/unsuccessful result, respectively. If these cases do not apply to x, then there must be a path, whose labels do not appear in D_2. From a root node of some constituent to x, and x has at least one out-going edge. This is when we can also find n feature structures which unifies with D_2 but not with D_1. Thus the two graphs must have the same set of nodes. If a node is connected in one graph but not in the other, say D_1. Then there must be a root for some constituent which is not connected in D_1, either. This always makes different unification for the two graphs. Finally, suppose that there is an edge e in D_1 but not in D_2. Then we can choose a feature structures that introduces e connected to a new atom node in D_2 but fail to unify with D_1 because of conflict in e.

3. Bi-directionality of Unification Grammar

We define notions of parsing and generation. Let $G = \{r_1,...,r_n\}$ be an instance of a grammar over a set of alphabets $\sum$. A *parse tree* T is a finite ordered tree such that each terminal node in T corresponds to a terminal symbol in G, the root and other non-terminal nodes in T correspond to a start symbol and non-terminal symbols in G, respectively, and each non-terminal node and its immediate daughters correspond to LHS and RHS symbols of a phrase structure part of some r_i in G. For any given immediate subtree T' of T, a feature structure associated with the root of T' is obtained from a rule graph of a corresponding rule r_i, where each node x_j

(j=1,2,...) is unified with the root of a feature structure of the j-th immediate daughter. G successfully *parses* a sentence s, written *parse*(G,s,F), iff there is a parse tree T such that linear arrangement of terminal symbols in T agrees with s. A feature structure F must be associated with the root in T. G cannot parse s if such a tree does not exist (i.e. context free rules fail to reduce s into a start symbol, or it is impossible to construct an acyclic feature structure satisfying the equations). G successfully *generates* s from a given feature structure F, again written *generates* (G,F,s), iff there is a parse tree T such that the feature structure for the root of T is identical to F, and linear arrangement of terminal symbols in T agrees with s. By definitions, parse (G,s,F) holds iff generate (G,F,s)holds.

It can be easily shown that combination of the early deduction algorithm (Pereira & Warren (1983), Shieber et al. (1990)) with a DAG-merge unification algorithm[9] can eventually build any possible parse tree and a feature structure for a given sentence s using G because the tree is finite. Similarly, a DAG-marking algorithm(Wedekind (1988)) with a slight modification for handling unconnected nodes can generate a sentence, if possible, for a given feature structure F under G. Let us name these two algorithms PARSER (G,s,F) and GENERATOR (G,F,s), respectively. Hence, it immediately follows that PARSER (G,s,F) iff parse (G,s,F).

Proposition 2 *For any grammar G, a feature structure F, and a sentence s, GENERATOR* (G,F,s) *iff generate* (G,F,s).

DAG-marking algorithm succeeds only if it reconstructs a parse tree by non-deterministically choosing a grammar rule, marking up all the nodes and edges in a given feature structure which appear in equations of the rule, and checking if two destination nodes in an equation are the same node.[2] The algorithm fails if it encounters an equation having a node or a path which is not in the given feature structure, or if there is a node or a path yet to be marked but no grammar rule is applicable. Therefore, if the algorithm succeeds, we have a derivation tree (or a parse tree) for s. If G generates s, which means there is a parse tree T with the feature structure

[2] The input F may not include trivial nodes as they are just place holders. In this case, the algorithm simply ignores trivial equations in the rule graphs.

F, then there must be a sequence of non-deterministic choices of rules to construct T. Correctness of DAG-marking follows immediately from the definition of the rule graph and feature structures. Proof is harder when a grammar rule has unconnected nodes since F only specifies a feature structure with connected nodes. In this case, the algorithm simply assume a hole for each unconnected root x of constituent that could be unified with a feature structure for any derivation subtree with root x. That is, any derivation tree with root x will be a part of generation from F.

Proposition 3 *GENERATOR (G,F,s) hold iff PARSER (G,s,F) holds. G is called bi-directional.*

The proof is immediate using the previous proposition. Thus, we have the following result.

Theorem 1 *Any instance of our unification grammar is bi-directional.*

That is, for any instance G, the parser and generator are exactly the inverse algorithm to each other. Note that the property holds for only a set of valid sentences. Now we proceed to propositions which show undesirable properties of the grammar.

Proposition 4 *If we allow unnconnected rules, we can have an instance G of grammar and a feature structure F such that GENERATOR (G,F,s) holds for arbitrarily manys.*

Proposition 5 *The DAG-marking algorithm may not terminate for a certain grammar instance G and a feature structure F when generate(G,F,s) does not hold for any s.*

We give examples.

$S \rightarrow AB$

$\langle x_0 \rangle = \langle x_1 \rangle$

$S \rightarrow C$

$\langle x_0 \rangle = \langle x_1 \rangle$

$A \rightarrow a$

$\langle x_0\ root\rangle = a$

$B \rightarrow BB$

$\langle x_0\rangle = \langle x_1\rangle$

$\langle x_0\rangle = \langle x_2\rangle$

$B \rightarrow b$

$\langle x_0\ root\rangle = b$

$C \rightarrow DC$

$\langle x_0\rangle = \langle x_1\rangle$

$\langle x_0\rangle = \langle x_2\rangle$

$C \rightarrow c$

$\langle x_0\ root\rangle = c$

$D \rightarrow d$

$\langle x_0\ root\rangle = d$

Given a feature structure ⟪root a⟫, the algorithm assumes a hole for b in the first rule. A successful derivation will generate "a" followed by arbitrary many "b"'s because of the fourth and fifth rules which will fill the hole. The parsing of such a string $ab+$does not suffer this problem although these strings all have the same feature structure ⟪root a⟫. Hence, This shows a kind of asymmetry (Russel et al. (1990)) in parsing and generation. Note both parsing and generation suffer from the structural ambiguities caused by the fourth rule to derive a sequence of b's. Now, consider a feature structure ⟪root c⟫ as input. The generation from ⟪root c⟫ never succeeds because two feature structures for C and D in the sixth rule always disagree on the "root" value no matter how deep the sixth rule is expanded. Thus, the algorithm has to "know" somehow that the derivation from C will be in vain in order to terminate. Similary, if a grammar has a loop of non-branching rules, whose equation is only $\langle x_0\rangle = \langle x_1\rangle$, even these connected rules may force the algorithm to run forever. For this same reason, *functional uncertainty* (Kaplan et al. (1986)), which allows regular expressions in a path, would be problematic when it is used in a rule.

One way to handle this problem is to modify the algorithm so that it can detect infinite derivations. The other way is to define a safe subset of grammar instances such that the algorithm always terminates. For example, the depth-bounded unification grammar (Haas (1989)) is such a safe subset for parsing. This idea leads to the following theorem. Let *unconstrained*

derivation be a sequence q of grammar rule application of a form $A \rightarrow \cdots \rightarrow \alpha A \beta$ such that, in a combined rule graph D for q, the root of D is also the root of the constituent A in the derived sequence. Note the further derivation $A \rightarrow \cdots \rightarrow \alpha\alpha A \beta\beta$ will not change the rule graph.

Theorem 2 *If a grammar G has no unconstrained derivation and each rule in G is connected, then, for any sentence s, GENERATOR (G,F,s) fails in some bounded time iff generate (G,F,s) does not hold.*

Proof: THE "only-if" part is immediate. The "if" part is shown as follows. Let k be the length of the longest path in F and let N must introduce an edge from the root to its re-appearing constituent, which grows the length of the path by one. Thus, after trying such derivation $N*k$ times, any further derivation will exceed the size k and f fails to generate s.

This subset of grammar is so restrictive that even a simple $VP \rightarrow VPNP$ rule must be excluded. The restriction can be relaxed by introducing additional information which is similar to the well-formedness constraints (Kaplan & Bresnan (1982)) of LFG. Unconstrained derivation will be restated by the well-formedness condition and uniqueness of PRED fillers. Connectedness of a constituent is imposed by the completeness and coherence constraints if the constituent has a grammatical function.[3]

Proposition 6 *It is NP-complete to check whether a grammar G has an unconstrained derivation or not.*

Proof: By a simple reduction of the 3-SAT problem (Garey & Johnson (1979))

The implication of this proposition is that it is unlikely to determine efficiently if a given instance of the grammar is free from an infinite loop. Since a membership problem for such a restrictive subset seems intractable, a well-designed semantic structure (e.g. *monotonic* semantic structure (Shieber (1988)) might be inevitable to guarantee the termination. This problem also applies to the Pseudo-Unification grammar mentioned below.

4. Pseudo-Unification Grammar

PUG (Pseudo-Unification Grammar) was first proposed by Tomita and

Knight (1987) as an alternative unification grammar formalism for efficient processing of natural languages and was employed in the KBMT-89 system (KBMT 89 (1989)) at CMU. PUG consists of grammar rules similar to PATR-II. However, the interpretation of equations, such as $\langle x, p \rangle = \langle x, q \rangle$, is *snapshot identity* rather than *permanent identity* of feature structures. That is, two feature structures for $\langle x, p \rangle$ and $\langle x, q \rangle$ get unified and result in identical feature structures when the above equation is evaluated. After that, however, these two feature structures are treated as two unrelated structures. In other words, PUG can be thought of as Augmented Context-Free Grammar with feature assignment rules $\langle x, p \rangle = \langle x, q \rangle$, where the meaning of this equation is to unify these two feature values and assign the result to both LHS and RHS. The order of equations in a rule is important in PUG because equations are basically assignments and order-dependent.

Syntactically, PUG is defined exactly the same as our unification grammar in section 2, except that PUG has more types of equations. In addition to the "=" operator, PUG has

- (a constraint equation) =c,
- (value existence tests) =*defined* and =*undefined*,
- (plain assignment) < =, and
- (append value) <

A constraint equation "X=c v" holds if v is an atomic value and X already has value v. "X=*defined*" holds if X does not have an empty feature structure. "X=*undefined*" holds if X is empty. "X <= Y" always holds and the feature structure of X is overwritten by the feature structure of Y. "X < Y" holds if a feature structure of Y can be appended to X's. The resulting list of feature structures replaces the old value of X. Normally the equations in PUG rules is written from top to bottom to build feature structures for parsing. For generation, we view the equations from bottom to top to determine the constituent feature structures (See Gates et al. (1989) for more details). We can use the Tomita algorithm (Tomita (1985)) for parsing and DAG marking algorithm (with bookkeeping of instantiation that is required to handle constraint equations and value existence tests) for generation.

Since PUG has destructive operations, it is easily seen that "full" PUG

has the same problem as unconnected grammar rule does in the unification grammar. Another problem is ordering of equations. For example,

$S \leftarrow AB$

$\langle x_0 \rangle = \langle x_1 \rangle$

$\langle x_1 \rangle = \langle x_2 \rangle$

fails to carry the feature structure x_2 to x_0 because the second equation unifies x_2 with x_1 which was already unified with x_0. Thus, a generation algorithm cannot determine the feature structure of x_2 from x_0 during the generation. Similarly,

$S \leftarrow AB$

$\langle x_0\ a \rangle = \langle x_1 \rangle$

$\langle x_1 \rangle = \langle x_2 \rangle$

$\langle x_0\ b \rangle = \langle x_1 \rangle$

makes version inconsistency ; two versions of x_1 in $\langle x_0\ a \rangle$ and $\langle x_0\ b \rangle$ might disagree. Unconnectedness and version inconsistency mislead a generator in a sense that it might generate a sentence which will never be parsed successfully. Thus, we define a safe subset of PUG as follows to guarantee the bi-directionality.

Suppose $G=\{r_1,...,r_n\}$ is an instance of PUG such that for each rule r_i with k equations $\{e,...,e_k\}$, (1) there is one and only one equation of the form (empty paths in both LHS and RHS) $\langle x_0 \rangle = \langle x_i \rangle$ for some $i > 0$, (2) a rule graph D constructed from only "=" equations is a connected *tree*, (3) no edge is included in two or more equations, and (4) for each root x of every subtree in D, no equation that includes an edge going to x precedes the equations which make the subtree. This subset of PUG is also very restrictive even though no information loss and version problems occur in the subset.

Theorem 3 *This subset of PUG defined above satisfies the bi-directionality.*

The subset of PUG above is free from anomalies of destructive operations. The finite failure property, discussed in the previous section, also holds for this subset of PUG by excluding unconstrained derivation.

5. Concluding Remarks

In this paper we examined bi-directionality of a general unification grammar and Pseudo-Unification grammar. Although any instance of the unification grammar satisfies bi-directionality, there is a practical subset which has more desirable property. Future problems include the handling of extended formalisms such as disjunctions and set values in feature structures, characterization of other subsets such as unique generation (any successful feature structure generates unique sentence), and utilization of semantic representation (Wedekind (1988), Shieber et al. (1990)) that will play as an interlingua for machine translation. Frame-based semantic representation and syntax-semantics mapping rules are also promising candidate for bi-directional machine translation systems (KBMT 89 (1989), Takeda (1991)).

References

Garey, M. and D. Johnson (1979) *Computers and Intractability*, W. H. Freeman and Co., San Frandisco.

Gates, D., K. Takeda, T. Mitamura, L. Levin and M. Kee (1989) 'Analysis and Generation Grammar,' *Machine Translation*, 4 (1): 53-66.

Haas, A. (1989) 'A Parsing Algorithm for Unification Grammar,' *Computational Linguistics*, 15(4): 219-232.

Hasida, K. and S. Isizaki (1987) 'Dependency Propogation: A Unified Thory of Sentence Comprehension and Generation,' In *Proc. of the 10th IJCAI*, pp. 664-670.

Kaplan, R. and J. Bresnan (1982) 'Lexical-Functional Grammar: A Formal System for Generalized Grammatical Representation,' In J. Bresnan, editor, *Mental Representation of Grammatical Relations*, MIT Press, Cambridge, Mass., pp. 173-281.

Kaplan, R., J. Maxwell and A. Zaenen (1986) 'Functional Uncertainty,' *Technical Report CSLI Monthly* Vol. 2, No. 4, CSLI, Stanford.

Kay, M. (1984) 'Functional Unification Grammer: A Formalism for Machine Translation,' In *10th International Conference on* Computational Linguistics, Stanford, pp. 75-78.

KBMT 89 (1989) 'Special Issue on Knowledge-based Machine Translation I and II,' *Machine Transltion* 4 (2-3).

Knight, K. (1989) 'Unification: A Multidisciplinary Survey,' *ACM Computing Surveys* 21(1): 93-124.

Perira, F. and D. Warren (1980) 'Definite Clause Grammar for Lanuage Analysis,' *Artificial Intelligence* 13: 231-278.

Pereira, F. and D. Warren (1983) 'Parsing as Deduction,' *Proc. of 21st Annual Meeting of the Association for Computational Linguistics*, pp. 133-144.

Pollard, C. and I. A. Say (1987) 'An Information-based Syntax and Semantics,' *CSLI Lecture Notes* 13.

Russel, G., S. Warwick and J. Carroll (1990) 'Asymmetry in Parsing and Generating with Unification Grammars: Case Studies from ELU,' In *Proc. of the 28th Annual Meeting of the Association for Computational Linguistics*, Pittsburgh, pp. 205-210.

Shieber, S. M. (1986) 'An Introduction to Unification-based Approaches to Grammar,' *CSLI Lecture Notes* 4, Standford CA.

Shieber, S. M. (1988) 'A Uniform Architecture for Parsing and Generation,' In *Proc. of the 12th International Conference on Computational Linguistics*, pp. 614-619.

Shieber, S. M., F. C. N. Pereira, G. van Noord and R. C. Moore (1990) 'Semantic-Head-Driven Generation,' *Computational Linguistics* 16 (1): 30-42.

Takeda, K. (1991) 'Semantic Mapping and Bi-directional Machine Translation,' *Technical Report* (to appear), Tokyo Research Laboratory, IBM Research, Tokyo.

Tomita, M. (1985) *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*, Kluwer Academic Publishers, Boston, MA.

Tomita, M. and Kinght, K. (1987) 'Pseudo Unification and Full Unification,' *Technical Report CMU-CMT-88-MEMO*, Center for Machine Translation, Carnegie Mellon University.

Noord, G. van (1990) 'Reversible Unification Based Machine Translation,' In *Proc. of the 13th International Conference on Computational Lingustics*, Helsinki, pp. 299-304.

Wedekind, J. (1988) 'Generation as Structure Driven Derivation,' In *Proc. of the 12th International Conference on Computational Linguistics*, pp. 732-737.

Tokyo Research Laboratory
IBM Japan, Ltd.
5-19 Sanban-cho, Chiyoda-ku
Tokyo 102
Japan