MARK JOHNSON

TWO WAYS OF FORMALIZING GRAMMARS*

1. Introduction

A grammar is a formal device which both identifies a certain set of utterances as well-formed, and which also defines a transduction relation between these utterances and their linguistic representations. This paper focuses on two widely-used "formal" or "logical" representations of grammars in computational linguistics, Definite Clause Grammars and Feature Structure Grammars, and describes the way in which they express the *recognition problem* (the problem of determining if an utterance is in the language generated by a grammar) and the *parsing problem* (the problem of finding the analyses assigned by a grammar to an utterance).

Although both approaches are 'constraint-based', one of them is based on logical consequence relation, and the other is based on satisfiability. The main goal of this paper is to point out the different conceptual basis of these two ways of formalizing grammars, and discuss some of their properties.

1.1. *Definite-Clause Grammars, A Validity-based Approach*

The *definite-clause grammar* (DCG) framework originates in Colmerauer's work on Metamorphosis Grammars in the 1970's (Colmerauer 1978) and was developed and popularized by Pereira and Warren (1981), Pereira and Shieber (1987) and others. In this approach, a grammar (here taken to include the lexicon) is conceived of as a set of axioms. The well-formedness of an utterance and the fact that it has a certain linguistic structure are theorems that follow from these axioms, so both the recognition and parsing problems is one of determining if certain types of formulae are *logical consequences* of these axioms. Thus the well-formedness or grammaticality of a particular utterance is expressed by the fact that the corresponding formula is a consequence of the grammar axioms, and ungrammaticality by the fact that the corresponding formula is not a

* I would like to thank Edward Stabler and an anonymous *L&P* reviewer for their helpful comments on an earlier draft of this paper. Of course, all responsibility for errors in this paper rests with me.

Linguistics and Philosophy **17**: 221–248, 1994.

consequence of the grammar axioms (even though it may be consistent with them).

That is, if the grammar axioms are D and the formula $\mathsf{wf}(u, s)$ asserts that the utterance u is well-formed with linguistic reprsentation s (where s might be interpreted as a parse tree, etc.), then the recognition problem is the problem of determining if the following holds.[1]

$$D \models \exists \mathsf{s}\, \mathsf{wf}(u, s).$$

The *parsing problem* is the problem of finding all of the s such that the following holds for the given utterance u.

$$D \models \mathsf{wf}(u, s).$$

In general D is a finite set of closed formulae, so these problems are equivalent to the following validity problems, where D' is a conjunction of the members of D.

$$\models D' \rightarrow \exists \mathsf{s}\, \mathsf{wf}(u, s).$$
$$\models D' \rightarrow \mathsf{wf}(u, s).$$

To summarize, in the DCG approach the *intended interpretation* $\mathcal{M}_i$ is one in which linguistic representations and strings are conceptualized as individuals. The grammar axioms D state the essential properties of the intended interpretation $\mathcal{M}_i$, so if u is interpreted in $\mathcal{M}_i$ as a grammatical utterance with linguistic structure s, then $\mathsf{wf}(u, s)$ is true in *every* model of D.

1.2. *Feature Structures, A Satisfiability-based Approach*

The second framework is the *feature-structure* (FS) approach, where a grammar (which includes the lexicon) is conceived of as a set of *constraints*, and a well-formed linguistic representation is any structure that *satisfies* these constraints.[2] Specifically, the grammar and the utterance both impose constraints that the linguistic structure must meet. The well-formed or grammatical structures are those that satisfy the constraints imposed by the grammar. An utterance is well-formed iff one of these structures

[1] In this paper the following notation is used. Object-language expressions are written in sanserif font, e.g. x, y etc., while meta-language variables (ranging over object-language expressions) are written in italic font, e.g. x, y, etc.

[2] The version considered here is similar to HPSG (Pollard and Sag 1987) in that it is expressive enough that no external phrase structure component is required – the phrase structure rules are encoded as feature structure constraints – and is a simplification of systems proposed by Carpenter (1992).

also meets the additional constraint that it "corresponds" to the utterance in an appropriate way (e.g., the structure's yield (terminal string) is the string of words of that utterance). Thus grammaticality or well-formedness of an utterance corresponds to the satisfiability of a set of constraints, and ungrammaticality or ill-formedness corresponds to the unsatisfiability of that set.

In formalizing the recognition and parsing problems in this approach, linguistic representations can be regarded as interpretations, and the constraints as expressions or formulae (from some language of constraints) which these interpretations must satisfy in order to be considered well-formed linguistic representations. That is, a well-formed linguistic representation is a model of these formulae (rather than an individual in a model as in the DCG approach), and the set of all models of the grammatical constraints is the set of all well-formed linguistic representations. Thus unlike the DCG approach, in general there is no single intended model of a set of feature structure constraints.

Most of the work in this field has focussed on the development of specialized languages for expressing systems of constraints to be used as annotations on phrase-structures rules (e.g., the Feature Description Language of Kasper and Rounds (1990) and the attribute-value languages of Johnson (1988)). It seems that the language of first-order logic (in fact, usually decidable sublanguages thereof) is capable of expressing these kinds of constraints (Johnson 1990a, b, 1991a, b, Smolka 1992). Manaster-Ramer and Rounds (1987) and Carpenter (1992) propose extended versions of these systems that are expressive enough to be linguistically useful alone (i.e., without other descriptive devices such as a phrase-structure 'backbone'). This paper explores the degree to which such an extended feature system can be expressed in a first-order language.[3] This also aids comparison with the DCG approach, which is formulated in the same language.

The recognition problem is the problem of determining the simultaneous satisfiability of both grammar and utterance constraints. That is, if F is a formula expressing the grammatical constraints that every well-formed linguistic structure must satisfy (i.e. that is true in exactly the well-formed structures) and $yield(u)$ is a formula that is true in an interpretation (i.e., a linguistic structure) iff that interpretation corresponds to utterance u

[3] An interesting alternative not discussed in this paper is to extend a standard first-order language by adding 'feature-structure expressions' to that language. It seems that the most insightful semantics for such an extended language is based on abduction; see Höhfield and Smolka (1988) and Chen and Warren (1989) for details.

(say, has u as its phonological form), then utterance u is well-formed iff there exists a model $\mathcal{M}$ such that the following is true.

$$\mathcal{M} \models F \wedge yield(u).$$

The parsing problem is the problem of describing or characterizing the set of models of the conjoined constraints. Since this set may be infinite, it is not in general possible to exhaustively enumerate these models. There are two standard techniques for describing the models of the constraints, both exploiting the observation that infinite sets can have finite descriptions (e.g. the infinite set of integers greater than 7 has the finite description "$\{x \mid x > 7\}$"). These two techniques are discussed in detail in sections 2.8 and 2.10 of Johnson (1988).

The first technique exploits the observation that in cases where the possible constraints are restricted, it may be possible to show that the set of models possesses a certain structure, so that an infinite set of models can be finitely described, i.e., specified or identified with finite means. Usually, attention is restricted to a certain type of interpretation, e.g. acyclic deterministic finite automata (DFA) in Kasper and Rounds (1990), and attribute-value structures (AVS) in Johnson (1988). Kasper and Rounds (1990) showed that the set of DFA satisfying any constraint expressible in their Feature Description Logic is a finite union of principal filters (generated by the "minimal models" with respect to the "subsumption" ordering), and Johnson (1988) showed that the set of AVSs satisfying any constraint expressible in an attribute-value language is a finite union of finite differences of finitely-generated principal filters;[4] in both cases there are effective procedures for constructing these generators, which constitute a finite description of a (possibly infinite) set of models.

The second technique is a variation of the first one; it is based on the observation that every formula identifies a set of interpretations, namely those that satisfy it. Thus the formula $F \wedge yield(u)$ is a description of the set of its models (although perhaps not a very useful one). For some constraint languages (including those of Kasper and Rounds (1990) and Johnson (1988)) there exist algorithms that reduce an arbitrary formula to an equivalent formula in a "normal form", from which one can "read off' the important properties of the models (see sections 2.8 and 2.10 of Johnson (1988) for further discussion).

Independently of the existence of normal forms, however, if it can be shown that if

[4] Because attribute-value languages can express negated constraints, Johnson (1988) requires "negative" minimal models (i.e. "inequality arcs") as well as "positive" minimal models.

$$F \wedge \mathit{yield}(u) \models A$$

for some formula A, then A is true of every linguistic representation that satisfies the grammar constraints and corresponds to the utterance u, i.e. A is a description of the well-formed structures of u. Thus information about an utterance can be extracted by computing the logical consequences of the (grammar and utterance) constraints.[5]

For example, if the utterance u is ungrammatical, then

$$F \wedge \mathit{yield}(u) \models \text{false}$$

because there are no models of these constraints.

2. Formalizing context-free grammars

Both approaches are capable of expressing grammars considerably more complicated than the context-free grammars described in this section, but it is instructive to consider these simpler systems. This paper follows standard linguistic practice in assuming that the right hand side of each production in the grammars being formalized is either a (possibly empty) sequence of non-terminals or else a single terminal. This assumption simplifies the formalization somewhat without restricting the class of languages that can be expressed.

First, formalizations of the recognition problem for following simple context-free grammar (based on the simple grammar of Shieber 1986) are presented.

Then the axioms are modified so that a representation of the parse tree is produced as well. Finally, the axioms are further modified to include agreement features, so that ungrammatical utterances such as **Knights sleeps* are not generated.

$$\begin{array}{l} \mathsf{S} \rightarrow \mathsf{NP\ VP} \\ \mathsf{VP} \rightarrow \mathsf{V\ NP} \\ \mathsf{NP} \rightarrow \mathsf{uther} \\ \mathsf{NP} \rightarrow \mathsf{knights} \\ \mathsf{VP} \rightarrow \mathsf{sleeps} \\ \mathsf{V} \rightarrow \mathsf{like} \end{array} \tag{1}$$

[5] Not all the consequences A are informative, of course, since the set of consequences includes e.g. all tautologies. Correspondingly, not all of the logical consequences of the DCG axioms are of interest either.

Both of the formalizations presented exploit the following observation about context-free parse trees.[6]

The precedence relationships between the nodes of a parse tree can be described by associating each node with two positions in the terminal string, called its left and right *string positions*. These identify the extent of the substring that this node dominates (see e.g., Pereira and Shieber 1987 for a more detailed discussion).

The necessary precedence relationships can be established by requiring that for each node X and its children nodes $X_0 \ldots X_n$, the left string position of X is the same as the left string position of X_0, the right string position of X_i is the left string position of X_{i+1}, $0 \leq i < n$, and the right string position of X_n is the right string position of its parent X. Figure 1 sketches this relationship.

String positions are diacritics, in the sense that their precise identity is immaterial; all that matters is that the string positions are pairwise distinct. There are $n + 1$ string positions associated with a string of length n; it is convenient to use the suffixes of the string being analysed as these string positions. The string positions in Figure 1 show the substrings used to represent each of these string positions.

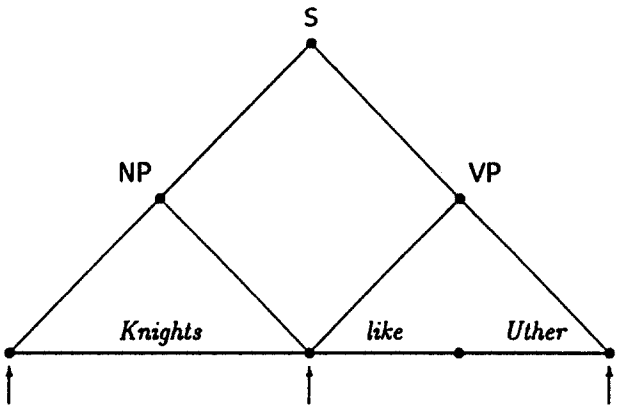

Fig. 1. The relationship between string positions associasted with parent and children nodes.

[6] The same observation also underlies tabular or chart parsing algorithms, and Chomsky's original formalization of context-free grammars in terms of phrase-markers.

2.1. *Definite-Clause Grammar*

In this approach, a context-free grammar is formalized as axioms that imply the existence of nodes in a 'bottom-up' fashion:[7] for each production $X \rightarrow X_1 \ldots X_n$ there is an axiom requiring that if there is a sequence of adjacent nodes $X_1, \ldots, X_n$ then there is also a node X (with the appropriate precedence relations).

2.1.1. *DCG Formulation of the Recognition Problem*

The recognition problem is formalized as follows in this approach. For each non-terminal symbol X a two-place relation symbol x is introduced, and for each terminal symbol w a constant symbol w is introduced.

Informally, the arguments of the relations x range over string positions, which are represented by sequences of terminal symbols. The relation $x(s_1, s_2)$ is true if s_2 is a suffix of s_1, and the 'difference' between s_1 and s_2 (i.e., the sequence s such that s_1 is the result of concatenating s and s_2) is a string of terminals derived by X.

Each production is translated to a separate Horn clause; a production expanding to non-terminals

$$X \rightarrow X_0 X_1 \ldots X_n$$

translates to the clause

$$\forall u_0 \ldots u_n\, x(u_0, u_n) \leftarrow x_0(u_0, u_1) \wedge x_1(u_1, u_2) \wedge \cdots \wedge x_n(u_{n-1}, u_n) \tag{2}$$

where the variables $u_0 \ldots u_n$ are intended to range over string positions.

Each production expanding to a terminal item

$$X \rightarrow w$$

translates to

$$\forall u\, x([w|u], u). \tag{3}$$

where again, the variable u is intended to range over string positions.

Lists or sequences in the DCG approach are formalized as follows. The constant [] is interpreted as the empty list and the binary function symbol $[\cdot|\cdot]$ is interpreted as the list constructor, where $[f|r]$ is interpreted as the

[7] "Bottom-up" here refers to direction of the implication arrow (from children to parents). Backward-chaining proof procedures, such as Prolog's SLD proof procedure (Lloyd 1984) using these axioms will traverse the nodes in the parse tree in a top-down fashion.

list whose first element is f and whose suffix is the list r. Further, $[x_0, \ldots, x_n]$ is an abbreviation for $[x_0|[\ldots|[x_n|[]]]]$, the list containing $x_0 \ldots x_n$. For example, three-element list containing a, b and c in that order is written [a, b, c], which is an abbreviation for [a|[b|[c|[]]]].

The recognition problem for sample grammar is formalized in this approach as follows.

$$D_0 = \left\{ \begin{array}{l} \forall x\, y\, z\ s(x, z) \leftarrow np(x, y) \wedge vp(y, z) \\ \forall x\, y\, z\ vp(x, z) \leftarrow v(x, y) \wedge np(y, z) \\ \forall x\ np([uther|x], x) \\ \forall x\ np([knights|x], x) \\ \forall x\ vp([sleeps|x], x) \\ \forall x\ vp([like|x], x) \end{array} \right\}$$

From these axioms we can draw conclusions such as

$$D_0 \models s([uther, sleeps], [])$$

which is interpreted as that *Uther sleeps* is a well-formed expression of category S.

2.1.2. *DCG formulation of the Parsing Problem*

The parsing problem can be formalized by translating each non-terminal category into a three-place relation symbol, where the additional argument (here, the first argument) is the parse tree associated with each constituent. That tree is encoded as a term where the name of the principal functor is the label of the root node, and its arguments are the tree's immediate subtrees.

Thus each production in the grammar expanding to non-terminals

$$X \rightarrow X_0\, X_1 \ldots X_k$$

is translated to

$$\forall u_0 \ldots u_k v_0 \ldots v_k\ x(x(v_0, \ldots, v_k), u_0, u_k) \leftarrow x_0(v_0, u_0, u_1) \wedge \cdots \wedge x_k(v_k, u_{k-1}, u_k) \tag{4}$$

and each production expanding to a terminal item

$$X \rightarrow w$$

is translated to

$$\forall u\ x(x(w), [w|u], u). \tag{5}$$

Thus the parsing problem for the sample grammar is formalized as follows.

$$D_1 = \left\{ \begin{array}{l} \forall x\, y\, z\, u\, v\ \mathsf{s}(\mathsf{s}(u, v), x, z) \leftarrow \mathsf{np}(u, x, y) \wedge \mathsf{vp}(v, y, z) \\ \forall x\, y\, z\, u\, v\ \mathsf{vp}(\mathsf{vp}(u, v), x, z) \leftarrow \mathsf{v}(u, x, y) \wedge \mathsf{np}(v, y, z) \\ \forall x\ \mathsf{np}(\mathsf{np}(\mathsf{uther}), [\mathsf{uther} | x], x) \\ \forall x\ \mathsf{np}(\mathsf{np}(\mathsf{knights}), [\mathsf{knights} | x], x) \\ \forall x\ \mathsf{vp}(\mathsf{vp}(\mathsf{sleeps}), [\mathsf{sleeps} | x], x) \\ \forall x\ \mathsf{v}(\mathsf{v}(\mathsf{like}), [\mathsf{like} | x], x) \end{array} \right\} \quad (6)$$

From these axioms we can deduce the following theorem,

$$D_1 \vdash \mathsf{s}(\mathsf{s}(\mathsf{np}(\mathsf{uther}), \mathsf{vp}(\mathsf{sleeps})), [\mathsf{uther}, \mathsf{sleeps}], []),$$

which demonstrates that *Uther sleeps* is a well-formed expression of category S with the parse tree represented by the term

$$\mathsf{s}(\mathsf{np}(\mathsf{uther}), \mathsf{vp}(\mathsf{sleeps})).$$

The binary relation wf, which holds of an utterance and one of its well-formed linguistic structures or analyses, can now be defined as follows.

$$\forall u\, a\ \mathsf{wf}(u, \mathsf{s}) \leftarrow \mathsf{s}(a, u, []).$$

2.1.3. *Syntactic features in the DCG approach*

The standard way to incorporate syntactic features to a Definite Clause Grammar is by adding "extra arguments" to the predicates that define the string positions spanned by syntactic categories. The values of the additional arguments encode particular syntactic features.

This sort of encoding of syntactic features is sometimes called a *position-value encoding*, since each syntactic feature is associated with a specific argument position in each predicate.[8]

In the example below an argument has been inserted in initial position in the np, vp and v predicates to represent number agreement features. The constant sg indicates singular number agreement, and pl indicates plural number agreement. The universally quantified variables a and b range over these agreement values, and a is used in the first two clauses of D_2 to specify a feature-passing requirement enforcing subject-verb agreement.

[8] More complex encodings of syntactic features are possible, e.g. by using compound terms to represented heirarchically structured features.

$$D_2 = \left\{ \begin{array}{l} \forall x\, y\, z\, u\, a\ s(s(u, v), x, z) \leftarrow np(a, u, x, y) \wedge vp(a, v, y, z) \\ \forall x\, y\, z\, u\, a\, b\ vp(a, vp(u, v), x, z) \leftarrow v(a, u, x, y) \wedge np(b, v, y, z) \\ \forall x\ np(sg, np(uther), [uther|x]) \\ \forall x\ np(pl, np(knights), [knights|x], x) \\ \forall x\ vp(sg, vp(sleeps), [sleeps|x], x) \\ \forall x\ v(pl, vp(like), [like|x], x) \end{array} \right\} \tag{8}$$

With this modification, it is now the case that

$$D_2 \nvdash \exists s\ wf([knights, sleeps], s)$$

corresponding to the fact that the verb *sleeps* disagrees in number with the noun phrase *knights*.

2.2. *Feature Structure Grammar*

In the feature-structure approach, a well-formed linguistic structure is conceptualized as a model that satisfies the constraints imposed by the grammar. Informally, the constraints corresponding to a context-free grammar require the existence of nodes in a 'top-down' fashion. First, the existence of a root node of appropriate category is required (say, by introducing a constant symbol to name it). Then a constraint is imposed which requires the existence of the other nodes needed to form a well-formed linguistic structure as determined by the grammar. More specifically, this constraint takes the form of a universally-quantified disjunction, which requires that if n is a node of category X then either:

- n is a non-terminal node that dominates nodes $n_0 \ldots n_k$ of categories $X_0 \ldots X_k$, the grammar contains a production $X \rightarrow X_0 \ldots X_k$ and the string positions of n and $n_0 \ldots n_k$ are appropriately related, or
- n is a terminal node whose string position spans the terminal item w, and the grammar contains a production $X \rightarrow w$.

Two other constraints must also be imposed if we want the interpretations that satisfy these constraints to correspond to context-free grammar parses. These are:

- no node dominates itself, and
- there are only finitely many nodes.

If we permit a node to dominate itself in a linguistic structure, then we

may incorrectly classify structures with cyclic dominance relations as well-formed, even though such structures cannot correspond to context-free grammar parse trees. For example, suppose the grammar contains only the single production $S \rightarrow S$, where S is the category of the root node. This grammar generates the empty language, and so should have no well-formed linguistic structures. But an interpretation containing one node (which must be the root node) labelled S which directly dominates itself satisfies the grammar constraints described above, regardless of the string positions (and hence the terminal string) associated with that node.

Similarly, if we permit infinitely many nodes in a linguistic structure, we may incorrectly classify such structures as well-formed. Consider the same grammar as before. Construct an interpretation that contains an infinite set of nodes $n_0, n_1, \ldots$ all labelled S, where n_0 is the root node and each n_i immediately dominates n_{i+1}. All of the nodes are associated with the same left and right string positions and so span the same terminal string. Even at this informal level, it is easy to see that each node n_i in this interpretation satisfies the constraints imposed by the grammar, regardless of the terminal string they actually span.

The constraint requiring the irreflexivity of the dominance relation is easy to state in first-order logic, and will be provided below. Unfortunately, the constraint requiring that the number of nodes is finite cannot be stated in first-order logic. However, a stronger version of this constraint, called the Off-line Parsability Constraint (Kaplan and Bresnan 1982, Pereira and Warren 1983), can be formalized in first-order logic, and is presented below. The Off-line Parsability Constraint is in fact sufficient to ensure the decidability of the recognition problem for feature-structure grammars with syntactic features encoded by equality constraints, as shown in the appendix.

We now proceed to show in detail how constraints corresponding to a context-free grammar can be expressed using first-order formulae. The formalization presented in this section may look unfamiliar because it is expressed in the language of first-order logic rather than one of the specialized feature structure languages usually used, but in fact it is closely related to standard treatments, such as the one provided by Manaster-Ramer and Rounds (1987).

2.2.1. *FS formulation of the Recognition Problem*

The recognition problem in this approach is formalized as a union of five sets of formulae that must be simultaneously satisfied by any interpretation

that possesses the structure of a well-formed linguistic representation.[9] The first two sets of formulae F_g and F_c encode the grammar. The third set of formulae F_d defines predicates such as dominates and directly-dominates. The fourth set F_{olp} specifies the Off-line Parsability Constraint, and the fifth set F_u specifies the constraints imposed by the utterance.

In the intended interpretations $\mathsf{node}(n)$ is true of a node n in a syntactic tree, $\mathsf{cat}(n)$ is n's syntactic category and $\mathsf{left}(n)$ and $\mathsf{right}(n)$ are its left and right string positions respectively.

The first set of formulae F_g, which encodes the grammar, consists of a single formulae. This formula expresses the grammatical constraints described informally at the start of this section. It has the form of a universally quantified implication, where Δ is specified below.

$$F_g = \{\forall \mathsf{n}\ \mathsf{node(n)} \rightarrow \Delta\}$$

In F_g, Δ is a disjunction whose disjuncts correspond to the productions of the grammar in the following way.

For each production in the grammar expanding to non-terminals

$$X \rightarrow X_0 \ldots X_k$$

the disjunction Δ contains a disjunct of the form

$$\begin{aligned}\exists \mathsf{n}_0 \ldots \mathsf{n}_k(&\mathsf{node}(\mathsf{n}_0) \wedge \cdots \wedge \mathsf{node}(\mathsf{n}_k)\\ &\wedge \mathsf{cat(n)} = x \wedge \mathsf{cat}(\mathsf{n}_0) = x_0 \wedge \cdots \wedge \mathsf{cat}(\mathsf{n}_k) = x_k\\ &\wedge \mathsf{directly\text{-}dominates}(\mathsf{n}, \mathsf{n}_0, \ldots, \mathsf{n}_k))\end{aligned}$$

and for each production expanding to a terminal

$$X \rightarrow w$$

the disjunction Δ contains a disjunct of the form

$$\mathsf{cat(n)} = x \wedge \mathsf{preterminal}(\mathsf{n}, w).$$

In these formulae attributes are formalized as unary functions:[10] e.g. $\mathsf{cat}(n) = x$ requires that n's category is x. The relations directly-dominates and preterminal will be defined below; informally directly-dominates $(n, n_0, \ldots, n_k)$ requires that node n directly dominates the sequence

[9] These models may also possess 'junk', e.g. structure disconnected from the linguistic representation we are interested in. Since the constraints ensure that the submodel corresponding to the linguistic structure is itself well-formed, the additional structure is of no concern.

[10] Partial functions are used to model attributes in Johnson (1990a,b). The use of total rather than partial functions here is discussed below.

of nodes $n_0 \ldots n_k$, and preterminal(n, w) requires that the string of terminals dominated by node n is the unit string w.

The set F_g associated with the sample grammar is shown as F_{g0} below. If an interpretation satisfies this formula, then every node must be of category s, np or vp, and if the node is not a preterminal node then there must exist other nodes of appropriate categories that it directly dominates.

$$
F_{g0} = \left\{ \begin{array}{l}
\forall n\ \mathsf{node}(n) \rightarrow ((\exists n_0 n_1\ \mathsf{node}(n_1) \wedge \mathsf{node}(n_2) \\
\qquad\qquad \wedge\ \mathsf{cat}(n) = \mathsf{s} \\
\qquad\qquad \wedge\ \mathsf{cat}(n_0) = \mathsf{np} \wedge \mathsf{cat}(n_1) = \mathsf{vp} \\
\qquad\qquad \wedge\ \mathsf{directly\text{-}dominates}(n, n_0, n_1)) \\
\qquad \vee\ (\exists n_0 n_1\ \mathsf{node}(n_1) \wedge \mathsf{node}(n_2) \\
\qquad\qquad \wedge\ \mathsf{cat}(n) = \mathsf{vp} \\
\qquad\qquad \wedge\ \mathsf{cat}(n_0) = \mathsf{v} \wedge \mathsf{cat}(n_1) = \mathsf{np} \\
\qquad\qquad \wedge\ \mathsf{directly\text{-}dominates}(n, n_0, n_1)) \\
\qquad \vee\ (\mathsf{cat}(n) = \mathsf{np} \wedge \mathsf{preterminal}(n, \mathsf{uther})) \\
\qquad \vee\ (\mathsf{cat}(n) = \mathsf{np} \wedge \mathsf{preterminal}(n, \mathsf{knights})) \\
\qquad \vee\ (\mathsf{cat}(n) = \mathsf{vp} \wedge \mathsf{preterminal}(n, \mathsf{sleeps})) \\
\qquad \vee\ (\mathsf{cat}(n) = \mathsf{v} \wedge \mathsf{preterminal}(n, \mathsf{like})))
\end{array} \right\}
$$

It is interesting to note that the implication sign in the single formula of F_g points in the opposite direction to the implication signs in the corresponding DCG axiomatization D_0. Informally, this is because in the DCG approach the well-formedness of the utterance is shown by proving that it is a consequence of the grammar axioms, whereas in the FS approach the well-formedness is shown by ensuring that the constraints derived from the utterance are not inconsistent with the grammar constraints.

The second set of formulae F_c defines the one-place predicates category and terminal (which will be used in the Off-line Parsability Constraint) and introduces inequalities that require that distinct constants naming categories and lexical items denote distinct individuals (Johnson 1990a, b, 1991a, b; Smolka 1992). In many treatments these inequalities are 'built in', i.e. specified implicitly in the interpretation procedure, but for explicitness here they are listed here.

The one-place predicates category and terminal are defined as follows. If the non-terminal categories of the grammar are $X_1, \ldots, X_m$, then F_c contains the following formula.

$$
\forall x \left(\mathsf{category}(x) \leftrightarrow \bigvee_{0<k\leq m} x = x_k \right). \tag{9}
$$

Similarly, if the terminal items introduced by the grammar are $w_1, \ldots, w_n$, then F_c contains the following formula.

$$\forall w\left(\mathsf{terminal}(w) \leftrightarrow Z \bigvee_{0<k\leq n} w = w_k\right). \tag{10}$$

In addition, F_c also contains inequalities that require that certain distinct constant symbols denote distinct individuals. For all pairs w_i, w_j of distinct terminal items in the grammar F_c contains an inequality of the form

$$w_i \neq w_j$$

(where for the purposes of this schema, the empty list nil is considered a terminal item).

Finally, for all pairs x_i, x_j of non-terminal categories in the grammar, F_c contains an inequality of the form

$$x_i \neq x_j.$$

These inequalities require that the syntactic categories are (pairwise) distinct and that the lexical items and nil are also (pairwise) distinct.

For the example grammar the following formulae appear in F_c.

$$F_{c_0} = \left\{ \begin{array}{cc} \forall x\ \mathsf{category}(x) \leftrightarrow (x = s \vee x = np \vee x = np) & \\ \forall x\ \mathsf{terminal}(x) \leftrightarrow (x = \mathsf{knights} \vee x = \mathsf{uther} \vee & \\ \qquad x = \mathsf{like} \vee x = \mathsf{sleeps}) & \\ \mathsf{uther} \neq \mathsf{knights} & \mathsf{uther} \neq \mathsf{sleeps} \\ \mathsf{uther} \neq \mathsf{like} & \mathsf{uther} \neq \mathsf{nil} \\ \mathsf{knights} \neq \mathsf{sleeps} & \mathsf{knights} \neq \mathsf{like} \\ \mathsf{knights} \neq \mathsf{nil} & \mathsf{sleeps} \neq \mathsf{like} \\ \mathsf{sleeps} \neq \mathsf{nil} & \mathsf{like} \neq \mathsf{nil} \\ s \neq np & s \neq vp \\ np \neq vp & \mathsf{sg} \neq \mathsf{pl} \end{array} \right\}$$

Now we turn to the third set of formulae F_d, which includes the definitions of the predicates directly-dominates and preterminal, as well as formulae that stipulate properties of the functions first and rest that are used to construct lists in the manner described in Shieber (1986). The utterance's sequence of terminal items is represented as such a list, and the functions left and right map nodes into sublists of this list.

The two-place relation preterminal requires that its first argument is a node whose string position spans exactly the terminal item given as the second argument. It is defined as follows (this definition should be com-

pared with the DCG axiom schema given in (3)). The first two conjuncts in the implication's antecedent in effect restrict the quantification to range over nodes and terminals. This suffices, since the only other place that the preterminal relation appears is in the single formula in F_g, and there the two arguments are required to be a node and a terminal respectively. The quantification restriction simplifies the proof of decidability of the recognition problem with the Off-line Parsability Constraint imposed.

$$\begin{aligned}\forall n\ w(&\mathsf{node}(n) \wedge \mathsf{terminal}(w) \wedge \mathsf{preterminal}(n, w)) \\ &\rightarrow (\mathsf{first}(\mathsf{left}(n)) = w \wedge \\ &\quad \mathsf{rest}(\mathsf{left}(n)) = \mathsf{right}(n)).\end{aligned} \tag{11}$$

The relations directly-dominates $(n, n_0, \ldots, n_k)$ require that the string positions of the parent node n stand in an appropriate structural relationship to the string positions of its children nodes $n_0 \ldots, n_k$, and that n dominates its children. One such relation is defined for each different production length n of grammar rules. Just as in the previous formula, the first $k + 1$ conjuncts in the implication's antecedent effectively restrict the quantification to range over $k + 1$-tuples of nodes.

$$\begin{aligned}\forall n\ n_0 \ldots n_k\ (&\mathsf{node}(n) \wedge \mathsf{node}(n_0) \wedge \cdots \wedge \mathsf{node}(n_k) \wedge \\ &\mathsf{directly\text{-}dominates}(n, n_0, \ldots, n_k)) \rightarrow \\ &(\mathsf{left}(n) = \mathsf{left}(n_0) \wedge \mathsf{right}(n) = \mathsf{right}(n_k) \wedge \\ &\mathsf{right}(n_0) = \mathsf{left}(n_1) \wedge \cdots \wedge \mathsf{right}(n_{k-1}) = \mathsf{left}(n_k) \wedge \\ &\mathsf{dominates}(n, n_0) \wedge \cdots \wedge \mathsf{dominates}(n, n_k)).\end{aligned} \tag{12}$$

The next two constraints require that the dominates relation is transitively closed and irreflexive, thus requiring that the directly-dominates relation is acyclic.

$$\begin{aligned}\forall n_1\ n_2\ n_3(&\mathsf{node}(n_1) \wedge \mathsf{node}(n_2) \wedge \mathsf{node}(n_3) \wedge \\ &\mathsf{dominates}(n_1, n_2) \wedge \mathsf{dominates}(n_2, n_3)) \\ &\rightarrow \mathsf{dominates}(n_1, n_3).\end{aligned} \tag{13}$$

$$\forall n(\mathsf{node}(n) \rightarrow \neg \mathsf{dominates}(n, n)). \tag{14}$$

Finally, we turn to the constraints defining the properties of lists. A constraint stipulating that the empty list element nil contains no elements is required. The encoding of the parsing problem requires in addition a constraint that every node is also distinct from nil. (These constraints could be implicitly stated in a sorted logic, as discussed below).

Since features are modeled by total functions here, first and rest are defined on all elements in the domain of the model, including nil. In general the value of first and rest on non-list elements is irrelevant, but

it is necessary to ensure that nil is distinct from any nonempty list element. The following formulae enforce these constraints for lists of nodes.

$$\mathsf{first(nil)} = \mathsf{nil}. \tag{15}$$

$$\mathsf{rest(nil)} = \mathsf{nil}. \tag{16}$$

$$\forall \mathsf{x}\ \mathsf{node(x)} \rightarrow \mathsf{x} \neq \mathsf{nil}. \tag{17}$$

This completes the third set of constraints. The set of formulae (11) through (17) inclusive is called F_d.

As mentioned above, it is necessary to impose a constraint that bounds the number of nodes in a model. Although a constraint satisfied in exactly the interpretations containing a finite number of nodes cannot be stated in first-order logic, the Off-line Parsability Constraint can be. The Off-line Parsability Constraint requires that there is at most one node for any given triple of left string-position, right string-position and category label.[11] The predicate string-position will be defined below; it is true of exactly the string-positions in the utterance being recognized. The appendix shows that the satisfiability problem for feature-structure grammars as formalized here with the Off-line Parsability Constraint is decidable.

$$F_{olp} = \left\{ \begin{array}{l} \forall \mathsf{n}\ \mathsf{node(n)} \rightarrow \mathsf{string\text{-}position(left(n))}. \\ \forall \mathsf{n}\ \mathsf{node(n)} \rightarrow \mathsf{string\text{-}position(right(n))}. \\ \forall \mathsf{n}\ \mathsf{node(n)} \rightarrow \mathsf{category(cat(n))}. \\ \forall \mathsf{n}_1\ \mathsf{n}_2\ (\mathsf{node}(\mathsf{n}_1) \wedge \mathsf{node}(\mathsf{n}_2) \wedge \\ \qquad \mathsf{left}(\mathsf{n}_1) = \mathsf{left}(\mathsf{n}_2) \wedge \\ \qquad \mathsf{right}(\mathsf{n}_1) = \mathsf{right}(\mathsf{n}_2) \wedge \\ \qquad \mathsf{cat}(\mathsf{n}_1) = \mathsf{cat}(\mathsf{n}_2)) \rightarrow \mathsf{n}_1 = \mathsf{n}_2. \end{array} \right\}$$

Finally, the utterance itself imposes constraints on the linguistic structure. The set of formulae F_u express these constraints. Specifically, F_u contains the formulae

$$\mathsf{node(root)}, \tag{18}$$

$$\mathsf{cat(root)} = \mathsf{s} \tag{19}$$

which require that there is a root node (denoted by the constant root) whose category is s.

The string of terminals of the utterance is encoded as a set of equations.

[11] Shieber (1992) presents a generalization of the Off-line Parsability Constraint discussed here that does not require that every node is associated with an atomic category, but is still sufficient to ensure decidability.

For each terminal w_i, if w_i is the ith (terminal) item in the utterance to be recognized, F_u contains an equation of the form

$$\mathsf{first}(\mathsf{rest}^{i-1}(\mathsf{left}(\mathsf{root}))) = w_i. \tag{20}$$

In addition, F_u contains the following equations, where n is the number of terminal items in the utterance to be recognized. The equations (20–22) encode the string being analysed and require that the 'end' of the terminal sequence is terminated in the empty list.

$$\mathsf{rest}^n(\mathsf{left}(\mathsf{root})) = \mathsf{nil}, \tag{21}$$

$$\mathsf{right}(\mathsf{root}) = \mathsf{nil}. \tag{22}$$

All that remains is to define the predicate **string-position**, which is true of exactly the string positions.

$$\forall \mathsf{p}\ \mathsf{string\text{-}position}(\mathsf{p}) \leftrightarrow \bigvee_{0 \leq k \leq n} \mathsf{p} = \mathsf{rest}^k(\mathsf{left}(\mathsf{root})). \tag{23}$$

The formulae (18–23) constitute the members of the set F_u.

For the sample utterance *Uther sleeps*, the following formulae comprise F_u.

$$F_u = \left\{ \begin{array}{l} \mathsf{node}(\mathsf{root}) \\ \mathsf{cat}(\mathsf{root}) = \mathsf{s} \\ \mathsf{first}(\mathsf{left}(\mathsf{root})) = \mathsf{uther} \\ \mathsf{first}(\mathsf{rest}(\mathsf{left}(\mathsf{root}))) = \mathsf{sleeps} \\ \mathsf{rest}(\mathsf{rest}(\mathsf{left}(\mathsf{root}))) = \mathsf{nil} \\ \mathsf{right}(\mathsf{root}) = \mathsf{nil} \\ \forall \mathsf{p}\ \mathsf{string\text{-}position}(\mathsf{p}) \leftrightarrow \mathsf{p} = \mathsf{left}(\mathsf{root}) \vee \\ \qquad\qquad\qquad\qquad\qquad \mathsf{p} = \mathsf{rest}(\mathsf{left}(\mathsf{root})) \vee \\ \qquad\qquad\qquad\qquad\qquad \mathsf{p} = \mathsf{rest}(\mathsf{rest}(\mathsf{left}(\mathsf{root}))) \end{array} \right\}$$

The recognition problem is then equivalent to the problem of determining the satisfiability of

$$F = \{F_g \cup F_c \cup F_d \cup F_{olp} \cup F_u\}.$$

Thus the well-formedness of the utterance *Uther sleeps* is expressed by the existence of a satisfying model $\mathcal{M}$ such that

$$\mathcal{M} \models F_{g_0} \cup F_{c_0} \cup F_d \cup F_{olp} \cup F_u.$$

In general, if all of the constraints are satisfiable then there are infinitely

many models that satisfy these constraints. This is usually the case in feature-structure systems.[12]

2.2.2. *FS Formulation of the Parsing Problem*

The parsing problem can be formalized by means of an additional attribute **children** that encodes the immediate dominance relationships between nodes.[13] To encode the parse tree as part of the feature structure a new function where for every node n, **children**(n) is a list of n's immediate descendants.

This can be done by including the appropriate instances of the following schema in F_d (we call the resulting set F'_d).

$$\forall n\ w\ (\mathsf{node}(n) \wedge \mathsf{terminal}(w) \wedge \mathsf{preterminal}(n, w)) \rightarrow \mathsf{children}(n) = \mathsf{nil}. \tag{24}$$

$$\begin{aligned}\forall n\ n_0 \ldots, n_k\ (&\mathsf{node}(n) \wedge (\mathsf{node}(n_0) \wedge \cdots \wedge \mathsf{node}(n_k)) \wedge \\ &\mathsf{directly\text{-}dominates}(n, n_0, \ldots, n_k)) \\ &\rightarrow (\mathsf{first}(\mathsf{children}(n)) = n_0 \wedge \cdots \wedge \\ &\quad \mathsf{first}(\mathsf{rest}^k(\mathsf{children}(n))) = n_k \wedge \\ &\quad \mathsf{rest}^{k+1}(\mathsf{children}(n)) = \mathsf{nil})\end{aligned} \tag{25}$$

Requiring the satisfaction of (24) and (25) does not change the set of utterances accepted, but it does require that every satisfying model include an encoding of the parse tree (as the value of the **children** function on the nodes). Since the set of all constraints require that every satisfying model include some parse tree of the utterance, if a formula is a (valid) consequence of all of the constraints then it is true of every parse tree of the utterance. This can be used to obtain information about the utterance's parse tree(s). Thus for example

$$F_{g_0} \cup F_c \cup F'_d \cup F_{olp} \cup F_u \models \mathsf{cat}(\mathsf{first}(\mathsf{children}(\mathsf{root}))) = \mathsf{np}$$

since the category of the first child of the parse tree for the utterance *Uther sleeps* is NP. If the utterance were ambiguous – say if the category of the first child of the root is either an NP or an AP – then only a disjunctive consequent would follow from the union of constraints. Of course, since an utterance is well-formed iff the corresponding constraints are satisfiable, it makes no sense to investigate the consequences of an inconsistent set of constraints.

---

[12] Carpenter (1992) proposes a sorted feature structure system which does not have this property.

[13] This attribute is called **daughters** in HPSG and some other grammars.

2.2.3. *Syntactic Features in the FS Approach*

An important property of the feature structure approach is the ease with which syntactic features can be added to a grammar. A feature is treated as a function from nodes, and a feature constraint is an equality or inequality over these values.

Subject-verb agreement in the sample grammar can be treated as follows. The agreement features are encoded using the function agreement and the constants sg and pl. This change is incorporated into the grammar as follows. First, we add the inequality

$$\mathsf{sg} \neq \mathsf{pl}$$

to the inequalities in the set F_c, reflecting the fact that the singular agreement feature value (whatever we take it to be) must be distinct from the plural agreement feature value. Call the resulting set of formulae F_{c_1}.

Then we add appropriate equations that constrain the agreement feature values associated with each of the nodes.

$$F_{g_1} = \left\{ \begin{array}{l} \forall n\ \mathsf{node}(n) \rightarrow (\exists n_0 n_1\ \mathsf{node}(n_1) \wedge \mathsf{node}(n_2) \wedge \\ \qquad \mathsf{cat}(n) = \mathsf{s} \wedge \\ \qquad \mathsf{cat}(n_0) = \mathsf{np} \wedge \mathsf{cat}(n_1) = \mathsf{vp} \wedge \\ \qquad \mathsf{agreement}(n_0) = \mathsf{agreement}(n_1) \wedge \\ \qquad \mathsf{directly\text{-}dominates}(n, n_0, n_1)) \vee \\ (\exists n_0 n_1\ \mathsf{node}(n_1) \wedge \mathsf{node}(n_2) \wedge \\ \qquad \mathsf{cat}(n) = \mathsf{vp} \wedge \\ \qquad \mathsf{cat}(n_0) = \mathsf{vp} \wedge \mathsf{cat}(n_1) = \mathsf{np} \wedge \\ \qquad \mathsf{agreement}(n) = \mathsf{agreement}(n_0) \wedge \\ \qquad \mathsf{directly\text{-}dominates}(n, n_0, n_1)) \vee \\ (\mathsf{cat}(n) = \mathsf{np} \wedge \mathsf{terminal}(n, \mathsf{uther}) \wedge \\ \ \mathsf{agreement}(n) = \mathsf{sg}) \vee \\ (\mathsf{cat}(n) = \mathsf{np} \wedge \mathsf{terminal}(n, \mathsf{knights}) \wedge \\ \ \mathsf{agreement}(n) = \mathsf{pl}) \vee \\ (\mathsf{cat}(n) = \mathsf{vp} \wedge \mathsf{terminal}(n, \mathsf{sleeps}) \wedge \\ \ \mathsf{agreement}(n) = \mathsf{sg}) \vee \\ (\mathsf{cat}(n) = \mathsf{v} \wedge \mathsf{terminal}(n, \mathsf{like}) \wedge \\ \ \mathsf{agreement}(n) = \mathsf{pl}) \end{array} \right\}$$

With this modification the grammar correctly excludes the ungrammatical utterance **Knight sleeps*; that is,

$$F_{g_1} \cup F_{c_1} \cup F_d \cup F_{olp} \cup F'_u \tag{26}$$

is unsatisfiable, where

$$F'_u = \left\{ \begin{array}{l} \text{node(root)} \\ \text{cat(root)} = \text{s} \\ \text{first(left(root))} = \text{knights} \\ \text{first(rest(left(root)))} = \text{sleeps} \\ \text{rest(rest(left(root)))} = \text{nil} \\ \text{right(root)} = \text{nil} \end{array} \right\}.$$

3. Differences between the approaches

The last section sketched how the recognition and parsing problems can be conceptualized as either validity problems (as in the DCG approach) or satisfiability problems (as in the FS approach).

The methods express grammars in quite different ways. For example, consider the modifications that need to be made when an additional production is added to the grammar. In the DCG approach, an additional axiom is added to the axiom set D, logically strengthening the system. On the other hand, in the FS approach another disjunct is added to Δ in F_g, resulting in a weaker system.

We can also try to compare the approaches by comparing the computational complexity of the validity and the satisfiability problems for first order formulae.

Since the class of valid first-order formulae is recursively enumerable but not recursive, the class of satisfiable first-order formulae is co-recursively enumerable, but not recursive or recursively enumerable. Thus it would seem that the two approaches should differ in their computational properties; in the general case the set of well-formed utterances should be recursively enumerable in the DCG approach, and the set of ill-formed utterances should be recursively enumerable in the FS approach.

However, as noted above, in order to correctly encode the recognition and parsing problems the FS approach requires the additional constraint that the model be finite. If we restrict attention to *finite models*, the satisfiability problem for first-order formulae becomes recursively enumerable, but not recursive (Fagin 1990). Thus the computational complexity of the two approaches is the same. This is as expected; after all, both approaches are capable of formalizing grammars for recursively enumerable languages (see Manaster-Ramer and Rounds 1987 or Johnson 1988 for examples of how to construct such grammars).

3.1. *Equality and 'Data-structures'*

Both approaches use terms and equality to encode 'structured entities' such as strings, trees, etc. Because the DCG approach is validity-based, while the FS approach is satisfiability-based, different constructs must be used in each approach to represent similar data structures.

Satisfiability (truth in at least one interpretation) and validity (truth in all interpretations) have very different properties. For example, $g(a) = f(b)$ is satisfiable: a one-element interpretation in which a, b, $g(a)$ and $f(b)$ all (necessarily) have the same denotation makes this equation true. On the other hand this equation is not valid, since there are models in which $g(a)$ and $f(b)$ do not denote the same element.

Informally, this difference can be described as follows: the equality $t_1 = t_2$ is valid only if t_1 and t_2 are identical terms, whereas $t_1 = t_2$ is satisfiable (unless other constraints imply that the terms are not equal).

In computing valid consequences, terms behave like data structures. Since t denotes the same individual as $f(a, b)$ in all interpretations (i.e. $t = f(a, b)$ is valid) iff t actually is the term $f(a, b)$, the term $f(a, b)$ functions as a compound entity from which a and b can be recovered.

In computing satisfiability, (unary) terms behave like features. Since in the absence of equality axioms $f(a, b) = f(c, d)$ is satisfiable in an interpretation in which $a \neq c$ and $b \neq d$, the (value of the) term $f(a, b)$ does not function as a compound entity from which a and b can be recovered. On the other hand the fact that functions are single-valued can be exploited to encode attributes or features of entities: thus in any model of $f(a) = c \wedge f(a) = d$, it must be the case that $c = d$.

Thus in a validity-based system a compound data-structure with two elements a and b can be represented by a term $f(a, b)$, where f can be regarded as the name of the 'record type'. In a satisfiability-based system the same compound data-structure with two elements a and b can be represented by the conjunction

$$\mathsf{first}(x) = a \wedge \mathsf{second}(x) = b \wedge \mathsf{arity}(x) = 2.$$

Even constant symbols behave differently in satisfiability and validity-based approaches. The equation $\mathsf{a} = \mathsf{b}$ is not a valid formula, but it is a satisfiable formula. The effect of this is that the satisfiability-based approach in general requires inequality axioms that require that distinct constant symbols denote distinct entities; something that the validity-based approach does not require.

The representation of lists in the two systems reflects this difference. In a validity-based approach nonempty lists are usually represented using a

two-argument function $[\cdot|\cdot]$, where $[e|l]$ is interpreted as a list whose first element is e and whose rest is l. In a satisfiability-based approach the same list could be represented by the conjunction $\mathbf{first}(x) = e \wedge \mathbf{rest}(x) = l$.

One difficulty that arises in the particular satisfiability-based formulation here is that **first** and **rest** are interpreted as total functions (like all function symbols in a standard first-order interpretation), so they are defined on all elements, including the empty list element nil.

This is only a minor technical difficulty, and techniques for dealing with it are well-known in the literature. One way of doing this is to interpret function symbols as denoting *partial* functions, which can be modeled in first-order logic by introducing a new "equality relation" and a new element that serves as the "undefined value" as in Johnson (1988, 1990); another way is to formulate the attributes as relations as in Johnson (1991a,b). The conceptually neatest solution is probably to adopt a *sorted logic*, where the functions **first** and **rest** are only defined on the sort **proper-list**, to which nil does not belong. Such systems have been proposed in a similar setting by Höhfeld and Smolka (1988) and Carpenter (1992), and in a more linguistic setting by Pollard and Sag (1987).

In this paper a less-principled but simpler technique is used: the axioms $\mathbf{first}(\mathsf{nil}) = \mathsf{nil}$ and $\mathbf{rest}(\mathsf{nil}) = \mathsf{nil}$ ensure that the empty list "contains" nothing other than the empty list.

Note that in many ways the satisfiability-based system is more flexible than the validity-based system.

First, the validity-based approach represents basic data-structures with a fixed arity, and an element of the data structure is identified by the position it occurs in (a *position-value encoding*). The satisfiability-based approach represents data-structures of indeterminate arity, and an element is identified by the name of the function whose value it is of the compound entity (an *attribute-value encoding*). It is often easier to "add additional components" to the satisfiability-based representation of such data structures, since this can be done without having to change the arity of all occurrences of the term representing the data-structure, as has to be done in the validity-based approach.

Second, self-referential or cyclic structures cannot be directly represented in the validity based approach. For example, there is no term t such that $\mathsf{f}(t) = t$ is valid, i.e., $\mathsf{f}(t)$ and t denote the same individual in all interpretations.

But such structures can be directly represented in the satisfiability-based approach: the equation $\mathbf{first}(x) = x$ is satisfiable.

Third, the data structures in the validity-based approach are terms (hence *finite* by definition), and so can only represent elements from

countable sets of data structures. No such restriction holds for the satisfiability-based approach, and imposing a finiteness requirement on models changes the computational properties of the system, as noted above.

3.2. *Reification and Ambiguity*

In the DCG approach all of the infinitely many linguistic structures are simultaneously represented in the minimal Herbrand model of the DCG axioms, whereas in the FS approach a model corresponds to a single linguistic structure. That is, in the DCG approach the linguistic structures are *reified*, i.e. considered as individuals, (infinitely) many of which are simultaneously present in the universe of discourse, while in the FS approach only the component "nodes" of one linguistic structure are always individuals in the domain.

This difference is reflected in the treatment of *ambiguous utterances*. In the DCG approach if an utterance u is ambiguous with respect to a grammar with a DCG axiomatization D and t_1 and t_2 are the terms representing two of its different parse trees, then

$$D \models \mathsf{wf}(u, t_1) \wedge \mathsf{wf}(u, t_2).$$

Informally, since both structures are "present" in the minimal Herbrand model of D, ambiguity is reflected by a conjunction of consequences, each corresponding to the different analyses. That is, we can infer that both t_1 and t_2 are structures for s.

On the other hand, in the FS approach, if F is a set of (satisfiable) feature-structure constraints corresponding to an ambiguous utterance and grammar, and A_1 and A_2 are formulae describing two different parse trees that satisfy these constraints, then in general

$$F \not\models A_1 \wedge A_2$$

but only the weaker

$$F \models A_1 \vee A_2$$

holds.

This is because a model for F represents a single well-formed structure, and an ambiguous utterance will be one where there are two essentially different models for F, say $\mathcal{M}_1$ and $\mathcal{M}_2$, where $\mathcal{M}_1 \models A_1$ and $\mathcal{M}_2 \models A_2$. Thus in the feature-structure approach, ambiguity is reflected as a *disjunction* of consequences.

The 'disjunctive' treatment of ambiguity that follows in the FS approach is in some ways more natural, since an ambiguous utterance (with perhaps

the exception of jokes involving ambiguity, etc.) probably has only one interpretation, even though it is not clear from the words in the utterance which interpretation that is.

In defense of the 'conjunctive' treatment of ambiguity, one might argue that a grammar only serves to relate the words of utterances with their possible parse trees, so that ambiguous utterances actually have two parse trees, even if only one interpretation.

In fact, in some applications it may be necessary to reify linguistic structures, as the DCG approach does. For example, Chomsky's (1988) new 'economy' theory requires that a well-formed linguistic structure (here, a quadruple consisting of a D-structure, an S-structure, and LF and PF representations) be produced by as few 'transformational operations' as possible. The economy requirement of such a theory might be formalizable as a quantification over linguistic structures; the well-formed structures being the ones involving no more transformational operations than any other structure with the same D-structure.

4. Conclusion

Given that both use the language of first-order logic, it is perhaps surprising how different the two formalizations of the same context-free grammar are. The difference arises from the fact that the DCG formalization formalizes the difference between grammatical and ungrammatical utterances as the difference between valid and invalid formulae, whereas the feature structure formalization formalizes the difference between grammatical and ungrammatical utterances as the difference between satisfiable and unsatisfiable formulae.

There are a number of interesting questions about the relationship between the DCG and the FS formalizations remaining to be answered.

For example, it would be interesting to know to what extent these two approaches can be unified. That is, can we devise an axiomatization A where the formula corresponding to the parsing problem (say $\mathsf{wf}(u, s)$, as in the DCG approach) is satisfiable iff it is valid? This is equivalent to saying that for all u and s, either $A \models \mathsf{wf}(u, s)$ or $A \models \neg \mathsf{wf}(u, s)$, i.e., wf *represents* the parse relation in the theory defined by A. Stabler (1992) answers this question affirmitively for CFGs, but also points out that the theory of CFGs is essentially incomplete (because it includes the theory of strings, and there is no complete axiomatization of the theory of strings).

It may also be profitable to see if the relationship between the DCG and the FS approaches is similar to other familiar relationships between theories. For example, it seems that the feature structure formalization

is closely related to completion (Clark 1978, Lloyd 1984) of the DCG axioms.

Finally, it might be possible to generalize the validity-based and satisfiability-based approaches away from the first-order languages used in this paper. Ideally, such a treatment would encompass a wider class of methods for formalizing grammars, including those based on fixed-point constructions, such as Rounds' (1988) LFP systems.

BIBLIOGRAPHY

Carpenter, R.: 1992, *The Logic of Typed Feature Structures*, Cambridge Tracts in Theoretical Computer Science 32, Cambridge University Press, Cambridge, England.

Chen, W. and D. S. Warren: 1989, *Abductive Logic Programming*, m.s., Department of Computer Science, State University of New York at Stony Brook.

Chomsky, N.: 1988, *Some Notes on Economy of Derivation and Representation*, m.s., Department of Linguistics and Philosophy, Massachusetts Institute of Technology.

Clark, K. L.: 1978, 'Negation as Failure', in H. Gallaire and J. Minker (eds.), *Logic and Databases*, Plenum Press, New York.

Colmerauer, A.: 1978, 'Metamorphosis Grammars', in Leonard Bloc (ed.), *Natural Language Communication with Computers*, Springer-Verlag, Berlin, Germany.

Fagin, R.: 1990, 'Finite-Model Theory – a Personal Perspective', in S. Abiteboul and P. C. Kannellakis (eds.), *ICDT '90: 3rd International Conference on Database Theory*, Springer-Verlag, New York.

Gallier, J.: 1986, *Logic for Computer Science*, Harper and Row, New York.

Höhfeld, M. and G. Smolka: 1988, *Definite Relations over Constraint Languages*, LILOG Report 53, IBM Deutschland, Stuttgart.

Johnson, M.: 1988, *Attribute Value Logic and the Theory of Grammar*, CSLI Lecture Notes Series, University of Chicago Press.

Johnson, M.: 1990a, 'Expressing Disjunctive and Negative Feature Constraints in Classical First-Order Logic', *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, Pittsburgh, Pennsylvania.

Johnson, M.: 1990b, 'Features, Frames and Quantifier-Free Formulae', in P. Saint-Dizier and S. Szpakowicz (eds.), *Logic and Logic Grammars for Language Processing*, Ellis Horwood, New York.

Johnson, M.: 1991a, 'Features and Formulae', *Computational Linguistics* **17**(1), 131–152.

Johnson, M.: 1991b, 'Logic and Feature Structures', *Proceedings of IJCAI-91*, Sydney, Australia.

Johnson, M.: to appear, 'Computing with Features as Formulae', *Computational Linguistics*.

Kaplan, R. and J. Bresnan: 1982, 'Lexical-Functional Grammar: A Formal System for Grammatical Representation', in J. Bresnan (ed.), *The Mental Representation of Grammatical Relations*, The MIT Press.

Kasper, R. and W. Rounds: 1990, 'The Logic of Unification in Grammar', *Linguistics and Philosophy* **13**(1), 35–58.

Kay, M.: 1979, 'Functional Unification Grammar', *Proceedings of the 5th Annual Meeting of the Berkeley Linguistics Association*, Berkeley, California.

Lloyd, J. W.: 1984, *Foundations of Logic Programming*, Springer-Verlag, Berlin.

Manaster-Ramer, A. and W. Rounds: 1987, 'A Logical Version of Functional Grammar', *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford University, Palo Alto, California.

Nelson, G. and Oppen, D. C.: 1980, 'Fast Decision Procedures Based on Congruence Closure', *J. ACM* **27**(2), 356–364.

Pereira, F.: 1982, *Logic for Natural Language Analysis*, Ph.D. thesis, The University of Edinburgh. Reprinted as Technical Note 275, Artificial Intelligence Center, SRI International, Menlo Park, California.

Pereira, F. and S. Shieber: 1984, 'The Semantics of Grammar Formalisms seen as Computer Languages', *The Proceedings of COLINC 1984 Association for Computational Linguistics*, Stanford University, Palo Alto, California.

Pereira, F. and S. Shieber: 1987, *Prolog for Natural Language Analysis*, CSLI Lecture Notes Series, University of Chicago Press.

Pereira, F. and D. H. D. Warren: 1983, 'Parsing as Deduction', *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, MIT, Cambridge, Massachusetts.

Pollard, C. and I. Sag: 1988, *Head-Driven Phrase Structure Grammar*, CSLI Lecture Notes Series, University of Chicago Press.

Rounds, W. C.: 1988, 'LFP: A Logic for Linguistic Descriptions and an Analysis of its Complexity', *Computational Linguistics* **14**(4), 1–10.

Stabler, E.: 1992, *The Logical Approach to Syntax: Foundations, Specifications and Implementations of Theories of Government and Binding*. The MIT Press, Cambridge, Massachusetts.

Shieber, S.: 1986, *An Introduction to Unification-Based Theories of Grammar*, CSLI Lecture Notes Series, University of Chicago Press.

Shieber, S.: 1992, *Constraint-Based Grammar Formalisms*, The MIT Press, Cambridge, Massachusetts.

Smolka, G.: 1992, 'Feature-Constraint Logics for Unification Grammars', *Journal of Logic Programming* **12**(1&2), 51–88.

APPENDIX

This appendix shows that if there is an interpretation which satisfies all of the formulae in the sets of feature-structure constraints (including the Off-line Parsability Constraint), then there is a finite model which satisfies these formulae. This implies the decidability of systems of such constraints.

Intuitively, the proof is very similiar to the one given in Johnson (1988). First, we show that the Off-line Parsability Constraint implies that the number of individuals that satisfy the **node** predicate is finite. Then we show that because the domain of quantification is essentially restricted to the domains of the finite predicates **node**, **terminal**, **category** and **string-position**, the satisfiability problem for the original feature-structure formulae can be reduced to the satisfiability problem for formulae in the quantifier-free subset first-order logic. Since the latter problem is decidable (Nelson and Oppen 1980, Gallier 1986) the former is too. Moreover, models for the quantifier-free formulae are easily modified to provide finite models for the original feature-structure formulae.

LEMMA 1. *Let* $F = \{F_g \cup F_c \cup F_d \cup F_{olp} \cup F_u\}$ *be a feature-structure for-*

mulae set of the kind described above, and suppose $\mathcal{M} \models F$. *Then* $[\![\mathsf{node}]\!]_{\mathcal{M}}$, $[\![\mathsf{category}]\!]_{\mathcal{M}}$, $[\![\mathsf{terminal}]\!]_{\mathcal{M}}$ *and* $[\![\mathsf{string\text{-}position}]\!]_{\mathcal{M}}$ *are all finite sets.*

Because $\mathcal{M} \models F$, any model $\mathcal{M}$ must satify the formulae (9), (10) and (23), so $[\![\mathsf{category}]\!]_{\mathcal{M}}$, $[\![\mathsf{terminal}]\!]_{\mathcal{M}}$ and $[\![\mathsf{string\text{-}position}]\!]_{\mathcal{M}}$ are all finite sets.

Now, the fourth formula in F_{olp} requires that the value of at least one of the functions left, right or cat must differ on every pair of distinct nodes n_1 and n_2. But the first three formulae in F_{olp} require that for any node n, these functions take their values over the finitely bounded sets just identificd ($[\![\mathsf{left}(n)]\!]_{\mathcal{M}}$ and $[\![\mathsf{right}(n)]\!]_{\mathcal{M}}$ range over $[\![\mathsf{string\text{-}position}]\!]_{\mathcal{M}}$, and $[\![\mathsf{cat}(n)]\!]_{\mathcal{M}}$ ranges over $[\![\mathsf{category}]\!]_{\mathcal{M}}$). Hence the $[\![\mathsf{node}]\!]_{\mathcal{M}}$ is also a finite set. ■

The next technical lemma establishes the satisfiability of a certain class of first-order sentences.

LEMMA 2. *Let* ψ *be any formula of the form* $\forall x_1 \ldots x_k(p_1(x_1) \wedge \cdots \wedge p_k(x_k) \rightarrow \varphi)$, *where* φ *is any quantifier-free formula. Suppose there is an interpretation* $\mathcal{M}$ *such that* $\mathcal{M} \models \psi$ *and* $[\![p_i]\!]_{\mathcal{M}}$ *is finite for each* $0 < i \leqslant n$. *Then there is a finite model* $\mathcal{M}_f$ *such that* $\mathcal{M}_f \models \psi$. *Moreover, the satisfiability problem for the class of formulae of the form* ψ *is decidable.*

We prove the lemma for the case $k = 1$, i.e., $\psi = \forall x(p(x) \rightarrow \varphi)$. The proof extends straightforwardly to larger values of k. Suppose $[\![p]\!]_{\mathcal{M}} = \{e_1, \ldots, e_n\}$. Then let $c_1, \ldots, c_n$ be constant symbols not appearing in ψ. The c_i will be used as names for the n elements of $[\![p]\!]_{\mathcal{M}}$.

Let φ' be the result of replacing each atom of the form $p(t)$ (where t is any term) in φ with the conjunction of equalities $(t = c_1 \vee \cdots \vee t = c_n)$. Let

$$\psi' = \bigwedge_{0<i\leqslant n} \varphi'[c_i/x].$$

Then ψ' is satisfied by an interpretation $\mathcal{M}'$ which agrees with $\mathcal{M}$ everywhere except possibly on the c_i, where $[\![c_i]\!]_{\mathcal{M}'} = e_i$ for $0 < i \leqslant n$. Moreover, since ψ' is a satisfiable quantifier-free formula, it has a finite model $\mathcal{M}'_f$ (see section 10.6 of Gallier 1986). We use $\mathcal{M}'_f$ to construct a finite model $\mathcal{M}_f$ for ψ as follows. Let $\mathcal{M}_f$ agree with $\mathcal{M}'_f$ everywhere except possibly on p, where

$$[\![p]\!]_{\mathcal{M}_f} = \{[\![c_i]\!]_{\mathcal{M}'_f}, \ldots, [\![c_n]\!]_{\mathcal{M}'_f}\}.$$

Clearly $\mathcal{M}_f \models \varphi[c_i/x]$ for each $0 < i \leqslant n$, and hence $\mathcal{M}_f \models \psi$. The finite model property immediately implies the decidability of the class Ψ (a decision

procedure can be obtained by enumerating both the set of finite models and the set of inconsistent formulae). ■

The theorem follows by combining Lemmas 1 and 2.

THEOREM 3. *Let F be a satisfiable feature-structure formula set as specified above. Then F has a finite model. Moreover, the satisfiability problem for the class of such feature-structure formulae sets is decidable.*

By Lemma 1, in any satisfying model **node**, **category**, **terminal** and **string-position** all denote finite sets. Now consider the formula ϕ, obtained by conjoining all of the formulae in F. Clearly ϕ has the same models as F. Further, ϕ can be transformed into a formula ϕ' that meets the conditions of Lemma 2 by moving all universal quantifiers outward, skolemizing the existential quantifiers, and performing minor rearrangements of the propositional connectives. It is easy to check that the finite model for ϕ' exhibited by Lemma 2 is also a model of F. ■

Cognitive and Linguistic Sciences
Box 1978
Brown University
Providence RI 02912
U.S.A.