LFG Generation by Grammar Specialization

Jürgen Wedekind*
University of Copenhagen

Ronald M. Kaplan**
Nuance Communications, Inc.

This article describes an approach to Lexical-Functional Grammar (LFG) generation that is based on the fact that the set of strings that an LFG grammar relates to a particular acyclic f-structure is a context-free language. We present an algorithm that produces for an arbitrary LFG grammar and an arbitrary acyclic input f-structure a context-free grammar describing exactly the set of strings that the given LFG grammar associates with that f-structure. The individual sentences are then available through a standard context-free generator operating on that grammar. The context-free grammar is constructed by specializing the context-free backbone of the LFG grammar for the given f-structure and serves as a compact representation of all generation results that the LFG grammar assigns to the input. This approach extends to other grammatical formalisms with explicit context-free backbones, such as PATR, and also to formalisms that permit a context-free skeleton to be extracted from richer specifications. It provides a general mathematical framework for understanding and improving the operation of a family of chart-based generation algorithms.

1. Introduction

Algorithms providing compact representations of alternative syntactic analyses have been the state-of-the-art in parsing for many years. For context-free grammars, for example, the well-known chart parsing algorithms have been used for more than four decades. These assign to a sentence not just one possible analysis but a chart that compactly represents all possible syntactic analyses. Algorithms have also been developed that extend packing to the functional specifications of unification grammars by producing compact representations of feature-structure ambiguities as well. One that is pertinent to (but not restricted to) Lexical-Functional Grammar (LFG) is the contexted constraint satisfaction method developed by Maxwell and Kaplan (1991). These algorithms lead to better average time performance because they carefully manage the ambiguities that are rampant in natural language. They work by dividing the parsing problem into two phases, a recognition or satisfiability phase that creates the compact representation and determines whether there is at least one parse, and an enumeration phase in which the alternative parses are produced one by one. Parsing performance

* Center for Language Technology, University of Copenhagen, Njalsgade 140, 2300 Copenhagen S, Denmark. E-mail: jwedekind@hum.ku.dk.
** Nuance Communications, Inc., 1198 East Arques Avenue, Sunnyvale, CA 94085, USA. E-mail: Ronald.Kaplan@nuance.com.

Submission received: 24 March 2011; revised submission received: 31 October 2011; accepted for publication: 28 December 2011.

is typically identified with the complexity of the first phase (e.g., the cubic bound for context-free parsing), because the collection of all parses can be delivered to a client application merely by presenting the compact representation. A client may be able to select a limited number of particularly desirable parses, perhaps the smallest or the most probable, without doing a full enumeration (Johnson and Riezler 2002; Kaplan et al. 2004).

Lang (1994) gives a clear formal characterization of the first phase of context-free chart parsing.[1] He observes that the recognition problem consists of finding the intersection of the language of the grammar with the input string, and then testing to see whether that intersection is empty. Many language classes are closed under intersection with a regular set, and the result of the intersection of a language $L(G)$ with a regular language α is describable as a specialization G_α of G that assigns to all and only the strings in α effectively the same parse trees as G would assign. Lang argues that a chart for an input string s (a trivial regular language) and a context-free grammar G can be regarded as a specialization G_s of G that derives either the empty language (if s does not belong to $L(G)$) or a language consisting of just that input. In this view a parsing chart/grammar is a representation that makes it possible to enumerate all the derivation trees of the string, guaranteeing that each tree can be produced in a backtrack-free way in time proportional to its size. This guarantee holds even for an infinitely ambiguous string: It would take forever to enumerate all valid derivations, but any particular one can be read out in linear time. The procedure for tree enumeration follows directly from the standard context-free generation algorithm applied to the grammar G_s.

The generation problem for LFG and other description-based grammatical formalisms can also be viewed from this perspective. Several algorithms have been proposed for generation that avoid redundant recomputation by storing intermediate processing results in a chart-like auxiliary data structure (e.g., Shieber 1988; Kay 1996; Shemtov 1997; Neumann 1998; Carroll et al. 1999; Moore 2002; Carroll and Oepen 2005; Cahill and van Genabith 2006; White 2006; de Kok and van Noord 2010). Most of them can be construed as having a first phase that provides a compact representation for alternative results, in this case for the strings that the grammar provides for a given functional or semantic input. The individual generated strings are then produced by an enumeration procedure operating on this compact representation.

In this article we observe that the edges of a generation chart can be interpreted as rules of a specialized context-free grammar, just as in Lang's (1994) characterization of parsing. We present a generation algorithm that specializes the context-free backbone of a given LFG grammar to a grammar that describes exactly the strings that the LFG grammar relates to a given acyclic f-structure. Derivations of the resulting grammar simulate all and only those derivations of the LFG grammar whose derived strings are assigned to that input.[2] Thus the generated string set is a context-free language compactly represented by the specialized grammar, and the individual members of that language can be enumerated, just as for parsing, by using standard context-free generation algorithms.

Our approach can be seen as a generalization and formalization of other chart-based generation algorithms, producing all and only correct outputs for larger combinations of grammars and inputs. It extends to unification grammars with explicit context-free backbones, such as PATR (Shieber et al. 1983), and also to formalisms that permit a context-free skeleton to be extracted from richer specifications. But it does not extend

1 Dymetman (1997) extends this characterization to unification grammars.

2 The word "derivation" here and in the following is used only to characterize the notion of well-formedness in LFG and is not meant to undermine the contrast between LFG and conventional transformational approaches to syntax.

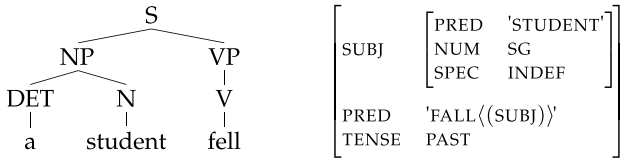


Figure 1
The components of an LFG representation: string, constituent structure, functional structure.

to cyclic input structures because, as we will show by example, an LFG grammar might relate to a cyclic structure a set of strings that is not context-free. Because acyclic structures are normally assumed to be the only f-structures that are motivated for linguistic analysis (Kaplan and Bresnan 1982), this restriction does not seem to limit the applicability of our algorithm for natural language generation.

We begin with some background so that we can make the problem and its solution more explicit. Along with many other description-based grammar formalisms, an LFG grammar G assigns to every string s in its language at least one f-structure. This situation can be characterized in terms of a derivation relation Δ_G, defined as follows:

(1) $\Delta_G(s, F)$ iff G assigns to the string s the f-structure F

In the LFG approach a sentence s and its f-structure F are not directly related. Their relation is mediated by a valid c-structure for s (Kaplan 1995). The arrangement of the three components of an LFG representation is illustrated in Figure 1. This representation is derivable by a grammar that includes the annotated (nonterminal) rules in (2a–c) and lexical expansions in (2d–f). Annotated lexical c-structure rules are just notational variants of traditional LFG lexical entries.

(2) a. S → NP VP
(↑ SUBJ) = ↓ ↑ = ↓

b. NP → DET N
↑ = ↓ ↑ = ↓

c. VP → V
↑ = ↓

d. DET → a
(↑ SPEC) = INDEF
(↑ NUM) = SG

e. N → student
(↑ PRED) = 'STUDENT'
(↑ NUM) = SG

f. V → fell
(↑ PRED) = 'FALL⟨(SUBJ)⟩'
(↑ TENSE) = PAST

In accordance with the basic architecture of LFG, an LFG grammar provides a set of licensing conditions that determine grammatical representations by descriptive, model-based rather than procedural methods. The well-formedness of the representation in Figure 1 with respect to the grammar in (2) is thus characterized as follows.

The c-structure is valid or well-formed because we can assign to each nonterminal node a grammar rule that licenses or justifies the local mother–daughters configuration constituted by the node and its immediate daughters. If we assume that the c-structure of Figure 1 consists of the nodes $root, n_1, .., n_8$ and these nodes are related and labeled as depicted in Figure 2, then the rule-mapping ρ that justifies the c-structure is given by (3).

(3) $\rho_{root} = (2a)$, $\rho_{n_1} = (2b)$, $\rho_{n_2} = (2c)$, $\rho_{n_3} = (2d)$, $\rho_{n_4} = (2e)$, $\rho_{n_5} = (2f)$

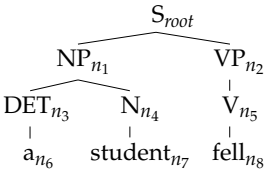


Figure 2
The c-structure of Figure 1 with explicitly specified nodes.

A description of the f-structure (called the **f-description**) for this tree and rule-mapping is constructed by instantiating the annotations of all justifying rules in the following way. For each rule justifying a local mother–daughters configuration, all occurrences of the ↑ symbol (called a **metavariable**) in the functional annotations of the daughters are replaced by the mother node, and for each of the daughter categories, all occurrences of the ↓ metavariable in its annotations are replaced by the corresponding daughter node.[3] Thus, the ↓ of the annotations on a daughter category of a rule and the ↑ of the annotations of the rule that further expands that category are always instantiated with the same node. The complete f-description is the union of the instantiated descriptions of all the justifying rules. The f-description obtained from the c-structure in Figure 1 and the rules of the justifying mapping in (3) is given in (4).

$$(4)\quad \left\{ \begin{array}{c} (root\ \text{SUBJ}) = n_1, root = n_2, \\ n_1 = n_3, n_1 = n_4, \\ n_2 = n_5, \\ (n_3\ \text{SPEC}) = \text{INDEF}, (n_3\ \text{NUM}) = \text{SG}, \\ (n_4\ \text{PRED}) = \text{'STUDENT'}, (n_4\ \text{NUM}) = \text{SG}, \\ (n_5\ \text{PRED}) = \text{'FALL}\langle(\text{SUBJ})\rangle\text{'}, (n_5\ \text{TENSE}) = \text{PAST} \end{array} \right\}$$

The f-structure in Figure 1 is associated with the given c-structure because it satisfies the f-description in (4), and furthermore, it is the unique minimal solution for this description. In general, LFG requires the f-description of a grammatical sentence to be satisfiable and thus to have at least one model. Each such satisfying model consists of a universe and an interpretation function that assigns unary (partial) functions to the attributes (SUBJ, NUM, SPEC, etc.) and elements of the universe to the atomic feature values (SG, INDEF, etc.), as well as to the nodes in the description. Among the models satisfying a given f-description there is an (up to isomorphism unique) minimal model, one that is not properly subsumed by other satisfying models.[4] This minimal model represents the f-description's minimal solution, the one from which the f-structure for the sentence is obtained. Conventional attribute–value matrices where the nodes (or node numbers) are attached to the left brackets are LFG-typical representations of

3 Our instantiation procedure uses the nodes themselves instead of the related f-structure variables of Kaplan and Bresnan (1982) or the more complex ϕ-terms of Kaplan (1995). This is mathematically equivalent to the other representations but simplifies our illustrations.

4 For some sentences and some grammars (e.g., those involving functional uncertainty) there are f-descriptions with several non-isomorphic minimal models. As we discuss in a subsequent article (Wedekind and Kaplan forthcoming), these also lie within the bounds of our context-free construction.

exactly such minimal models. The attribute–value matrix representation of the minimal model of the f-description (4) is given in (5).

(5)
$$\begin{bmatrix} root & n_1 \begin{bmatrix} \text{PRED} & \text{'STUDENT'} \end{bmatrix} \\ n_2 & \text{SUBJ} \quad n_3 \begin{bmatrix} \text{NUM} & \text{SG} \end{bmatrix} \\ n_5 & n_4 \begin{bmatrix} \text{SPEC} & \text{INDEF} \end{bmatrix} \\ & \text{PRED} \quad \text{'FALL}\langle(\text{SUBJ})\rangle\text{'} \\ & \text{TENSE} \quad \text{PAST} \end{bmatrix}$$

The solution in (5) is converted to the f-structure representation in Figure 1 by removing the node labels that record the relation of the f-structure to the c-structure. From a formal point of view, an f-structure is obtained from the minimal model of an f-description by restricting its interpretation function to the attributes and atomic feature values of the grammar, thus disregarding the nodes and their interpretation.[5]

We now turn to the generation problem. A generator for G provides for any given f-structure F the set of strings that are related to it by the grammar:

(6) $Gen_G(F) = \{s \mid \Delta_G(s, F)\}$

The algorithm presented in this article accomplishes the generation task by producing a context-free grammar for $Gen_G(F)$, for a given LFG grammar G and any acyclic input f-structure F.

The abstract generator characterization in (6) is of course dual to the one for a parser for G, since a parser produces for any given terminal string s the set of f-structures that are assigned to it by G:

(7) $Par_G(s) = \{F \mid \Delta_G(s, F)\}$

For parsing, the Kaplan and Bresnan (1982) proscription of nonbranching dominance chains guarantees that $Par_G(s)$ will contain only a finite number of f-structures, but this condition does not ensure the finiteness of the set of strings $Gen_G(F)$ that are related to an f-structure F. This is illustrated by the simple grammar in (8):

(8) $\text{S} \rightarrow \underset{\uparrow = \downarrow}{\text{a S}}\ \text{b}$ $\qquad$ $\text{S} \rightarrow \underset{(\uparrow \text{H}') = \text{V}'}{\text{c}}$

$\text{S} \rightarrow \underset{(\uparrow \text{H}) = \text{V}}{\text{a}}\ \text{b}$

This generates for the input (9)

(9) [H V]

the infinite context-free language $\{\text{a}^n\ \text{b}^n \mid 1 \leq n\}$.

From a cognitive point of view it seems unrealistic that the number of sentences that a natural language grammar relates to an f-structure is infinite. As a minimum, there should be some relationship that bounds the size of the c-structure of a sentence

5 Note that the interpretation of the node constants that we restrict out of the minimal models can be seen to represent the structural correspondence function that maps individual nodes of the c-structure tree into elements of the f-structure (cf. Kaplan 1995).

by the size of the f-structures associated with it. Such a structural relationship would then force the related sentences to form a finite set. Studies to determine intuitively plausible restrictions are rather scarce, however, and proposals for such restrictions are not yet generally accepted. It is thus still an open question whether grammars of actual natural languages satisfy the particular resource-boundedness restrictions on which termination of some existing chart-based generators depends.

Even if only finite sets of sentences are related to the f-structures, these sets might still be very large. Experiments with a broad-coverage German LFG grammar (Dipper 2003; Rohrer and Forst 2006) have shown that, because of the scrambling that German allows, a given f-structure might be related to a huge set of long sentences.[6] This has consequences at least for those approaches that assume the output of generation to be a word lattice (Langkilde and Knight 1998) or a finite-state machine representing the (finite) set of all generated sentences. A lattice can represent a large collection of strings compactly only if they are characterized by independent sets of alternative substrings. Scrambling languages, however, have alternative substrings that reappear in different positions with complex cooccurrence dependencies and therefore cannot be shared in a lattice representation (see Langkilde [2000] for discussion). Our context-free grammars (and also Langkilde's [2000] and Knight and Langkilde's [2000] forest representations) offer a much more compact encoding under these circumstances, and their structure and formal properties are as well understood as lattices and finite-state machines.

Our approach might also be more appropriate than existing chart-based approaches for optimality-theoretic generation (Kuhn 2001, 2002, 2003). An optimality-theoretic LFG system consists of two components: a universal LFG grammar and a language-specifically ordered set of violable constraints (Bresnan 2000). The universal LFG grammar is used to produce the candidate space of possible analyses (consisting of the c-structure/f-structure pairs that are derivable by the grammar). The optimal and thus grammatical analyses are those candidates that violate the fewest constraints. A technical problem comes from the fact that the universal grammar by design may assign an infinite number of c-structures and string realizations to a given f-structure, and the optimal outputs can be identified only by evaluating all of these against the collection of constraints. Our context-free characterization provides a finite evaluation procedure even for an infinite candidate space. By virtue of the pumping lemma for context-free languages (Bar-Hillel, Perles, and Shamir 1961; see also Hopcroft and Ullman 1979) we can enumerate the c-structure trees assigned to an input f-structure one by one in order of increasing depth. Because the number of constraint violations increases beyond a certain number of recursive category expansions, the optimal results from the infinite space can be chosen after examining only a finite number of relatively small structures (see Kuhn [2003] for details).

Similar to Lang's approach to parsing (see also Billot and Lang 1989), we provide a general framework encompassing all forms of chart generation in a single formalism. This is because existing chart-based generators can be understood as concrete but somehow restricted algorithm/datastructure implementations of our context-free grammar construction. These restrictions may lead them to produce incorrect outputs in some situations. Because we show the correctness of the output grammar for unrestricted

6 The German grammar was developed as part of the Parallel Grammar project (ParGram), a research and development consortium that has produced large-scale LFG grammars for several languages (Butt et al. 1996, 2002). These grammars are developed on the XLE system, a high-performance platform for LFG parsing and generation. More information on the ParGram project and XLE can be found at: `http://pargram.b.uib.no/`.

LFG grammars, our framework allows us to examine, compare, and improve on existing chart-based generation techniques.

The organization of this article is as follows. In the next section we define the fundamental formal objects of LFG theory and the relevant relationships among them. Section 3 is the technical core of the article. There we present and prove the correctness of the context-free grammar-construction algorithm for LFG grammars with arbitrary equational constraints and acyclic input f-structures. The grammar construction abstracts away from specific details of data structure and computational strategy not essential to the mathematical argument. Performance and computational strategy are then briefly considered in Section 4, and Section 5 compares our approach to other generation algorithms. In Section 6 we identify a fundamental limitation of our approach, demonstrating that the context-free property does not hold for elementary equational constraints if the input f-structure contains cycles. On the other hand, if the input is acyclic, the basic context-free construction can be extended beyond simple equations to the additional descriptive devices proposed by Kaplan and Bresnan (1982) and still in common use. This is shown in Section 7. The last section highlights some additional consequences of this approach.

The present article elaborates on ideas that we first presented in Kaplan and Wedekind (2000). In that paper we outlined a context-free grammar construction for a subclass of LFG grammars with restricted functional annotations and single-rooted input structures. Here we consider a more general class of grammars and inputs that requires a more rigorous mathematical analysis.

2. Preliminaries

We start with a formal characterization of LFG grammars with equational statements. Let V^* denote the set of all finite strings over V. An LFG grammar G over a set Σ of attribute and value symbols is defined as follows:

Definition 1
An **LFG grammar** G (over attribute–value set Σ) is a 4-tuple (N, T, S, R) where N is a finite set of nonterminal categories, T is a finite set of terminal symbols, $\mathrm{S} \in N$ is the root category, and R is a finite set of annotated productions of the form

$$\begin{array}{c} A \rightarrow X_1 \; .. \; X_m \\ \quad\; D_1 \quad\; D_m \end{array}$$

with $A \in N$ and $X_1..X_m \in (N \cup T)^*$. (Note that R might contain ϵ-productions, although these do not appear in most current linguistic descriptions.) Each annotated description D_j $(j = 1, .., m)$ is a (possibly empty) finite set of equalities between expressions of the form $(\uparrow\ \sigma)$, $(\downarrow\ \sigma)$, or v where v is a value of Σ and σ is a possibly empty sequence of attributes of Σ. When σ is empty, $(\uparrow\ \sigma)$, $(\downarrow\ \sigma)$ are equivalent to $\uparrow$ and $\downarrow$, respectively.[7]

7 Note that this definition permits equations containing terms of the form $(\downarrow\ \sigma)$, with σ nonempty, and that it does not require $\uparrow$ and $\downarrow$ to occur in the annotation of each category. It thus allows for grammars that assign to sentences multiply rooted f-structures or f-structures consisting of totally unconnected parts. We take the "f-structure of a sentence" to be the collection of all elements that correspond to c-structure nodes, even those that are not accessible from the root node's f-structure.

We next define how instantiated descriptions are obtained from the rules by substituting for the ↑ and ↓ metavariables elements drawn from a collection of terms. C-structure nodes are included among the terms, but later on we also make use of additional elements. We define a function *Inst* that assigns to each m-ary rule r, term t, and term sequence $t_1..t_m$ the instantiated description that is obtained from the annotations of r and the terms by substituting t for ↑ and t_j for ↓ in the annotations of all $j = 1, .., m$ daughters. In the following definition we use the (more compact) linear rule notation $A \rightarrow (X_1, D_1)..(X_m, D_m)$ that we prefer in more formal specifications.

Definition 2
Let r be an m-ary LFG rule $A \rightarrow (X_1, D_1)..(X_m, D_m)$ $(m \geq 0)$ and $\tau = (t, t_1..t_m)$ be a pair of a term and a sequence of terms of length m. Then the **instantiated description** that results from r and τ is given by

$$Inst(r, \tau) = \bigcup_{j=1}^{m} Inst(D_j, t, t_j)$$

where $Inst(D_j, t, t_j)$ is the instantiated description produced by substituting t for all occurrences of ↑ in D_j and substituting t_j for all occurrences of ↓ in D_j.

The derivation relation for LFG grammars (Δ_G) is defined as already described informally in the previous section. This is based on context-free derivation trees. Let us assume that *root* is the root node of any c-structure c, and that *dts* is a function that assigns to each nonterminal node n of c the sequence of its immediate daughters ($dts(n)$). Context-free derivations are then defined as follows:

Definition 3
A labeled tree c and a rule-mapping ρ from the nonterminal nodes of c into the rules of context-free grammar G is a **context-free derivation** of string s from nonterminal B in G iff

(i) the label (category) of *root* is B,
(ii) the yield is s,
(iii) for each nonterminal node n with label A and $dts(n) = n_1..n_m$ with labels $X_1, .., X_m$, respectively, $\rho_n = A \rightarrow X_1..X_m$.

When we informally described LFG derivations, we pointed out that we obtain the f-structure from the (up to isomorphism) unique minimal model of the f-description by restricting it to the attribute–value set Σ. This is formalized in the following definition by requiring the f-structure to be isomorphic ($\cong$) to $M|\Sigma$, the restriction to Σ of a minimal model M of the derived f-description. The effect of the isomorphism is to abstract away from the particular properties of different f-structure models that have no linguistic significance. Moreover, because we operate on an arbitrary member of the class of isomorphic structures without regard to any of its accidental or nonsignificant properties, we know that our analysis applies to all members of the class.

Definition 4
A labeled tree c and a mapping ρ from the nonterminal nodes of c into R is an **LFG derivation** of string s with functional description FD and f-structure F in LFG grammar G iff

(i) the label of *root* is S,
(ii) the yield is s,
(iii) for each nonterminal node n with label A and $dts(n) = n_1..n_m$ with labels $X_1, .., X_m$, respectively, $\rho_n = A \rightarrow (X_1, D_1)..(X_m, D_m)$,
(iv) $FD = \bigcup_{n \in Dom(\rho)} Inst(\rho_n, (n, dts(n)))$,
(v) FD is satisfiable,
(vi) $FD \not\vdash a = v$ if v is an atomic feature value and a is any other constant (atomic feature value or node) occurring in FD,
(vii) $FD \not\vdash (v\ \sigma) = (v\ \sigma)$ if v is an atomic feature value and σ is a nonempty sequence of attributes,
(viii) $M|\Sigma \cong F$ where M is a minimal model of FD.

Conditions (vi) and (vii) are syntactic versions of the constant/constant and constant/complex clash conditions that together capture LFG's functional uniqueness condition (the denotations of an atomic feature value and any other distinct atomic feature value or node constant have to be distinct (vi); atomic feature values have no attributes (vii)).[8] A model of an f-description, like the restricted one in (viii), is a pair $(\mathcal{U}, I)$ consisting of a universe $\mathcal{U}$ and an interpretation function I. The interpretation function assigns to each constant occurring in the f-description an element of $\mathcal{U}$ and to each attribute a unary partial function on $\mathcal{U}$.

Note that we create the f-description by instantiating the $\uparrow$'s and $\downarrow$'s by the nodes of a given c-structure. Thus, we conceive of these terms as constants and will refer to them on the f-description level sometimes as node constants rather than nodes. Because the instantiating nodes are uniquely determined if we have a mapping ρ licensing a given c-structure, in the following we abbreviate $Inst(\rho_n, (n, dts(n)))$ by $Inst(\rho_n)$.

In general, two descriptions D and D' are said to be **equivalent** ($D \equiv D'$) iff the restrictions of their minimal models to Σ are isomorphic.

Definition 5
Let D and D' be two descriptions with minimal models M and M'. Then $D \equiv D'$ iff $M|\Sigma \cong M'|\Sigma$.

From Definition 4 we obtain the derivability relation Δ as follows.

Definition 6
A terminal string s is **derivable** with f-structure F in G ($\Delta_G(s, F)$) iff there is a derivation of s with F (with some f-description FD) in G.

8 Usually, conditions (vi) and (vii) are taken to be additional nonlogical axiom schemata of some traditional equational logic expressive enough to axiomatize LFG's underlying feature logic. Because we are not primarily interested in completely axiomatizing LFG's formal devices within some appropriate meta-theory, we enforce the special properties of LFG's atomic feature values by definition and assume that standard first-order logic with equality is used to determine satisfiability.

In this context we repeat the definition of the set of strings $Gen_G(F)$ that an LFG grammar G relates to a given f-structure F:

Definition 7
For any LFG grammar G and any f-structure F

$$Gen_G(F) = \{s \in T^* \mid \Delta_G(s, F)\}.$$

In the next section we establish the basic result of this article: We present an algorithm to construct for an arbitrary LFG grammar G and any acyclic f-structure F a particular context-free grammar that provides a formal representation for the language $Gen_G(F)$.

3. Constructing the Specialized Grammar for $Gen_G(F)$

In the process of generation, the c-structures and the f-descriptions for an input f-structure F are the unknowns that must be discovered to confirm that a given string belongs to the set $Gen_G(F)$. The set of valid c-structures that G provides for F is clearly a subset of the trees that are generated by the context-free backbone of G. But this subset might be infinite, as we have already seen with the input (9) and the grammar in (8), because there is in general no fixed finite upper bound on the length of the strings related to F or the size of their c-structures. Whether or not a given tree is a valid c-structure for F then depends on the properties of the f-description that arises by instantiating with the proper node constants the annotations on the individual rules that license the derivation of that tree. The valid c-structures are just those trees for which F is the f-structure of the resulting f-description.

Because of the possibly unbounded size of the c-structures, there is also no fixed upper bound on the number of node constants that may occur in an f-description for F. However, because the number of f-structure elements to which the node constants actually refer is bounded by the size of F, it must be possible to obtain for any derived f-description FD an equivalent description whose constants are drawn from a fixed finite set. For instance, if we introduce a distinct canonical constant for each element of F, we can create an equivalent description by substituting for each node constant in FD the canonical constant associated with the functional element corresponding to that node. This substitution typically reduces the number of distinct terms needed for instantiation, and its usual effect is to replace several different node constants with a single canonical term. But these replacements will provide an equivalent description because we substitute a given term for two node constants if and only if it logically follows from FD that those two nodes map to the same element of F. Thus, if FD discriminates between two elements of F, so will the description that results from such a reducing substitution.

Our context-free grammar construction crucially depends on the ability to find for every f-description of F (from every possible c-structure) an equivalent description that involves only a finite number of distinct instantiation terms. This is what enables us to simulate all the conditions for correct LFG generation with a finite set of context-free category labels and a finite set of context-free productions, and thus to rely on the finite control of the rule-by-rule category matching process of context-free generation to produce the strings in $Gen_G(F)$.

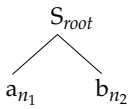
The set of terms that correspond directly to the elements of F is large enough to enforce all functional discriminations for an f-description associated with a complete c-structure, as we have suggested. But unfortunately that set may not be large enough to keep track of all necessary distinctions as an f-description is created in an incremental context-free derivation process. For some f-structures and some grammars it may not follow from the description associated with one portion of a derivation tree that two nodes map to the same functional element, even though that identity does follow when equations in the f-description for the entire tree are taken into account.

Suppose that a three-daughter LFG start rule provides the instantiated annotations $(root\ \text{F}) = n_1$, $(root\ \text{G}) = n_2$, and $root = n_3$. Based only on this information we cannot tell whether n_1 and n_2 can map to the same element of the f-structure and therefore whether it is correct to substitute the same canonical constant for both of them. It depends on whether the larger description that incorporates the expansion of the third daughter implies the identity of the n_1 and n_2 structures. The same-constant substitution would preserve equivalence only if the larger description implies that $(n_3\ \text{F}) = (n_3\ \text{G})$. We must have two distinct constants available until that implication is deduced in the course of the derivation, even if the input f-structure does not contain separate elements for those constants to correspond to.

Thus the set of constants needed to correctly reduce an arbitrary description as a derivation proceeds incrementally may be larger than the number of elements in the input f-structure and larger than what is required for an equivalent description for a complete derivation. However, for each acyclic F we show that there is always a finite set of canonical terms that can maintain all necessary functional discriminations as a derivation unfolds. We use this set to construct a reducing substitution that permits generation to be carried out under finite control. In contrast, we observe in Section 6 that the partial descriptions of cyclic structures cannot safely be reduced without an unbounded number of canonical terms.

For the derivations that the simple LFG grammar in (8) provides for the input [H V], we can accomplish the reduction of the f-description space with only two terms, the canonical constant *root* and a separate canonical constant $\perp$ that serves as a value for all nodes that do not occur in an f-description. Let us start with the shortest derivation for the given input. This consists of the c-structure in (10), which is licensed by the rule-mapping $\rho_{root} = {}^{\text{S}\rightarrow}_{(\uparrow\ \text{H}) = \text{V}}{}^{\text{b}}$.

(10)



If we pair the licensing rule with its instantiating nodes $(root, n_1\ n_2)$, we arrive at the instantiated rule $\left({}^{\text{S}\rightarrow}_{(\uparrow\ \text{H}) = \text{V}}{}^{\text{b}}, (root, n_1\ n_2)\right)$. The reduction can be accomplished by applying to the instantiating nodes a substitution that replaces *root* by *root* and both n_1 and n_2 by $\perp$. This produces the instantiation $\left({}^{\text{S}\rightarrow}_{(\uparrow\ \text{H}) = \text{V}}{}^{\text{b}}, (root, \perp\ \perp)\right)$ from which we obtain the description $\{(root\ \text{H}) = \text{V}\}$. This is identical to the description that arises from the original node-instantiated rule, because the $\downarrow$ does not occur in the annotations. The reduction for all other derivations of [H V] is illustrated in Figure 3. The substitutions of the node constants of the schematically represented derivations by canonical terms are indicated by assigning the canonical term values to the nodes. If we consider the resulting instantiation of the applied rules at the bottom of column (b), we observe that

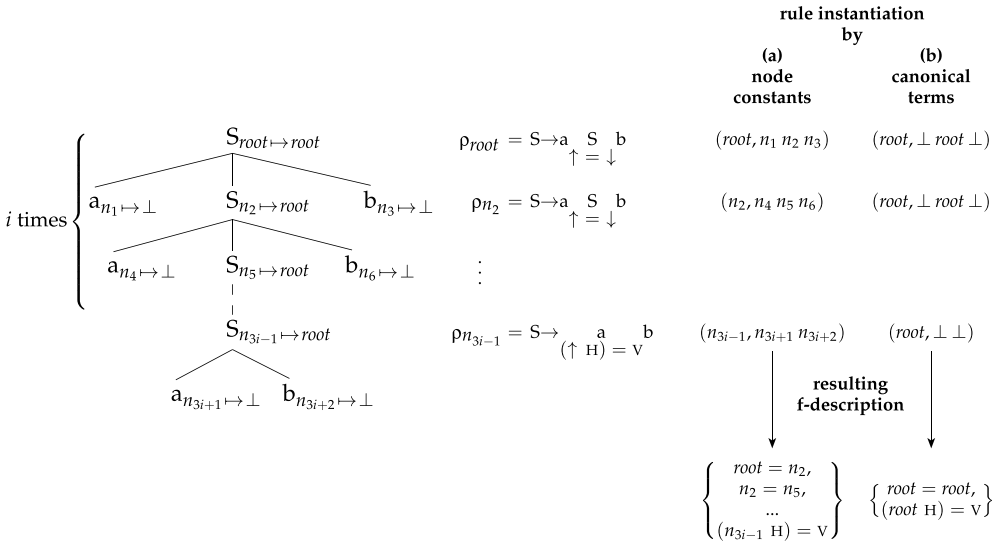


Figure 3
Schematic representation of the derivations of length > 1 with f-structure [H V] admitted by the grammar in (8). The right-hand side shows the instantiated descriptions produced by appropriately instantiating metavariables by node constants in column (a) and by particular canonical canonical terms in column (b).

the f-description of each derivation of the input in G reduces to the description (11a) and that this is the description that results for any derivation with the set of instantiated rules depicted in (11b).

(11) a. $\left\{ \begin{array}{l} root = root, \\ (root\ H) = V \end{array} \right\}$

b. $\left\{ \begin{array}{l} \left(S \rightarrow a\ \underset{\uparrow = \downarrow}{S}\ b, (root, \perp\ root\ \perp) \right), \\ \left(S \rightarrow \underset{(\uparrow H) = V}{a}\ b, (root, \perp\ \perp) \right) \end{array} \right\}$

The reducibility of the f-description space for F provides the key insight for our context-free grammar construction. The construction is accomplished in three steps. In the first step we identify (as illustrated earlier) a finite set of canonical terms that can serve in reducing the f-description space that G provides for F.

In the second step we use these terms to construct a set of instantiated rules of G. These instantiations are "appropriate" in the sense (to be made precise later) that they maintain all necessary distinctions. They are formed by associating with the metavariables canonical terms that can legitimately be used to reduce the corresponding nodes of a local tree of a potential derivation of F. For the grammar (8) and the terms *root* and $\perp$, for example, there are only three appropriately instantiated rules, the two rules contained in (11b) and $\left(S \rightarrow \underset{(\uparrow H') = V'}{c}, (root, \perp) \right)$. We then determine all collections of appropriately instantiated rules that together provide descriptions of F without mistakenly collapsing a functional discrimination. For our particular example there are just two collections of instantiated rules that provide a description of [H V], namely, $\left\{ \left(S \rightarrow \underset{(\uparrow H) = V}{a}\ b, (root, \perp\ \perp) \right) \right\}$ and the set in (11b). These collections are drawn

from the power set of the appropriately instantiated rules, so there is only a finite number of them and each contains a finite number of instantiated rules. This ensures that we can determine the f-description space that G provides for F without knowing the details of the derivations for F and their reductions.

In the third and final step we create the context-free grammar that simulates exactly those derivations in G whose strings are assigned the f-structure F. The categories of this new grammar consist of refinements of the categories of the context-free backbone of G together with a distinct root category S_F. The original categories are augmented with two additional components, a canonical instantiation term as used in the first step, and a subset of one of the instantiated-rule collections determined in the second step. The term component is used to encode the reducing substitution for the f-description of a simulated derivation, and the rule component is used to record the reduced instantiations of the licensing LFG rules whose application must still be simulated in order to complete that derivation. The productions of the new grammar are created from the rules contained in the instantiated-rule collections by replacing the original categories by a certain number of their refinements, and then adding a particular set of start rules. The start rules expand the root category of the new grammar to the original start symbol augmented by *root* and one of the instantiated-rule collections determined in the second step.

The context-free grammar thus constructed has a much larger set of categories and many more rules than G. It is organized so that the normal matching of categories in a context-free derivation globally ensures that the refined rules simulate all derivations of F in G whose f-description is reducible to a description provided by one of the instantiated-rule collections determined in the second step. Because we have already indicated that every f-description of F must be reducible to a description provided by one of these instantiated-rule collections, the constructed grammar simulates exactly the set of derivations that G provides for F. The strings of $Gen_G(F)$ are obtained by removing the additional components from the categories of the terminal strings. With r_1 abbreviating $\left({}^{\mathrm{S}\to \mathrm{a}}_{\uparrow = \downarrow}{}^{\mathrm{S}}_{\mathrm{b}}, (root, \bot\ root\ \bot)\right)$ and r_2 abbreviating $\left({}^{\mathrm{S}\to}_{(\uparrow \mathrm{H}) = \mathrm{V}}{}^{\mathrm{a}}_{\mathrm{b}}, (root, \bot\ \bot)\right)$ our construction produces for the LFG grammar in (8) and the input [H V] a context-free grammar that contains the rules in (12).

(12) a. $S_F \rightarrow$ S:*root*:$\{r_1, r_2\}$
b. $S_F \rightarrow$ S:*root*:$\{r_2\}$
c. S:*root*:$\{r_1, r_2\} \rightarrow$ a:$\bot$:$\emptyset$ S:*root*:$\{r_1, r_2\}$ b:$\bot$:$\emptyset$
d. S:*root*:$\{r_1, r_2\} \rightarrow$ a:$\bot$:$\emptyset$ S:*root*:$\{r_2\}$ b:$\bot$:$\emptyset$
e. S:*root*:$\{r_2\} \rightarrow$ a:$\bot$:$\emptyset$ b:$\bot$:$\emptyset$

These derive the set of terminal strings $\{\mathrm{a}{:}\bot{:}\emptyset^n\ \mathrm{b}{:}\bot{:}\emptyset^n \mid 1 \leq n\}$. By removing the terminal refinements we obtain $\{\mathrm{a}^n\ \mathrm{b}^n \mid 1 \leq n\}$ and thus exactly the set of strings that the grammar in (8) relates to [H V]. The rules (12b,e) simulate the derivation with the c-structure in (10) and the rules (12a,c–e) simulate the derivations of length > 1. Rule (12c) simulates recursions of the S rule of (8) and an application of rule (12d) terminates a recursion because it consumes r_1. An application of rule (12e) consumes r_2 and terminates the derivations.

In the remainder of this section we first identify the finite set of canonical terms that can be used to reduce the f-description space for a given f-structure F derivable with an LFG grammar G. We then investigate in Section 3.2 the problem of reducing the f-description space for F and G. In Section 3.3 we give a precise recipe for constructing the context-free grammar for F and G, and in Section 3.4 we illustrate this with a few examples.

3.1 Identifying the Reducing Terms

Our reduction of the f-description space makes use of the fact that we can eliminate certain node constants from an f-description *FD* without risk of producing a description not equivalent to the original. This is because some node constants can be defined in terms of others. We proceed rule-wise top–down based on the following definability relation.

Definition 8
Let r be an m-ary LFG rule, t be a term, and $a_1..a_m$ be a sequence of constants of length m, each of them not occurring in t. A constant a_j is **m(other)-definable** in $Inst(r, (t, a_1..a_m))$ iff there is a (possibly empty) σ such that $Inst(r, (t, a_1..a_m)) \vdash a_j = (t\ \sigma)$.

If the constant a_j that instantiates the ↓ for a particular daughter is m-definable in terms of $(t\ \sigma)$ in $Inst(r, (t, a_1..a_m))$, then all functional discriminations will be preserved if a_j is eliminated in favor of the term $(t\ \sigma)$ from any description containing this instantiated description of r.

To illustrate the elimination process, let us assume that our grammar includes among its rules the ones in (13).

(13) a. S → NP VP
(↑ SUBJ) = ↓ ↑ = ↓

b. VP → V ADVP
↑ = ↓ ↑ = ↓

c. ADVP → ADV ADVP
(↑ ADJ) = (↓ ELE) ↑ = ↓

d. ADVP → ADV
(↑ ADJ) = (↓ ELE)

e. NP → John
(↑ PRED) = 'JOHN'

f. V → fell
(↑ PRED) = 'FALL⟨(SUBJ)⟩'
(↑ TENSE) = PAST

g. ADV → today
(↑ PRED) = 'TODAY'

h. ADV → quickly
(↑ PRED) = 'QUICKLY'

In this grammar fragment we use equational annotations in the adverbial phrase rules (13c) and (13d) instead of the more traditional set-membership statements (Kaplan and Bresnan 1982). This is another way of allowing for multi-valued attributes that was the original motivation for introducing set representations into LFG theory. We use the equational treatment here to illustrate the fact that undefinable and hence ineliminable constants can arise even when equality is the only formal device in an f-description. The adverbial rules also illustrate that undefinable constants can figure in the description of multiply rooted f-structures. The rules in (13) provide, for example, the derivation depicted in Figure 4. The figure shows the f-structure on the left-hand side and the instantiated descriptions of the licensing rules associated with the nodes of the c-structure on the right-hand side. This more conventional way of depicting derivations (Kaplan and Bresnan 1982) makes it easy to see the mother-definability relation and at the same time permits the licensing rule-mapping to be read from the annotated c-structure. The complete f-description is shown as a set in (14).

(14) $$\left\{\begin{array}{c}(root\ \text{SUBJ}) = n_1, (n_1\ \text{PRED}) = \text{'JOHN'},\\ root = n_2, n_2 = n_4, n_2 = n_5, n_5 = n_8,\\ (n_4\ \text{PRED}) = \text{'FALL}\langle(\text{SUBJ})\rangle\text{'}, (n_4\ \text{TENSE}) = \text{PAST},\\ (n_5\ \text{ADJ}) = (n_7\ \text{ELE}), (n_7\ \text{PRED}) = \text{'TODAY'},\\ (n_8\ \text{ADJ}) = (n_{10}\ \text{ELE}), (n_{10}\ \text{PRED}) = \text{'QUICKLY'}\end{array}\right\}$$

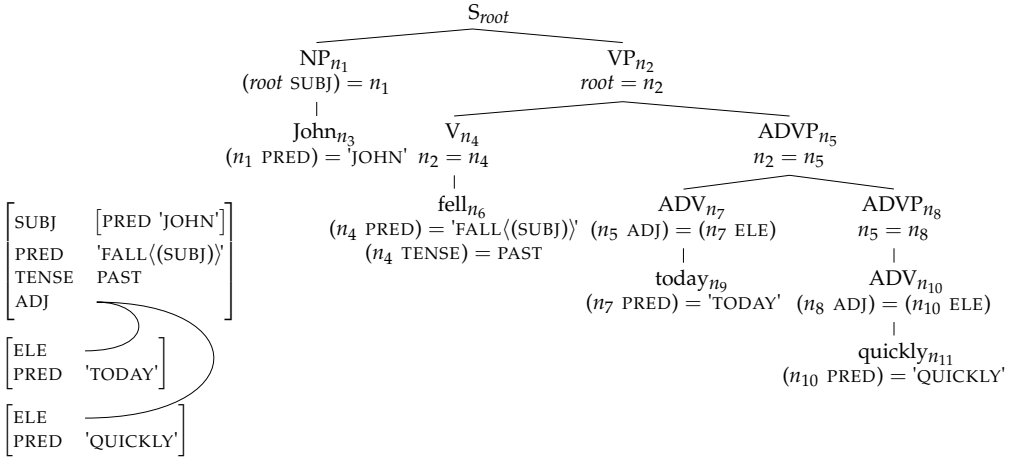


Figure 4
The derivation of *John fell today quickly* with the rules in (13).

In this derivation, the node constants n_1, n_2, n_4, n_5, and n_8 are m-definable whereas *root*, the adverbial nodes n_7 and n_{10}, and the terminal nodes are not. For all m-definable constants, we can construct definitions rule-wise top–down in the following way. We begin with the start rule and derive from its instantiated description $\{(root\ \text{SUBJ}) = n_1, root = n_2\}$ the definitions $n_1 = (root\ \text{SUBJ})$ and $n_2 = root$ for the m-definable daughters n_1 and n_2. We then continue with the rules that expand n_1 and n_2. Let us consider the VP rule that expands n_2. For the m-definable n_2 we use the already constructed definition to replace n_2 by its defining term *root* in the instantiated description of the VP rule. From $Inst(\text{VP} \rightarrow (\text{V}, \{\uparrow = \downarrow\})(\text{ADVP}, \{\uparrow = \downarrow\}), (root, n_4\ n_5))$ we then derive the definitions $n_4 = root$ and $n_5 = root$ for its m-definable daughters, and so forth. If we run like this through the whole derivation, for all m-definable daughters we obtain defining terms that do not contain mother-definable node constants. For our example these are the ones in (15).

(15) $n_1 = (root\ \text{SUBJ})$
$n_2 = root$
$n_4 = root$
$n_5 = root$
$n_8 = root$

By substituting all mother-definable node constants by their defining terms we can then produce from the original f-description the equivalent description in (16).

$$(16)\quad \left\{ \begin{array}{c} (root\ \text{SUBJ}) = (root\ \text{SUBJ}), (root\ \text{SUBJ PRED}) = \text{'JOHN'}, \\ root = root, \\ (root\ \text{PRED}) = \text{'FALL}\langle(\text{SUBJ})\rangle\text{'}, (root\ \text{TENSE}) = \text{PAST}, \\ (root\ \text{ADJ}) = (n_7\ \text{ELE}), (n_7\ \text{PRED}) = \text{'TODAY'}, \\ (root\ \text{ADJ}) = (n_{10}\ \text{ELE}), (n_{10}\ \text{PRED}) = \text{'QUICKLY'} \end{array} \right\}$$

Notice that for each acyclic f-structure F the maximal length of the σ in the defining terms is bounded by the depth of F.

Thus we see that a mother-definable constant can be eliminated in favor of a constant corresponding to a higher node and a sequence of attributes leading down

through the f-structure. The constants that are not eliminable are the root constant *root* (if it occurs in *FD*) and all daughter constants that occur in *FD* but are not mother-definable. At least for acyclic f-structures, however, we can show that there is an upper bound on the number of these remaining constants. This is because the remaining constants must each denote one of the elements of the given f-structure, but no two of them can denote the same element. This is a consequence of LFG's instantiation procedure and functional uniqueness condition, and the acyclicity of the f-structure.

Given LFG's instantiation procedure, as formalized in Definition 2, two distinct node constants can be related in a single equation only if the nodes stand in a mother–daughter relationship. Thus a daughter and a node external to the mother cannot be related directly by instantiation but only as a consequence of a deduction involving at least one instantiated annotation of some other licensing rule. Because of LFG's functional uniqueness condition the equations involved in such a deduction cannot contain atomic feature values. The constant/complex clash condition (vii) of Definition 4 prevents atomic values from being substituted for proper subterms and the constant/constant clash condition (vi) prevents them from being equated to nodes. Thus such a deduction can only involve equations relating a daughter to its mother (or a node to itself).

If an undefinable daughter corefers with a node external to the mother, then the deduction that relates them must involve an instantiated annotation of that daughter that is (up to symmetric permutation) of the form $(\uparrow\ \sigma) = (\downarrow\ \sigma')$ with $|\sigma'| > 0$ (such as the adverbial annotations previously mentioned), and there must be deductions from other equations that induce a cycle. This is demonstrated in the following lemma.

Lemma 1
Let c and ρ *be a derivation with f-description FD for an acyclic f-structure in G. If* n_j *is a daughter of n and* n_j *is not m-definable in Inst*(ρ_n) *then FD* $\nvdash n_j = (n'\ \chi)$ *for all* n' *not dominated by* n_j *and all (possibly empty) sequences of attributes* χ.

Proof
Let n_j be a daughter of n that is not m-definable in $Inst(\rho_n)$ and suppose that $n_j = (n'\ \chi)$ would follow from *FD* for node n' not dominated by n_j. Assume further that *FD* and the instantiated descriptions of the licensing rules are closed under symmetry. Now recall that the rule of substituting equals for equals has the form

$$\frac{e \qquad t = t'}{e'}$$

where e is an equation containing subterm t and e' is obtained from e by replacing one occurrence of t in e by t'. Then we know that there is an equation $t = t'$ that is either in *FD* (or follows from *FD* by partial reflexivity[9]) such that $n_j = (n'\ \chi)$ is derivable from $t = t'$ by a left-branching substitution proof of the form

9 It may be the case that such a left-branching proof must start with a reflexive equation $t = t$ that is not in *FD* but can be inferred by partial reflexivity from an equation $(t\ \sigma) = t''$ in *FD*. Partial reflexivity is the restriction of reflexivity to well-defined (object denoting) terms. It is a sound inference rule for the theory of partial functions for which full reflexivity does not hold.

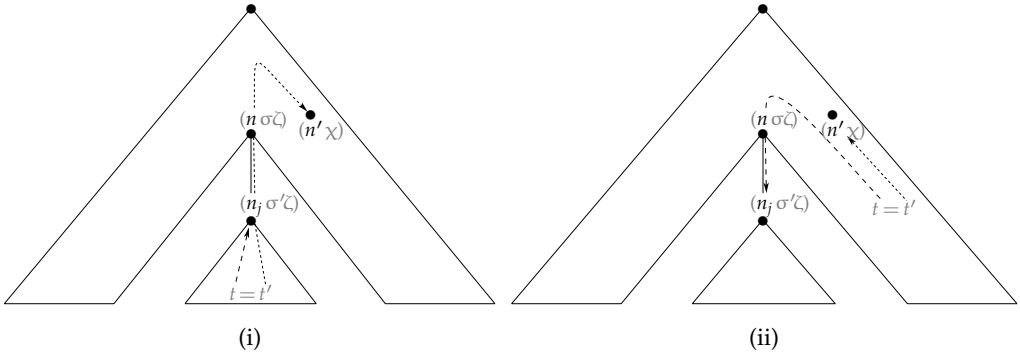


Figure 5
Possible dominance relations between n_j and the node occurring in t'. (i) The node occurring in t' is dominated by n_j. Note that n might occur in t. (ii) The node occurring in t' is not dominated by n_j. As a special case, t might be $(n_j\ \sigma')$. The dashed and dotted arrows indicate the tree traversal performed in the rewriting proofs of $n_j = (n'\ \chi)$ from $t = t'$, that is, the sequences of nodes that must appear in the equations used to rewrite t to n_j and t' to $(n'\ \chi)$, respectively.

$$\dfrac{t = t' \qquad \dfrac{t_1 = t'_1}{e_1} \quad \ddots \quad \dfrac{e_{m-1} \qquad t_m = t'_m}{n_j = (n'\ \chi)}}$$

where t is rewritten to n_j and t' to $(n'\ \chi)$ by a sequence of substitutions all justified by equations $t_i = t'_i \in FD$, $i = 1, .., m$ (cf. Statman 1977; Wedekind 1994, Section 4). Because of LFG's instantiation procedure and the constant/constant and constant/complex clash conditions, each premise of this proof must have the form $(\bar{n}\ \bar{\sigma}) = (\check{n}\ \check{\sigma})$ where either $\bar{n}$ and $\check{n}$ are in a mother–daughter relation or $\bar{n} = \check{n}$. Depending on the dominance relation between n_j and the node occurring in t' there are two possible cases. These are illustrated in Figure 5. (i) If the node occurring in t' is dominated by n_j then there must be a premise $(n_j\ \sigma') = (n\ \sigma)$ from $Inst(\rho_n)$ such that t' is rewritten to $(n_j\ \sigma'\zeta)$ and $(n_j\ \sigma'\zeta)$ to $(n\ \sigma\zeta)$. Because $FD \vdash t = n_j$ and hence $FD \vdash n_j = (n_j\ \sigma'\zeta)$, $|\sigma'| = 0$ due to acyclicity. Thus $n_j = (n\ \sigma) \in Inst(\rho_n)$, contradicting the undefinability assumption. (ii) If the node occurring in t' is not dominated by n_j there must be a premise $(n\ \sigma) = (n_j\ \sigma')$ from $Inst(\rho_n)$ such that either t is rewritten to $(n\ \sigma\zeta)$ and then to $(n_j\ \sigma'\zeta)$, or $t = (n_j\ \sigma')$. Since $FD \vdash t = n_j$, we get in both cases $|\sigma'| = 0$ because of acyclicity and thus the same contradiction as in (i). ■

The following corollary follows directly from Lemma 1.

Corollary 1
Let c and ρ be a derivation with f-description FD for an acyclic f-structure in G. If n_j is a daughter of n and n_j is not m-definable in $Inst(\rho_n)$ then

(i) $FD \not\vdash n_j = root$, *and*

(ii) $FD \not\vdash n_j = n'_i$ *for any distinct node n'_i that is not definable in terms of its mother n' in $Inst(\rho_{n'})$.*

From Corollary 1 it immediately follows that the denotations of *FD*'s undefinable node constants are biunique. So, their number must be less than or equal to the size of the universe of a minimal model M of *FD*. Suppose F is the f-structure for a derivation with f-description *FD*. Because F is isomorphic to $M|\Sigma$ for any minimal model M of *FD*, and $M|\Sigma$ and M share the same universe, we can use F's (finite) universe to define the constants that we require. Thus for each element a of the universe of F that is not denoted by an atomic value we introduce a constant a_{a}.[10]

Definition 9
Let F be an f-structure with $F = (\mathcal{U}, I)$. We define the set of constants $\mathcal{C}_F$ by

$$\mathcal{C}_F = \{a_{\mathsf{a}} \mid \mathsf{a} \in \mathcal{U} \text{ and there is no atomic feature value } v \text{ with } I(v) = \mathsf{a}\}.$$

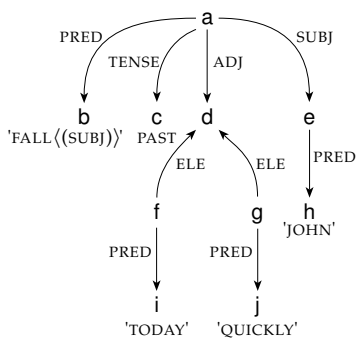
The set $\mathcal{C}_F$ provides a sufficient number of constants to produce an equivalent reduced description by a biunique renaming of the node constants that remain after the m-definable ones are eliminated.

The renaming of the remaining undefinable constants can be accomplished, for example, if we map in the natural way each mother-undefinable daughter n corresponding to a in the isomorphic image F of $M|\Sigma$ to the constant a_{a}. Because of Corollary 1, such a mapping must be biunique. Hence it can be used to rename all undefinable daughters occurring in *FD* and will thus produce an equivalent description where all nodes except *root* are replaced by constants drawn from F.

As an illustration we pick for the f-structure depicted in Figure 4 the structure with the universe in (17a) and the interpretation function whose directed acyclic graph representation is given in (17b).[11]

(17) a. $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f}, \mathsf{g}, \mathsf{h}, \mathsf{i}, \mathsf{j}\}$

b.



The constants we obtain from the structure (17) by Definition 9 are the ones in (18).

(18) $\{a_{\mathsf{a}}, a_{\mathsf{d}}, a_{\mathsf{e}}, a_{\mathsf{f}}, a_{\mathsf{g}}\}$

10 From a computational point of view a constant a_{a} can be regarded as the address of a or a pointer to a.

11 According to our formalization, attribute symbols are interpreted by unary partial functions over the universe and atomic value symbols by elements of the universe. Thus the graph indicates, for example, that the interpretation function assigns to the attribute symbol PRED the (unary) partial function $\{(\mathsf{a}, \mathsf{b}), (\mathsf{e}, \mathsf{h}), (\mathsf{f}, \mathsf{i}), (\mathsf{g}, \mathsf{j})\}$ and to the attribute symbol ELE the partial function $\{(\mathsf{f}, \mathsf{d}), (\mathsf{g}, \mathsf{d})\}$. Furthermore, it interprets the atomic value symbol 'FALL⟨(SUBJ)⟩' as denoting b and PAST as denoting c.

Now, let M be a minimal model of our original f-description. Because an isomorphism between $M|\Sigma$ and our structure (17) must map the denotation of n_7 in M to f and the denotation of n_{10} to g, we can rename n_7 by a_f and n_{10} by a_g and obtain from (16) the equivalent description (19).

$$(19)\quad \left\{\begin{array}{c}(root\ \textsc{subj}) = (root\ \textsc{subj}), (root\ \textsc{subj pred}) = \text{'JOHN'},\\ root = root,\\ (root\ \textsc{pred}) = \text{'FALL}\langle(\textsc{subj})\rangle', (root\ \textsc{tense}) = \textsc{past},\\ (root\ \textsc{adj}) = (a_f\ \textsc{ele}), (a_f\ \textsc{pred}) = \text{'TODAY'},\\ (root\ \textsc{adj}) = (a_g\ \textsc{ele}), (a_g\ \textsc{pred}) = \text{'QUICKLY'}\end{array}\right\}$$

We next compose the substitution that is induced by the definitions of the definable daughters and the substitution that we used to rename the undefinable daughters. This provides a substitution that allows us to produce from the original f-description an equivalent description in a single transformation. If we compose the two substitutions of our example, that is, the one induced by the definitions in (15) and the renaming substitution $\{(n_7, a_f), (n_{10}, a_g)\}$, we arrive at the reducing substitution in (20).

$$(20)\quad \{(n_1, (root\ \textsc{subj})), (n_2, root), (n_4, root), (n_5, root), (n_7, a_f), (n_8, root), (n_{10}, a_g)\}$$

We now give a precise specification of a (finite) set of terms that can serve as the range of the reducing substitutions for all derivations of an f-structure F. This set is obtained from the constants in $\mathcal{C}_F$ and the attributes of F in the following way. We first provide the constants $\mathcal{C}_F$ with their intended interpretation by expanding F in the natural way to the canonical structure $\hat{F}$ for $\Sigma \cup \mathcal{C}_F$.

Definition 10
Let F be an f-structure with $F = (\mathcal{U}, I)$. We define the **canonical expansion** $\hat{F}$ of F to $\Sigma \cup \mathcal{C}_F$ by

$$\hat{F} = (\mathcal{U}, \hat{I}) \text{ with } \hat{I} = I \cup \{(a_a, a) \mid a_a \in \mathcal{C}_F\}.$$

In the canonical expansion $\hat{F}$, each element of the universe is denoted by exactly one constant. Each new constant a_a is interpreted by a and each atomic feature value by its original denotation.

A set $\mathcal{T}_F$ of canonical terms that includes the ranges of the reducing substitutions for all possible derivations of F is a set that contains all terms of the form $(a_a\ \sigma)$ that are defined in $\hat{F}$ but do not denote an element already designated by an atomic feature value. It also contains all terms that we obtain from those by substituting *root* for their constant symbols. This set includes all constants of $\mathcal{C}_F$ (because σ can be empty) and thus all possible constant values for the mother-undefinable daughters of a derivation for F. Because each element in the universe of $\hat{F}$ is denoted by a constant, $\mathcal{T}_F$ also contains all possible defining terms for the mother-definable nodes of that derivation. Terms referring to the denotation of an atomic feature value are not required, since there are (because of the constant/constant clash condition) no node constants with the same denotation as any atomic feature value.

The node constant *root* is substituted for the $\mathcal{C}_F$ constants in every term to account for the fact that different derivations may associate different functional elements with the root of the c-structure. That would be the case, for example, if our grammar contains

in addition the S and VP rules (21a,b) and alternatively derives the adverbials with the rules (21c,d).

(21) a. $\text{S} \rightarrow \underset{\uparrow = (\downarrow \text{ ADJ})}{\text{S}} \quad \underset{\uparrow = \downarrow}{\text{ADVP}}$

b. $\text{VP} \rightarrow \underset{\uparrow = \downarrow}{\text{V}}$

c. $\text{ADVP} \rightarrow \underset{\uparrow = (\downarrow \text{ ELE})}{\text{ADV}} \quad \underset{\uparrow = \downarrow}{\text{ADVP}}$

d. $\text{ADVP} \rightarrow \underset{\uparrow = (\downarrow \text{ ELE})}{\text{ADV}}$

With such a grammar we can derive the f-structure of Figure 4 also with an f-description where *root* denotes the ADJ value and where the top of the f-structure is denoted by the mother-undefinable S node that is expanded by the original start rule. As this example indicates, a grammar might produce several f-descriptions for the same f-structure by anchoring the description at different f-structure elements and then moving along different paths through the structure. This is why the term set must contain the entire set of constants $\mathcal{C}_F$ (and the terms containing them) and not just the ones for the f-structure roots.

Thus $\mathcal{T}_F$ contains sufficiently many constant symbols and defining terms for the reducing substitutions to make all the distinctions that could arise from any c-structure and f-description for the given F. It is defined formally in the following way.

Definition 11
Let $F = (\mathcal{U}, I)$ be an f-structure. On the basis of the canonical expansion $\hat{F} = (\mathcal{U}, \hat{I})$ of F to $\Sigma \cup \mathcal{C}_F$ we first define the set of terms $\overline{\mathcal{T}_F}$

$$\overline{\mathcal{T}_F} = \{(a_{\mathsf{a}}\ \sigma) \in Dom(\hat{I}) \mid a_{\mathsf{a}} \in \mathcal{C}_F \text{ and there is no value } v \in \Sigma \text{ s.t. } \hat{I}(v) = \hat{I}(a_{\mathsf{a}}\ \sigma)\}.$$

The set of terms $\mathcal{T}_F$ that we will use for the grammar construction is then defined by

$$\mathcal{T}_F = \overline{\mathcal{T}_F} \cup \{(root\ \sigma) \mid \text{there is a term } (a_{\mathsf{a}}\ \sigma) \in \overline{\mathcal{T}_F}\} \cup \{\bot\}.$$

For mathematical convenience we add the dummy constant $\bot$ as a value for those nodes of the c-structure that are not interpreted in a minimal model of the f-description. These are just the ones that do not occur in the f-description. The complete set of terms for the structure in (17) is given in (22).

(22) $$\left\{\begin{array}{c} a_{\mathsf{a}}, a_{\mathsf{d}}, a_{\mathsf{e}}, a_{\mathsf{f}}, a_{\mathsf{g}}, \\ (a_{\mathsf{a}}\ \text{ADJ}), (a_{\mathsf{a}}\ \text{SUBJ}), \\ (a_{\mathsf{f}}\ \text{ELE}), (a_{\mathsf{g}}\ \text{ELE}) \end{array}\right\} \cup \left\{\begin{array}{c} root, \\ (root\ \text{ADJ}), (root\ \text{SUBJ}), \\ (root\ \text{ELE}) \end{array}\right\} \cup \{\bot\}$$

We have illustrated that we can reduce the f-description of every derivation of an acyclic f-structure F to an equivalent description if we replace the node constants by terms of $\mathcal{T}_F$. But this assumes that the c-structure and the f-description are already known. Our grammar construction requires us to simulate this reduction without knowing in advance the details of either the c-structure or a particular f-description. And that means only on the basis of the possible values of the reducing substitutions, namely $\mathcal{T}_F$, and the rules of G.

3.2 Reducing the f-Description Space

We now shift our attention to the rules of G and their instantiating terms, that is, to the arguments of the *Inst* function. These are pairs consisting of an m-ary rule r of G and its

instantiating terms $(t, t_1..t_m)$. Let us call such a pair an **instantiation** of r, or sometimes simply an **instantiated rule**. Let us further extend the reducing substitutions that we constructed for the derivations of F to total functions by assigning *root* to *root* and $\perp$ to each non-denoting node constant. Now recall that the f-description of a particular derivation for F consists of the union of the instantiated descriptions of the rules that together license that derivation. If we consider these licensing rules together with their node instantiation, that is, pairs of the form $(r, (n, n_1..n_m))$, and use a reducing substitution for that derivation to replace the node constants in the instantiations by canonical terms, then we obtain a collection of instantiated rules of the form $(r, (t, t_1..t_m))$ all of which are instantiated by terms of $\mathcal{T}_F$. The union of the instantiated descriptions of these rules is identical to the description that the reducing substitution produces from the original f-description. Because R and $\mathcal{T}_F$ are finite, the set of all instantiated-rule collections that we obtain from the (possibly infinite) set of derivations of F by reducing their node-instantiated licensing rules must be finite too. This fact is crucial for our grammar construction.

We further observe that the instantiated rules that result from this substitution are also appropriate in the following sense.

Definition 12
Let r be an m-ary LFG rule in R of G ($m \geq 0$), F be an f-structure, $(t, t_1..t_m) \in \mathcal{T}_F \times \mathcal{T}_F^m$, and $a_1..a_m$ be a sequence of length m of pair-wise distinct constants not in $\mathcal{T}_F$. Then the instantiated rule $(r, (t, t_1..t_m))$ is **appropriately instantiated** (by terms of $\mathcal{T}_F$) iff the following conditions are satisfied:

(i) if $t_j = \perp$ then a_j is not interpreted in a minimal model of $Inst(r, (t, a_1..a_m))$,
(ii) if a_j is m-definable in $Inst(r, (t, a_1..a_m))$ then $Inst(r, (t, a_1..a_m)) \vdash a_j = t_j$,
(iii) otherwise $t_j \in \mathcal{C}_F$, $t_j \neq t$ and $t_j \neq t_i$ for all $i = 1, .., m$ with $i \neq j$.

In the following the set of all appropriately instantiated rules is denoted by IR_F ($IR_F = \{(r, \tau) \in R \times (\mathcal{T}_F \times \mathcal{T}_F^*) \mid (r, \tau)$ is appropriately instantiated$\}$).

The constants $a_1..a_m$ in this definition provide the same discriminations as the daughter nodes of any local tree licensed by the rule. This definition is satisfied by rules that result from eliminating node constants in favor of terms in the way that we have described. Such term-instantiated rules satisfy condition (ii), because whenever the mother is instantiated by t and an m-definable daughter n_j is reduced to a term $(t\ \sigma) \in \mathcal{T}_F$ then also $Inst(r, (t, a_1..a_m)) \vdash a_j = t_j\ (= (t\ \sigma))$. Condition (iii) is satisfied, because of the pair-wise distinctness of the values for the mother-undefinable nodes, due to Corollary 1. And condition (i) holds, because non-denoting node constants are mapped to $\perp$.[12] The set IR_F of all possible appropriately instantiated rules is large but finite, because R and $\mathcal{T}_F$ are finite.

For our start rule (13a) S $\rightarrow$ (NP, $\{(\uparrow$ SUBJ$) = \downarrow\}$)(VP, $\{\uparrow = \downarrow\}$), only the two instantiations in (23) are appropriate.

12 Note that we cannot establish the converse of (i). This is because a daughter node constant that is not interpreted in a minimal model of the instantiated description of a rule might occur in a statement introduced by a rule expanding that daughter. In a minimal model corresponding to a larger derivational context such a daughter constant might thus belong to the interpreted symbols.

(23) a. $\left(^{\mathrm{S}\rightarrow}_{(\uparrow \mathrm{SUBJ})=\downarrow}{}^{\mathrm{NP}}_{\uparrow=\downarrow}{}^{\mathrm{VP}}, (root, (root\ \mathrm{SUBJ})\ root)\right)$

b. $\left(^{\mathrm{S}\rightarrow}_{(\uparrow \mathrm{SUBJ})=\downarrow}{}^{\mathrm{NP}}_{\uparrow=\downarrow}{}^{\mathrm{VP}}, (a_{\mathrm{a}}, (a_{\mathrm{a}}\ \mathrm{SUBJ})\ a_{\mathrm{a}})\right)$

The rule ADVP $\rightarrow$ (ADV, $\{(\uparrow \mathrm{ADJ}) = (\downarrow \mathrm{ELE})\}$)(ADVP, $\{\uparrow = \downarrow\}$), on the other hand, has many appropriate instantiations, among them the ones in (24).

(24) a. $\left(^{\mathrm{ADVP}\rightarrow}_{(\uparrow \mathrm{ADJ})=(\downarrow \mathrm{ELE})}{}^{\mathrm{ADV}}_{\uparrow=\downarrow}{}^{\mathrm{ADVP}}, (root, a_{\mathrm{f}}\ root)\right)$

b. $\left(^{\mathrm{ADVP}\rightarrow}_{(\uparrow \mathrm{ADJ})=(\downarrow \mathrm{ELE})}{}^{\mathrm{ADV}}_{\uparrow=\downarrow}{}^{\mathrm{ADVP}}, (root, a_{\mathrm{g}}\ root)\right)$

c. $\left(^{\mathrm{ADVP}\rightarrow}_{(\uparrow \mathrm{ADJ})=(\downarrow \mathrm{ELE})}{}^{\mathrm{ADV}}_{\uparrow=\downarrow}{}^{\mathrm{ADVP}}, (a_{\mathrm{a}}, a_{\mathrm{g}}\ a_{\mathrm{a}})\right)$

d. $\left(^{\mathrm{ADVP}\rightarrow}_{(\uparrow \mathrm{ADJ})=(\downarrow \mathrm{ELE})}{}^{\mathrm{ADV}}_{\uparrow=\downarrow}{}^{\mathrm{ADVP}}, ((a_{\mathrm{a}}\ \mathrm{ADJ}), a_{\mathrm{g}}\ (a_{\mathrm{a}}\ \mathrm{ADJ}))\right)$

Instantiations that are not appropriate for this rule are, for example, the ones in (25).

(25) a. $\left(^{\mathrm{ADVP}\rightarrow}_{(\uparrow \mathrm{ADJ})=(\downarrow \mathrm{ELE})}{}^{\mathrm{ADV}}_{\uparrow=\downarrow}{}^{\mathrm{ADVP}}, (root, root\ root)\right)$

b. $\left(^{\mathrm{ADVP}\rightarrow}_{(\uparrow \mathrm{ADJ})=(\downarrow \mathrm{ELE})}{}^{\mathrm{ADV}}_{\uparrow=\downarrow}{}^{\mathrm{ADVP}}, (a_{\mathrm{a}}, a_{\mathrm{a}}\ a_{\mathrm{a}})\right)$

They are not properly discriminating, because the ADV node of any derived f-description must denote an entity distinct from the denotation of the mother and the other daughter node.

The instantiations in (23a) and (24a) are the ones obtained from the derivation in Figure 4 and the reducing substitution (20). Note that the appropriately instantiated rule (23b) that does not associate the S node with the root constant might result from derivations where the top of the f-structure is not denoted by the root node of the c-structure, as illustrated with the rules in (21).

So far we have considered only the individual instantiated rules that we obtain from the licensing rules of a derivation for F by replacing the node constants as described by terms of $\mathcal{T}_F$. As a consequence of Corollary 1, we also observe that our reducing substitutions never replace undefinable daughters of two distinct node-instantiated licensing rules by one and the same constant. That is, the term-instantiated rules that result from two distinct node-instantiated licensing rules always satisfy the following compatibility relation.

Definition 13
Two appropriately instantiated rules $(r, (t, t_1..t_m))$ and $(r', (t', t'_1..t'_l))$ of IR_F are **compatible** iff

$$t_i \neq t'_j \text{ for all } t_i, t'_j \in \mathcal{C}_F \text{ with } t_i \neq t \text{ and } t'_j \neq t' \ (1 \leq i \leq m, 1 \leq j \leq l).$$

Given appropriateness, the conditions $t_i \in \mathcal{C}_F$ and $t_i \neq t$ imply that t_i is not definable in terms of t in the instantiated description of r. In essence, two instantiated rules are compatible only if there are no repetitions of daughter constants instantiating mother-undefinable daughters: All shared daughter constants instantiate mother-definable daughters. Incompatible instantiations do not respect the biuniqueness property given by Corollary 1 and therefore cannot appear together in the set of $\mathcal{T}_F$-instantiated rules for any derivation of F. Note that this compatibility relation is symmetric, but reflexive only for those instantiated rules $(r, (t, t_1..t_m))$ where each daughter that is instantiated

by a constant from $\mathcal{C}_F$ is mother-definable. As a consequence of Corollary 1, only an instantiated rule that is compatible with itself can emerge from two separate applications of r in a derivation of F.

The instantiated rules in (26a–c), for example, are compatible while the ones in (26d) are not. The latter rules mistakenly introduce an identity that, because of Corollary 1, can never be derived by the grammar. The rules in (26a) result from reducing the licensing rules of the derivation in Figure 4 with the reducing substitution (20).

(26) a. $\left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ ADJ})=(\downarrow \text{ ELE})}{\text{ADV}} \underset{\uparrow=\downarrow}{\text{ADVP}}, (root, a_{\text{f}}\ root)\right)$ $\left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ ADJ})=(\downarrow \text{ ELE})}{\text{ADV}}, (root, a_{\text{g}})\right)$

b. $\left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ ADJ})=(\downarrow \text{ ELE})}{\text{ADV}} \underset{\uparrow=\downarrow}{\text{ADVP}}, (root, a_{\text{g}}\ root)\right)$ $\left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ ADJ})=(\downarrow \text{ ELE})}{\text{ADV}}, (root, a_{\text{f}})\right)$

c. $\left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ ADJ})=(\downarrow \text{ ELE})}{\text{ADV}} \underset{\uparrow=\downarrow}{\text{ADVP}}, (root, a_{\text{a}}\ root)\right)$ $\left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ ADJ})=(\downarrow \text{ ELE})}{\text{ADV}}, (root, a_{\text{f}})\right)$

d. $\left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ ADJ})=(\downarrow \text{ ELE})}{\text{ADV}} \underset{\uparrow=\downarrow}{\text{ADVP}}, (root, a_{\text{f}}\ root)\right)$ $\left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ ADJ})=(\downarrow \text{ ELE})}{\text{ADV}}, (root, a_{\text{f}})\right)$

Our observations lead to a definition that characterizes reducing substitutions entirely in terms of the identified properties of the $\mathcal{T}_F$-instantiated rules and thus in a way that will permit us to simulate their construction by a refinement of the context-free backbone of G. In the following definition we use $\mathcal{N}_c$ to denote the nodes of a c-structure c and $\gamma[\psi]$ to indicate the expression that is obtained from an expression γ (term, sequence of terms, formula, set of formulas, etc.) and a substitution ψ (mapping from constants to terms) by replacing all occurrences of constants a in γ simultaneously by $\psi(a)$.

Definition 14
Let c and ρ be a derivation of f-structure F in G and ψ be a mapping from $\mathcal{N}_c$ into $\mathcal{T}_F$. Then ψ is a **reducing substitution** for the given derivation iff $\psi(root) = root$, and for all $n, n' \in Dom(\rho)$ with $n \neq n'$

(i) $(\rho_n, (n, dts(n))[\psi])$ is appropriately instantiated, and
(ii) $(\rho_n, (n, dts(n))[\psi])$ is compatible with $(\rho_{n'}, (n', dts(n'))[\psi])$.

That reducing substitutions in fact preserve equivalence is then established by the following lemma.

Lemma 2
Let c and ρ be a derivation with f-description FD and f-structure F in G. If ψ is a reducing substitution for c and ρ, then $FD \equiv FD[\psi]$.

Proof
We prove the lemma by induction on the number of nodes, according to a left-to-right, top–down traversal of the c-structure. Let c and ρ be a derivation with f-description FD and f-structure F in G, $M = (\mathcal{U}, I)$ a minimal model of FD, and ψ a reducing substitution for c and ρ. We first define for each node n of c the set $\mathcal{N}_n$ consisting of all nodes higher than n, all nodes of the same depth as n but preceding (on the left), and n. Now for each $\mathcal{N}_n$ with $|\mathcal{N}_n| = i$ let the function ψ_i be the restriction of ψ to $\mathcal{N}_n$ ($\psi|\mathcal{N}_n$). Then we can show by induction for each $i = 1, .., |\mathcal{N}_c|$ that $FD \equiv FD[\psi_i]$, that is, left-to-right, top–down. The equivalence is established by constructing a minimal model M_i also on the universe $\mathcal{U}$ of M. Thus the isomorphism between $M|\Sigma$ and $M_i|\Sigma$ is the identity function.

The basis, $i = 1$, is trivial, because $\psi_1 = \{(root, root)\}$ by definition. Thus $FD[\psi_1] = FD$ and $M_1 = M$ is a minimal model of $FD[\psi_1]$. Hence $FD \equiv FD[\psi_1]$. For the induction step, let $i > 1$. Then $FD \equiv FD[\psi_{i-1}]$ by hypothesis. Let $M_{i-1} = (\mathcal{U}, I_{i-1})$ be a minimal model of $FD[\psi_{i-1}]$, and suppose that node n_j with mother n is the next node in the sequence (i.e., $|\mathcal{N}_{n_j}| = i$).
If n_j is not interpreted in M, it does not occur in FD and hence not in $FD[\psi_{i-1}]$. Thus $FD[\psi_i] = FD[\psi_{i-1}]$, $M_i = M_{i-1}$ is a minimal model of $FD[\psi_i]$, and $FD \equiv FD[\psi_i]$.
If n_j is interpreted in M, there are two cases to consider.
(a) If n_j is m-definable in $Inst(\rho_n, (\psi_{i-1}(n), dts(n)))$ and $\psi(n_j) = t_j$ then $FD[\psi_{i-1}] \vdash n_j = t_j$. Because n_j does not occur in t_j and hence not in $FD[\psi_i]$, $FD[\psi_{i-1}]$ is logically equivalent to the definitional extension $FD[\psi_i] \cup \{n_j = t_j\}$ of $FD[\psi_i]$. Because t_j occurs in $FD[\psi_i]$, $M_i = M_{i-1}|(Dom(I_{i-1})\backslash\{n_j\})$ is a minimal model of $FD[\psi_i]$. Hence $M_{i-1}|\Sigma \cong M_i|\Sigma$ and $FD \equiv FD[\psi_i]$.
(b) If n_j is not m-definable in $Inst(\rho_n, (\psi_{i-1}(n), dts(n)))$ then $\psi(n_j) \in \mathcal{C}_F$. Let $\psi(n_j) = a_{\mathsf{a}}$. Then a_{a} cannot occur in $FD[\psi_{i-1}]$, because the instantiation is appropriate and pairwise compatible and $a_{\mathsf{a}} \neq root$ $(= \psi(root))$.[13] So the model M_i that results from M_{i-1} by renaming n_j by a_{a} must be a minimal model of $FD[\psi_i]$. Thus $M_{i-1}|\Sigma \cong M_i|\Sigma$ and $FD \equiv FD[\psi_i]$.
Hence, $FD \equiv FD[\psi_{|\mathcal{N}_c|}] = FD[\psi]$. ■

Appropriateness and compatibility do not ensure that undefinable daughter constants are distinct from the root. This case is covered, however, because we kept *root* for the root.

We indicated earlier that for an arbitrary derivation of an acyclic f-structure F we can—dependent on a minimal model of its f-description—construct a substitution with range $\mathcal{T}_F$ that satisfies the conditions of Definition 14. We now provide a rigorous proof of this assertion.

Lemma 3
For every derivation of an acyclic f-structure F in G there exists a reducing substitution.

Proof
Suppose there is a derivation c and ρ with f-description FD and f-structure F in G, that FD has minimal model $M = (\mathcal{U}, I)$, and that h is an isomorphism between $M|\Sigma$ and F. Suppose furthermore that c has depth k. For each $i = 0, .., k$ we define by induction a function $\psi_i : \mathcal{N}_c \mapsto \mathcal{T}_F$ as follows. For the root ($i = 0$) we set $\psi_0(root) = root$. Suppose we have defined ψ_{i-1}, $0 < i \leq k$. We then set $\psi_i(n) = \psi_{i-1}(n)$ for all $n \in Dom(\psi_{i-1})$. Now, let n_j be a node of depth i with mother n. If n_j is not interpreted in M we set $\psi_i(n_j) = \perp$. If n_j is interpreted in M we set

$$\psi_i(n_j) = \begin{cases} (\psi_{i-1}(n)\ \sigma) \text{ s.t. } Inst(\rho_n) \vdash (n\ \sigma) = n_j & \text{if } n_j \text{ is m-definable in } Inst(\rho_n) \\ a_{h(I(n_j))} & \text{otherwise.} \end{cases}$$

13 Of course, the constant a_{a} cannot occur as a proper subterm of any other ψ_{i-1} value. If ψ_{i-1} were to map a node n' to $(a_{\mathsf{a}}\ \sigma)$ then n' must be m-definable and there must be a node dominating n' that is not m-definable and mapped to a_{a}. Because $a_{\mathsf{a}} \neq root$, a_{a} must instantiate a daughter of another rule, contradicting compatibility.

Now, let $\psi = \psi_k$. Then ψ trivially satisfies the appropriateness conditions (i) and (ii) by definition. Appropriateness condition (iii) and compatibility follow by Corollary 1. Thus ψ is a reducing substitution for c and ρ. ■

Lemma 3 ensures that there exists a reducing substitution for every derivation of an acyclic f-structure F in G. By Lemma 2 we know that such a substitution preserves equivalence.[14] Thus, the collections of instantiated rules that result from the derivations for F and their reducing substitutions must belong to the set consisting of all possible collections of appropriately instantiated and pair-wise compatible rules that together provide descriptions of F. If we extend the *Inst* function in the obvious way to sets of instantiated rules IR

$$Inst(IR) = \bigcup_{(r,\tau)\in IR} Inst(r,\tau)$$

then this set is defined as follows.

Definition 15
Let F be an f-structure. Then IRD_F is the set of all sets $IR \subseteq IR_F$ such that

(i) for all $(r,\tau),(r',\tau') \in IR$ with $(r,\tau) \neq (r',\tau')$, (r,τ) is compatible with (r',τ'),
(ii) $M|\Sigma \cong F$, for a minimal model M of $Inst(IR)$.

This is a finite set whose size is bounded by a function of the sizes of R and $\mathcal{T}_F$.

Lemma 2 also shows that we can produce an equivalent description for any derived f-description of F, not only with the model-dependent substitutions used in the proof of Lemma 3, but in general with any mapping that satisfies the definition of a reducing substitution. This is important for our grammar construction, because it provides the conditions that we have to control to make sure that we simulate the derivations of f-descriptions for F together with equivalence-preserving substitutions. Under these conditions we can reduce the sets of node-instantiated licensing rules of the simulated derivations to collections that are also included in IRD_F. IRD_F can be determined without knowing the details of the valid derivations for F, just on the basis of F and the LFG grammar G alone.

3.3 Producing the Context-free Grammar G_F

The context-free grammar G_F that simulates all valid derivations for F in G is specified in the following definition. From this we can produce all strings in $Gen_G(F)$ by conventional context-free generation algorithms.

Definition 16
Let $G = (N, T, S, R)$ be an LFG grammar and F be an acyclic f-structure. For G and F we construct a context-free grammar $G_F = (N_F, T_F, S_F, R_F)$ in the following way. The collection of nonterminals N_F is the (finite) set

$$\{S_F\} \cup (N \times \mathcal{T}_F \times \bigcup\{Pow(IR) \mid IR \in IRD_F\})$$

14 Note that the particular substitution that we construct in the proof of Lemma 3 reduces *FD* to an equivalent description that is satisfied in an expansion of $\hat{F}$ by an interpretation for *root* in $\mathcal{U}$.

where S_F is a new root category. Categories in N_F other than S_F are written $A{:}t{:}IR$, where A is a category in N, t is a term in $\mathcal{T}_F$, and IR is a subset of a set of instantiated rules in IRD_F. T_F is the set $T \times \mathcal{T}_F \times \{\emptyset\}$.[15] The rules R_F are constructed from the annotated rules R of G. We include all and only rules of the form:

(i) $S_F \rightarrow S{:}root{:}IR_{root}$, where IR_{root} is any element of IRD_F,

(ii) $A{:}t{:}IR \rightarrow X_1{:}t_1{:}IR_1..X_m{:}t_m{:}IR_m$ such that

(a) there is an $r \in R$ expanding A to $X_1..X_m$,

(b) $IR = \{(r, (t, t_1..t_m))\} \cup \bigcup_{j=1}^{m} IR_j$,

(c) if $(r, (t, t_1..t_m)) \in IR_j$ $(j = 1, .., m)$, or $(r', \tau') \in IR_i \cap IR_j$ and $i \neq j$ $(i, j = 1, .., m)$, then $(r, (t, t_1..t_m))$, respectively (r', τ'), is compatible with itself.

We define the projection $Cat(X{:}t{:}IR) = X$ for every category in $N_F \cup T_F$ except S_F and extend this function in the natural way to strings of categories and sets of strings of categories. Note that the set

$$Cat(L(G_F)) = \{s \mid \exists s' \in L(G_F) \text{ such that } Cat(s') = s\}$$

is context-free, because the set of context-free languages is closed under homomorphisms such as *Cat*.[16]

Before presenting our main theorem and its proof let us sketch how the derivations for F in G are simulated by the context-free grammar G_F.

The grammar G_F expands the root symbol S_F to complex categories of the form $S{:}root{:}IR_{root}$ containing the root category S of G as their first component. A derivation from $S{:}root{:}IR_{root}$ in G_F then consists of a phrase structure tree whose nodes are labeled with refinements of the categories of the original LFG grammar. By taking the *Cat* projection of every category, we obtain the c-structure of at least one derivation for F in G that is simulated by the derivation from $S{:}root{:}IR_{root}$ in G_F. The term component of the augmented categories encodes a reducing substitution ψ for the simulated derivation with the given c-structure. That is, if a node n in the G_F derivation is labeled by $X{:}t{:}IR$, then $\psi(n) = t$ for the corresponding LFG c-structure tree.

The component IR contains all instantiated rules of G that are required to license the subderivation in G that corresponds (under the *Cat* projection) to the subderivation from n in G_F, except that the licensed nodes are replaced in the instantiated rules by their ψ values.[17] Thus, the additional components of the root label $S{:}root{:}IR_{root}$ record

15 The set of terminals T_F is constructed from the full term set $\mathcal{T}_F$ instead of just $\perp$ to allow for the possibility of $\downarrow$ appearing in lexical entries (e.g., Zaenen and Kaplan 1995).

16 Cf. Hopcroft and Ullman (1979).

17 The rule component IR is a refinement of the third component of the categories defined in Kaplan and Wedekind (2000). The categories there were distinguished by $Inst(IR)$, the descriptions produced by collecting the instantiated annotations from our third-component rules, and thus give a more compact representation whenever different subderivations provide the same instantiated description. As we demonstrated, such a simpler representation is sufficient to control the generation process for grammars with a conventional set of descriptive devices. Instantiated descriptions, however, do not provide enough information for grammars with devices whose evaluation requires the c-structure to be taken into account, as, for example, functional precedence (Bresnan 1995; Zaenen and Kaplan 1995). We show in Wedekind and Kaplan (forthcoming) that these devices can be modeled with our more elaborate rule representation.

that $\psi(root)$ is set to *root* (the initial condition for reducing substitutions) and that the node-instantiated licensing rules of the simulated derivation are reduced to IR_{root} by ψ. Each application of a rule $A{:}t{:}IR \rightarrow X_1{:}t_1{:}IR_1..X_m{:}t_m{:}IR_m$ that expands a nonterminal node n of the derivation in G_F simulates the application of an LFG rule with context-free backbone $A \rightarrow X_1..X_m$ whose instantiation with $(t, t_1..t_m)$ combines with the instantiated-rule components of all daughters to form the rule component IR of the mother.[18] Now, ψ must be a reducing substitution for the simulated derivation, because all instantiated rules in IR_{root} are appropriately instantiated and pair-wise compatible and because condition (iic) of Definition 16 ensures that rules that are not self-compatible can only be used once for licensing the *Cat* projection. Thus, because of Lemma 2, the derivation in G_F simulates a derivation of an f-description in G that ψ reduces to the equivalent description provided by IR_{root}.

We can also see that every derivation for F in G is simulated by a derivation in G_F. We know from Lemmas 2 and 3 that we can construct for every derivation of an f-description for F in G a reducing substitution ψ that produces a description equivalent to the original one. Based on ψ we can then augment the category labels of the c-structure of a derivation for F in G by term and rule components that record ψ and the licensing rules (with the node constants replaced by their ψ values). We thus obtain a derivation from $S{:}root{:}IR_{root}$ where the instantiated description provided by IR_{root} is equivalent to the original f-description. Because G_F contains a start rule for every set of appropriately instantiated and pair-wise compatible rules that provides a description of F, there must also be a rule that expands S_F to $S{:}root{:}IR_{root}$ and the terminal string of the derivation for F in G must be the *Cat* projection of a derivable string in G_F.

We are now prepared to prove our main theorem.

Theorem
For any LFG grammar G and any acyclic f-structure F, $Gen_G(F) = Cat(L(G_F))$.

Proof
We prove first that $Gen_G(F) \subseteq Cat(L(G_F))$. Suppose there is a derivation c and ρ of a terminal string s with f-description FD and f-structure F in G. By Lemma 3, there exists a reducing substitution ψ for c and ρ. Thus $FD \equiv FD[\psi]$ by Lemma 2. We construct a derivation c' and ρ' of s' from $S{:}root{:}IR_{root}$ with $Cat(s') = s$. We obtain c' by relabeling each node n with label X by $X{:}\psi(n){:}\{(\rho_{\bar{n}}, (\bar{n}, dts(\bar{n}))[\psi]) \mid n \text{ dominates nonterminal node } \bar{n}\}$. That means that the c-structures of both derivations share the same tree skeleton. We define ρ' for each nonterminal node n with label $A{:}\psi(n){:}IR$ and $dts(n) = n_1..n_m$ with labels $X_1{:}\psi(n_1){:}IR_1, .., X_m{:}\psi(n_m){:}IR_m$ by $\rho'_n = A{:}\psi(n){:}IR \rightarrow X_1{:}\psi(n_1){:}IR_1..X_m{:}\psi(n_m){:}IR_m$. Because $FD[\psi] = Inst(IR_{root})$ by construction of IR_{root} and because $IR_{root} \subseteq IR_F$ and condition (i) of Definition 15 hold by the properties of ψ, IR_{root} must be an element of IRD_F. Thus $S_F \rightarrow S{:}root{:}IR_{root}$ is in R_F. Moreover, $Ran(\rho') \subseteq R_F$, because by construction the rule components are subsets of IR_{root}, the rule components of the terminals are empty, and the rules satisfy (iia,b) of Definition 16 by construction and (iic) because ψ is a reducing substitution. Thus $s \in Cat(L(G_F))$.

We now prove that $Cat(L(G_F)) \subseteq Gen_G(F)$. Suppose there is a G_F derivation c' and ρ' of s' from $S{:}root{:}IR_{root}$ with $Cat(s') = s$ and $IR_{root} \in IRD_F$. We first construct a new c-structure c with the same tree skeleton as c' by relabeling each node n with label $X{:}t{:}IR$

18 Note that the licensing LFG rule might not be uniquely determined if the derivation in G_F simulates recursions.

by X. We define a substitution ψ by setting $\psi(n) = t$ for each node n with label $X{:}t{:}IR$. We then show that there is a mapping ρ into R licensing c with $FD \equiv Inst(IR_{root})$. By induction on the depth of the subtrees we first define for each nonterminal n a function ρ^n from all nonterminal nodes dominated by n into R such that

(a) ρ^n licenses the subtree of c with root n,

(b) $IR = \{(\rho^n_{\bar{n}}, (\bar{n}, dts(\bar{n}))[\psi]) \mid n$ dominates nonterminal node $\bar{n}\}$ if n has label $A{:}t{:}IR$ in c',

and, for all $\bar{n}, \check{n} \in Dom(\rho^n)$ with $\bar{n} \neq \check{n}$

(c) $(\rho^n_{\bar{n}}, (\bar{n}, dts(\bar{n}))[\psi])$ is appropriately instantiated and

(d) $(\rho^n_{\bar{n}}, (\bar{n}, dts(\bar{n}))[\psi])$ is compatible with $(\rho^n_{\check{n}}, (\check{n}, dts(\check{n}))[\psi])$.

Suppose n with $dts(n) = n_1..n_m$ is expanded by $\rho'_n = A{:}t{:}IR \rightarrow X_1{:}t_1{:}IR_1..X_m{:}t_m{:}IR_m$ in c', then there is a rule $r \in R$ satisfying the conditions of Definition 16(ii). Thus, r expands A to $X_1..X_m$, $IR = \{(r, (t, t_1..t_m))\} \cup \bigcup_{j=1}^{m} IR_j$, and $(r, (n, dts(n))[\psi]) = (r, (t, t_1..t_m))$ by definition of ψ. If n is a preterminal node then $IR_j = \emptyset$ (for each $j = 1, .., m$). We then set $\rho^n = \{(n, r)\}$ and (a)–(d) hold trivially. If ρ^{n_j} has been defined for all nonterminal daughters n_j we set $\rho^n = \{(n, r)\} \cup \bigcup\{\rho^{n_j} \mid n_j$ is a nonterminal daughter of $n\}$. Then (a)–(c) by construction of ρ'_n and by the inductive hypothesis, and (d) by Definition 16(iic) and because $IR \subseteq IR_{root}$ by Definition 15(i). So, $\rho = \rho^{root}$ licenses c, ψ is a reducing substitution for c and ρ, and $FD \equiv FD[\psi]$ by Lemma 2. Then $FD[\psi] = Inst(IR_{root})$ by (b) and thus s is derivable in G with F. ■

The following corollary is an immediate consequence of this theorem.

Corollary 2
For any LFG grammar G and any acyclic f-structure F, $Gen_G(F)$ is a context-free language.

3.4 A Few Examples

In the preceding sections we have shown how to construct a context-free grammar that generates exactly the set of strings that an LFG grammar assigns to a given f-structure. Those strings can be produced by running a context-free generator with that grammar. In this section we provide examples to illustrate the derivation space of the constructed context-free grammar and the correspondence between the derivations of the constructed grammar and the derivations of the original LFG grammar.

As one illustration of the correspondences between the derivations, let us consider the f-structure F in (27) and the LFG grammar with the rules (13) and the VP rule in (2).

(27)
$$\begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{'JOHN'} \end{bmatrix} \\ \text{PRED} & \text{'FALL}\langle(\text{SUBJ})\rangle\text{'} \\ \text{TENSE} & \text{PAST} \end{bmatrix}$$

For this grammar there is only one derivation of a string with the given f-structure, the one that is depicted in the upper part of Figure 6. The figure shows the derivation with all its components, that is, the c-structure, the rule-mapping ρ together with the node instantiation of the licensing rules, and the f-description. This LFG derivation is

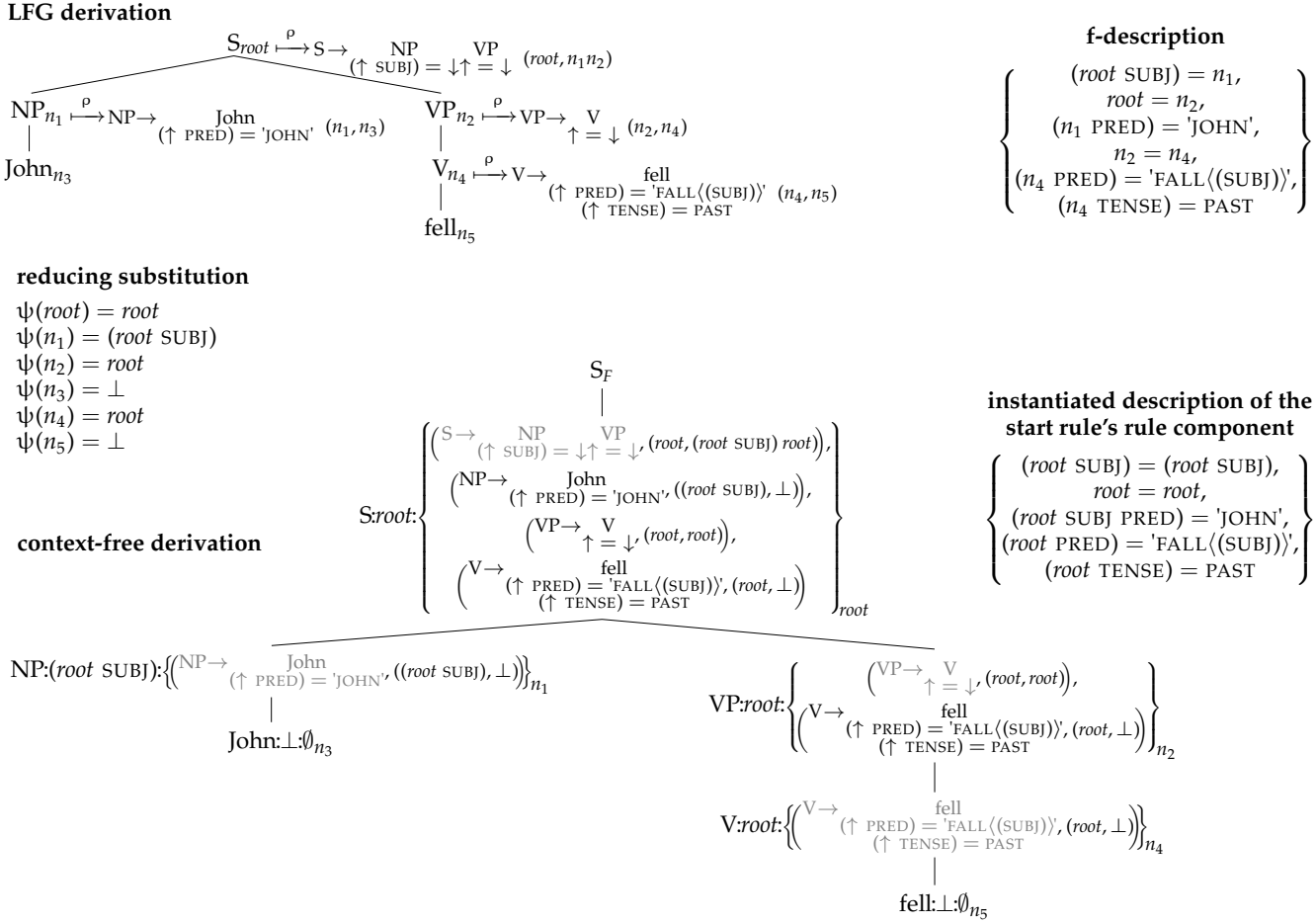


Figure 6
The LFG derivation for (27) and the rules in (2) and (13) with the corresponding context-free derivation of the constructed grammar.

simulated in the constructed context-free grammar by the derivation that is shown in the lower part of the figure.

Both the depicted substitution ψ and the subtree to which S_F expands are related to the original LFG derivation by the construction of the first half of our proof. That is, ψ is a reducing substitution and the context-free derivation specializes the category label of each node n of the original c-structure. The term component is n's ψ value. The rule component is the set of all instantiated rules that result from the licensing rules of the corresponding n-dominated LFG subderivation. These are instantiated by replacing the instantiating nodes of the LFG derivation by their ψ values. Thus, the instantiated description provided by the rule component of the start rule is equivalent to the original f-description and hence the context-free derivation tree at the bottom of Figure 6 is licensed completely by the rules of the constructed grammar. Note that the *Cat* projection of the terminal string of the context-free derivation is the terminal string of the c-structure, the sentence *John fell*.

On the other hand, the depicted LFG derivation and the context-free derivation are also related by the construction of the second half of the proof. The c-structure is the *Cat* projection of the constituent structure that S_F's daughter derives. The reducing substitution maps each node of this c-structure to the term of its complex label in the corresponding context-free derivation. And the LFG rule that the licensing mapping maps to each node is the rule of the node label's rule component that licenses the node and its daughters in the *Cat* projection. This is instantiated by the term components of the applied context-free rule and combines with the rule components of the daughters to form the rule component of the mother. These licensing LFG rules for the immediate daughters are shown in gray in the rule component of the node labels in the context-free derivation.

As a more complicated illustration, we sketch the derivations of the context-free grammar G_F produced for the f-structure F given in (17) and the grammar comprising the rules in (13). This LFG grammar produces two terminal strings for the given input, *John fell today quickly* and *John fell quickly today*. A set of pair-wise compatible appropriately instantiated rules that yields a description of the input f-structure is, for example, the one contained in the start rule (28). This set arises from reducing the node-instantiated licensing rules of the derivation in Figure 4 with the reducing substitution (20) extended by mapping non-denoting nodes to $\perp$.

(28)
$$S_F \rightarrow S{:}root{:}\left\{\begin{array}{l}\left(\begin{array}{l}S\rightarrow \begin{array}{cc}NP & VP\\ (\uparrow \text{SUBJ}) = \downarrow & \uparrow = \downarrow\end{array}\end{array}, (root, (root\ \text{SUBJ})\ root)\right),\\ \left(\begin{array}{l}NP\rightarrow \begin{array}{c}John\\ (\uparrow \text{PRED}) = \text{'JOHN'}\end{array}\end{array}, ((root\ \text{SUBJ}), \perp)\right),\\ \left(\begin{array}{l}VP\rightarrow \begin{array}{cc}V & ADVP\\ \uparrow = \downarrow & \uparrow = \downarrow\end{array}\end{array}, (root, root\ root)\right),\\ \left(\begin{array}{l}V\rightarrow \begin{array}{c}fell\\ (\uparrow \text{PRED}) = \text{'FALL}\langle(\text{SUBJ})\rangle\text{'}\\ (\uparrow \text{TENSE}) = \text{PAST}\end{array}\end{array}, (root, \perp)\right),\\ \left(\begin{array}{l}ADVP\rightarrow \begin{array}{cc}ADV & ADVP\\ (\uparrow \text{ADJ}) = (\downarrow \text{ELE}) & \uparrow = \downarrow\end{array}\end{array}, (root, a_f\ root)\right),\\ \left(\begin{array}{l}ADV\rightarrow \begin{array}{c}today\\ (\uparrow \text{PRED}) = \text{'TODAY'}\end{array}\end{array}, (a_f, \perp)\right),\\ \left(\begin{array}{l}ADVP\rightarrow \begin{array}{c}ADV\\ (\uparrow \text{ADJ}) = (\downarrow \text{ELE})\end{array}\end{array}, (root, a_g)\right),\\ \left(\begin{array}{l}ADV\rightarrow \begin{array}{c}quickly\\ (\uparrow \text{PRED}) = \text{'QUICKLY'}\end{array}\end{array}, (a_g, \perp)\right)\end{array}\right\}$$

The only useful rule of G_F for expanding the daughter of rule (28) is the rule (29). All other admissible distributions of the members of the mother's rule component also result in rules of G_F. But these other rules cannot be used to produce a terminal

(29)
$$\text{S}{:}root{:}\left\{\begin{array}{l}\left(\text{S}\rightarrow \underset{(\uparrow \text{SUBJ})=\downarrow}{\text{NP}}\ \underset{\uparrow=\downarrow}{\text{VP}},\ (root, (root\ \text{SUBJ})\ root)\right),\\ \left(\text{NP}\rightarrow \underset{(\uparrow \text{PRED})=\text{'JOHN'}}{\text{John}},\ ((root\ \text{SUBJ}), \bot)\right),\\ \left(\text{VP}\rightarrow \underset{\uparrow=\downarrow}{\text{V}}\ \underset{\uparrow=\downarrow}{\text{ADVP}},\ (root, root\ root)\right),\\ \left(\text{V}\rightarrow \underset{\substack{(\uparrow \text{PRED})=\text{'FALL}\langle(\text{SUBJ})\rangle\text{'}\\ (\uparrow \text{TENSE})=\text{PAST}}}{\text{fell}},\ (root, \bot)\right),\\ \left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ADJ})=(\downarrow \text{ELE})}{\text{ADV}}\ \underset{\uparrow=\downarrow}{\text{ADVP}},\ (root, a_{\text{f}}\ root)\right),\\ \left(\text{ADV}\rightarrow \underset{(\uparrow \text{PRED})=\text{'TODAY'}}{\text{today}},\ (a_{\text{f}}, \bot)\right),\\ \left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ADJ})=(\downarrow \text{ELE})}{\text{ADV}},\ (root, a_{\text{g}})\right),\\ \left(\text{ADV}\rightarrow \underset{(\uparrow \text{PRED})=\text{'QUICKLY'}}{\text{quickly}},\ (a_{\text{g}}, \bot)\right)\end{array}\right\}\rightarrow$$
$$\text{NP}{:}(root\ \text{SUBJ}){:}\left\{\left(\text{NP}\rightarrow \underset{(\uparrow \text{PRED})=\text{'JOHN'}}{\text{John}},\ ((root\ \text{SUBJ}), \bot)\right)\right\}\quad \text{VP}{:}root{:}\left\{\begin{array}{l}\left(\text{VP}\rightarrow \underset{\uparrow=\downarrow}{\text{V}}\ \underset{\uparrow=\downarrow}{\text{ADVP}},\ (root, root\ root)\right),\\ \left(\text{V}\rightarrow \underset{\substack{(\uparrow \text{PRED})=\text{'FALL}\langle(\text{SUBJ})\rangle\text{'}\\ (\uparrow \text{TENSE})=\text{PAST}}}{\text{fell}},\ (root, \bot)\right),\\ \left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ADJ})=(\downarrow \text{ELE})}{\text{ADV}}\ \underset{\uparrow=\downarrow}{\text{ADVP}},\ (root, a_{\text{f}}\ root)\right),\\ \left(\text{ADV}\rightarrow \underset{(\uparrow \text{PRED})=\text{'TODAY'}}{\text{today}},\ (a_{\text{f}}, \bot)\right),\\ \left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ADJ})=(\downarrow \text{ELE})}{\text{ADV}},\ (root, a_{\text{g}})\right),\\ \left(\text{ADV}\rightarrow \underset{(\uparrow \text{PRED})=\text{'QUICKLY'}}{\text{quickly}},\ (a_{\text{g}}, \bot)\right)\end{array}\right\}$$

(30)
$$\text{VP}{:}root{:}\left\{\begin{array}{l}\left(\text{VP}\rightarrow \underset{\uparrow=\downarrow}{\text{V}}\ \underset{\uparrow=\downarrow}{\text{ADVP}},\ (root, root\ root)\right),\\ \left(\text{V}\rightarrow \underset{\substack{(\uparrow \text{PRED})=\text{'FALL}\langle(\text{SUBJ})\rangle\text{'}\\ (\uparrow \text{TENSE})=\text{PAST}}}{\text{fell}},\ (root, \bot)\right),\\ \left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ADJ})=(\downarrow \text{ELE})}{\text{ADV}}\ \underset{\uparrow=\downarrow}{\text{ADVP}},\ (root, a_{\text{f}}\ root)\right),\\ \left(\text{ADV}\rightarrow \underset{(\uparrow \text{PRED})=\text{'TODAY'}}{\text{today}},\ (a_{\text{f}}, \bot)\right),\\ \left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ADJ})=(\downarrow \text{ELE})}{\text{ADV}},\ (root, a_{\text{g}})\right),\\ \left(\text{ADV}\rightarrow \underset{(\uparrow \text{PRED})=\text{'QUICKLY'}}{\text{quickly}},\ (a_{\text{g}}, \bot)\right)\end{array}\right\}\rightarrow$$
$$\text{V}{:}root{:}\left\{\left(\text{V}\rightarrow \underset{\substack{(\uparrow \text{PRED})=\text{'FALL}\langle(\text{SUBJ})\rangle\text{'}\\ (\uparrow \text{TENSE})=\text{PAST}}}{\text{fell}},\ (root, \bot)\right)\right\}\quad \text{ADVP}{:}root{:}\left\{\begin{array}{l}\left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ADJ})=(\downarrow \text{ELE})}{\text{ADV}}\ \underset{\uparrow=\downarrow}{\text{ADVP}},\ (root, a_{\text{f}}\ root)\right),\\ \left(\text{ADV}\rightarrow \underset{(\uparrow \text{PRED})=\text{'TODAY'}}{\text{today}},\ (a_{\text{f}}, \bot)\right),\\ \left(\text{ADVP}\rightarrow \underset{(\uparrow \text{ADJ})=(\downarrow \text{ELE})}{\text{ADV}},\ (root, a_{\text{g}})\right),\\ \left(\text{ADV}\rightarrow \underset{(\uparrow \text{PRED})=\text{'QUICKLY'}}{\text{quickly}},\ (a_{\text{g}}, \bot)\right)\end{array}\right\}$$

string. For instance, when the instantiated S rule $\left(^{\mathrm{S}\rightarrow}{}_{(\uparrow \mathrm{SUBJ})=\downarrow}{}^{\mathrm{NP}}{}_{\uparrow=\downarrow}{}^{\mathrm{VP}}, (root, (root\ \mathrm{SUBJ})\ root)\right)$ is distributed over the daughters, the derivation will not produce a terminal string because S is not reachable from either NP or VP in the context-free skeleton of the grammar in (13). Similarly, the verbal and adverbial categories are not reachable from NP and NP is not reachable from VP.

We see then that the left daughter of (29) matches the mother of (31) that derives the terminal symbol "John:⊥:∅".

(31) $\mathrm{NP}{:}(root\ \mathrm{SUBJ}){:}\left\{\left(^{\mathrm{NP}\rightarrow}{}_{(\uparrow \mathrm{PRED})=\text{'JOHN'}}{}^{\mathrm{John}}, ((root\ \mathrm{SUBJ}), \bot)\right)\right\} \rightarrow \mathrm{John}{:}\bot{:}\emptyset$

Now, for the right daughter of (29), only the expansion with (30) gives rise to a terminal string. By applying (32) to the left daughter of (30) we first derive the terminal symbol "fell:⊥:∅".

(32) $\mathrm{V}{:}root{:}\left\{\left(^{\mathrm{V}\rightarrow}{}_{\substack{(\uparrow \mathrm{PRED})=\text{'FALL}\langle(\mathrm{SUBJ})\rangle\text{'} \\ (\uparrow \mathrm{TENSE})=\mathrm{PAST}}}{}^{\mathrm{fell}}, (root, \bot)\right)\right\} \rightarrow \mathrm{fell}{:}\bot{:}\emptyset$

Rule (33) then is the only possible rule of G_F whose application (to the right daughter of (30)) will lead to a terminal string.

(33)
$$\mathrm{ADVP}{:}root{:}\left\{\begin{array}{l}\left(^{\mathrm{ADVP}\rightarrow}{}_{(\uparrow \mathrm{ADJ})=(\downarrow \mathrm{ELE})}{}^{\mathrm{ADV}}{}_{\uparrow=\downarrow}{}^{\mathrm{ADVP}}, (root, a_{\mathrm{f}}\ root)\right), \\ \left(^{\mathrm{ADV}\rightarrow}{}_{(\uparrow \mathrm{PRED})=\text{'TODAY'}}{}^{\mathrm{today}}, (a_{\mathrm{f}}, \bot)\right), \\ \left(^{\mathrm{ADVP}\rightarrow}{}_{(\uparrow \mathrm{ADJ})=(\downarrow \mathrm{ELE})}{}^{\mathrm{ADV}}, (root, a_{\mathrm{g}})\right), \\ \left(^{\mathrm{ADV}\rightarrow}{}_{(\uparrow \mathrm{PRED})=\text{'QUICKLY'}}{}^{\mathrm{quickly}}, (a_{\mathrm{g}}, \bot)\right)\end{array}\right\} \rightarrow$$
$$\mathrm{ADV}{:}a_{\mathrm{f}}{:}\left\{\left(^{\mathrm{ADV}\rightarrow}{}_{(\uparrow \mathrm{PRED})=\text{'TODAY'}}{}^{\mathrm{today}}, (a_{\mathrm{f}}, \bot)\right)\right\} \quad \mathrm{ADVP}{:}root{:}\left\{\begin{array}{l}\left(^{\mathrm{ADVP}\rightarrow}{}_{(\uparrow \mathrm{ADJ})=(\downarrow \mathrm{ELE})}{}^{\mathrm{ADV}}, (root, a_{\mathrm{g}})\right), \\ \left(^{\mathrm{ADV}\rightarrow}{}_{(\uparrow \mathrm{PRED})=\text{'QUICKLY'}}{}^{\mathrm{quickly}}, (a_{\mathrm{g}}, \bot)\right)\end{array}\right\}$$

All other legitimate distributions of the instantiated rules of the mother over the daughters also produce categories that fail to derive terminal strings. Some of these distributions will figure in derivations that fail, as we observed earlier, because the LFG rules predicted in the rule component of our categories collectively do not derive a terminal string (ADVP is not reachable from ADV in this particular case). This example illustrates that derivations can also fail to produce any sentence because of mismatches of the term component of an augmented daughter category and the terms instantiating the left-hand categories of the rules in the rule component that expand that daughter category. That is why the alternative rule in G_F in which the adverbial daughter has the triple category $\mathrm{ADV}{:}a_{\mathrm{f}}{:}\left\{\left(^{\mathrm{ADV}\rightarrow}{}_{(\uparrow \mathrm{PRED})=\text{'QUICKLY'}}{}^{\mathrm{quickly}}, (a_{\mathrm{g}}, \bot)\right)\right\}$ does not produce a terminal string. Note moreover that condition (iic) of Definition 16 blocks recursions of ADVP producible by the context-free backbone of the original grammar, because the instantiated recursive ADVP rule $\left(^{\mathrm{ADVP}\rightarrow}{}_{(\uparrow \mathrm{ADJ})=(\downarrow \mathrm{ELE})}{}^{\mathrm{ADV}}{}_{\uparrow=\downarrow}{}^{\mathrm{ADVP}}, (root, a_{\mathrm{f}}\ root)\right)$ cannot be distributed over the daughters.

For the same reasons, (34) is the only useful rule that matches the right daughter of (33).

(34) $$\mathrm{ADVP}{:}root{:}\left\{\begin{array}{l}\left(^{\mathrm{ADVP}\rightarrow}{}_{(\uparrow \mathrm{ADJ})=(\downarrow \mathrm{ELE})}{}^{\mathrm{ADV}}, (root, a_{\mathrm{g}})\right), \\ \left(^{\mathrm{ADV}\rightarrow}{}_{(\uparrow \mathrm{PRED})=\text{'QUICKLY'}}{}^{\mathrm{quickly}}, (a_{\mathrm{g}}, \bot)\right)\end{array}\right\} \rightarrow \mathrm{ADV}{:}a_{\mathrm{g}}{:}\left\{\left(^{\mathrm{ADV}\rightarrow}{}_{(\uparrow \mathrm{PRED})=\text{'QUICKLY'}}{}^{\mathrm{quickly}}, (a_{\mathrm{g}}, \bot)\right)\right\}$$

With rules (35) and (36) then we obtain the terminal string "John:⊥:∅ fell:⊥:∅ today:⊥:∅ quickly:⊥:∅".

(35) $\text{ADV}{:}a_{\mathsf{f}}{:}\left\{\left(^{\text{ADV}\rightarrow}_{(\uparrow \text{PRED}) = \text{'TODAY'}}{}^{\text{today}}, (a_{\mathsf{f}}, \bot)\right)\right\} \rightarrow \text{today}{:}\bot{:}\emptyset$

(36) $\text{ADV}{:}a_{\mathsf{g}}{:}\left\{\left(^{\text{ADV}\rightarrow}_{(\uparrow \text{PRED}) = \text{'QUICKLY'}}{}^{\text{quickly}}, (a_{\mathsf{g}}, \bot)\right)\right\} \rightarrow \text{quickly}{:}\bot{:}\emptyset$

The *Cat* projection of this string is *John fell today quickly*, the only sentence whose derivation G_F simulates by starting with rule (28).

The only other derivation of a string with f-structure F is simulated if we use a rule like (28) except that the ADVP rules are instantiated as in (37), that is, exactly the other way around.

(37) $\left(^{\text{ADVP}\rightarrow}_{(\uparrow \text{ADJ}) = (\downarrow \text{ELE})}{}^{\text{ADV}}{}^{\text{ADVP}}_{\uparrow = \downarrow}, (root, a_{\mathsf{g}}\ root)\right)$

$\left(^{\text{ADVP}\rightarrow}_{(\uparrow \text{ADJ}) = (\downarrow \text{ELE})}{}^{\text{ADV}}, (root, a_{\mathsf{f}})\right)$

If we begin with this alternative starting rule, we can derive the string "John:⊥:∅ fell:⊥:∅ quickly:⊥:∅ today:⊥:∅" with the corresponding sentence *John fell quickly today*.

There are alternative derivations in G_F that also simulate these two LFG derivations, and in that sense the grammar G_F allows for spurious ambiguities. These derivations differ from the given ones in that the instantiating constants of $\mathcal{C}_F$ are biuniquely renamed (e.g., a_{f} by a_{a} and a_{g} by a_{d}) or some of the terminal daughters with no $\downarrow$ in their annotation are biuniquely instantiated by otherwise unused constants of $\mathcal{C}_F$. In the next section we consider some computational strategies for eliminating rules that fail to produce terminal strings or give rise to spurious ambiguities.

4. Computational Considerations

So far we imposed only loose restrictions on the ingredients of the generation grammar G_F, and a faithful implementation of the grammar definition may create categories and rules that are either useless or redundant. Useless rules cannot participate in the simulation of any LFG derivation while redundant ones simulate only the same derivations as other rules and categories in the grammar. There are a number of techniques for avoiding the construction of these unnecessary and undesirable grammar elements.

If the equations in an LFG rule provide alternative definitions for one and the same daughter, a naive implementation would produce distinct but equivalent daughter instantiations. Rule and category instantiations that express only uninformative variation can be eliminated by normalizing the rule annotations in advance of generation so that there is exactly one canonical function-assigning equation for each mother-definable daughter and by using that equation to construct its defining term. Normalization can be accomplished by exploiting symmetry and substitutivity to reduce the annotations of the rules to some normal form according to an appropriate complexity norm, as suggested by Johnson (1988). Another off-line computation can identify terminal daughters that are introduced with rules that do not contain $\downarrow$ and so will never be interpreted. Without loss of generality we can disregard other instantiating constants that might be drawn from $\mathcal{C}_F$ and systematically instantiate all of those terminals with the distinguished constant $\bot$.

We can remove another major source of redundancy by ignoring derivations that differ only by renaming of the instantiating constants of $\mathcal{C}_F$. This can arise if IRD_F

contains rule sets that are identical up to renaming of the instantiating canonical constants, as indicated in Section 3.4. We observed in conjunction with Lemma 3 that the f-description of every derivation for F can be reduced to an equivalent description that is satisfied in the canonical model $\hat{F}$ expanded by some interpretation of *root*. Thus the generation grammar can be constructed by considering only the set $IRD_{\hat{f}}$ containing those elements of IRD_F whose instantiated descriptions are modeled by some *root* expansion of $\hat{F}$.

Even with these refinements, the last example in Section 3.4 illustrates the fact that our recipe for constructing G_F may produce other useless categories and expansion rules. These cannot play a role in any derivation either because they are unreachable from the root symbol S_F or because they do not lead to a terminal string. We can borrow strategies from conventional context-free grammar processing to control the production of these useless items.

A top–down approach to grammar construction is the simplest way of avoiding categories and rules that are unreachable from the root symbol. It corresponds most directly to the specification of Definition 16. The algorithm maintains three data-structures, an agenda $\mathcal{A}$ of categories whose expansion rules have yet to be constructed, a set $\mathcal{V}$ of terminal categories and nonterminal categories that have already been considered for expansion, and a set $\mathcal{R}$ of constructed context-free rules. All three structures are empty at the outset. The first step of the algorithm is to add the root category S_F to $\mathcal{A}$. Then at each subsequent step a category α is selected from $\mathcal{A}$ and moved to $\mathcal{V}$, all rules $\alpha \rightarrow \beta_1..\beta_m$ satisfying conditions (i) (with $IRD_{\hat{f}}$ instead of IRD_F) and (ii) of Definition 16 are added to the rule set $\mathcal{R}$, and each of the nonterminals β_j not already in $\mathcal{V}$ is added to the agenda. Because Definition 16 provides for a finite number of categories, the agenda eventually will become empty. At that point the algorithm terminates with $\mathcal{R}$ containing a subset of R_F sufficient to simulate all and only the LFG derivations for F. As indicated, this algorithm has the desirable property of creating just those categories and rules of G_F that are accessible from the root symbol. It is guided incrementally by the c-structure skeleton of the LFG grammar. It is also guided by properties of the input f-structure as the rule component for each new category is a subset of some element IR_{root} of $IRD_{\hat{f}}$. But this procedure has the disadvantage of typically producing many categories that derive no terminal string.

An alternative strategy is to construct the categories and rules in bottom–up fashion. The bottom–up algorithm uses the same three sets, all empty at the outset. Here the first step is to add to the agenda $\mathcal{A}$ all of the elements in the set T_F of terminal categories. In each subsequent step a category is selected from $\mathcal{A}$ and moved to $\mathcal{V}$, as in the top–down approach. In this case, however, we add to $\mathcal{R}$ all rules $\alpha \rightarrow \beta_1..\beta_m$ that satisfy conditions (i) and (ii) of Definition 16 and where the selected category is at least one of the daughters β_j and all other daughter categories already exist in $\mathcal{V}$. If α is not S_F, we further require α's rule component to be a subset of some IR_{root} so that this process is also constrained at each step by the input f-structure. The category α is added to the agenda if it is not already present in $\mathcal{V}$. This algorithm also terminates when the agenda is empty. It ensures that every category we construct can derive a terminal string, but it does not guarantee that every bottom–up sequence will reach the root symbol.

A more serious shortcoming of both strategies is that they presuppose the prior computation of all elements of $IRD_{\hat{f}}$, but neither specifies how to instantiate those rule sets in an efficient manner. A straightforward modification of the bottom–up algorithm can sidestep this difficulty. We can replace the subset test on the rule component of each α with a check to see whether the instantiated description of that component is satisfied in $\hat{F}$ expanded by some interpretation of *root*. This test makes reference just

to the canonical model of the input, examining only those features that are relevant to each potential new category. We reject a category if it fails this test, knowing that its rule component cannot be a subset of any element of $IRD_{\hat{F}}$. This is similar in spirit to the step-by-step subsumption test of other bottom–up generation algorithms (e.g., Shieber 1988 and Kay 1996). A further restriction is needed to filter the creation of start rules. Rules of the form $S_F \rightarrow S{:}root{:}IR$ are included in $\mathcal{R}$ only when some *root* expansion of $\hat{F}$ is not only a model for $Inst(IR)$ but a minimal one at that. We know in that case that we have arrived at one of the elements of $IRD_{\hat{F}}$. The minimality condition is an analogue of the completeness requirement of other algorithms.

The incremental satisfiability test of this modified algorithm depends on the interpretation of the node constant *root*, and we saw in Section 3.2 that *root* may denote different elements of the universe in different derivations of F. Although its eventual denotation cannot be uniquely predicted at intermediate steps of the bottom–up process, we can avoid reconsideration of *root* denotations already determined to be unsatisfactory by carrying along the satisfying denotations in an auxiliary data structure associated with each category in $\mathcal{A}$ and $\mathcal{V}$. For an LFG rule r that expands A with the c-structure categories $X_1..X_m$, a rule $A{:}t{:}IR \rightarrow X_1{:}t_1{:}IR_1..X_m{:}t_m{:}IR_m$ is only added to $\mathcal{R}$ if there is at least one *root* expansion of $\hat{F}$ that satisfies $Inst(r, (t, t_1..t_m))$ and whose *root* denotation is shared across all daughters. The *root* denotations of all such $\hat{F}$ expansions are then associated with $A{:}t{:}IR$. The complexity of this test is proportional to the complexity of the instantiated description of the LFG rule and not of the instantiated description of the entire rule component IR, because the rule components of the daughter categories do not need to be reevaluated.

For further optimizations we can make use of context-free strategies that take top–down and bottom–up information into account at the same time. For instance, we can simulate a left-corner enumeration of the search space, considering categories that are reachable from a current goal category and match the left corner of a possible rule. As another option, we can precompute a reachability table for the context-free backbone of G and use it as an additional filter on rule construction. In general, almost any of the traditional algorithms for parsing context-free grammars can be reformulated as a strategy for avoiding the creation of useless categories and rules. We can also use enumeration strategies that focus on the characteristics of the input f-structure. A head-driven strategy (cf., e.g., Shieber et al. 1990; van Noord 1993) identifies the lexical heads first, finds the rules that expand to them, and then uses information associated with those heads, such as their grammatical function assignments, to pick other categories to expand.

5. Other Chart-based Approaches

A bottom–up strategy for grammar construction comes closest to the algorithms of previous chart-based generation proposals. There is a correspondence between the edges that are added incrementally to a generation chart and the context-free rules that we add to the grammar. But chart edges in these proposals typically collapse some of the distinctions that we have in our rules and categories, and therefore these algorithms cannot faithfully interpret the full set of grammatical dependencies. For some grammars and inputs they may produce strings that should not belong to the generated language. In an attempt to guarantee termination these algorithms may also include grammar restrictions or processing limits that unduly narrow the set of legitimate results. We will illustrate some correspondences and differences with the modified ($\hat{F}$-guided)

algorithm sketched in the previous section by comparing its first few steps with the operations of Kay's (1996) chart-generation algorithm.

To facilitate the comparison, we have adapted the grammar for one of Kay's examples to an equivalent grammar in the LFG formalism. The LFG grammar is given in (38).

(38) a. $\underset{(\uparrow \text{ ARG1}) = \downarrow}{\text{NP}} \quad \underset{\uparrow = \downarrow}{\text{VP}}$

b. $\underset{\uparrow = \downarrow}{\text{DET}} \quad \underset{\uparrow = \downarrow}{\text{N}}$

c. $\underset{\uparrow = \downarrow}{\text{V}} \quad \underset{(\uparrow \text{ ARG2}) = \downarrow}{\text{NP}}$

d. $\underset{(\uparrow \text{ SPEC}) = \text{DEF}}{\text{the}}$

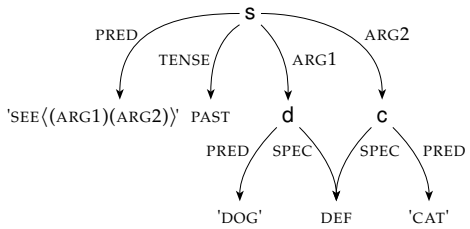
e. $\underset{(\uparrow \text{ PRED}) = \text{'CAT'}}{\text{cat}}$

f. $\underset{(\uparrow \text{ PRED}) = \text{'DOG'}}{\text{dog}}$

g. $\underset{\substack{(\uparrow \text{ PRED}) = \text{'SEE}\langle(\text{ARG1})(\text{ARG2})\rangle\text{'} \\ (\uparrow \text{ TENSE}) = \text{PAST}}}{\text{saw}}$

This grammar with its particular lexical rules has the sentence *The dog saw the cat* in its language, and that sentence is assigned the f-structure in (39), a direct encoding of Kay's semantic specification.[19] This shows the f-structure elements that are used to define the constants in $\mathcal{C}_F$.

(39)



Taking this f-structure as input, the first step of our bottom–up algorithm is to initialize the agenda with the terminal categories $T_F = \{\text{the}{:}\bot{:}\emptyset, \text{dog}{:}\bot{:}\emptyset, ..\}$. Those categories are sufficient to complete the right sides of the given lexical rules, and so in the next steps the terminal categories are moved to $\mathcal{V}$ and rules including those in (40) are constructed. These are the ones that can potentially contribute to the generation of the noun phrase *the dog*: The instantiated descriptions produced with these terms pass our satisfiability test on $\hat{F}$.

(40) a. $\text{DET}{:}a_{\text{d}}{:}\left\{\left(\underset{(\uparrow \text{ SPEC}) = \text{DEF}}{\text{DET}\rightarrow \text{the}}, (a_{\text{d}}, \bot)\right)\right\} \rightarrow \text{the}{:}\bot{:}\emptyset$

b. $\text{N}{:}a_{\text{d}}{:}\left\{\left(\underset{(\uparrow \text{ PRED}) = \text{'DOG'}}{\text{N}\rightarrow \text{dog}}, (a_{\text{d}}, \bot)\right)\right\} \rightarrow \text{dog}{:}\bot{:}\emptyset$

c. $\text{DET}{:}(\textit{root}\ \text{ARG1}){:}\left\{\left(\underset{(\uparrow \text{ SPEC}) = \text{DEF}}{\text{DET}\rightarrow \text{the}}, ((\textit{root}\ \text{ARG1}), \bot)\right)\right\} \rightarrow \text{the}{:}\bot{:}\emptyset$

d. $\text{N}{:}(\textit{root}\ \text{ARG1}){:}\left\{\left(\underset{(\uparrow \text{ PRED}) = \text{'DOG'}}{\text{N}\rightarrow \text{dog}}, ((\textit{root}\ \text{ARG1}), \bot)\right)\right\} \rightarrow \text{dog}{:}\bot{:}\emptyset$

19 Kay provides a flat, unordered collection of separate propositions as input to the generation process, but the difference between a flat and hierarchical arrangement is not material to our discussion. We have translated his constants s, d, c into the elements of our f-structure input, and we have mapped his propositions (dog(d), arg1(s, d)..) into equivalent attribute–value relationships. By the same token, because here we are focusing on the organization of data structures, we note without further comment that his active-passive computational schema is but one way of specializing our general bottom–up algorithm.

Rules (40a,b) correspond directly to the lexical edges that are added to the chart in the initialization step of Kay's algorithm. A lexical edge includes the word (the *Cat* projection of our right-hand complex category), a syntactic category (a left-hand c-structure category) paired with an instantiation term, and instantiated semantic propositions (an instantiated description collected from our rule annotations).[20] The chart edges that parallel the first two rules are shown in (41).

(41)

Words	*Category*	*Semantics*
the	DET:a_{d}	$(a_{\mathrm{d}}\ \textsc{spec}) = \textsc{def}$
dog	N:a_{d}	$(a_{\mathrm{d}}\ \textsc{pred}) = \text{'DOG'}$

Note that the instantiating term that corresponds to Kay's semantic index d is the canonical constant a_{d} drawn from $\mathcal{C}_F$. It is a significant limitation that ground-level terms like these are the only ones available for instantiation. We observed at the beginning of Section 3 that the set $\mathcal{C}_F$ is in general not large enough to equivalently reproduce the discriminations that are required for grammars that allow for undefinable daughters and path equations and for inputs that contain reentrancies. Thus, as originally presented, Kay's algorithm is correct only for a very restricted set of unification grammars.

In contrast, we draw from the larger term set $\mathcal{T}_F$ that includes in addition the collection of path-terms that combine constants with sequences of attributes. Rules (40c,d) make use of the path-term (*root* ARG1), and it is not unreasonable to extend Kay's approach to create the corresponding edges shown in (42). This would allow his algorithm to be applied to a broader set of grammars and inputs.

(42)

the	DET:(*root* ARG1)	(*root* ARG1 SPEC) = DEF
dog	N:(*root* ARG1)	(*root* ARG1 PRED) = 'DOG'

Continuing with the bottom–up strategy, the categories above will be moved from the agenda to $\mathcal{V}$, the rule in (43) will be created from the right-side categories of (40c,d), another NP rule will be created from the constant-instantiated rules in (40a,b), and both new categories will be placed on the agenda.[21]

(43)
$$\text{NP:}(\textit{root}\ \textsc{arg1}):\left\{\begin{array}{l}\left(\text{NP}\rightarrow \underset{\uparrow=\downarrow}{\text{DET}}\ \underset{\uparrow=\downarrow}{\text{N}},\ ((\textit{root}\ \textsc{arg1}),(\textit{root}\ \textsc{arg1})\ (\textit{root}\ \textsc{arg1}))\right),\\ \left(\text{N}\rightarrow \underset{(\uparrow\ \textsc{pred})=\text{'DOG'}}{\text{dog}},\ ((\textit{root}\ \textsc{arg1}),\bot)\right),\\ \left(\text{DET}\rightarrow \underset{(\uparrow\ \textsc{spec})=\textsc{the}'}{\text{the}},\ ((\textit{root}\ \textsc{arg1}),\bot)\right)\end{array}\right\}\rightarrow$$
$$\text{DET:}(\textit{root}\ \textsc{arg1}):\left\{\left(\text{DET}\rightarrow \underset{(\uparrow\ \textsc{spec})=\textsc{the}'}{\text{the}},\ ((\textit{root}\ \textsc{arg1}),\bot)\right)\right\}\ \text{N:}(\textit{root}\ \textsc{arg1}):\left\{\left(\text{N}\rightarrow \underset{(\uparrow\ \textsc{pred})=\text{'DOG'}}{\text{dog}},\ ((\textit{root}\ \textsc{arg1}),\bot)\right)\right\}$$

20 Kay's instantiated semantics corresponds more directly to the third components of the categories of the less sophisticated grammar construction of Kaplan and Wedekind (2000). These instantiated descriptions collapse some of the distinctions of our third-component rules that are not needed for the limited range of dependencies that Kay is considering.

21 The NP based on the rules (40a,b) will not survive into a larger derivation in our framework. This is because all NP daughters are mother-definable in this grammar, and therefore the a_{d} instantiation is not appropriate for the ARG1 daughter of S.

Our extended version of Kay's algorithm also combines determiner and noun edges to make up the NP edges in (44).

(44)

the dog	NP:(*root* ARG1)	(*root* ARG1) = (*root* ARG1), (*root* ARG1 SPEC) = DEF, (*root* ARG1 PRED) = 'DOG'
the dog	NP:a_d	$a_d = a_d$, (a_d SPEC) = DEF, (a_d PRED) = 'DOG'

These edges reveal another significant difference between Kay's algorithm and our approach. The *Words* fields now consist of sequences of words, the (*Cat* projections of the) terminal strings for the full noun phrases. These strings are constructed by concatenating the *Words* from the two component edges in the order specified by the grammar rule that justifies the combination. That is, an edge does not incorporate the justifying rule but instead records a single member of the yield of the subtree beneath the category of the edge. The effect is that the incremental construction of the chart is intermixed with the process of recursively assembling the terminal strings of longer and longer phrases. The advantage of Kay's strategy is that after termination the generated strings can be read out as the *Words* of all the edges whose *Category* is the start category paired with the top-level index and whose *Semantics* exactly matches the original input: There is no need for a separate context-free generation phase.

The disadvantage is that an additional condition must be imposed to guarantee that only a finite number of edges will be created so that the chart-construction process does in fact terminate. Kay proposes a use-once restriction that bounds the size of the derivable constituents by the number of predicates in the input. For some grammars and inputs his algorithm will only produce a proper subset of the full set of generable strings. Another disadvantage in comparison to our approach and other approaches in the chart-based family is that Kay's chart edges do not record intermediate generation results in a compact form that allows operations on the generated string set to be carried out in advance of enumerating the individual strings.[22]

Kay's algorithm is one of a family of chart-based approaches that differ in detail but have similar characteristics at an abstract level. A common thread is that each edge contains a semantic or feature-structure representation aggregated from all of the edges in the subtree that it dominates, and edge creation is filtered by testing whether these representations subsume the generation input. Each algorithm in the family also imposes one or more additional restrictions in an attempt to guarantee termination of the string generation process. Kay appeals to a use-once processing condition, as noted earlier, that ensures termination but may only produce a proper subset of the complete output set.

Shieber's (1988) algorithm and its refinements are closer to our approach in that they do not associate individual terminal strings with the edges of the chart. Each edge contains a semantic or feature-structure representation and a sequence of immediate daughter edges from which that representation can be assembled. The individual substrings consistent with that representation are obtained by a recursive traversal reaching down to the terminal edges. The chart-construction phase of these algorithms (and our grammar construction) will not terminate if the number of distinct edges is not bounded by the size of the input. This may be the case for cyclic inputs, because they have

22 Maxwell (2006) describes a variant of Kay's algorithm that provides a more compact representation for the generated string sets and also deals efficiently with disjunctive input structures.

infinitely many distinct unfoldings all of which subsume the input. A separate question, even with a bounded chart, is whether the string-production traversal is guaranteed to terminate with a finite set of strings. A grammar may give rise to infinitely many strings if it has recursive or iterative rules whose feature structures subsume the same portion of the input. Any finite set of output strings for such a grammar and input will necessarily be incomplete.

Shieber suggests that the end-to-end generation process will terminate and produce a finite but complete set of output strings for a restricted class of semantically monotonic grammars. Shieber's condition requires that the semantic representation of every mother phrase is subsumed by the semantic structure of each of its daughter phrases. In LFG terms this condition amounts to the requirement that each daughter is mother-definable (with an annotation of the form $(\uparrow\ \sigma) = \downarrow$ for $|\sigma| \geq 0$) and, as a consequence, that strings can be generated only for single-rooted inputs. On deeper analysis, however, we see that this restriction is not sufficient to ensure that the generation process will terminate with a finite output set. It does not by itself preclude grammars that assign cyclic feature structures and therefore the chart-construction process may be unbounded. And with an acyclic input and a finite chart the complete set of output strings may still be unbounded since several daughters in a recursive rule may subsume exactly the same portion of the mother's semantic representation. A formal example of this is the monotonic grammar in (8) that produces the string set $\{a^n\ b^n \mid 1 \leq n\}$. A stronger restriction on the form of the annotations, namely, that σ is never empty, will guarantee a finite chart and a finite and complete output set, but monotonic grammars in this sense cannot naturally identify the functional or semantic head-daughters that figure prominently in so many linguistic descriptions. It seems that monotonicity is not a particularly helpful restriction and that some other constraint, either on grammars or processing steps, is needed to guarantee an output set containing only a finite number of syntactic variants (cf., e.g., Neumann 1994; Moore 2002).

If we translate Shieber's and other similar algorithms to our framework, we see that their instantiations need only terms involving *root* and none of the constants in $\mathcal{C}_F$ or the terms containing those constants.[23] This is because these algorithms are not set up to control subsumption accurately for multi-rooted inputs and grammars with mother-undefinable daughters, and in fact their result set may be incorrect in those cases. As we have demonstrated, maintaining all of the proper discriminations requires the larger term set and a mechanism with the same effect as our appropriateness and compatibility conditions.

Comparing other chart-based generation proposals to our bottom–up strategy for creating a generation grammar has brought out some similarities but also highlighted some important differences. Chart edges contain information that summarizes the syntactic and semantic contribution of their subtrees and also allows for the correlated terminal strings to be read out by a straightforward traversal. These algorithms cannot attain correctness, completeness, and termination without imposing limits on the kinds of grammatical dependencies that the generator can faithfully interpret, the range of structures that can be provided as input, or the size and number of output strings that can be produced. Our approach operates correctly on a larger class of grammars and inputs because we have more instantiating terms and therefore are able to maintain

23 Moore (2002) observed that the basic properties of the algorithm do not change if semantically vacuous constituents are allowed. In this case the translation would require the additional term $\perp$ to reduce nodes that are not interpreted in a model of the f-description.

appropriate discriminations without special restrictions. The resulting grammar gives a finite encoding of the complete set of generated outputs in a well-understood formal system. These can be enumerated on demand in our separate context-free generation phase.

6. Cycles

We have established the context-free result only for acyclic f-structures; the result does not hold for cyclic inputs. This is because the f-structures that correspond to subderivations of a derivation of a cyclic structure are not necessarily bounded by the size of the input. So we might need an infinite number of terms in order to reproduce correctly any discrimination made in the f-description for some subderivation of a cyclic input structure. The following example demonstrates that the set of strings that a grammar relates to a particular cyclic input might not be context-free.[24]

Consider the LFG grammar $G = (\{S, A, C\}, \{a, b, c\}, S, R)$ with the annotated rules R given in (45).

(45) a. $S \rightarrow$ A (↑ F) = ↑ (↑ G) = ↓ (↑ G) = ↓ (↑ F) = (↓ F) C

b. $A \rightarrow a$ A (↑ G) = ↓ (↑ F) = (↓ F) b

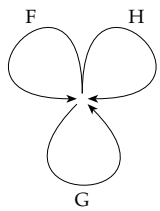
c. $C \rightarrow c$ C (↑ G) = ↓

d. $A \rightarrow a$ b (↑ F) = (↑ H)

e. $C \rightarrow$ c (↑ H F) = (↑ H G)

Now, let F be the following input f-structure.

(46)



The set of terminal strings that are derivable with F is $\{a^n\ b^n\ c^n \mid 1 \leq n\}$, a language that is not context-free. Each top–down derivation for a terminal string that gets assigned the given input f-structure F starts with the S rule. Suppose $a^i\ b^j\ C$ is derived from S by $i - 1$ ($i > 0$) applications of (45b) and one application of (45d). Such a string gets assigned a c-structure and an f-structure of the form depicted in Figure 7 where the C node is mapped to the leftmost G value. The f-structure corresponding to the subderivation

24 Wedekind (2006) provides another example of such a grammar. This runs counter to an assertion in Kaplan and Wedekind (2000) that cyclic structures lie within the scope of our context-free analysis. This was based on reasoning that we now understand to be incorrect.

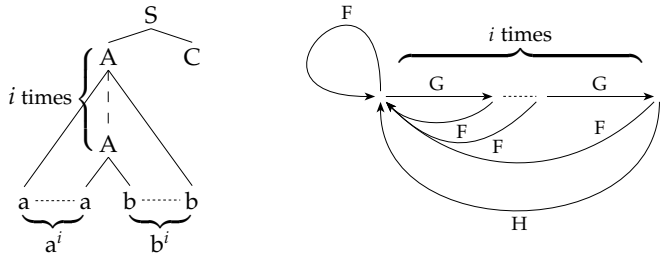


Figure 7
The derivation of $a^i\ b^i\ C$ in grammar (45).

up to this point is arbitrarily larger than the original input, but the rest of the derivation forces the distinguished F, G, and H attributes to collapse into the simple cycles. Because the rightmost G value is the only position where this structure can be folded up to F using the annotations of (45e), (45c) has to be applied exactly $i-1$ times yielding $a^i\ b^i\ c^{i-1}\ C$. With one application of (45e) we obtain F and the sentence $a^i\ b^i\ c^i$. Thus $Gen_G(F) = \{a^n\ b^n\ c^n \mid 1 \leq n\}$.

In general our grammar construction will produce correct outputs for the term set drawn from any finite unfolding of a cyclic input structure, but a complete characterization of the output strings would require an infinite term set. We have not yet investigated the formal properties of the languages that are related to cyclic structures. It is an open research question whether a more expressive system (e.g., indexed grammars or other forms of controlled grammars) can give a finite characterization of the complete string set and whether our context-free grammar construction can be extended to produce such a formal encoding.

7. Other Descriptive Devices

We have shown that the context-free grammar of Definition 16 produces the strings in $Gen_G(F)$ for an LFG grammar G that characterizes f-structures by means of equality and function application, the most primitive descriptive devices of the LFG formalism. In this section we extend the grammar-construction procedure so that it produces context-free generation grammars that simulate the other formal devices that were originally proposed by Kaplan and Bresnan (1982).[25]

Completeness and Coherence. The result holds trivially when we also take into account LFG's devices for enforcing the subcategorization requirements of individual predicates, the completeness and coherence conditions. Both conditions are concerned with the semantic-form PREDicate values that consist of a predicate and a list of governable grammatical functions, as for example, 'FALL⟨(SUBJ)⟩' with the list ⟨(SUBJ)⟩ and 'JOHN' with the empty list. An f-structure is **complete** if each substructure (including the entire structure) that contains a PRED also contains all governable grammatical functions its semantic form subcategorizes for. And an f-structure is **coherent** if all its governable functions are subcategorized by a local semantic form. If an input f-structure F is not complete and coherent, the LFG derivation relation Δ_G does not associate it with any

25 Because LFG theory has evolved away from the original c-structural encoding of long-distance dependencies, we will not consider it here. In Wedekind and Kaplan (forthcoming) we describe the construction for grammars that use functional uncertainty, the device that superseded the initial mechanism for characterizing long-distance dependencies.

strings, and the set $Gen_G(F)$ is empty. Thus, when we determine by inspection that an input f-structure fails to satisfy these conditions, we maintain the context-free result by assigning it a trivial grammar that generates the empty context-free language.

C-Structure Regular Predicates and Disjunctive Functional Constraints. The construction in Section 3.3 produces context-free generation grammars for LFG grammars whose c-structure rules are of an elementary form: Their right-hand sides consist of concatenated sequences of annotated categories, and the equations in the annotation sets are interpreted as simple conjunctions of f-structure requirements. The full LFG notation is more expressive, allowing functional requirements to be stated as arbitrary Boolean combinations of basic assertions. It also allows the right-hand sides of c-structure rules to denote arbitrary regular languages over annotated categories. Rules with the richer notation can be normalized to rules of the necessary elementary form by simple transformations. First, in the regular right-side of each rule every category X with a Boolean combination of primitive annotations is replaced by a disjunction of X's each associated with one of the alternatives of the disjunctive normal form of the original annotation. Then the augmented regular right-sides are converted to a collection of right-linear rewriting rules by systematically introducing new nonterminals and their expansions, as described by Chomsky (1959) (see also Hopcroft and Ullman 1979). The new nonterminals are annotated with $\uparrow = \downarrow$ equations as needed to ensure that f-structure requirements are properly maintained. The result of these transformations is a set of productions all of which are in conventional context-free format and have no internal disjunctions and which together define the same string/f-structure mapping as a grammar encoded in the original, linguistically more expressive, notation.

Constraining Statements and Negation. The statements in an LFG f-description are divided into two classes: defining and constraining statements. The constraining statements are evaluated once all defining statements have been processed and a minimal model (of the defining statements) has been constructed. The constraining devices introduced by Kaplan and Bresnan (1982) are constraining equations and inequalities, and existential and negative existential constraints. If a constraining statement is contained in an f-description FD, it is evaluated against a minimal model M of the defining statements of FD in the obvious way: $M \models t =_c t'$ iff $M \models t = t'$ (constraining equation), $M \models t$ iff $\exists t'(M \models t = t')$ (existential constraint), $M \models \neg\gamma$ iff $M \not\models \gamma$ (negation of a constraining or defining statement).

We can extend our grammar construction to descriptions with constraining statements by adjusting the definition of IRD_F. We modify condition (ii) of Definition 15 so that $M|\Sigma \cong F$ for a minimal model M of just the defining statements of $Inst(IR)$ and additionally require $M \models \gamma$ for all constraints γ of $Inst(IR)$. Then a context-free grammar based on this revised definition will properly reflect the defining/constraining distinction.

The proof of this depends on one further technicality, however. Recall that the constructions that we used in the proof of our main theorem yield in both proof directions $FD[\psi] = Inst(IR_{root})$. As a consequence, the constraining statements in $Inst(IR_{root})$ are exactly the ones that result from those in FD by substitution with ψ. Suppose that M and M_{root} are minimal models of the defining part of FD and $Inst(IR_{root})$, respectively. In order to establish also that M satisfies all constraints in FD iff M_{root} satisfies the ones contained in $Inst(IR_{root})$, it is sufficient to show that $M \models t = t'$ iff $M_{root} \models t[\psi] = t'[\psi]$ holds for all denoting terms. This follows (with M' as M_{root}) from the isomorphic mapping of term denotations provided by Lemma 2′, a slightly stronger version of Lemma 2.

Lemma 2′
Let c and ρ *be a derivation with f-description FD and f-structure F in G. If* ψ *is a reducing substitution for c and* ρ *and* $M = (\mathcal{U}, I)$ *and* $M' = (\mathcal{U}', I')$ *are minimal models of the defining parts of FD and FD*[ψ], *respectively, then there is an isomorphism h between* $M|\Sigma$ *and* $M'|\Sigma$ *such that* $h(I(t)) = I'(t[\psi])$ *for each interpreted term t or* $t[\psi]$.[26]

Membership Statements. Membership statements are formulas of the form $t' \in t$. Membership in LFG is interpreted just as a binary relation between functional elements, and a model satisfies a membership statement $t' \in t$ iff the membership relation holds between the denotation of t' and the denotation of t. Membership statements may introduce daughters that are undefinable in terms of their mother and therefore may be instantiated by $\mathcal{C}_F$ constants as we illustrated earlier in our treatment of the $(\uparrow \text{ADJ}) = (\downarrow \text{ELE})$ annotation. Then, if we expand the isomorphism-based determination of the equivalence of feature structures and feature descriptions in the usual way to sets and set descriptions, membership statements can be handled by our original construction without further modification.[27]

Semantic Form Instantiation. As described earlier, semantic forms are the single-quoted values of PRED attributes in terms of which the completeness and coherence conditions are defined. They are also instantiated, in the sense that for each occurrence of a semantic form in a derivation a new and distinct indexed form is chosen. Because of this special property, semantic forms occurring in annotated rules may be regarded as metavariables that are substituted by the instantiation procedure similar to the familiar ↑ and ↓ symbols. The distinguishing indices on semantic forms are usually only displayed in a graphical representation of an f-structure if this is necessary for clarity, but distinctively indexed semantic forms are always available for appropriately instantiating the LFG rules, just like the other constants that we draw from the input structure. We can extend the mechanism for controlling the correct instantiation of undefinable daughters to ensure that the semantic forms of all simulated derivations are correctly instantiated. As part of an appropriate instantiation of an LFG rule we also substitute for the prototypical semantic forms in the rule distinct indexed forms, drawn from F, and we expand the compatibility condition to this larger set of instantiations.

8. Consequences and Observations

We have shown that a given LFG grammar can be specialized to a context-free grammar that characterizes all and only the strings that correspond to a given (acyclic) f-structure. We can now understand different aspects of generation as pertaining either to the way the specialized grammar G_F is constructed or to well-known properties of context-free grammars and context-free generation.

It follows as an immediate corollary, for example, that it is decidable whether the set $Gen_G(F)$ is empty, contains a finite number of strings, or contains an infinite number of strings. This can be determined by inspecting G_F with standard context-free tools, once

26 The proof requires an elaboration of the argument used in the proof of Lemma 2. Following the inductive construction of that proof, it is easy to see that $I(t) = I_i(t[\psi_i])$ holds for all terms t and $t[\psi_i]$ that are interpreted in $M = (\mathcal{U}, I)$ or $M_i = (\mathcal{U}, I_i)$. Because there must be an isomorphism h between $M_{|\mathcal{N}_c|}$ and any other minimal model $M' = (\mathcal{U}', I')$ of the defining part of $FD[\psi]$, $h(I_{|\mathcal{N}_c|}(t[\psi])) = I'(t[\psi])$ and thus $h(I(t)) = I'(t[\psi])$ for each interpreted term t or $t[\psi]$.

27 Rounds (1988) proposes a bisimulation-based characterization of sets and set membership. This would require a more sophisticated analysis, but it is more of mathematical than linguistic interest.

it has been constructed. If the language is infinite, we can make use of the context-free pumping lemma to identify a finite number of short strings from which all other strings can be produced by repetition of subderivations. Wedekind (1995) first established the decidability of LFG generation and proved a pumping lemma for the generated string set; our theorem provides alternative and very direct proofs of these previously known results.

We also have an explanation for another observation of Wedekind (1995). Kaplan and Bresnan (1982) showed that the Nonbranching Dominance Condition (sometimes called Off-line Parsability) is a sufficient condition to guarantee decidability of the membership problem. Wedekind noted, however, that this condition is not necessary to determine whether a given f-structure corresponds to any strings. We now see more clearly why this is the case: If there is a context-free derivation for a given string that involves a nonbranching dominance cycle, we know that there is another derivation for that same string that has no such cycle. Thus, the generated language is the same whether or not derivations with nonbranching dominance cycles are allowed.

There are practical consequences to the two phases of LFG generation. The grammar G_F can be provided to a client as a finite representation of the set of perhaps infinitely many strings that correspond to the given f-structure, and the client can then control the process of enumerating individual strings. The client may choose to produce the shortest ones just by avoiding recursive category expansions. Or the client may apply an n-gram model (Langkilde 2000), a stochastic context-free grammar model (Cahill and van Genabith 2006) or a more sophisticated statistical language model trained on a collection of derivations to identify the most probable derivation and thus the presumably most fluent sentence from the set of possibilities (Velldal and Oepen 2006; de Kok, Plank, and van Noord 2011; Zarrieß, Cahill, and Kuhn 2011).

We have assumed in our construction that terminals are morphologically unanalyzed, full-form words. A more modular arrangement is to factor morphological generalizations into a separate formal specification with less expressive power than LFG rules can provide, namely, a regular relation (Karttunen, Kaplan, and Zaenen 1992; Kaplan and Kay 1994). The analysis of a sentence then consists of mapping the string of words into a string of morphemes to which the LFG grammar is then applied. The full relation between strings of words and associated f-structures is then the composition of the regular morphology with an LFG language over morpheme strings. To generate with such a combined system, we can produce the context-free morpheme strings corresponding to the input f-structure, and then pass those results through the morphology. Because the class of context-free languages is closed under composition with regular relations and regular relations are closed under inversion, the resulting set of word strings will remain context-free.

Our proof also depends on the assumption that the input F is fully specified so that the set of possible instantiations is finite. Dymetman (1991), van Noord (1993), and Wedekind (1999, 2006) have shown that it is in general undecidable whether or not there are any strings associated with a structure that is an arbitrary extension of the f-structure provided as the input. Indeed, our proof of context-freeness does not go through if we allow new elements to be hypothesized arbitrarily, beyond the ones that appear in F; if this is permitted, we cannot establish a finite bound on the number of possible categories. This is unfortunate, because there may be interesting practical situations in which it is convenient to leave unspecified the value of a particular feature. If we know in advance that there can be only a finite number of possible values for an underspecified feature, however, the context-free result can still be established. We create from F a set of alternative structures $\{F_1, .., F_n\}$ by filling in all possible values of

the unspecified features, and for each of them we produce the corresponding context-free grammar. Because a finite union of context-free languages is context-free, the set of strings generated from any of these structures must again remain in that class. Of course, this is not a particularly efficient technique: It introduces and propagates features that the grammar may never actually interrogate, and it needlessly repeats the construction of common subgrammars that do not make reference to the alternative feature specifications. The amount of computation may be reduced by adapting methods from the parsing literature that operate on conjunctive equivalents of disjunctive feature constraints (e.g., Karttunen 1984; Maxwell and Kaplan 1991).

Our theorem helps us to understand better the problem of ambiguity-preserving generation. We showed previously that the problem is undecidable in the general case (Wedekind and Kaplan 1996). But our generation result does enable us to make that decision under certain recognizable circumstances, namely, if the intersection of the sentence sets assigned to the different f-structures is computable. This is true if the sentences belong to some formally restricted subsets of the context-free languages, for example, finite sets or regular languages; this is the unstated presupposition of Knight and Langkilde's (2000) parse-forest technique. For a set of f-structures $\{F_1, .., F_n\}$ we construct the context-free grammars G_{F_i} and inspect them with standard context-free tools to determine whether $L(G_{F_i})$ belongs to an intersectable subclass ($i = 1, .., n$). If each of them meets this condition, we can compute the intersection $\bigcap_{i=1}^{n} L(G_{F_i})$ to find any sentences that are derived ambiguously with f-structures $F_1, .., F_n$.

We have shown in this article that the context-free property also holds for other descriptive devices as originally proposed by Kaplan and Bresnan (1982). In Wedekind and Kaplan (forthcoming) we broaden the grammar-construction procedure so that it produces context-free generation grammars that simulate the more sophisticated mechanisms that were introduced and adopted into later versions of the LFG formalism. Among these are devices for the f-structure characterization of long-distance dependencies and coordination: functional uncertainty (Kaplan and Maxwell 1988a; Kaplan and Zaenen 1989), set distribution for coordination, and the interaction of uncertainty and set distribution (Kaplan and Maxwell 1988b). We also extend to devices whose evaluation depends on properties of the c-structure to f-structure correspondence, namely, functional categories and extended heads (Zaenen and Kaplan 1995; Kaplan and Maxwell 1996) and functional precedence (Bresnan 1995; Zaenen and Kaplan 1995). Of course, the context-free result trivially holds for purely abbreviatory notations such as templates, lexical rules, and complex categories (Butt et al. 1996; Kaplan and Maxwell 1996; Dalrymple, Kaplan, and Holloway King 2004; Crouch et al. 2008); these clearly help in expressing linguistic generalizations but can be formally treated in the obvious way by translating their occurrences into the more basic descriptions that they abbreviate. In contrast, the restriction operator (Kaplan and Wedekind 1993) requires more careful consideration. Restriction can cause the functional information associated with intermediate c-structure nodes not to be included in the f-structures of higher nodes. This is formally quite tractable if the restricted information is provided to the generator as a separately rooted f-structure. Otherwise, the f-structure input is essentially underspecified, and thus, as discussed earlier, a context-free generation grammar can be produced just in case restriction can eliminate only a finite amount of information (see also Wedekind 2006).

A final comment concerns the generation problem for other high-order grammatical formalisms. The PATR formalism also augments a context-free backbone with a set of feature-structure constraints, but it differs from LFG in that its metavariables allow

constraints on one daughter to refer directly to sister feature structures that may not be mother-definable. It is relatively straightforward to extend our lemmas and theorem so that they apply to a more general notion of definability that encompasses sisters as well as mothers. We can thus establish the context-free result for a broader family of formalisms that share the property of being endowed with a context-free base. On the other hand, it is not clear whether the string set corresponding to an underlying Head-driven Phrase Structure Grammar (HPSG) feature structure is context-free. HPSG (Pollard and Sag 1994) does not make direct use of a context-free skeleton, and operations other than concatenation may be used to assemble a collection of substrings into an entire sentence. We cannot extend our proof to HPSG unless the effect of these mechanisms can be reduced to an equivalent characterization with a context-free base. Grammars written for the ALE system's logic of typed feature structures (Carpenter and Penn 1994), however, do have a context-free component and therefore are amenable to the treatment we have outlined.

In sum, this article offers a new way to conceptualize the generation problem for LFG and other higher-order grammatical formalisms with context-free backbones. Distinguishing the grammar-specialization phase from a string-enumeration phase provides a mathematical framework for understanding the formal properties of the generated string sets. It also provides a framework for analyzing and understanding the computational behavior of existing approaches to generation. Existing algorithms operate properly on restricted grammars and inputs and thus only approximate a complete solution to the problem. They typically implement particular techniques for optimizing the size of the search space and bounding the amount of computation required by the generation process. Our formulation can allow a larger and perhaps more attractive set of candidates to be safely considered, and it also makes available a collection of familiar tools that may suggest new ways of improving algorithmic performance.

From a more general perspective, there has been no deep tradition for the formal analysis of higher-order generation akin to the richness of our mathematical and computational understanding of parsing. The approach outlined in this article, we hope, will serve as a major step in redressing that imbalance.

Acknowledgments

We are indebted to John Maxwell for many fruitful and insightful discussions of the LFG generation problem. Hadar Shemtov, Martin Kay, and Paula Newman have also offered criticisms and suggestions that have helped to clarify many of the mathematical and computational issues. We also thank the anonymous reviewers for their valuable comments on an earlier draft. This work was carried out while R. M. K. was at the Palo Alto Research Center and at Microsoft Corporation.

References

Bar-Hillel, Yehoshua, Micha A. Perles, and Eliahu Shamir. 1961. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft, und Kommunikationsforschung*, 14:143–172.

Billot, Sylvie and Bernard Lang. 1989. The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 143–151, Vancouver.

Bresnan, Joan. 1995. Linear order, syntactic rank, and empty categories: On weak crossover. In Mary Dalrymple, Ronald M. Kaplan, John T. Maxwell III, and Annie Zaenen, editors, *Formal Issues in Lexical-Functional Grammar*, CSLI Publications, Stanford, CA, pages 241–274.

Bresnan, Joan. 2000. Optimal syntax. In Joost Dekkers, Frank van der Leeuw, and Jeroen van de Weijer, editors, *Optimality Theory: Phonology, Syntax and Acquisition*. Oxford University Press, Oxford, pages 334–385.

Butt, Miriam, Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer. 2002. The parallel grammar project. In *Proceedings of the Workshop on*

Grammar Engineering and Evaluation, pages 1–7, Taipei.
Butt, Miriam, Tracy Holloway King, Maria-Eugenia Niño, and Frédérique Segond. 1996. *A Grammar Writer's Cookbook*. CSLI Publications, Stanford, CA.
Cahill, Aoife and Josef van Genabith. 2006. Robust PCFG-based generation using automatically acquired LFG approximations. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 1033–1040, Sydney.
Carpenter, Bob and Gerald Penn. 1994. ALE 2.0 user's guide. Technical report, Carnegie Mellon University, Pittsburgh, PA.
Carroll, John A., Ann Copestake, Dan Flickinger, and Victor Poznański. 1999. An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the 7th European Workshop on Natural Language Generation*, pages 86-95, Toulouse.
Carroll, John A. and Stephan Oepen. 2005. High efficiency realization for a wide-coverage unification grammar. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing*, pages 165–176, Jeju Island.
Chomsky, Noam. 1959. On certain formal properties of grammars. *Information and Control*, 2:137–167.
Crouch, Richard, Mary Dalrymple, Ronald M. Kaplan, Tracy Holloway King, John T. Maxwell III, and Paula Newman. 2008. XLE documentation. Technical report, Palo Alto Research Center, Palo Alto, CA. Available at `http://www2.parc.com/isl/groups/nltt/xle/doc/xle_toc.html`.
Dalrymple, Mary, Ronald M. Kaplan, and Tracy Holloway King. 2004. Linguistic generalizations over descriptions. In *Proceedings of the International Lexical-Functional Grammar Conference 2004*, pages 199–208, Stanford, CA.
de Kok, Daniël, Barbara Plank, and Gertjan van Noord. 2011. Reversible stochastic attribute–value grammars. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 194–199, Portland, OR.
de Kok, Daniël and Gertjan van Noord. 2010. A sentence generator for Dutch. In Eline Westerhout, Thomas Markus, and Paola Monachesi, editors, *Computational Linguistics in the Netherlands 2010: Selected Papers from the 20th CLIN Meeting*. Netherlands Graduate School of Linguistics, Utrecht, pages 75–90.
Dipper, Stefanie. 2003. *Implementing and Documenting Large-scale Grammars–German LFG*. Ph.D. thesis, University of Stuttgart.
Dymetman, Marc. 1991. Inherently reversible grammars, logic programming and computability. In *Proceedings of the ACL Workshop: Reversible Grammar in Natural Language Processing*, pages 20–30, Berkeley, CA.
Dymetman, Marc. 1997. Charts, interaction-free grammars, and the compact representation of ambiguity. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1002–1009, Nagoya.
Hopcroft, John E. and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.
Johnson, Mark. 1988. *Attribute–Value Logic and the Theory of Grammar*. CSLI Publications, Stanford, CA.
Johnson, Mark and Stefan Riezler. 2002. Statistical models of syntax learning and use. *Cognitive Science*, 26(3):239–253.
Kaplan, Ronald M. 1995. The formal architecture of Lexical-Functional Grammar. In Mary Dalrymple, Ronald M. Kaplan, John T. Maxwell III, and Annie Zaenen, editors, *Formal Issues in Lexical-Functional Grammar*, Stanford, CA, pages 7–27.
Kaplan, Ronald M. and Joan Bresnan. 1982. Lexical-Functional Grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, MA, pages 173–281.
Kaplan, Ronald M. and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.
Kaplan, Ronald M. and John T. Maxwell III. 1988a. An algorithm for functional uncertainty. In *Proceedings of the 12th International Conference on Computational Linguistics*, pages 297–302, Budapest.
Kaplan, Ronald M. and John T. Maxwell III. 1988b. Constituent coordination in Lexical-Functional Grammar. In *Proceedings of the 12th International Conference on Computational Linguistics*, pages 303–305, Budapest.

Kaplan, Ronald M. and John T. Maxwell III. 1996. LFG grammar writer's workbench. Technical report, Xerox Palo Alto Research Center, Palo Alto, CA. Available at ftp://ftp.parc.xerox.com/pub/lfg/lfgmanual.pdf.

Kaplan, Ronald M., Stefan Riezler, Tracy Holloway King, John T. Maxwell III, Alexander Vasserman, and Richard Crouch. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of the Human Language Technology Conference and the 4th Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 97–104, Boston, MA.

Kaplan, Ronald M. and Jürgen Wedekind. 1993. Restriction and correspondence-based translation. In *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics*, pages 193–202, Utrecht.

Kaplan, Ronald M. and Jürgen Wedekind. 2000. LFG generation produces context-free languages. In *Proceedings of the 18th International Conference on Computational Linguistics*, pages 425–431, Saarbrücken.

Kaplan, Ronald M. and Annie Zaenen. 1989. Long-distance dependencies, constituent structure, and functional uncertainty. In Mark Baltin and Anthony Kroch, editors, *Alternative Conceptions of Phrase Structure*. Chicago University Press, Chicago, IL, pages 17–42.

Karttunen, Lauri. 1984. Features and values. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd Annual Meeting of the Association for Computational Linguistics*, pages 28–33, Stanford, CA.

Karttunen, Lauri, Ronald M. Kaplan, and Annie Zaenen. 1992. Two-level morphology with composition. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 141–148, Nantes.

Kay, Martin. 1996. Chart generation. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 200–204, Santa Cruz, CA.

Knight, Kevin and Irene Langkilde. 2000. Preserving ambiguities in generation via automata intersection. In Henry A. Kautz and Bruce W. Porter, editors, *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence*, pages 697–702, Austin, TX.

Kuhn, Jonas. 2001. *Formal and Computational Aspects of Optimality-Theoretic Syntax*. Ph.D. thesis, University of Stuttgart.

Kuhn, Jonas. 2002. OT syntax: Decidability of generation-based optimization. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 48–55, Philadelphia, PA.

Kuhn, Jonas. 2003. *Optimality-Theoretic Syntax: a Declarative Approach*. CSLI Publications, Stanford, CA.

Lang, Bernard. 1994. Recognition can be harder than parsing. *Computational Intelligence*, 10(4):486–494.

Langkilde, Irene. 2000. Forest-based statistical sentence generation. In *Proceedings of the 6th Applied Natural Language Processing Conference*, pages 170–177, Seattle, WA.

Langkilde, Irene and Kevin Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*, pages 704–710, Montreal.

Maxwell III, John T. 2006. Efficient generation from packed input. In Miriam Butt, Mary Dalrymple, and Tracy Holloway King, editors, *Intelligent Linguistics Architectures: Variations on Themes by Ronald M. Kaplan*. CSLI Publications, Stanford, CA, pages 19–34.

Maxwell III, John T. and Ronald M. Kaplan. 1991. A method for disjunctive constraint satisfaction. In Masaru Tomita, editor, *Current Issues in Parsing Technology*. Kluwer, Dordrecht, pages 173–190.

Moore, Robert C. 2002. A complete, efficient sentence-realization algorithm for unification grammar. In *Proceedings of the 2nd International Natural Language Generation Conference*, pages 41–48, New York, NY.

Neumann, Günter. 1994. *A Uniform Computational Model for Natural Language Parsing and Generation*. Ph.D. thesis, Saarland University.

Neumann, Günter. 1998. Interleaving natural language parsing and generation through uniform processing. *Artificial Intelligence*, 99:121–163.

Pollard, Carl and Ivan Sag. 1994. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, Chicago, IL.

Rohrer, Christian and Martin Forst. 2006. Improving coverage and parsing quality of a large-scale LFG for German. In *Proceedings of the 5th Conference on Language Resources and Evaluation (LREC-2006)*, pages 2206–2211, Genoa.

Rounds, William C. 1988. Set values for unification-based grammar formalisms and logic programming. Research report CSLI-88-129, CSLI, Stanford University, Stanford, CA.

Shemtov, Hadar. 1997. *Ambiguity Management in Natural Language Generation*. Ph.D. thesis, Stanford University.

Shieber, Stuart M. 1988. A uniform architecture for parsing and generation. In *Proceedings of the 12th International Conference on Computational Linguistics*, pages 614–619, Budapest.

Shieber, Stuart M., Hans Uszkoreit, Fernando C. N. Pereira, Jane Robinson, and Mabry Tyson. 1983. The formalism and implementation of PATR-II. In Barbara J. Grosz and Mark E. Stickel, editors, *Research on Interactive Acquisition and Use of Knowledge*. SRI Final Report 1894, SRI International, Menlo Park, CA, pages 39–79.

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

Statman, Richard. 1977. Herbrand's theorem and Gentzen's notion of a direct proof. In Jon Barwise, editor, *Handbook of Mathematical Logic*. North-Holland, Amsterdam, pages 897–912.

van Noord, Gertjan. 1993. *Reversibility in Natural Language Processing*. Ph.D. thesis, Utrecht University.

Velldal, Erik and Stephan Oepen. 2006. Statistical ranking in tactical generation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 517–525, Sydney.

Wedekind, Jürgen. 1994. Some remarks on the logic of unification grammars. In Christopher J. Rupp, Michael Rosner, and Roderick Johnson, editors, *Constraints, Language and Computation*. Academic Press, London, pages 29–76.

Wedekind, Jürgen. 1995. Some remarks on the decidability of the generation problem in LFG- and PATR-style unification grammars. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, pages 45–52, Dublin.

Wedekind, Jürgen. 1999. Semantic-driven generation with LFG- and PATR-style grammars. *Computational Linguistics*, 25(2):277–281.

Wedekind, Jürgen. 2006. On some formal properties of LFG generation. In Miriam Butt, Mary Dalrymple, and Tracy Holloway King, editors, *Intelligent Linguistic Architectures: Variations on Themes by Ronald M. Kaplan*. CSLI Publications, Stanford, CA, pages 53–72.

Wedekind, Jürgen and Ronald M. Kaplan. 1996. Ambiguity-preserving generation with LFG- and PATR-style grammars. *Computational Linguistics*, 22(4):555–558.

Wedekind, Jürgen and Ronald M. Kaplan. Forthcoming. LFG generation with rich descriptive devices. Working paper, University of Copenhagen and Nuance Communications, Inc.

White, Michael. 2006. CCG chart realization from disjunctive inputs. In *Proceedings of the 4th International Natural Language Generation Conference*, pages 12–19, Sydney.

Zaenen, Annie and Ronald M. Kaplan. 1995. Formal devices for linguistic generalizations: West Germanic word order in LFG. In Mary Dalrymple, Ronald M. Kaplan, John T. Maxwell III, and Annie Zaenen, editors, *Formal Issues in Lexical-Functional Grammar*, CSLI Publications, Stanford, CA, pages 215–239.

Zarrieß, Sina, Aoife Cahill, and Jonas Kuhn. 2011. Underspecifying and predicting voice for surface realisation ranking. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1007–1017, Portland, OR.