Chapter 23

Formal and computational properties of LFG

Ronald M. Kaplan
Stanford University

Jürgen Wedekind
University of Copenhagen

This chapter first reviews the basic architectural concepts that underlie the formal theory of Lexical-Functional Grammar. The LFG formalism provides a simple set of devices for describing the common properties of all human languages and the particular properties of individual languages. It postulates two levels of syntactic representation for a sentence, a constituent structure and a functional structure. These are related by a piecewise correspondence that permits the abstract functional structure to be described in terms of configurations of constituent structure phrases. We then survey the mathematical and computational properties of this simple framework. We demonstrate that the recognition/parsing, realization/generation, emptiness, and other more specific decision problems are unsolvable for grammars in the unrestricted LFG formalism. A first set of restrictions guarantees decidability of recognition, realization, and other problems for grammars that are still suitable for linguistic description, but the solutions to these problems in the worst case are computationally impractical. The class of LFG grammars that meet an additional set of restrictions is equivalent to the class of mildly context-sensitive grammars, and the recognition and realization problems for grammars in this class are thus not only decidable but tractable as well.

Ronald M. Kaplan and Jürgen Wedekind. Formal and computational properties of LFG. To appear in: Dalrymple, Mary (ed.). Handbook of Lexical Functional Grammar. Berlin: Language Science Press. [preliminary page numbering]

1 Introduction

The basic features of the LFG formalism are quite simple and have remained remarkably stable since they were first introduced by Kaplan & Bresnan (1982).* An LFG grammar assigns to each sentence in its language at least one constituent structure (c-structure) and at least one functional structure (f-structure). The c-structure is a phrase-structure tree that represents the order of words and their grouping into phrases. The f-structure is a hierarchical attribute-value matrix that represents the underlying grammatical relations that are expressed by configurations of c-structure nodes. The c-structure is determined in the traditional way by the rules of a context-free grammar. The f-structure is a minimal model for the functional description (f-description) that is constructed from annotations associated with the categories of rules that license the nodes of the c-structure. The f-description is obtained by instantiating those annotations on the assumption that there is a piece-wise correspondence ϕ between the nodes of the c-structure and the units of a satisfying f-structure.

This simple correspondence architecture still lies at the core of LFG theory even as it has been extended and refined to provide more insightful accounts of long distance dependencies (Kaplan & Zaenen 1989), coordination (Kaplan & Maxwell 1988), and other syntactic phenomena. In this chapter we focus on the mathematical and computational properties of the basic formalism. As is well known, its expressive power goes far beyond the capabilities of the context-free c-structure grammar. This is because the annotations may associate information that originates from different (and possibly arbitrarily distant) nodes with the same f-structure unit. The result is that such a unit must satisfy requirements that come from words in the string or nodes in the tree that do not stand in a local mother-daughter relationship. A string with an otherwise well-formed c-structure is excluded from the language if such context-sensitive f-structure requirements are inconsistent. We know that some degree of context sensitivity is needed for recognizing and parsing natural languages (Bresnan et al. 1982; Culy 1985; Shieber 1985), but the basic LFG formalism may allow for more expressive power than is actually required.

Indeed, Kaplan & Bresnan (1982) used a reduction from the Turing machine halting problem to show that the recognition/parsing problem is undecidable for

*Jürgen Wedekind passed away just as work on this chapter was coming to an end. Jürgen was a master of the LFG formalism, with deep insights into its mathematical and computational properties and how they relate to important principles of linguistic analysis. His early passing is a great loss to the LFG community. He will also be missed as a close friend and collaborator. RMK

unrestricted LFG grammars (see also Johnson 1988). This is the computationally important problem of determining whether or not a given string belongs to the language of the grammar and is assigned at least one c-structure and corresponding f-structure. Wedekind (2014) proved the undecidability of the realization problem, also of practical significance. This is the problem of determining whether the language contains at least one string to which an arbitrary given f-structure is assigned. Wedekind's undecidability proof used a reduction from the emptiness problem for the intersection of context-free languages. He also used that reduction to show the undecidability of the emptiness problem for unrestricted LFGs (Wedekind 1999). This is the problem of determining whether or not there are any strings at all in the language of a given LFG grammar. The emptiness problem for LFGs was previously shown to be undecidable by reductions from Hilbert's Tenth Problem (Roach 1983) and Post's Correspondence Problem (Nishino 1991).

We revisit these undecidability results in Section 4. We provide alternative proofs within a single, conceptually simple, framework. In Appendix A we use this framework to show that other more specific decision problems are also unsolvable.

We consider in Section 5 some formal conditions that are sufficient to guarantee decidability of the recognition and realization problems. Kaplan & Bresnan (1982) showed that recognition is decidable if c-structures with non-branching dominance (NBD) chains and/or unlimited empty nodes are excluded, and they argued that this is a reasonable restriction for LFG grammars that describe natural languages. This parsing-oriented limitation does not reduce the complexity of generation (Wedekind 2014), but an unrelated restriction has been shown to ensure the decidability of that problem (Wedekind & Kaplan 2012). This raises the question whether there is a single, linguistically plausible, condition that applies indifferently to both parsing and generation. We introduce in Section 5 such a uniform condition, proper anchoring, but we also demonstrate that this particular condition is not strong enough to guarantee that these problems can be solved with practical efficiency. In the worst case recognition and generation may take an amount of time that is exponential in the length of an input sentence or f-structure.

This leads us to examine in Section 7 a stronger set of restrictions that not only guarantee decidability of recognition and realization as well as emptiness but also ensure that those problems can be solved in polynomial time. This follows from the fact that LFG grammars that meet these additional restrictions are mildly context-sensitive in their expressive power and thus also have the known mathematical and computational properties of that class of formal grammars.

2 Basic LFG formalism

We show in Figure 1 the c-structure and f-structure that the annotated c-structure rules in (1) and lexical entries in (2) would assign to the sentence *He sees the girl*.

(1) S → NP (↑ SUBJ)=↓ VP ↑=↓ (↑ TENSE)

NP → (Det) ↑=↓ N ↑=↓

VP → V ↑=↓ NP (↑ OBJ)=↓

(2)

he	N	(↑ PRED)='PRO' (↑ AGR PERS)=3 (↑ AGR NUM)=SG
sees	V	(↑ PRED)='SEE⟨SUBJ OBJ⟩' (↑ TENSE)=PRES (↑ SUBJ AGR PERS)=3 (↑ SUBJ AGR NUM)=SG
the	Det	(↑ SPEC)=DEF
girl	N	(↑ PRED)='GIRL' (↑ AGR PERS)=3 (↑ AGR NUM)=SG (↑ SPEC)

The correspondence function ϕ is indicated by the arrows between the c-structure nodes and the f-structure units and also, redundantly, by the columns of node identifiers *root*, $n_1, n_2, \ldots$ attached to the f-structure units. We see even in this simple example that the function ϕ is typically many-to-one (heads and coheads of grammatical constituents are mapped into the same f-structure) but is not onto (the AGR/agreement f-structure units are not the image of any node). The function ϕ may also be partial, if nodes necessary for c-structure well-formedness have no f-structure significance.

The phrasal categories of this c-structure obviously meet the node admissibility conditions of the annotated rewriting rules (1). Lexical entries are interpreted also as annotated rewriting rules that relate the lexical categories of the c-structure to the words of the sentence. The entry for *the*, for example, is interpreted as the rule

(3) Det → the (↑ SPEC)=DEF

and the normal node admissibility conditions also license the proper lexical expansions for the tree.

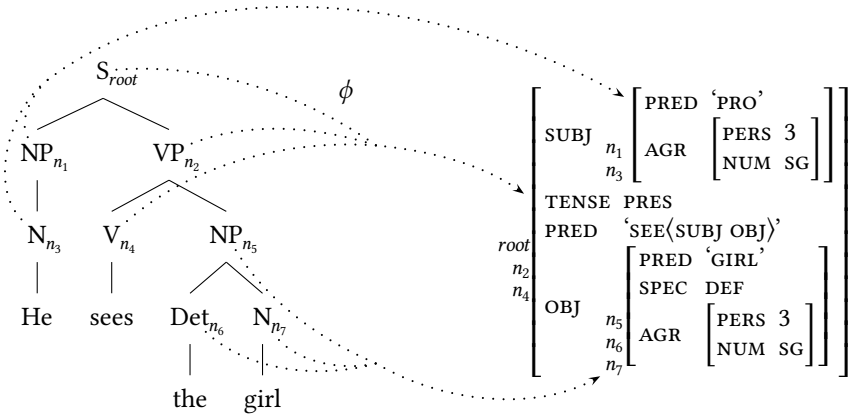


Figure 1: Illustration of the basic LFG architecture: A c-structure c and f-structure f related by the correspondence function ϕ from the nodes of c to the units of f. The f-structure units are indexed by the nodes to which they correspond.

The description that the f-structure must satisfy is constructed from the annotations associated with the daughter categories of the rules that license particular nodes in the c-structure. Each side of an equation designates an element of a corresponding f-structure, and the equation is satisfied if both sides designate the same element. The metavariable ↓ in an annotation designator instantiates to the f-structure corresponding to the node that matches the associated rule category (n_1 for ↓ in the annotation (↑ SUBJ)=↓ attached to the NP in the S rule), and the metavariable ↑ denotes the f-structure corresponding to the mother of that node (the node *root* for that rule). To be precise, if $*$ instantiates to the matching node and $M(*)$ instantiates to its mother, then ↓ and ↑ are abbreviations for $\phi(*)$ and $\phi(M(*))$ respectively. The metavariable instantiations are easy to read from the *annotated c-structure* in Figure 2. This is a phrase-structure tree whose nodes are labeled with the category-annotation pairs that appear in grammar rules and lexical entries.

The first NP is identified as n_1 and its mother is *root*, so the annotation (↑ SUBJ)=↓ instantiates directly to $(\phi(root)\,\text{SUBJ})=\phi(n_1)$. Since a parenthesized designator denotes the element reached by traversing a path of attributes from a starting f-structure, the f-structure in Figure 1 satisfies this equation because $\phi(n_1)$ is the SUBJ of $\phi(root)$ under the illustrated ϕ correspondence. The full f-description for this annotated c-structure is the conjunction of instantiated equations collected from all of its nodes, shown in (4).

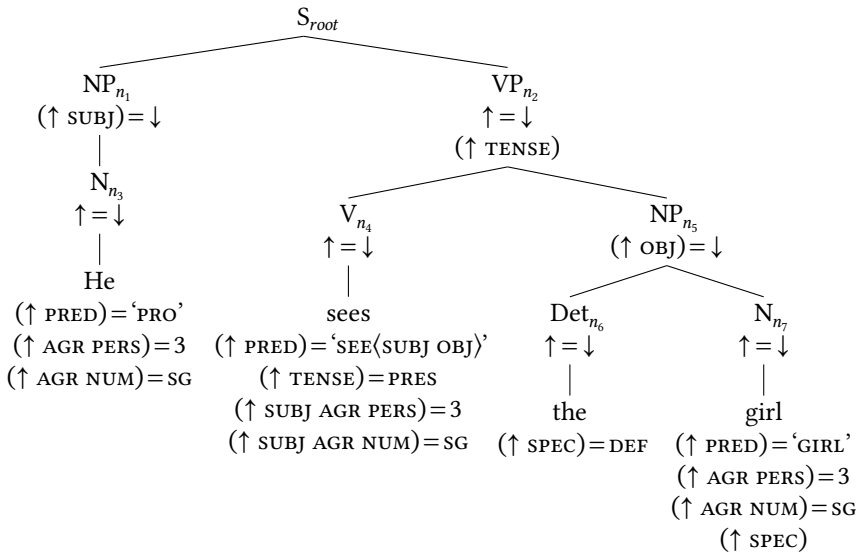


Figure 2: Annotated c-structure for *He sees the girl* with the rules in (1) and lexicon in (2).

(4)
$(\phi(root)\ \text{SUBJ})=\phi(n_1)$
$\phi(root)=\phi(n_2)$
$(\phi(root)\ \text{TENSE})$
$\phi(n_1)=\phi(n_3)$
$(\phi(n_3)\ \text{PRED})=\text{'PRO'}$
$(\phi(n_3)\ \text{AGR PERS})=3$
$(\phi(n_3)\ \text{AGR NUM})=\text{SG}$
$\phi(n_2)=\phi(n_4)$
$(\phi(n_4)\ \text{PRED})=\text{'SEE}\langle\text{SUBJ OBJ}\rangle\text{'}$
$(\phi(n_4)\ \text{PRED})=\text{PRES}$
$(\phi(n_4)\ \text{SUBJ AGR PERS})=3$
$(\phi(n_4)\ \text{SUBJ AGR NUM})=\text{SG}$
$(\phi(n_2)\ \text{OBJ})=\phi(n_5)$
$\phi(n_5)=\phi(n_6)$
$\phi(n_5)=\phi(n_7)$
$(\phi(n_6)\ \text{SPEC})=\text{DEF}$
$(\phi(n_7)\ \text{PRED})=\text{'GIRL'}$
$(\phi(n_7)\ \text{AGR PERS})=3$
$(\phi(n_7)\ \text{AGR NUM})=\text{SG}$
$(\phi(n_7)\ \text{SPEC})$

We can test each equation separately to verify that the f-structure in Figure 1 meets all the specifications in (4). The equation $(\phi(n_4)\ \text{SUBJ AGR NUM})=\text{SG}$ is satisfied, for example, because ϕ maps n_4 to the outermost f-structure, and that f-structure has a path from SUBJ through AGR to NUM, ending in the atomic value SG. That value is consistent with the requirement that the equation $(\phi(n_3)\ \text{AGR NUM})=\text{SG}$ imposes on the f-structure of n_3. In contrast, this grammar would assign no f-structure to the string *They sees the girl* because the f-description for its c-structure would require its subject f-structure to have inconsistent values for AGR NUM, a violation of the *Uniqueness Condition* of Kaplan &

Bresnan (1982). The correspondence ϕ and the instantiated metavariables ensure that the properties of the subject NP are consistent with the verb's agreement specification even though they do not appear together in a local mother-daughter configuration.

The f-structure in this configuration also meets the additional well-formedness conditions of LFG theory. We see that it is a *minimal* model of the f-description in the sense that at least one equation or combination of equations will no longer be satisfied if any attribute or value is removed (for example $(\phi(n_6)\ \text{SPEC}) = \text{DEF}$ fails without the SPEC feature of the OBJ).[1] Conversely, a structure with any features beyond those already present, say if the SUBJ is extended with TENSE PAST, is not minimal, because the f-description is still satisfied when that feature is removed. The minimal model is unique[2] for a given annotated c-structure and contains all and only the linguistically relevant features that are expressed by the words of a sentence.

The minimal model is important in LFG theory for another reason. It is the basis for the distinction between *defining annotations* and *constraining annotations*. The defining annotations are the simple equalities between two designators whose instantiations determine the attributes and values of the minimal model. That f-structure must then also satisfy the instantiations in the f-description of any constraining annotations. The grammar in (1) contains two constraining annotations, the positive existential constraints $(\uparrow\ \text{TENSE})$ and $(\uparrow\ \text{SPEC})$. The instantiation $(\phi(\textit{root})\ \text{TENSE})$ is satisfied because the conjunction of defining equations in the f-description specify a particular value (PRES) for the attribute TENSE in the f-structure corresponding to the S node. This constraint excludes strings whose main verb is a participle instead of a tensed form (e.g. **He seeing the girl*) without depending on participles setting up a uniqueness clash by also adding a TENSE feature with an otherwise unnecessary and uninformative value (e.g. NONE). Similarly the instantiation $(\phi(n_7)\ \text{SPEC})$ excludes singular common nouns that have no specifier (e.g. **He sees girl*). The formalism also allows for con-

[1] Strictly speaking, a minimal model of the f-description includes not only the attributes and values of the f-structure but also the association of those elements with the nodes of the c-structure as instantiated via the ϕ correspondence, as depicted in Figure 1. Technically, what we usually regard as the f-structure is the restriction of such a model to just those attributes and values.

[2] As a notational convenience, the LFG formalism allows for primitive annotations to be embedded in disjunctive formulas that then might have several solutions. There is an obvious transformation of the grammar that converts disjunctions of annotations within a rule to an equivalent set of alternative rules with annotations that are no longer disjunctive. The minimal models are unique for the annotated c-structures assigned by the rules of such a transformed grammar.

straints that test the minimal model for the absence of a feature, e.g. ¬(↑ OBJ); for the presence or absence of a specific attribute value, e.g. (↑ VOICE)=$_c$ PASSIVE or (↑ VOICE)≠ ACTIVE; for the identity of two f-structures, e.g. (↑ SUBJ)=$_c$(↑ OBJ); and for any value other than a specific one, e.g. (↑ SUBJ NUM)≠ SG. Constraining annotations help to avoid clutter in the f-structure by assigning syntactic significance to the presence or absence of unmarked or default features and also by capturing the difference between constituents that provide values for features and constituents that check those values. See Kaplan (2019) for a fuller discussion of underspecified values in LFG.

The quoted values of the PRED attributes in Figure 1 carry the subcategorization restrictions of the predicates they represent, and they characterize the essential interaction between syntax and semantics while staying agnostic about the details of any particular underlying semantic theory. The semantic form 'SEE⟨SUBJ OBJ⟩' contains a list of grammatical-function designators that the predicate subcategorizes for. The *Completeness Condition* requires that all listed functions appear locally in the minimal f-structure, and the *Coherence Condition* precludes the local appearance of any governable functions (COMP, OBL, XCOMP...) not included in the list. The semantic form also indicates that SEE is the semantic relation and, by virtue of their order in the list, that SUBJ and OBJ respectively map to the first and second arguments of that relation. Semantic forms do not require special treatment in our formal analysis because they can be interpreted as succinct abbreviations for collections of other annotations. Thus the positive and negative constraints (5a-b) express the subcategorization requirements of 'SEE⟨SUBJ OBJ⟩'. The semantic relation and the mapping of functions to arguments can be coded with distinguished attributes REL, ARG1, ARG2 as in (5c-d).[3]

(5) a. Completeness: (↑ SUBJ) (↑ OBJ)
b. Coherence: ¬(↑ COMP) ¬(↑ OBL) ¬(↑ XCOMP) ...
c. Semantic relation: (↑ REL)=SEE
d. Argument mapping: (↑ ARG1)=(↑ SUBJ)
(↑ ARG2)=(↑ OBJ)
e. Instantiation: (↑ PRED SOURCE)=*

[3]Halvorsen & Kaplan (1988) introduced a separate semantic projection, σ, as an alternative to distinguished attributes in formulating these essential properties of the syntax-semantics interface. In that more explicit arrangement σ would be a qualifier on the (5c-d) designators. In Glue Semantics they are elaborated in collections of linear logic premises (Dalrymple et al. 1993; Dalrymple 1999).

Semantic forms are instantiated in LFG theory to mark the difference between syntactically-implied semantic coreference (as in constructions of functional control) and unrelated repetitions of similar expressions. Equation (5e) records as the value of the additional distinguished attribute SOURCE the daughter node at which a particular PRED is introduced. This makes each occurrence unique, and also supports a precedence order that can be used in regulating long distance dependencies (see Kaplan & Zaenen forthcoming [this volume]).

3 Technical preliminaries

In preparation for the mathematical analysis in the following sections we now introduce more precise specifications of the LFG derivation machinery.

The annotated c-structure is often described as the result of a special derivation process for an LFG grammar G that treats categories and annotations separately. But it is helpful for formal reasoning to regard it as a normal derivation of the *annotated c-structure grammar* for G, an ordinary context-free grammar with a systematically modified set of rules. Suppose $X{:}A$ is an annotated category in the right side of a rule in the traditional LFG grammar format. Then for every rule expanding X the annotated grammar contains a version in which the left side is also decorated with those particular annotations. For example, because NP in (1) is annotated in S with the SUBJ assignment and in VP with the OBJ assignment, the NP rule is replaced by the rules in (6).

(6)
$$\begin{array}{llll} \text{NP} & \rightarrow \text{(Det)} & \text{N} \\ (\uparrow \textsc{subj}) = \downarrow & \uparrow = \downarrow & \uparrow = \downarrow \end{array}$$

$$\begin{array}{llll} \text{NP} & \rightarrow \text{(Det)} & \text{N} \\ (\uparrow \textsc{obj}) = \downarrow & \uparrow = \downarrow & \uparrow = \downarrow \end{array}$$

With this reformulation the normal category matching of context-free derivations allows us to make direct use of all established properties (decidability, closure, pumping) of context-free grammars and their derivations. The traditional LFG c-structure in Figure 1 is obviously just the annotation-free projection of the annotated c-structure in Figure 2.

For every annotated c-structure there is an instantiated f-description that defines a function ϕ mapping its nodes to their corresponding minimal-model f-structure units, if the f-description is satisfiable. There is also a function *Yield* that maps its nodes to the substrings of the sentence that they dominate. The set of G's derivations is then characterized by the relation Δ_G defined in (7).

(7) $\Delta_G(s, c, f)$ iff c is an annotated c-structure of G, s is the terminal string of c, and f is the minimal model for the satisfiable f-description instantiated from c.

Note that an annotated c-structure c uniquely determines both the string s and f-structure f in a derivation triple. Moreover, without further stipulation we know that the length of the string $|s|$ and the number of units $|f|$ in the f-structure are both bounded by (functions of) $|c|$, the number of nodes in the c-structure. That is, there are grammar-dependent functions $\overleftarrow{b}_G$ and $\vec{b}_G$ such that

(8) For all $(s, c, f) \in \Delta_G, |s| \leq \overleftarrow{b}_G(|c|)$ and $|f| \leq \vec{b}_G(|c|)$.

The function $\overleftarrow{b}_G$ depends on the number of daughters in the longest c-structure rule and $\vec{b}_G$ depends on the most complicated annotated category.

The language, f-structure, parsing, and generating projections of Δ_G are defined in (9).

(9) $L(G) = \{s \mid \Delta_G(s, c, f)$ for some c and $f\}$ = the language of G
$F(G) = \{f \mid \Delta_G(s, c, f)$ for some s and $c\}$ = the f-structures of G
$Par_G(s) = \{f \mid \Delta_G(s, c, f)$ for some $c\} \subseteq F(G)$
$Gen_G(f) = \{s \mid \Delta_G(s, c, f)$ for some $c\} \subseteq L(G)$

A parser for an LFG grammar G provides for any given string s the set of f-structures (if any) that are related to it by the grammar, and a generator provides all the strings that the grammar relates to a given f-structure (if any).

These projections allow for succinct statements of the emptiness, recognition, and realization decision problems (10).

(10) Emptiness: is $L(G)$ empty? (equivalently, are $F(G)$ or Δ_G empty?)
Recognition: for any string s is $Par_G(s)$ empty?
Realization: for any f-structure f is $Gen_G(f)$ empty?

We show in the next section that the emptiness, recognition, and realization problems are all undecidable for unrestricted LFG grammars. This implies immediately that the parsing and generation are also unsolvable. Our demonstrations involve simple phrase-structure rules with elementary defining annotations as exemplified in (11).

(11)

$(\uparrow/\downarrow$ SUBJ NUM$)=$SG	assign an atomic value
$(\uparrow$ SUBJ$)=\downarrow$	assign a function to a daughter f-structure
$(\downarrow$ OBJ$)=(\uparrow$ SUBJ$)$	daughter-mother control
$(\uparrow$ XCOMP SUBJ$)=(\uparrow$ SUBJ$)$	traditional functional control

Annotations of these types are not exceptional, they are commonly found in linguistic grammars.

4 Undecidable problems

A standard method for showing that a formal problem of interest is undecidable is the reduction technique. A problem P is said to be *reducible* to problem P' if for any instance of P an instance of P' can be constructed such that solving the instance of P' will solve the instance of P as well. Thus, if P reduces to P' and P is undecidable, then P' must also be undecidable. As noted, this general strategy has been applied with reductions from different source problems (Turing machine halting, Hilbert's Tenth, Post Correspondence, emptiness of context-free intersection) to address the LFG emptiness, recognition, and realization problems. Here we present a single reduction-source framework, based on the emptiness problem of context-free intersection, that recapitulates these previous results.

4.1 The emptiness problem

The emptiness problem for context-free intersection is the problem of determining whether or not the languages generated by two given context-free grammars G_1 and G_2 have an empty intersection ($L(G_1) \cap L(G_2) = \emptyset$). This problem is known to be undecidable. The reduction of this emptiness problem to questions of the LFG formalism depends on the ability to construct for every context-free grammar G an LFG grammar whose f-structures contain encodings of all and only the strings of $L(G)$. We show in (12) one way in which a string pqr can be encoded in the attributes and values of an f-structure, as a H(ead)-T(ail) list representation.

(12)
$$\begin{bmatrix} \textsc{h} & \textsc{p} & \\ \textsc{t} & \begin{bmatrix} \textsc{h} & \textsc{q} & \\ \textsc{t} & \begin{bmatrix} \textsc{h} & \textsc{r} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Without loss of generality, let G be an arbitrary context-free grammar in Chomsky Normal Form, that is, a context-free grammar with only binary branching rules of the form $A \rightarrow B\ C$ for nonterminal expansions and unary rules $A \rightarrow a$ for terminals. The schematic rules in (13) provide a template for an LFG grammar *String*(G) that creates head-tail encodings (12) for the strings of $L(G)$.

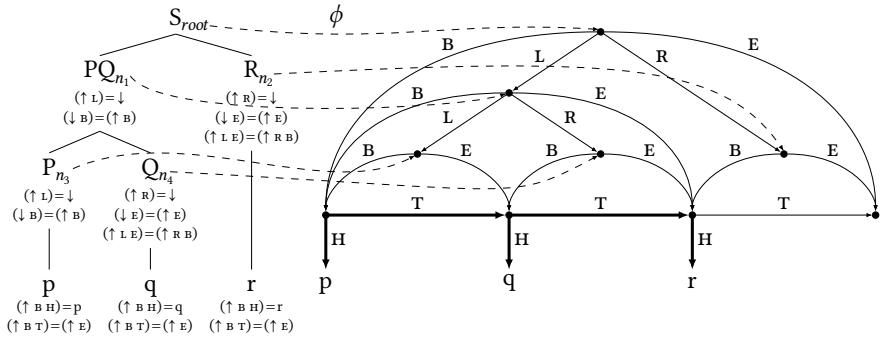


Figure 3: An annotated c-structure and f-structure derived with head-tail string encoding rules of the form in (13). Thick lines show the string encoding, thin lines show the construction scaffolding. The ϕ correspondence is depicted with dashed lines.

(13) a. $A \rightarrow$ B ($\uparrow$ L)=$\downarrow$ ($\downarrow$ B)=($\uparrow$ B) C ($\uparrow$ R)=$\downarrow$ ($\downarrow$ E)=($\uparrow$ E) ($\uparrow$ L E)=($\uparrow$ R B)

b. $A \rightarrow$ a ($\uparrow$ B H)=a ($\uparrow$ B T)=($\uparrow$ E)

The annotations on the binary rules (13a) transmit the string encodings from their daughter f-structures to their mother f-structure. The attributes L(eft) and R(ight) are the scaffolding needed to concatenate the encodings from the daughters by linking the end of the left-daughter encoding to the beginning of the right. Rules of the form (13b) create for each terminal the one-element head-tail encoding of their right side, with B and E attributes marking its beginning and end. Control equations such as ($\downarrow$ B)=($\uparrow$ B) and ($\uparrow$ B T)=($\uparrow$ E) are the essential ingredient in this and other string-encoding formulations: Crucially, they allow terminal-string information to propagate transparently through all intermediate nodes to the f-structure of the root. Figure 3 shows the annotated c-structure and a graphical f-structure representation for a derivation containing a head-tail encoding of a single string.

Now suppose that G_1 and G_2 are arbitrary context-free grammars in Chomsky Normal Form and assume without loss of generality that their nonterminals are disjoint and that the strings of each language end with a marker # distinct from all other terminals. We construct a new LFG grammar G by combining the rules of *String*(G_1) and *String*(G_2) with root categories S_1 and S_2 respectively and introducing a new root category S with start rule (14).

(14) S $\rightarrow$ S_1 ($\downarrow$ B)=($\uparrow$ B) S_2 ($\downarrow$ B)=($\uparrow$ B)

By construction of the string grammars and the B annotations of the start rule, only the string encodings of the two derived f-structures can interact. Because the string encodings are compatible only if the derived strings are identical, the LFG language $L(G)$ contains all and only strings ss for $s \in L(G_1) \cap L(G_2)$. The emptiness of context-free intersection is undecidable so the question whether $L(G)$ is empty must also be undecidable.

4.2 The recognition problem

We prove that the LFG recognition problem is undecidable by exhibiting one particular string that belongs to $L(G)$ only if $L(G_1) \cap L(G_2) \neq \emptyset$. We modify the string grammars for G_1 and G_2 by treating each terminal a other than # as a nonterminal category and adding for each of them a trivial rule

(15) $a \rightarrow \epsilon$

The effect is that only the string # belongs to the language of each of the modified string grammars, but that single string is assigned all and only the f-structures that respectively encode the original context-free languages. Again with the starting rule (14) the concatenation ## belongs to the language of the modified grammar if and only if $L(G_1) \cap L(G_2) \neq \emptyset$, that is, if and only if $Par_G(\#\#)$ is not empty.

Empty nodes are disfavored in some modern versions of LFG, particularly when long-distance dependencies are characterized by functional uncertainty rather than traces (Kaplan & Zaenen 1989; Dalrymple et al. 2015). But the undecidability of recognition can also be demonstrated with grammars $String(G_1)$ and $String(G_2)$ redefined so as to produce the same head-tail string encodings from nonbranching dominance chains without the benefit of empty nodes.

For each binary rule $A \rightarrow B\ C$ the string encoding grammars will now contain a nonbranching rule of the form (16a). This immediately derives only the left daughter B but pushes the right-daughter category C on a simulated stack for expansion lower in the derivation. Since B is the left daughter of A, the encodings of their terminal strings have a shared B(eginning).

(16) a. $A \rightarrow$ B ($\downarrow$ STK CAT)=C ($\downarrow$ STK NXT)=($\uparrow$ STK) ($\downarrow$ B)=($\uparrow$ B)

b. $A \rightarrow$ C ($\uparrow$ STK CAT)=C ($\uparrow$ STK NXT)=($\downarrow$ STK) ($\uparrow$ B T)=($\downarrow$ B) ($\uparrow$ B H)=a

c. $A \rightarrow$ # ($\uparrow$ B H)=#

Corresponding to each terminal rule $A \rightarrow a$, for $a \neq \#$, there is a collection of rules of the form (16b), one for each right-daughter category C whose expansion may

have been deferred until it reemerges at the top of the stack. The annotations pop that category from the stack while adding the terminal a to the front of the head-tail encoding of the terminal string under C. Finally, for each unary rule $A \rightarrow \#$ the string grammar contains a rule of the form (16c) to terminate the NBD derivations and install # as the final item of every string encoding. Because # is the distinguished end-of-string marker, these preterminals never appear as left daughters of binary rules and are thus always the last categories to be removed from the stack. As before, if NBD string grammars $String(G_1)$ and $String(G_2)$ are combined into an LFG grammar G with rule (14), then $Par_G(\#\#) \neq \emptyset$ if and only if $L(G_1) \cap L(G_2) \neq \emptyset$.

The complexity of recognition arises from the fact that, in order to assign f-structures to the strings of infinite languages, annotated c-structure grammars must include rules for recursive subderivations (rule sequences that derive a node labeled with an annotated category A from an A-labeled dominating node), and such recursive subderivations must be allowed to stack one above another. The string grammars in our undecidability proofs show that recursive subderivations can assign to a single string (#) a set of f-structures each encoding one of the strings of an infinite context-free language. Unlike the f-structures that correspond to the sentences of natural languages, those f-structures are determined only by the annotations on nonterminal categories without regard to any lexical information carried by the input string or even its length (there is no function of $|s|$ that bounds the sizes of c and f in all derivation triples).

4.3 The realization problem

We turn now to the realization problem. Also using a reduction from the emptiness of context-free intersection, Wedekind (2014) proved that realization is undecidable for unrestricted LFG grammars if there are cyclic paths in the input f-structure.[4] Whereas the emptiness and recognition demonstrations are based on head-tail string encodings, Wedekind's proof is formulated in terms of an alternative way of encoding the strings of a language, a descending chain of attributes as illustrated in (17).

(17) $\begin{bmatrix} \text{B} & [\text{p} & [\text{q} & [\text{r} & [\,]]]] \\ \text{E} & \multicolumn{4}{c}{\text{---------}} \end{bmatrix}$

[4] Wedekind & Kaplan (2012) established that the realization problem is decidable if the input f-structure f contains no cycles. For an acyclic f-structure the string-set $Gen_G(f)$ can be described by a context-free grammar, and the emptiness problem for context-free grammars is decidable.

The beginning of the encoding for string pqr is still accessible as the value of the B attribute, but now the end is identified by the reentrant value of the top-level E attribute. Grammars that encode context-free languages in this way are created by replacing the annotations on the terminal rules (13b) with functional control annotations as in (18).

(18) $A \rightarrow \underset{(\uparrow \text{B}\ a)=(\uparrow \text{E})}{a}$

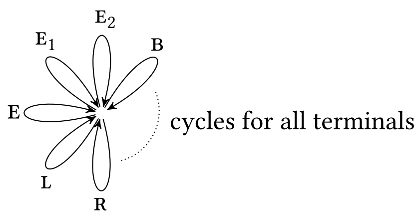
The scaffolding illustrated in Figure 3 is unchanged but the H attributes at the bottom are removed and the sequence of T attributes is replaced by the sequence of terminal-attributes.

The essence of Wedekind's (2014) proof is then captured by combining attribute-chain string-encoding grammars for arbitrary context-free grammars G_1 and G_2 into an LFG grammar G with start rule (19).

(19)
$$\text{S} \rightarrow \begin{array}{ccc} \text{S}_1 & \text{S}_2 & \# \\ (\uparrow \text{L})=\downarrow & (\uparrow \text{R})=\downarrow & (\uparrow \text{E}_1\ \text{E}_1)=\uparrow \\ (\downarrow \text{B})=(\uparrow \text{B}) & (\downarrow \text{B})=(\uparrow \text{B}) & \bigwedge_{x \text{ an attribute}} (\uparrow \text{E}_2\ \text{E}_1)=(\uparrow \text{E}_2\ \text{E}_1\ x) \\ (\downarrow \text{E})=(\uparrow \text{E}_1) & (\downarrow \text{E})=(\uparrow \text{E}_2) & \end{array}$$

In the absence of atomic values there can be no atom-value clashes to exclude mismatching combinations, and the language $L(G)$ therefore contains all strings $s_1 s_2\#$ for $s_1 \in L(G_1)$ and $s_2 \in L(G_2)$. However, strings belonging to the intersection of $L(G_1)$ and $L(G_2)$ are distinguished by the fact that the end points E_1 and E_2 of their descending attribute-chain encodings are the same. In that case $(\phi(root)\ \text{E}_1)=(\phi(root)\ \text{E}_2)$ and the annotations on the terminal # entail by simple substitutions that $(\phi(root)\ x)=\phi(root)$ for all attributes x. Thus all and only strings $ss\#$ for $s \in L(G_1) \cap L(G_2)$ receive the one-element cyclic f-structure f in (20).

(20)



The realization problem is undecidable because $Gen_G(f) \neq \emptyset$ if and only if $L(G_1) \cap L(G_2) \neq \emptyset$. This shares with the recognition proof the property that infinitely many annotated c-structures of arbitrary size may have to be inspected

to determine whether there is at least one that is related to a single input of a fixed size (a cyclic f-structure in this case).[5]

The undecidability results we have demonstrated here, together with other simple reductions from the emptiness problem of context-free intersection, can be used to show that other properties of the unrestricted LFG formalism are also undecidable. The following is a partial list of these undecidable questions.

(21) a. Generation from underspecified f-structures: Is there a sentence that realizes an f-structure with more features than a given one? (Wedekind 1999)

b. Ambiguity-preserving generation: Is there a single string that realizes two different f-structures? (Wedekind & Kaplan 1996)

c. Finite versus infinite ambiguity: Is any string in the language infinitely ambiguous? (Jaeger et al. 2005)

d. Ranking in Optimality-theoretic LFG: Can an optimal derivation always be identified? (Kuhn 2003)

e. Economy of Expression: Can the smallest c-structure for a given f-structure be identified?[6]

Appendix A includes simple proofs showing that a number of more specific questions are also undecidable. Additional restrictions are clearly necessary to provide a linguistic formalism that is mathematically manageable.

5 Conservation and decidability

The recognition and realization problems are undecidable for unrestricted LFG grammars because there is no finite number of (size-bounded) annotated c-structures whose inspection is sufficient to determine whether there is a valid derivation for a given input string/f-structure. As a consequence, there is no systematic relationship between the length of a string and the sizes of its f-structure parses or the size of an f-structure and the lengths of its generated strings. Moreover, a grammar can assign infinitely many f-structures to a single string and

[5] Cyclic f-structures have been proposed in the analysis of complex adjunction and coordination constructions (Zweigenbaum 1988; Fang & Sells 2007; Haug & Nikitina 2012; Przepiórkowski & Patejuk 2012) and thus cannot be excluded from the LFG formalism. More to the point, example (52b) in the Appendix A shows that it is undecidable whether an arbitrary LFG grammar produces cyclic f-structures.

[6] This follows from the fact that realization is undecidable in the general case (as just sketched): if it cannot be decided whether there are any c-structures at all for an f-structure input, then the smallest such structure cannot be determined.

infinitely many strings to a single f-structure. These properties seem implausible for language as a medium of communication.

From a broader perspective, these excesses can be cast in terms of the "grammatical mapping problem", the problem of characterizing in an explanatory and computable way the relation Γ between the sentences of a language and representations of their meanings (presumably logical formulas that can be interpreted in a representation of the world) (Kaplan & Bresnan 1982; Kaplan 1987). If s is a sentence of a language and m represents one of its meanings (that is, $(s, m) \in \Gamma$), then pretheoretically we expect the derivational machinery that translates between s and m to be information-conserving in the following sense.

(22) *Principle of Conservation*
For all $(s, m) \in \Gamma$, $|m|$ is bounded by $|s|$ and $|s|$ is bounded by $|m|$.

The size of the meaning representation can be defined in any reasonable way. The crucial claim is that the derivational machinery does not by itself add or subtract, in either direction, arbitrary amounts of information. The additional linguistically appealing property of bidirectional finite ambiguity follows as an immediate corollary.

(23) *Finite Ambiguity*
If Γ is conservative, then each sentence expresses only a finite number of meanings and each meaning is expressed by only a finite number of sentences.

In LFG-based approaches the grammatical mapping Γ is typically conceptualized as the composition of the grammar-defined syntactic derivations Δ_G and the semantic derivations Σ that map primarily between syntactic f-structures and corresponding representations of meaning.[7]

(24) $(s, m) \in \Gamma_G$ iff $(f, m) \in \Sigma$ and $(s, c, f) \in \Delta_G$ for some c-structure c.

An end-to-end mapping $(s, m) \in \Gamma_G$ is conservative if the semantic derivation $(f, m) \in \Sigma$ has grammar-dependent bounds in both directions and is thus information-conserving, and s and f of the triple $(s, c, f) \in \Delta_G$ are also co-bounded (the syntactic derivation is also conservative). Recalling that $|s|$ and $|f|$ are both bounded by $|c|$ in any derivation triple (8), it follows that

[7] This is not to discount the influence of linguistic features that may be formalized in other projections within the LFG correspondence architecture. The bounding requirements of the Conservation Principle would also govern mappings that include those other projections.

(25) An LFG syntactic derivation $(s, c, f) \in \Delta_G$ is *conservative* if also $|c|$ is bounded by both $|s|$ and $|f|$.

The syntactic recognition/parsing and realization/generation problems are solvable if only conservative derivations are defined to be linguistically relevant, in accordance with principle (22). In each direction only a finite number of size-limited annotated c-structures must be enumerated and inspected to determine whether a derivation belongs to Δ_G.[8]

With respect to Σ, Glue Semantics (Dalrymple et al. 1993; Dalrymple 1999) determines a meaning representation m for a string by a linear-logic deduction applied to a collection of premises associated with an f-structure f assigned to that string. The resource-sensitive nature of linear logic suggests that m will naturally be bounded by $|f|$, but that has not yet been clearly established. It is also unknown whether or under what additional conditions the f-structures that correspond to a given meaning representation m are bounded by $|m|$.[9] With the expectation that those issues will be resolved in future research, we return here to our focus on Δ_G, the syntactic component of Γ_G.

Kaplan & Bresnan (1982) were the first to show the undecidability of the recognition problem for unrestricted LFG grammars and the first to address it by imposing an information-conserving constraint on the derivations in Δ_G. Their constraint restricts the derivations of the annotated c-structure grammar so as to limit the distribution of empty nodes and nonbranching dominance chains.[10] The effect is to include as NBD-valid c-structures only those where every recursive subderivation contains at least one pair of terminal-dominating sisters. This specifically excludes the derivations that our demonstrations of recognition undecidability rely on. All NBD-valid derivations are conservative in the parsing direction, since the annotated c-structure is bounded by the length of the string, and the recognition and parsing problems are therefore solvable.

[8] However, the emptiness problem remains undecidable even if attention is confined only to conservative derivations. All derivations for the grammars constructed with rules (13) and (14) are conservative in the sense of (25). Emptiness requires consideration of all possible string or f-structure inputs, not just particular ones that are presented for parsing or generation. By the same token, it is undecidable whether all derivations for a given grammar are conservative.

[9] Generation from an f-structure not bounded by $|m|$ can be reduced to the undecidable problem of generating from an arbitrarily underspecified f-structure (Wedekind 1999).

[10] The NBD constraint in LFG was a specific and early example of a family of what have become known generically as *Off-line Parsability* conditions. A number of variants of Off-line Parsability have been proposed for other grammatical frameworks. See Jaeger et al. 2005 for a survey.

By a symmetrical argument, syntactic derivations will be conservative in the generation direction if they are restricted so that the size of the annotated c-structure is bounded as a function of the size of the f-structure. Unfortunately, the NBD condition is not sufficient to pick out just those information-conserving derivations and thus to ensure also that the realization and generation problems are decidable (cf. Wedekind (2014); Wedekind & Kaplan (2020)). The attribute-chain string-encoding grammars and the combining start rule (19) used in the undecidability proof for realization are ϵ-free, and it is only (nonrecursive) terminal rules that do not branch. A condition stronger than the NBD restriction is needed to guarantee that generation is conservative and thus decidable.

It has also been noted, on the other hand, that the original NBD condition may be too strong. It disallows recursive nonbranching dominance chains in every context, even when an errant subderivation is a component of a discontinuous constituent supported intuitively by an element elsewhere in the string. For example, Johnson (1986) observed that it proscribes the straightforward analysis of the Dutch double infinitive construction as provided by the grammar of Bresnan et al. (1982) and illustrated in (26).

(26)

(dat)	hij	het	boek	heeft	kunnen	lezen
(that)	he	the	book	has	able	read

'(that) he has been able to read the book'

Recursive applications of the nonbranching VP rule (27) would be required to match the level of the OBJ 'het boek' with the level of its governing predicate in the discontinuous, extended-head configuration.[11]

(27) $\text{VP} \rightarrow \quad \underset{(\uparrow \text{XCOMP}) = \downarrow}{\text{VP}}$

We address these shortcomings of Kaplan and Bresnan's NBD restriction by introducing an alternative way of identifying a subclass of conservative derivations that is better attuned to the natural flow of linguistic information. It takes into

[11] Johnson's particular example does not violate the very early refinement of the constraint wherein functional annotations are also taken into account in determining whether a category is recursive. This was introduced soon after the original formulation and later described by Kaplan & Maxwell (1996) and Dalrymple 2001. But this slightly weaker version would still disallow the intended analyses of sentences with more intransitive verbs and deeper XCOMP embeddings as in

(dat)	hij	het	boek	moet	haben	kunnen	lezen
(that)	he	the	book	must	have	able	read

'(that) he must have been able to read the book'

account the architectural correspondence between c-structures and f-structures to impose a new bound on the size of generation c-structures while relaxing the bound in the parsing direction. Our new condition makes use of the following definitions.

Let c be an annotated c-structure and let n and n' be two distinct nodes in c with n dominating n'. The *subderivation from n to n'*, denoted by $c^n_{n'}$, is the derivation that we obtain from c by removing from the subderivation rooted by n the subtree under n'. Two subderivations $c^n_{n'}$ and $c^{\bar{n}}_{\bar{n}'}$ are said to be *stacked* if the bottom node of one dominates the top node of the other. A subderivation $c^n_{n'}$ is *recursive* if n and n' are both labeled with the same annotated category.

The admissibility of recursive subderivations is then defined in terms of f-structure and string anchors.

(28) Let c be an annotated c-structure with terminal string s and f-structure f. We say that a recursive subderivation $c^n_{n'}$ is
   a. *f-anchored in* f_k if there is a node $\bar{n}$ of $c^n_{n'}$ such that $\phi(\bar{n}) = f_k$ and
   b. *s-anchored in* s_j if there is a node $\bar{n}$ of $c^n_{n'}$ such that $\bar{n}$ or a node in $\phi^{-1} \circ \phi(\bar{n})$ dominates s_j.

We refer to f_k and s_j as the f- and s-anchors of $c^n_{n'}$. The subclass of properly anchored derivations is then defined as follows.

(29) A derivation $(s, c, f) \in \Delta_G$ is *properly anchored* iff
   a. every recursive subderivation $c^n_{n'}$ of c is f- and s-anchored and
   b. the f-anchors of any two recursive subderivations in a stack are distinct, and so are their s-anchors.

If (s, c, f) is a properly anchored derivation, then every recursive subderivation is anchored in both a functional unit of the f-structure and an element of the string (29a). Moreover, requirement (29b) ensures that the anchoring for stacked recursive subderivations is one-to-one. The anchoring of stacked recursive subderivations of such a c-structure is illustrated in Figure 4.

If N is the set of annotated nonterminal categories for a grammar G, any subderivation $c^n_{n'}$ with a path length equal to $|N|$ must be recursive. The annotated c-structures of properly anchored derivations are thus bounded by the respective sizes of their corresponding strings and f-structures, as stated in the following lemma.

(30) The depth of the c-structure c of all properly anchored derivations $(s, c, f) \in \Delta_G$ is bounded by $|N|(|s| + 1)$ and $|N|(|f| + 1)$, respectively, for a string of length $|s|$ and an f-structure of $|f|$ units.

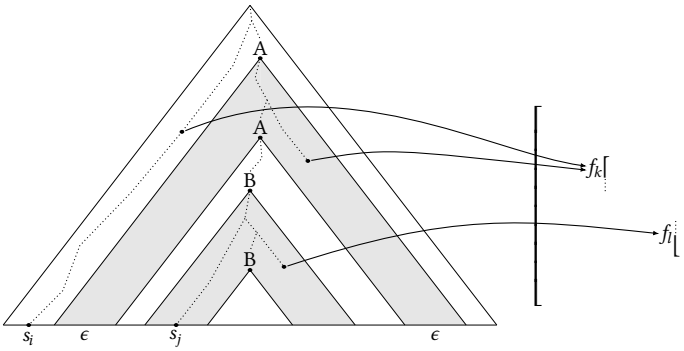


Figure 4: A c-structure with two stacked subderivations, highlighted in gray. The subderivations are f- and s-anchored at f_k, s_i and f_l, s_j, respectively, with $k \neq l$ and $i \neq j$. The upper subderivation is discontinuously string-anchored because none of its internal nodes dominates a terminal (it derives the empty string ϵ) while the lower subderivation is continuously string-anchored.

Lemma (30) implies that the properly anchored derivations for an unrestricted LFG grammar are conservative, and that the recognition and realization problems are therefore decidable if unanchored derivations are excluded from linguistic consideration. This is because only a finite number of size-bounded annotated c-structures need to be inspected in order to solve these problems.

The conditions for proper anchoring include derivations that the NBD condition does not admit and exclude derivations that NBD classifies as valid. NBD and proper anchoring, however, do agree on the status of derivations for the schematic grammars in (31).

(31) a. $\text{S} \rightarrow \underset{(\uparrow \text{GF}) = \downarrow}{\text{S}} \quad \underset{(\uparrow \text{PRED}) = \text{`P}\langle \text{GF} \rangle\text{'}}{\text{a}}$ $\quad \text{S} \rightarrow \underset{(\uparrow \text{PRED}) = \text{`A'}}{\text{a}}$ NBD-valid anchored

b. $\text{S} \rightarrow \underset{\substack{(\uparrow \text{GF}) = \downarrow \\ (\uparrow \text{PRED}) = \text{`P}\langle \text{GF} \rangle\text{'}}}{\text{S}} \quad \text{S} \rightarrow \underset{(\uparrow \text{PRED}) = \text{`A'}}{\text{a}}$ NBD-invalid shared s-anchor

The recursive subderivations for (31a) are branching and they are thus both valid and properly string-anchored. Each subderivation is also f-anchored to a distinct unit in its f-structure's GF hierarchy. If GF is a governable grammatical function, then the f-structures of all derivations are complete and coherent and correspond to the *lexical* meanings carried by the repetitively longer strings. The grammar (31b) provides the same set of complete and coherent f-structures but associates all of them to the single one-element string. That string is infinitely ambiguous

and there is no bound on the size of the constructional meaning representations determined by the recursive subderivations. These linguistically implausible subderivations are nonbranching and their string anchors cannot be distinct (29b). They are appropriately excluded as neither NBD-valid nor properly anchored.

In (32) we show grammars that provide branching derivations for every string in the set $\{a^n \mid n > 1\}$, and every derivation is thus NBD-valid but properly anchored only with respect to strings. The recursive subderivations of these grammars are excluded because they do not meet the f-anchoring conditions of (29).

(32) a. S → S a S → a NBD-valid, no f-anchor

b. S → S a S → a NBD-valid, shared f-anchor
↑=↓ (↑ PRED)='A'

The subderivations of (32a) have no f-anchors (28a) while the f-anchors for the subderivations of (32b) are not pairwise distinct (29b). The effect of the proper anchoring conditions for these configurations is consistent with other exclusionary proposals, in particular, the Different-Words version of Economy of Expression (Dalrymple et al. 2015).

As a final point of comparison, we note that the branching requirement of the NBD condition is essentially a special case of the string-anchor conditions (29) when recursive subderivations are stacked. The string anchors for valid derivations must be dominated by nodes contained within each particular subderivation. In contrast, (28b) admits stacked recursive subderivations whose distinct anchors may be dominated by nodes elsewhere in the c-structure. The linguistically significant relationship is captured in the composition $\phi^{-1} \circ \phi$. It requires only that the dominating node is an extended (co-)head of a node in a recursive subderivation, a component of the same discontinuous constituent (Zaenen & Kaplan 1995; Bresnan et al. 2016). The situation is schematized by the grammar (33).

(33) S → A P NBD-invalid, anchored
↑=↓ ↑=↓

A → A P → P p
(↑ GF)=↓ (↑ GF)=↓ (↑ PRED)='P⟨GF⟩'

A → a P → p
(↑ PRED)='A' (↑ PRED)='P⟨GF⟩'

The highest A and P nodes each dominate a separate stack of recursive subderivations. The subderivations of the P stack contain their distinct p string anchors,

but the A stack is a nonbranching (invalid) chain over the single terminal. Because of the parallel GF function assignments, the P nodes serve as extended heads for the A nodes of the A–P discontinuous constituents, and the p terminals can thus act as distinct s-anchors for the A subderivations.[12] The number of A subderivations in each properly anchored derivation is thus bounded by the length of the p substring, and only those finitely-many derivations are made available for further filtering by the Completeness and Coherence subcategorization conditions. Derivations of this type are the basis for a natural account of the discontinuous constituents in Johnson's (1986) Dutch double infinitive examples.[13]

The proper anchoring condition (29) establishes a manageable relationship between strings and f-structures by virtue of the mediating role that recursive c-structures play in the LFG syntactic architecture. This relationship is information-conserving in the sense of (22) and (25). It crucially depends on the ϕ correspondence and the linguistically motivated notion of extended heads to correlate the depth of c-structure recursion with the sizes of strings and f-structures, as indicated by Lemma (30). The set of properly anchored derivations for a given string or f-structure is finitely enumerable. It follows that recognition and realization are decidable for that restricted subset of derivations and so are other input-specific problems as listed in (21) and in Appendix A. It is possible, for example, to identify the most economical (properly anchored) derivation for a given f-structure because there are only a finite number of candidates whose c-structures must be compared. Proper anchoring, however, is not sufficient to ensure decidability of the emptiness problem (the demonstration in Section 4.1 involves only properly anchored derivations), and other questions that require consideration of all possible string and f-structure inputs also remain undecidable.

6 Intractability of parsing and generation

The recognition/parsing, realization/generation, and other problems are decidable for the conservative, properly-anchored derivations of arbitrary LFG grammars. But the fact that the number of derivations for a given input is finite does

[12] This arrangement of parallel function assignments gives rise to the so-called "zipper" configuration discussed below and by Maxwell & Kaplan (1996) and Kaplan & Wedekind (2020).

[13] Note also that the same verbs could be reused as anchors for a different stack of recursive subderivations, for example, in the hypothetical case that the language allows an elaboration of this construction with a ditransitive lower verb and a dislocated OBL NP. This is because pairwise distinctness (29b) applies on a per-stack basis.

not mean that it is small, and indeed the computational cost of solving these problems may be very high. We show in this section that recognition and realization are intractable in the worst case, that is, for arbitrary grammars they cannot be solved in a number of processing steps polynomial in the size of a given input. Their intractability is demonstrated by the usual technique of reducing these problems to another problem that is already known to be intractable. The technique requires that the reduction itself is computable in polynomial time so that we know that the reduction procedure does not hide the complexity of the problems of interest.

The 3-SAT problem is the problem in the NP-complete complexity class often used for polynomial-time reductions that establish the intractability of other problems. This is the problem of determining the satisfiability of a Boolean formula in conjunctive normal form where each of the conjoined clauses is a disjunction of three literals. That is, each formula is a conjunction of the form $C_1 \wedge .. \wedge C_n$, each clause C_j is a disjunction of the form $l_{j_1} \vee l_{j_2} \vee l_{j_3}$, and each literal l_{j_i} is a propositional variable p_k or a negated variable $\neg p_k$. The question is whether there is at least one way of assigning truth values to the variables that makes all the clauses be true. The three-clause formula in (34a) is a simple problem that is satisfiable under several assignments among which is the one in (34b).

(34) a. $(p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee \neg p_3)$
b. p_1=TRUE, p_2=FALSE, p_3=FALSE

We show that the recognition problem is intractable by providing a small LFG grammar G such that the set of f-structures $Par_G(s) \neq \emptyset$ if and only if the string s is an encoding of a satisfiable Boolean problem in conjunctive normal form. A formula is presented as a sequence of substrings one corresponding to each clause. The substring for the i^{th} clause begins with the letter c followed by the string of digits $d_1..d_j$ that represents the integer i. This is followed by substrings that identify the literals that make up that clause. Every occurrence of a positive literal p_k is encoded as the character + followed by the digits representing the integer k, and every occurrence of a negative literal $\neg p_k$ is represented as the character − followed by the digits for k. According to this scheme the formula (34a) is presented as the string of characters (35).[14]

(35) c1 +1+2+3 c2 −1−2+3 c3 −1+2−3

[14] We would of course see longer digit strings, not just singletons, for problems with ten or more clauses or variables.

There is a simple information-conserving LFG grammar *G* that maps a string representing any satisfiable Boolean problem into f-structures that recapitulate the problem and make explicit the truth-value assignments that solve it. The linear order of clause and literal substrings is recast into descending chains of digit attributes in the f-structure. The sequences for the signed propositional variables of all literals are attached at the bottom of the attribute chain of their containing clause, and the grouping of literals within clauses is thus maintained. The lower PROB(lem) substructure shown in (36)[15] corresponds to the problem string (35).

(36)
$$\left[\begin{array}{ll}
\textsc{sol} & \left[\begin{array}{ll} 1_p & [\textsc{val}\ \textsc{true}] \\ 2_p & [\textsc{val}\ \textsc{false}] \\ 3_p & [\textsc{val}\ \textsc{false}] \end{array}\right] \\
\textsc{prob} & \left[\begin{array}{ll}
1_c & \left[\begin{array}{ll} + & \left[\begin{array}{ll} 1_p & [\textsc{val}\ \textsc{true}] \\ 2_p & [\textsc{val}\ \textsc{true}] \\ 3_p & [\textsc{val}\ \textsc{true}] \end{array}\right] \end{array}\right] \\
2_c & \left[\begin{array}{ll} + & [3_p\ [\textsc{val}\ \textsc{true}]] \\ - & \left[\begin{array}{ll} 1_p & [\textsc{val}\ \textsc{false}] \\ 2_p & [\textsc{val}\ \textsc{false}] \end{array}\right] \end{array}\right] \\
3_c & \left[\begin{array}{ll} + & [2_p\ [\textsc{val}\ \textsc{true}]] \\ - & \left[\begin{array}{ll} 1_p & [\textsc{val}\ \textsc{false}] \\ 3_p & [\textsc{val}\ \textsc{false}] \end{array}\right] \end{array}\right]
\end{array}\right]
\end{array}\right]$$

The upper SOL(ution) substructure corresponds to the truth-value assignment (34b) that makes all clauses be true.

Let S_{num} be the root category of a descending attribute-chain grammar for the regular language $Digit^+$ of arbitrarily long digit sequences, with the scaffolding attributes B and E giving access to the top and bottom of the descending chains (as in (17) above). Then the rules in (37) provide a c-structure and an f-structure for the string encoding of every well-formed and satisfiable Boolean formula. In particular, the f-structure for one of the derivations for string (35) appears as (36) when the innocuous scaffolding attributes are not displayed.

(37) a. $S \rightarrow \quad Clause^+$
$(\uparrow \textsc{prob}) = (\downarrow \textsc{b})$
$(\uparrow \textsc{sol}) = (\downarrow \textsc{sol})$

[15] The clause and propositional variable subscripts *c* and *p* are provided just for readability; they are not actually part of the formal structure.

b. Clause → c Snum Lit* Lit Lit*
(↑ B)=(↓ B) (↑ CE)=↓ (↑ CE)=↓ (↑ CE)=↓
(↓ E)=(↑ CE) (↑ SOL)=(↓ +)
(↑ SOL)=(↓ −)

c. Lit → { + Snum | − Snum }
(↑ +)=(↓ B) (↑ −)=(↓ B)
(↓ E VAL)=TRUE (↓ E VAL)=FALSE

The start rule (37a) recognizes the conjunction of arbitrarily many clause constituents.[16] Every clause consists of one or more literals, and every literal consists of a positive or negative marking followed by the identifier of its propositional variable. The (↑ PROB)=(↓ B) annotation promotes all the clause attribute chains to the problem substructure. The additional clause-ending scaffolding attribute CE makes it possible to connect the positive and negative literals to the bottom of their containing-clause chains. The truth-value assignments in (37c) attach the value TRUE at the bottom of the variable chains of positive literals and FALSE at the bottom of negative literals, thereby encoding the truth-value assignments that make each literal be true. Finally, just one of the true literals is selected to make the clause true, and the SOL annotations incorporate the variable and truth-assignment of that literal (whether it happens to be positive or negative) into the global solution.

A derivation in G will succeed only if the literals chosen locally and independently for each clause result in SOL truth-value assignments that are globally consistent. If a problem is unsatisfiable, then the f-description for every c-structure derivation will be inconsistent. Thus for a string s encoding an arbitrary Boolean problem, the set $Par_G(s) \neq \emptyset$ if and only if that problem is satisfiable for at least one consistent set of truth-value assignments.[17]

With this abstract formal grammar it is easy to see the potential source of computational complexity for LFG recognition. For each literal of every clause, rule (37b) produces an alternative annotated c-structure that makes a different contribution to SOL. The number of properly anchored derivations that must be inspected for global consistency thus grows in the worst case as an exponential in

[16] For succinctness and clarity we use LFG's traditional Kleene $^+$ and * notations to specify repeating category sequences rather than their right-recursive equivalents. For example, the single rule (37a) is equivalent to the two rules S → Clause (↑ PROB)=(↓ B) (↑ SOL)=(↓ SOL) and S → Clause (↑ PROB)=(↓ B) ↑=↓ S . (↑ SOL)=(↓ SOL)

[17] Berwick (1982) provided the first NP-completeness proof for the LFG recognition problem and Stanley Peters (p.c. 1982) offered a different argument. The demonstration here uses far less of the LFG machinery than those earlier proofs and generalizes to problems with arbitrary numbers of clauses and variables.

the number of clauses. For example, there will be 3^n derivations to consider in the case of a 3-SAT problem with n clauses each of which has three literals. Linguistic grammars will also have this exponential complexity profile if their fixed number of rules describe morphosyntactic agreement dependencies that range over the full length of the input string. We can also see, schematically, a configuration that is sufficient to guarantee tractability while still allowing for input strings of arbitrary length. Suppose there is a constant k that limits the number of clauses that a single S can expand to, as in (38), but with a new starting category S′ that allows for the concatenation of an arbitrary number of k-limited S's.

(38) $\mathrm{S}' \rightarrow \mathrm{S}^{+}$ $\qquad$ $\mathrm{S} \rightarrow \mathrm{Clause}^{\leq k}$
$(\uparrow \textsc{prob}) = (\downarrow \textsc{b})$
$(\uparrow \textsc{sol}) = (\downarrow \textsc{sol})$

Crucially, there are no annotations on S′ to link the f-structures of the S nodes, and thus there can be no interaction among the truth assignments of the embedded clauses. The worst case complexity for a string of n 3-literal clauses is proportional to $\frac{n}{k} \cdot 3^k$. This is exponential in the grammar-dependent constant k but polynomial in the length of the input. This foreshadows the tractability of k-bounded LFG grammars that we discuss in the next section.

For recognition the Boolean problem string and f-structure are organized so that the signed propositional variables are grouped within clauses, and the grammar checks for consistency of variable truth values in the global SOL structure. For the reduction of the LFG realization problem to Boolean satisfiability, the string and corresponding f-structure are transposed so that a Boolean problem is presented with its clauses grouped within its propositional variables. We again provide a small LFG grammar G' now with the property that the string set $Gen_{G'}(f) \neq \emptyset$ if and only f is the encoding of a satisfiable Boolean problem.[18] The transposed string presentation and equivalent f-structure for problem (34a) are shown in (39).

(39) a. p1 +1−2−3 p2 +1−2+3 p3 +1+2−3

[18] See Wedekind & Kaplan (2021) for a fuller discussion of the technical issues particularly concerning the realization problem.

b.
$$\left[\begin{array}{ll} 1_p & \left[\begin{array}{ll} + & [1_c\ ] \\ - & \left[\begin{array}{l} 2_c \\ 3_c \end{array}\right] \end{array}\right] \\ 2_p & \left[\begin{array}{ll} + & \left[\begin{array}{l} 1_c \\ 3_c \end{array}\right] \\ - & [2_c\ ] \end{array}\right] \\ 3_p & \left[\begin{array}{ll} + & \left[\begin{array}{l} 1_c \\ 2_c \end{array}\right] \\ - & [3_c\ ] \end{array}\right] \end{array}\right]$$

The string indicates that variable 1 occurs in a positive literal in the first clause but in negative literals in the second and third clauses. The linear order of variables and clauses in the string is reflected in the f-structure's descending attribute chains. The clause identifiers are grouped according to the signed propositional variables of the literals that they contain.

The LFG grammar G' in (40) establishes the relationship between the equivalent string and f-structure expressions of any well-formed Boolean formula, whether satisfiable or not.

(40) a. S → Var$^+$
(↑ PROB)=(↓ B)
(↑ SOL)=(↓ SOL)

b. Var → p Snum {+ Snum | − Snum}$^+$
(↑ B)=(↓ B) (↑ VE +)=(↓ B) (↑ VE −)=(↓ B)
(↓ E)=(↑ VE)

A sentence consists of a sequence of proposition-variable substrings each of which begins with a variable identifier followed by any number of digit substrings representing the clauses in which that variable appears. Each clause is prefixed with + and − to indicate whether the variable appears in a positive or negative literal. The annotations promote the variable's descending digit-chain to the top and attach the clause identifiers under the + or − attributes at the bottom of each variable chain, according to whether the clause is positively or negatively marked. This produces the f-structure displayed in (39b) (again with omission of the scaffolding attributes B/E and now VE). If f is an input f-structure for realization that expresses an arbitrary Boolean problem in this way, then the set $Gen_{G'}(f)$ includes a string of the form (39a).

Both the input f-structure and the grammar must be elaborated so that LFG realization distinguishes between satisfiable and unsatisfiable Boolean problems.

Along with the encoding of a particular problem the f-structure must specify the necessary and sufficient conditions for a solution, namely, that every clause is true under at least one consistent assignment of truth values to the variables. The input f-structure represents this requirement by attaching a value TRUE at the bottom of every clause identifier in the top-level SOL substructure and wherever the clause appears in the problem encoding under PROB. F-structure (41) is the elaboration of (39b) with this additional information.

(41)
$$\left[\begin{array}{ll}
\textsc{sol} & \left[\begin{array}{l} 1_c\ \ [\textsc{val true}] \\ 2_c\ \ [\textsc{val true}] \\ 3_c\ \ [\textsc{val true}] \end{array}\right] \cdots\cdots\cdots \\
\textsc{prob} & \left[\begin{array}{ll}
1_p & \left[\begin{array}{ll} + & \left[1_c\ \ [\textsc{val true}]\right] \\ - & \left[\begin{array}{l} 2_c\ \ [\textsc{val true}] \\ 3_c\ \ [\textsc{val true}] \end{array}\right] \end{array}\right] \\
2_p & \left[\begin{array}{ll} + & \left[\begin{array}{l} 1_c\ \ [\textsc{val true}] \\ 3_c\ \ [\textsc{val true}] \end{array}\right] \\ - & \left[2_c\ \ [\textsc{val true}]\right] \end{array}\right] \\
3_p & \left[\begin{array}{ll} + & \left[\begin{array}{l} 1_c\ \ [\textsc{val true}] \\ 2_c\ \ [\textsc{val true}] \end{array}\right] \\ - & \left[3_c\ \ [\textsc{val true}]\right] \end{array}\right]
\end{array}\right]
\end{array}\right]$$

In this depiction the dotted line shows that the clause identifiers and their truth values in the solution are equated to all of their occurrences in the problem-statement substructure.

F-structure (41) is correctly assigned to the satisfiable problem string (39a) if the single Var expansion rule above is replaced by the alternatives in (42).

(42) a. Var → p Snum { + Snum | − Snum }$^+$
(↑ B)=(↓ B) (↑ VE +)=(↓ B) (↑ VE −)=(↓ B)
(↓ E)=(↑ VE) (↑ SOL)=(↓ B) (↑ SOL)=(↓ B)
(↓ E VAL)=TRUE

b. Var → p Snum { + Snum | − Snum }$^+$
(↑ B)=(↓ B) (↑ VE +)=(↓ B) (↑ VE −)=(↓ B)
(↓ E)=(↑ VE) (↑ SOL)=(↓ B) (↑ SOL)=(↓ B)
(↓ E VAL)=TRUE

The SOL annotations in both versions lift all the clause identifiers, whether positive or negative, to the top-level. The rules differ in that (42a) also attaches the

value TRUE only at the bottom of every positive-clause chain while (42b) attaches TRUE only to the bottom of every negative clause. Thus for every variable there is a choice in every derivation between the two expansions, corresponding to a guess of consistent truth-value assignments for every variable.

If a problem is satisfiable, then each clause will be assigned TRUE under at least one variable, that value will be carried with the clause identifier into the SOL structure, and it will propagate by equality to all of the other (positive or negative) occurrences of that clause. The result will be an f-structure configured as in (41), and the string corresponding to the problem substructure will be a realization of that f-structure.

Grammar G' will also derive annotated c-structures and f-structures for a string that represents an unsatisfiable problem, but each of those f-structures will be missing a required truth value for at least one of the clauses. For the trivially unsatisfiable problem $p_1 \wedge \neg p_1$ the input f for realization is the f-structure (43a) and (43b) is its corresponding string expression.

(43) a.
$$\begin{bmatrix} \text{SOL} & \begin{bmatrix} 1_c & [\text{VAL TRUE}] \\ 2_c & [\text{VAL TRUE}] \end{bmatrix} \cdots \\ \text{PROB} & \begin{bmatrix} 1_p & \begin{bmatrix} + & [1_c & [\text{VAL TRUE}]] \\ - & [2_c & [\text{VAL TRUE}]] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

b. p1 +1−2

With just one variable there is only one choice between the alternative Var expansion rules, giving rise to two derivations. Assigning TRUE to the positive literal produces f-structure (44a) and (44b) results if the negative literal is selected. Neither of these is complete for all the attributes and values of (43a) and thus string (43b) (and any other string that corresponds to the problem substructure) does not belong to $Gen_{G'}(f)$.

(44) a.
$$\begin{bmatrix} \text{SOL} & \begin{bmatrix} 1_c & [\text{VAL TRUE}] \\ 2_c & \end{bmatrix} \cdots \\ \text{PROB} & \begin{bmatrix} 1_p & \begin{bmatrix} + & [1_c & [\text{VAL TRUE}]] \\ - & [2_c &] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

b.
$$\begin{bmatrix} \text{SOL} & \begin{bmatrix} 1_c & \\ 2_c & [\text{VAL TRUE}] \end{bmatrix} \cdots \\ \text{PROB} & \begin{bmatrix} 1_p & \begin{bmatrix} + & [1_c &] \\ - & [2_c & [\text{VAL TRUE}]] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Any Boolean satisfiability problem can thus be reduced to the realization problem for the simple LFG grammar G' if the problem is translated to an input f-structure that encodes the problem and the requirement of truth for all clauses. A derivation for G' will map a string to that f-structure if and only if the Boolean problem is satisfiable. As for recognition, realization is intractable because the

number of derivations whose f-structures must be compared to the input is exponential in the size of the problem, in this case the number of variables it contains.

7 k-Bounded LFG grammars and tractability

These intractability results for the conservative, properly anchored derivations of arbitrary grammars raise the question whether there are other formal restrictions that will guarantee that the computationally important problems of recognition and realization can be solved in polynomial time. Seki et al. 1993 first established the connection between a much more restricted subclass of LFG grammars and Linear Context-Free Rewriting Systems (LCFRS), formal systems that can describe only mildly context-sensitive dependencies and for which the recognition problem is tractable (Kallmeyer 2010). The Seki et al. *finite-copying grammars* permit rules with the very limited functional annotations in (45a) provided that all their derivations also satisfy the bounding condition (45b).

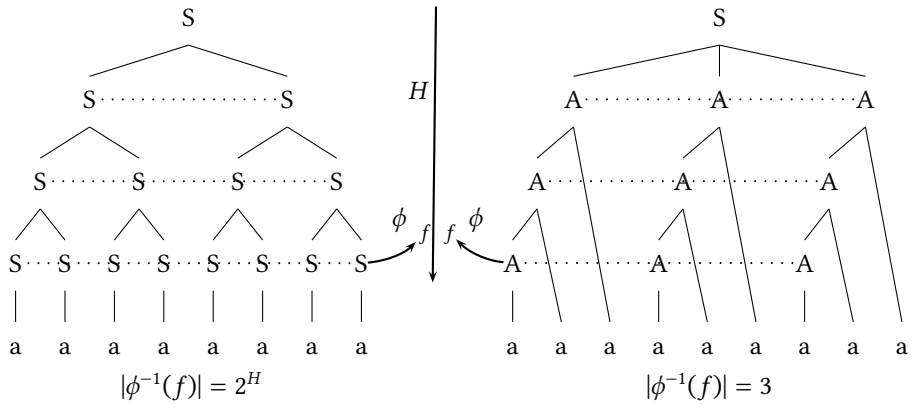
(45) a. Each category on the right-side of a rule can be annotated with at most one function assignment of the form $(\uparrow \textsc{f}) = \downarrow$ and any number of atom-value assignments only of the form $(\uparrow \textsc{a}) = \textsc{v}$.

b. There is a constant k such that no more than k nodes map to the same f-structure element f in any derivation. That is, $|\phi^{-1}(f)| \leq k$ for every f.[19]

Structure sharing in finite-copying grammars can only be achieved through instantiated function-assigning annotations. This specific type of structure sharing is occasionally referred to as "zipper" unification. That is, if two distinct nodes n_1 and n_2 map to the same f-structure in a derivation, then there must always be a node $\hat{n}$ dominating these nodes such that the sequences of function-assigning annotations on the paths from $\hat{n}$ to n_1 and n_2, respectively, must be identical, that is, form a "zipper".

The bounding condition (45b) limits the number of non-local dependencies that can arise through structure sharing and thus proscribes c-structure recursions that give rise to zippers of size greater than the constant k. Indeed, Seki et al. have shown that the recognition problem is NP-complete for grammars

[19] This condition can also be expressed in terms of an extended-head formulation: $|\phi^{-1} \circ \phi(n)| \leq k$ for every c-structure node n. The parameter k may also be regarded as a formal characterization of the linguistic notion *degree of discontinuity* (Chomsky 1953).



a. C-structure for grammar (46a) b. C-structure for grammar (46b)

Figure 5: Zipper nodes in depth-balanced c-structures

that meet the notational restrictions (45a) but do not satisfy the bounding condition (45b). Thus the bounding condition is crucial for tractable performance even with the severe notational restrictions of the finite-copying formalism.

Grammars with these limited annotations are expressive enough to specify the kinds of derivations depicted in Figure 5. The derivation on the left is produced by the simple recursive rules in (46a) while the one on the right is derived with the grammar (46b).

(46) a. S → S S S → a
(↑ L)=↓ (↑ L)=↓ (↑ L)=#

b. S → A A A A → A a A → a
(↑ L)=↓ (↑ L)=↓ (↑ L)=↓ (↑ L)=↓ (↑ L)=#

These grammars both meet the finite-copying notational restrictions (45a), and the derivations of both grammars have nodes that share structure in the zipper configurations indicated by the dotted lines. But the difference in these structure-sharing configurations corresponds to a difference in computational complexity. For all derivations of grammar (46a) the number of nodes in the set $\phi^{-1}(f)$ is an exponential in the height H of those nodes, as indicated in Figure 5a. In contrast, for all derivations of grammar (46b) the number of nodes in a structure-sharing set is bounded by a constant (3 in this case) that is independent of their height (Figure 5b). Grammar (46b) but not (46a) meets the finite-boundedness property

(45b), and this is a decidable property for all derivations of such notationally restricted grammars. Note that the string and f-structure sizes are correlated in the derivations of both grammars. They are thus not distinguished by the conditions of proper anchoring.

The restrictions (45a) are obviously too severe for linguistic description. The notation disallows, for example, the trivial ↑ = ↓ annotations that mark the heads and coheads in the functional domain of a predicate, the (↑ XCOMP SUBJ) = (↑ OBJ) equations of functional control, and all other ways of relating the f-structures of different nodes. They also exclude multi-attribute value specifications, such as (↑ SUBJ NUM) = SG, that encode agreement requirements, and any direct specification of feature values on daughter nodes, as in (↓ CASE) = NOM.

Wedekind & Kaplan 2020 take the Seki et al. 1993 finite-copying grammars as the starting point for developing a subclass of LFG grammars that are more suitable for linguistic description but are similarly limited in their expressive power. The *k-bounded* LFG grammars of Wedekind and Kaplan allow the richer set of annotations in (47).

(47) Basic annotations

↑ = ↓	(co)head identifier
(↑ F) = ↓	function assignment
(↑/↓ A B C ⋯) = V	general atom-value assignments

Reentrancies

(↑ F) = (↑ H)	local-topic link
(↓ G) = (↑ H)	daughter-mother control
(↓ G) = (↓ H)	daughter sharing
(↓ G) = ↑	promotion
(↑ F) = ↑	mother cycle
(↓ G) = ↓	daughter cycle
(↑ F G) = (↑ H)	functional control

The annotations in this enlarged set include those that are commonly used in natural language grammars and that remain compatible with theoretical conventions such as the Principle of Functional Locality (Kaplan & Bresnan 1982). In *k*-bounded grammars these more flexible annotations are accompanied with additional conditions that also limit the number of non-local dependences that can arise through structure sharing. The *k*-bounded LFG grammars thus enjoy the same mathematical and computational properties that Seki et. al identified: They characterize only mildly context sensitive languages for which recognition is tractable. The additional conditions that a *k*-bounded grammar *G* must meet are listed in (48).

(48) a. Each right-side category is annotated with at most one function assignment (↑ F)=↓, and (co)head identifiers ↑=↓ and function assignments always appear in complementary distribution (to keep separate the properties of heads and their complements).

b. The *functional domains* of G (the collections of ↑=↓-annotated nodes that map to the same f-structure) are bounded by a grammar-dependent constant h (so G can be converted to an equivalent grammar $G^{\setminus\uparrow=\downarrow}$ that is free of ↑=↓ annotations).

c. The derivations of the grammar formed by removing all reentrancies from $G^{\setminus\uparrow=\downarrow}$ are bounded by a grammar-dependent constant k, as in (45b). (Wedekind & Kaplan 2020 call this the *reentrancy-free kernel* of G.)

d. Reentrancies are nonconstructive.

Nonconstructivity is an implicit property of derivations in broad coverage LFG grammars that has been mentioned (but not well formalize) in the LFG literature as a requirement for functional uncertainty and off-path constraints (Crouch et al. 2008; Zaenen & Kaplan 1995) (Dalrymple et al. 1995: page 133). The reentrancies of a grammar are nonconstructive if they cannot extend the ϕ mapping from c-structure nodes to f-structure units beyond the correspondences established by simple function assignments (the zipper-forming annotations of finite-copying grammars).

The difference between constructive and nonconstructive reentrancies is illustrated in Figure 6. On the left side the reentrancies are constructive because they cause the nodes n_2 and n_5 to map to the same f-structure element. If reentrancies are nonconstructive, as in the derivation on the right side, they do not introduce node-to-f-structure mappings that are not entailed by function assignments alone, and thus they do not affect the bounds that function assignments establish on the ϕ^{-1} node classes. Nonconstructive reentrancies only propagate the limited atom-value information that the grammar attaches to individual nodes and not the unregulated amount of information that might be associated recursively with entire subtrees.

Wedekind & Kaplan 2020 have shown that the nonconstructivity condition (48d) is decidable if the ϕ^{-1} node classes of a grammar are k-bounded and if any two-attribute functional control annotations can be reduced to shorter ones (e.g. shrinking (↑ XCOMP SUBJ)=(↑ OBJ) to (↓ SUBJ)=(↑ OBJ) when conjoined with (↑ XCOMP)=↓). While it is undecidable in general whether every functional control annotation can be shortened (see example (52c) in Appendix A), they can

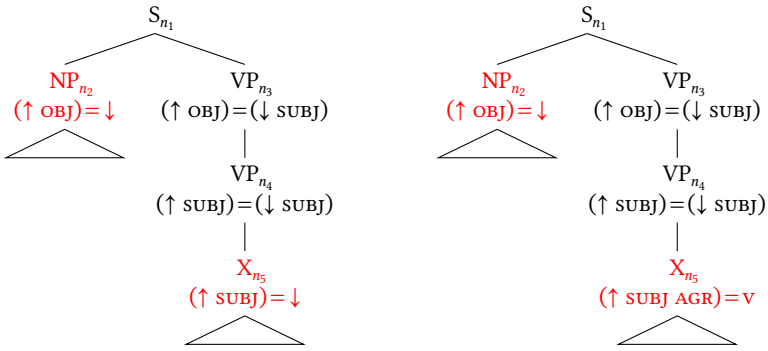


Figure 6: Constructive and nonconstructive reentrancies.

always be reduced to daughter-mother controls in derivations that meet the requirements of the Coherence Condition. Wedekind & Kaplan 2020 provide a formal specification of nonconstructivity, this expected consequence of Coherence, and other technical requirements that are sufficient to decide whether an arbitrary LFG grammar belongs to the k-bounded subclass and therefore describes only mildly context-sensitive languages.

Wedekind & Kaplan 2020 also prove that for any LFG grammar G with the properties defined in (47) and (48) there is a linear context free rewriting system that accepts all and only the strings in $L(G)$ and allows recovery of the f-structures that G assigns to each such string. The tractability of LCFRS recognition thus establishes for k-bounded LFG grammars that recognition of individual input strings can be accomplished in time polynomial in their length. Here we sketch a simpler demonstration that is framed entirely within the LFG formalism. This is based on a line of argument that Lang 1994 and others have developed for the recognition problem of context-free grammars.

On this approach to context-free recognition the solution is partitioned into two phases. Given an input string s and an arbitrary context-free grammar G with $|G|$ rules, the first step is to specialize G to a context-free grammar G_s with the property that $s \in L(G)$ if and only if $L(G_s) \neq \emptyset$. The second step then is to determine whether or not the language $L(G_s)$ is empty. In the context-free case the procedure for specializing G to s and the size of the resulting grammar are both polynomial in the length of the input, and for context-free grammars the emptiness problem is bounded by a polynomial in grammar size. It follows on

this particular argument (among many others) that context-free recognition is bounded by a polynomial in $|s|$.

This two-part strategy immediately carries over to LFG recognition. The specialization of an arbitrary LFG grammar G to a given input s can be extracted from the chart data structures provided by any number of context-free parsing algorithms modified to keep track of the annotations of matching c-structure categories (equivalently, to operate unmodified on left-side annotated rules as in (6) above). This is a polynomial process that results in an annotated LFG grammar G_s of size also polynomial in $|s|$ that assigns to s all and only the f-structures that G assigns to s. In particular, $Par_G(s) = \emptyset$ if and only if $Par_{G_s}(s) = \emptyset$, and this is equivalent to the question whether $L(G_s) = \emptyset$.

We noted above that the emptiness problem for arbitrary LFG grammars remains undecidable even if only properly anchored derivations are taken into account. However, if G belongs to the subclass of k-bounded grammars then so does G_s, and the emptiness problem for arbitrary k-bounded grammars is not only decidable but solvable with worst-case complexity that is polynomial in grammar size. A proof of this property is outlined in Appendix B. Thus, following the context-free argument, for any input string s and k-bounded LFG grammar G, in time polynomial in $|s|$ it can be determined whether $s \in L(G)$.

Wedekind & Kaplan 2012 applied a similar two-phase strategy to prove that the realization problem is decidable for an arbitrary LFG grammar G and an arbitrary acyclic input f-structure f (see also Kaplan & Wedekind (2000)). They specialized G to a grammar G_f with the property that the string-set $Gen_G(f) = \emptyset$ if and only if $L(G_f) = \emptyset$. The grammar G_f is context-free and its emptiness is therefore decidable. In the general case the specialization phase is not tractable and the resulting G_f may be exponentially larger than G. If G is k-bounded, however, the consistency and completeness of all LFG derivations for any f, even cyclic ones, can be simulated with an annotation-free polynomial expansion of the categories and rules of G.

Thus the recognition and realization problems for k-bounded grammars can be solved in polynomial time: for arbitrary inputs it can be determined whether the sets $Par_G(s)$ and $Gen_G(f)$ are empty. But the k-bounded restrictions are not sufficient to guarantee that those sets contain only a finite number of elements. The context-free grammar G_f, for example, can describe a language with arbitrarily long strings, if G allows for unlimited morphological markers in subtrees with nodes that are not in the domain of the ϕ projection. And the f-structures for a given string can also be arbitrarily large, if the grammar permits stacked recursive subderivations. If useless rules are removed from G_f and if annotations are carried along in the grammar G_s^* as described in Appendix B, then the generation

algorithm for context free grammars can be used to enumerate the elements of $Gen_G(f)$ and $Par_G(s)$, one after the other and each in linear time. But obviously the generation and parsing enumerations will never terminate in the face of infinite ambiguity. The derivations for a k-bounded grammar are not necessarily conservative in the sense of (25), even though the emptiness tests for recognition and realization have tractable solutions.

The proper-anchoring/conservation and k-bounded restrictions target different sources of mathematical and computational complexity. Proper anchoring limits the height of recursive subderivations in a stack but imposes no constraint on the number of stacks in a single derivation. The k-bounded restrictions limit the degree of discontinuity but say nothing to relate the sizes of strings and f-structures. The combination of constraints provides for conservative, finitely-ambiguous, derivations with tractable recognition and realization. We have suggested above that conservation is a plausible pretheoretic property of natural communication, and we have also argued that the k-bounded patterns of information flow are compatible with other linguistic principles (Kaplan & Wedekind 2019; Wedekind & Kaplan 2020). The k-bounded restrictions (47-48) and the proper anchoring condition (29) are different ways of moderating the excessive mathematical and computational power of the basic LFG formalism while preserving in different ways its suitability for linguistic description.

8 Summary

Lexical-Functional Grammar is equipped with a simple architecture that formalizes a piecewise correspondence between structures of different types, the phrase-structure trees of the constituent structure and the attribute-value matrices of the functional structure. We have shown that f-structure encodings of the strings of arbitrary context-free grammars can be produced by straightforward application of the formalism's most primitive annotations. From that it follows that recognition/parsing, realization/generation, and other mathematical and computational questions are easily proved to be undecidable.

One source of this excessive power, at least for the recognition and realization problems, is the fact that an unrestricted grammar may establish no systematic relationship between the sizes of input strings and the sizes of corresponding f-structures. This is inconsistent with the Principle of Conservation (22) that we suggest is a pretheoretic property of language as a medium of communication: the derivational machinery that maps in both directions between strings and their f-structures does not add or subtract arbitrary amounts of information.

Problems that relate to specific inputs, including recognition/parsing and realization/generation, become decidable if unconservative derivations are excluded from consideration.

The annotated c-structure is the generative component of the LFG formalism and serves as the intermediary between strings and f-structures. Thus we have proposed a condition on recursive c-structure subderivations that ensures that strings and f-structures stand in a conservative relationship. A derivation is properly anchored if each recursive subderivation is anchored in elements of both the string and f-structure and recursive subderivations in a stack do not share the same anchors. For parsing this condition improves on the original prohibition of derivations with nonbranching dominance chains but applies to the generation problem as well.

The proper anchoring condition is strong enough to ensure decidability but we show that it is not strong enough to guarantee tractability. Tractability for recognition and realization is the computationally important property of the k-bounded LFG grammars and derivations. These grammars are in the class of mildly context-sensitive grammars, even though their derivations are not necessarily conservative. The subclass of LFG grammars and derivations that meet the conditions of both proper anchoring and k-boundedness has attractive mathematical and computational properties and may serve as a better foundation for a formal theory of natural language syntax.

Appendix A: Other undecidable questions

In Section 4 we used the descending attribute-chain string encoding (17) for arbitrary Chomsky Normal Form context-free grammars to prove the undecidability of the realization problem. We apply that same encoding here to show that several more specific properties are undecidable for unrestricted LFG grammars. The start rule (49) follows the pattern laid out earlier in (19). It denotes the ends of the S_1 and S_2 substrings as E_1 and E_2 respectively and includes a place-holder P for grammatical fragments that we will use to encode other decision problems. As noted before, there are no atomic values and therefore no atom-value clashes in the attribute-chain string encodings, and the set of derivations can only be filtered by properties spelled out in P.

(49)
$$\begin{array}{lcccc} \mathrm{S} & \rightarrow & \mathrm{S}_1 & \mathrm{S}_2 & P \\ & & (\uparrow \textsc{l}) = \downarrow & (\uparrow \textsc{r}) = \downarrow & \\ & & (\downarrow \textsc{b}) = (\uparrow \textsc{b}) & (\downarrow \textsc{b}) = (\uparrow \textsc{b}) & \\ & & (\downarrow \textsc{e}) = (\uparrow \textsc{e}_1) & (\downarrow \textsc{e}) = (\uparrow \textsc{e}_2) & \end{array}$$

If any realization of P expresses a particular property that is satisfied only if $F(G)$ contains f-structures with equal E_1 and E_2 values, then that property must be undecidable.

As a first example, the alternative annotations on the terminal # in (50) shows that it is undecidable whether a minimal model satisfies either defining or constraining equalities between two f-structure units.

(50) $S \rightarrow$ S_1 ($\uparrow L)=\downarrow$, ($\downarrow B)=(\uparrow B)$, ($\downarrow E)=(\uparrow E_1)$ S_2 ($\uparrow R)=\downarrow$, ($\downarrow B)=(\uparrow B)$, ($\downarrow E)=(\uparrow E_2)$ # $\left\{\begin{array}{l}(\uparrow E_1)=(\uparrow E_2)\\(\uparrow E_1)=_c(\uparrow E_2)\\(\uparrow E_1)\neq(\uparrow E_2)\end{array}\right\}$

The function assignments on X and Y in (51) show that it is in general undecidable whether there are derivations with nodes that ϕ maps to the same f-structure.

(51) $S \rightarrow$ S_1 ($\uparrow L)=\downarrow$, ($\downarrow B)=(\uparrow B)$, ($\downarrow E)=(\uparrow E_1)$ S_2 ($\uparrow R)=\downarrow$, ($\downarrow B)=(\uparrow B)$, ($\downarrow E)=(\uparrow E_2)$ X ($\uparrow E_1)=\downarrow$, ($\uparrow E_1)=(\uparrow E_2)$ Y ($\uparrow E_2)=\downarrow$

It follows from this that any other property that depends on nodes mapping to the same f-structure is also undecidable.

Thus, expanding the nonterminals X and Y with the rules (52a) shows that the satisfiability of existential constraints or constraints between atomic values is undecidable and, as a consequence, that Completeness and Coherence are also undecidable. The annotations (52b) establish that it is undecidable whether an arbitrary LFG grammar gives rise to cyclic f-structures, and (52c) shows that functional control annotations cannot decidably be reduced to combinations of function assignments and daughter-mother controls.

(52) a. $X \rightarrow$ x ($\uparrow F)=V$ $Y \rightarrow$ y $\left\{\begin{array}{l}(\uparrow F)\\\neg(\uparrow F)\\(\uparrow F)=_c V\\(\uparrow F)\neq V\end{array}\right\}$

b. $X \rightarrow$ x ($\uparrow F\ G)=(\uparrow H)$ $Y \rightarrow$ y ($\uparrow F)=(\uparrow H)$

c. $X \rightarrow$ x ($\uparrow F\ G)=(\uparrow H)$ $Y \rightarrow$ y ($\uparrow F)=\downarrow$

Appendix B: Emptiness of k-bounded LFG grammars

We sketch here the proof that the complexity of the emptiness problem for an arbitrary k-bounded LFG grammar G is polynomial in $|G|$, the size of its rule set. The argument makes use of the three grammar transformations listed in (53). Each of these can be carried out in polynomial time, as indicated below, and each guarantees that G and the transformed grammar G' are *co-empty*, that is, that the set of derivations $\Delta_G = \emptyset$ if and only $\Delta_{G'} = \emptyset$.

(53) a. $\uparrow = \downarrow$ removal: For any k-bounded LFG grammar G there is a co-empty $\uparrow = \downarrow$-free k-bounded grammar $G^{\setminus\uparrow=\downarrow}$.

b. Zipper removal: For any $\uparrow = \downarrow$-free k-bound LFG grammar G there is a co-empty 1-bounded (zipper-free) LFG grammar G^z.[20]

c. Annotation removal: For any 1-bounded LFG grammar G there is a co-empty annotation-free grammar G^a, and G^a is context-free.

Applying these transformations in sequence to an arbitrary k-bounded LFG grammar G results in a co-empty context free grammar $G^* = G^{\setminus\uparrow=\downarrow,z,a}$ whose size $|G^*|$ is a polynomial function of $|G|$. The string-set $L(G) = \emptyset$ if and only if the context free language $L(G^*) = \emptyset$, and this can be determined by the well-known emptiness algorithm for context free grammars, which is polynomial in the size of the grammar.

For (53a), the $\uparrow = \downarrow$ annotations in an arbitrary k-bounded grammar G are eliminated by replacing each $\uparrow = \downarrow$-annotated category in one rule with the right-side of each of the rules that expand that category. Let R be the smallest set that includes the rules of G and is closed under the convention (54). In this template δ, θ, and ψ are strings of annotated categories, and α may be a set of annotations with $\uparrow$ substituted for $\downarrow$.

(54) If R contains rules of the form

$$\mathrm{A} \rightarrow \delta \underset{\substack{\uparrow=\downarrow \\ \alpha}}{\mathrm{B}} \theta \quad \text{and} \quad \mathrm{B} \rightarrow \psi$$

then R also contains the rule $\mathrm{A} \rightarrow \delta \underset{\alpha}{\psi} \theta$

The $\uparrow = \downarrow$-free grammar $G^{\setminus\uparrow=\downarrow}$ is constructed by removing from R any rules with $\uparrow = \downarrow$ annotations. Note that a replacement sequence can never be longer than the

[20] Unlike the transformations that are often used in proofs of other formal-language properties, zipper removal does not preserve the language $L(G)$: the grammars G and G' generally are not weakly equivalent.

limit on the number of nodes in a functional domain, the parameter h of condition (48b). As a consequence, the growth of the grammar is bounded by a polynomial in $|G|$. Moreover, the resulting grammar $G^{\setminus\uparrow=\downarrow}$ accepts exactly the same strings as G and assigns them the same f-structures, although with c-structures that are not as deep.

For (53b), the rules of a zipper-free 1-bounded grammar are created from sets of up to k rules of a $\uparrow=\downarrow$-free k-bounded grammar G. The zipper daughters, occurrences of right-side categories with the same function assignments, are replaced with a single new daughter labeled by the concatenation (notated with $\cdot$) of the labels of the zipper daughters and annotated with the union of the zipper-daughter annotations. Let R now be the smallest set that includes the rules of a $\uparrow=\downarrow$-free grammar G and is closed under the following:

(55) a. If $A_1 \rightarrow \delta_1,\ ...,\ A_j \rightarrow \delta_j \quad (1 \leq j \leq k)$ are rules in R,
then R also contains the rule $A_1 \cdot ... \cdot A_j \rightarrow \delta_1 \ ... \ \delta_j$

b. If R contains a rule of the form

$$\begin{array}{lllllll} A \rightarrow & \delta & B_1 & \theta & B_2 & \psi \\ & & (\uparrow \textsc{f})=\downarrow & & (\uparrow \textsc{f})=\downarrow & \\ & & \alpha_1 & & \alpha_2 & \end{array}$$

then R also contains the rule

$$\begin{array}{lllllll} A \rightarrow & B_1 \cdot B_2 & \delta & \theta & \psi \\ & (\uparrow \textsc{f})=\downarrow & & & \\ & \alpha_1 \ \alpha_2 & & & \end{array}$$

The zipper-free grammar G^z is then created by removing from R any rule with multiple assignments for the same function or with annotations that are locally unsatisfiable. Local (within-rule) satisfiability of a rule with n daughters is tested by instantiating all metavariables with distinct constants $b_0, b_1, ..., b_n$ that stand for a putative mother node and its daughters. b_0 is substitute for $\uparrow$ in all annotations and b_i is substituted for $\downarrow$ in the annotations of the i^{th} daughter. The local f-description thus created is then solved using standard deductive-closure techniques.

The size of a zipper-free grammar G^z is exponential in k but polynomial in $|G|$, because there are at most $|G|^k$ rule combinations that must be considered. For every derivation in G of a string s with discontinuous subtrees for a particular grammatical function there is a corresponding derivation in G^z that assigns the same f-structure to a string s^z. The two strings contain the same words but not necessarily in the same order: the words are permuted so that the words of discontinuous subtrees for s are contiguous in s^z.

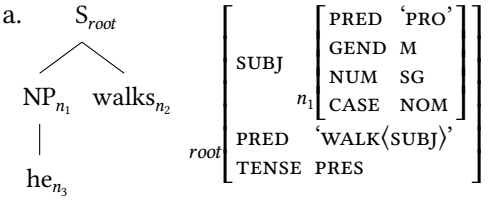
The annotation-removal transformation (53c) is based on the fact that atomic values in a 1-bounded grammar can only propagate between mothers and daughters within a single subtree. This is because, by definition, there are no nodes n and n' in separate subtrees with $\phi(n) = \phi(n')$. Atomic values in sister subtrees may have different values for a particular feature, but that can only result in an overall unsatisfiable f-description if annotation chains relative to a common mother put them in contact. Chains of atom-value annotations carried by the categories of a 1-bounded LFG derivation can be simulated by an elaborated set of refined c-structure categories in a corresponding annotation-free derivation. An annotation-free derivation is context-free and will fail if and only if the f-description for the 1-bounded LFG derivation is unsatisfiable.

The $\uparrow = \downarrow$-free and zipper-free rules in (56) provide the derivation (57a) for the sentence *He walks.*

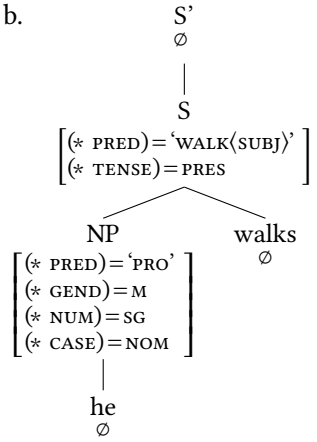
(56)
$$\begin{array}{lcccc} \text{S} \rightarrow & \text{NP} & \text{walks} & \text{NP} \rightarrow & \text{he} \\ & (\uparrow \textsc{subj}) = \downarrow & (\uparrow \textsc{pred}) = \text{`}\textsc{walk}\langle \textsc{subj} \rangle\text{'} & & (\uparrow \textsc{pred}) = \text{`}\textsc{pro}\text{'} \\ & (\downarrow \textsc{case}) = \textsc{nom} & (\uparrow \textsc{tense}) = \textsc{pres} & & (\uparrow \textsc{gend}) = \textsc{m} \\ & & (\uparrow \textsc{subj num}) = \textsc{sg} & & (\uparrow \textsc{num}) = \textsc{sg} \\ & & & & (\uparrow \textsc{case}) = \textsc{nom} \end{array}$$

The f-description is satisfiable because the case assigned to the subject NP matches the case of *he*, and the subject's number, entailed by the combination $(\uparrow \textsc{subj}) = \downarrow$ and $(\uparrow \textsc{subj num}) = \textsc{sg}$, also matches the number of *he*. The connection between the S and NP feature annotations is simulated by the refined NP category in (57b).

(57) a.



b.



Starting from a new category S', tree (57b) is the context free derivation provided by the category-refined, annotation-free rules in (58).

(58) $\underset{\emptyset}{\text{S'}} \rightarrow \underset{\begin{bmatrix}(\uparrow \text{PRED}) = \text{`WALK}\langle\text{SUBJ}\rangle\text{'} \\ (\uparrow \text{TENSE}) = \text{PRES}\end{bmatrix}}{\text{S}}$

$$\underset{\begin{bmatrix}(\uparrow \text{PRED}) = \text{`WALK}\langle\text{SUBJ}\rangle\text{'} \\ (\uparrow \text{TENSE}) = \text{PRES}\end{bmatrix}}{\text{S}} \rightarrow \underset{\begin{bmatrix}(\uparrow \text{PRED}) = \text{`PRO'} \\ (\uparrow \text{GEND}) = \text{M} \\ (\uparrow \text{NUM}) = \text{SG} \\ (\uparrow \text{CASE}) = \text{NOM}\end{bmatrix}}{\text{NP}} \underset{\emptyset}{\text{walks}} \qquad \underset{\begin{bmatrix}(\uparrow \text{PRED}) = \text{`PRO'} \\ (\uparrow \text{GEND}) = \text{M} \\ (\uparrow \text{NUM}) = \text{SG} \\ (\uparrow \text{CASE}) = \text{NOM}\end{bmatrix}}{\text{NP}} \rightarrow \underset{\emptyset}{\text{he}}$$

Note that the c-structure derivation for the string *Him walks* would have an unsatisfiable f-description. The corresponding category mismatch excludes a derivation with refined categories.

For an arbitrary 1-bounded grammar G the co-empty annotation-free grammar G^a produces derivation trees whose nodes are labeled with refined categories of this form. A refined category is a pair $c{:}m$ consisting of a c-structure category label c of G together with a refinement matrix m of atom-value feature specifiers $(\uparrow \text{P Q R} \ldots) = \text{V}$. The feature specifiers simulate in a G^a derivation the possible interactions of atomic values in the f-description of a corresponding G derivation, as illustrated. Importantly, Wedekind & Kaplan 2020 show that a finite set of specifiers is sufficient to simulate all possible atom-value interactions. These are the specifiers containing no more than ℓ of G's attributes, where ℓ is the number of attributes in the longest atom-value assignment in G.

Let N be the smallest set of refined categories and let R be the smallest set of refined rules, rules with refined-category labels, that are closed under the following conditions (see Wedekind & Kaplan 2020 for additional technical details).

(59) a. If S is the start symbol of G and S' is a category distinct from other G categories, N contains S:∅ and S':∅ and R contains S':∅ → S:∅.

b. If *term* is a terminal symbol of G, N contains *term*:∅.

c. If r is a refinement of a rule $A_0 \rightarrow \underset{\alpha_1}{A_1} \ldots \underset{\alpha_n}{A_n}$ of G with a sequence of refined categories $A_0{:}m_0, \ldots, A_n{:}m_n$ in N, then R contains r and N contains the refined categories of r.

The refinement of a rule $A_0 \rightarrow \underset{\alpha_1}{A_1} \ldots \underset{\alpha_n}{A_n}$ of G with a sequence of refined categories $A_0{:}m_0, \ldots, A_n{:}m_n$ is produced by instantiating the ↑ and ↓ metavariables with distinct mother-daughter constants $b_0, b_1, \ldots, b_n$, as above, but also including in the local f-description atom-value equations instantiated from the feature-specifier matrices. The additional equations are created by substituting b_i for all of the asterisks in each m_i. A refined rule r is constructed if this augmented f-description

is satisfiable. Each category A_i in the original G rule (including the mother category) is replaced by a refined category $A_i{:}m'_i$ where the feature specifiers of m'_i are formed by substituting * for b_i in each length-limited atom-value equation $(b_i \text{ P Q R}...)=\text{V}$ that the f-description entails. The newly refined categories are added to N.

The annotation-free grammar G^a is then constructed in the following way. S':∅ is its starting category, its terminals categories are of the form *term*:∅ for each terminal *term* of G, and its context-free rules are constructed from the refined rules in R by using standard context-free algorithms to eliminate useless rules, those that cannot participate in successful derivations, and then removing their functional annotations. The context-free derivations in G^a correspond to all and only the c-structures of G with satisfiable f-descriptions. Because the feature specifiers in a refined category are limited in length by the grammar parameter ℓ, $|G^a|$ is only polynomially larger than $|G|$ and its emptiness can be determined in polynomial time. We also note that if the annotations are not removed from the useful rules of R, the set of f-structures for a grammar with those still-annotated rules will be exactly the f-structures of G.

Acknowledgments

This chapter has benefited from helpful comments and suggestions from John Maxwell, Mary Dalrymple, and three anonymous reviewers.

References

Berwick, Robert C. 1982. Computational complexity and Lexical-Functional Grammar. *American Journal of Computational Linguistics* 8(3–4). 97–109. DOI: 10.3115/981923.981926.

Bresnan, Joan, Ash Asudeh, Ida Toivonen & Stephen Wechsler. 2016. *Lexical-Functional Syntax*. 2nd edn. (Blackwell Textbooks in Linguistics 16). Malden, MA: Wiley-Blackwell.

Bresnan, Joan, Ronald M. Kaplan, Stanley Peters & Annie Zaenen. 1982. Cross-serial dependencies in Dutch. *Linguistic Inquiry* 13(4). 613–635. https://www.jstor.org/stable/4178298. Reprinted in Savitch, Bach, Marsh & Safran-Naveh (1987: 286-319).

Chomsky, Noam. 1953. Systems of syntactic analysis. *Journal of Symbolic Logic* 18(3). 242–256. DOI: 10.2307/2267409.

Crouch, Richard, Mary Dalrymple, Ronald M. Kaplan, Tracy Holloway King, John T. Maxwell III & Paula Newman. 2008. *XLE Documentation*. Xerox Palo Alto Research Center. Palo Alto, CA. https://ling.sprachwiss.uni-konstanz.de/pages/xle/doc/xle_toc.html.

Culy, Christopher D. 1985. The complexity of the vocabulary of Bambara. *Linguistics and Philosophy* 8. 345–351.

Dalrymple, Mary (ed.). 1999. *Semantics and syntax in Lexical Functional Grammar: The resource logic approach* (Language, Speech, and Communication). Cambridge, MA: The MIT Press. DOI: 10.7551/mitpress/6169.001.0001.

Dalrymple, Mary. 2001. *Lexical Functional Grammar* (Syntax and Semantics 34). New York: Academic Press. DOI: 10.1163/9781849500104.

Dalrymple, Mary, Ronald M. Kaplan & Tracy Holloway King. 2015. Economy of Expression as a principle of syntax. *Journal of Language Modelling* 3(2). 377–412. DOI: 10.15398/jlm.v3i2.82.

Dalrymple, Mary, Ronald M. Kaplan, John T. Maxwell III & Annie Zaenen (eds.). 1995. *Formal issues in Lexical-Functional Grammar*. Stanford: CSLI Publications.

Dalrymple, Mary, John Lamping & Vijay Saraswat. 1993. LFG semantics via constraints. In *Proceedings of the 6th conference of the European chapter of the ACL (EACL 1993)*, 97–105. Association for Computational Linguistics. DOI: 10.3115/976744.976757.

Fang, Ji & Peter Sells. 2007. A formal analysis of the verb copy construction in Chinese. In Miriam Butt & Tracy Holloway King (eds.), *Proceedings of the LFG '07 conference*, 198–213. Stanford: CSLI Publications.

Halvorsen, Per-Kristian & Ronald M. Kaplan. 1988. Projections and semantic description in Lexical-Functional Grammar. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, 1116–1122. Tokyo. Reprinted in Dalrymple, Kaplan, Maxwell & Zaenen (1995: 279–292).

Haug, Dag & Tatiana Nikitina. 2012. The many cases of non-finite subjects: The challenge of "dominant" participles. In Miriam Butt & Tracy Holloway King (eds.), *Proceedings of the LFG '12 conference*, 292–311. Stanford: CSLI Publications.

Jaeger, Efrat, Nissim Francez & Shuly Wintner. 2005. Unification grammars and off-line parsability. *Journal of Logic, Language and Information* 14(2). 199–234. DOI: 10.1007/s10849-005-4511-1.

Johnson, Mark. 1986. The LFG treatment of discontinuity and the double infinitive construction in Dutch. In *Proceedings of the 5th West Coast Conference on Formal Linguistics*, 102–118. Stanford: CSLI Publications.

Johnson, Mark. 1988. *Attribute-value logic and the theory of grammar*. Stanford: CSLI Publications.

Kallmeyer, Laura. 2010. On mildly context-sensitive non-linear rewriting. *Research on Language and Computation* 8. 341–363. DOI: 10.1007/s11168-011-9081-6.

Kaplan, Ronald M. 1987. Three seductions of computational psycholinguistics. In Peter Whitelock, Mary McGee Wood, Harold L. Somers, Rod Johnson & Paul Bennett (eds.), *Linguistic theory and computer applications*, 149–188. London: Academic Press. Reprinted in Dalrymple, Kaplan, Maxwell & Zaenen (1995: 339–367).

Kaplan, Ronald M. 2019. Formal aspects of underspecified features. In Cleo Condoravdi & Tracy Holloway King (eds.), *Tokens of meaning: Papers in honor of Lauri Karttunen*, 349–369. Stanford: CSLI Publications.

Kaplan, Ronald M. & Joan Bresnan. 1982. Lexical-Functional Grammar: A formal system for grammatical representation. In Joan Bresnan (ed.), *The mental representation of grammatical relations*, 173–281. Cambridge, MA: The MIT Press. Reprinted in Dalrymple, Kaplan, Maxwell & Zaenen (1995: 29–130).

Kaplan, Ronald M. & John T. Maxwell III. 1988. Constituent coordination in Lexical-Functional Grammar. In *COLING '88: Proceedings of the 12th Conference on Computational Linguistics*, 303–305. Budapest. DOI: 10.3115/991635.991696. Reprinted in Dalrymple, Kaplan, Maxwell & Zaenen (1995: 199–210).

Kaplan, Ronald M. & John T. Maxwell III. 1996. *LFG Grammar Writer's Workbench*. Xerox Palo Alto Research Center. Palo Alto, CA. https://www.researchgate.net/profile/John_Maxwell5/publication/2760068_Grammar_Writer's_Workbench/links/0c96052405e97928e9000000.pdf.

Kaplan, Ronald M. & Jürgen Wedekind. 2000. LFG generation produces context-free languages. In *COLING 2000 volume 1: The 18th International Conference on Computational Linguistics*, 425–431. DOI: 10.3115/990820.990882.

Kaplan, Ronald M. & Jürgen Wedekind. 2019. Tractability and discontinuity. In Miriam Butt, Tracy Holloway King & Ida Toivonen (eds.), *Proceedings of the LFG '19 Conference*, 130–148. Stanford: CSLI Publications.

Kaplan, Ronald M. & Jürgen Wedekind. 2020. Zipper-driven parsing for LFG grammars. In Miriam Butt & Ida Toivonen (eds.), *Proceedings of the LFG '20 conference*, 169–189. Stanford: CSLI Publications.

Kaplan, Ronald M. & Annie Zaenen. 1989. Long-distance dependencies, constituent structure, and functional uncertainty. In Mark Baltin & Anthony Kroch (eds.), *Alternative conceptions of phrase structure*, 17–42. Chicago: University of Chicago Press. Reprinted in Dalrymple, Kaplan, Maxwell & Zaenen (1995: 137–165).

Kaplan, Ronald M. & Annie Zaenen. Forthcoming. Long-distance dependencies. In Mary Dalrymple (ed.), *Handbook of Lexical Functional Grammar*. Berlin: Language Science Press.

Kuhn, Jonas. 2003. *Optimality-Theoretic Syntax – A declarative approach*. Stanford: CSLI Publications.

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

Maxwell, John T., III & Ronald M. Kaplan. 1996. Unification-based parsers that automatically take advantage of context freeness. In Miriam Butt & Tracy Holloway King (eds.), *Proceedings of the LFG '96 conference*, 1–31. Stanford: CSLI Publications.

Nishino, Tetsuro. 1991. Mathematical analysis of Lexical-Functional Grammars—complexity, parsability, and learnability. *Language Research* 27(1). 867–915.

Przepiórkowski, Adam & Agnieszka Patejuk. 2012. The puzzle of case agreement between numeral phrases and predicative adjectives in Polish. In Miriam Butt & Tracy Holloway King (eds.), *Proceedings of the LFG '12 conference*, 490–502. Stanford: CSLI Publications.

Roach, Kelly. 1983. LFG languages over a one-letter alphabet. Unpublished manuscript, Xerox Palo Alto Research Center.

Savitch, Walter J., Emmon Bach, William Marsh & Gila Safran-Naveh (eds.). 1987. *The formal complexity of natural language* (Studies in Linguistics and Philosophy). Dordrecht: Springer.

Seki, Hiroyuki, Ryuichi Nakanishi, Yuichi Kaji, Sachiko Ando & Tadao Kasami. 1993. Parallel multiple context-free grammars, finite-state translation systems, and polynomial-time recognizable subclasses of Lexical-Functional Grammars. In *Proceedings of the 31st annual meeting of the Association for Computational Linguistics*, 130–139. Columbus, OH: Association for Computational Linguistics. DOI: 10.3115/981574.981592.

Shieber, Stuart M. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy* 8(3). 333–343. DOI: 10.1007/978-94-009-3401-6_12.

Wedekind, Jürgen. 1999. Semantic-driven generation with LFG- and PATR-style grammar. *Computational Linguistics* 25(2). 277–281. https://www.aclweb.org/anthology/J99-2006.

Wedekind, Jürgen. 2014. On the universal generation problem for unification grammars. *Computational Linguistics* 40(3). 533–538. DOI: 10.1162/coli_a_00191.

Wedekind, Jürgen & Ronald M. Kaplan. 1996. Ambiguity-preserving generation with LFG- and PATR-style grammars. *Computational Linguistics* 22(4). 555–558. https://www.aclweb.org/anthology/J96-4005.

Wedekind, Jürgen & Ronald M. Kaplan. 2012. LFG generation by grammar specialization. *Computational Linguistics* 38(4). 867–915. DOI: 10.1162/coli_a_00113.

Wedekind, Jürgen & Ronald M. Kaplan. 2020. Tractable Lexical-Functional Grammar. *Computational Linguistics* 46(2). 515–569. DOI: 10.1162/coli_a_00384.

Wedekind, Jürgen & Ronald M. Kaplan. 2021. LFG generation from acyclic f-structures is NP-hard. *Computational Linguistics*. DOI: 10.1162/coli_a_00419.

Zaenen, Annie & Ronald M. Kaplan. 1995. Formal devices for linguistic generalizations: West Germanic word order in LFG. In Jennifer S. Cole, Georgia M. Green & Jerry L. Morgan (eds.), *Linguistics and computation*, 3–27. Stanford: CSLI Publications. Reprinted in Dalrymple, Kaplan, Maxwell & Zaenen (1995: 215–240).

Zweigenbaum, Pierre. 1988. *Attributive adjectives, adjuncts and cyclic f-structures in Lexical-Functional Grammar*. DIAM Rapport Interne RI-58a. Paris: Departement Intelligence Artificielle et Medecine.