Chapter 38

LFG and Dependency Grammar

Dag Haug
University of Oslo

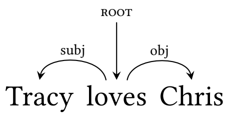
This chapter discusses Dependency Grammar from the perspective of LFG. I first introduce the key ideas behind Dependency Grammar and how they relate to LFG concepts. I then show how both LFGs and Dependency Grammars can be translated into Multiple Context-Free Grammars to study formal differences between the frameworks. Next I discuss two recent efforts to translate from LFG analyses to the version of Dependency Grammar adopted in Universal Dependencies. Finally I show how Glue semantics can be applied to dependency structures.

1 Introduction

Dependency Grammar (DG) is a tradition for syntactic analysis based on binary, asymmetric relations (called dependency relations or just *dependencies*) between words. These relations are typically labelled, giving rise to a set of labels that can be thought of as grammatical functions, which are of course also important in LFG. In fact, the correspondence between dependencies in DG and grammatical functions in LFG and their central role in both theories is the main similarity, formally and conceptually, between the two frameworks.

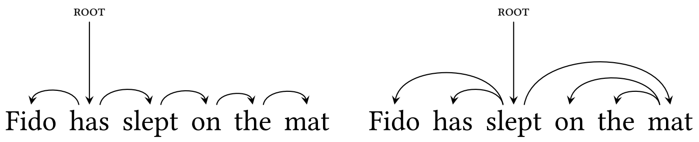
The primacy of dependencies is what holds together work in the DG tradition. As we will see, it is characteristic of almost all DG theories that they acknowledge a level of syntax that we will call the *core dependencies*. This is a set of dependencies restricted so as to form a tree over the words of a sentence, i.e. a structure where each word has exactly one head, except the root word, which has none (or equivalently, is attached to a synthetic root node). (1) shows a very simple example of this.

Dag Haug. 2023. LFG and Dependency Grammar. In Mary Dalrymple (ed.), *Handbook of Lexical Functional Grammar*, 1829–1859. Berlin: Language Science Press. DOI: 10.5281/zenodo.10186040

(1)


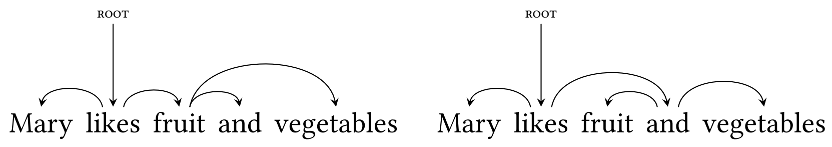
Most theoretical work and concrete analyses have seen the need to introduce additional mechanisms or levels of structure beyond core dependencies to give the theory more analytical bite; this goes all the way back to Tesnière (1959), the founding work of modern DG. However, there is typically little agreement about the additional mechanisms or levels of structure between individual scholars working in the DG tradition. So, while the core dependency representation is often acknowledged as theoretically inadequate, it has enjoyed considerable popularity as a simplified representation with practical applications in computational linguistics and natural language processing.

But even restricting attention to core dependencies, there are a number of choice points where different dependency frameworks make different decisions. For example, when the core dependencies model structures with a lexical word and one or more function words (for example, articles and nouns, auxiliaries and full verbs, or prepositions and their complements), we must take a stance on whether the lexical or the functional word is the head: the co-head option often used in LFG is not available. (2) shows what the (unlabelled) dependency structure of a simple sentence would look like if we take function words as heads (left) or lexical words as heads (right).

(2)


It is obviously not necessary to treat all function words the same, and so there are intermediate variants between these two extremes, taking for example prepositions and articles as heads, but not auxiliary verbs.

Another point at which dependency grammarians diverge is the treatment of coordination. Because coordination is normally thought of as symmetric, it is not easy to represent with directed dependencies. Here the most common competing analyses, shown in (3), involve taking the first conjunct as the head (left), which entails giving up on symmetry; or to make the conjunction the head (right) and maintain symmetry, but at the cost of dissociating the conjuncts from their normal head (e.g. the verb), which is the basis for most morphosyntactic and semantic constraints.

(3)

ROOT
Mary likes fruit and vegetables
ROOT
Mary likes fruit and vegetables

Faced with the choices illustrated in (2) and (3) many linguists in the DG tradition have felt that neither analysis is satisfactory, and they have therefore reacted by enriching the dependency formalism in various ways that result in data structures that have more in common with LFG. I discuss some key examples of this in Section 2. Even if much theoretical work in DG assumes such enriched data structures, most practical applications of DG rely on core dependencies, thereby forcing choices that, at least from an LFG perspective, are somewhat arbitrary.

One key difference between DG and LFG is that dependency grammarians typically do not formalize their work and in many cases do not provide (even informal) rules that generate the constructions they are interested in but content themselves with providing analyses of the whole structure. This goes back to the earliest dependency grammarians such as Tesnière, but has become even more prominent with the increasing use of dependency structures in data-driven parsing, where the goal is not to define a grammar that recognizes (or generates strings from) a formal language, but to parse strings into a single plausible structural representation. Nevertheless, it *is* possible to conceive of DGs as formal grammars. In Section 3 I discuss how this can be done using the framework of Multiple Context-Free Grammars. While this is not an approach that most dependency grammarians follow, it yields a useful framework for comparing DG and LFG. Another useful perspective on DG and LFG is offered by recent efforts to translate LFG resources into DG resources, which I discuss in Section 4. Section 5 explores the potential for combining dependency grammars with Glue semantics, the standard semantic framework in LFG.

2 The dependency grammar tradition and LFG

The idea of using binary, labelled, asymmetric relations to analyze syntax is found in the work of Pāṇini, Ancient Greek and Roman grammarians and the speculative grammarians of the Middle Ages (Covington 1984). On its own, this idea is too vague to define a theoretical framework and both Pāṇini and the speculative grammarians have also been seen as forerunners of generative grammar (Kiparsky 1993, Chomsky 1966). What defines the modern dependency grammar tradition, which started with Tesnière (1959), is the attempt to base syntax primarily, or even exclusively, on the concept of core dependencies, as opposed to the concept of constituency developed in American structuralism and the generative tradition. Although there have been a number of attempts to develop dependency grammar into a full-fledged grammatical theory (the most well-known ones being Functional Generative Description (Sgall et al. 1986); Meaning–Text Theory (MTT) (Mel'čuk 1988); and Word Grammar (Hudson 1984, 2010)), none of these are very widespread beyond the environments where they originated and hence there is no single, coherent version of DG as a formal framework. The focus of this section is therefore not to identify assumptions made in specific frameworks, but rather to compare ideas that are common in the dependency grammar tradition with LFG.

2.1 Dependency graphs and f-structures

There is an obvious similarity between dependencies, as found in DG, and the binary, labelled, asymmetric relations between the nodes of an LFG f-structure.[1] In both cases, the relations form a directed labelled graph over nodes corresponding to linguistic material. The similarity even extends to the set of labels used, which in both cases contain traditional grammatical functions such as subject and object. Formally, however, there are two important differences: First, the nodes of the f-structure are not words, but correspond to zero, one or several words/c-structure terminals. This is how LFG escapes the indeterminacy of direction of headedness in constructions which combine lexical and functional words that we saw in (2). Second, labelled dependencies are not necessarily functional, i.e. there may be two or more daughters bearing the same relation to the same head, in violation of LFG's uniqueness condition.[2]

In addition to these two formal differences, there are in practice many more differences, because DG analyses rarely use the full power of a directed graph and instead typically emphasize the core dependencies, which form a tree spanning the words of the sentence. To the extent that e.g. multiple heads are used, one of the heads is typically considered "primary". Even so, the formal similarities

[1]To emphasize the parallelism between f-structures and dependency graphs, we rely here on the graph-theoretic interpretation of attribute-value matrices, where feature structures and atomic values are nodes, and attributes are labelled edges between these nodes, and not the "official" interpretation of f-structures as functions (Kaplan 1995, Kaplan & Bresnan 1982). The graph-theoretic interpretation is standard in most other unification-based frameworks from Functional Unification Grammar (Kay 1979) onwards, and was, to my knowledge, first formalized by Moshier & Rounds (1987). It is used in HPSG (Richter 2021); see Przepiórkowski 2023: section 4 [this volume] for discussion of the differences between the two views.

[2]LFG can deal with several dependents bearing the same relation by using set-valued attributes e.g. for ADJUNCT; this introduces the concept of sets, which also has no counterpart in DG.

between dependencies and f-structures mean that similar theoretical questions can arise in both DG and LFG and even that one can think of LFG's f-structures as dependency graphs that take a particular view on certain foundational questions in DG.[3]

An overarching question in the DG tradition (see e.g. de Marneffe & Nivre 2019: 199f.) is whether dependency relations are sufficient for analyzing syntax. In one sense, the answer is obviously no. Like f-structures, dependency structures say nothing about word order. This is dealt with in the c-structure in LFG, and scholars within dependency grammar have also seen the need to enrich the theory with a mechanism for constraining word order. I return to this in Section 3. But more fundamentally, one might ask whether core dependencies, tree structures over words, are sufficient to capture functional aspects of syntax like f-structures do in LFG.

In fact, it is not too hard to see that core dependencies cannot fully represent the functional relations of a sentence. Consider for example, the subject in a raising construction.

(4) It seems to rain.

The expletive *it* bears a functional relation to the raising verb *seems* as witnessed by agreement; but the form of the expletive is licensed by the lexical verb *rain* (and would be different in e.g. *There seems to be a problem*), giving evidence for a second functional relation. If one insists on core dependencies, one of the two relations must be privileged.

The alternative is to increase the expressivity of the theory, and this is in fact what Tesnière did when he introduced two other kinds of relations beside dependencies that can hold between words, namely junction (*jonction*) and transfer (*translation*). Junction is the relation that holds between coordinated items that are either dependents of the same head or heads of the same dependent. Translation is the relation that holds between lexical words and functional words that license their appearing in various dependencies. For example, complementizers "translate" verbs so as to license their appearing in object position according to the analysis in Tesnière (1959: 24); similar analyses are given for determiners and adpositions.

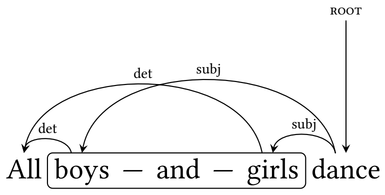
Crucially, words that are linked by junction or transfer form a complex node (*nucleus dissocié*) in the dependency graph and jointly contract dependency relations. In this way, their dependents end up having more than one head; and they

[3]Furthermore, on the implementation side, Bröker (1998) shows how DGs can be encoded as LFGs and implemented in the XLE platform.

can collectively bear a single dependency relation to their head. In this respect, Tesnière's analyses are in fact quite close to standard LFG f-structures, where coordination is analyzed in terms of a set-valued attribute (5) and function words such as e.g. auxiliaries form a single f-struture node with their lexical verb (6).

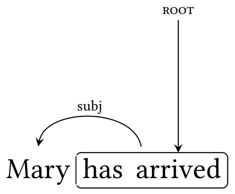
(5) a. All boys and girls dance.

b. $$\left[\begin{array}{ll} \textsc{pred} & \text{‘}\textsc{dance}\langle\textsc{subj}\rangle\text{’} \\ \textsc{subj} & \left\{\begin{array}{l} \left[\begin{array}{ll} \textsc{pred} & \text{‘}\textsc{boy}\text{’} \\ \textsc{spec} & \boxed{1}\left[\begin{array}{ll}\textsc{pred} & \text{‘}\textsc{all}\text{’}\end{array}\right] \end{array}\right] \\ \\ \left[\begin{array}{ll} \textsc{pred} & \text{‘}\textsc{girl}\text{’} \\ \textsc{spec} & \boxed{1} \end{array}\right] \end{array}\right\} \end{array}\right]$$

c. 

(6) a. Mary has arrived.

b. $$\left[\begin{array}{ll} \textsc{pred} & \text{‘}\textsc{arrive}\langle\textsc{subj}\rangle\text{’} \\ \textsc{tense} & \textsc{perfect} \\ \textsc{subj} & \left[\begin{array}{ll}\textsc{pred} & \text{‘Mary’}\end{array}\right] \end{array}\right]$$

c. 

In this respect, both Tesnière's theory and LFG's f-structure reject the idea that syntactic dependencies can be adequately captured in a tree structure over the words of a sentence. Nevertheless, LFG's approach is much more general than Tesnière's. Tesnière allows many-to-one relations between words and dependency nodes based on relations that are not dependencies, but he maintains the tree structure over dependency nodes. Therefore, the only way a word can have two heads is if those heads form a single node by junction or transfer, as in (5c) and (6c); but LFG also allows for a word to have two heads that do not form a group, as in the analysis of functional control verbs (7).

(7) a. Chris persuaded Mary to come

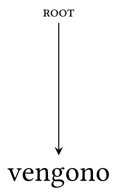
b. $$\left[\begin{array}{ll} \textsc{pred} & \text{‘}\textsc{persuade}\langle\textsc{subj, obj, xcomp}\rangle\text{’} \\ \textsc{subj} & \left[\textsc{pred}\ \text{‘Chris’}\right] \\ \textsc{obj} & \boxed{1}\left[\textsc{pred}\ \text{‘Mary’}\right] \\ \textsc{xcomp} & \left[\begin{array}{ll} \textsc{pred} & \text{‘}\textsc{come}\langle\textsc{subj}\rangle\text{’} \\ \textsc{subj} & \boxed{1} \end{array}\right] \end{array}\right]$$

Such dependencies cannot be expressed in Tesnière’s formalism, because *persuade* and *come* share the dependent *Mary*, despite not forming a group. Moreover, *Mary* bears a different syntactic relation to each of them, which again is not possible in Tesnière’s formalism. More recent versions of dependency grammar have typically accounted for control and raising verbs by positing more levels of representation, see Section 2.2.

Finally, an important difference between Tesnière’s dependency graphs and f-structures is that f-structures may contain nodes that correpond to no overt word. A typical case is pro-drop, as in (8) from Italian.

(8) a. vengono
come-PRS.3PL

b. $$\left[\begin{array}{ll} \textsc{pred} & \text{‘}\textsc{come}\langle\textsc{subj}\rangle\text{’} \\ \textsc{subj} & \left[\textsc{pred}\ \text{‘}\textsc{pro}\text{’}\right] \end{array}\right]$$

c. 

Again, Tesnière’s formalism cannot capture this: dependency nodes may correspond to one word, or more words if they form a group by junction or transfer, but not to zero. The strategy in later versions of DG has been the same as that used to address phenomena where LFG uses structure sharing, namely to introduce more levels of representation.

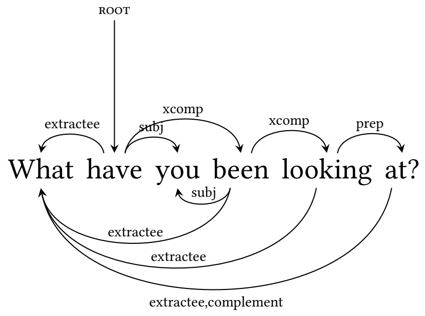
In sum, one can say from an LFG perspective that Tesnière’s dependency graphs, while certainly more expressive than core dependencies, are insufficiently general to deal with the complex functional relations that exist in natural language sentences.

2.2 Other levels of syntactic representation

Tesnière's strategy was to enrich dependency graphs so as to be able to represent more functional relations than core dependencies can do. More recent versions of DG have instead opted to keep the core dependencies simple and instead go beyond a single level of grammatical description to accommodate more information. One prominent example is the so-called tectogrammatical layer found in Functional Generative Description (Sgall et al. 1986) and the associated Prague Treebanks (Hajič et al. 2020). This layer is annotated with an enriched dependency tree that will contain nodes that do not correspond to words (e.g. pro-dropped subjects) and secondary edges capturing multiple head-phenomena such as control.[4]

Melčuk's Meaning–Text Theory explicitly distinguishes a deep syntactic level between the semantic level and surface syntax. However, as pointed out by Kahane (2003), the deep syntactic level is the least defined level of MTT and it is not clear how much information it is supposed to contain. What is clear, however, is that grammatically imposed coreference relations are resolved in deep syntax, opening up a way to deal with, e.g., control.

In Word Grammar (Hudson 1984, 2010), too, control is treated by loosening the tree constraint on dependency structures. Example (9), from Hudson (2003: 521), illustrates how structure sharing is used to analyze raising (*you* shared by *have* and *been*)[5] and extraction (*what* shared by *have*, *been*, *looking* and *at*).

(9)


[4]The status of the tectogrammatical layer is not entirely clear: the Prague Dependency Treebank annotation guidelines (https://ufal.mff.cuni.cz/pdt2.0/doc/manuals/en/t-layer/html/ch02.html) say that it "represents the semantic structure of the sentence", but Hajič et al. (2020) describe it as "deep syntax". The difference may be merely terminological.

[5]Instead of Hudson's *sharer*, I have used the LFG relation XCOMP which Hudson explicitly mentions as an alternative name for the same concept. The diagram in Hudson (2003: 521) does not have a subject relation between *you* and *looking*, although *looking* is an XCOMP of *been*. It is unclear whether this is just an error.

The dependency graph in (9) is essentially identical to the standard LFG analysis (except that in extraction, LFG usually has structure sharing only between the gap and the filler position, without involving the intermediate f-structures). However, in Word Grammar, the edges above and below the words have different status:

> This diagram also illustrates the notion 'surface structure' […]. Each dependency is licensed by the grammar network, but when the result is structure-sharing just one of these dependencies is drawn above the words; the totality of dependencies drawn in this way constitutes the sentence's surface structure. In principle any of the competing dependencies could be chosen, but in general only one choice is compatible with the 'geometry' of a well-formed surface structure, which must be free of 'tangling' (crossing dependencies – i.e. discontinuous phrases) and 'dangling' (unintegrated words). There are no such constraints on the non-surface dependencies." (Hudson 2003: 521)

This illustrates the point that I made in the introduction: different varieties of dependency grammar may have different notions of "deep syntax", but they all share the idea that there is an interesting representation of syntactic dependencies that is a rooted tree over nodes that stand in a one-to-one correspondence with the words of the sentence. This is very different from LFG: all edges of an f-structure graph are equal. The subject edge that connects the subject of a control construction to the control verb has exactly the same status as the subject edge that connects the subject to the non-finite verb. Thus, there is no "privileged subgraph" of the f-structure that forms a rooted tree over the words. By contrast, Hudson's distinction between the surface structure and the non-surface dependencies gives rise to such a privileged subgraph, although it must be said that the distinction between surface and non-surface dependencies is not further developed in Word Grammar.

Dependency grammars also differ in their treatment of "null words", i.e. cases where LFG would have an f-structure node that does not correspond to any surface word, as in e.g. pro-drop. Most dependency analyses would simply leave out such subjects, as we saw in (8). But here too, many dependency grammars introduce the missing subjects in "deeper" projections, for example in the tectogrammatical layer of Functional Generative Description. In fact, Word Grammar is one of the few dependency grammar frameworks that acknowledge empty elements in the core syntactic graph. Creider & Hudson (2006) present an argument for this that runs along standard lines of LFG thinking. In Ancient Greek, predicate nouns and adjectives agree in case (and adjectives also in number and gender) with their subjects; and subjects of infinitives are in the accusative.

(10) Ancient Greek (Xenophon, Anabasis 1.3.6)

nomízo:	gàr	humâ:s	emoì	eînai	kaì	patrída	kaì	phílous
think-1.PRS	for	you-ACC	me-DAT	be-INF	and	fatherland-ACC	and	friends-ACC

'For I think you are to me both fatherland and friends'

But crucially, the predicative is accusative also when the accusative subject is absent (11), even in cases where there is a coreferential element in the higher clause (12).

(11) Ancient Greek (Isocrates 2.15)

philánthro:pon	eînai	deî
humane-ACC	be-INF	must

'one must be humane'

(12) Ancient Greek (Plato, Alcibiades 2, 141a7)

exarkései	soi	túrannon	genésthai
suffice-FUT	you-DAT	king-ACC	become-INF

'it will be enough for you to become king'

In (12), we observe that the predicate noun *turannon* does not agree directly with its logical subject *soi*, but rather with the unexpressed subject of the infinitive. Since case agreement is generally agreed to be syntactic (whereas agreement in number and gender could potentially be semantic), Creider & Hudson (2006) conclude that the unexpressed subject of the infinitive must nevertheless be present in the syntax. This is unsurprising from an LFG point of view, but does not seem to be generally accepted in DG. It is unclear, for example, how Functional Generative Description would deal with this kind of data, since null words are inserted only at the tectogrammatical layer, where there is no case feature.

3 Word order and generative power in DG and LFG

In most versions of dependency grammar, it is assumed that the nodes of a dependency structure are not linearly ordered in themselves: a dependency relation implies no particular linear order between a head and its dependents, but can be related to different surface linearizations. This view goes back to Tesnière (1959: chapter 7), who distinguishes sharply between structural order (dependencies) and linear order. The main exception to this is Functional Generative Description, which assumes a linear order on the nodes even in the tectogrammatical layer, to capture information structure.

But even if the nodes of the dependency structures are not linearly ordered, it is possible (and in fact necessary for most languages) to constrain the relation between dependency structure and linearization. One much-discussed constraint is *projectivity*.[6]

(13) A dependency graph is projective iff for every edge $n_h \rightarrow n_d$ it contains, n_h dominates all nodes that occur between n_h and n_d (where domination is the transitive closure of the edge/dependency relation)

An early result due to Gaifman (1965) is that projective dependency grammars are weakly equivalent to context-free grammars.[7] This result may in fact have led to a lack of interest in dependency grammar because it was widely believed in the sixties and seventies (and eventually proved in the eighties) that natural languages are *not* context-free. On the other hand, the recognition problem for a dependency grammar with no linearization constraints at all (thus allowing arbitrary discontinuities) is NP complete (Neuhaus & Bröker 1997).[8]

With the increasing popularity of dependency grammars in the 2000s, this led to the search for *intermediate* linearization constraints between strict projectivity and arbitrary non-projectivity. One important class of constraints is based on the notion of *block degree* (Holan et al. 1998). Intuitively, projectivity as defined in (13) ensures that the subgraph of n_h (i.e. n_h and the set of nodes it dominates) forms a single block of adjacent nodes. We can instead allow the subgraph to form *two* blocks of adjacent nodes, interrupted by a continuous set of words. We say that n_h has block degree 2; and the block degree of a dependency tree is the highest block degree of any of its nodes. Equivalently, we can speak of gap degree, which is block degree minus 1 (i.e., the number of allowed gaps). (14) illustrates this with an example from Latin.

[6]It seems that this term originated with a technical report by P. Ihm and Y. Lecerf "Eléments pour une grammaire générale des langues projectives", Bruxelles 1960, but I have been unable to find this paper.

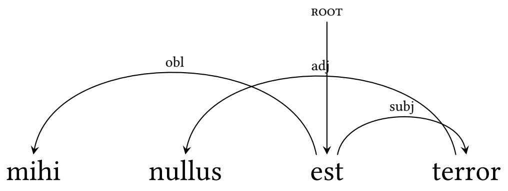
[7]See also Hays (1964).

[8]As we will see in Section 4, this is not an issue in data-driven parsing, which sidesteps the recognition problem and aims directly at providing a contextually plausible parse.

(14) a. Latin

Mihi	nullus	est	terror
me.DAT	none.NOM	is	fear.NOM

'I have no fear.'

b.


The gap degree of *est* is 0, since its subgraph is continuous; but the gap degree of *terror* is 1, since there is one gap in its subgraph – *est* intervenes between *terror* and *nullus*, but is not dominated by *terror*. As a result, the gap degree of the whole tree is 1.

To study the computational complexity of the dependency grammars that could generate structures like (14), and their relationship to LFG grammars, it is convenient to use phrase structure-based systems that allow discontinuities, so-called Linear Context-Free Rewriting Systems (LCFRS, Vijay-Shanker et al. 1987) or the notational variant Multiple Context-Free Grammars (MCFG, Seki et al. 1991). The MCFG formalism is a generalization of CFG which retains ordinary CFG productions for the expression of categorial structure, but uses explicit *yield functions* to compute the yield of the mother node from the yields of the daughters. In an ordinary CFG, yield computation is conflated with category formation: a rule such as DP → D NP says both that the category DP is formed of a D and an NP, and that the yield of the resulting DP is formed by concatenating the yields of D and NP. In effect, then, a CFG can be seen as an MCFG with concatenation as the only yield function.[9]

To allow for greater expressivity, MCFG allows yields to be *tuples* of strings. For example, we may want to say that the yield of DP is a pair (2-tuple) consisting of the yields of D and NP. This pair will then be the input to further yield functions that apply to productions with DP on the right-hand side. More generally, we may allow yields to be *n*-tuples of strings. The interesting point is that there is a close correspondence between yield components in an MCFG and blocks in a corresponding dependency structure. We can extract MCFG rules from dependency trees, as shown in Kuhlmann (2013), where a formal exposition is given. Here I just provide an intuitive understanding of how the tree in (14b) gives rise to the rules in Table 1.

[9]See Clark (2014) for an accessible introduction for linguists and Kallmeyer (2010: chapter 6) for a more formal introduction.

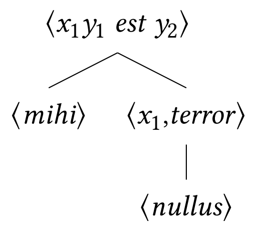
Table 1: Rules extracted from the tree in (14b)

rule	yield function	compact notation
ADJ $\rightarrow g()$	$g = \langle \textit{nullus} \rangle$	ADJ $\rightarrow \langle \textit{nullus} \rangle$
OBL $\rightarrow h()$	$h = \langle \textit{mihi} \rangle$	OBL $\rightarrow \langle \textit{mihi} \rangle$
SUBJ $\rightarrow i($ADJ$)$	$i = \langle x_1, \textit{terror} \rangle$	SUBJ $\rightarrow \langle x_1, \textit{terror} \rangle$ (ADJ)
ROOT $\rightarrow j($OBL SUBJ$)$	$j = \langle x_1 y_1 \textit{ est } y_2 \rangle$	ROOT $\rightarrow \langle x_1 y_1 \textit{ est } y_2 \rangle$(OBL SUBJ)

Looking at *nullus* in (14b), we see that it has no dependents, hence the right-hand side of the first rule is a constant function which fixes the yield to the string *nullus*, and similarly for *mihi*. For *terror*, things are more interesting. It takes one dependent, an ADJ, and hence its yield function i depends on the value of that argument. Concretely, the yield of the node *terror* is a tuple, consisting of the yield of the ADJ dependent which is represented as x_1,[10] and the string *terror*. Finally, the verb takes two arguments, SUBJ and OBL. The yield is constructed by concatenating the yield of the OBL (i.e. x_1), the first component of the SUBJ (i.e. y_1), the string *est*, and the second component of SUBJ (y_2).

With the rules in Table 1, we can construct the MCFG derivation tree in (15).

(15)



But notice that because the MCFG grammar is lexical, i.e. each rule introduces exactly one lexical item, the tree in (15) is isomorphic to the dependency tree in (14b). In other words, a lexicalized MCFG can simply be interpreted as a dependency grammar which simultaneously restricts word order.

This allows us to compare the generative capacity and the parsing complexity of dependency grammars with other formalisms. Under a reasonable constraint on discontinuities,[11] the expressivity of an MCFG depends only on the maximal

[10]The convention is that we use x for the yield of the first dependent and y for the yield of the second dependent, and subscript those variables with an index referring to blocks of the yield.

[11]Namely wellnestedness; a tree is wellnested if there are no disjoint subtrees that overlap linearly.

block degree of the grammar, giving rise to a hierarchy of *k*-MCFGs, where *k* is the block degree of the most complex yield function in the grammar. It turns out that 2-MCFGs (and hence dependency grammars that allow maximally one gap) are weakly equivalent to Tree Adjoining Grammars and 'classical' Combinatory Categorial Grammar, as was proven by Bodirsky et al. (2005).[12]

Even more interesting from an LFG perspective, there is also a result that a subclass of LFG grammars, so-called *finite copying LFGs*, can be translated into weakly equivalent MCFGs/LCFRSs (Seki et al. 1993). Finite copying LFGs are quite restricted in what functional annotations they allow, in particular they do not allow head annotations (↑=↓) or reentrancies, and also impose the crucial constraint that the grammar puts an upper bound on the number of c-structure nodes corresponding to a single f-structure. Wedekind & Kaplan (2020) show that we can impose this upper bound while still allowing head annotations and reentrancies, as long as they are nonconstructive. This allows most functional equations that are used in linguistic work, including functional control equations of the type (↑ F G)=(↑ H). Wedekind & Kaplan (2020) call these grammars *k*-bounded LFGs and prove that for any *k*-bounded LFG, a weakly equivalent *k*-MCFG can be constructed. Moreover, the MCFG rules can be annotated with functional descriptions that allow us to construct the f-structure that the corresponding *k*-bounded LFG assigns to the sentence, yielding a strongly equivalent MCFG.

These results allow us to compare dependency grammars and LFGs in a precise way. First of all, dependency grammars and *k*-bounded LFGs are weakly equivalent. Nevertheless, although strongly equivalent MCFGs can be constructed from both dependency grammars and *k*-LFGs, it is not the case that we can construct a strongly equivalent dependency grammar from an LFG. The interpretation of an MCFG as a dependency grammar relies on unique lexicalization: each rule contains a single lexical item interpreted as the head. The MCFGs that Wedekind & Kaplan (2020) construct from LFGs are not lexicalized in this way. They do contain functional descriptions that allow us to identify the head but, since LFG allows co-heads, the head is not guaranteed to be unique. Moreover, the functional descriptions in the MCFG constructed from an LFG may contain reentrancies, i.e. words having more than one head, which have no interpretation on the dependency grammar side, thus losing information. A final, minor point is that Kuhlmann's interpretation of MCFGs as dependency grammars say nothing about edge labels; it would be natural and straightforward to interpret LFG's ordinary function assignments as such labels.

[12]See also Kuhlmann (2007, 2010).

In sum, then, the formal analysis tells us that the difference between k-bounded LFGs and dependency grammars resides exactly in the availability of co-heads and reentrancies, which provide important information from a linguistic point of view. Finally, it should be noted that the restriction to k-bounded LFGs, while preserving coverage of many, perhaps most, linguistic phenomena, is nevertheless not trivial. Rambow (2014) argued that unbounded scrambling as found in German and other free word order languages falls outside the generative capacity of MCFGs (and mildly context sensitive grammar formalisms in general) and hence k-bounded LFGs.

The comparison of dependency grammars and LFGs through MCFGs is also interesting from other points of view. As Wedekind & Kaplan (2020) point out, the effect of converting an LFG to an MCFG is to precompute the interaction between f- and c-structure and construct a grammar that recognizes all and only the c-structures whose f-descriptions are satisfiable. From a practical point of view, this may be an advantage in parsing. But from the perspective of theoretical LFG, it can be argued that MCFGs and the dependency grammars they give rise to conflate c- and f-structure, making it harder to state linguistic generalizations. The advantage of LFG's projection architecture is precisely "to account for significant linguistic generalizations in a factored and modular way by means of related but appropriately dissimilar representations" (Kaplan 1989: 309). Seen from the dependency grammar side, the formal results offer a choice: Kuhlmann's translation to MCFGs makes it possible to enrich dependency grammars with an account of word order in a single component; but Wedekind & Kaplan's (2020) results show that MCFGs can be "modularized" into a word order component and a functional component (which is not surprising given that MCFGs generalize CFGs precisely by dissociating dominance and linearization) to give something very close to LFG. Either way, the formal analysis exposes similarities and differences between the frameworks. In principle, this paves the way for cross-fertilization on the theoretical side, but in practice such gains are limited by the fact that, as I pointed out in Section 1, dependency grammarians typically do not think in terms of (formal or informal) rules that generate the constructions they are interested in but content themselves with providing analyses of the whole structure.

4 DG and LFG in computational linguistics

4.1 Data-driven dependency parsing

On the computational side, there is a similar difference between DG on the one hand, and LFG and most other formal linguistic traditions on the other hand, in that there has generally been little interest in developing formal grammars that can generate or parse languages. There are some exceptions to this: in the framework of Constraint Dependency Grammar (Maruyama 1990), there is for example a broad-coverage parser of German (Foth et al. 2005); and Constraint Grammar (Karlsson et al. 1995) is a widely used system in which implemented grammars have been created for a wide variety of languages. Many of these grammars content themselves with assigning syntactic function labels to words, without building a full syntax tree, but even so, many have proven useful in practical tasks.

Nevertheless, the dominant use of DG in computational linguistics is closely associated with machine learning approaches where computers find patterns in human annotated data. For such approaches, it is sufficient that annotators provide case-by-case analyses of the corpus without actually abstracting the rules that would create these analyses. Consistency remains a goal, since it makes the patterns easier to learn, but it is not enforced in the way it would be in grammar-based annotation such as typical scenarios for creating LFG parsebanks, where annotators choose between alternative analyses provided by the underlying grammar.

As we have seen several times so far, the constraints on core dependency syntax, namely the unique mother and the one-to-one correspondence between nodes and tokens, mean that many theoretically relevant distinctions cannot be encoded. On the flip side, this makes the annotation task easier as the annotator does not have to be trained in drawing the distinctions. The result is also often more accessible to end users: while grammar-based treebanks contain much more information than dependency trees, this information is typically encoded in a specific theoretical framework and not always easily accessible to users without training in that framework. In short, core dependency trees offer a tradeoff between practical considerations and theoretical depth, which may be attractive for many applications where the deeper linguistic distinctions do not matter much.

On top of that, the simple target structure makes it possible to train very efficient statistical dependency parsers. This approach is fundamentally different from the formal grammar approach to DG developed by Kuhlmann (2013), which we saw in Section 3. Data-driven parsers learn from human annotation and try to provide the most plausible parse in context, without judging acceptability or enumerating possible parses. In this context, non-projective dependencies are not an issue and can be captured efficiently (McDonald et al. 2005). Nivre (2008) introduced algorithms that could produce projective dependency parses in time linear of the input and algorithms that allow non-projective parses and run in quadratic time. Such results led to a huge increase of interest in dependency parsing, which quickly became dominant in statistical approaches to computational linguistics.

Data-driven parsing requires annotated data and the last decade has seen a large increase in the number of dependency treebanks that are available, especially driven by the Universal Dependencies (UD) initiative.[13] UD developed out of the Stanford dependencies for English (de Marneffe & Manning 2008) (which means that there is a certain amount of LFG heritage) as an effort to create an annotation scheme that can be used across languages. Though it has been driven mainly by practical considerations in NLP research, it has in recent years also been used for linguistic research (e.g. Hahn et al. 2020, Berdicevskis & Piperski 2020).

As of release 2.9 (November 2021), UD contains 217 treebanks from 122 languages. A comparison with LFG's ParGram approach reveals the strengths and weaknesses of the approach.[14] Drawing on the long tradition of using DG to provide case-by-case analyses rather than abstracting grammars has made it possible to achieve an unprecedented breadth of coverage. On the other hand, the analyses are more shallow than those provided by LFG grammars and the lack of underlying grammars makes the UD project much more prone to inconsistencies both within and across treebanks.

4.2 Converting LFG parsebanks to dependency treebanks

The existence of annotated resources in both LFG and DG formats makes it possible to study differences between the two from a different perspective than the formal language approach we adopted in Section 3. In this section, we look at work on converting LFG-based resources to dependency structures to see how the two formats compare and to what extent information can be preserved when converting to the less expressive DG format.

For completeness, we mention that there has also been some work on enriching them to yield LFG-structures, e.g. by Forst (2003) and Haug (2012). However, both Forst and Haug started from relatively rich dependency annotations (with secondary edges), so that the conversion to f-structures was not difficult and

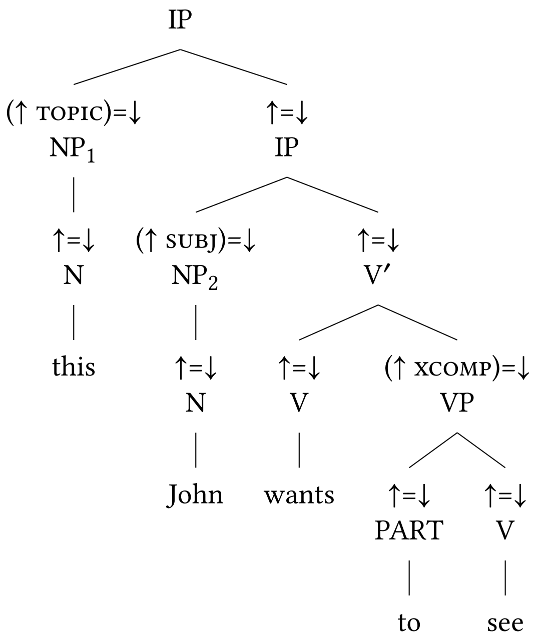
[13]See https://universaldependencies.org/ and de Marneffe et al. (2021).

[14]For more on ParGram, see Forst & King 2023 [this volume].

other issues were more important (e.g. the creation of c-structures from the dependency representations by Haug).

Several conversion algorithms have been developed to convert LFG structures to dependency structures. Here I discuss two recent approaches, by Meurer (2017) and Przepiórkowski & Patejuk (2020),[15] which contrast in interesting ways, since Meurer starts from the c-structure and Przepiórkowski and Patejuk from the f-structure. Both are natural starting points: the f-structure represents grammatical functions, just like the target dependency structure; but the c-structure has the advantage that its terminal nodes are in one-to-one correspondence with the words of the sentence, just like in the dependency structure. Both algorithms target the particular style of dependency annotation adopted in UD, but proceed in two steps, namely first the creation of a dependency structure, and second, the modification of that structure to comply with the exact representation chosen in UD. Here we focus on the first step. To illustrate how the two algorithms work, we consider the LFG structure in (16)–(17).

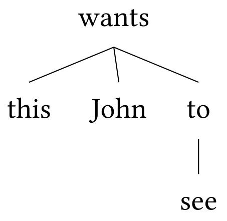
(16)



[15] Dione (2020) presents an approach that combines Meurer (2017) and Przepiórkowski & Patejuk (2020). For older work, see Øvrelid et al. (2009) and Çetinoğlu et al. (2010).

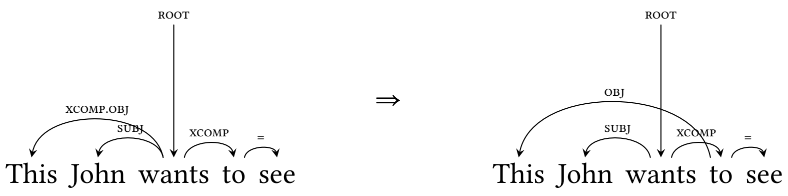
(17)
$$\left[\begin{array}{ll}
\textsc{pred} & \text{‘}\textsc{want}\langle\textsc{subj}, \textsc{xcomp}\rangle\text{’} \\
\textsc{topic} & \boxed{1}\left[\textsc{pred}\ \text{‘}\textsc{pro}\text{’}\right] \\
\textsc{subj} & \boxed{2}\left[\textsc{pred}\ \text{‘}\textsc{John}\text{’}\right] \\
\textsc{xcomp} & \left[\begin{array}{ll}
\textsc{pred} & \text{‘}\textsc{see}\langle\textsc{subj}, \textsc{obj}\rangle\text{’} \\
\textsc{subj} & \boxed{1} \\
\textsc{obj} & \boxed{2}
\end{array}\right]
\end{array}\right]$$

In Meurer’s approach, the first step is to “lexicalize” the c-structure tree by recursively replacing each non-terminal node with its functional head node, as determined by the annotation ↑=↓. This is straightforward for IP, NP_1 and NP_2 in (16): *wants*, *this* and *John* are uniquely linked to these nodes via an unbroken chain of ↑=↓. But more generally, the challenge here is the same as in lexicalizing an MCFG that results from the Wedekind-Kaplan construction: co-heads and absence of heads mean there might be no unique daughter to lift. To find a unique head in such cases the algorithm proceeds as follows: 1) if no daughter of x is a functional head, attach all daughters to the mother of x and proceed as before; 2) if more than one daughter of x is a functional head, choose the one with the shortest embedding path; 3) if there is a tie, choose the leftmost node. For the VP in our example, case 3 applies and we choose *to* as the head; it is therefore lifted to the VP node, while *see* is only lifted to the V node. These lifting operations yield the tree in (18).

(18)


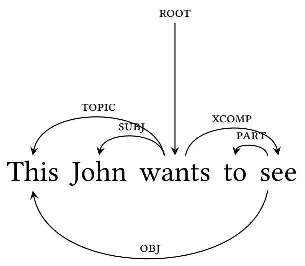
We then need to label the edges. Meurer’s algorithm does that by labelling the edge between nodes x and their daughter y in the resulting tree with the f-structure path from $\phi(x)$ to $\phi(y)$. So, the edge from *John* to *wants* is labelled SUBJ since that is the path from the f-structure of *wants* to the f-structure of *John*. But because of reentrancies in the f-strucure, the path between two f-structures is not always unique: for example, there is a path from the f-structure of *wants* to the f-structure of *this* that is labelled TOPIC, but there is another path that is labelled XCOMP OBJ. In such cases, Meurer chooses the shortest path that contains only grammatical functions (i.e. no discourse functions); in our case that yields the complex label XCOMP.OBJ where the two elements of the f-structure path have been concatenated with a dot. Co-heads present another problem for the labelling approach: *to* and *see* share the same f-structure, so there is no path. In such cases a dummy relation = is used.

This algorithm produces a projective dependency graph with complex labels, as shown for our example in the lefthand side of (19). In the next step, the complex labels are resolved and nodes attached accordingly, potentially introducing non-projectivity. For our example, when the complex relation xcomp.obj is resolved, we obtain the non-projective tree on the right-hand side of (19).

(19)


This then is the input to the final step, where the dependency tree is normalized according to the UD annotation standard.

Przepiórkowski & Patejuk (2020), by contrast, start from the f-structure, which already represents the syntactic dependencies. This means the challenge is different, namely to match the nodes of the f-structure to the words of the sentence, which are the nodes of the target dependency graph. F-structure nodes may correspond to zero, one or several words; they are given by the ϕ^{-1}, which is part of the source annotation. F-structures that correspond to no words (e.g. in pro-drop) may simply be ignored in the dependency structure; but for f-structures that correspond to more than one word, the "true" head that will take the f-structure's place in the corresponding dependency structure must be identified, and the other words in ϕ^{-1} must be attached with appropriate relations. The basic algorithm is simple: if there is a verbal token in ϕ^{-1}, choose that as the true head; otherwise, choose a nominal or adjectival token; otherwise an explicit lexical conjunction. The other nodes are then attached to the true head with a relation labelled by their own preterminal category. This produces the structure in (20) from the f-structure in (17).

(20)


As we can see, the output from the algorithm of Przepiórkowski & Patejuk (2020) is not a tree, but a graph, where all f-structure relations are preserved, including two incoming edges to *this*. This is exploited to produce enhanced UD, which allows for this kind of graph structure; but the output is also trimmed to produce a basic UD structure.

(19)–(20) illustrate the output of the first steps in the conversions, where the target is to produce the desired data structure, namely a dependency tree or graph over words. As mentioned, the next step is to normalize this structure to the concrete requirements of the UD annotation standard. This is less interesting from our point of view, but it is worth looking at a few topics that display divergences between standard LFG solutions and choices that are made in the dependency grammar community as exemplified by UD.

First, UD subscribes to the primacy of content words. This means that content words are typically heads of function words, for example in structures consisting of auxiliary and verb, adposition and noun, and determiner and noun, as illustrated in the lower graph of (2) in Section 1. Also, there are no nested structures of function words, so e.g. in structures with multiple auxiliaries (*may have been understood*), all the auxiliaries attach directly to the lexical verb. While UD may be extreme among dependency grammar approaches in adopting this principle across the board, similar analyses are found for some of these structures in other frameworks. By contrast, such analyses are non-existing in the LFG literature, except for noun-determiner structures (where the determiner is often analyzed as a SPEC dependent of the noun): function words are typically either co-heads, or lone heads, taking a lexical word as their dependent. For example, there are analyses of auxiliaries as co-heads specifying features of the f-structure where the lexical verb contributes the PRED, and alternative analyses where auxiliary verbs take XCOMP dependents, potentially in a cascading sequence ending in the lexical verb.

In fact, the difference between the co-head analysis and the UD dependent analysis of function words is rather slight, as revealed by the conversion procedure of Przepiórkowski & Patejuk (2020). In f-structures that have functional co-heads, the lexical head will be chosen as the head during conversion, and hence the function words will end up as dependents. And in fact, given that UD uses a flat structure for multiple function words means that the two representations are more or less equivalent, a point made in the UD documentation too,[16] where it is said that function word relations are different from dependency relations between content words and in fact form Tesnière-style nuclei.

[16]https://universaldependencies.org/u/overview/syntax.html

This in turn opens the door to theoretical cross-fertilization. What are good criteria for choosing between the two analyses? The UD argument is that primacy of lexical words maximizes parallelism across languages, and the exact same argument has been raised in the LFG literature (Butt et al. 1996). On the other hand, Dyvik (1999) has countered that this leads to a stipulative, rather than empirical, notion of language universality and also that it can lead to analyses that are language-internally unmotivated. Recently, Osborne & Gerdes (2019) criticized the UD approach and argued that functional words should always be heads. They were apparently unaware of the LFG literature on the topic, perhaps because it is cast in terms of co-heads vs. xcomps. But as we have seen, the difference between a co-head analysis and a UD-style annotation is very slight, and so the arguments made in the LFG context are certainly relevant also for the DG community.

The other main divergence between initial dependencies, as resulting from the conversion algorithms, and the target UD structures concern coordination. Here LFG makes use of an additional data structure, sets, which have no equivalents in standard dependency grammar or in UD. (Although as we saw above, Tesnière's junction comes close.) There are many competing analyses of coordination in the dependency literature,[17] maybe suggesting that the basic data structure of dependency trees is ill-suited to model coordination, as Tesnière argued. The UD choice is to take the first conjunct as the head and attach the other conjuncts to it with a special dependency relation conj, whereas conjunctions and punctuation marks are attached to their following conjunct with cc and punct. It is known that this annotation style cannot capture all important structural differences, such as the difference between a dependent of the first conjunct and a shared dependent of multiple conjuncts, or different style of nested coordinations. The conversion procedure exposes this lack of expressivity, but also makes it possible to quantify its effect. As observed by Przepiórkowski & Patejuk (2020), only twelve out of 21,732 utterances in the Polish LFG structure bank are effected.[18]

More generally, Przepiórkowski & Patejuk (2020) conclude that the information loss in converting from LFG to (enhanced) UD is in fact negligible, except in the case of pro-drop structures. As the UD effort continues to expand, there is therefore considerable potential for theoretical cross-fertilization.

[17]And also in (pre-UD) dependency treebanks, see Popel et al. (2013).

[18]See Przepiórkowski & Patejuk (2019) for a proposal as to how nested coordination could be captured in UD.

5 Semantics

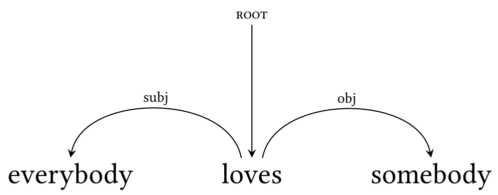
Tesnière in general pays much less attention to meaning than to structure, but at various points he does talk about semantic dependencies. Several versions of dependency grammars (Functional Generative Description, Meaning–Text Theory) have taken this up and operate with a separate level of semantic structure. There are also various graph-based semantic representation languages such as Abstract Meaning Representation (AMR, Banarescu et al. 2013), which arguably are semantic dependency representations without an accompanying syntactic representation. All such semantic dependency graphs, whether they are coupled to syntax or not, differ considerably from standard logic-based formalizations of meaning as used in LFG and most other formal frameworks. They will not be further discussed here.

Robaldo's Dependency Tree Semantics (Robaldo 2006) is much closer to standard conceptions of formal semantics, as it aims to transform dependency trees into structures that can be interpreted model-theoretically. But for the purposes of comparison with LFG, it is more interesting to observe that Bröker (2003: 308), in his discussion of the formal foundations of dependency grammar, briefly suggested that the similarity between dependency trees and LFG's functional structure could make the application of Glue semantics (Dalrymple et al. 1993, Dalrymple 1999; see also Asudeh 2023 [this volume]) to dependency grammar a promising research area. Gotham & Haug (2018) flesh out this idea and show how to combine Universal Dependencies with Partial Compositional Discourse Representation Theory (Haug 2014).

On the formal side, there are few if any obstacles to such an application. The fundamental idea behind glue semantics is to have linear logic terms guide the composition of corresponding lambda terms. In the first order glue setting, the terms of the linear logic are the f-structures, atomic formulae are formed by applying predicates to the f-structures (in type-theoretic terms, these predicates act like unary type constructors) and complex formulae are formed with $\multimap$, which acts as a binary function type constructor. Consider (21), which gives the meanings, dependency structure and f-structure for *Everybody loves somebody.* We write e_1 for $e(1)$, i.e. the application of the type constructor/predicate e to the syntactic object/term with index 1.

(21) a.

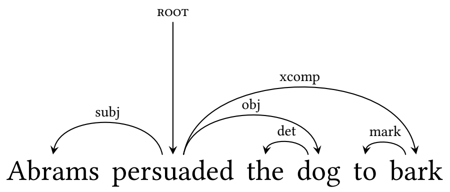
everybody	$\lambda P.\forall x.person(x) \rightarrow P(x)$	$(e_1 \multimap t_2) \multimap t_2$
somebody	$\lambda P.\forall x.person(x) \wedge P(x)$	$(e_3 \multimap t_2) \multimap t_2$
loves	$\lambda x.\lambda y.love(x, y)$	$e_1 \multimap e_3 \multimap t_2$

b. 

c. $$\begin{bmatrix} \textsc{pred} & \text{`}\textsc{love}\langle\ \textsc{subj}\ \rangle\text{'} \\ \textsc{subj} & \begin{bmatrix}\textsc{pred} & \text{`}\textsc{everybody}\end{bmatrix} \\ \textsc{obj} & \begin{bmatrix}\textsc{pred} & \text{`}\textsc{somebody}\text{'}\end{bmatrix} \end{bmatrix}$$

Clearly, it makes no difference whether we interpret the glue types in (21a) over the dependency tree in (21b) or the f-structure in (21c): in both cases we just need the same mapping between the indices 1, 2, 3 and the corresponding f-structures or dependency nodes.

However, while the formal properties of the two theories are similar enough that Glue semantics can be used for both LFG and DG, a practical consideration is that dependency trees typically do not contain all the semantically relevant information that we find in the corresponding f-structure. Control structures are a case in point (22).

(22) 

The dependency tree in (22) lacks the information that *the dog* is the subject of *to bark*. However, the label XCOMP does tell us that the missing subject of *to bark* is one of the dependents of *persuaded*. As a result, the best we can do is to introduce a discourse referent x_2 that is the subject of the infinitive clause and must be linked to one of the participants in the matrix event, though we do not know which one, unless we have access to the lexical information that *persuade* is an object control verb. We see that it is possible to compensate for some of the information loss in dependency trees, although the result only becomes useful if we have other, lexical information sources available: dependency trees on their own do not typically come with the rich semantic lexical entries that glue (and other formal semantic theories) require. We refer to Gotham & Haug (2018) for more details.

6 Summary

We have seen that the basic relations for analyzing functional syntax, dependencies in DG and grammatical functions in LFG, are very similar, both formally and conceptually. Nevertheless, the focus on core dependencies that is often seen in DG work leaves other levels of analysis less well-developed than in LFG. Word order, in particular, has not received much attention, but we have seen that it can be interestingly restricted through the use of lexicalized MCFGs, offering a point of comparison to LFGs, which can also be translated to MCFGs. Another point of comparison is offered by work on converting LFG parsebanks to dependency treebanks. Finally, we saw that the similarity between DG and LFG also means that they can use the same syntax-semantics interface in the form of Glue semantics.

All in all, the considerable similarities between the two theories suggest there is ample room for mutually benefiting discussion, especially if the increasing use of DG in computational linguistics triggers a corresponding interest in theoretical DG.

Acknowledgements

I thank the two anonymous reviewers, one non-anonymous reviewer (Adam Przepiórkowski) and the editor Mary Dalrymple for very helpful comments on a previous version of this chapter.

References

Asudeh, Ash. 2023. Glue semantics. In Mary Dalrymple (ed.), *Handbook of Lexical Functional Grammar*, 651–697. Berlin: Language Science Press. DOI: 10.5281/zenodo.10185964.

Banarescu, Laura, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer & Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, 178–186. Sofia: Association for Computational Linguistics. https://www.aclweb.org/anthology/W13-2322.

Berdicevskis, Aleksandrs & Alexander Piperski. 2020. Corpus evidence for word order freezing in Russian and German. In *Proceedings of the fourth workshop on Universal Dependencies (UDW 2020)*, 26–33. Barcelona: Association for Computational Linguistics. https://aclanthology.org/2020.udw-1.4.

Bodirsky, Manuel, Marco Kuhlmann & Mathias Möhl. 2005. Well-nested drawings as models of syntactic structure. In James Rogers (ed.), *Proceedings of FG-MoL 2005: The 10th conference on Formal Grammar and the 9th meeting on Mathematics of Language*, 195–203. Stanford: CSLI Publications. http://cslipublications.stanford.edu/FG/2005/bodirsky.pdf.

Bröker, Norbert. 1998. How to define a context-free backbone for DGs: Implementing a DG in the LFG formalism. In *Processing of dependency-based grammars*. https://aclanthology.org/W98-0504.

Bröker, Norbert. 2003. Formal foundations of dependency grammar. In Vilmos Ágel (ed.), *Dependency and valency* (Handbook of Linguistics and Communication Sciences 25), 294–310. Berlin: De Gruyter.

Butt, Miriam, María-Eugenia Niño & Frederique Segond. 1996. Multilingual processing of auxiliaries in LFG. In D. Gibbon (ed.), *Natural language processing and speech technology: Results of the 3rd KONVENS conference*, 111–122. Berlin: Mouton de Gruyter.

Çetinoğlu, Özlem, Jennifer Foster, Joakim Nivre, Deirdre Hogan, Aoife Cahill & Josef van Genabith. 2010. LFG without c-structures. In *NEALT proceedings*, vol. 9, 43–54.

Chomsky, Noam. 1966. *Cartesian linguistics: A chapter in the history of rationalist thought*. New York: Harper & Row. DOI: 10.1017/cbo9780511803116.

Clark, Alexander. 2014. An introduction to multiple context free grammars for linguists. https://alexc17.github.io/papers/mcfgsforlinguists.pdf.

Covington, Michael A. 1984. *Syntactic theory in the High Middle Ages*. Cambridge, UK: Cambridge University Press. DOI: 10.1017/cbo9780511735592.

Creider, Chet & Richard Hudson. 2006. Case agreement in Ancient Greek: Implications for a theory of covert elements. In Kensei Sugayama & Richard Hudson (eds.), *Word Grammar: New perspectives on a theory of language structure*, 35–53. London: Continuum.

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, Ronald M. Kaplan, John T. III Maxwell & 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.

de Marneffe, Marie-Catherine & Christopher D. Manning. 2008. The Stanford typed dependencies representation. In *COLING 2008: Proceedings of the Workshop on Cross-framework and Cross-domain Parser Evaluation*, 1–8. Manchester. DOI: 10.3115/1608858.1608859.

de Marneffe, Marie-Catherine, Christopher D. Manning, Joakim Nivre & Daniel Zeman. 2021. Universal Dependencies. *Computational Linguistics* 47(2). 255–308. DOI: 10.1162/coli_a_00402.

de Marneffe, Marie-Catherine & Joakim Nivre. 2019. Dependency grammar. *Annual Review of Linguistics* 5(1). 197–218. DOI: 10.1146/annurev-linguistics-011718-011842.

Dione, Cheikh M. Bamba. 2020. From LFG to UD: A combined approach. In *Proceedings of the fourth workshop on Universal Dependencies (UDW 2020)*, 57–66. Barcelona: Association for Computational Linguistics. https://aclanthology.org/2020.udw-1.7.

Dyvik, Helge. 1999. The universality of f-structure: Discovery or stipulation? The case of modals. In Miriam Butt & Tracy Holloway King (eds.), *Proceedings of the LFG '99 conference*, 1–11. Stanford: CSLI Publications.

Forst, Martin. 2003. Treebank conversion – Establishing a testsuite for a broad-coverage LFG from the TIGER treebank. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03) at EACL 2003*, 205–216. Association for Computational Linguistics. https://www.aclweb.org/anthology/W03-2404.

Forst, Martin & Tracy Holloway King. 2023. Computational implementations and applications. In Mary Dalrymple (ed.), *Handbook of Lexical Functional Grammar*, 1083–1123. Berlin: Language Science Press. DOI: 10.5281/zenodo.10185986.

Foth, Kilian, Wolfgang Menzel & Ingo Schröder. 2005. Robust parsing with weighted constraints. *Natural Language Engineering* 11(1). 1–25. DOI: 10.1017/S1351324903003267.

Gaifman, Haim. 1965. Dependency systems and phrase-structure systems. *Information and Control* 8(3). 304–337. DOI: 10.1016/s0019-9958(65)90232-9.

Gotham, Matthew & Dag Haug. 2018. Glue semantics for Universal Dependencies. In Miriam Butt & Tracy Holloway King (eds.), *Proceedings of the LFG '18 conference*, 208–226. Stanford: CSLI Publications.

Hahn, Michael, Dan Jurafsky & Richard Futrell. 2020. Universals of word order reflect optimization of grammars for efficient communication. *Proceedings of the National Academy of Sciences* 117(5). 2347–2353. DOI: 10.1073/pnas.1910923117.

Hajič, Jan, Eduard Bejček, Jaroslava Hlaváčová, Marie Mikulová, Milan Straka, Jan Štěpánek & Barbora Štěpánková. 2020. Prague Dependency Treebank - Consolidated 1.0. In *Proceedings of the 12th International Conference on Lan-*

guage Resources and Evaluation (LREC'20), 5208–5218. Marseille: European Language Resources Association (ELRA). https://www.aclweb.org/anthology/2020.lrec-1.641.

Haug, Dag. 2012. From dependency structures to LFG representations. In Miriam Butt & Tracy Holloway King (eds.), *Proceedings of the LFG '12 conference*, 271–291. Stanford: CSLI Publications.

Haug, Dag. 2014. Partial dynamic semantics for anaphora: Compositionality without syntactic coindexation. *Journal of Semantics* 31(4). 457–511. DOI: 10.1093/jos/fft008.

Hays, David G. 1964. Dependency theory: A formalism and some observations. *Language* 40(4). 511–525. DOI: 10.2307/411934.

Holan, Tomáš, Vladislav Kuboň, Karel Oliva & Martin Plátek. 1998. Two useful measures of word order complexity. In *Processing of dependency-based grammars*. https://aclanthology.org/W98-0503.

Hudson, Richard. 1984. *Word Grammar*. Oxford: Blackwell.

Hudson, Richard. 2003. Word Grammar. In Vilmos Ágel (ed.), *Dependency and valency*, vol. 25 (Handbook of Linguistics and Communication Sciences), 508–526. Berlin: De Gruyter. DOI: 10.1093/oxfordhb/9780199738632.013.0019.

Hudson, Richard. 2010. *An introduction to Word Grammar*. Cambridge, UK: Cambridge University Press. DOI: 10.1017/cbo9780511781964.

Kahane, Sylvain. 2003. The Meaning-Text Theory. In Vilmos Ágel (ed.), *Dependency and valency* (Handbook of Linguistics and Communication Sciences 25). Berlin: De Gruyter.

Kallmeyer, Laura. 2010. *Parsing beyond context-free grammars*. Berlin: Springer. DOI: 10.1007/978-3-642-14846-0.

Kaplan, Ronald M. 1989. The formal architecture of Lexical-Functional Grammar. *Journal of Information Science and Engineering* 5. 305–322. Revised version published as Kaplan (1995).

Kaplan, Ronald M. 1995. The formal architecture of Lexical-Functional Grammar. In Mary Dalrymple, Ronald M. Kaplan, John T. III Maxwell & Annie Zaenen (eds.), *Formal issues in Lexical-Functional Grammar*, 7–27. Stanford: CSLI Publications. Earlier version published as Kaplan (1989).

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).

Karlsson, Fred, Atro Voutilainen, Juha Heikkilae & Arto Anttila. 1995. *Constraint grammar: A language-independent system for parsing unrestricted text*. Berlin: Walter de Gruyter.

Kay, Martin. 1979. Functional grammar. In *Proceedings of the 5th annual meeting of the Berkeley Linguistics Society*, 142–158. Berkeley: Berkeley Linguistics Society. DOI: 10.3765/bls.v5i0.3262.

Kiparsky, Paul. 1993. Paninian linguistics. In Ronald Asher & Seumas Simpson (eds.), *Encyclopedia of languages and linguistics*. Oxford: Pergamon.

Kuhlmann, Marco. 2007. *Dependency structures and lexicalized grammars*. Saarbrücken: Universität des Saarlandes. (Doctoral dissertation). http://www.ida.liu.se/~marku61/pdf/kuhlmann2007dependency.pdf.

Kuhlmann, Marco. 2010. *Dependency structures and lexicalized grammars: An algebraic approach*. Berlin: Springer. DOI: 10.1007/978-3-642-14568-1.

Kuhlmann, Marco. 2013. Mildly non-projective dependency grammar. *Computational Linguistics* 39(2). 355–387. DOI: 10.1162/COLI_a_00125.

Maruyama, Hiroshi. 1990. Structural disambiguation with constraint propagation. In *Proceedings of the 28th annual meeting of the Association for Computational Linguistics*, 31–38. Pittsburgh: Association for Computational Linguistics. DOI: 10.3115/981823.981828.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov & Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology conference and Conference on Empirical Methods in Natural Language Processing*, 523–530. Vancouver: Association for Computational Linguistics. DOI: 10.3115/1220575.1220641.

Mel'čuk, Igor A. 1988. *Dependency syntax: Theory and practice*. Albany, NY: State University of New York Press.

Meurer, Paul. 2017. From LFG structures to dependency relations. *Bergen Language and Linguistics Studies* 8. 183–201. DOI: 10.15845/bells.v8i1.1341.

Moshier, M. D. & W. C. Rounds. 1987. A logic for partially specified data structures. In *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (POPL '87), 156–167. Munich: Association for Computing Machinery. DOI: 10.1145/41625.41639.

Neuhaus, Peter & Norbert Bröker. 1997. The complexity of recognition of linguistically adequate dependency grammars. In *Proceedings of the 35th annual meeting of the Association for Computational Linguistics and 8th conference of the European chapter of the ACL* (ACL '98/EACL '98), 337–343. Association for Computational Linguistics. DOI: 10.3115/976909.979660.

Nivre, Joakim. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics* 34(4). 513–553. DOI: 10.1162/coli.07-056-r1-07-027.

Osborne, Timothy & Kim Gerdes. 2019. The status of function words in dependency grammar: A critique of Universal Dependencies (UD). *Glossa: A Journal of General Linguistics* 4(1). 17. DOI: 10.5334/gjgl.537.

Øvrelid, Lilja, Jonas Kuhn & Kathrin Spreyer. 2009. Cross-framework parser stacking for data-driven dependency parsing. *Traitement automatique des langues* 50(3). 109–138.

Popel, Martin, David Mareček, Jan Štěpánek, Daniel Zeman & Zdeněk Žabokrtský. 2013. Coordination structures in dependency treebanks. In *Proceedings of the 51st annual meeting of the Association for Computational Linguistics (volume 1: long papers)*, 517–527. Sofia. http://www.aclweb.org/anthology/P13-1051.

Przepiórkowski, Adam. 2023. LFG and HPSG. In Mary Dalrymple (ed.), *Handbook of Lexical Functional Grammar*, 1861–1918. Berlin: Language Science Press. DOI: 10.5281/zenodo.10186042.

Przepiórkowski, Adam & Agnieszka Patejuk. 2019. Nested coordination in Universal Dependencies. In *Proceedings of the third workshop on Universal Dependencies (UDW SyntaxFest 2019)*, 58–69. Paris: Association for Computational Linguistics. DOI: 10.18653/v1/W19-8007.

Przepiórkowski, Adam & Agnieszka Patejuk. 2020. From Lexical Functional Grammar to Enhanced Universal Dependencies: The UD-LFG treebank of Polish. *Language Resources and Evaluation* 54. 185–221. DOI: 10.1007/s10579-018-9433-z.

Rambow, Owen. 2014. *Formal and computational aspects of natural language syntax*. Philadelphia: University of Pennsylvania. (Doctoral dissertation).

Richter, Frank. 2021. Formal background. In Stefan Müller, Anne Abeillé, Robert D. Borsley & Jean-Pierre Koenig (eds.), *Head-Driven Phrase Structure Grammar: The handbook* (Empirically Oriented Theoretical Morphology and Syntax), 89–124. Berlin: Language Science Press. DOI: 10.5281/zenodo.5599822.

Robaldo, Livio. 2006. *Dependency tree semantics*. Turin: University of Turin. (Doctoral dissertation).

Seki, Hiroyuki, Takahashi Matsumura, Mamoru Fujii & Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science* 88(2). 191–229. DOI: 10.1016/0304-3975(91)90374-b.

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.

Sgall, Petr, Eva Hajičová & Jarmila Panevová. 1986. *The meaning of the sentence in its semantic and pragmatic aspects*. Prague: Academia.

Tesnière, Lucien. 1959. *Éléments de syntaxe structurale*. Paris: Klincksieck.

Vijay-Shanker, K., David J. Weir & Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th annual meeting of the Association for Computational Linguistics*, 104–111. Stanford: Association for Computational Linguistics. DOI: 10.3115/981175.981190.

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