Proceedings of the Third International Workshop on Computational Semantics (IWCS-3), KUB, Tilburg, January 1999

A Functional Interpretation Scheme for D-Tree Grammars

Mark Hepple

Department of Computer Science
University of Sheffield, UK
hepple@dcs.shef.ac.uk

Abstract

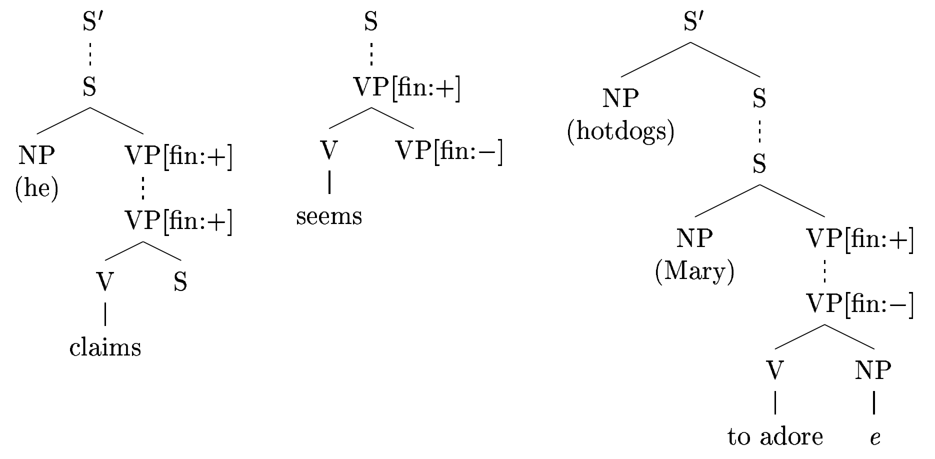
This paper suggests a new method for interpreting D-Tree Grammar (DTG) derivations that is inspired by ideas from the categorial area. The standard treatment of DTG interpretation is based on the *derivation tree* (a record of the steps made during the derivation) and requires that the derivation process be constrained in ways that would not otherwise be required. The new method suggested is based on the *derived tree*, rather than the derivation tree. As such it allows the constraints on the derivation process mentioned to be eliminated, both as an unnecessary complication of the approach and as as obstacle to possible analyses that might be formulated within the framework. A 'glue language' style variant of the approach is also described, which makes possible a treatment of quantification.

Keywords: D-Tree Grammar, categorial grammar, glue language

1 Introduction

This paper considers some similarities between D-Tree Grammars and type-logical grammars, suggested by a categorial parsing approach in which complex formulae are compiled to simpler formulae (analogous to tree fragments), between which 'inclusion' requirements hold (analogous to domination relations). This comparison suggests an approach to providing a functional semantics for D-Tree derivations, under which the interpretation for a derivation is constructed on the basis of its *derived* tree, which takes the form of a standard phrase structure tree. The standard approach to interpreting D-Tree Grammar derivations is, in contrast, based on the *derivation tree*, which is a record of the composition steps made during the process of derivation. This latter method requires the derivation process to be subject to certain constraints, that both complicate the framework and rule out certain accounts of phenomena that could otherwise be formulated. By virtue of being based on the derived tree rather than the derivation tree, the new method for interpreting derivations allows these constraints on the derivation process to be eliminated. We consider a treatment of pied-piping which this change makes possible. The approach, however, has some limitations, as is discussed in relation to quantification, and we consider how these limitations can be overcome by a reformulation that is inspired by the glue language approach to interpreting LFG derivations (Dalrymple *et al.*, 1993).

(1)



2 D-Tree Grammars

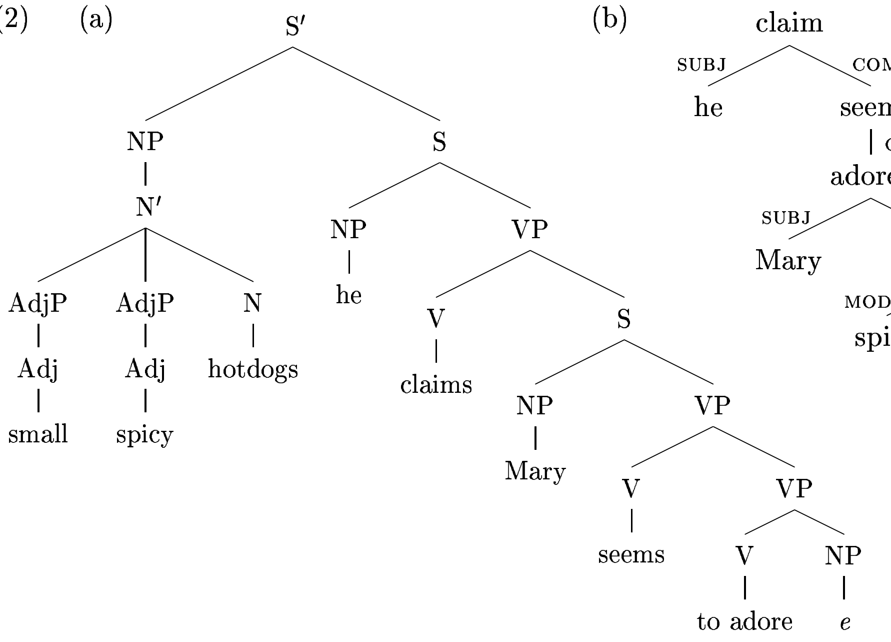
The D-Tree Grammar (DTG) formalism is introduced in (Rambow *et al.*, 1995). The basic derivational unit of this formalism is the d-tree, which (loosely) consists of a collection of tree fragments with domination links (d-links) between nodes in different fragments, that link them into a single graph. D-trees are seen as *partial descriptions* of trees. Some example d-trees, drawn from (Rambow *et al.*, 1995), are shown in (1), that could be used to derive the tree (2a) for *Hotdogs$_i$, he claims Mary seems to adore t_i* (for which a few simple other d-trees are required). (The words shown in brackets in (1) are there only make explicit the correspondences to (2a).

The main operation for composing d-trees is *subsertion*. When a d-tree α is subserted into a d-tree γ, a fragment of α is substituted at a suitable node within γ, with other (dominating) fragments of α being inserted into d-links of γ or above its root node. For example, the third d-tree in (1) might be subserted into the second, with the bottom fragment of the former being substituted at the VP[fin:−] node of the latter to create a new fragment (for *seems to adore e*). The other dominating fragments of the two initial d-trees would now appear in a single merged d-link chain in some order (different orders being possible). A second composition operation *sister-adjunction* is used in handling modification, and adds in a modifier subtree as an additional daughter to an already existing local tree. At the end of a derivation, the final d-tree can be reduced to a conventional phrase structure tree by removing any d-links (the top and bottom nodes of each d-link being merged, which is possible only if the two bear the same category).

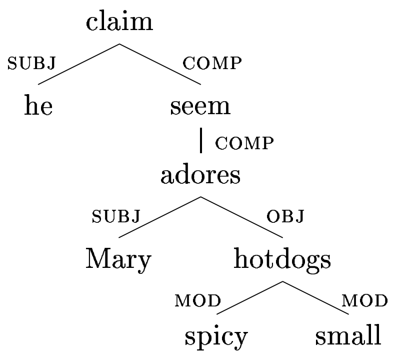
Rambow *et al.* (1995) motivate the D-Tree Grammar approach in terms of some problems that arise for some related formalisms, such as TAG (Tree-Adjoining Grammar (Joshi *et al.*, 1975)) and MCTAG-DL (Multi-Component TAG with Domination Links (Becker *et al.*, 1991)), involving linguistic coverage and the semantic interpretation of derivations. It is this latter issue that is the main concern of this paper.

For TAG, the interpretation of derivations has been handled not in terms of the derived phrase-structure tree, but rather the *derivation* tree, which is a record of the steps of the derivation, i.e. which tree has been substituted or adjoined at which site in which other tree

(2) (a)



(b)



at each stage. However, as Rambow *et al.* (1995) discuss,[1] this approach is problematic, as there is not a simple consistent relation between the TAG composition steps and the semantic relations established, with the consequence that TAG derivation trees do not provide a good representation of the semantic dependencies of the sentence.

DTG aims to overcome this problem by having a straightforward relation between the tree composition operations and their semantic import. Thus, subsertion consistently establishes head-complement relations (the subserted tree corresponding to the complement), and sister-adjunction consistently establishes head-modifier relations (the sister-adjoined tree being the modifier). The approach is such that the derivation tree for a phrase structure such as (2a) is isomorphic to its dependency structure, shown in (2b), and so provides a suitable basis for interpretation. For this idea to work out, the derivation process must be constrained so that derivation trees are in fact trees.[2]

Although the DTG use of derivation trees for handling interpretation improves on the situation for TAG, there are some ways in which the approach might still be seen as unsatisfactory. Firstly, we might question whether the need to formulate derivation as a *process* is really in keeping with the spirit of the DTG framework. Given that the basic units of the approach are lexicalised partial tree descriptions, we might hope to see a purely declarative, constraint-based, treatment of 'derivation', in which we simply ask of a given tree whether it satisfies the conjoined requirements of the partial tree descriptions

[1]See also Candito & Kahane (1998a) for further discussion of the issues arising.

[2]This requirement would be violated if two tree fragments that originate with the same initial d-tree δ were substituted into different structures during the derivation, as the lexical anchor of δ would then be interpreted as the dependent of two different heads, and a derivation 'tree' that was not a tree returned. This possibility is blocked by constraining the propagation of a property *substitutability*, which a tree fragment must have to be the substituted element in a subsertion step. Candito & Kahane (1998b) suggest a variant of DTG with a view to allowing graph structured meaning representations.

associated with the words that it dominates. Secondly, the restriction of the derivation process that is needed to ensure that interpretable derivation trees are returned may rule out some possible syntactic analyses that might otherwise be offered within the framework. An example is outlined later in the paper, involving a possible treatment of pied-piping.

This paper presents an approach for interpreting DTG analyses that is based purely on the derived tree, rather than the derivation tree. As such, it eliminates the need to constrain the derivation process, and so resolves both of the problems discussed above. The approach is 'functional' in that it involves associating lambda expressions with the tree fragments in initial d-trees, which allow us to compute a meaning term for a complete analysis. The approach is suggested by some similarities between DTG and type-logical grammars that are observed in the context of a parsing approach for the latter, outlined in the next section, in which higher-order formulae are compiled to first-order formulae.

3 Type-logical Grammar & First-order Compilation

The associative Lambek calculus (Lambek, 1958) is the most familiar representative of the 'type-logical' tradition within categorial grammar, but a range of such systems have been proposed, which differ in their resource sensitivity (and hence, implicitly, their underlying notion of 'linguistic structure'). Some of these proposals are formulated using a 'labelled deduction' methodology (Gabbay, 1996), whereby the types in a proof are associated with labels, under a specified discipline, which record proof information used in ensuring correct inferencing. Such a labelling system must be overlaid upon a 'backbone logic', commonly a fragment of linear logic such as its implicational fragment. This fragment can be seen as constituting a categorial grammar in its own right, one that is very simple but which is mostly adequate for the purposes of this paper, and to which we next turn. (The fragment's most obvious limitation for grammar purposes is that its types cannot encode word order information).

3.1 Implicational Linear Logic

In linear logic proofs, each assumption is used precisely once. Natural deduction rules of *elimination* and *introduction* for linear implication ($\multimap$) are as in (3). (The premises of the $\multimap$E rule can appear in either order. In the $\multimap$I rule, [B] indicates a discharged or withdrawn assumption.) Eliminations and introductions correspond to steps of functional application and abstraction, respectively, as the lambda-term labelling reveals.

(3)
$$\frac{\mathrm{B} \multimap \mathrm{A} : a \qquad \mathrm{B} : b}{\mathrm{A} : (ab)}\,{\multimap}\mathrm{E} \qquad\qquad \frac{\begin{array}{c}[\mathrm{B} : v] \\ \mathrm{A} : a\end{array}}{\mathrm{B} \multimap \mathrm{A} : \lambda v.a}\,{\multimap}\mathrm{I}$$

The proof in (4) illustrates 'hypothetical reasoning', where an additional assumption, or 'hypothetical', is used that is later discharged. The involvement of hypotheticals is driven by the presence of higher-order formulae (i.e. functors seeking an argument that bears a functional type): each corresponds to a subformula of a higher-order formula, e.g.

Z in (4) is a subformula of (Z ⊸ Y) ⊸ X.[3]

(4)
$$\dfrac{\dfrac{\dfrac{\dfrac{[\mathrm{Z} : z] \quad \mathrm{Z} \multimap \mathrm{W} : w}{\mathrm{W} : (wz)} \quad \mathrm{W} \multimap \mathrm{Y} : y}{\mathrm{Y} : (y(wz))}}{\mathrm{Z} \multimap \mathrm{Y} : \lambda z.y(wz)} \quad (\mathrm{Z} \multimap \mathrm{Y}) \multimap \mathrm{X} : x}{\mathrm{X} : x(\lambda z.y(wz))}$$

3.2 First-order Compilation

Hepple (1996) shows how deductions in implicational linear logic can be recast as deductions involving only *first-order* formulae (i.e. where any arguments sought by functors bear *atomic* types) and using only a single inference rule (a variant of ⊸E). The compilation reduces higher-order formulae to first-order formulae by *excising* subformulae corresponding to hypotheticals, e.g. so (Z ⊸ Y) ⊸ X gives Y ⊸ X *plus* Z. A system of indexing is used to ensure correct use of excised subformulae, to prevent invalid reasoning, e.g. the excised Z *must* be used to derive the argument of Y ⊸ X. Each compiled formula has an index set with one member, e.g. $\langle\{j\}, \mathrm{Z}, z\rangle$ (sometimes written $\langle j, \mathrm{Z}, z\rangle$), which serves as its unique identifier. The index set of a derived formula identifies the assumptions used to derive it. The single inference rule (5) ensures correct propagation of indices (where $\uplus$ is *disjoint* union). Each argument slot of a compiled functor also has an index set, which identifies any assumptions that *must* be used in deriving its argument, as enforced by the rule condition $\alpha \subseteq \psi$.

(5)
$$\dfrac{\langle\psi, \mathrm{B}, b\rangle \quad \langle\phi, (\mathrm{B} : \alpha) \multimap \mathrm{A}, \lambda v.a\rangle}{\langle\pi, \mathrm{A}, a[b/\!\!/v]\rangle} \qquad \begin{array}{l} \pi = \phi \uplus \psi \\ \alpha \subseteq \psi \end{array}$$

In proving Z ⊸ W, W ⊸ Y, (Z ⊸ Y) ⊸ X ⇒ X, for example, compilation yields the assumption formulae of the following proof:

$$\dfrac{\dfrac{\dfrac{\langle j, \mathrm{Z}, z\rangle \quad \langle l, (\mathrm{Z} : \emptyset) \multimap \mathrm{W}, \lambda v.wv\rangle}{\langle\{j, l\}, \mathrm{W}, wz\rangle} \quad \langle k, (\mathrm{W} : \emptyset) \multimap \mathrm{Y}, \lambda u.yu\rangle}{\langle\{j, k, l\}, \mathrm{Y}, y(wz)\rangle} \quad \langle i, (\mathrm{Y} : \{j\}) \multimap \mathrm{X}, \lambda t.x(\lambda z.t)\rangle}{\langle\{i, j, k, l\}, \mathrm{X}, x(\lambda z.y(wz))\rangle}$$

The leftmost (F1) and rightmost (F2) assumptions of the proof both come from the formula (Z ⊸ Y) ⊸ X, and F1 requires its argument to include F2. Compilation has removed the need for an explicit introduction step in the proof, c.f. proof (4), but the effects of this step have been compiled into the semantics of the formulae. Thus, the term of F1 includes an apparently vacuous abstraction over variable z, which is the term assigned to F2. The semantics of rule (5) is handled not by simple application, but rather direct substitution for

[3]The relevant subformulae can be can be identified via a notion of *polarity* (see Hepple, 1996). The statement of the compilation procedure below is based on the presentation in (Hepple, 1998).

the variable of a lambda expression, employing a version of substitution which specifically does not act to avoid 'accidental' binding.[4] Hence, in the final step of the proof, the variable z falls within the scope of the abstraction, and so becomes bound. (Note that this method requires care to be taken with variable identities, so that the only accidental binding to occur is that which is intended.)
A procedure τ for compiling an individual formula (plus its term, e.g. X : x) to indexed first-order form is specified by the following cases:

(τ0) $\tau(\mathrm{T}\colon t) = \tau(\langle i, \mathrm{T}, t\rangle)$ where i a fresh index

(τ1) $\tau(\langle \phi, \mathrm{X}, s\rangle) = \langle \phi, \mathrm{X}, s\rangle$ where X atomic

(τ2) $\tau(\langle \phi, \mathrm{Y} \multimap \mathrm{X}, s\rangle) = \tau(\langle \phi, (\mathrm{Y}\colon \emptyset) \multimap \mathrm{X}, s\rangle)$ where Y has no index set

(τ3) $\tau(\langle \phi, (\mathrm{Y}\colon \psi) \multimap \mathrm{X}_1, s\rangle) = \langle \phi, (\mathrm{Y}\colon \psi) \multimap \mathrm{X}_2, \lambda x.t\rangle \cup \Gamma$
where Y is atomic; x a fresh variable; $\tau(\langle \phi, \mathrm{X}_1, (sx)\rangle) = \langle \phi, \mathrm{X}_2, t\rangle \uplus \Gamma$

(τ4) $\tau(\langle \phi, ((\mathrm{Z} \multimap \mathrm{Y})\colon \psi) \multimap \mathrm{X}, s\rangle) = \tau(\langle \phi, (\mathrm{Y}\colon \pi) \multimap \mathrm{X}, \lambda y.s(\lambda z.y)\rangle) \cup \tau(\langle i, \mathrm{Z}, z\rangle)$
where i a fresh index; y, z fresh variables; $\pi = i \cup \psi$

4 Relating DTG and Type-logical Grammar

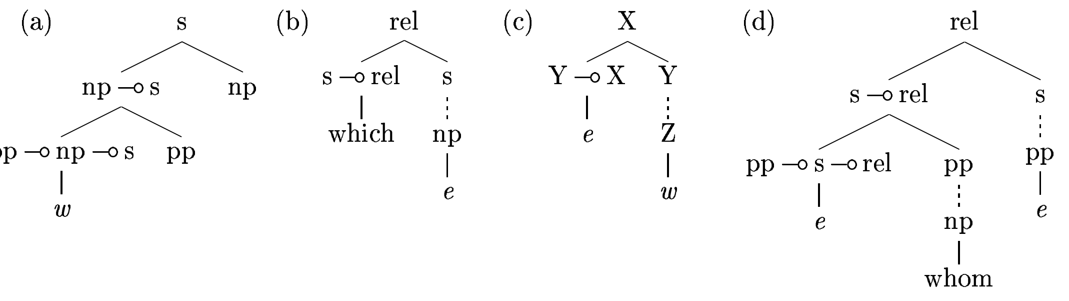
The above compilation produces results that bear more immediate similarities to the D-Tree approach than the original type-logical system, as discussed in Hepple (1998).[5] First-order formulae are easily viewed as tree fragments (in a way that higher-order formulae are not), e.g. a word w with formula pp $\multimap$ np $\multimap$ s might be viewed as akin to (6a) below (modulo the order of daughters which is not encoded). For a higher-order formula, the inclusion requirement between its first-order derivatives is analogous to a domination link within a d-tree, e.g. a relative pronoun (np $\multimap$ s) $\multimap$ rel (c.f. directional rel/(s/np)) would yield s $\multimap$ rel plus np, which we can view as akin to (6b).

By default, it is natural to associate the string of the initial formula with its main residue under compilation, as in (6b). Following proposals in (Moortgat, 1988; 1996), some categorial systems have used connectives $\uparrow$ ('extraction') and $\downarrow$ ('infixation'), where

[4]The move of allowing accidental binding is clearly not an innocuous one, but rather means that the terms associated with compiled formulae are not lambda terms in the conventional sense. For example, as an anonymous reviewer has emphasised, the term $\lambda t.x(\lambda z.t)$ in the proof above cannot be replaced with, say, $\lambda t.x(\lambda w.t)$, even though the two should be equivalent under α-conversion. When a subformula is excised from a higher-order formula during first-order compilation, the two are not then truly separate. Rather, an inclusion relation remains which is enforced by the indexing method. Such a relation implicitly obtains also for the lambda terms associated with the formulae, and so there is a sense in which the abstraction over z in $\lambda t.x(\lambda z.t)$ is *not* vacuous, but instead is an abstraction over the occurrence of z with the hypothetical, and in that case it makes sense that α-conversion should not apply (unless the hypothetical's variable is also simultaneously changed). At the end of a derivation, however, all variables will be within the scope of the lambdas that bind them, and the final lambda term is a 'normal' one, to which α-conversion can be freely applied. There is some extent of a parallel here with formalisms allowing partial descriptions of lambda expressions, such as the Constraint Language over Lambda Structures of (Egg *et al.*, 1998), used for underspecified semantics, which allows lambda bindings specified via dominance relations.

[5]See (Joshi *et al.*, 1997; Henderson, 1992) for other work connecting categorial formalisms (namely, the Lambek calculus and CCG, respectively) to tree-oriented formalisms.

$Y{\uparrow}Z$ is a "Y missing Z somewhere" and a type $X{\downarrow}(Y{\uparrow}Z)$ infixes its string to the position of the missing Z. Thus, a word *w* with type $X{\downarrow}(Y{\uparrow}Z)$ (c.f. the linear type $(Z \multimap Y) \multimap X$) compiles to $Y \multimap X$ and Z, abd is akin to (6c). As another example, the PP pied-piping relative pronoun type $\mathrm{rel}/(\mathrm{s}{\uparrow}\mathrm{pp}){\downarrow}(\mathrm{pp}{\uparrow}\mathrm{np})$, from (Morrill, 1992), which infixes to an NP site within a PP, is akin to (6d).

(6)


5 A Functional Approach to Interpreting DTG Derivations

The rest of this paper explores the idea of providing a functional semantics for DTG derivations, in a manner akin to that of categorial grammar.[6] The approach envisaged is one in which each tree fragment (i.e. maximal unit containing no dominance links) of an initial d-tree is associated with a lambda term. At the end of a derivation, the meaning of the resulting tree would be computed by working bottom up, applying the meaning term of each basic tree fragment to the meanings computed for each complete subtree added in at the fragment's frontier nodes, in some fixed fashion (e.g. such as in their right-to-left order). Strictly, terms would be combined using the special substitution operation of rule (5) (allowing variable capture in the manner discussed). Suitable terms to associate with tree fragments will be arrived at by exploiting the analogy between d-trees and higher-order formulae under compilation.

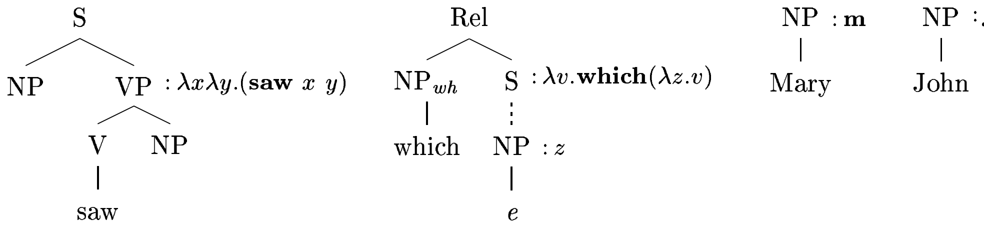
For example, consider a simple grammar consisting of the four d-trees in (7), of which only that for *which* has more than one fragment. Each tree fragment is associated with a meaning term, shown to the right of ":". The two fragments in the d-tree for *which* each have their own term, which are precisely those that would be assigned for the two compiled formulae in (6b) (assuming the meaning term for the precompilation formula rel/(s/np) to be just **which**).[7] This grammar allows the phrase-structure (8a) for *Mary saw John*, whose interpretation is produced by 'applying' the term for *saw* to that for the NP *John*

[6]See Muskens & Krahmer (1998) for an alternative approach to the functional interpretation of a DTG-like formalism, one which allows a treatment of quantification. Their approach extends the use of partial descriptions to the semantics, and employs a technique of internalising the logical binding mechanism, so as to allow partial tree descriptions to be associated with underspecied semantics.

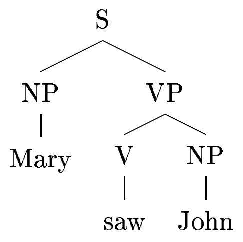
[7]A treatment of *wh*-movement based on this structure is useful for expositional purposes, but clearly differs from the standard TAG/DTG approach, where a moved *wh*-item originates with a structure that includes the governor of the extraction site (typically a verb that subcategorises for the moved item). Such structures present no problem for this approach, i.e. we could simply pre-combine the d-trees of *which* and *saw* given in (7), to give a single tree of two fragments (*which* being associated with the upper fragment, and both *saw* and the 'trace' with the lower fragment.

(i.e. the subtree added in at the rightmost frontier node of *saw*'s single tree fragment), and then to that of the NP *Mary*, giving **(saw j m)**, as shown in (8b). The grammar allows the tree (9a) for the relative clause *which Mary saw.* Here, the object position of *saw* is filled by the lower fragment of *which*'s d-tree, so that the subtree rooted at S has interpretation **(saw** z **m)**. Combining this with the term of the upper fragment of *which* gives interpretation **which(**λz.**saw** z **m)**, as shown in (9b).

(7)



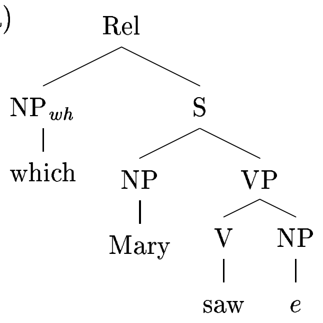
(8) (a)



(b) $[\,\lambda x\lambda y.(\mathbf{saw}\ x\ y)\ +\ \mathbf{j}\ +\ \mathbf{m}\,]$

$\rightsquigarrow\ (\mathbf{saw\ j\ m})$

(9) (a)

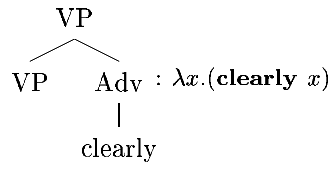


(b) $[\,\lambda v.\mathbf{which}(\lambda z.v)\ +$
$[\,\lambda x\lambda y.(\mathbf{saw}\ x\ y)\ +\ z\ +\ \mathbf{m}\,]\,]$

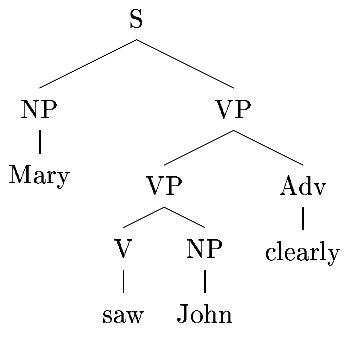
$\rightsquigarrow\ \mathbf{which}(\lambda z.(\mathbf{saw}\ z\ \mathbf{m}))$

The tree composition steps required to derive the trees above would be handled in DTG by the subsertion operation. As noted earlier, DTG has a second composition operation *sister-adjunction*, used in handling modification, which adds in a modifier subtree as an additional daughter to an already existing local tree. A key motivation for this operation is so that DTG derivation trees distinguish argument vs. modifier dependencies, so as to provide an appropriate basis for interpretation. Categorial grammars typically make no such distinction in syntactic derivation, where all combinations are simply of functions and arguments. Rather, the distinction is implicit as a property of the lexical meanings of the functions that participate. Accordingly, we recommend elimination of the sister-adjunction operation, with all composition being handled instead by subsertion. Thus, a VP modifying adverbial might have d-tree (10a), and give structures such as (10b). (Such structures are more in line with the standard TAG treatment than that of DTG.)

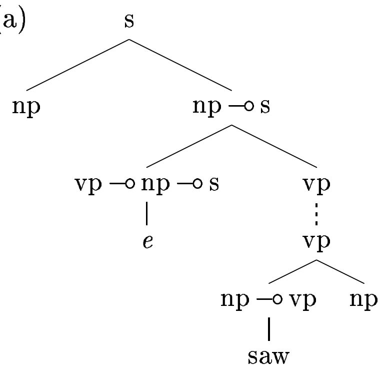
(10) (a)



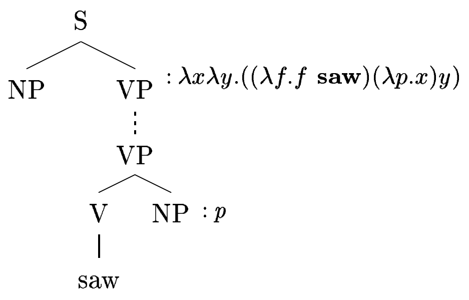
(b)



(11) (a)



(b)



Such an analysis requires a different lexical d-tree for *saw* to that in (7), one where the VP node is 'stretched' as in (11b) to allow possible inclusion of modifiers. As a basis for arriving at suitable functional semantics for (11b), consider the following. A categorial approach might make *saw* a functor (np\s)/np with semantics **saw**. This functor could be type-raised to (np\s)↓((np\s)↑((np\s)/np)) with semantics $(\lambda f.f\ \mathbf{saw})$. By substituting the two embedded occurrences of (np\s) with the atom vp we get (np\s)↓(vp↑(vp/np)), which compiles to first-order formulae as in (11a), which are analogous to the desired d-tree (11b), so providing the meaning terms there assigned. Using (11b) to derive the structure (8a) involves identifying the two VP nodes. Such a derivation gives an interpretation as in (12a), whilst a derivation of (10b) gives the interpretation (12b).

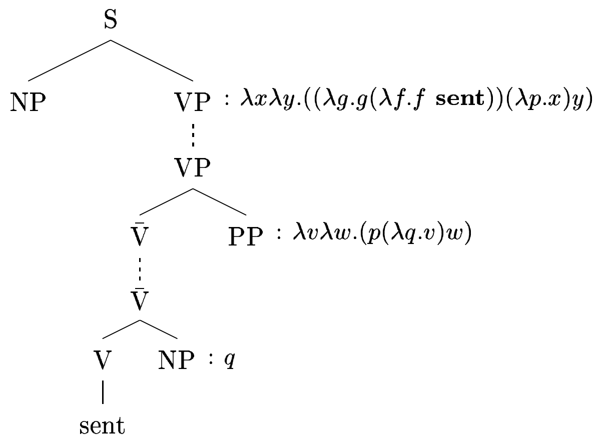
(12) (a) $[\ \lambda x \lambda y.((\lambda f.f\ \mathbf{saw})\,(\lambda p.x)\,y)\ +\ [\ \lambda u.pu\ +\ \mathbf{j}\ ]\ +\ \mathbf{m}\ ]\ \ \rightsquigarrow\ (\mathbf{saw\ j\ m})$

(b) $[\ \lambda x \lambda y.((\lambda f.f\ \mathbf{saw})\,(\lambda p.x)\,y)\ +\ [\ \lambda v.(\mathbf{clearly}\ v)\ +\ [\ \lambda u.pu\ +\ \mathbf{j}\ ]\ ]\ +\ \mathbf{m}\ ]$
$\rightsquigarrow\ (\mathbf{clearly}\ (\mathbf{saw\ j})\ \mathbf{m})$

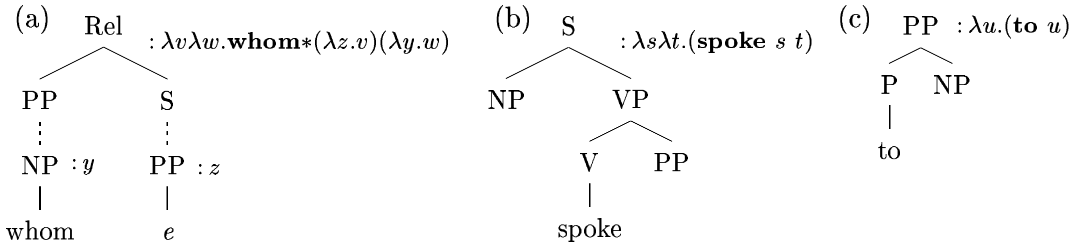
For a ditransitive verb, we might want a structure providing more than one locus for inclusion of modifiers, such as (13). The semantics provided for this d-tree is arrived at by a similar process of reasoning to that for the previous case, except that the initial categorial type of the verb is type-raised *twice* (hence the subterm $(\lambda g.g(\lambda f.f\ \mathbf{sent}))$ of the upper fragment's term).

Let us next consider a possible account that can be handled with this new approach to interpreting derivations that would be problematic under the standard approach. The compiled formulae in (6d), stemming from Morrill's proposal for handling pied-piping, are suggestive of the d-tree (14a) for a pied-piping relative pronoun. The semantics of this d-tree is given as if it were compiled from a linear formula for the relative pronoun having semantics **whom***. This symbol here stands in place of a more complex term $\lambda f \lambda g.\mathbf{whom}(\lambda x.f(gx))$ for the relative pronoun's semantics, which is one that puts together the meanings of the pied-piped material and of the sentence from which it is extracted, effectively performing reconstruction at the semantic level.

(13)

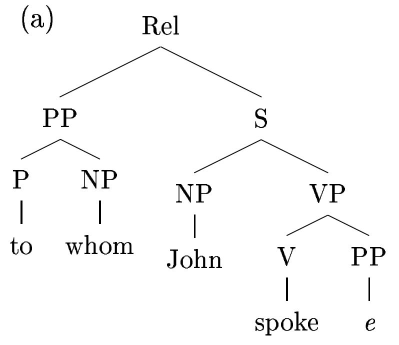

S
NP
VP : $\lambda x \lambda y.((\lambda g.g(\lambda f.f\ \mathbf{sent}))(\lambda p.x)y)$
VP
$\bar{V}$
PP : $\lambda v \lambda w.(p(\lambda q.v)w)$
$\bar{V}$
V
NP : q
sent

(14)


(a)
Rel : $\lambda v \lambda w.\mathbf{whom}{*}(\lambda z.v)(\lambda y.w)$
PP
S
NP : y
PP : z
whom
e
(b)
S : $\lambda s \lambda t.(\mathbf{spoke}\ s\ t)$
NP
VP
V
PP
spoke
(c)
PP : $\lambda u.(\mathbf{to}\ u)$
P
NP
to

Using the additional d-trees in (14b) and (14c), we can derive the relative clause *to whom John spoke* as in (15a), which receives the interpretation in (15b) that simplifies (when we substitute for **whom***) to $\mathbf{whom}(\lambda z.\mathbf{spoke}\ (\mathbf{to}\ z)\ \mathbf{j})$. This treatment is ruled out by the restrictions on the derivation process that are required for the standard treatment of DTG semantics, as the relative pronoun d-tree has two fragments that would need to be substituted into other structures in subsertion steps during the derivation.

(15) (a)



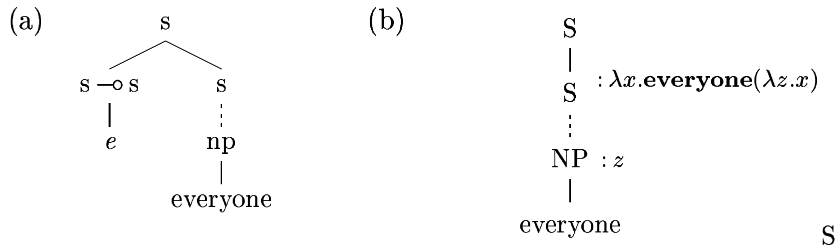
(b)

$$\begin{aligned}&[\ \lambda v \lambda w.\mathbf{whom}^*(\lambda z.v)(\lambda y.w)\\ &\quad +\ [\ \lambda s \lambda t.(\mathbf{spoke}\ s\ t)\ +\ z\ +\ \mathbf{j}\ ]\\ &\quad +\ [\ \lambda u.(\mathbf{to}\ u)\ +\ y\ ]\ ]\\ &\leadsto\ \mathbf{whom}^*(\lambda z.\mathbf{spoke}\ z\ \mathbf{j})(\lambda y.\mathbf{to}\ y)\\ &\leadsto\ \mathbf{whom}(\lambda z.\mathbf{spoke}\ (\mathbf{to}\ z)\ \mathbf{j})\end{aligned}$$

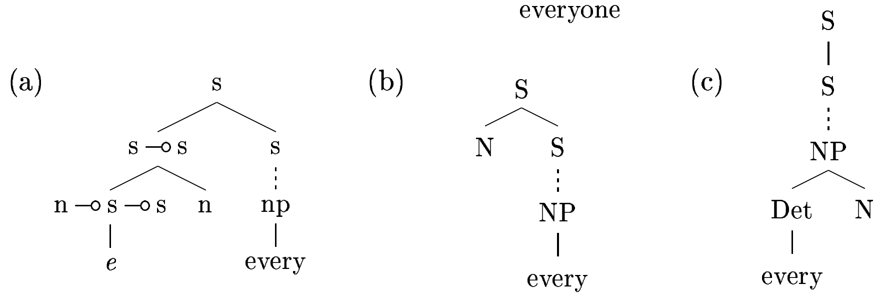
5.1 Limitations of the approach

We next consider a case that the outlined approach does not handle, which reveals something of its limitations: quantification. Following a suggestion of (Moortgat, 1996), the connectives $\uparrow$ ('extraction') and $\downarrow$ ('infixation') have been used in a categorial treatment of quantification. The lexical quantified NP *everyone*, for example, might be assigned type s$\downarrow$(s$\uparrow$np), so that it has scope at the level of some sentence node but its string appears in some NP position. First-order compilation yields the results (16a). The corresponding d-tree (16b) is unusual from a phrase-structure point of view in that its upper fragment is a purely interpretive projection, but would serve to produce appropriate interpretations.

(16)



(17)



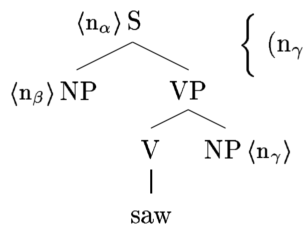
A simple quantifier *every* has type s$\downarrow$(s$\uparrow$np)/n, to combine firstly with a noun, with the combined string of *every*+noun then infixing to a NP position. First-order compilation, however, produces the result (17a), comparable to (17b), which is clearly an inappropriate structure. What we would hope for is a structure more like that in (17c), but although it is perfectly possible to specify an initial higher-order formula that produces first-order formulae comparable to this d-tree, the results do not provide a suitable basis for interpretation. More generally, the highly restrictive approach to semantic composition that is characteristic of the approach outlined is such that a fragment cannot have scope above its position in structure (although a d-tree having multiple fragments has access to multiple possible scopes). This means, for example, that *no* semantics for (17c) will be able to get hold of and manipulate the noun's meaning as something separate from that of the sentence predicate, rather the former must fall within the latter.

6 A Glue-like Treatment of Interpretation

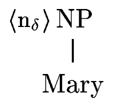
One possibility for overcoming the limitations of the approach that were discussed in the previous section would be to loosen the tight coupling between derivations and the functional semantics, perhaps in a manner akin to the glue language method for interpreting

Lexical-Functional Grammar derivations (Dalrymple *et al.*, 1993). We shall next briefly, and *informally*, sketch such a proposal. Each lexical d-tree is associated with one or more lambda terms which bear linear types, in which the type atoms that appear correspond to node identities in the associated d-tree,[8] as in the following (where the correspondence of type atoms to nodes is shown in the tree by markers in angle brackets beside nodes):

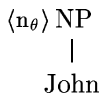
(18)



$\left\{ (\mathrm{n}_\gamma \multimap \mathrm{n}_\beta \multimap \mathrm{n}_\alpha) : \lambda x \lambda y.(\mathbf{saw}\ x\ y) \right.$



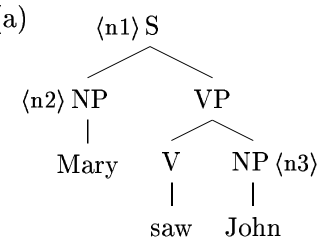
$\left\{ \mathrm{n}_\delta : \mathbf{m} \right.$



$\left\{ \mathrm{n}_\theta : \mathbf{j} \right.$

Given a derived tree such as (19a), we can 'fix' node identities, arriving at fully specified typed lambda terms as in (19b). These terms combine to produce a term with type n1 (i.e. the root node of the tree), corresponding to the meaning for the entire sentence. Note how the correspondence between tree nodes and types serves to restrict deduction over the lambda terms to produce results appropriate to the syntactic derivation.

(19) (a)



(b) $(\mathrm{n3} \multimap \mathrm{n2} \multimap \mathrm{n1}) : \lambda x \lambda y.(\mathbf{saw}\ x\ y)$
$\mathrm{n3} : \mathbf{j}$
$\mathrm{n2} : \mathbf{m}$

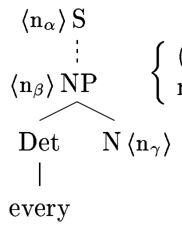
$\rightsquigarrow \quad \mathrm{n1} : (\mathbf{saw\ j\ m})$

The benefit of having a glue-like separation of structures and formulae comes from the additional freedom a glue-like approach allows in the mapping from syntactic structures to typed terms. This can be seen with the following d-tree for the quantifier *every*, where the meaning of the noun that will appear within the lower fragment is marked to be fed directly to the major meaning term for the quantifier (rather than being 'trapped' within some embedded interpretation for the noun phrase), and where node atoms can be reused to create endocentric functors such as $(\mathrm{n1} \multimap \mathrm{n1})$ (a feature which makes it possible for the upper fragment of the d-tree to be just a single node, rather than a S-over-S interpretive projection as in (16a)).[9]

[8]The LFG glue analysis uses a mapping ('σ-projection function') from syntactic analyses (in their case f-structures rather than phrase structure trees) to glue terms. Note that the clean separation of types and lambda terms in the above proposal makes it more in line with the 'categorial-style' formulation of the LFG glue account given in (Dalrymple *et al*, 1997), rather than its standard formulation.

[9]See (Shieber & Schabes, 1990) for a *Synchronous TAG* treatment of quantification, where the semantics is treated as a second system of tree representations that are operated upon synchronously with syntactic trees. Although operations upon syntactic and semantic representations in that account are *synchronous*, they are not *parallel* in the way that is rigidly required in categorial semantics (and in the initial formulation of the functional interpretation scheme for DTG). The non-parallelism that their approach allows is strongly

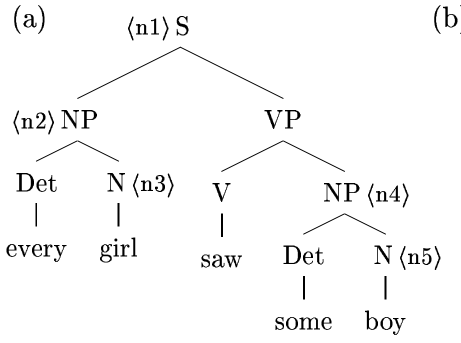
(20)



$$\begin{cases} (\mathrm{n}_\gamma \multimap \mathrm{n}_\alpha \multimap \mathrm{n}_\alpha) : \lambda u \lambda v.(\textbf{every}\ u\ (\lambda w.v)) \\ \mathrm{n}_\beta : w \end{cases}$$

Given a similar lexical d-tree for *some*, we can derive *Every girl saw some boy* as in (21a), yielding the formulae in (21b), which combine to give a result of type n1 in two alternative ways, corresponding to the alternative scope readings of the sentence, shown in (21c).

(21) (a)



(b)
$(\mathrm{n3} \multimap \mathrm{n1} \multimap \mathrm{n1}) : \lambda u \lambda v.(\textbf{every}\ u\ (\lambda w.v))$
$\mathrm{n2} : w$
$(\mathrm{n5} \multimap \mathrm{n1} \multimap \mathrm{n1}) : \lambda r \lambda s.(\textbf{some}\ r\ (\lambda t.s))$
$\mathrm{n4} : t$
$(\mathrm{n4} \multimap \mathrm{n2} \multimap \mathrm{n1}) : \lambda x \lambda y.(\textbf{saw}\ x\ y)$
$\mathrm{n3} : \textbf{girl}$
$\mathrm{n5} : \textbf{boy}$

(c) $\rightsquigarrow$ $\mathrm{n1} : (\textbf{every girl}\ (\lambda w.(\textbf{some boy}\ (\lambda t.(\textbf{saw}\ t\ w)))))$
$\rightsquigarrow$ $\mathrm{n1} : (\textbf{some boy}\ (\lambda t.(\textbf{every girl}\ (\lambda w.(\textbf{saw}\ t\ w)))))$

7 Conclusion

Two approaches for interpreting DTG derivations have been described, both of which allow DTG's current process-based interpretation model to be dispensed with, along with the constraints it requires, and so allow the possibility of formulating accounts within the formalism that would otherwise be excluded.

The first method involves associating the tree fragments in lexical d-trees with lambda terms, with the meaning for a derivation being computed by combining the lambda terms of the basic tree fragments in a manner that is fixed by the form of the derived phrase structure. The initial lambda terms are arrived at by exploiting an analogy between d-trees and the results of compiling higher-order linear formulae under the method of (Hepple, 1996). The success of the method is suggestive of some real content to the analogy, i.e. that d-trees are, in some sense, higher-order objects.

The second method is a variant of the first inspired by glue language work, which gains some freedom in assembling basic meanings, enabling a treatment of quantification, whilst losing some of the first method's simplicity.

comparable to the freedom gained in the move to a glue-like treatment above, although a glue-like approach is chosen here as it is more in keeping with a functional analysis. Similar non-parallelism is allowed by the Muskens & Krahmer (1998) account mentioned earlier, and is again crucially important in allowing quantification to be handled.

References

Becker, T., Joshi, A. & Rambow, O. 1991. 'Long distance scrambling and tree adjoining grammars. *Proc. EACL–91.*

Candito, M.–H. & Kahane, S. 1998a. 'Can the TAG derivation tree represent a semantic graph? An answer in the light of Meaning-Text Theory.' *Proc. Fourth Workshop on Tree-Adjoining Grammars and Related Frameworks (TAG+4).*

Candito, M.–H. & Kahane, S. 1998b. 'Defining DTG derivations to get semantic graphs.' *Proc. Fourth Workshop on Tree-Adjoining Grammars and Related Frameworks.*

Dalrymple, M., Lamping, J. & Saraswat, V. 1993. 'LFG semantics via constraints.' *Proc. EACL–6*, Utrecht.

Dalrymple, M., Gupta, V., Lamping, J. & Saraswat, V. 1997. 'Relating resource-based semantics to categorial semantics.' *Proc. MOL–5.*

Egg, M., Niehren, J., Ruhrberg, P. & Xu, F. 1998. 'Contraints over Lambda-Structures in Semantic Underspecification.' *Proc. COLING–ACL'98 Joint Conference.*

Gabbay, D. 1996. *Labelled deductive systems. Volume 1.* Oxford University Press.

Henderson, J. 1992. 'A Structural Interpretation of CCG.' UPenn Technical Report, MS-CIS-92-49.

Hepple, M. 1996. 'A Compilation-Chart Method for Linear Categorial Deduction.' *Proc. COLING–96.*

Hepple, M. 1998. 'Memoisation for Glue Language Deduction and Categorial Parsing.' *Proc. COLING–ACL'98 Joint Conference.*

Hepple, M. 1998. 'On Some Similarities Between D-Tree Grammars and Type-Logical Grammars.' *Proc. Fourth Workshop on Tree-Adjoining Grammars and Related Frameworks.*

Joshi, A., Levy, L. & Takahashi, M. 1975. Tree Adjunct Grammars. *Journal of the Computer and System Science*, **10**.

Joshi, A. & Kulick, S. 1997. 'Partial proof trees as building blocks for a categorial grammar.' *Linguistics and Philosophy.*

Lambek, J. 1958. 'The mathematics of sentence structure.' *American Mathematical Monthly*, **65**.

Moortgat, M. 1988. *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus.* Foris, Dordrecht.

Moortgat, M. 1996. 'Generalized quantifiers and discontinuous constituency.' H. Bunt and A. van Horck (eds). *Discontinuous Constituency*, Mouton de Gruyter.

Morrill, G. 1992. 'Categorial Formalisation of Relativisation: Pied Piping, Islands and Extraction Sites.' Research Report LSI-92-23-R, Universitat Politécnica de Catalunya.

Muskens, R. & Krahmer, E. 1998. 'Description Theory, LTAGs and Underspecified Semantics.'*Proc. Fourth Workshop on Tree-Adjoining Grammars and Related Frameworks.*

Rambow, O., Vijay-Shanker, K. & Weir, D. 1995. 'D-Tree Grammars.' *Proc. ACL–95.*

Shieber, S.M. & Schabes, Y. 1990. 'Synchronous tree-adjoining grammar.' *Proceedings of COLING–90.*