On Some Similarities Between D-Tree Grammars and Type-Logical Grammars

Mark Hepple
Department of Computer Science, University of Sheffield,
Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK
hepple@dcs.shef.ac.uk

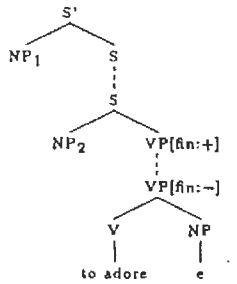
1 Introduction

This paper discusses some similarities between D-Tree Grammars and type-logical grammars that are suggested in the context of a parsing approach for the latter that involves compiling higher-order formulae to first-order formulae.[1] This comparison suggests an approach to providing a functional semantics for D-Tree derivations, which is outlined.

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 between nodes in different fragments (that link them into a single graph).

(1)



The above example d-tree, drawn from (Rambow *et al.*, 1995), allows topicalisation of the verb's object, as in (e.g.) *Hotdogs$_i$, he claims Mary seems to adore* t_i, where NP_1 is the fronted object, and NP_2 the verb's subject.[2] The main operation[3] for composing d-trees is *subsertion*, which, loosely, combines two d-trees to produce another, by substituting a fragment of one at a suitable node in the other, with other (dominating) fragments of the first being intercalated into domination links of the second. The approach is motivated by problems of related formalisms (such as TAG and MCTAG-DL[4]) involving linguistic coverage and the semantic interpretation of derivations.

3 Type-logical Grammar

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 the implicational or multiplicative[5] fragment of linear logic. For this paper, we can ignore labellings, and instead focus on the 'core functional structure' projected by linear formulae.[6]

4 Implicational Linear Logic & First-order Compilation

In linear logic proofs, each assumption is used precisely once. Natural deduction rules of *elimination* and *introduction* for linear implication ($\multimap$) are:[7]

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

The proof in (3) illustrates 'hypothetical reasoning', where an additional assumption, or 'hypothetical', is used that is latter 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,

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

[2] The indexation is my own, for expositional purposes.

[3] A second operation, *sister-adjunction*, used in handling modification, is discussed later in the paper.

[4] Multi-Component TAG with Domination Links (Becker *et al.*, 1991).

[5] The multiplicative fragment extends the implicational one with $\otimes$ ('tensor'), akin to the Lambek product.

[6] This means, most notably, that the representations discussed lack any encoding of linear order requirements, which would be handled within the labelling system.

[7] Eliminations and introductions correspond to steps of functional application and abstraction, respectively, as the lambda-term labelling reveals. In the $\multimap$I rule, [B] indicates a discharged or withdrawn assumption.

e.g. Z in (3) is a subformula of Xo–(Yo–Z).[8]

$$(3)\quad \dfrac{\mathrm{X}:x(\lambda z.y(wz)) \qquad \dfrac{\mathrm{Y}:y(wz)}{\mathrm{Yo–Z}:\lambda z.y(wz)}}{\mathrm{X}:x(\lambda z.y(wz))}$$

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 o–E). The compilation reduces higher-order formulae to first-order formulae by *excising* subformulae corresponding to hypotheticals, e.g. so Xo–(Yo–Z) gives Xo–Y *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 Xo–Y. Each compiled formula has an index set with one member (e.g. $\{j\}:\mathrm{Z}$), which serves as its unique identifier. The index set of a derived formula identifies the assumptions used to derive it. The single inference rule (4) 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 $\alpha \subseteq \psi$.

$$\dfrac{\{i\}:\mathrm{Xo–(Y:\{}j\mathrm{\})}\ \lambda t.x(\lambda z.t) \qquad \dfrac{\{k\}:\mathrm{Yo–(W:}\emptyset\mathrm{)}\ \lambda u.yu \qquad \dfrac{\{l\}:\mathrm{Wo–(Z:}\emptyset\mathrm{)}\ \lambda v.wv \qquad \{j\}:\mathrm{Z}\ z}{\{j,l\}:\mathrm{W}:wz}}{\{j,k,l\}:\mathrm{Y}:y(wz)}}{\{i,j,k,l\}:\mathrm{X}:x(\lambda z.y(wz))}$$

In proving Xo–(Yo–Z), Yo–W, Wo–Z ⇒ X, for example, compilation yields the assumption formulae of the proof above. The leftmost (F1) and rightmost (F2) assumptions both come from Xo–(Yo–Z), and F1 requires its argument to include F2. Compilation has removed the need for an explicit introduction step in the proof, c.f. proof (3), 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 (4) is handled not by simple application, but rather direct substitution for the variable of a lambda expression, employing a version of substitution which specifically does not act to avoid accidental binding. Hence, in the final step of the proof, the variable z falls within the scope of the abstraction, and so becomes bound.

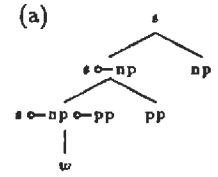
$$(4)\quad \dfrac{\phi:\mathrm{Ao–(B:}\alpha\mathrm{)}:\lambda v.a \qquad \psi:\mathrm{B}:b}{\pi:\mathrm{A}:a[b/\!\!/v]} \qquad \begin{array}{l}\pi = \phi \uplus \psi \\ \alpha \subset \psi\end{array}$$

[8]The relevant subformulae can be precisely characterised in terms of a notion *polarity*: hypotheticals correspond to maximal positive-polarity subformulae of higher-order formulae. See (Hepple, 1996) for details.

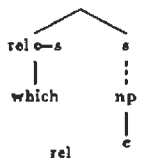
5 Relating The Two Systems

The above compilation produces results that bear more immediate similarities to the D-Tree approach than the original type-logical system. 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 so–npo–pp might be viewed as akin to (5a) 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 rel/(s/np) would yield rel o–s plus np, which we can view as akin to (5b).

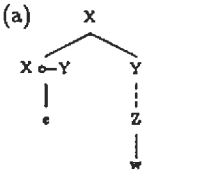
(5) (a)



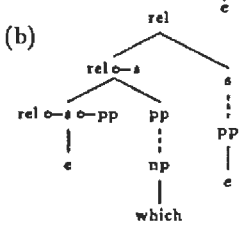
(b)



(6) (a)



(b)



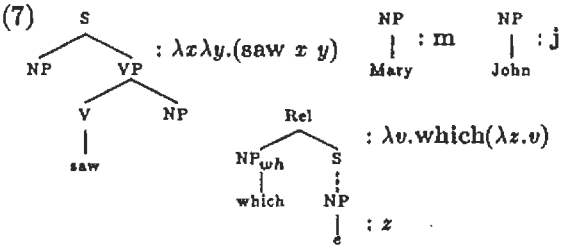
By default, it is natural to associate the string of the initial formula with its main residue under compilation, as in (5b). Following proposals in (Moortgat, 1988; 1996), some categorial systems have used connectives ↑ ('extraction') and ↓ ('infixation'), where Y↑Z is a "Y missing Z somewhere" and a type X↓(Y↑Z) infixes its string to the position of the missing Z. Thus, a word w with type X↓(Y↑Z), compiling to Xo–Y and Z, is akin to (6a). For example, the PP pied-piping relative pronoun type rel/(s↑pp)↓(pp↑np), from (Morrill, 1992), which infixes to an NP site within a PP, is akin to (6b).

6 A Functional Approach to Interpreting DTG Derivations

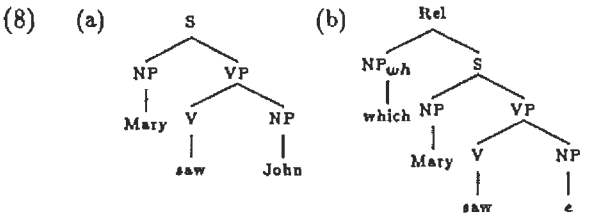
The rest of this paper explores the idea of providing a functional semantics for DTG derivations, or rather of some DTG-like formalism, in a manner akin to that of categorial grammar. The approach envisaged is one in which each tree fragment (i.e. maximal unit containing no dominance links) of an initial 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 (4) (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.

(7)



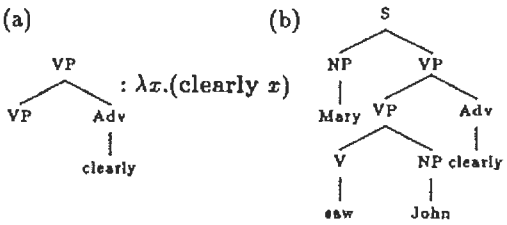
(8)



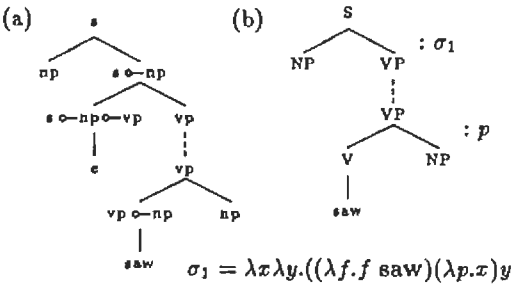
For example, consider a simple grammar consisting of the four d-trees in (7), of which only that for *which* has more than one fragment. Each tree fragment is associated with a meaning term, shown to the right of ":". The two fragments in the d-tree for *which* each have their own term, which are precisely those that would be assigned for the two compiled formulae in (5b) (assuming the meaning term for the precompilation formula rel/(s/np) to be just **which**). 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* (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**). The grammar allows the tree (8b) for the relative clause *which Mary saw*.[9] 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**).

The tree composition steps required to derive the trees in (8) 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.[10] 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 (9a), and give structures such as (9b).[11]

(9)



(10)



Such an analysis requires a different lexical d-tree for *saw* to that in (7), one where the VP node is 'stretched' as in (10b) to allow possible inclusion of modifiers. As a basis for arriving at suitable functional semantics for (10b), 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 ($\lambda f.f$ **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 (10a), which are analogous to the desired d-tree (10b), so providing the meaning terms there assigned. Using (10b) to derive the structure (8a) involves identifying the two

[9] The treatment of *wh*-movement here exemplified 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 precombine the d-trees of *which* and *saw* given in (7).

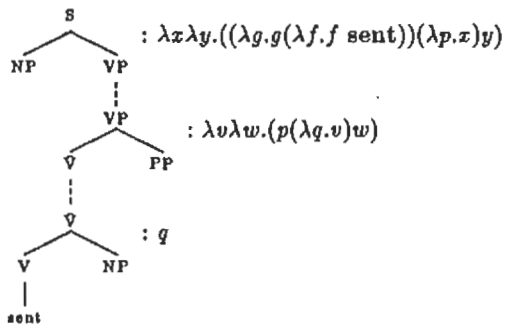
[10] This is not to say that the distinction has no observable reflex: modifiers are in general recognisable as endocentric categorial functors (i.e. having the same argument and result type).

[11] Such an analysis is more in line with the standard TAG treatment than that of DTG.

VP nodes. Such a derivation gives the interpretation $((\lambda f.f\ \textbf{saw})(\lambda p.p\ \textbf{j})\textbf{m})$ which simplifies to (**saw j m**). A derivation of (9b) gives interpretation $((\lambda f.f\ \textbf{saw})(\lambda p.\textbf{clearly}(p\ \textbf{j}))\textbf{m})$ which simplifies to (**clearly (saw j) m**).

For a ditransitive verb, we might want a structure providing more than one locus for inclusion of modifiers, such as (11). The semantics provided for this d-tree is arrived at by a similar process of reasoning to that for the previous case, except that it involves type-raising the initial categorial type of the verb *twice* (hence the subterm $(\lambda g.g(\lambda f.f\ \textbf{sent}))$ of the upper fragment's term).

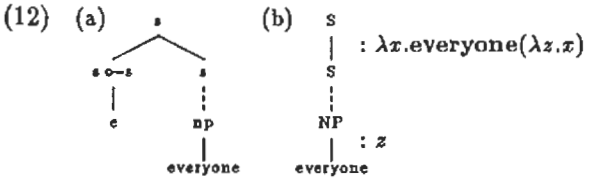
(11)



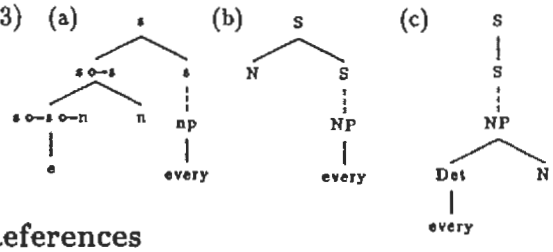
The interpretation approach outlined appears quite promising so far. We next consider a case it does not handle, which reveals something of its limitations: quantification. Following a suggestion of (Moortgat, 1996), the connectives ↑ ('extraction') and ↓ ('infixation') have been used in a categorial treatment of quantification. The lexical quantified NP *everyone*, for example, might be assigned type s↓(s↑np), so that it has scope at the level of some sentence node but its string will appear in some NP position. First-order compilation yields the results (12a). The corresponding d-tree (12b) is unusual from a phrase-structure point of view in that it's upper fragment is a purely interpretive projection, but this d-tree would serve to produce appropriate interpretations. So far so good.

A simple quantifier *every* has type s↓(s↑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 (13a), comparable to the d-tree (13b), which is clearly an inappropriate structure. What we would hope for is a structure more like that in (13c), 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 (13c) will be able to get hold of and manipulate the noun's meaning as something separate from that of the sentence predicate (c.f. s↑np), rather the former must fall within the latter.[12]

(12)



(13)



References

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

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

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

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

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.

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

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

[12] See (Shieber & Schabes, 1990) for a treatment of quantification within the Synchronous TAG formalism, in which the semantics is treated as a second system of tree representations that are operated upon synchronously with syntactic trees. Their account cannot be adapted to the present approach because their operations upon syntactic and semantics representations, though *synchronous*, are not *parallel* in the way that is rigidly required in categorial semantics.