TYPE-RAISING AND DIRECTIONALITY IN COMBINATORY GRAMMAR*

Mark Steedman
Computer and Information Science, University of Pennsylvania
200 South 33rd Street
Philadelphia PA 19104-6389, USA
(Internet: steedman@cis.upenn.edu)

ABSTRACT

The form of rules in combinatory categorial grammars (CCG) is constrained by three principles, called "adjacency", "consistency" and "inheritance". These principles have been claimed elsewhere to constrain the combinatory rules of composition and type raising in such a way as to make certain linguistic universals concerning word order under coordination follow immediately. The present paper shows that the three principles have a natural expression in a unification-based interpretation of CCG in which directional information is an attribute of the arguments of functions grounded in string position. The universals can thereby be derived as consequences of elementary assumptions. Some desirable results for grammars and parsers follow, concerning type-raising rules.

PRELIMINARIES

In Categorial Grammar (CG), elements like verbs are associated with a syntactic "category", which identifies their functional type. I shall use a notation in which the argument or domain category always appears to the right of the slash, and the result or range category to the left. A forward slash / means that the argument in question must appear on the right, while a backward slash \ means it must appear on the left.

(1) enjoys := $(S\backslash NP)/NP$

The category $(S\backslash NP)/NP$ can be regarded as both a syntactic and a semantic object, in which symbols like S are abbreviations for graphs or terms including interpretations, as in the unification-based categorial grammars of Zeevat et al. [8] and others (and cf. [6]).

Such functions can combine with arguments of the appropriate type and position by rules of functional application, written as follows:

(2) *The Functional Application Rules:*

a. $X/Y \quad Y \Rightarrow X \quad (>)$

b. $Y \quad X\backslash Y \Rightarrow X \quad (<)$

Such rules are also both syntactic and semantic rules of combination in which X and Y are abbreviations for more complex objects which combine via unification. They allow context-free derivations like the following (the application of rules is indicated by indices $>, <$ on the underlines):

(3)
```
Mary    enjoys    musicals
----  ---------  --------
 NP   (S\NP)/NP      NP
      ---------------->
             S\NP
      -------------<
            S
```

The derivation can be assumed to build a compositional interpretation, (*enjoy' musicals'*) *mary'*, say.

Coordination can be included in CG via the following rule, allowing constituents of like type to conjoin to yield a single constituent of the same type:

(4) $X \quad conj \quad X \Rightarrow X$

(5)
```
I     love     and    admire   musicals
-- --------- ---- --------- --------
NP (S\NP)/NP conj (S\NP)/NP     NP
   ------------------------&
             (S\NP)/NP
```

The rest of the derivation is exactly as in (3).

In order to allow coordination of contiguous strings that do not constitute constituents, CCG allows certain operations on functions related to Curry's combinators [1]. Functions may *compose*, as well as apply, under rules like the following:

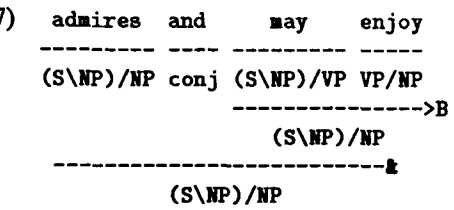
(6) *Forward Composition:*

$X/Y \quad Y/Z \Rightarrow_{\mathbf{B}} X/Z \quad (> \mathrm{B})$

The rule corresponds to Curry's combinator **B**, as the subscripted arrow indicates. It allows sentences like *Mary admires, and may enjoy, musicals* to be accepted, via the functional composition of two verbs (indexed as >B), to yield a composite of the same category as a transitive verb. Crucially, composition also yields the appropriate interpretation for the composite verb *may prefer* in this sentence (the rest of the derivation is as in (3)):

*Thanks to Michael Niv and Stu Shieber. Support from: NSF Grant CISE IIP CDA 88-22719, DARPA grant no. N0014-90-J-1863, and ARO grant no. DAAL03-89-C0031.

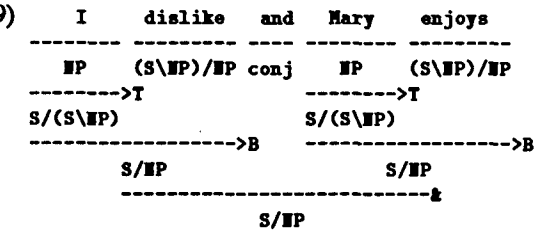
(7)



CCG also allows type-raising rules, related to the combinator **T**, which turn arguments into functions over functions-over-such-arguments. These rules allow arguments to compose, and thereby take part in coordinations like *I dislike, and Mary enjoys, musicals*. They too have an invariant compositional semantics which ensures that the result has an appropriate interpretation. For example, the following rule allows such conjuncts to form as below (again, the remainder of the derivation is omitted):

(8) *Subject Type-raising:*
$NP : y \Rightarrow_{\mathbf{T}} S/(S\backslash NP) \quad (> \mathrm{T})$

(9)



This apparatus has been applied to a wide variety of phenomena of long-range dependency and coordinate structure (cf. [2], [5], [6]).[1] For example, Dowty proposed to account for the notorious "non-constituent" coordination in (10) by adding two rules that are simply the backward mirror-image versions of the composition and type raising rules already given (they are indicated in the derivation by <B and <T).[2] This is a welcome result: not only do we capture a construction that has been resistant to other formalisms. We also satisfy a prediction of the theory, for the two backward rules are clearly expected once we have chosen to introduce their mirror image originals. The earlier papers show that, *provided type raising is limited to the two "order preserving" varieties exemplified in these examples*, the above reduction is the only one permitted by the lexicon of English. A number of related cross-linguistic regularities in the dependency of gapping upon basic word order follow ([2], [6]). The construction also strongly suggests that *all* NPs (etc.) should be considered as type raised, preferably in the lexicon, and that categories like NP should not reduce at all. However, this last proposal seems to implies a puzzling extra ambiguity in the lexicon, and for the moment we will continue to view type-raising as a syntactic rule.

The universal claim depends upon type-raising being limited to the following schemata, which do not of themselves induce new constituent orders:

(11) $X \Rightarrow_{\mathbf{T}} \mathrm{T}/(\mathrm{T}\backslash X)$
$X \Rightarrow_{\mathbf{T}} \mathrm{T}\backslash(\mathrm{T}/X)$

If the following patterns (which allow constituent orders that are not otherwise permitted) were allowed, the regularity would be unexplained, and without further restrictions, grammars would collapse into free order:

(12) $X \Rightarrow_{\mathbf{T}} \mathrm{T}/(\mathrm{T}/X)$
$X \Rightarrow_{\mathbf{T}} \mathrm{T}\backslash(\mathrm{T}\backslash X)$

But what are the principles that limit combinatory rules of grammar, to include (11) and exclude (12)?

The earlier papers claim that all CCG rules must conform to three principles. The first is called the *Principle of Adjacency* [5, p.405], and says that rules may only apply to *string-adjacent non-empty categories*. It amounts to the assumption that combinators will do the job. The second is called the *Principle of Directional Consistency*. Informally stated, it says that *rules may not override the directionality on the "cancelling" Y category in the combination*. For example, the following rule is excluded:

(13) $* \quad X\backslash Y \quad Y \quad => \quad X$

The third is the *Principle of Directional Inheritance*, which says that *the directionality of any argument in the result of a combinatory rule must be the same as the directionality on the corresponding argument(s) in the original functions*. For example, the following composition rule is excluded:

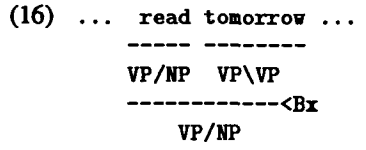
(14) $* \quad X/Y \quad Y/Z \quad => \quad X\backslash Z$

However, rules like the following *are* permitted:

(15) $Y/Z \quad X\backslash Y \quad => \quad X/Z \quad (< \mathrm{Bx})$

This rule (which is not a theorem in the Lambek calculus) is used in [5] to account for examples like *I shall buy today and read tomorrow, the collected works of Proust*, the crucial combination being the following:

(16)



The principles of consistency and inheritance amount

[1] One further class of rules, corresponding to the combinator **S**, has been proposed. This combinator is not discussed here, but all the present results transfer to those rules as well.

[2] This and other long examples have been "floated" to later positions in the text.

to the simple statement that *combinatory rules may not contradict the directionality specified in the lexicon.* But how is this observation to be formalised, and how does it bear on the type-raising rules? The next section answers these questions by proposing an interpretation, grounded in string positions, for the symbols / and \ in CCG. The notation will temporarily become rather heavy going, so it should be clearly understood that *this is not a proposal for a new CCG notation.* It is a semantics for the metagrammar of the *old* CCG notation.

DIRECTIONALITY IN CCG

The fact that directionality of arguments is inherited under combinatory rules, under the third of the principles, strongly suggests that it is a property of arguments themselves, just like their categorial type, NP or whatever, as in the work of Zeevat et al. [8][9]. However, the feature in question will here be grounded in a different representation, with significantly different consequences, as follows. The basic form of a combinatory rule under the principle of adjacency is $\alpha\ \beta\ \Rightarrow\ \gamma$. However, this notation leaves the linear order of α and β implicit. We therefore temporarily expand the notation, replacing categories like NP by 4-tuples, of the form $\{\alpha, DP_\alpha, L_\alpha, R_\alpha\}$, comprising: a) a *type* such as NP; b) a *Distinguished Position*, which we will come to in a minute; c) a *Left-end* position; and d) a *Right-end* position. The Principle of Adjacency finds expression in the fact that all legal combinatory rules must have the the form in (17), in which the right-end of α is the same as the left-end of β: We will call the position P_2, to which the two categories are adjacent, the *juncture.*

The Distinguished Position of a category is simply the one of its two ends that coincides with the juncture when it is the "cancelling" term Y. A rightward combining function, such as the transitive verb *enjoy*, specifies the distinguished position of its argument (here underlined for salience) as being that argument's *left*-end. So this category is written in full as in (18)a, using a *non*-directional slash /. The notation in (a) is rather overwhelming. When positional features are of no immediate relevance in such categories, they will be suppressed. For example, when we are thinking of such a function *as* a function, rather than as an argument, we will write it as in (18)b, where VP stands for $\{VP, DP_{VP}, L_{VP}, R_{VP}\}$, and the distinguished position of the verb is omitted. It is important to note that while the binding of the NP argument's Distinguished Position to its left hand end L_{np} means that *enjoy* is a rightward function, the distinguished position is *not* bound to the right hand end of the verb, R_{verb}. It follows that the verb can potentially combine with an argument elsewhere, just so long as it is to the right. This property was crucial to the earlier analysis of heavy NP shift. Coupled with the parallel independence in the position of the result from the position of the verb, it is the point at which CCG parts company with the directional Lambek calculus, as we shall see below.

In the expanded notation the rule of forward application is written as in (19). The fact that the distinguished position must be one of the two ends of an argument category, coupled with the requirement of the principle of Adjacency, means that *only* the two order-preserving instances of functional application shown in (2) can exist, and only consistent categories can unify with those rules.

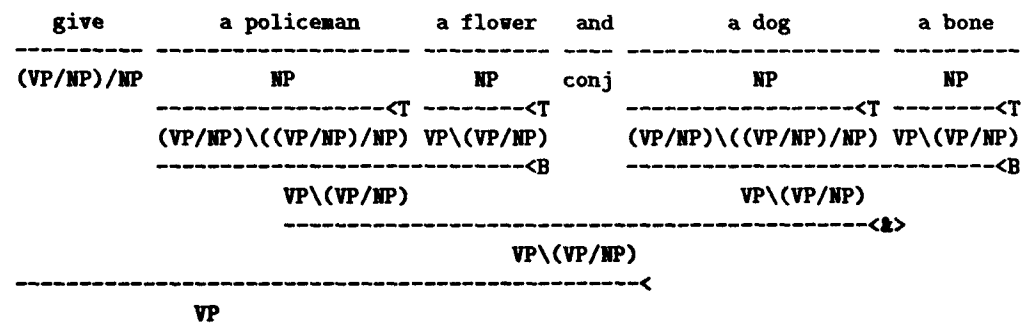
A combination under this rule proceeds as follows. Consider example (20), the VP *enjoy musicals*. The derivation continues as follows. First the positional variables of the categories are bound by the positions in which the words occur in the string, as in (21), which in the first place we will represent explicitly, as numbered string positions.[3] Next the combinatory rule (19) applies, to unify the argument term of the function with the real argument, binding the remaining positional variables including the distinguished position, as in (22) and (23). At the point when the combinatory rule applies, the constraint implicit in the distinguished position must actually hold. That is, the distinguished position must be adjacent to the functor.

Thus the Consistency property of combinatory rules follows from the principle of Adjacency, embodied in the fact that all such rules identify the distinguished position of the argument terms with the juncture $P2$, the point to which the two combinands are adjacent, as in the application example (19).

The principle of Inheritance also follows directly from these assumptions. The fact that rules correspond to combinators like composition forces directionality to be inherited, like any other property of an argument such as being an NP. It follows that only instances of the two very general rules of composition shown in (24) are allowed, as a consequence of the three Principles. To conform to the principle of consistency, it is necessary that L_y and R_y, the ends of the cancelling category Y, be distinct positions – that is, that Y not be coerced to the empty string. This condition is implicit in the Principle of Adjacency (see above), although in the notation of

[3] Declaritivising position like this may seem laborious, but it is a tactic familiar from the DCG literature, from which we shall later borrow the elegant device of encoding such positions implicitly in difference-lists.

(10)



(17) $\{\alpha, DP_\alpha, P_1, P_2\} \quad \{\beta, DP_\beta, P_2, P_3\} \quad \Rightarrow \quad \{\gamma, DP_\gamma, P_1, P_3\}$

(18) a. *enjoy* $:= \{\{VP, DP_{VP}, L_{VP}, R_{VP}\}/\{NP, \underline{L_{np}}, L_{np}, R_{np}\}, DP_{verb}, L_{verb}, R_{verb}\}$

b. *enjoy* $:= \{VP/\{NP, \underline{L_{np}}, L_{np}, R_{np}\}, L_{verb}, R_{verb}\}$

(19) $\{\{X, DP_x, P1, P3\}/\{Y, P2, P2, P3\}, P1, P2\} \quad \{Y, P2, P2, P3\} \quad \Rightarrow \quad \{X, DP_x, P1, P3\}$

(20)

1		2		3
	enjoy		musicals	
	$\{VP/\{NP, L_{arg}, L_{arg}, R_{arg}\}, L_{fun}, R_{fun}\}$		$\{NP, DP_{np}, L_{np}, R_{np}\}$	

(21)

1		2		3
	enjoy		musicals	
	$\{VP/\{NP, L_{arg}, L_{arg}, R_{arg}\}, 1, 2\}$		$\{NP, DP_{np}, 2, 3\}$	

(22)

1		2		3
	enjoy		musicals	
	$\{VP/\{NP, L_{arg}, L_{arg}, R_{arg}\}, 1, 2\}$		$\{NP, DP_{np}, 2, 3\}$	
	$\{X/\{Y, P2, P2, P3\}, P1, P2\}$		$\{Y, P2, P2, P3\}$	

(23)

1		2		3
	enjoy		musicals	
	$\{VP/\{NP, 2, 2, 3\}, 1, 2\}$		$\{NP, 2, 2, 3\}$	
	$\{VP, 1, 3\}$			

(24) a. $\{\{X, DP_x, L_x, R_x\}/\{Y, P2, P2, R_y\}, P1, P2\} \quad \{\{Y, P2, P2, R_y\}/\{Z, DP_z, L_z, R_z\}, P2, P3\}$
$\Rightarrow \quad \{\{X, DP_x, L_x, R_x\}/\{Z, DP_z, L_z, R_z\}, P1, P3\}$

b. $\{\{Y, P2, L_y, P2\}/\{Z, DP_z, L_z, R_z\}, P1, P2\} \quad \{\{X, DP_x, L_x, R_x\}/\{Y, P2, L_y, P2\}, P2, P3\}$
$\Rightarrow \quad \{\{X, DP_x, L_x, R_x\}/\{Z, DP_z, L_z, R_z\}, P1, P3\}$

(25) *The Possible Composition Rules*:

a.	X/Y	Y/Z	$\Rightarrow_{\mathbf{B}}$	X/Z	(> B)
b.	X/Y	$Y\backslash Z$	$\Rightarrow_{\mathbf{B}}$	$X\backslash Z$	(> Bx)
c.	$Y\backslash Z$	$X\backslash Y$	$\Rightarrow_{\mathbf{B}}$	$X\backslash Z$	(< B)
d.	Y/Z	$X\backslash Y$	$\Rightarrow_{\mathbf{B}}$	X/Z	(< Bx)

the appendix it has to be explicitly imposed. These schemata permit only the four instances of the rules of composition proposed in [5] [6], given in (25) in the basic CCG notation. "Crossed" rules like (15) are still allowed (because of the non-identity noted in the discussion of (18) between the distinguished position of arguments of functions and the position of the function itself). They are distinguished from the corresponding non-crossing rules by further specifying DP_z, the distinguished position on Z. However, no rule violating the Principle of Inheritance, like (14), is allowed: such a rule would require a *different* distinguished position on the two Zs, and would therefore not be functional composition at all. This is a desirable result: the example (16) and the earlier papers show that the non-order-preserving instances (b, d) are required for the grammar of English and Dutch. In configurational languages like English they must of course be carefully restricted as to the categories that may unify with Y.

The implications of the present formalism for the type-raising rules are less obvious. Type raising rules are unary, and probably lexical, so the principle of adjacency does not apply. However, we noted earlier that we only want the *order-preserving* instances (11), in which *the directionality of the raised category is the reverse of that of its argument.* But how can this reversal be anything but an arbitrary property?

Because the directionality constraints are grounded out in string positions, the distinguished position of the subject argument of a predicate *walks* – that is, the *right-hand edge* of that subject – is equivalent to the distinguished position of the predicate that constitutes the argument of an order-preserving raised subject *Gilbert* that is, the *left*-hand edge of that predicate. It follows that both of the order-preserving rules are instances of the single rule (26) in the extended notation: The crucial property of this rule, which forces its instances to be order-preserving, is that *the distinguished position variable DP_{arg} on the argument of the predicate in the raised category is the same as that on the argument of the raised category itself.* (The two distinguished positions are underlined in (26)). Of course, the position is unspecified at the time of applying the rule, and is simply represented as an unbound unification variable with an arbitrary mnemonic identifier. However, when the category combines with a predicate, this variable will be bound by the directionality specified in the predicate itself. Since this condition will be transmitted to the raised category, *it will have to coincide with the juncture of the combination.* Combination of the categories in the non-grammatical order will therefore fail, just as if the original categories were combining without the mediation of type-raising.

Consider the following example. Under the above rule, the categories of the words in the sentence *Gilbert walks* are as shown in (27), before binding. Binding of string positional variables yields the categories in (28). The combinatory rule of forward application (19) applies as in example (29), binding further variables by unification. In particular, DP_g, R_{np}, DP_w, and $P2$, are all bound to the juncture position 2, as in (30). By contrast, the same categories in the opposite linear order fail to unify with any combinatory rule. In particular, the backward application rule fails, as in (31). (Combination is blocked because 2 cannot unify with 3).

On the assumption implicit in (26), the only permitted instances of type raising are the two rules given earlier as (11). The earlier results concerning word-order universals under coordination are therefore captured. Moreover, we can now think of these two rules as a single underspecified order-preserving rule directly corresponding to (26), which we might write less long-windedly as follows, augmenting the original simplest notation with a non-directional slash:

(33) *The Order-preserving Type-raising Rule*:
$$X \Rightarrow_{\mathbf{T}} \mathrm{T}|(\mathrm{T}|X) \quad (\mathrm{T})$$

The category that results from this rule can combine in *either* direction, but will always preserve order. Such a property is extremely desirable in a language like English, whose verb requires some arguments to the right, and some to the left, but whose NPs do not bear case. The general raised category can combine in both directions, but will still preserve word order. It thus eliminates what was earlier noted as a worrying extra degree of categorial ambiguity. The way is now clear to incorporate type raising directly into the lexicon, substituting categories of the form $\mathrm{T}|(\mathrm{T}|X)$, where X is a category like NP or PP, directly into the lexicon in place of the basic categories, or (more readably, but less efficiently), to keep the basic categories and the rule (33), and exclude the base categories from all combination.

The related proposal of Zeevat et al. [8],[9] also has the property of allowing a single lexical raised category for the English NP. However, because of the way in which the directional constraints are here grounded in relative string position, rather than being primitive to the system, the present proposal avoids certain difficulties in the earlier treatment. Zeevat's type-raised categories are actually order-*changing*, and require the lexical category for the English predicate to be S/NP instead of $S\backslash NP$. (Cf. [9, pp.

(26) $\{X, DP_{arg}, L_{arg}, R_{arg}\} \Rightarrow \{\mathrm{T}/\{\mathrm{T}/\{X, \underline{DP_{arg}}, L_{arg}, R_{arg}\}, \underline{DP_{arg}}, L_{pred}, R_{pred}\}, L_{arg}, R_{arg}\}$

(27) 1 Gilbert $\{S/\{S/\{NP, DP_g, L_g, R_g\}, DP_g, L_{pred}, R_{pred}\}, L_g, R_g\}$ 2 walks $\{S/\{NP, R_{np}, L_{np}, R_{np}\}, DP_w, L_w, R_w\}$ 3

(28) 1 Gilbert $\{S/\{S/\{NP, DP_g, 1, 2\}, DP_g, L_{pred}, R_{pred}\}1, 2\}$ 2 walks $\{S/\{NP, R_{np}, L_{np}, R_{np}\}, DP_w, 2, 3\}$ 3

(29) 1 Gilbert $\{S/\{S/\{NP, DP_g, 1, 2\}, DP_g, L_{pred}, R_{pred}\}, 1, 2\}$ $\{X/\{Y, P2, P2, P3\}, P1, P2\}$ 2 walks $\{S/\{NP, R_{np}, L_{np}, R_{np}\}, DP_w, 2, 3\}$ $\{Y, P2, P2, P3\}$ 3

(30) 1 Gilbert $\{S/\{S/\{NP, 2, 1, 2\}, 2, 2, 3\}, 1, 2\}$ 2 walks $\{S/\{NP, 2, 1, 2\}, 2, 2, 3\}$ 3

$\{S, 1, 3\}$

(31) 1 *Walks $\{S/\{NP, R_{np}, L_{np}, R_{np}\}, 1, 2\}$ $\{Y, P2, P1, P2\}$ 2 Gilbert $\{S/\{S/\{NP, DP_g, 2, 3\}, DP_g, L_{pred}, R_{pred}\}, 2, 3\}$ $\{X/\{Y, P2, P1, P2\}, P2, P3\}$ 3

(32) $*\{X, DP_{arg}, L_{arg}, R_{arg}\} \Rightarrow \{\mathrm{T}/\{\mathrm{T}/\{X, \underline{DP_{arg}}, L_{arg}, R_{arg}\}, \underline{DP_{pred}}, L_{pred}, R_{pred}\}, L_{arg}, R_{arg}\}$

207-210]). They are thereby prevented from capturing a number of generalisations of CCGs, and in fact exclude functional composition entirely.

It is important to be clear that, while the order preserving constraint is very simply imposed, it is nevertheless an additional stipulation, imposed by the form of the type raising rule (26). We could have used a unique variable, DP_{pred} say, in the crucial position in (26), unrelated to the positional condition DP_{arg} on the argument of the predicate itself, to define the distinguished position of the predicate argument of the raised category, as in example (32). However, this tactic would yield a completely unconstrained type raising rule, whose result category could not merely be substituted throughout the lexicon for ground categories like NP without grammatical collapse. (Such categories immediately induce totally free word-order, for example permitting (31) on the English lexicon). It seems likely that type raising is universally confined to the order-preserving kind, and that the sources of so-called free word order lie elsewhere. Such a constraint can therefore be understood in terms of the present proposal simply as a requirement for the lexicon itself to be consistent. It should also be observed that a uniformly order-*changing* category of the kind proposed by Zeevat et al. is not possible under this theory.

The above argument translates directly into unification-based frameworks such as PATR or Prolog. A small Prolog program, shown in an appendix, can be used to exemplify and check the argument.[4] The program makes no claim to practicality or efficiency as a CCG parser, a question on which the reader is refered to [7]. Purely for explanatory simplicity, it uses type raising as a syntactic rule, rather than as an offline lexical rule. While a few English lexical categories and an English sentence are given by way of illustration, the very general combinatory rules that are included will of course require further constraints if they are not to overgenerate with larger fragments. (For example, `>B` and `>Bx` must be distinguished as outlined above, and the latter must be greatly constrained for English.) One very general constraint, excluding all combinations with or into NP, is included in the program, in order to force type-raising and exemplify the way in which further constrained rule-instances may be specified.

CONCLUSION

We can now safely revert to the original CCG notation described in the preliminaries to the paper, modified only by the introduction of the general order-preserving type raising rule (26), having established the following results. First, the earlier claims concerning word-order universals follow from first principles in a unification-based CCG in which directionality is an attribute of arguments, grounded out in string position. The Principles of Consistency and Inheritance follow as theorems, rather than stipulations. A single general-purpose order-preserving type-raised category can be assigned to arguments, simplifying the grammar and the parser.

[4] The program is based on a simple shift-reduce parser/recogniser, using "difference list"-encoding of string position (cf. [4], [3]).

REFERENCES

[1] Curry, Haskell and Robert Feys: 1958, *Combinatory Logic*, North Holland, Amsterdam.

[2] Dowty, David: 1988, Type raising, functional composition, and non-constituent coordination, in Richard T. Oehrle, E. Bach and D. Wheeler, (eds), *Categorial Grammars and Natural Language Structures*, Reidel, Dordrecht, 153–198.

[3] Gerdeman, Dale and Hinrichs, Erhard: 1990. Functor-driven Natural Language Generation with Categorial Unification Grammars. *Proceedings of COLING 90, Helsinki*, 145-150.

[4] Pereira, Fernando, and Stuart Shieber: 1987, *Prolog and Natural Language Analysis*, CSLI/Univ. of Chicago Press.

[5] Steedman, Mark: 1987. Combinatory grammars and parasitic gaps. *Natural Language & Linguistic Theory*, 5, 403-439.

[6] Steedman, Mark: 1990, Gapping as Constitutent Coordination, *Linguistics and Philosophy*, 13, 207-263.

[7] Vijay-Shankar, K and David Weir: 1990, 'Polynomial Time Parsing of Combinatory Categorial Grammars', *Proceedings of the 28th Annual Conference of the ACL*, Pittsburgh, June 1990.

[8] Zeevat, Henk, Ewan Klein, and Jo Calder: 1987, 'An Introduction to Unification Categorial Grammar', in N. Haddock et al. (eds.), *Edinburgh Working Papers in Cognitive Science, 1: Categorial Grammar, Unification Grammar, and Parsing*.

[9] Zeevat, Henk: 1988, 'Combining Categorial Grammar and Unification', in U. Reyle and C. Rohrer (eds.), *Natural Language Parsing and Linguistic Theories*, Dordrecht, Reidel, 202-229.

APPENDIX

```
%% A Lexical Fragment: parse will bind position (via list-encoding):

category(gilbert, cat(np, _, P1, P2)).
category(brigitte, cat(np, _, P1, P2)).
category(walks,cat(cat(s,_,_,_)/cat(np,P2,_,P2),_,P3,P4)).
category(love, cat(cat(vp,_,_,_)/cat(np,P3,P3,_),_,P1,P2)).
category(must,cat(cat(cat(s,_,_,_)/cat(np,P2,_,P2),_,_,_)/cat(vp,P5,P5,_),_,P3,P4)).
category(madly, cat(cat(vp,_,_,_)/cat(vp,P2,_,P2),_,P3,P4)).

%% Application and (overgeneral) Composition: Partial evaluation of DPy with the actual Juncture P2
%% imposes Adjacency.   DPy (=P2) must not be == Y's other end (see <B and >B).  Antecedent \+ Y=np
%% disallows ALL combination with unraised NPs.
reduce(cat(cat(X,DPx,P1,P3)/cat(Y,P2,P2,P3),_,P1,P2),
       cat(Y, P2, P2,P3),
       cat(X,DPx,P1,P3)) :- \+ Y=np.                                          %>

reduce(cat(Y,P2,P1,P2),
       cat(cat(X,DPx,P1,P3)/cat(Y,P2,P1,P2),_,P2,P3),
       cat(X,DPx,P1,P3)) :- \+ Y=np.                                          %<

reduce(cat(cat(X,DPx,X1,X2)/cat(Y,P2,P2,Y2),_,P1,P2),
       cat(cat(Y,P2,P2,Y2)/cat(Z,DPz,Z1,Z2),_,P2,P3),
       cat(cat(X,DPx,X1,X2)/cat(Z,DPz,Z1,Z2),_,P1,P3)) :- \+ Y=np,\+ Y2==P2.  %>B, cf. ex. 24a

reduce(cat(cat(Y,P2,Y1,P2)/cat(Z,DPz,Z1,Z2),_,P1,P2),
       cat(cat(X,DPx,X1,X2)/cat(Y,P2,Y1,P2),_,P2,P3),
       cat(cat(X,DPx,X1,X2)/cat(Z,DPz,Z1,Z2),_,P1,P3)) :- \+ Y=np,\+ Y1==P2.  %<B, cf. ex. 24b

%% Order Preserving Type Raising: the rule np -> T|(T|np).
raise(cat(np,DPnp,P1,P2),                                                     % Binds P1, P2
      cat(cat(T,DPt,T1,T2)/cat(cat(T,DPt,T1,T2)/cat(np,DPnp,P1,P2),DPnp,_,_), % cf. ex. 26
          _,P1,P2)).

%% Parse simulates reduce-first shift-reduce recogniser with backtracking (inefficiently)
parse([Result], [], Result).                                  % Halt
parse([Cat1|Stack], Buffer, Result) :-                        % Raise (syntactic)
     raise(Cat1, Cat2),
     parse([Cat2|Stack], Buffer, Result).
parse([Cat2, Cat1|Stack], Buffer, Result) :-                  % Reduce
     reduce(Cat1, Cat2, Cat3),
     parse([Cat3|Stack], Buffer, Result).
parse(Stack, [Word|Buffer], Result) :-                        % Shift
     category(Word, cat(W,DPw,[Word|Buffer],Buffer)),         % Position is list-encoded
     parse([cat(W,DPw,[Word|Buffer],Buffer)|Stack], Buffer, Result).

%% Example crucially involving bidirectional T (twice) and <Bx:
% | ?- parse([],[gilbert,must,love,madly,brigitte],R).
%
% R = cat(s,_37,[gilbert,must,love,madly,brigitte],[]) % ; -- plus 4 more equivalent derivations
%
% yes
```