Frozen Scope in Type Logical Grammar[*]

HIROYUKI UCHIDA

Abstract

I discuss frozen scope between object quantificational NPs (QNPs) in two kinds of ditransitive-verb construction. In order to derive a correct structural relation between the two objects, I adopt Morrill's ditransitive verb category type with certain modifications. For scope alternation, I reformulate Hendriks' argument(-slot) raising in a Lambek Calculus system. The proposed analysis requires an overt functor between the two QNPs, and only the prepositional construction has a pronoun *to* as such a functor, which explains why only this construction has scope ambiguity. Formulation of the proposal in a deductive grammar system requires modifying the division of labour between the grammar system and the lexicon. The proposal predicts (excessively) strong locality constraints on scope switch.

1 Introduction

Bruening (2001:235) reports that the double object (DO) construction in (1a) has only the surface scope reading, *a>every*, while the prepositional (PP) construction in (1b) has both the surface and the inverse scope readings.[1]

(1) (a) The teacher showed a (#different) student every book.
**every>a*, *a>every*
(b) The teacher showed a (different) book to every student.
a>every, *every>a*

If we merge the verb *show* successively with its two objects in (1a), as the bracketing in (2a) suggests, then the universal QNP c-commands the indefinite and we should get the inverse scope reading. If we adopt an equivalent of May's QR and assume that the two quantifiers as higher order functors are applied to the

*I am grateful to Annabel Cormack, Neil Smith, Sally McConnell-Ginet, Michael Moortgat, Nicholas Allott, Alison Hall, Robert Truswell and Marc Richards for comments and informative discussions. Needless to say, they are not responsible for any of the mistakes in the paper.

[1] *a>every* means that the indefinite takes wide scope over the universal quantifier.

verbal expression in two different orders, as (2b) and (2c) suggest, then we should get both the surface and the inverse scope reading with (1a). Neither is the case in Bruening's judgment.

(2) (a) The teacher [[showed a student] every book]
(b) [Some_student$_{((et)t)}$ [λx_e.[Every_book$_{((et)t)}$ [λy_e.[the teacher showed x y]]]]]
(c) [Every_book$_{((et)t)}$ [λy_e.[Some_student$_{((et)t)}$ [λx_e.[the teacher showed x y]]]]]

In order to avoid generating the c-command relation in (2a), I adopt the lexical entry for the ditransitive verb *show* in (3a), which Morrill uses (cf. Morrill 1994:128).[2]

(3) (a) show: (NP1\S)/(NP2•NP3);
show″ = $\lambda\alpha.\lambda x.$[show′(π1(α))(π2(α))(x)] {type:((e×e),(e,t)}
(b) show: ((NP1\S)/NP3)/NP2;
$\lambda x.\lambda y.\lambda z.$[show′($x$)($y$)($z$)] {type: (e,(e,(e,t)))}

The ditransitive verb category in (3a) requires the two internal NP arguments to be merged first as (NP2•NP3) (the corresponding semantic type is (e×e)), before the result is merged with the verb. Informally, the connective • merges two categories/logical expressions into an ordered pair, which is one complex category/logical expression.[3] The functor category (NP1\S)/(NP2•NP3) has two arguments, the complex object argument, (NP2•NP3), and the subject argument, NP1. Accordingly, the logical expression **show″** in (3a) also has two arguments, where its first argument is of type (e×e) for the two objects combined, such as **(tom′•mary′)** for *Tom Mary* in the sentence *Bob showed Tom Mary*. But in the equivalent lambda expression on the right-hand side of the equation in (3a), we can use the standard three place functor **show′** of type (e(e(et))), by means of the **π** operators. Given **(tom′•mary′)** of type (e×e), **π1** selects the left member and **π2** selects the right member. Then the selected **tom′** and **mary′** are put into the first and the second argument slots of **show′**. This means that, though the number of the syntactic arguments of the verb *show* is two, we can preserve the three arguments of the semantic functor **show′** in the normalized logical form.

Unlike Morrill, however, I do not use the curried ditransitive verb category/logical expression in (3b), which requires the two objects to be merged

[2] For convenience, I have added numbering on NPs to show the linear order among NPs in the PF string. The category NP is mapped to type e in the semantics (see section 2).

[3] See section 2 for details.

successively with the verb. (3b) would lead to the wrong c-command relation in (2a). It would also over-generate scope readings in my analysis, as we see in section 3.

To switch scope between two QNPs, I reformulate Hendriks' argument-slot raising as categorial argument-slot raising in a Lambek Calculus system, "**NL**."[4] We can switch scope only if there is a functor of a certain category between the two object QNPs.

Because we can switch scope only between co-argument QNPs of an intervening functor in a non-associative grammar system NL, certain locality constraints fall out. In order to switch scope between QNPs that are farther apart, we have to be able to treat the phonological string between the two QNPs as a complex predicate.

Section 2 informally explains the grammar system that I use. Section 3 explains the proposed scope-switch system in its initial form and applies it to the ditransitive constructions. Section 4 reconsiders my proposal in a deductive grammar system. Instantiation of my proposal in such a system requires adjusting the division of labour between the grammar and the lexicon. I also generalize the initial argument slot raising system, which leads to slightly different empirical predictions while still preserving the merits of the initial system. Section 5 briefly discusses the locality of scope taking and a controlled introduction of association to the grammar system. Section 6 provides a concluding summary.

2. Type Logical Grammar (Informal)

Type Logical Grammar (TLG) assigns category types to lexical items, as in (4). The notation is <*phonological item*; syntactic category; logical expression>.

(4) (a) Tom: <*Tom*; NP; tom′>
(b) love: <*love*; (NP\S)/NP; λ*x*.λ*y*.love′(*x*)(*y*)>
(c) Meg: <*Meg*; NP; meg′>

I assume that a syntactic proof is based on the syntactic categories only. The PF strings and semantic representations are derivative, as we will see. Syntactic categories and semantic types are defined in usual ways, as in (5) and (6). *F* is the set of categories and *TYP* is the set of semantic types. For a more complete exposition, I refer the reader to Chapter 1 of Bernardi (2002).

[4] Cormack (1985) also argues for switching the scope of QNPs by means of the related functor.

(5) (a) The set of basic categories, AT = {NP, S, N..}, is a subset of F.
(b) If A and B are members of F, so are (A\B), (B/A) and (A•B).
(c) Nothing is a member of F except on the basis of (a) and (b).

(6) (a) The set of basic types, B = {e, t…} is a subset of *TYP*.
(b) If a and b are members of *TYP*, so are (a, b) and (a×b).
(c) Nothing is a member of *TYP* except on the basis of (a) and (b).

Given the sets F and *TYP*, we can define a mapping function, **type**: $F \times TYP$, as in (7).

(7) (a) type(NP) = e type(S) = t
(b) type(A\B) = type(B/A) = (type(A), type(B))
(c) type(A•B) = (type(A)×type(B))

In section 4, I provide a slightly more formal definition of syntactic categories, phonological expressions, and typed lambda expressions as semantic representations. Taking the logical nature of TLG more seriously, we interpret syntactic categories as sets of phonological/semantic objects in that section.

Given the categories provided by the lexical items, the grammar then checks if it can derive a single category (for example, S for sentential expressions) out of the lexical categories. The derivations are based on the syntactic rules of the connectives, such as / and \, plus some general syntactic rule(s).

Reflecting the symmetry of the logical system, TLG has both an introduction rule and an elimination rule for each categorial connective, such as / and \, in the so-called natural deduction presentation. However, in this section, I show only the rules that I use in section 3.[5]

(8) (a) / Elimination (/E):
Syntax: ((NP\S)/NP, NP) $\Rightarrow$ NP\S
Semantics: $((\lambda x.\lambda y.\text{love}'(x)(y), \text{m}') \rightarrow (\lambda x.\lambda y.\text{love}'(x)(y))(\text{m}') \rightarrow_{\beta \text{ red.}}$
$\lambda y.\text{love}'(\text{m}')(\text{y})$
PF form: $(love, Meg) \rightarrow (love \cdot Meg)$
(b) \ Elimination (\E):
Syntax: (NP, NP\S) $\Rightarrow$ S
Semantics: $(\text{m}', \lambda x.\text{smoke}'(x)) \rightarrow (\lambda x.\text{smoke}'(x))(\text{m}') \rightarrow_{\beta \text{ red.}} \text{smoke}'(\text{m}')$

---

[5] Logical forms are given the simplified form, **Functor(Argument$_1$)…(Argument$_n$),** as in **like′(*x*)(*y*)**, abbreviating **((like′ *x*) *y*)** etc., which informally means ***y*** likes ***x***. The connective '·' over PF items represents temporal precedence.

PF form: (*Meg*, *smoke*) → (*Meg*·*smoke(s)*).

Informally, the functor category Y/X requires the argument category X immediately to the right, producing the output category Y as a result, and X\Y requires X immediately to the left, producing the output Y. In the semantics, they are both interpreted as function application. The base grammar system that I adopt is the non-associative, non-commutative system, NL, which configures categories into binary structures, as indicated by round brackets.[6] Because of the non-associativity, bracketing as in (Y\Z)/X is taken seriously. That is, the order of the two arguments must be X first and Y second, with Z as the final output.

Lambek calculus has another binary connective •, which explicitly represents successive merges of category types into binary structures in NL. I only show the introduction rule for •, with the corresponding semantics and PF concatenation.

(9) • Introduction. (•I)
(a) Syntax: (NP1, NP2) ⇒ (NP1•NP2)
(b) Semantics: (tom′, meg′) → (tom′•meg′)
(c) PF form: (*tom*, *meg*) → (*tom*·*meg*)

The semantics and the phonology are provided informally here. For the semantics, in (9b), the connective • used in the typed lambda expression is neither associative nor commutative, just as the categorial connective • is non-associative and non-commutative. This is a necessary assumption to prevent a mix-up of the two arguments **tom′** of type e and **meg′** of type e in the logical term **(tom′•meg′)** of type (e×e), in their argument order with regard to a verbal functor, for example. The connective · in phonological terms is non-commutative and non-associative, though I mostly abstract away from phonological structures in this paper. In order to decompose the complex logical expression, **(α•β)**, into its subparts, we use the pair of operators **π1** and **π2**, which select the first member **α** and the second member **β** of the complex expression respectively. We see its applications in the next section.

[6] I often omit the round brackets when the structure is made out of two sister categories only, such as, **NP, NP\S**. The LF and the PF forms are bracketed accordingly, to preserve the soundness and completeness of the syntax to the intended interpretations, though I omit the brackets for these expressions as well, unless bracketing has some importance to the issue discussed there.

3 Proposal

3.1 Hendriks' argument(-slot) raising

In order to switch scope by means of a functor, we need to be able to modify the type of the functor. My scope switch system is based on Hendriks' proposal (Hendriks 1987). Hendriks' argument(-slot) raising is defined over semantic types, as in (10).

(10) (a) Type Shift: (a*, (b, (c*, t))) ⇒ (a*, (((b,t),t), (c*, t)))
(b) Semantics: P ⇒ λ*x**.λ*Q*.λ*y**.*Q*(λ*z*.P(*x**)(*z*)(*y**))
Types of variables. P: (a*, (b, (c*, t))), *Q*: ((b,t),t), *x**: a*, *y**: c*, *z*: b,

N.B. a*, b, c* are meta-variables for some types, where a* and c* are such that (a*,(b..)) = (b..), (e,(b..)), (e,(e,(b..))), etc. and (c*,t) = t, (e,t), (e,(e,t)), etc.
(cf. Hendriks 1987: 109)

If a functor has an argument slot of type b, then we can raise it to type ((b,t),t), so that the functor now can be applied to type ((b,t),t) argument, rather than to type b argument. The meta-variables *x** of type a* and *y** of type b* mean that the functor *P* can have any number of arguments (including zero arguments) before and after the argument *z* of type b, which is type-raised by the rule.

Hendriks' system allows scope alternation among any co-arguments of a functor. If we use the curried ditransitive-verb entry in (3b) with his system, it cannot block the unattested inverse scope reading with the DO construction in (1a). Given the functor **show′** of type (e,(e,(e,t))), we can apply argument raising to its two object argument slots in two orders, deriving the two scope readings.

(11) (a) Type Shift: (e, (e, (e, t))) ⇒ (((e,t),t), (e,(e,t))) ⇒ (((e,t),t),(((e,t),t),(e,t)))
(b) Semantics: show′ ⇒ λ*Q1*.λ*y*.λ*z*.*Q1*(λ*x*.show′(*x*)(*y*)(*z*)) ⇒
λ*Q1*.λ*Q2*.λ*z*.*Q2*(λ*y*.*Q1*(λ*x*.show′(*x*)(*y*)(*z*)))

(12) (a) Type Shift: (e, (e, (e, t))) ⇒ (e, (((e,t),t),(e,t))) ⇒ (((e,t),t),(((e,t),t),(e,t)))
(b) Semantics: show′ ⇒ λ*Q2*.λ*y*.λ*z*.*Q2*(λ*y*.show′(*x*)(*y*)(*z*)) ⇒
λ*Q1*.λ*Q2*.λ*z*.*Q1*(λ*x*.*Q2*(λ*y*.show′(*x*)(*y*)(*z*))),
where *Q1*, *Q2*: (et)t, and *x*, *y*, *z*: e.

Applying the outputs of (11) and (12) to two QNP objects of type ((et)t) and to a normal type e subject such as *Tom* in (13a) derives the two scope readings in (13b) and (13c).

(13) (a) Tom showed some student every book.
(b) $\text{Some\_student}'_{(et)t}(\lambda x.\text{Every\_book}'_{(et)t}(\lambda y.\text{give}'(x)(y)(\text{tom}')))$
Some>Every
(c) $\text{Every\_book}'(\lambda y.\text{Some\_student}'(\lambda x.\text{give}'(x)(y)(\text{tom}')))$
Every>Some

Because of this, I use only the category (NP\S)/(NP•NP) in (3a) for ditransitive verbs, together with a modified argument-slot raising that I show in the next subsection.

Argument-slot raising should naturally lead to certain locality constraints on scope-taking; scope of a QNP should stay within the domain of the functor which takes the QNP as an argument. However, because Hendriks uses a fully-associative grammar system with an abstraction rule, a QNP can take scope beyond the domain of its local functor, by treating an indefinitely long phonological string as a complex functor.

(14) (a) Some student [said that Tom read] every book.
(b) $\text{Every\_book}'(\lambda y.\text{Some\_student}'_{(et)t}(\lambda x.\text{say}'_{(t(et))}(\text{read}'_{(e(et))}(y)(\text{tom}'))(x)))$

Given (14a), association (re-bracketing) and λ abstraction allow us to treat the string *said that Tom read* as a complex type (e(et)) functor. We can then apply argument-slot raising to its type e argument slot of this complex functor.[7] This derives the unacceptable scope reading *every >some*, as in (14b).

In order to solve these problems, I modify Hendriks' argument-slot raising in a non-associative and non-commutative grammar system NL.

3.2 Syntactic category type raising in NL

In this sub-section, I reformulate Hendriks' argument-slot raising so that it is constrained by syntactic category. Initially, I limit an application of argument-slot raising to a functor that has maximally one NP argument at each side, as in (15a).

(15) (a) (NP1\T)/NP2, where T is normally S.
(b) (NP1\T)/NP2 ⇒ (NP1\T)/((S/NP2)\S) ⇒ ((S/(NP1\S))\T)/((S/NP2)\S)
(c) (NP1\T)/NP2 ⇒ ((S/(NP1\S))\T)/NP2 ⇒ ((S/(NP1\S))\T)/((S/NP2)\S)

NP1 or NP2 can be missing in (15). For example, (15) can be applied to NP\S for intransitive verbs (though it does not switch scope, for lack of another QNP). The semantic rules of (15b) and (15c) are (16) and (17), when T is instantiated as S.

---

[7] The explanation is sketchy for space reasons. See Hendriks (1987:112-115) for details.

(16) (a) Type Shift: (e,(e,t))) ⇒ (((e,t),t),(e,t)) ⇒ (((e,t),t),(((e,t),t),t))
(b) Semantics: P ⇒ λ*Q1*.λ*y*.*Q1*(λ*x*.P(*x*)(*y*)) ⇒
λ*Q1*.λ*Q2*.*Q2*(λ*y*.*Q1*(λ*x*.(P(*x*)(*y*))))

Variable types *Q1, Q2*: (et)t;*x, y*: e.

(17) (a) Type Shift: (e, (e, t)) ⇒ (e, (((e,t),t), t)) ⇒ (((e,t),t),(((e,t),t),t))
(b) Semantics: P ⇒ λ*Q2*.λ*y*.*Q2*(λ*y*.P(*x*)(*y*)) ⇒
λ*Q1*.λ*Q2*.*Q1*(λ*x*.*Q2*(λ*y*.P(*x*)(*y*)))

T is normally S. For example, when the functor is a transitive verb, such as *love,* as in *Every boy loves some girl*, which has the lexical category (NP\S)/NP and the logical entry, **λ*x*.λ*y*.love′(*x*)(*y*)**, applying (15)~(17) derives the two higher order functors as in (18a) and (18b), which lead to the surface and the inverse scope readings, respectively.

(18) (a) ((S/(NP1\S))\T)/((S/NP2)\S); λ*Q1*.λ*Q2*.*Q2*(λ*y*.*Q1*(λ*x*.love′(*x*)(*y*)))
(b) ((S/(NP1\S))\T)/((S/NP2)\S); λ*Q1*.λ*Q2*.*Q1*(λ*x*.*Q2*(λ*y*.love′(*x*)(*y*)))

However, (15) also covers functors that have the specified number of NP arguments as above but do not have the output category S, such as prepositions. In such cases, as well as the argument-slot raising, we also apply value-raising to the functor, with regard to the T category, as I show in section 3.4.

Next, I adopt the ditransitive verb entry in (19a) that Morrill uses.

(19) (a) show: (NP1\S)/(NP2•NP3); show′′ = λα.λ*x*.[show′(π1(α))(π2(α))(*x*)]
(b) show: ((NP1\S)/NP3)/NP2; λ*x*.λ*y*.λ*z*.[show′(*x*)(*y*)(*z*)]
(cf. Morrill, 1994: 128)

(16a) requires the two NP objects to be merged first, before the resultant category NP2•NP3 is merged with this verbal functor. In the logical form, the operator π1 selects the first member of the complex NP argument α of type e×e and π2 selects the second member. The two parts of the complex object NP argument are placed back into the appropriate type e argument slots of the ditransitive verb expression, **show′**, preserving the three argument slots in the normalized logical form.

Unlike Morrill, however, I do not adopt the curried entry (19b). This entry not only leads to an undesirable scope reading in (13c), but it also derives a wrong c-command relation between the two object positions in (20a). Given the standard

binding asymmetry as in (20b) and (20c), the right object should not c-command the left object.[8]

(20) (a) [Tom [[showed Nancy] herself]]
(b) Tom showed Nancy_1 herself_1 (in the mirror).
(c) *Tom showed herself_1 Nancy_1 (in the mirror).

In sub-section 3.1, we have seen that an associative grammar with an abstraction rule can nullify the locality constraints that should naturally follow from argument-slot raising. Because the λ abstraction rule or its categorical correspondents /, \ introduction rules are essential in TLG, I preserve them and adopt the non-associative system NL as the base grammar.

3.3 Application 1: DO construction

First, I apply my analysis to the DO construction in (21a).

(21) (a) Meg showed a (# different) student every book. *$every>a$, $a>every$
(b) show: <*show*; (NP1\S)/(NP2•NP3); λα.λ*x*.[show′(π1(α))(π2(α))(*x*)]>
(c) a student: <*a·student*; (S/NP)\S; λ*A*.some′(st′)(λ*u*.*A*(*u*))>
(d) every book: <*every·book*; (S/NP)\S; λ*B*.every′(bk′)(λ*v*.*B*(*v*))>
Types: Every′, Some′: (et)((et)t), α: e×e, *x*, *u*, *v*: e, *A*, *B*: (et).

In the DO construction, there is no functor between the two object QNPs. Thus we have to merge the two QNPs directly, by using a slightly modified •I rule in (22).[9]

(22) (a) Syntax (• Introduction(Q)):

$$\dfrac{\dfrac{\underline{a\cdot student} \qquad \underline{every\cdot book}}{\underline{(S/NP2)\backslash S} \qquad (S/NP3)\backslash S}\ \bullet_{I(Q)}}{(S/(NP2\bullet NP3))\backslash S}$$

[8] For more binding data, see Barss and Lasnik (1986) and Larson (1988). Though I emphasized the c-command relation for convenience, my binding explanation is not directly dependent on this relation. See Uchida (2005) for my analysis of binding data.

[9] I should derive •I(Q) from the basic •I in (9), though I have failed to do so. See section 4.2.1.

(b) Semantics:

a·student	*every·book*	
λ*A*.some′(st′)(λ*u*.*A*(*u*))	λ*B*.every′(bk′)(λ*v*.*B*(*v*))	•I(Q)
λ*R*.some′(St′)(λ*u*.every′(bk′)(λ*v*.*R*(*u*•*v*)))		

The natural deduction presentation as in (22) is roughly like the standard syntactic tree notation upside down, with the lexical expressions aligned at the top in the PF left to right order. The output logical expression in the bottom line in (22b) has type (((e×e),t),t), where its argument R has type ((e×e),t). The logical expression shows that the scope relation is the surface scope, *some>every*, between the two object QNPs.

Note that •I just juxtaposes the two items to be merged, preserving the current structural relation between the two items and the internal structure in each item. Arguably, •I(Q) does a little more than juxtaposing the two items, but considering •I(Q) is a variant of •I, a natural requirement is that •I(Q) preserves the scope relation represented in the surface linear order between the two QNPs that •I(Q) merges. As I have just shown, this is the case.

When we merge the result with the ditransitive verb *show*, we apply argument-slot raising to *show*.

(23) (a) (NP1\S)/(NP2•NP3) ⇒ (NP1\S)/((S/(NP2•NP3))\S)
(b) λα.λ*z*.[show′(π1(α))(π2(α))(*z*)] ⇒
λQ2.λ*z*.Q2(λα.show′(π1(α))(π2(α))(*z*))
Types: Q^2: (((e×e),t),t); α: (e×e); *z*: e.

The output of (20) is a functor whose first argument Q^2 is the complex QNP type in (19) which is made out of the two object QNPs. The rest of the derivation is shown in (21).[10]

(24) (a) Syntax:

Meg	*show*	*a·student·every·book* D
NP1	(NP1\S)/((S/(NP2•NP3))\S)	(S/(NP2•NP3))\S /E
	(NP1\S) \E	
S		

[10] D in the derivation means that I have omitted the derivation from the lexical level to this level.

(b) Semantics:

Meg	*show*	*a·student·every·book* D
meg′	λ*Q*2.λ*z*.*Q*2(λα.show′(π1(α))(π2(α))(*z*))	λ*R*.some′(st′)(λ*u*.every′(bk′)(λ*v*.*R*(*u*•*v*))) /E

λz.[λ*R*.[some′(st′)(λ*u*.every′(Bk′)(λ*v*.*R*(*u*•*v*)))](λα.show′(π1(α))(π2(α))(*z*))] β red

λz.[some′(st′)(λ*u*.every′(bk′)(λ*v*.[λα.[show′(π1(α))(π2(α))(*z*)](*u*•*v*)]))] β red

λz.[some′(St′)(λ*u*.every′(bk′)(λ*v*.show′(π1(*u*•*v*))(π2(*u*•*v*))(*z*)))] π dist

λz.[some′(st′)(λ*u*.every′(bk′)(λ*v*.show′(*u*)(*v*)(*z*)))]

some′(st′)(λ*u*.every′(bk′)(λ*v*.show′(*u*)(*v*)(meg′)))

π1 and **π2** selects the left and the right members of ***u*•*v***, which are put into the appropriate argument slots of **show′**. The only scope reading is: *a student>every book*.

3.4 Application 2: PP construction

Next, I apply my analysis to the PP construction in (25a). (25b) is the lexical assignment to the preposition *to*. The other items have the same entries as in (21).

(25) (a) Meg showed a (different book) to every student. *a>every*, *every>a*
(b) to: <*to*; (NP2\(NP3•NP2))/NP3; λ*x*.λ*y*.(*x*•*y*) {of type (e,(e,(e×e)))}>

Note that (25b) reverses the linear order between the two object NPs in the output. This enables us to use the uniform entry for the verb *show* in both the constructions.

As for scope alternation, (25b) does not have the normal output category S, as we discussed for (15). Its output category is NP3•NP2. Because of this, we value raise the output category (NP3•NP2) to (S/(NP3•NP2))\S.[11] Then, we can apply

[11] See Hendriks (1987: 109) for value raising. Value raising is necessary in (26) and (27) to get the semantics right. When we merge two QNP expressions of type (et)t, the output should not be an expression of type e×e; it should be a complex QNP expression of type (((e×e),t),t), which corresponds to the value-raised output category, (S/(NP3•NP2))\S. Value raising is also necessary to make the scope switch process compatible with the syntactic calculus at the basic level. That is, if the output category stayed as NP3•NP2 after the process as in (26) and (27), then the merge of the argument-slot raised functor with two QNPs would produce the category, NP3•NP2. Omitting the preposition category for convenience, this process corresponds to the sequent, (S/NP)\S, ((*to*), (S/NP)\S) ⊦ (NP•NP). This sequent has the same effect as type lowering (though it is by way of binary merges, rather than a unary operation). Type lowering, such as (S/(NP•NP))\S ⊦ NP•NP, is not provable in NL (i.e. it is not supported by NL), and we do not want an operation that goes against this basic property of the grammar system.

argument-slot raising to this functor in two different orders, deriving two scope readings.
Order 1. Argument & value raising to the preposition *to*, the surface scope reading.

(26) (a) Syntax: (NP2\(NP3•NP2))/NP3 ⇒ (NP2\((S/(NP3•NP2))\S))/((S/NP3)\S)
⇒ (((S/NP2)\S)\((S/(NP3•NP2))\S))/((S/NP3)\S)
(b) Semantics: λ*x*.λ*y*.*x*•*y* ⇒ λQ1.λ*y*.λ*R*.*Q1*(λ*x*.*R*(*x*•*y*))
⇒ λ*Q1*.λ*Q2*.λ*R*.*Q2*.(λ*y*.*Q1*(λ*x*.*R*(*x*•*y*)))
Types: *Q*1, *Q*2: (et)t; *R*: ((e×e),t); *x, y*: e.

Order 2. Argument & value raising to the preposition *to*, the inverse scope reading.

(27) (a) Syntax: (NP2\(NP3•NP2))/NP3 ⇒ (((S/NP2)\S)\((S/(NP3•NP2))\S))/NP3
⇒ (((S/NP2)\S)\((S/(NP3•NP2))\S))/((S/NP3)\S)
(b) Semantics: λ*x*.λ*y*.*x*•*y* ⇒ λ*x*.λ*Q2*.λ*R*.*Q2*(λ*y*.*R*(*x*•*y*))
⇒ λ*Q1*.λ*Q2*.λ*R*.*Q1*.(λ*x*.*Q2*(λ*y*.*R*(*x*•*y*)))

Note that in (26b), the argument slot for the left object QNP (= Q2) takes wide scope over the argument slot for the right object QNP (= Q1), representing the surface scope reading. The scope relation is the opposite in (27b), representing the inverse scope reading. For lack of space, I only show the inverse scope derivation by applying the output of (27) to its two QNP arguments.

(28) (a) Syntax:

a·book	*to*	*every·student*	
(S/NP2)\S	(((S/NP2)\S)\((S/(NP3•NP2))\S))/((S/NP3)\S)	(S/NP3)\S	/E
	((S/NP2)\S)\((S/(NP3•NP2))\S)		\E
	((S/(NP3•NP2))\S)		

(b) Semantics:

a·book	*to*	*every·student*	
λ*B*.some′(bk′)(λ*u*.*B*(u))	λ*Q*1.λ*Q*2.λ*R*.*Q*1.(λ*x*.*Q*2(λ*y*.*R*(*x*•*y*)))	λ*A*.every′(st′)(λ*v*.*A*(*v*))	/E
	λ*Q*2.λ*R*.every′(st′)(λ*v*.*Q*2(λ*y*.*R*(*v*•*y*)))		\E
	λ*R*.every′(st′)(λ*v*.some′(bk′)(λ*u*.*R*(*v*•*u*))		

The logical expression in the bottom line shows the scope relation: *every>a*, which is the inverse scope reading. If we use (26), instead of (27), we can derive the surface scope reading. I only show the result of its merge with its two QNP arguments in (29).

(29) (a) Syntax: (S/(NP3•NP2))\S
(b) Semantics: λ*R*.some′(bk′)(λ*u*.every′(st′)(λ*v*.R(*v*•*u*))

I omit the derivation steps up to the sentential level, which is similar to (24).

4 Deductive views of Categorial Grammar.

In this section, I re-evaluate my proposal while taking the deductive nature of Type Logical Grammar more seriously. It leads to the division of labour between the logical grammar system and the lexical rules that are not fully supported by the syntactic proof. I also generalize the initial proposal, which leads to slightly different empirical predictions.

4.1 Type Logical Grammar
4.1.1 Basics

Type Logical Grammar (TLG) has been investigated by Morrill (1994), Moortgat (1997) and others. TLG has developed from Lambek calculus (Lambek, 1958), which reformulated the Classical Categorial Grammar (as in Ajdukiewicz, 1935 and Bar-Hillel, 1953) as a deductive proof system.[12] Lambek's introduction of the multiplicative connective • led to the well-known derivability relation in (30). It sets up the connectives /, \, and • as a residuated triple, which, informally, allows us to move categorial formula components between the antecedent side and the succedent side of sequents, where each sequent has the form,
Sequent: **Antecedent ⊢ Succedent**.[13]

(30) $B \vdash A\backslash C \Leftrightarrow A \bullet B \vdash C \Leftrightarrow A \vdash C/B$

To show categorial derivation, I have used natural deduction presentation (ND) so far. ND is useful for showing step-wise composition of interface representations (such as typed lambda expressions at LF) from lexical levels. But in order to show the derivability/provability of non-standard combinatory rules (such as function composition and type lifting) from the basic axioms of the grammar, the so-called Gentzen sequent presentation is more useful, which I use in this section.

---

[12] I do not provide a detailed history of Categorial Grammar. See Versmissen (1996) and Bernardi (2002) for such development.

[13] Bernardi shows that the pair of / and \ alone (i.e. without •) already exhibits a certain kind of residuation, supported by the provability of the sequent A ⊢ (B/A)\B. See Bernardi (2004: 130).

Before I show the axioms in that presentation, I provide the standard definitions of the set of (categorial) formulas and the set of structures.

The set *F* of (categorial) formulas is defined as in (31).

(31) $F ::= At \mid F/F \mid F\backslash F \mid F \bullet F$

At is the set of atomic formulas, such as NP, N and S. Given *At*, we close the set *F* by successively applying binary connectives /, \ and • to any two formulas, either atomic or derived, creating complex formulas such as NP\S, S/(NP\S) and (NP\S)/(NP•NP). Not all the members of *F* have to be included in the lexicon of natural languages. *At* might presumably be universal across languages, but each language typically has a different set of complex-categories in its lexicon. For the presentation of the axioms and proofs, I use formula meta-variables A, B, C…$\in F$.

Given *F, S* represents the set of structures, which configure formulas into certain structures.

(32) $S ::= F \mid (S, S)$

Example structures are (NP, NP\S) and (NP, ((NP\S)/NP, NP)). In NL, which is non-associative and non-commutative, the bracketing and the left-to-right order are taken seriously. That is, (A, (B, C))⇎((A, B), C), and (A, B)⇎(B, A). I use the standard structural meta-variables Γ, Γ′, Δ, Δ′… ∈ *S*.

4.1.2 Gentzen sequent presentation.

The axioms are made out of the identity axiom and the so-called logical rules, which are pairs of Left and Right rules for /, \ and •. The cut axiom is convenient for presentation reasons, but has been proved to be admissible.[14] Thus, we can omit the cut axiom out of the system without influencing the provability of any sequent.

(33) Identity Axiom: $A \vdash A$ Cut $\dfrac{\Gamma \vdash A \quad \Delta[A] \vdash B}{\Delta[\Gamma] \vdash B}\,Cut$

(34) Logical rules:

(a) /L $\dfrac{\Delta[B] \vdash C \quad \Gamma \vdash A}{\Delta[(B/A, \Gamma)] \vdash C}\,/L$ /R $\dfrac{(\Gamma, A) \vdash B}{\Gamma \vdash B/A}\,/R$

[14] That is, whenever we can prove a sequent by using Cut, we can also prove it without Cut. See Versmissen (1996: 9-11) for some cut elimination proofs.

$$\text{(b) } \backslash\text{L}\ \frac{\Gamma \vdash A \quad \Delta[B] \vdash C}{\Delta[(\Gamma, A\backslash B)] \vdash C}\backslash\text{L} \qquad \backslash\text{R}\ \frac{(A, \Gamma) \vdash B}{\Gamma \vdash A\backslash B}\backslash\text{R}$$

$$\text{(c) } \bullet\text{L}\ \frac{\Gamma[(A, B)] \vdash C}{\Gamma[(A \bullet B)] \vdash C}\bullet\text{L} \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{(\Gamma, \Delta) \vdash (A \bullet B)}\bullet\text{R}$$

The notation $\Gamma[\Delta]$ means that the constituent structure Δ occupies a distinguished position in the structure Γ containing it.

Given the axioms in (33) and (34), the grammar aims to prove goal sequents, which represent syntactic derivability relations. (35b) is an example of a goal sequent that is provable in NL.

(35) (a) Antecedent ⊢ Succedent

(b) (NP, ((NP\S)/NP, NP)) ⊢ S

The antecedent is a member of the set *S* in (32). In Lambek Calculus, the antecedent structure cannot be empty. The succedent is exactly one formula, such as S, S/NP, but not a complex structure, such as (NP, NP\S).

As an example, I provide the proof of the sequent (35b) for the sentence *Tom loves Meg*, with the lexical entries as in (4) in section 2.

(36)
$$\cfrac{\cfrac{NP \vdash NP \quad S \vdash S}{NP, \quad NP\backslash S \vdash S}\backslash\text{L} \qquad NP \vdash NP}{\mathbf{(NP, ((NP\backslash S)/NP, NP)) \vdash S}}/\text{L}$$

We start each proof with atomic identity axioms, as the top line of the proof in (36) shows.[15] From bottom to top, each step of the proof eliminates one occurrence of a connective. This is because the logical rules in (34) are set up in that way. Because of this **sub-formula** property, Gentzen sequent proofs are decidable.[16]

The next question is how we read off the PF-LF pairing from the Gentzen sequent proof. I only provide some informal rules here. I provide more formal interpretation rules in 4.1.3 and 4.1.4.

For each proof, we look at the sequent at the bottom line, that is, the sequent to be proved. In (36), the goal sequent is (NP, ((NP\S)/NP, NP)) ⊢ S. Replacing each

---

[15] For convenience, I start some proofs with non-atomic identity axioms (e.g. NP\S ⊢ NP\S). But the complex identity axioms that I use can be derived from atomic identity axioms.

[16] Though Cut can introduce an indefinite number of intermediate steps to proofs, it is admissible, and therefore does not influence the decidability. See Versmissen (1996: 9-11).

categorial formula with the corresponding PF and LF expressions (the lexical assignments as in (4) show which categories correspond to which PF-LF items), we can derive the PF and the LF sequents.

(37) (a) PF: (*Tom*, (*love*, *Meg*)) ⊢ (*Tom*·(*love*·*Meg*))
(b) LF: (tom′, (love′, meg′)) ⊢ love′(meg′)(tom′)

(37a) means that out of the three PF items, *Tom, like, Meg,* we can derive the sentential expression (*Tom*·(*likes*·*Meg*)). I abstract away from the relevance of the binary structure in the phonological string, but ignoring the brackets, we get the string *Tom like(s) Meg*. The LF sequent in (37b) represents the same kind of derivability relation at the semantic level.

Though this informal PF-LF paring works in linguistic analyses, it is not very satisfactory in some ways. For one, seeing the natural language syntax as a deductive proof system, we want to set up the interpretation rules in such a way that we can prove the soundness and completeness of the syntactic system with regard to its intended interpretation (at the two interfaces). Though I do not have any space to discuss the soundness and completeness proof, it is important to set up the interpretation rules as in the semantics of logical languages. Also, we want to know how the syntactic proofs based on categorical formulas derive the two interface sequents as in (37a) and (37b) step by step. Because of these, in the next two sub-sub-sections, I show the interpretation rules in a slightly more formal way.

4.1.3 Model Theoretic Interpretation at PF

In the Model Theoretic interpretation at PF, formulas (i.e. syntactic categories) are interpreted as sets of phonological expressions.[17] For atomic formulas, the valuation function v_0 maps members of *At* to the corresponding sets of phonological expressions.

(38) (a) v_0(NP) = {*Tom*, *Meg, the*·*book*…}
(b) v_0(S) = {*Tom*·*smoke*, *Tom*·*love*·*Meg*, *Tom*·*give*·*the*·*book*·*to*·*Meg*...}

The atomic interpretation function v_0 is then extended to the general interpretation function v as in (39).

[17] We could interpret categorial formulas as sets of ordered pairs of phonological and semantic objects. However, for convenience, I deal with the interpretations at the two interfaces separately.

(39) (a) $v(A \bullet B) = \{a \cdot b \mid a \in v(A) \ \& \ b \in v(B)\}$
(b) $v(A \backslash C) = \{ b \mid \forall a \in v(A).(a \cdot b \in v(C))\}$
(c) $v(C / A) = \{ b \mid \forall a \in v(A).(b \cdot a \in v(C))\}$

For example, by (39b), $v(NP\backslash S)=\{b \mid \forall a \in v(NP).(a \cdot b \in v(S))\}$. If the PF unit *smoke* is a member of $v(NP\backslash S)$, then for each $\boldsymbol{a} \in v(NP)$, the PF expression $\boldsymbol{a \cdot smoke}$ must be a member of $v(S)$. The binary connective $\cdot$ over phonological expressions is non-associative and non-commutative. That is, we interpret categorial formulas as sub-sets of a groupoid.[18]

The (syntactic) derivability relation expressed by $\vdash$ is interpreted as set-inclusion (or sub-set) relations, as shown in (40).

(40) (a) Syntax: Antecedent $\vdash$ Succedent.
(b) Interpretation: ||Antecedent|| $\subseteq$ ||Succedent||.

4.1.4 Model Theoretic Interpretation of NL at LF

As in the phonological interpretation, we interpret categorial formulas as sets of semantic objects, represented by typed lambda expressions.[19]

(41) (a) $v_0(NP)=\{tom', meg', the\_book'...\}$
(b) $v_0(S)=\{smoke'(tom'), love'(tom')(meg'), give'(meg')(the\_book'))(tom')\ldots\}$

The atomic function v_0 is extended to the general interpretation function v in (42).

(42) (a) $v(A \bullet B) = \{a \bullet b \mid a \in v(A) \ \& \ b \in v(B)\}$
(b) $v(A \backslash C) = v(C / A) = \{b \mid \forall a \in v(A).(b(a) \in v(C))\}$

---

[18] NL is complete for general groupoid models, but not for binary tree structures with tree adjunction as addition. See Morrill (1994: 66-67) for this point, which refers to Došen (1992) and Venema (1994). See these references, and Versmissen (1996) for some soundness and completeness proofs.

[19] I abstract away from the issue of whether the lambda language represents an independent level of semantic representation in the mind, or is just used as meta-representation of model theoretic objects in the world. In either case, I assume that the interpretation of typed lambda expressions as model theoretic objects is straightforward. See Gamut (1991: 83-91), for example.

By (42b), if **smoke′** is a member of v(NP\S), then for each $\boldsymbol{a} \in v$(NP), **smoke′**($\boldsymbol{a}$) must be a member of v(S).

(42b) interprets directional syntactic functors as sets of non-directional logical functors. However, the multiplicative connective ● over lambda terms is non-commutative and non-associative, which corresponds to the non-commutativity and non-associativity of the syntactic system. We do β reduction in two directions, as in (43a) and (43b). The argument should appear in the direction that the syntactic functor category requires it. That is, to the left in (NP●(NP\S)) for (43a) and to the right in ((NP\S)/NP●NP) for (43b).

(43) (a) $(\text{tom}' \bullet \lambda x.\text{smoke}'(x)) \Rightarrow (\lambda x.\text{smoke}'(x))(\text{tom}') \Rightarrow \text{smoke}'(\text{tom}')$
(b) $(\lambda x.\lambda y.\text{love}'(x)(y) \bullet \text{meg}') \Rightarrow (\lambda x.\lambda y.\text{love}'(x)(y))(\text{meg}') \Rightarrow \lambda y.\text{love}'(\text{meg}')(y)$

As I mentioned above, we want to show how the typed lambda expressions as meaning representations are derived in the syntactic proofs. This is easier to do in Natural Deduction (ND) proofs that I used in section 3. ND proofs look like linguists' syntactic trees up-side down. Thus, we can decorate the leaf nodes with lexical expressions, such as **tom′**, **λ*x*.λ*y*.like′(*x*)** and **meg**,′ and then successively merge them to derive the result expression, such as **like′(meg′)(tom′)**, at the root node. In contrast, the Gentzen sequent presentation is not convenient for showing the step-wise derivation of the concrete lambda expressions in this way. To show how we derive the LF interface representations in Gentzen sequent proofs, we decorate the proofs with a meta-language of typed lambda expressions.[20] The basic idea is as in (44) (cf. Morrill, 1994: 16, 92-95).

(44) $(A_1 : x_1, (A_2 : x_2, \ldots (A_{n-1} : x_{n-1}, A_n : x_n))) \vdash B : \phi$

For the formulas $\mathbf{A_1 \ldots A_n}$ in the antecedent of each sequent in the syntactic proof, we provide typed variables $x_1, x_2, \ldots, x_n$. The typed variables $x_1, x_2, \ldots, x_n$ represent any lambda expressions of the semantic types corresponding to the syntactic categories $\mathbf{A_1 \ldots A_n}$, according to the mapping function, **type**, in (7). The sequent in (44) says that with variables $x_1 \ldots x_n$ of the semantic types, **type**(A_1)…**type**(A_n), placed in the binary structural configuration, we can derive the logical expression

[20] Labelled Deductive Systems, proposed by Gabbay (1996) and discussed by Kurtonina (1994) in TLG, decorate categorial formulas with extra elements such as typed lambda terms and phonological expressions. To them, lambda terms and PF expressions are part of the syntactic structures and therefore, they need to define interpretation rules for them. I use typed meta-variables as decorations just to show how we interpret the syntactic proofs step by step and then produce concrete lambda expressions as a result. The proofs themselves are solely based on the syntactic categories. I leave the different implications of these two assumptions for another paper.

ϕ, whose semantic type is **type**(B). Now if we replace the variables $x_1 \ldots x_n$ in the antecedent with concrete lambda expressions $\alpha_1 \ldots \alpha_n$ of the right types, then in the succedent of the sequent, we derive the lambda expression ϕ', which is ϕ except that $\alpha_1 \ldots \alpha_n$ replaces $x_1 \ldots x_n$ inside ϕ**.** (45) provides an example of the use of this meta-lambda-language in the goal sequent.

(45) (a) (NP: x_1, ((NP\S)/NP: x_2, NP: x_3)) ⊢ B: $x_2(x_3)(x_1)$
(b) (NP: tom′, ((NP\S)/NP: love′, NP: meg′)) ⊢ B: love′(meg′)(tom′)

In (45a), we label the syntactic categories with typed meta-variables. Replacing the variables with concrete lambda expressions, as in (45b), we can show the specific semantic sequent that we associate with the phonological sequent in (37a) via the syntactic proof. Note that, in (45a), even though I used concrete categorical formulas such as NP and (NP\S)/NP, rather than formula meta-variables such as A and (A\B)/A (for presentation reasons), we still use the meta-variables x_1, x_2, x_3 for the lambda expressions.

Now we decorate the axioms in Gentzen sequent presentation with meta-variables for typed lambda expressions.

(46) Axiom: A: u ⊢ A: u Cut $\dfrac{\Gamma \vdash A: \delta \qquad \Delta[A: u] \vdash B}{\Delta[\Gamma] \vdash B[u: \rightarrow \delta]}$

(47) Logical rules:

(a) /L $\dfrac{\Delta[B: X] \vdash C: \gamma \qquad \Gamma \vdash A: \beta}{\Delta[(B/A: \alpha, \Gamma)] \vdash C: \gamma[X: \rightarrow \alpha(\beta)]}$ /L /R $\dfrac{(\Gamma, A: x) \vdash B: \phi}{\Gamma \vdash B/A: \lambda x.\phi}$ /R

(b) \L $\dfrac{\Gamma \vdash A: \beta \qquad \Delta[B: X] \vdash C: \gamma}{\Delta[(\Gamma, A\backslash B: \alpha)] \vdash C: \gamma[X: \rightarrow \alpha(\beta)]}$ \L \R $\dfrac{(A: x, \Gamma) \vdash B: \phi}{\Gamma \vdash A\backslash B: \lambda x.\phi}$ \R

(c) •L $\dfrac{\Gamma[(A: u, B: v)] \vdash C: \gamma}{\Gamma[(A \bullet B: u \bullet v)] \vdash C: \gamma[(u,v): \rightarrow u \bullet v]}$ •L •R $\dfrac{\Gamma \vdash A: u \quad \Delta \vdash B: v}{(\Gamma, \Delta) \vdash (A \bullet B): u \bullet v}$ •R

$\alpha(\beta)$ is free for X in γ.
x is a fresh variable.

Categorial formulas such as NP and NP\S are paired with lambda variables of the corresponding types. Γ and Δ represent binary structures made out of such pairs. For example, Γ might be (NP: x, (NP\S): P). Again, the variables are assigned appropriate types according to the mapping function, **type**, defined in (7). For example, **type**(NP) = e for x and **type**(NP\S) = (et) for P in the example in the previous sentence. In (47), $\gamma[X: \rightarrow \alpha(\beta)]$ means that we replace the free variable X inside the expression γ with the expression $\alpha(\beta)$. In order to prevent an accidental

binding in /L and \L, α(β) has to be free for X in γ. That is, no free variable in α(β) ends up being bound by an operator in γ.

In /R and \R, by saying that x is fresh, we mean that there is no free occurrence of x inside Γ. In each application of /R or \R, exactly one free variable x is bound.

As an example, I decorate the proof in (36) with typed lambda expressions.

(48) $\dfrac{\dfrac{\dfrac{}{\text{NP: } x \vdash \text{NP: } x} \quad \dfrac{}{\text{S: } X \vdash \text{S: } X}\ \text{L}\backslash}{\text{NP: } x,\ \text{NP}\backslash\text{S: } P \vdash \text{S: } X[X{:=}P(x)]\ (= P(x))} \quad \dfrac{}{\text{NP: } y \vdash \text{NP: } y}\ /\text{L}}{\text{NP: } x,\ ((\text{NP}\backslash\text{S})/\text{NP: } R,\ \text{NP: } y) \vdash \text{S: } P(x)[P{:=}R(y)]\ (= R(y)(x))}$

$\dfrac{}{\textbf{NP: } \boldsymbol{x}\textbf{, ((NP}\backslash\textbf{S)/NP: } \boldsymbol{R}\textbf{, NP: } \boldsymbol{y}\textbf{) } \vdash \textbf{ S: } \boldsymbol{R(y)(x)}}$ [instantiated as]

NP: tom′, ((NP\S)/NP: like′, NP: meg′) ⊢ S: like′(meg′)(tom′)

Types. R: (e(et)); P: (et); x, y: e

The sequent in bold face is the one that is proved in (48). Instantiating x as **tom′**, R as **like′** (= **λ*u*.λ*v*.like′(*u*)(*v*)**) and y as **meg′**, we have the sequent in the bottom line. Note that the meta-variables are all replaced by constant expressions. This means that use of (meta) free variables decorating Gentzen sequent proofs does not necessarily lead to the problems that Jacobson (1999) mentions with regard to the use of free variables in the concrete logical expressions that represent the meanings of natural language expressions. We can easily assign the variable free requirements at the level of the instantiated lambda expressions as in the bottom line in (48).

In the next sub-sub-section, I discuss the provability of some of the higher order rules that I used in my proposal.

4.2.1 Argument-slot raising and •R(Q)

In this sub-sub-section, I check the provability of the combinatory rules related to the QNP categories.

In NL, argument-slot raising is provable only for the subject argument-slot, not for an object argument-slot, as shown in (49).[21]

[21] Argument slot lowering is provable for all the argument positions. For example, NL can prove the sequent, (NP\S)/((S/NP)\S) ⊢ (NP\S)/NP, for the object slot of transitive verbs. However, in order to derive different categories (representing different scope readings) from the uniform lexical category, the lexically assigned category should be the lower category, e.g. (NP\S)/NP for transitive verbs. Thus, the scope switch mechanism that we use should be argument slot raising, rather than lowering.

(49) Subject-slot raising: OK

$$\dfrac{\dfrac{S \vdash S \quad NP\backslash S \vdash NP\backslash S}{S/(NP\backslash S),\ NP\backslash S \vdash S}\ {\scriptstyle /R}}{NP\backslash S \vdash (S/(NP\backslash S))\backslash S}\ {\scriptstyle \backslash L}$$

Object-slot raising: Not OK

$$\dfrac{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots}{(NP\backslash S)/NP,\ (S/NP)\backslash S \nvdash NP\backslash S}\ {\scriptstyle /R}\qquad (NP\backslash S)/NP \vdash (NP\backslash S)/((S/NP)\backslash S)\ {\scriptstyle /L?}$$

Though NL can prove argument-slot raising only for the subject slot, it can merge the raised functor category with its QNP arguments both for subject QNPs and object QNPs. To show this, I first provide the proofs of the identity axioms of two Generalized Quantifier (GQ) categories, for subject QNPs and object QNPs.

(50)

$$\dfrac{\dfrac{\dfrac{NP \vdash NP \quad S \vdash S}{NP,\ NP\backslash S \vdash S}\ {\scriptstyle \backslash R}}{S/(NP\backslash S),\ NP\backslash S \vdash S}\ {\scriptstyle /R}}{S/(NP\backslash S) \vdash S/(NP\backslash S)}\ {\scriptstyle \backslash L}\qquad \dfrac{\dfrac{\dfrac{S \vdash S \quad NP \vdash NP}{S/NP,\ NP \vdash S}\ {\scriptstyle /R}}{S/NP \vdash S/NP \quad S \vdash S}\ {\scriptstyle \backslash L}}{S/NP,\ (S/NP)\backslash S \vdash S}\ {\scriptstyle \backslash R}}{(S/NP)\backslash S \vdash (S/NP)\backslash S}\ {\scriptstyle /L}$$

Given these identity axioms, we can prove (51).

(51)

$$\dfrac{\dfrac{S/(NP\backslash S) \vdash S/(NP\backslash S) \quad S \vdash S}{S/(NP\backslash S),\ (S/(NP\backslash S))\backslash S \vdash S}\ {\scriptstyle \backslash L} \qquad (S/NP)\backslash S \vdash (S/NP)\backslash S}{S/(NP\backslash S),\quad (((S/(NP\backslash S))\backslash S)/((S/NP)\backslash S),\quad (S/NP)\backslash S) \vdash S}\ {\scriptstyle /L}\ {\scriptstyle \text{PF mapping}}$$

Every boy *loves* *some girl* ⊢ S

(51) proves that we can use the argument-slot raised transitive verb category with two GQ categories as its arguments without an association rule. In fact, we do not even have to use different GQ categories for the subject and the object QNPs. I could have replaced the identity axiom for the object QNP, (S/NP)\S ⊢ (S/NP)\S, with the one for the subject QNP, S/(NP\S) ⊢ S/(NP\S).

I show how the syntactic proof in (51) derives the semantic sequent. I omit the categorial formulas and only show the semantic meta-variables in (52), for readability.

(52)

$$\dfrac{\dfrac{Q1 \vdash Q1 \quad X \vdash X}{Q1,\ P \vdash \mathrm{S}\colon X[X{:=}\,P(Q1)] = P(Q1) \qquad Q2 \vdash Q2}\ {\scriptstyle /L}}{Q1,\quad (R,\quad Q2) \vdash P(Q1)[P{:=}\,R(Q2)] = R(Q2)(Q1)}\ {\scriptstyle \backslash L}$$

Meta-variable types: $Q1$, $Q2$: (et)t, P: ((et)t)t, R: ((et)t)(((et)t)t)

Variables *Q1* and *Q2* are for GQ type expressions (of type (et)t). *R* is for the argument-slot raised logical expressions for *love*. If we apply argument-slot raising to *love* (with its lexical entries, (NP\S)/NP; $\lambda x.\lambda y.\text{love}'(x)(y)$, in the syntax and the semantics), then *R* can be instantiated as two argument-slot raised functors, representing the two orders of raising its internal and external argument slots. The two functor expressions in bold face in (53b) and (53c) are the two versions, representing the two scope readings, the subject wide scope and the object wide scope readings respectively.

(53) (a) *Q1,* (*R,* *Q2*)
(b) $\lambda A.\text{every}'(\text{boy}')(A)$, (**$\lambda Q2.\lambda Q1.Q1(\lambda y.Q2(\lambda x.\text{love}'(x)(y)))$**, $\lambda B.\text{some}'(\text{girl}')(B)$)
(c) $\lambda A.\text{every}'(\text{boy}')(A)$, (**$\lambda Q2.\lambda Q1.Q2(\lambda x.Q1(\lambda y.\text{love}'(x)(y)))$**, $\lambda B.\text{some}'(\text{girl}')(B)$)

Of course, as (49) showed, NL cannot derive the argument-slot raised functors in (53b) and (53c) from the lexical entries for the verb *love*. Because of that, (53) at the moment can be derived only by using a purely 'lexical rule' which operates on the verbal functor.[22] Such a rule (e.g. as formulated in (15)~(17)) is not fully supported by the syntactic system. On the other hand, once we are equipped with the lexical argument-slot raising in (53), then the syntactic proof in (51) and the derivation of the meta-lambda sequent in (52) based on that proof do not require any association rule to merge the raised functor with its QNP arguments. The order of the merges of the three expressions is the same both for (53b) and (53c), though their scope readings are different. We can switch scope without an association rule. Considering that a fully associative grammar system over-generates with regard to the attested natural language data, this is one merit of the proposed scope switch system.

I have achieved the more restrictive syntax at the cost of using a lexical rule that is not fully supported by the logical grammar. This might be the last resort strategy for TLG.[23] Also, the claim of restrictiveness should be evaluated with the total

---

[22] It is slightly misleading to call this a 'lexical rule', because it needs to be able to take as input certain non-lexical expressions, such as a complex predicate, as we see in section 5.

[23] Moortgat's scope constructors as in Moortgat (1991) and the following efforts to derive the constructors from the basic rules of the grammar can be interpreted as an instantiation of Hendriks' idea in TLG without making such a compromise. On the other hand, his analysis over-generates without additional locality constraints that do not have any inherent relations to the scope switch mechanism itself. Also, treating QNP scope in the same way as (in-situ) Wh expressions, the analyses based on his scope constructor do not explain the different behaviours of quantifier scope on the one hand, and Wh movement on the other, in terms of the observed locality constraints. See Bernardi (2002) for a treatment of QNP scope using Moortgat's scope constructor in a Multi-Modal TLG setting.

grammar system in mind, which might require some controlled introduction of structural association. I will come back to this point in section 5.

Next, I discuss the (non)-derivability of •I(Q) in (22) from the basic •I in (9). In section 3, I used •I(Q) to merge the two object QNPs.

First, when the two objects are normal NPs, it is easy to prove the derivation of VP (which is used as shorthand for NP\S), using the proposed ditransitive category formula, shown in Gentzen sequent presentation in (54).

(54)
$$\dfrac{\dfrac{\mathrm{VP} \vdash \mathrm{VP} \qquad (\mathrm{NP}, \mathrm{NP}) \vdash (\mathrm{NP}\bullet\mathrm{NP})}{\mathrm{VP}/(\mathrm{NP}\bullet\mathrm{NP}),\ (\mathrm{NP}, \mathrm{NP}) \vdash \mathrm{VP}}\,/\mathrm{L} \qquad \dfrac{\mathrm{NP} \vdash \mathrm{NP} \qquad \mathrm{NP} \vdash \mathrm{NP}}{(\mathrm{NP}, \mathrm{NP}) \vdash (\mathrm{NP}\bullet\mathrm{NP})}\,\bullet\mathrm{L}}$$

•I in Natural Deduction presentation is •R in (34c) in Gentzen sequent presentation. Then, •I(Q) in (22) will be •R(Q), which merges (S/NP1)\S and (S/NP2)\S in this left to right order, producing (S/(NP1•NP2))\S as the result. •R(Q), however, is not easily provable from •R.

We can derive the categorial formula (S/(NP•NP))\S from two NPs.

(55)
$$\dfrac{\dfrac{\dfrac{\mathrm{S} \vdash \mathrm{S} \qquad (\mathrm{NP}, \mathrm{NP}) \vdash \mathrm{NP}\bullet\mathrm{NP}}{\mathrm{S}/(\mathrm{NP}\bullet\mathrm{NP}),\ (\mathrm{NP}, \mathrm{NP}) \vdash \mathrm{S}}\,\backslash\mathrm{R}}{(\mathrm{NP}, \mathrm{NP}) \vdash (\mathrm{S}/(\mathrm{NP}\bullet\mathrm{NP}))\backslash\mathrm{S}} \qquad \dfrac{\dfrac{\dfrac{X \vdash X \qquad (u, v) \vdash (u\bullet v)}{R, (u, v) \vdash X[X{:=}R(u\bullet v)] = R(u\bullet v)}\,/\mathrm{R}}{(u, v) \vdash \lambda R.R(u\bullet v)} \qquad \dfrac{u \vdash u \qquad v \vdash v}{\bullet\mathrm{R}}}{\bullet\mathrm{R}}\,/\mathrm{L}}$$

However, this does not work. We want to merge two QNP categories, not two NPs, to derive (S/NP•NP)\S. In (55), the derived (meta)-lambda expression, $\lambda R.R(u\bullet v)$, is not the one that we want. The sequent that we want to prove is the one in (56), for the expression (*Tom showed*) *a student every book*.

(56) (a) Syntax: (S/NP)\S, (S/NP)\S ⊢ (S/(NP•NP))\S

(b) LF: $\lambda A.\mathrm{some}'(\mathrm{st}')(A), \lambda B.\mathrm{every}'(\mathrm{bk}')(B) \vdash \lambda R.\mathrm{some}'(\mathrm{st}')(\lambda u.\mathrm{every}'(\lambda v.R(u\bullet v)))$

(c) PF: *some·student, every·book* ⊢ (*some·student*)· (*every·book*)

(56) is not provable in NL. Part of the reason is that the type lowering is illegal.

(57) $\dfrac{\dfrac{\text{(S/NP1)\S} \vdash \text{NP1} \quad \text{(S/NP2)\S} \vdash \text{NP2}}{\text{S} \vdash \text{S} \quad \text{((S/NP1)\S), (S/NP2)\S)} \vdash \text{NP1}\bullet\text{NP2}}\ /\text{L}}{\text{S/(NP1}\bullet\text{NP2), ((S/NP1)\S, (S/NP2)\S)} \vdash \text{S}}\ \backslash\text{R}$ ($!1^{st}$ line, not provable)

(S/NP1)\S, (S/NP2)\S ⊢ (S/(NP1•NP2))\S

(57) is not provable in NL because the sequent (S/NP)\S ⊢ NP is not provable.

Though I do not go into details, introducing a structural association rule to the grammar (in a modally controlled way) does not easily solve the problem either.

Note that the un-derivability of •R(Q) from •R is far more problematic than the un-provability of argument-slot raising (for object slots). This case is about the merge of two syntactic categories, which is certainly a matter of syntax. It also has to do with the compositional derivation of the semantic meaning of the two object QNPs combined from the lexically assigned meanings of the two QNPs. However, without an easy solution at hand, I leave the non-provability of •R(Q) in the current grammar system for further research. •R(Q) stays as an axiom for the moment.

4.2.2 Generalizing the argument-slot category raising.

As I pointed out in the last sub-sub-section, the reformulated argument-slot raising in (15), repeated here as (58), is a lexical process. Because of that, there is nothing wrong about defining the rules over specific input categories.

(58) (a) (NP1\T)/NP2, where T is normally S.
(b) (NP1\T)/NP2 ⇒ (NP1\T)/((S/NP2)\S) ⇒ ((S/(NP1\S))\T)/((S/NP2)\S)
(c) (NP1\T)/NP2 ⇒ ((S/(NP1\S))\T)/NP2 ⇒ ((S/(NP1\S))\T)/((S/NP2)\S)

However, generalizing the input categories might still have some merits, including better empirical predictions. For example, while maintaining the assumption that all the functor categories have maximally one syntactic argument selection at each side, we can generalize these argument-slot categories into A and B, rather than limiting them to NP arguments, as shown in (59a). We can also generalize the goal category as C.

(59) (a) (B\C)/A
(b) (B\C)/A ⇒ (B\C)/((C/A)\C) ⇒ ((C/(B\C)\C)/((C/A)\C)
(c) (B\C)/A ⇒ ((C/(B\C))\C)/A ⇒ ((C/(B\C))\C)/((C/A)\C)

After generalizing (58) as (59), the semantic computation will also be generalized as in (60) and (61).

(60) (a) Type Shift: (a,(b, c))) ⇒ (((a,c),c),(b,c))) ⇒ (((a,c),c),(((b,c),c),c))
(b) Semantics: P ⇒ λ*Q1*.λ*β*.*Q1*(λ*α*.P(*α*)(*β*)) ⇒
λ*Q1*.λ*Q2*.*Q2*(λ*β*.*Q1*(λ*α*.P(*α*)(*β*)))

(61) (a) Type Shift: (a, (b, c)) ⇒ (a,(((b,c),c),c)) ⇒ (((a,c),c),(((b,c),c),c))
(b) Semantics: P ⇒ λα.λ*Q2*.*Q2*(λ*β*.P(*α*)(*β*)) ⇒
λ*Q1*.λ*Q2*.*Q1*(λ*α*.*Q2*(λ*β*.P(*α*)(*β*)),
where *Q1*: ((a,c),c), *Q2*: ((b,c),c), and *α*: a, *β*: b.

The question is whether the generalized argument-slot raising over-generates scope readings. It does lead to more scope readings than the less general rule in (15). As an example, when all the three arguments of the ditransitive verb are QNPs in the DO construction, as in (62), the revised analysis predicts that the two object QNPs together can take wide scope over the subject QNP.

(62) (a) A (different) teacher showed me every book. *a > every, every > a*
(b) A (?*different) teacher showed a (certain) student every book.
*a teacher > a student > every book, ?*a student > every book > a teacher…*

In (B\C)/A in (59), A and B can be any argument category. If we instantiate A as NP•NP, B as NP and C as S, then we get the category that I use for ditransitive verbs in English, that is, (NP\S)/(NP•NP). Now, we can apply the argument slot raising to the two arguments, that is, the subject NP and the complex object, NP•NP, in two orders. The scope relation between the two object QNPs is still fixed as the surface scope, because there is no intervening functor between the two object QNPs, but we can switch the scope of the two QNPs together and the scope of the subject QNP. (62a) has the reading in which the right object QNP *every book* can take scope over the subject indefinite *a teacher*. This judgment supports the more general formulation in (59) as opposed to (58).[24] The inverse scope reading in question, *a student > every book > a teacher*, is more difficult to get in (62b). However, some speakers do get this reading in adequate contexts, and the difficulty might be because of the processing complexity of dealing with the scope of three QNPs, not because of the impossibility of the scope switch in the formal system.

[24] The left object *me* is lexically assigned the category NP. Though it is type raised to (S/NP)\S in the syntactic derivation (i.e. for the syntax to merge *me* with the lexically higher order *every book* by •R(Q), producing the complex object category (S/(NP•NP))\S), the logical term for *me* (say, **the_speaker′)** of type e) is put back to the corresponding argument slot of the logical expression **show′** in the normalized logical form. The normalized logical form for the inverse scope reading will be: **every′(book′)(λ*x*.some′(teacher′)(λ*y*.show′(the_speaker′)(*x*)(*y*)))**.

Considering that we have to derive the inverse scope reading for (62a) somehow, the more general argument-slot raising in (59) (with the semantics as in (60) and (61)) might be preferable than the initially proposed argument-slot raising.

Also, the more general argument-slot raising in (59) still does not allow a scope switch across a tensed clause. Thus, some desirable locality constraints still fall out from the proposed analysis.

(63) (a) A boy said that Tom liked every girl.
*a boy > every girl, *every girl > a boy*
(b) say: <*say*; (NP\S)/S; $\lambda\phi.\lambda x.\text{say}'(\phi)(x)$ {of type (t(et))}>

Computationally, we can apply the argument-slot raising to the two arguments of (NP\S)/S for *say*, deriving the category, (((S/NP)\S)\S)/((S/S)\S). However, in whichever order we apply the two argument-slot raisings, we cannot alternate the scopes of the two QNPs. The categorial type that the grammar derives for the string *that Tom liked every girl* is S and its semantic type is t. Even though the grammar system raises this embedded clause category S to (S/S)\S, normalization on the logical form places the embedded type t expression back into the internal argument slot of **say′** (the position of the variable ϕ to the right in (63b)).[25] If any QNP is contained in the type t expression that fills the internal argument slot of **say′** in the normalized form, then that QNP cannot take scope outside that argument-slot. In other words, the QNP's scope cannot go beyond the embedded clause.

Because QNPs 'bind' type e argument-slots, in order for a QNP to take scope long-distance, we have to create a complex functor of the form (NP\C)/NP, (rather than (NP\S)/S, for example). We have already seen that in NL, we cannot treat the string *say that Tom likes* in (63a) as (NP\S)/NP, because of the lack of an association rule.

In this sub-section, we saw that the argument-slot raising is not fully supported by NL. Though this means that we have to have it as a separate lexical rule, we can keep the syntactic system more restrictive. The rule •R(Q) is not provable from the basic axioms, and thus, stays as a separate axiom, a problem left for further research. I have also shown that I can generalize the proposed argument-slot raising, by defining it over general categories such as A, B, C, without losing the good points of the initial proposal, such as strong locality constraints falling out from the system.

---

[25] This proof for (63a), in which we type raise the argument S to (S/S)\S, to which the argument-raised functor category (QNP\S)/(S/S)\S is applied, can be normalized to a proof in which neither the type raising nor the internal argument slot raising takes place. Such spurious ambiguity of the proof is a weak point of the Gentzen sequent presentation, not of the abstract proof system, which might be better represented by another presentation.

In the next section, I briefly discuss the locality of quantifier scope and the necessity of a controlled introduction of a structural association rule and its implication to my proposal.

5 Locality

In this section, I briefly discuss locality constraints on QNP scope in my analysis.[26]

(64) (a) A girl loves every boy. *a>every*, *every>a*
(b) [An [apple [in [every box]]]] was mouldy. *#a>every*, *every>a*
(c) A teacher stood in front of each classroom. *a>every*, *every>a*

For (64a), we simply apply argument-slot raising to the functor *love* in two orders, deriving the two scope readings, as we have briefly seen in section 3.

(64b) is harder to deal with. For a sentence as in (64b), the preposition *in* is normally assigned the following lexical entry (cf. Carpenter 1997: 233).[27]

(65) in: <*in*; (N\N)/NP; $\lambda x.\lambda A.\lambda y.(A(y) \wedge \text{in}'(x)(y))$> {semantic type: e((et)(et))}

My analysis can derive the surface scope reading easily. I apply the argument-slot raising to the single NP argument-slot in (N\N)/NP. Ignoring the details, the output of this operation is (66a) (**in**$^{\mathbf{a}}$ shows that the functor has been argument-slot raised).

(66) (a) in^{a}: <*in*; (N\N)/((S/NP)\S); $\lambda Q1.\lambda A.\lambda y.(A(y) \wedge Q1(\lambda x.\text{in}'(x)(y)))$>
Variables, *Q1*: ((et)t); *A*: (et); *x, y*: e
(b) a: <*a*; (S/(NP\S))/N; $\lambda A.\lambda B.\text{some}'(A)(B)$> {**some′** of type (et)((et)t)}
(c) apple: <*apple*; N; $\lambda x.\text{apple}'(x)$> {**apple′** of type (et)}
(d) every box: <*every·box*; (S/NP)\S; $\lambda B.\text{every}'(\text{box}')(B)$> {type (et)t}

Then, (66a) is merged with the QNP *every box* and the common noun *apple* in this order. Next, the indefinite determiner, *a*, as in (66b) takes this output as an

---

[26] (64b) is by Cormack (p.c), though I have changed the determiner *some* to *an*.

[27] The PP, *in every box*, occupies a different structural position in a sentence such as, *John* [[*found* [*an apple*]] [*in every box*]], as the bracketing suggests. (Thanks to Cormack for pointing out the structural difference). In this case, the right QNP is merged later than the indefinite NP, and thus, we can naturally derive the scope relation, *every>a*. Though I do not discuss how to derive the scope reading, *a>every*, in this paper, this sentence is easier to deal with than (64b), because the PP *every box* is not contained inside the indefinite NP.

argument. Thus, we derive the surface scope logical expression for *an apple in every box*, as in (67). The bracketing in PF represents the order of the merges.

(67) (a) category: S/(NP\S)
(b) LF: $\lambda B.\text{some}'(\lambda y.(\text{apple}'(y) \wedge \text{every}'(\text{box}')(\lambda x.\text{in}'(x)(y))))(B)$
(c) PF: (*an* ·(*apple* ·(*in* ·(*every* ·*box*))))

Merging the QNP in (67) with the VP *was mouldy* produces the logical form, **some′(λ*y*.(apple′(*y*) ∧ every′(box′)(λ*x*.in′(*x*)(*y*))))(λ*y*.mouldy′(*y*))** for (64b). In actual communication, it is difficult to interpret the sentence in (64b) with this surface scope reading, but I assume that this is only a pragmatic difficulty.

The inverse scope is harder to derive. The category (N\N)/NP for *in* does not have an NP argument-slot for the QNP, *some apple*. Note that we cannot simply assign an alternative lexical category in the form of (NP\T)/NP (where T is some output category), ignoring the subject (Q)NP structure indicated by the brackets in (64b). As in (67b), the expression **in′** has to appear inside the nominal restriction of the existential determiner **some′** (i.e. the **A** position in the form, **some′(A)(B)**). If we merged the determiner *a* and the noun *apple* before we merge the result with the PP, *in every box*, this would already fix the nominal restriction, as we can see in the expression, **λ*B*.some′(λy.apple(y))(*B*)**. Then we would not be able to incorporate the PP logical expression into the nominal restriction part any more.

However, given the standard entry for *in* as in (65), we can apply value raising, as well as argument-slot raising. Ignoring the details, this produces the argument-slot raised and value raised functor for *in*, as in (68).

(68) (a) Category: (N\(**((S/(NP\S))/N)\(S/(NP\S))**))/((S/NP)\S)
(b) LF: $\lambda Q.\lambda A.\lambda D.\lambda B.Q(\lambda x.D(\lambda y.A(y) \wedge \text{in}'(x)(y))(B))$
The expression is of type, ((et)t), ((et), (((et)((et)t)), ((et)t))), where
the variable types are, *Q*: (et)t; *A*, *B*: (et); *D*: (et)((et)t); *x, y*: e

In (68a), the NP argument-slot of *in* has been raised to (S/NP)\S, for the QNP, *every box*. I have also value raised N to ((S/(NP\S))/N)\(S/(NP\S)) (which is in bold face in (68a)). Given the other items as in (66), we derive the following output for the QNP, *an apple in every box*, which leads to the inverse scope reading.

(69) (a) Category: S/(NP\S)
(b) LF: $\lambda B.\text{every}'(\text{box}')(\lambda x.\text{some}'(\lambda y.\text{apple}'(y) \wedge \text{in}'(x)(y))(B))$
(c) PF: (*an* ·(*apple* ·(*in* ·(*every* ·*box*))))

The order of the merges stays the same for (67) and (69), but in (67), the determiner, *a*, is merged as a functor, while in (69), it is merged as an argument, as a result of the value raising to *in*. This functor-argument alternation is supported by NL, which does not distinguish the structural positions of functors and arguments. Thus, other than the object-argument-slot raising part of the proposed scope switch system, the analysis is still within the non-associative grammar system, NL.

Given some adequate lexical entries, we can treat the string *stand in front of* in (64c) as of category (NP\S)/NP, deriving the scope ambiguity within NL.

Infinitival constructions as in (70) require some more sophistication. As (70) is scopally ambiguous, we want to treat the string *try to review* as a complex predicate. But, if we treat *try* as (NP\S)/(NP\S), and (*to*) *review* as (NP\S)/NP, then we cannot derive a category form of (NP\T)/NP in the non-associative system NL.

(70) A student tried to review every paper. *a>every*, *every>a*

However, we can assign multiple categories to the functor *try*, so that it can be applied to verbal functors of various arities, while preserving the number of the arguments of the verbal functors, as shown in (71).[28] [29]

(71) *try*: (a) <*try*; (NP\S)/(NP\S); $\lambda A.\lambda x.\text{try}'(A)(x)$>
(b) <*try*; ((NP\S)/NP)/((NP\S)/NP); $\lambda P.\lambda x.\lambda y.\text{try}'(P)(x)(y)$>
(c) <*try*; ((NP\S)/(NP•NP))/((NP\S)/(NP•NP)); $\lambda R.\lambda \alpha.\lambda y.\text{try}'(R)(\alpha)(y)$>
Variable types. *A*: (et), *P*: (e(et)), *R*: ((e×e),(e,t)), α: e×e, *x, y*: e

In order to avoid polymorphic category assignments as in (71), we could introduce associativity in a modally controlled way, as I briefly explain later. However, there seems to be a significant linguistic difference between the complex predicate formation as in *try to review* or *may have reviewed* as in the sentence, *A student may have reviewed each paper*, on the one hand, and the phenomena that roughly correspond to A-bar movement in the traditional linguistic theory. I do not have space to discuss the differences here, but the bottom line is that I want to reserve the introduction of a controlled association rule to capture the latter phenomena, not the former, and hence, choose to use the more complex lexical assignments as in (71) to switch scope in the infinitival constructions. Further sophistication is

---

[28] Thanks to Michael Moortgat for suggesting this line of analysis.

[29] Alternatively, we can use a type variable in the lexical category, as in <*try;* X/X; try′>, where X can be instantiated as different functor categories such as (NP\S) and (NP\S)/NP and the semantic term **try′** is assigned the corresponding semantic types. Though these two formulations have different implications elsewhere, they have the same implication in the current discussion. That is, the syntactic system can stay non-associative.

required to explain object control constructions as in (72), thought I leave such cases for another occasion.

(72) An editor asked Tom to review every paper. *a>every*, ?**every>a*

Note that we have switched scope only within a tensed clause within the associative system NL. It is very difficult to treat a string across a tensed clause as a complex predicate without introducing a structural association rule to NL.

(73) (a) Two teachers *reported that* every student *smoked*.
*two>every, *every>two*
(b) Two editors [*wondered if Tom had reviewed*] every paper.
*two>every, *every>two*

Inability of the proposed scope switch system to treat the italicised expression in (73b) as a complex predicate correctly predicts that the sentence does not have the inverse scope reading. In (73a), the italicized expressions are discontinuous in the PF string and my analysis strongly predicts that the scope switch is impossible, which is supported by the data. Thus, the scope-switch system naturally leads to the roughly tensed-clause bound locality constraints on quantifier scope.

Though my analysis can explain the scope data within NL, the total grammar system will still require a controlled introduction of association rule. For example, I have to explain Wh movement of one of the objects in the two kinds of ditransitive constructions, which will require an association rule.

(74) (a) Who$_1$ [*did Meg give each book to*] *(t_1)*?
(b) What$_1$ [*did Meg give to each student*] *(t_1)*?

Without going too much into details, we want to treat the italicized expressions (without the trace positions) in (74a) and in (74b) as categorial type S/NP,[30] while the Wh expressions of category Wh/(S/NP) take them as arguments. In order to do this, however, we need to use a structural association rule.

Morrill (1994), Moortgat (1997) and others have shown that, by using unary operators in categorial formulas, it is possible to introduce association/permutation into NL in a controlled manner, so that the grammar will not collapse into a fully

[30] In the multi-modal TLG as in Moortgat (1997), S/NP might be S/◊NP (or Q/◊NP, if the do-inversion changes S for declarative sentences to Q for question sentences), where, informally, the unary operator ◊ marks the category that triggers structural association. The Wh functor would then be Wh/(S/◊NP), then, where Wh is the category for Wh question sentences. See Vermaat (2006) for a detailed analysis of Wh constructions.

associative/commutative system. On the other hand, as I mentioned in footnote 23, these analyses, which treat Wh movements and QNP scope by using the same schema, fail to explain the rather peculiar locality constraints that are applicable only to quantifier scope, but not to Wh movement (e.g. QNP scope roughly stays within the tensed clause, while Wh movement is generally not blocked by a mere tensed clause). The challenge for me then is whether I can introduce associativity to the grammar system in the way that respects the differences between Wh movements and QNP scope. A merit of my proposal is that the locality of QNP scope is a result of the scope resolution system in terms of the functor that takes the QNP as a syntactic argument. In other words, the scope of a QNP should stay inside the domain of the functor that takes it as a syntactic argument (by way of argument-slot raising), unless some larger quantificational phrase that contains the QNP as its sub-expression takes a wider scope with regard to the functor of that larger quantificational expression.[31] Whatever association rule I will add to the basic grammar system NL should not destroy this merit. I leave this investigation for further research.[32]

5 Conclusion

For scope alternation, the proposed NP argument-slot analysis based on Hendriks (1987) requires an overt functor of the category form (NP\T)/NP between the two QNPs. This explains why the scope relation between the two object QNPs is frozen in the double object construction, as opposed to the prepositional construction. We have an overt functor only in the prepositional construction. Formulated in the non-associative grammar NL, it leads to certain locality constraints. Instantiating my proposal in a deductive grammar system requires the definition of argument-slot raising as a lexical operation that is not fully supported by the logical grammar system. However, being equipped with this lexical operation, we can switch QNP scope within NL. Generalizing the argument-slot raising so that it will be applicable to (B\C)/A, rather than more specific (NP\T)/NP, might lead to a better result. As well as this generalization of the argument-slot raising, I leave for further research the derivation of an axiomatic •I(Q) rule from the basic axiom •I and the

---

[31] As an example, consider the PP ditransitive construction with a subject QNP and an object QNP, such as *A teacher showed* [$_{\text{QDP2}}$ [$_{\text{QDP1}}$ *every book*] *to Mary*]. Though the scope of *every book* as QDP1 stays within the domain of the preposition *to*, that is, the domain marked by the outer pair of square brackets. QDP2 that contains QDP1 resolves its scope with regard to its functor *showed*. As a result, QDP1 can take wide scope over the subject QDP, *a teacher*.

[32] Considering that the tense information is relevant to the locality of QNP scope, how to incorporate tense into TLG should be part of this investigation.

controlled introduction of structural association into the grammar system to explain more complicated data such as Wh movement.

References

Ajdukiewicz, K. (1935) Die syntaktische Konnexität. *Studia Philosophica* 1, 1-27, translated in S. McCall(ed.), 1967, *Polish Logic: 1920-1939,* Oxford University Press, Oxford, 207-231.

Bar-Hillel, Y. (1953) A quasi-arithmetical notation for syntactic description. *Language* 29:47-58.

Barss, A, and H. Lasnik. (1986) A note on anaphora and double objects. *Linguistic Inquiry* 17, 347-354.

Bernardi, R. (2002) *Reasoning with Polarity in Categorial Type Logic.* Ph.D. thesis, Utrecht University, Utrecht, The Netherlands.

Bernardi, R. (2004) Analyzing the Core of Categorial Grammar. *Journal of Logic, Language and Information (JoLLI). 13(2): 121-137, Spring 2004.*

Bruening, B. (2001) QR obeys superiority: frozen scope and ACD. *Linguistic Inquiry* 32, 233-273.

Carpenter, B. (1997) *Type Logical Grammar*. Cambridge, Mass.: MIT Press.

Cormack, A. (1985) VP Anaphora: Variables and Scope. In F. Landman and F. Veltman (eds.), *Variables of Formal Semantics*. Dordrecht: Reidel.

Došen, K. (1992) A Brief Survey of Frames for the Lambek Calculus, *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 38, 179-187.

Gabbay, D. (1996) *Labelled Deductive Systems, volume 1.* Oxford: Oxford University Press.

Hendriks, H. (1987) Type change in semantics: the scope of quantification. In E. Klein and J. van Benthem (eds.), *Categories, Polymorphism and Unification*. Centre for Cognitive Science, University of Edinburgh; ILLC; University of Amsterdam, 96-119.

Jacobson, P. (1999) Towards a variable-free semantics. *Linguistics and Philosophy,* 22: 117-185.

Lambek, J. (1958) The mathematics of sentence structure, *American Mathematical Monthly* 65, 154-170.

Larson, R. (1988) On the double object construction. *Linguistic Inquiry* 19, 335-391.

L.T.F. Gamut (1991) *Logic, Language, and Meaning, volume 2, Intensional logic and logical grammar*. Chicago: University of Chicago Press.

Moortgat, M. (1991) Generalized quantification and discontinuous type constructors. Technical report, Institute for Language and Speech, University of Utrecht. To appear in W. Sijtsma and A. van Horck (eds,) *Discontinuous Constituency*, Berlin, de Gruyter.

Moortgat, M. (1997) Categorial type logic. In J. van Benthem and A. ter Meulen (eds.), *Handbook of Logic and Language*. Cambridge, Mass: MIT Press, 93-178.

Morrill, G. (1994) *Type Logical Grammar: Categorial Logic of Signs*. Dordrecht: Kluwer.

Uchida, H. (2005) Double object in categorial grammar. In Judit Gervain (ed.), *Proceedings of ESSLLI'06 Student Session,* http://www.sissa.it/~gervain/ProceedingsFinalVersion.pdf, 377-388.

Venema, Y. (1994) Tree Models and (labelled) categorial grammar. In P. Dekker and M. Stokhof(eds.), *Proceedings of the 9th Amsterdam Colloquium.* Institute of Logic, Language and Information, Amsterdam: 703-722.

Vermaat, W. (2006) *The Logic of Variation; a cross-linguistic account of wh-question formation* PhD. thesis, Utrecht University, Utrecht, The Netherlands.

Versmissen, K. (1996) *Grammatical Composition; modes, models, modalities, logical and linguistic aspects of multi-modal categorial grammars.* OTS dissertation series. Utrecht, The Netherlands: Led.