

Types as Graphs: Continuations in Type Logical Grammar

CHRIS BARKER¹ and CHUNG-CHIEH SHAN²

¹*Department of Linguistics, University of California, San Diego, USA*

E-mail: barker@ucsd.edu

²*Department of Computer Science and Center for Cognitive Science, Rutgers, The State University of New Jersey*

E-mail: ccshan@rutgers.edu

(Received: 7 July 2004; accepted: 29 June 2005)

Abstract. Using the programming-language concept of CONTINUATIONS, we propose a new, multi-modal analysis of quantification in Type Logical Grammar. Our approach provides a geometric view of *in-situ* quantification in terms of graphs, and motivates the limited use of empty antecedents in derivations. Just as continuations are the tool of choice for reasoning about evaluation order and side effects in programming languages, our system provides a principled, type-logical way to model evaluation order and side effects in natural language. We illustrate with an improved account of quantificational binding, weak crossover, *wh*-questions, superiority, and polarity licensing.

Key words: continuations, type-logical grammar, resource sensitivity, quantification, polarity licensing, binding, superiority, evaluation order, staging, delimited continuations

1. Introduction

In recent research, the programming-language concept of CONTINUATIONS has provided much insight into a variety of natural-language phenomena: quantification and coordination (Barker, 2002; de Groote, 2001), binding and interrogation (Shan, 2002; Shan and Barker, 2006), focus and hypallage (Barker, 2004), and polarity sensitivity (Shan, 2004). As we explain in detail below, continuations are a compositional mechanism that provides expressions with access to (a portion of) their semantic context.

Most linguistic research on continuations (including much of our own work) is couched in combinatory categorial grammars that, like Jacobson's (1999, 2000) and Steedman's (2000), accomplish the main work by type-shift operators. To explore the role of continuations in linguistic formalisms other than combinatory categorial grammars, we propose in this paper a continuation-based approach to quantification in Type Logical Grammar (TLG). We show how continuations lead to an unusually parsimonious multimodal account of quantifier scope. In particular, we use continuations to reduce the ternary type constructor q proposed by Moortgat

for *in-situ* quantification to multimodal TLG. The basic analysis involves one new mode, related to the default mode by two structural postulates. As we explain in Section 3, these postulates can be understood geometrically in terms of univalent graphs.

Continuations have traditionally been used to explore computational issues like evaluation order (such as left-to-right versus right-to-left, and call-by-value versus call-by-name) and side effects. At first blush, logical proof is independent of evaluation order: a TLG derivation has no obvious notion corresponding to ‘before’ versus ‘after’, or ‘value’ versus ‘name’. Yet, by the Curry-Howard isomorphism, logical proof is the same thing as computation, so whenever evaluation order plays a role in computation, it must play a role in logical proof. If so, we need a way to reason about evaluation order in TLG derivations, which are independent of order. This is precisely what continuations provide: a principled tool for reasoning about order of evaluation in an order-independent way.

Moortgat (1997) and others emphasize that TLG provides a perspicuous framework for exploring the resource-sensitivity of natural languages. Order of evaluation is just one more kind of structural resource. It is an empirical question whether natural languages are sensitive to evaluation order (and other constraints on side effects). We argue they are in Sections 8 and 9. Thus adding continuations to TLG enables us to explore a new realm of the Curry-Howard correspondence, and provides for the grammar-writer a new and (we argue) useful type of resource sensitivity.

This paper consists of two main parts. The next three sections present the core idea of implementing *in-situ* quantification by residuating on contexts and subexpressions. The remainder of the paper extends the basic analysis in a variety of ways, most importantly adding explicit control over the order in which subexpressions are evaluated (Section 8). This lets us provide new TLG analyses of quantificational binding, weak crossover, *wh*-questions, superiority, and polarity licensing that improve on the empirical predictions of previous TLG accounts (Section 9).

2. Quantification in Type Logical Grammar

There are a variety of approaches to quantification in TLG, including Moortgat’s type constructor q (1988, 1995, 1996) and at least three multimodal implementations of q (Morrill, 1994; Moortgat, 1995, 2000), briefly summarized by Bernardi (2003). The multimodal analyses all reconstruct the q operator in terms of modes and structural postulates, and our analysis below does the same. One important difference is that our analysis deliberately exploits continuations, which, as we will see, enables the grammar-writer to explicitly reason about order of evaluation and other side-effects.

The q operator constructs types of the form $q(A, B, C)$. Conceptually, an expression of this type behaves locally as if it were an expression of type A , takes

scope over an expression of type B , and yields a result expression of type C . For instance, a quantificational NP like *everyone* may have type $q(np, s, s)$: it functions locally as if it were a normal NP but takes scope over a clause to yield a clause. (As one might expect, this type corresponds semantically to a generalized quantifier.) For another example, an *in-situ wh*-phrase may have type $q(np, s, wh)$: it functions locally as an NP but takes scope over a clause and turns the clause over which it takes scope into a question, of type wh . (Moortgat (2000) gives other examples motivating analyses for which $B \neq C$.)

Multimodal implementations of q seek to characterize how quantification depends on structural resources such as associativity or commutativity. For instance, Moortgat (1997, 125) argues that since quantifiers can take scope both over material to their left and to their right, the plain Lambek calculus, whether associative or not, simply does not have the expressive power to deal with quantification in a general way. He concludes that some degree of commutativity is essential, and our analysis below is consistent with that claim.

Though we will not present other multimodal implementations of q in detail, we describe their general features here.

Morrill (1994) uses three binary modes: the non-associative default mode \circ , an associative mode \circ_a , and a wrapping mode \circ_w . Three postulates allow the modes to interact in such a way that a quantificational NP of type $(s/w np)\backslash_w s$ has the desired behavior.

In the first of his two analyses, Moortgat (1995) uses two binary modes and three unary modes. There are three postulates. The type of a quantificational NP is

$$\diamond(s/w(\Box^4 np\backslash_w s)). \tag{1}$$

Moortgat's second analysis (2000) has three binary modes and four single-sided postulates. The type of a quantificational NP is

$$(s/+(np\backslash_-s)) \circ_+ (np\backslash_-np). \tag{2}$$

Note that $np\backslash_-np$, the right-hand argument of the fusion connective, in effect introduces an empty antecedent into the derivation. Empty antecedents also play an important role in our analysis of quantification, as discussed below in Section 6.

Because q was designed to capture the behavior of quantificational elements, it naturally resembles how quantificational elements are treated in a continuation-based grammar such as Barker's (2002) and Shan and Barker's (2006). Our TLG analysis has two binary modes, the default, external mode \circ and an additional, internal mode \odot . Unlike most TLG analyses of quantification, neither mode needs to be associative for our purposes (nor do we require either mode to be non-associative). As explained below, the two modes are governed by two postulates, and the lexical

type of a quantificational NP is $s//(\text{np}\backslash s)$. Here we write $//$ and \backslash for residuation in the additional mode \odot .

3. Fusion Types as Trees

In this section, we introduce a graphical interpretation of TLG types in terms of univalent graphs. In Section 4, we apply this fairly abstract discussion to implement q for linguistic analyses. In Section 5, we relate quantification in TLG to continuations in logic and computation via the Curry-Howard isomorphism.

Every TLG type built from atomic symbols using only \circ can be uniquely drawn as a binary tree. A binary tree is an acyclic univalent graph in which every leaf node but one (the ROOT) is labeled with an atomic symbol. A univalent graph is a graph in which every node either is a leaf or has exactly three edges. For example, the type

$$A = a \circ ((b \circ c) \circ d) \tag{3}$$

can be drawn as follows.

$$a \circ ((b \circ c) \circ d) \iff \begin{array}{c} & b & c \\ & \diagdown & / \\ & & | \\ a & \diagup & \diagdown \\ | & & d \end{array} \tag{4}$$

In our graphs, edges are unoriented, so the following two graphs are identical:

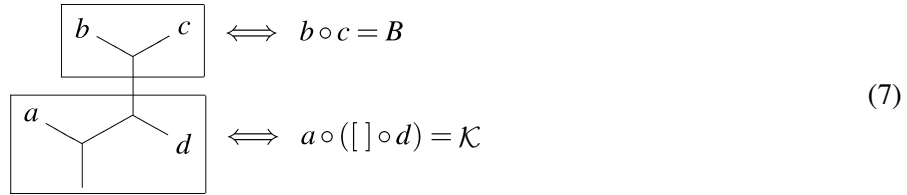
$$\begin{array}{c} a \\ | \end{array} = \begin{array}{c} | \\ a \end{array} \tag{5}$$

However, we do orient nodes by designating, for each trivalent node, one of the two cyclic orderings of its edges as clockwise. Hence each of the following graphs are equal to two others, but not to their mirror images.

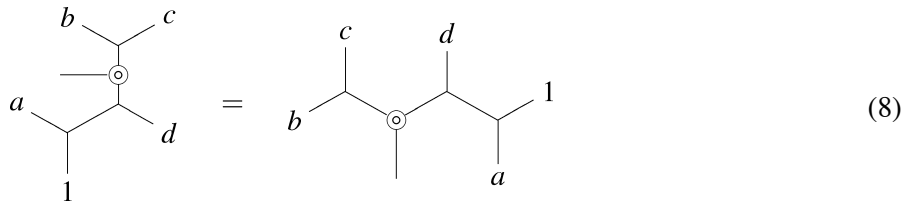
$$\begin{array}{c} a & & b \\ & \diagdown & / \\ & & | \\ & & c \end{array} = \begin{array}{c} b & & c \\ & \diagdown & / \\ & & | \\ & & a \end{array} = \begin{array}{c} c & & a \\ & \diagdown & / \\ & & | \\ & & b \end{array} \not\equiv \begin{array}{c} a & & c \\ & \diagdown & / \\ & & | \\ & & b \end{array} = \begin{array}{c} c & & b \\ & \diagdown & / \\ & & | \\ & & a \end{array} = \begin{array}{c} b & & a \\ & \diagdown & / \\ & & | \\ & & c \end{array} \tag{6}$$

Together, these two conventions ensure that each type corresponds to only one graph.

Suppose now that A and B are two types, both built using only \circ , such that B appears as part of A . For instance, A might be the type $a \circ ((b \circ c) \circ d)$ as above, and B might be the type $b \circ c$. We write $A = \mathcal{K}[B]$, where \mathcal{K} is the context of B relative to A ; that is, $\mathcal{K} = a \circ ([] \circ d)$. Graphically speaking, we have decomposed A into two parts, B and \mathcal{K} , by ripping apart the graph at the upper vertical edge, calling the side that contains the root node \mathcal{K} , and calling the other side B .



To represent such a decomposition as a graph, we insert a new, special node \odot in the middle of the edge that connects the two components.



This new node \odot connects the two components of the decomposition to a new root node. The old root node becomes a leaf with the special label 1. We know which side of the \odot corresponds to the context, since that's the side that contains the 1 node marking the position of the old root. Starting at the \odot node, we always put the context side right after the subexpression side (where “after” means moving clockwise).

To convert the diagram (8) back to a type, we introduce a new binary mode, the *continuation* or *context* mode. We write fusion, left residuation, and right residuation for the continuation mode as \odot , \backslash , and $/$. The graph in (8) thus corresponds to the formula

$$(b \circ c) \odot (d \circ (1 \circ a)). \tag{9}$$

This mode is internal (or unpronounceable, informally speaking).

The continuation mode \odot treats the decomposition $A = \mathcal{K}[B]$ as a structure in its own right: $B \odot K$ decomposes A into the subexpression B and the context \mathcal{K} .¹

¹We use script \mathcal{K} to stand for the usual notion of a context as an expression containing a hole (where ‘usual’ means as in discussions of programming languages, like Barendregt’s presentation of the λ -calculus (1981)); and we use plain K to stand for the TLG type that we interpret as representing a context.

In the trivial case, the graph is ripped apart at the root edge into two sides: the null context ($[\]$), and the expression A viewed as its own (improper) subexpression. Because the null context is represented by the type 1 , we henceforth treat 1 as the right identity for \odot . Thus $B \odot 1$ is logically equivalent to just B . (The presence of a right identity affects the nature of the \odot mode, and plays an important role in the discussion below; see especially Section 6.)

In the original type $A = a \circ ((b \circ c) \circ d)$ in (3), the type d follows a and fuses with $b \circ c$, whereas in (9), the type d precedes a and fuses with $1 \circ a$. It appears that our types represent a context by scrambling the original elements – in fact, by turning the original expression inside-out. But this reordering only occurs in the symbolic representation as types, as in (9). When viewed graphically, as in (8), there is no scrambling, and no turning inside-out: we have simply inserted a continuation node into the chosen edge, without reordering anything. Put another way, we have not changed the structure of the formula; we have merely changed our perspective on the formula by placing one subformula in the foreground and backgrounding the rest. Thus the context $d \circ (1 \circ a)$ is not really “inside out” – that’s just what the context looks like from the perspective of the continuation node. The usefulness of the graphical interpretation is precisely that it makes the mapping from contexts to types as simple as choosing an edge.

This technique – representing a meta-level notion of context like $\mathcal{K} = a \circ ([] \circ d)$ as a object-level formula like $K = d \circ (1 \circ a)$ – embodies the crucial insight of continuations. We shall see that allowing the logic to operate on types representing contexts in effect provides expressions with (limited) access to their own context.

We are now ready to introduce some structural rules to relate the various equivalent ways in which the same graph can be decomposed. If we find some type of the form $B \circ C$ inside a context \mathcal{K} , we can consider moving either B or C into the context part of the decomposition. In other words, if $A = \mathcal{K}[B \circ C]$, then three decompositions of A are available: one that isolates B , one that isolates $B \circ C$, and one that isolates C . If the type K represents the context of $B \circ C$ (so that it contains 1 in place of the root of A), then Figure 1 depicts these three decompositions. Since these are three decompositions of the same graph, we want them to be logically equivalent, that is, mutually derivable. To this end, we add two structural

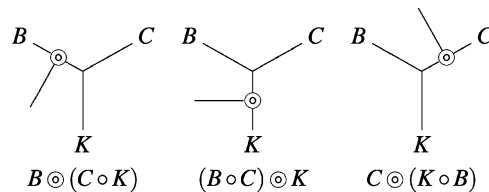


Figure 1. Three ways to decompose $A = \mathcal{K}[B \circ C]$ into a subexpression and a context.

postulates.²

$$\begin{aligned}
 B \odot (C \circ K) &\dashv\vdash (B \circ C) \odot K && \text{(Left)} \\
 (B \circ C) \odot K &\dashv\vdash C \odot (K \circ B) && \text{(Right)}
 \end{aligned}
 \tag{10}$$

Using these structural rules, any decomposition of a (connected) graph A can be derived from the trivial decomposition $A \odot 1$, which represents A itself inside the null context (which, we repeat, is logically equivalent to just A , given that 1 is the right identity for \odot). For example, the decomposition in (8) can be derived from the trivial decomposition in two steps.

$$(a \circ ((b \circ c) \circ d)) \odot 1 \dashv\vdash^{Right} ((b \circ c) \circ d) \odot (1 \circ a) \dashv\vdash^{Left} (b \circ c) \odot (d \circ (1 \circ a))
 \tag{11}$$

The continuation mode \odot and the structural postulates Left and Right expand the machinery of residuation built-in to categorical grammar beyond describing syntactic elements that take arguments leftward or rightward. We can now describe syntactic elements that take arguments not leftward or rightward, but ‘inward’ or

² Unfortunately, in the presence of these two structural postulates and the right identity 1 for the \odot mode, the default mode \circ becomes commutative.

$$\begin{array}{ccccccc}
 B \circ C & \xrightarrow{\text{Pop/Push}} & (B \circ C) \odot 1 & \xrightarrow{\text{Left}} & B \odot (C \circ 1) & \xrightarrow{\text{Pop/Push}} & B \odot ((C \circ 1) \odot 1) & \xrightarrow{\text{Left}} & B \odot (C \odot (1 \circ 1)) \\
 C \circ B & \xrightarrow{\text{Pop/Push}} & (C \circ B) \odot 1 & \xrightarrow{\text{Right}} & B \odot (1 \circ C) & \xrightarrow{\text{Pop/Push}} & B \odot ((1 \circ C) \odot 1) & \xrightarrow{\text{Right}} & B \odot (C \odot (1 \circ 1))
 \end{array}$$

There are at least three ways to prevent the default mode \circ from commuting thus.

1. In Section 8 below, to enforce a notion of left-to-right evaluation, we restrict the Right rule to apply only when the formula B is surrounded by \diamond . With this restriction, the Right inferences above (especially the one in the lower-right corner) no longer apply.
2. We can remove 1 from the grammar and replace every type A throughout the lexicon that is either a top-level formula or on the numerator side of a slash with the type $A // (A \setminus A)$. This move prevents 1 from being freely introduced by Push, as in the derivation above.
3. We can change the Left and Right rules to

$$B \odot (C \triangleleft K) \dashv\vdash (B \circ C) \odot K \quad \text{(Left)} \quad (B \circ C) \odot K \dashv\vdash C \odot (K \triangleright B) \quad \text{(Right)},$$

where \triangleleft and \triangleright are two new binary modes. Informally speaking, we distinguish between ‘left contexts’ (\triangleleft) and ‘right contexts’ (\triangleright) to rule out the adjacent application of Left and Right above. Rules like these may be familiar from Belnap’s display logic (1982) and Huet’s zipper data-structure (1997; Abbott et al., 2003; Hinze and Jeuring, 2001).

Perhaps solutions 1 and 2 are natural if one blames the presence of multiple 1 ’s in the derivation above for making the default mode commutative, whereas solutions 1 and 3 are natural if one blames the commutativity on confusing Left with Right, or subexpression with context. Because solution 1 is deployed below, we leave the commutativity issue aside here.

‘outward’, in senses to be explained in the next section. We will argue, following Barker (2002), that this is the very essence of *in-situ* quantification.

4. Scope-Taking as Residuation on Contexts and Subexpressions

Let us apply the techniques developed above to some syntactic categories and lexical items of linguistic relevance. We assume a type of noun phrases (more precisely, proper nouns) np and a type of clauses s . For example, *Alice saw Bob* is a clause.

$$\frac{\frac{\text{Alice} \vdash np \quad \frac{\text{saw} \vdash (np \backslash s) / np \quad \text{Bob} \vdash np}{\text{saw} \circ \text{Bob} \vdash np \backslash s} / E}{\text{Alice} \circ (\text{saw} \circ \text{Bob}) \vdash s} \backslash E}{\text{Alice} \circ (\text{saw} \circ \text{Bob}) \vdash s} \backslash E \quad (12)$$

In general, any two NPs with *saw* in between form a clause.

$$\frac{\frac{\frac{\text{np} \vdash np \quad \text{Id}}{\text{np} \vdash np} \text{Id} \quad \frac{\text{saw} \vdash (np \backslash s) / np \quad \frac{\text{np} \vdash np \quad \text{Id}}{\text{np} \vdash np} / E}{\text{saw} \circ np \vdash np \backslash s} / E}{\text{np} \circ (\text{saw} \circ np) \vdash s} \backslash E}{\text{np} \circ (\text{saw} \circ np) \vdash s} \backslash E \quad (13)$$

We proceed by drawing a (limited) analogy between what we call contexts here, and clauses containing a gap. The string *Alice saw* is a gapped clause. In other words, it is a context whose hole can be filled in with np to yield an s . Following the graphical approach explained in the previous section, we can represent the gapped clause as a type expression, or equivalently, as a univalent graph.

$$(1 \circ \text{Alice}) \circ \text{saw} \iff \begin{array}{c} \text{saw} \\ \diagdown \quad \diagup \\ \quad \quad \quad | \\ \text{Alice} \quad \quad 1 \end{array} \quad (14)$$

Figure 2 uses the structural rules in (10) to prove that this context is a gapped clause, in other words, that the type in (14) entails the type $np \backslash s$. The latter type is the type of something that gives s when fused on the left with an np using \odot ; in other words, a gapped clause has the type of a context that encloses an np to give an s . Push and Pop reflect the fact that 1 is a right identity for \odot ; they are roughly equivalent to introducing (Push) an empty antecedent into the derivation, and later eliminating (Pop) it. They are discussed in more detail in Section 6, where we implement Push and Pop in terms of standard TLG technology based on unary modes.

This treatment of gapped clauses is insensitive to the linear location of the gap: it works uniformly regardless of whether the gap is at the left or right edge of the clause, unlike several type-logical treatments of extraction and quantification.

$$\begin{array}{c}
\vdots (13) \\
\frac{np \circ (saw \circ np) \vdash s}{(np \circ (saw \circ np)) \odot 1 \vdash s} \text{Push} \\
\frac{(np \circ (saw \circ np)) \odot 1 \vdash s}{(saw \circ np) \odot (1 \circ np) \vdash s} \text{Right} \\
\frac{(saw \circ np) \odot (1 \circ np) \vdash s}{np \odot ((1 \circ np) \circ saw) \vdash s} \text{Right} \\
\frac{np \odot ((1 \circ np) \circ saw) \vdash s}{(1 \circ np) \circ saw \vdash np \backslash s} \backslash I \\
\frac{\text{everyone} \vdash s // (np \backslash s) \quad (1 \circ np) \circ saw \vdash np \backslash s}{\text{everyone} \odot ((1 \circ np) \circ saw) \vdash s} // E \\
\frac{\text{everyone} \odot ((1 \circ np) \circ saw) \vdash s}{(saw \circ everyone) \odot (1 \circ np) \vdash s} \text{Right} \\
\frac{(saw \circ everyone) \odot (1 \circ np) \vdash s}{(np \circ (saw \circ everyone)) \odot 1 \vdash s} \text{Right} \\
\frac{(np \circ (saw \circ everyone)) \odot 1 \vdash s}{np \odot ((saw \circ everyone) \circ 1) \vdash s} \text{Left} \\
\frac{np \odot ((saw \circ everyone) \circ 1) \vdash s}{(saw \circ everyone) \circ 1 \vdash np \backslash s} \backslash I \\
\frac{\text{someone} \vdash s // (np \backslash s) \quad (saw \circ everyone) \circ 1 \vdash np \backslash s}{\text{someone} \odot ((saw \circ everyone) \circ 1) \vdash s} // E \\
\frac{\text{someone} \odot ((saw \circ everyone) \circ 1) \vdash s}{(someone \circ (saw \circ everyone)) \odot 1 \vdash s} \text{Left} \\
\frac{(someone \circ (saw \circ everyone)) \odot 1 \vdash s}{someone \circ (saw \circ everyone) \vdash s} \text{Pop}
\end{array}$$

Figure 4. Linear scope for *Someone saw everyone*.

$$\begin{array}{c}
\vdots (13) \\
\frac{np \circ (saw \circ np) \vdash s}{(np \circ (saw \circ np)) \odot 1 \vdash s} \text{Push} \\
\frac{(np \circ (saw \circ np)) \odot 1 \vdash s}{np \odot ((saw \circ np) \circ 1) \vdash s} \text{Left} \\
\frac{np \odot ((saw \circ np) \circ 1) \vdash s}{(saw \circ np) \circ 1 \vdash np \backslash s} \backslash I \\
\frac{\text{someone} \vdash s // (np \backslash s) \quad (saw \circ np) \circ 1 \vdash np \backslash s}{\text{someone} \odot ((saw \circ np) \circ 1) \vdash s} // E \\
\frac{\text{someone} \odot ((saw \circ np) \circ 1) \vdash s}{(someone \circ (saw \circ np)) \odot 1 \vdash s} \text{Left} \\
\frac{(someone \circ (saw \circ np)) \odot 1 \vdash s}{(saw \circ np) \odot (1 \circ someone) \vdash s} \text{Right} \\
\frac{(saw \circ np) \odot (1 \circ someone) \vdash s}{np \odot ((1 \circ someone) \circ saw) \vdash s} \text{Right} \\
\frac{np \odot ((1 \circ someone) \circ saw) \vdash s}{(1 \circ someone) \circ saw \vdash np \backslash s} \backslash I \\
\frac{\text{everyone} \vdash s // (np \backslash s) \quad (1 \circ someone) \circ saw \vdash np \backslash s}{\text{everyone} \odot ((1 \circ someone) \circ saw) \vdash s} // E \\
\frac{\text{everyone} \odot ((1 \circ someone) \circ saw) \vdash s}{(saw \circ everyone) \odot (1 \circ someone) \vdash s} \text{Right} \\
\frac{(saw \circ everyone) \odot (1 \circ someone) \vdash s}{(someone \circ (saw \circ everyone)) \odot 1 \vdash s} \text{Right} \\
\frac{(someone \circ (saw \circ everyone)) \odot 1 \vdash s}{someone \circ (saw \circ everyone) \vdash s} \text{Pop}
\end{array}$$

Figure 5. Inverse scope for *Someone saw everyone*.

Raising, or even systems fairly closely related to TLG such as Jacobson's (1999, 2000) combinatory categorial grammar, as discussed by Barker (2005).

5. Continuations in Logic and Computation

The remainder of the paper refines the basic analysis above with a view towards detailed empirical description. One main goal will be to control order of evaluation explicitly and argue for its linguistic relevance. Since evaluation order is a

computational notion that has not been discussed in TLG before, we first introduce continuations and evaluation order from logical and computational points of view, then turn to their linguistic applications.

5.1. CLASSICAL LOGIC

In a proof in classical logic, as in an utterance in natural language, a subexpression can implicitly take its context as argument. We illustrate this idea in Gentzen’s sequent calculus for classical logic LK (1935). A sequent in LK is of the form $\Gamma \vdash \Delta$, where Γ and Δ are each a sequence of formulas. It means that the conjunction of Γ entails the disjunction of Δ . We treat Γ and Δ as sets rather than sequences of formulas, by omitting in proofs the structural rules Contract and Exchange (which apply freely in LK).

The Cut rule in LK ‘cancels an intermediate formula’ A between two subproofs.

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'} \text{Cut} \quad (15)$$

The cut can be reduced away either by substituting the left subproof into the right subproof or vice versa. The effect of the direction of substitution is starkest when both subproofs immediately discard the A by weakening the same sequent $\Gamma \vdash \Delta$.

$$\begin{array}{c} \vdots \alpha \\ \Gamma \vdash \Delta \end{array} \iff \frac{\frac{\vdots \alpha}{\Gamma \vdash \Delta} \text{Weaken}}{\Gamma \vdash A, \Delta} \iff \frac{\frac{\vdots \beta}{\Gamma \vdash \Delta} \text{Weaken}}{\Gamma, A \vdash \Delta} \iff \frac{\vdots \beta}{\Gamma \vdash \Delta} \quad (16)$$

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta} \text{Cut}$$

(This example is due to Lafont (Girard et al., 1989, Appendix B).) The reduction selects one of α and β and discards the other: either α takes its ‘context’ β as argument and discards it, or β takes its ‘context’ α as argument and discards it. Hence LK is not confluent.

When A is the type of individuals (np or e) and falsum \perp is the type of clauses (s or t), this substitution roughly lets a quantificational NP α take scope over the rest of a sentence (de Groote, 2001). In classical logic (unlike in intuitionistic logic), because α may use its context more than once (by contraction) or not at all (by weakening), reducing different cuts gives different proofs. This nonconfluence, which results when multiple subproofs take their contexts as argument, roughly corresponds to the ambiguity that results when multiple quantifiers take scope (de Groote, 2001).

The Curry-Howard isomorphism says that to reduce a proof of a formula is to run a program of a type (Howard, 1980). Griffin (1990) discovered that this isomorphism applies to not just intuitionistic but also classical logic: for a subproof to take its

context as argument is for a subprogram to manipulate its CONTINUATION, which is the future of the computation from the completion of the subprogram onward. The order in which subproofs are reduced corresponds to the order (Meyer and Wand, 1985, page 223; Danvy and Filinski, 1989, page 15; Papaspyrou, 1998) in which subprograms are evaluated: the context (or future, or scope) of an earlier subexpression contains later subexpressions.

Each DOUBLE-NEGATION translation (Gentzen, 1933; Gödel, 1933; Kolmogorov, 1925) of proofs, from classical logic to intuitionistic logic, corresponds (Murthy, 1990) to a CONTINUATION-PASSING-STYLE translation (Fischer, 1972, 1993; Plotkin, 1975) of programs, from a language with facilities for manipulating continuations to a language without. These translations disambiguate order and restore confluence in their output, just as the TLG derivations in Figures 4 and 5 make scope explicit in a logic with no (built-in) facility for *in-situ* quantification: *someone* is evaluated before *everyone* in Figure 4, and after in Figure 5.

5.2. COMPUTING WITH DELIMITED CONTINUATIONS

Our informal notion of context in LK only roughly corresponds to quantifier scope in natural language, for two related reasons. First, not every quantifier takes matrix scope, so we want to substitute into a subproof only part of its context. Computationally speaking, we want to let a subexpression in a program manipulate not the entire future of the computation, but only a prefix of that future. Second, not every quantifier takes scope over a clause of the same type to give a clause of the same type (Moortgat, 1996; Bernardi and Moot, 2001; Bernardi, 2002; see also Section 9 below), so a single clause type \perp is not enough. For these reasons, *in-situ* quantification in natural language really corresponds to DELIMITED continuations.

In this section, we approach the ideas of delimiting continuations and restricting evaluation contexts from the computational side of the Curry-Howard correspondence, that is, by extending the λ -calculus. However, Section 6 shows how delimited continuations underlie our TLG analysis of *in-situ* quantification, and Section 7 and Section 8 show how restricting evaluation contexts enhance it.

Landin (1966, 1998), Reynolds (1998), and the Scheme programming language (Kelsey et al., 1998) introduced the idea of letting a subprogram manipulate the future of the computation. For example, in the program

$$1 + 10 \times (\mathcal{F}k.k(3) + k(4)), \quad (17)$$

the subprogram $\mathcal{F}k.k(3) + k(4)$ invokes its continuation $1 + 10 \times []$ twice: once on 3 and once on 4. The program executes as follows.

$$1 + 10 \times (\mathcal{F}k.k(3) + k(4)) \Rightarrow (1 + 10 \times 3) + (1 + 10 \times 4) \Rightarrow 72 \quad (18)$$

In linguistic terms, the quantificational expression $\mathcal{F}k.k(3)+k(4)$ takes matrix scope over its entire context. Felleisen (1987, 1988) extended this idea to manipulate only a prefix of the future. The prefix is delimited by a PROMPT, written $\#$. For example, the program

$$1 + \#(10 \times (\mathcal{F}k.k(3) + k(4))) \quad (19)$$

binds k to the *delimited* continuation $10 \times []$, which excludes the context beyond the prompt $\#$. The program executes as follows.

$$\begin{aligned} &1 + \#(10 \times (\mathcal{F}k.k(3) + k(4))) \\ &\Rightarrow 1 + \#(10 \times 3 + 10 \times 4) \Rightarrow 1 + 70 \Rightarrow 71 \end{aligned} \quad (20)$$

In linguistic terms, the quantificational expression $\mathcal{F}k.k(3) + k(4)$ takes scope at the prompt.

The result of a program that uses constructs like \mathcal{F} and $\#$ depends on the order in which subexpressions are evaluated. For example, the program

$$(\mathcal{F}k.1) + (\mathcal{F}k.2) \quad (21)$$

results in either 1 or 2, depending on which side of $+$ is evaluated first. The program

$$(\lambda x.1)(\mathcal{F}k.2) \quad (22)$$

also results in either 1 or 2, depending on whether the function $\lambda x.1$ is invoked first or the argument $\mathcal{F}k.2$ is evaluated first.

One popular evaluation order in programming languages is LEFT-TO-RIGHT and CALL-BY-VALUE. Left-to-right evaluation forces $\mathcal{F}k.1$ in (21) to be evaluated first, so the result is 1. Call-by-value evaluation means to always evaluate the argument before invoking the function, so (22) gives 2. Roughly, a VALUE here is an expression that cannot execute further, like 72 and $\lambda x.1$ but not $\mathcal{F}k.2$.

To regulate evaluation, Felleisen (1987) introduced EVALUATION CONTEXTS. An evaluation context is a context in which a subprogram can be evaluated. For example, the context of $\mathcal{F}k.2$ in (21) is $(\mathcal{F}k.1) + []$, so the program (21) can result in 2 only if the context $(\mathcal{F}k.1) + []$ is an evaluation context. To enforce left-to-right evaluation, we can prohibit such an evaluation context, where the hole $[]$ appears to the right of a non-value like $\mathcal{F}k.1$. To enforce call-by-value evaluation, so that the function $\lambda x.1$ in (22) cannot take the argument $\mathcal{F}k.2$ right away, we can restrict λ -conversion (also known as β -reduction) to only operate when the argument is a value, and prohibit an evaluation context where the hole $[]$ appears inside a λ -abstraction.

6. Empty Antecedents Delimit Continuations

With the concept of a delimited continuation in hand, we can turn to the role of empty antecedents. Throughout this paper, we assume a right identity 1 for the continuation mode \odot , or equivalently, two structural rules Push and Pop that operate to the right of the \odot mode. From a logical perspective, a logic with 1 corresponds to a modal frame with right identity (Restall, 2000). From a computational perspective, it corresponds to a programming language with delimited continuations, since Pop introduces and Push eliminates the prompts mentioned in Section 5.2. Assuming that 1 is present is equivalent to allowing the antecedent to be empty in the conclusion of the $\backslash\text{I}$ rule, because we can treat each occurrence of 1 as just shorthand for the type $s\backslash s$ or some other type of the form $A\backslash A$, which can be derived using such a permissive $\backslash\text{I}$ rule.

However, dating back at least to Lambek’s original paper on his linguistic calculus (1958), there is a tradition of prohibiting empty antecedents in TLG. There is some linguistic justification for the prohibition. For instance, assume that *very* has a category appropriate for an adjective modifier, say, $(n/n)/(n/n)$ as in *a very tall man*. Then since the identity type n/n would be a theorem if we allow empty antecedents, we incorrectly derive the ungrammatical **a very man*.

Yet there is no logical reason to disallow empty antecedents (Moot, 2002, page 74). Nor is this work the first to find empty antecedents linguistically useful in certain situations. For one, Moot (2002, page 85) suggests that allowing empty antecedents can be convenient given a certain analysis of pied piping. For another, consider the type of Moortgat’s (2000) *in-situ*-quantificational NP, shown in (2) and repeated below.

$$(s/+(np_s)) \circ_+ (np_np). \quad (2)$$

Moortgat informally says that the role of the identity type np_np is to ‘stay in place’ while the quantificational part $s/+(np_s)$ ‘travels upwards’. If empty antecedents (or a right identity) were available for Moortgat’s \circ_- mode, the lexical type of a quantificational NP could be simply $s/+(np_s)$. Thus Moortgat implicitly motivates making empty antecedents available for the \circ_- mode, at least under some circumstances.

The useful derivations in our system never fuse two 1’s together, as in $1 \circ 1$ or more complex structures like $1 \circ ((1 \circ 1) \circ 1)$. It turns out that the presence of the right identity 1 can thus be simulated at the cost of proliferating combination modes, structural postulates, and lexical types. The basic idea of this simulation is to translate types with 1 into logically equivalent types without 1. We add two new unary modes to the grammar, which are internal like \odot :

$$\diamond_{\circ_1} A \quad \text{to replace} \quad A \circ 1, \quad \diamond_{1\circ} A \quad \text{to replace} \quad 1 \circ A. \quad (23)$$

The computation in (24) then instructs us to add the two postulates in (25).

$$\begin{array}{ccc}
 B \odot (C \circ 1) & \xrightarrow{\text{Left}} & (B \circ C) \odot 1 & \xrightarrow{\text{Right}} & C \odot (1 \circ B) \\
 & & \begin{array}{c} \text{Pop/Push} \\ \text{---} \\ B \circ C \end{array} & & \\
 & & B \circ C & &
 \end{array} \tag{24}$$

$$B \odot \diamond_{\circ 1} C \dashv\vdash B \circ C \quad (\text{Left}') \quad B \circ C \dashv\vdash C \odot \diamond_{1 \circ} B \quad (\text{Right}') \tag{25}$$

Finally, if any lexical item takes a type of the form $A \backslash A$ or $A // A$ as argument, then another lexical type needs to be added that does not take that argument. For example, a lexical item of type $wh // (s \backslash s)$ must be made ambiguous between that type and the type wh . The semantic value of the argument missing from the latter version is the identity function on s .

To summarize: the selective use of empty antecedents (or equivalently, the presence of an identity for the \odot mode) to delimit scope is motivated by this and other linguistic analyses in TLG, but can be obviated by adding the two unary modes in (23) and the two postulates in (25), and introducing lexical polysemy.³

7. Refining the Analysis of Quantification by Refining Notions of Context

This section extends the coverage of the analysis of quantification above in two ways. First, we discuss situations in which one quantificational NP contains another, as in *every dean of a school*. The challenge is to understand how *every* can be evaluated before (i.e., take scope over) something it contains. Second, we discuss scope islands: expressions that limit the scope of quantifiers they contain. For instance, it is generally assumed that *everyone* in *Someone thought everyone left* cannot take scope outside of the embedded tensed clause *everyone left*; if so, then tensed clauses are scope islands, and the challenge is to enforce them.

In both cases, we will limit scope-taking possibilities by limiting what we allow as a legitimate semantic context. By limiting the access of an expression to its context, we limit what it can take scope over. This same strategy will provide the mechanism for controlling order of evaluation in general, as explained in Section 8.

As explained in Section 3, contexts and subexpressions can be represented as types in multimodal TLG using a new mode and two structural postulates with a geometric interpretation. The postulates for this new mode \odot – be they described pictorially as in Figure 1, or symbolically as in (10) – specify a notion of context

³ Because the postulates in (25) are EXPANDING, the grammar thus extended is no longer guaranteed to be PSPACE-parsable by virtue of Moot’s PSPACE-completeness result for $NL \diamond_{\mathcal{R}}$ - (2002, Section 9.2). If non-expanding postulates are desired, empty antecedents can still be obviated, at the cost of further complicating the grammar: add yet another unary mode \diamond_{\circ} , and replace $B \circ C$ with $\diamond_{\circ}(B \circ C)$ throughout.

that allows descending recursively into either the left branch or the right branch of a \circ -fused constituent. That is:⁴

A context is one of the following.

- a. A ‘hole to the left’ $[] \circ C$, embedded in some outer context \mathcal{K} .
This alternative is represented by the Left postulate as $C \circ K$, and constructs contexts of the form $((\dots \circ C) \circ B) \circ A$.
- b. A ‘hole to the right’ $B \circ []$, embedded in some outer context \mathcal{K} .
This alternative is represented by the Right postulate as $K \circ B$, and constructs contexts of the form $A \circ (B \circ (C \circ \dots))$.
- c. The null context $[]$, the ground case for recursion. This alternative is represented by the Push and Pop postulates as 1. (26)

In (a) and (b) above, the type K represents the outer context \mathcal{K} .

By changing the structural postulates relating the continuation mode \odot to other modes, we can express different notions of what should count as an accessible context: adding postulates broadens the notion; removing postulates constrains the notion. Different notions of context can be mixed in the same grammar by using one \odot -like mode for each notion. This section and the next demonstrate the utility of this flexibility with three linguistic applications. We present these applications as separate amendments to the basic continuations analysis, but these amendments are fully compatible with each other.

7.1. RESTRICTORS

Having analyzed simple quantificational NPs like *everyone* and *someone* in Section 4, we now turn to quantifiers that take a common noun or a more complex restrictor constituent as argument, as in *most schools* or *every dean of a school*. We assume that common nouns denote functions from individuals (type np) to propositions or truth values (type s), but they cannot directly combine with NPs in the default mode (hence **Harvard school* is unacceptable). Thus we introduce a new mode \circ_n (the letter n being mnemonic for ‘noun’) and assign nouns such as *school* to the category $np \setminus_n s$.⁵ The new mode \circ_n is an internal one, so the direction of this slash is arbitrary, but it is chosen in analogy with the direction of the default-mode slash in the type $np \setminus s$ of an intransitive verb.

⁴ Borrowing notation for evaluation contexts in programming languages (Section 5.2), this notion of context can be expressed more succinctly by

$$\mathcal{K}[] ::= \mathcal{K}[[] \circ C] \mid \mathcal{K}[B \circ []] \mid [] .$$

⁵ We cannot reuse the continuation mode \odot for the \circ_n mode and assign *school* to the category $np \setminus s$. If we did, then the Right_n postulate that we are about to add in (32) would predict that the strings *Alice is a dean of Harvard* and **Harvard is an Alice dean of* are equal in grammaticality and

Unchanged cases:

- a. A ‘hole to the left’ $[] \circ C$, embedded in some outer context \mathcal{K} .
- b. A ‘hole to the right’ $B \circ []$, embedded in some outer context \mathcal{K} .
- c. The null context $[]$, the ground case for recursion.

Added cases:

- d. A ‘hole to the left’ $[] \circ_n C$, embedded in some outer context \mathcal{K} .
- e. A ‘hole to the right’ $B \circ_n []$, embedded in some outer context \mathcal{K} . (30)

by adding the following structural postulates alongside those in (10).

$$B \odot (C \circ_n K) \dashv\vdash (B \circ_n C) \odot K \quad (\text{Left}_n) \quad (31)$$

$$(B \circ_n C) \odot K \dashv\vdash C \odot (K \circ_n B) \quad (\text{Right}_n) \quad (32)$$

A welcome consequence of this revision is that a scope ambiguity between two quantifiers is predicted even when one is located within the other’s restrictor. This ambiguity is illustrated in the following sentence.

Every dean of a school griped. (33)

As Dalrymple et al. note (1999, Section 2.5.1), it can be tricky to account for linear scope in this sentence because there is no overt clausal boundary (“syntactic unit at the f-structure level”) under *every dean* where *a school* can take scope. Nevertheless, the ‘imaginary clausal boundary’ in the category $np \setminus_n s$ for nouns suffices for *a school* to take scope over. Figure 7 proves that *dean of a school* has the type $np \setminus_n s$, just like a common noun. The Right_n structural rule is crucial to this derivation. As indicated by the E deduction there, *a school* takes scope entirely within this complex noun, so inserting this proof into a derivation for (33) generates the linear-scope reading. The inverse-scope reading is also generated, without using the additional postulates in (31)–(32). The top of that derivation proves

$$(\text{every} \circ (\text{dean of} \circ np)) \circ \text{griped} \vdash s. \quad (34)$$

7.2. ISLANDS

If we add a new mode of combination to a grammar without also adding any postulate relating the new mode to the continuation mode \odot , then contexts would be unable to cross the new mode. In other words, the new mode would be an island.

For example, many people believe that quantificational NPs cannot take scope outside of a tensed clause. Tensed clauses can be made into scope islands in the standard TLG way using a pair of (external) unary modalities \diamond_i and \square_i^\dagger . For instance, if the lexical type of *thought* is $(\square_i^\dagger(np \setminus s))/s$, then *Someone thought everyone left*

sensitivity, also often behave asymmetrically with respect to the location of quantifiers. The continuations view of *in-situ* quantification captures quantifier ordering through the following refinement of the notion of context. This refinement recapitulates the use of continuations to study evaluation order in programming languages, sketched in Section 5.2 above. The same concept of evaluation order is applied again in Section 9 below to other natural language phenomena.

Figure 5 successfully derives inverse scope because *everyone* in object position can take as argument its context

$$(1 \circ \text{someone}) \circ \text{saw}, \quad (38)$$

or equivalently,

$$\text{someone} \circ (\text{saw} \circ []). \quad (39)$$

In general, the scope of one quantifier contains another just in case the latter appears in the context argument of the former. Imagine for the moment that we are studying a language that resembles English but mandates linear scope. To rule out inverse scope, we can refine our notion of context so as to rule out any quantifier linearly located to the left of the hole. Then (39), in which the quantifier *someone* occurs to the left of the hole, would no longer be a legitimate context, whereas

$$\text{Alice} \circ (\text{saw} \circ []) \quad (40)$$

and

$$[] \circ (\text{saw} \circ \text{everyone}) \quad (41)$$

would still be considered contexts. (The context (40) is used in Figures 2 and 3 to derive *Alice saw everyone*. The context (41) is used in (13) and Figure 4 to derive the linear-scope reading of *someone saw everyone*.)

We can implement this idea using a unary modality. Following programming-language terminology, we call an expression PURE if it contains no quantifier, or IMPURE otherwise. To distinguish pure expressions from impure ones, we tag the types of pure expressions with a unary modality \diamond . Any formula can be turned pure by embedding it under \diamond using the T postulate.

$$A \vdash \diamond A \quad (\text{T}) \quad (42)$$

The T postulate is analogous to QUOTATION or STAGING in programming languages, which turns executable code into static data. Two quotations can be concatenated using the K' postulate.

$$\diamond B \circ \diamond C \vdash \diamond(B \circ C) \quad (\text{K}') \quad (43)$$

Whereas the other rules introduced so far are double-sided, these rules are single-sided.⁶

We consider a derivation complete if it culminates in the type $\diamond s$, not just s . The type $\diamond s$ signifies a pure clause rather than a quantifier over clauses (propositions). By contrast, *everyone* and *someone* are impure: their type $\diamond s // (np \backslash \diamond s)$ is not surrounded by \diamond , though the propositions they quantify over and finally produce are pure (hence the \diamond in $\diamond s$).

To rule out inverse-scope contexts like (39), we modify our notion of context to require that the left branch of a \circ -fused constituent be pure before descending recursively into the right branch. That is, we revise (26) to

- Unchanged case:
 a. A ‘hole to the left’ $[] \circ C$, embedded in some outer context \mathcal{K} .
 Tightened case:
 b. A ‘hole to the right’ $\diamond B \circ []$, embedded in some outer context \mathcal{K} .
 Unchanged case:
 c. The null context $[]$, the ground case for recursion. (44)

In terms of structural postulates, we replace the Right postulate from (10) with a more specific instance.

$$B \odot (C \circ K) \dashv\vdash (B \circ C) \odot K \quad (\text{Left}) \quad (45)$$

$$(\diamond B \circ C) \odot K \dashv\vdash C \odot (K \circ \diamond B) \quad (\text{Right}) \quad (46)$$

With these changes to the grammar, only the linear-scope reading for *Someone saw everyone* (of type $\diamond s$) remains derivable. Figure 8 shows the derivation. It is shaped exactly like Figure 4, except the T and K’ postulates above are used after (13) to prove $\diamond np \circ (\diamond \text{saw} \circ np) \vdash \diamond s$. The inverse-scope derivation in Figure 5 is no longer valid because the Right postulate, restricted in (46), does not apply since *someone* is impure.

What we have just seen is that a linguistic preference for linear scope reflects a computational preference for left-to-right evaluation. Of course, in many languages (including English), inverse scope is available. That does not mean that order-of-evaluation effects are absent – it only means that, if they are present, they are more subtle. In the rest of this section, we examine one way to reintroduce inverse scope that gives rise to favorable empirical consequences in Section 9 below.

The basic idea is to treat inverse scope as MULTISTAGE PROGRAMMING (see Taha and Nielsen, 2003, and references therein). A multistage program is a program that generates another program (and then runs it, usually). The generating

⁶ These rules and their names are explained by Moot (2000, pages 39 and 156). Curiously, the present work seems to be the first linguistic application of K’.

$$\begin{array}{c}
\vdots (13) \\
\frac{np \circ (saw \circ np) \vdash s}{\diamond (np \circ (saw \circ np)) \vdash \diamond s} \diamond I \\
\frac{\diamond (np \circ (saw \circ np)) \vdash \diamond s}{\diamond np \circ \diamond (saw \circ np) \vdash \diamond s} K' \\
\frac{\diamond np \circ \diamond (saw \circ np) \vdash \diamond s}{\diamond np \circ (\diamond saw \circ \diamond np) \vdash \diamond s} K' \\
\frac{\diamond np \circ (\diamond saw \circ \diamond np) \vdash \diamond s}{\diamond np \circ (\diamond saw \circ np) \vdash \diamond s} T \\
\frac{\diamond np \circ (\diamond saw \circ np) \vdash \diamond s}{(\diamond np \circ (\diamond saw \circ np)) \circ 1 \vdash \diamond s} \text{Push} \\
\frac{(\diamond np \circ (\diamond saw \circ np)) \circ 1 \vdash \diamond s}{(\diamond saw \circ np) \circ (1 \circ \diamond np) \vdash \diamond s} \text{Right} \\
\frac{(\diamond saw \circ np) \circ (1 \circ \diamond np) \vdash \diamond s}{np \circ ((1 \circ \diamond np) \circ \diamond saw) \vdash \diamond s} \text{Right} \\
\frac{np \circ ((1 \circ \diamond np) \circ \diamond saw) \vdash \diamond s}{\text{everyone} \vdash \diamond s // (np \backslash \diamond s) \quad (1 \circ \diamond np) \circ \diamond saw \vdash np \backslash \diamond s} \backslash I \\
\frac{\text{everyone} \vdash \diamond s // (np \backslash \diamond s) \quad (1 \circ \diamond np) \circ \diamond saw \vdash np \backslash \diamond s}{\text{everyone} \circ ((1 \circ \diamond np) \circ \diamond saw) \vdash \diamond s} // E \\
\frac{\text{everyone} \circ ((1 \circ \diamond np) \circ \diamond saw) \vdash \diamond s}{(\diamond saw \circ \text{everyone}) \circ (1 \circ \diamond np) \vdash \diamond s} \text{Right} \\
\frac{(\diamond saw \circ \text{everyone}) \circ (1 \circ \diamond np) \vdash \diamond s}{(\diamond np \circ (\diamond saw \circ \text{everyone})) \circ 1 \vdash \diamond s} \text{Right} \\
\frac{(\diamond np \circ (\diamond saw \circ \text{everyone})) \circ 1 \vdash \diamond s}{(np \circ (\diamond saw \circ \text{everyone})) \circ 1 \vdash \diamond s} T \\
\frac{(np \circ (\diamond saw \circ \text{everyone})) \circ 1 \vdash \diamond s}{(np \circ (saw \circ \text{everyone})) \circ 1 \vdash \diamond s} T \\
\frac{(np \circ (saw \circ \text{everyone})) \circ 1 \vdash \diamond s}{np \circ ((saw \circ \text{everyone}) \circ 1) \vdash \diamond s} \text{Left} \\
\frac{np \circ ((saw \circ \text{everyone}) \circ 1) \vdash \diamond s}{\text{someone} \vdash \diamond s // (np \backslash \diamond s) \quad (saw \circ \text{everyone}) \circ 1 \vdash np \backslash \diamond s} \backslash I \\
\frac{\text{someone} \vdash \diamond s // (np \backslash \diamond s) \quad (saw \circ \text{everyone}) \circ 1 \vdash np \backslash \diamond s}{\text{someone} \circ ((saw \circ \text{everyone}) \circ 1) \vdash \diamond s} // E \\
\frac{\text{someone} \circ ((saw \circ \text{everyone}) \circ 1) \vdash \diamond s}{(\text{someone} \circ (saw \circ \text{everyone})) \circ 1 \vdash \diamond s} \text{Left} \\
\frac{(\text{someone} \circ (saw \circ \text{everyone})) \circ 1 \vdash \diamond s}{\text{someone} \circ (saw \circ \text{everyone}) \vdash \diamond s} \text{Pop}
\end{array}$$

Figure 8. Linear scope for *Someone saw everyone*, under left-to-right evaluation. Bernardi (2002, page 50) shows the $\diamond I$ rule used in this derivation (same as $\diamond R$ in the Gentzen presentation (Moortgat, 1997, Definition 4.16)), as well as the other natural-deduction rules for the unary operators \diamond and \square^l , namely $\diamond E$, $\square^l I$ (same as $\square^l R$), and $\square^l E$.

program is said to execute in an EARLIER or OUTER stage, and the generated program is said to execute in a LATER or INNER stage. More than two stages are also possible. Evaluation in each stage is ordered separately: later-stage code runs only after earlier-stage code generates it. Informally, then, we can treat the inverse-scope reading of *Someone saw everyone* as the following outer-stage program.

Run the program consisting of the word *someone*, the word *saw*, and everyone. (47)

Italics is significant here: The sentence above only uses one quantifier (*everyone*). It also mentions a word (*someone*) that is a quantifier, but does not use it. If the domain of people under discussion is Alice, Bob, and Carol, then (47) conjoins the meanings of the sentences *Someone saw Alice*, *Someone saw Bob*, and *Someone saw Carol*. These three sentences are the inner-stage programs.

A typical multistage programming language provides facilities for: creating programs (by quotation or staging); combining programs (by concatenation); and running programs (by UNQUOTATION or evaluation). Intuitively, a quoted value is

an inactive piece of program text that does not execute until the ‘wrapping’ \diamond is removed. Removing the wrapping turns inactive data into active code. We call this step “unquotation” despite the fact that “evaluation” or “eval” is the commonly used term for the same concept in the staged programming literature, to avoid confusion with the concept of evaluation order just discussed.

The T and K’ postulates in (42) and (43) are our facilities for quoting and concatenating programs, respectively. To run programs, we add the Unquote rule.⁷

$$\diamond\diamond_u A \vdash \diamond_u A \quad (\text{Unquote}) \tag{48}$$

This rule makes sense because the type of a quoted program is enclosed in a \diamond , and to run a program is to remove that \diamond . Although quotation applies freely (using T), unquotation does not. Rather, unquotation is restricted to types of the form $\diamond_u A$. Here \diamond_u is a unary modality that marks those types that are allowed as top-level types of quoted programs; this modality can be thought of as a feature checked by the Unquote rule. In particular, we hereafter take the clause type s to be shorthand for $\diamond_u s'$, so that we can derive it from the quoted clause type $\diamond s$ (shorthand for $\diamond\diamond_u s'$) using the Unquote rule. Here s' is an atomic formula distinct from s .⁸

Figure 9 shows that the inverse-scope reading of *Someone saw everyone* is once again derivable in the presence of Unquote. (Because s and $\diamond s$ entail each other in the presence of Unquote, we can restore the lexical types of *someone* and *everyone* from $\diamond s // (np \backslash \diamond s)$ back to $s // (np \backslash s)$.) As noted above, the Unquote rule is necessary to derive this reading if the Right rule is restricted as in (46). More precisely, in order for one quantifier to take inverse scope over another, the Unquote rule must apply to the clause produced by the narrower-scope quantifier (here *someone*).

The identity types (the 1’s) and the unary modes add considerable complexity to the basic analysis of *Someone saw everyone* given above in Figure 4. The payoff, as we shall see in the next section, is that these complications allow control over order of evaluation, which accounts for fairly subtle and complex linguistic phenomena.

⁷ In view of Moot’s PSPACE-completeness result for $NL\diamond_{\mathcal{R}-}$ (2002, Section 9.2), the K’ and Unquote postulates present computational difficulties for practical parsing applications because they are expanding. As it turns out, we can avoid expanding rules as in footnote 3: add a unary mode \diamond_i (where i stands for “impure”), and put \diamond_i around every subformula not surrounded by \diamond .

⁸ In this paper, s is the only type that can be unquoted, that is, the only type of the form $\diamond_u A$. A treatment of polarity licensing that is less simplistic than the one in Section 9.3 calls for multiple clause types that can be unquoted (Shan, 2004).

$$\begin{array}{c}
\vdots (13) \\
\frac{np \circ (saw \circ np) \vdash s}{(np \circ (saw \circ np)) \odot 1 \vdash s} \text{Push} \\
\frac{(np \circ (saw \circ np)) \odot 1 \vdash s}{np \odot ((saw \circ np) \circ 1) \vdash s} \text{Left} \\
\frac{np \odot ((saw \circ np) \circ 1) \vdash s}{(saw \circ np) \circ 1 \vdash np \backslash s} \backslash I \\
\frac{\text{someone} \vdash s // (np \backslash s) \quad (saw \circ np) \circ 1 \vdash np \backslash s}{\text{someone} \odot ((saw \circ np) \circ 1) \vdash s} // E \\
\frac{\text{someone} \odot ((saw \circ np) \circ 1) \vdash s}{(\text{someone} \circ (saw \circ np)) \odot 1 \vdash s} \text{Left} \\
\frac{(\text{someone} \circ (saw \circ np)) \odot 1 \vdash s}{\text{someone} \circ (saw \circ np) \vdash s} \text{Pop} \\
\frac{\text{someone} \circ (saw \circ np) \vdash s}{\diamond(\text{someone} \circ (saw \circ np)) \vdash s} \diamond I \\
\frac{\text{someone} \circ (saw \circ np) \vdash s \quad s \vdash s}{\diamond s \vdash s} \text{Unquote} \\
\frac{\diamond(\text{someone} \circ (saw \circ np)) \vdash s}{\diamond(\text{someone} \circ (saw \circ np)) \vdash s} \diamond E \\
\frac{\diamond(\text{someone} \circ (saw \circ np)) \vdash s}{\diamond \text{someone} \circ \diamond(saw \circ np) \vdash s} K' \\
\frac{\diamond \text{someone} \circ \diamond(saw \circ np) \vdash s}{\diamond \text{someone} \circ (\diamond saw \circ \diamond np) \vdash s} K' \\
\frac{\diamond \text{someone} \circ (\diamond saw \circ \diamond np) \vdash s}{\diamond \text{someone} \circ (\diamond saw \circ np) \vdash s} T \\
\frac{\diamond \text{someone} \circ (\diamond saw \circ np) \vdash s}{(\diamond \text{someone} \circ (\diamond saw \circ np)) \odot 1 \vdash s} \text{Push} \\
\frac{(\diamond \text{someone} \circ (\diamond saw \circ np)) \odot 1 \vdash s}{(\diamond saw \circ np) \odot (1 \circ \diamond \text{someone}) \vdash s} \text{Right} \\
\frac{(\diamond saw \circ np) \odot (1 \circ \diamond \text{someone}) \vdash s}{np \odot ((1 \circ \diamond \text{someone}) \circ \diamond saw) \vdash s} \text{Right} \\
\frac{np \odot ((1 \circ \diamond \text{someone}) \circ \diamond saw) \vdash s}{(1 \circ \diamond \text{someone}) \circ \diamond saw \vdash np \backslash s} \backslash I \\
\frac{\text{everyone} \vdash s // (np \backslash s) \quad (1 \circ \diamond \text{someone}) \circ \diamond saw \vdash np \backslash s}{\text{everyone} \odot ((1 \circ \diamond \text{someone}) \circ \diamond saw) \vdash s} // E \\
\frac{\text{everyone} \odot ((1 \circ \diamond \text{someone}) \circ \diamond saw) \vdash s}{(\diamond saw \circ \text{everyone}) \odot (1 \circ \diamond \text{someone}) \vdash s} \text{Right} \\
\frac{(\diamond saw \circ \text{everyone}) \odot (1 \circ \diamond \text{someone}) \vdash s}{(\diamond \text{someone} \circ (\diamond saw \circ \text{everyone})) \odot 1 \vdash s} \text{Right} \\
\frac{(\diamond \text{someone} \circ (\diamond saw \circ \text{everyone})) \odot 1 \vdash s}{\diamond \text{someone} \circ (\diamond saw \circ \text{everyone}) \vdash s} \text{Pop} \\
\frac{\diamond \text{someone} \circ (\diamond saw \circ \text{everyone}) \vdash s}{\text{someone} \circ (\diamond saw \circ \text{everyone}) \vdash s} T \\
\frac{\text{someone} \circ (\diamond saw \circ \text{everyone}) \vdash s}{\text{someone} \circ (saw \circ \text{everyone}) \vdash s} T
\end{array}$$

Figure 9. Inverse scope for *Someone saw everyone*, under left-to-right evaluation, using unquotation.

9. Beyond Quantification

In this section, we survey continuation-based analyses of several linguistic phenomena, and sketch how those analyses can be transliterated into TLG. The phenomena discussed are quite intricate, and we cannot hope to be comprehensive here – many additional details are available in the papers cited. Rather, our main purpose is to show how continuations offer explanatory insights previously unavailable in TLG, as well as improve the empirical coverage as compared to previous TLG accounts of the same phenomena.

9.1. BINDING AND CROSSOVER

Shan and Barker (2006) argue that order of evaluation provides a unified explanation for two distinct phenomena in English, weak crossover and superiority. Here we re-express that analysis in TLG, and compare our approach to Jäger’s (2005) TLG analysis of weak crossover. We offer the new explanatory insight that weak

crossover and superiority stem from the same underlying mechanism (namely, uniform default left-to-right evaluation). We offer improved empirical coverage in cases where pied-piping, *wh*-binding, and weak crossover interact.

Weak crossover is the name for the fact that a quantificational NP must usually precede any pronoun that it binds (Postal, 1971):

- a. Everyone_{*i*} loves his_{*i*} mother. (49)
 b. *His_{*i*} mother loves everyone_{*i*}.

We derive weak crossover from the assumption that people evaluate expressions from left to right by default, combined with the assumption that a quantifier must be evaluated before any pronoun that it binds.

To explain weak crossover, we must first implement binding in our grammar. We view binding dynamically, as the creation and use of discourse referents (Groenendijk and Stokhof, 1991; Heim, 1982; Kamp, 1981). Binding is an ideal arena in which to test the utility of continuations for describing natural language, for two reasons. First, continuations model side effects, and the storage and retrieval of values is a prototypical side effect in programming languages. Second, one of the key advantages of continuations is that they provide a general way to model multiple side effects and their interaction, as we model below how pronouns interact with quantifiers and *wh*-questions.

Many treatments of binding exist in the categorial-grammar literature (Dowty, 1992; Morrill, 2000; Moortgat, 1996; Hepple, 1990, 1992; Jacobson, 1999, 2000; Szabolcsi, 1989, 1992). Jäger (2005) provides a recent survey and discussion of approaches to anaphora in TLG. He proposes a new logical connective ‘|’ with special inference rules. Pronouns have category *np|np*; more generally, *X|Y* means ‘I function as something of category *X*, and need to be bound by something of category *Y*’. Jäger’s inference rules allow any expression of category *Y* to bind into the *X|Y* as long as the binder precedes the bound expression.

Two features of Jäger’s proposal are most relevant to us: resource duplication and linear order.

Resource duplication. In standard TLG, every linguistic resource is used exactly once, never duplicated. By contrast, binding uses a resource twice: once for the antecedent and then again for the bound pronoun. To deal with this mismatch, the inference rules for | incorporate a limited⁹ form of resource duplication.

⁹ Jäger details logical and linguistic reasons to limit resource duplication. Logically, Jäger’s system preserves the finite-reading property alongside cut elimination, decidability, and the subformula property. Linguistically, the sentence *The claim_{*i*} that someone saw everyone and its_{*i*} negation are both true* cannot possibly be true; to rule out the spurious reading where *someone saw everyone* scopes differently between the antecedent and the bound pronoun *it*, we must only duplicate formulas like *np*, not *someone* \circ (*saw* \circ *everyone*). Thus the | connective must not obey contravariance: $A \vdash B$ must not entail $np|B \vdash np|A$.

Linear order. Like us but unlike in most LF-based accounts of binding, Jäger recognizes the role of linear order. His system requires that the antecedent precede the pronoun it binds, and he gives empirical arguments that binding is sensitive to linear order. In particular, he correctly predicts weak crossover facts like those in (49).

In part because Jäger is primarily interested in resource duplication, he merely stipulates linear order: it follows from nothing, and the rules could just as easily have required that the binder follow its bound pronoun. We use continuations to provide a deeper explanation for the role of linear order in binding. Our account rules out weak crossover in quantificational binding by assuming that people evaluate expressions from left to right by default.

To help compare our approach with Jäger’s (2005), we also accomplish binding by means of a (single) inference rule that incorporates a limited form of resource duplication. However, since (unlike Jäger) we are working within a multimodal TLG, rather than introducing a new logical connective $|$, we introduce a new modality \circ_b for expressing binding relationships (‘b’ for ‘binding’):

$$\text{she} \vdash (np \backslash_b A) // (np \backslash A). \quad (50)$$

Conceptually, $np \backslash_b A$ is the category of an expression that would otherwise count as an A , but which contains a bindable pronoun. Thus the lexical entry for *she* turns any enclosing expression of category A into something of category $np \backslash_b A$. For instance, *John thought she left* would have category $np \backslash_b s$.

Our binding rule says that a subexpression np within a context \mathcal{K} can bind into the larger expression $\mathcal{K}[np]$.¹⁰

$$np \odot K \vdash np \circ_b (np \odot K) \quad (\text{Bind}) \quad (51)$$

To illustrate, Figure 10 derives *Everyone_i saw his_i mother*, in which the subexpression *everyone* binds into *[everyone] saw his mother*. On our analysis, the

¹⁰ Like Dalrymple et al. (1999), we limit resource duplication to np only – footnote 9 explains why. This form of limited resource duplication is much more primitive than Jäger’s, but suffices here because we focus on the role of linear order in binding and in other linguistic phenomena that do not duplicate resources. It would be nice to combine Jäger’s treatment of resource duplication with our treatment of linear order, because the former unifies NP anaphora with VP ellipsis while the latter unifies crossover with superiority, but we leave that to future work. One promising bridge between the two treatments is Morrill’s account of anaphora in terms of discontinuous constituency (2000): perhaps \odot is associative in nature with 1 as its left as well as right identity, and Jäger’s $|L$ and $|R$ rules can be rephrased as follows.

$$\frac{Y \vdash B \quad \Gamma[A \odot B] \vdash C}{\Gamma[(A|B) \odot Y] \vdash C} |L \quad \frac{A_n \odot \dots \odot A_1 \odot K \vdash B}{(A_n|C) \odot \dots \odot (A_1|C) \odot K \vdash B|C} |R$$

Here the type variable S ranges over all types. Conceptually, $np \setminus_{\gamma} S$ is the category of an S -question that seeks an np -answer. For example, a single-*wh* question has the type $np \setminus_{\gamma} s$, and a double-*wh* question has the type $np \setminus_{\gamma} np \setminus_{\gamma} s$. The lexical entries above specify that *wh*-words take scope *in situ*, so they by themselves generate *in-situ wh*-questions. To generate raised-*wh* questions, we add two postulates.¹¹

$$A \circ_{\gamma} (B \circ (C \odot K)) \vdash A \circ_{\gamma} (B \odot (C \circ K)) \quad (\text{Trace Left}) \quad (54)$$

$$A \circ_{\gamma} (C \circ (\diamond B \odot K)) \vdash A \circ_{\gamma} (C \odot (K \circ \diamond B)) \quad (\text{Trace Right}) \quad (55)$$

Figure 11 shows the basic usage scenario for these additions to the grammar. On one hand, the entire derivation culminates in the pied-piped raised-*wh* question *Whose mother did Alice see?*. On the other hand, the $\text{\textbackslash}E$ rule near the top concludes already with the *in-situ wh*-question *Alice saw whose mother?*. Of course, the result should be *Whose mother did Alice see?*, not the ungrammatical **Whose mother Alice saw?*; however, we ignore subject-auxiliary inversion and verb forms purely for the sake of keeping the derivations simpler.

Let us now examine the derivation of a double-*wh* question that abides by superiority, so that we can see what goes wrong in an attempt to violate superiority. Figure 12 derives the question *Who saw what?*, in which *who* is raised and *what* remains *in situ*. (We say that *who* is raised, even though it does not affect the string, because the derivation uses Trace Left near the end. We use *Who saw what?* as our example for simplicity.) In Figure 12, Trace Left applies to the raised *wh*-phrase *who* in the context $[] \text{ saw } \text{what}$. In other words, the raised *wh*-phrase takes scope over the rest of the sentence. Similarly, to derive the superiority violation **What did who see ___?* (or **what who saw*, to ignore subject-auxiliary inversion), *what* must take scope over the context *who saw []*. Just as in the previous section, taking scope over such a context requires using Right postulate twice, which introduces two \diamond s and necessitates Unquote. But questions cannot be unquoted: their types are of the form $A \setminus_{\gamma} S$, not $\diamond_u S$. Superiority is thus enforced.

For Jäger, *wh*-phrases have category $q/(np \uparrow s)$, where q is the category of questions and “ $np \uparrow s$ ” is an s with an *np*-gap. Because connecting gaps with their fillers uses a mechanism separate from binding, whatever might explain superiority in Jäger’s system will be independent of the explanation for weak crossover. On

¹¹ These two postulates are really the composition of the Left and Right postulates with a single new postulate

$$A \circ_{\gamma} (B \circ (_ \odot K)) \vdash A \circ_{\gamma} (B \odot K) \quad (\text{Trace}),$$

where $_$ is the empty string, in other words the identity for the default mode \circ . But as Section 6 explained, the default mode should not have an identity if we want *very* to have the type $(n/n)/(n/n)$. If multimodal TLG were to allow empty antecedents everywhere and use unary modalities to enforce non-emptiness when necessary, then we would be able to replace Trace Left and Trace Right with this Trace rule – an appealing prospect.

To a first approximation, a *wh*-phrase can bind a pronoun only if the trace of the *wh*-phrase precedes the pronoun. One explanation that has been popular at least since Reinhart (1983) is that the trace itself does the binding. On Jäger's system, the trace can indeed accomplish the binding in examples such as (56a) (see especially pages 124–125). The unacceptability of (56b) is also explained, since when the trace follows the pronoun, the prohibition against cataphora predicts the relative unacceptability of (56b).

But there are more complex examples which cast doubt on the assumption that it is the trace that is doing the binding:

- a. Whose_i father did John say ___ saw his_i mother?
 b. *Whose_i father did John say his_i mother saw ___? (57)

In each case, the trace is bound by the entire pied-piped *wh*-phrase *whose father*. That is, in each case, it is a father who is seeing or being seen. Yet the pronoun can be bound by the *wh*-word alone, as indicated by the indexation in (57a), suggesting that the *wh*-word must be able to bind the pronoun directly after all.

Interestingly, (57b) shows that binding still requires linear precedence of a sort: the pronoun can be bound by the *wh*-word *whose* only if the *wh*-trace precedes the pronoun. In our system, *whose* can bind a pronoun just as a quantificational NP does. But just as superiority is violated when an *in-situ wh*-phrase interferes between a raised *wh*-phrase and its trace, weak crossover is perpetrated when a bindable pronoun interferes between a raised *wh*-phrase and its trace. Put differently, (57b) constitutes a weak crossover violation because the corresponding *in-situ wh*-question *John said his_i mother saw whose_i father?* does. This complex pattern of interactions between pied-piping, binding of *wh*-traces, and binding of pronouns falls out from our unified account without additional stipulation.

This brief discussion hardly counts as a thorough comparison between our analysis and Jäger's; that would take far too much space here. Our only intention is to suggest that a continuation-based analysis of a complex phenomenon can hold its own against a robust previous TLG account, and to point out that our account provides a different pattern of explanation. In particular, we connect the linear order asymmetry of weak crossover with the computational concept of evaluation order; and we also provide an analysis on which weak crossover and superiority have distinct but closely parallel explanations. Thus continuations constitute a new and potentially valuable explanatory tool for TLG analyses.

9.3. POLARITY LICENSING

Polarity sensitivity (see, e.g., Ladusaw, 1979) has been a popular linguistic phenomenon to analyze in the type-logical (Bernardi, 2002; Bernardi and Moot, 2001), categorial (Dowty, 1994), and lexical-functional (Fry, 1997, 1999) approaches to grammar. The multitude of these analyses is in part due to the more explicit emphasis

that these traditions place on the syntax-semantics interface – be it in the form of the Curry-Howard isomorphism, Montague-style semantic rules, or linear logic as glue – and the fact that polarity sensitivity is a phenomenon that spans syntax and semantics.

On one hand, which polarity items are licensed or prohibited in a given linguistic environment depends, by and large, on semantic properties of that environment (Ladusaw, 1979; Krifka, 1995, *inter alia*). For example, to a first approximation, a negative polarity item (NPI) can occur only in a DOWNWARD-ENTAILING context, such as under the scope of a MONOTONICALLY DECREASING quantifier. A quantifier q , of type $(np \rightarrow s) \rightarrow s$, is monotonically decreasing just in case

$$\forall s_1 \cdot \forall s_2 (\forall x \cdot s_2(x) \Rightarrow s_1(x)) \Rightarrow q(s_1) \Rightarrow q(s_2). \quad (58)$$

Thus (59a) is acceptable because the scope of *no one* is downward-entailing, while (59b) and (59c) are unacceptable.

- a. No one saw anyone.
- b. *Everyone saw anyone.
- c. *Alice saw anyone. (59)

To account for these contrasts, Fry (1997, 1999), Bernardi (2002), and Bernardi and Moot (2001) all split the clause type s into several types, related by subtyping. Fry (1997, 1999) distinguishes between s and $\ell \multimap (s \otimes \ell)$ in linear logic, where ℓ stands for polarity licensing as a grammatical resource. Analogously, Bernardi (2002) and Bernardi and Moot (2001) distinguish between different unary modalities applied to s .

A simplistic version of Bernardi’s analysis is to distinguish between the clause types s (‘neutral clause’) and $\Box_p \diamond_p s$ (‘negative clause’). Here \diamond_p is a new unary

$$\begin{array}{c} \vdots (13) \\ \frac{np \circ (\text{saw} \circ np) \vdash s}{\diamond_p (np \circ (\text{saw} \circ np)) \vdash \diamond_p s} \diamond_p I \\ \frac{\diamond_p (np \circ (\text{saw} \circ np)) \vdash \diamond_p s}{np \circ (\text{saw} \circ np) \vdash s^-} \Box_p I \\ \frac{np \circ (\text{saw} \circ np) \vdash s^-}{(np \circ (\text{saw} \circ np)) \odot 1 \vdash s^-} \text{Push} \\ \frac{(np \circ (\text{saw} \circ np)) \odot 1 \vdash s^-}{(\text{saw} \circ np) \odot (1 \circ np) \vdash s^-} \text{Right} \\ \frac{(\text{saw} \circ np) \odot (1 \circ np) \vdash s^-}{np \odot ((1 \circ np) \circ \text{saw}) \vdash s^-} \text{Right} \\ \frac{np \odot ((1 \circ np) \circ \text{saw}) \vdash s^-}{\text{anyone} \vdash s^- \text{ // } (np \text{ // } s^-)} \text{ // I} \\ \frac{\text{anyone} \vdash s^- \text{ // } (np \text{ // } s^-)}{\text{anyone} \odot ((1 \circ np) \circ \text{saw}) \vdash s^-} \text{ // E} \\ \frac{\text{anyone} \odot ((1 \circ np) \circ \text{saw}) \vdash s^-}{(\text{saw} \circ \text{anyone}) \odot (1 \circ np) \vdash s^-} \text{Right} \\ \frac{(\text{saw} \circ \text{anyone}) \odot (1 \circ np) \vdash s^-}{(np \circ (\text{saw} \circ \text{anyone})) \odot 1 \vdash s^-} \text{Right} \\ \frac{(np \circ (\text{saw} \circ \text{anyone})) \odot 1 \vdash s^-}{np \circ (\text{saw} \circ \text{anyone}) \vdash s^-} \text{Pop} \end{array}$$

Figure 13. The negative clause (incomplete sentence) *np saw anyone*.

mode (the letter p stands for “polarity”), and we abbreviate $\Box_p^! \Diamond_p s$ to s^- . We choose the type $\Box_p^! \Diamond_p s$ so that $s \vdash s^-$ is a theorem in multimodal TLG.

$$\frac{\frac{\text{Id}}{\Diamond_p s \vdash \Diamond_p s}}{s \vdash \Box_p^! \Diamond_p s} \Box_p^! I \quad (60)$$

Splitting the clause type like this helps us account for polarity sensitivity, because we can now assign different types to different quantifiers in the lexicon:

$$\text{everyone} \vdash s // (np \backslash s), \quad \text{no one} \vdash s // (np \backslash s^-), \quad \text{anyone} \vdash s^- // (np \backslash s^-). \quad (61)$$

The type of *everyone* is unchanged: it takes scope over a neutral clause to form a neutral clause. The types of *no one* and *anyone* involve the newly introduced s^- : they both take scope over a negative clause, but *no one* forms a neutral clause whereas *anyone* forms a negative clause. As (60) shows, a neutral clause can be converted to a negative clause. Thus, as Figure 13 shows, *anyone* can take scope in $np \circ (\text{saw} \circ \text{anyone})$ to form a negative clause s^- , even though the verb *saw* produces a neutral clause s initially. As before, we consider a derivation complete if it culminates in the type s , not s^- . The fact that $np \circ (\text{saw} \circ \text{anyone})$ only has the type s^- (as derived above) and not s predicts that a clause like **Alice saw anyone* (59c) is not a grammatical sentence by itself. Moreover, because there is no way to convert a negative clause to a neutral clause, *everyone* cannot take scope over a negative clause, so **Everyone saw anyone* (59b) is ruled out as well. However, *no one* (unlike *Alice* or *everyone*) can take scope over *anyone* to form a complete (neutral) clause. Thus even our simplistic rendering of Bernardi’s account predicts correctly that *No one saw anyone* (59a) has a linear-scope reading (but no inverse-scope reading). We also predict correctly that the sentence

$$\text{No one introduced everyone to anyone.} \quad (62)$$

has no linear-scope reading, and that it does have an interpretation on which *no one* scopes over *anyone*, then *everyone*. (Kroch (1974), Linebarger (1980), and Szabolcsi (2004) discuss such INTERVENTION cases.)

On the other hand, a restriction on surface syntactic form, such as that imposed by polarity sensitivity, is by definition a matter of syntax. Besides, there are syntactic restrictions on the configuration relating the licenser to the licensee. For example, (59a) above is acceptable – *no one* manages to license *anyone* – but (63) below is not. As the contrasts in (64)–(66) further illustrate, the licenser usually needs to precede the licensee: (65) exonerates c-command; (66) exonerates subject-object asymmetry.

$$\text{*Anyone saw no one.} \quad (63)$$

a. Alice didn't visit anyone. (64)

b. *Anyone didn't visit Alice.

a. Nobody's mother saw anybody's father. (65)

b. *Anybody's mother saw nobody's father.

a. I gave nothing to anybody. (66)

b. *I gave anything to nobody.

c. I gave nobody anything.

d. *I gave anybody nothing.

These and other examples show that the syntactic relations allowed between licenser and licensee for polarity purposes are similar to those allowed between antecedent and pronoun for binding purposes. Because Fry, Bernardi, and Bernardi and Moot focus on quantification and scope, they easily characterize how *no* must scope over *any* in order to license it, but they leave it a mystery why *no* must precede *any* in order to license it. In particular, they wrongly accept all of (63)–(66). Ladusaw (1979) noted this mystery in his Inherent Scope Condition (ISC): “If the NPI is clausemate with the trigger, the trigger must precede” (Section 4.4). He went on to speculate that this left-right requirement is related to quantifier scope and sentence processing, as we claim:

I do not at this point see how to make this part of the ISC follow from any other semantic principle. This may be because the left-right restriction, like the left-right rule for unmarked scope relations, should be made to follow from the syntactic and semantic processing of sentences (Section 9.2)

Likewise, Fry (1999, Section 8.2) faults current accounts of polarity sensitivity for ignoring linear order.

Continuations provide precisely the missing link between linear order and quantifier scope. Recall from Section 8 that inverse scope can be generated, even under a regime of left-to-right evaluation, using the Unquote rule in (48). The crucial step, as illustrated in Figure 9, is to unquote the clause produced by the narrower-scope quantifier. In that example, *everyone* can take inverse scope over *someone*, because *someone* produces a neutral clause s . A neutral clause can be unquoted because its type s is shorthand for $\diamond_u s'$, which is enclosed in \diamond_u . By contrast, a negative clause cannot be unquoted because its type s^- is shorthand for $\square_p^! \diamond_p \diamond_u s'$, which is not enclosed in \diamond_u . Given that *anyone* produces s^- , then, inverse scope over *anyone* is impossible. This prediction correctly rules out the unacceptable examples in (63)–(66) while leaving (62) available.

This explanation for linear-order effects in polarity licensing is further developed elsewhere (Shan, 2004). We are not aware of any other account of polarity sensitivity that unifies its syntactic properties with weak crossover as we do using left-to-right evaluation.

9.4. REVERSING EVALUATION ORDER

One way to see that our implementation of evaluation order unifies linear order effects in weak crossover, superiority, and polarity is to impose right-to-left evaluation and see what happens. Because our Left and Right postulates embody left-to-right evaluation order, we can reverse default evaluation order while leaving all other aspects of the system unchanged.

More specifically, suppose that we replace the Left and Right postulates in (45)–(46) with

$$B \odot (\diamond C \circ K) \dashv\vdash (B \circ \diamond C) \odot K \quad (\text{Left}), \quad (67)$$

$$(B \circ C) \odot K \dashv\vdash C \odot (K \circ B) \quad (\text{Right}), \quad (68)$$

and the Trace Left and Trace Right postulates in (54)–(55) with¹²

$$A \circ_? (B \circ (\diamond C \odot K)) \vdash A \circ_? (B \odot (\diamond C \circ K)) \quad (\text{Trace Left}), \quad (69)$$

$$A \circ_? (C \circ (B \odot K)) \vdash A \circ_? (C \odot (K \circ B)) \quad (\text{Trace Right}). \quad (70)$$

We can still derive linear and inverse scope for quantifiers, but the predictions concerning the weak crossover judgments in (49), the superiority judgments in (52), and the polarity judgments in (63)–(66) reverse: binders must follow any pronouns they bind, *wh*-traces must follow any additional *wh*-phrases, and polarity triggers must follow the polarity items they license.

10. Conclusions

Let's take stock. Our core proposal implements the full power of the quantificational type constructor q using only a single new mode (\odot) along with two structural postulates, Left and Right. (If empty antecedents are forbidden, we also need to simulate Push and Pop.) We provided a geometric interpretation of types to explain how this parsimonious reduction of q constitutes an implementation of continuations: under the geometric interpretation, the continuation mode decomposes types into a subexpression in the foreground and a remainder in the background, where the remainder is interpreted as a context.

We then (Section 8) imposed left-to-right evaluation order by stipulating that a quantificational element could only take scope over an expression to its left if that expression had no side effects, as indicated by the presence of the unary modality \diamond . In order to restore inverse scope, we added an Unquote rule to remove the \diamond , though only for expressions in category s . Under the analogy with computation, the evaluation of an expression is delayed for as long as it is under the quotation

¹² As one would expect from footnote 11.

modality \diamond . By quoting a leftmost quantifier and then unquoting it later, we can delay the evaluation of the quantifier and let it take narrow (hence inverse) scope with respect to some following quantifier.

We proposed analyses on which pronominal binding (Section 9.1), the binding of a *wh*-trace (Section 9.2), and licensing of a negative polarity item (Section 9.3) were all accomplished via side-effects. Because bindable pronouns have their own side-effects, their presence disrupted the unquotation strategy, so that the analysis automatically predicts that a quantifier can take scope over a pronoun to its left but cannot bind it. That is, the analysis correctly allows inverse scope but not weak crossover violations. Similarly, the analysis predicts that a *wh*-phrase can bind its trace only when no other *wh*-phrase intervenes. That is, the analysis correctly allows multiple *wh*-questions but not superiority violations. Finally, the analysis automatically correctly requires a negative polarity licenser to precede its licensee.

The theoretical parsimony and empirical robustness of these analyses show how continuations offer new and valuable tools for multimodal TLG writers. In particular, continuation-based analyses offer new insights into quantification, binding, multiple *wh*-questions, and negative polarity.

Acknowledgements

We would not have undertaken this project without the encouragement of Bob Carpenter and David Dowty. Nor would we have gotten anywhere without many patient explanations from Philippe de Groote, the topological insights of Dylan Thurston, and Richard Moot's theorem prover for multimodal TLG analysis, made accessible through Dick Oehrle's web interface. We also gratefully acknowledge comments and suggestions from David Dowty, Gerhard Jäger, and Richard Moot. The second author performed this research while at Harvard University, supported by the United States National Science Foundation Grant BCS-0236592.

References

- Abbott, M., Altenkirch, T., Ghani, G.N., and McBride, C., 2003, "Derivatives of containers," in *TLCA 2003: Proceedings of the 6th International Conference on Typed Lambda Calculi and Applications*, Martin Hofmann, ed., pp. 16–30. Lecture Notes in Computer Science. Vol. 2701, Berlin: Springer-Verlag.
- Aoun, J. and Li, Y.-h. A., 1993, *Syntax of Scope*. Cambridge: MIT Press.
- Barendregt, H.P., 1981, *The Lambda Calculus: Its Syntax and Semantics*. Amsterdam: Elsevier Science.
- Barker, C., 2002, "Continuations and the nature of quantification," *Natural Language Semantics* **10**(3), 211–242.
- Barker, C., 2004, "Continuations in natural language," in *CW'04: Proceedings of the 4th ACM SIG-PLAN Continuations Workshop*, Hayo Thielecke, ed., pp. 1–11. Tech. Rep. CSR-04-1, School of Computer Science, University of Birmingham.

- Barker, C., 2005, "Remark on Jacobson 1999: Crossover as a local constraint," *Linguistics and Philosophy* **28**(4), 447–472.
- Belnap, N.D., Jr. 1982, "Display logic," *Journal of Philosophical Logic* **11**(4), 375–417.
- Bernardi, R., 2002, Reasoning with polarity in categorial type logic. Ph.D. thesis, Utrecht Institute of Linguistics (OTS), Utrecht University.
- Bernardi, R., 2003, CTL: *In situ* binding. http://www.let.uu.nl/~Raffaella.Bernardi/personal/q_solutions.pdf.
- Bernardi, R., and Moot, R. 2001, "Generalized quantifiers in declarative and interrogative sentences," *Journal of Language and Computation* **1**(3), 1–19.
- Chomsky, N., 1973, Conditions on transformations. in *A Festschrift for Morris Halle*, Stephen Anderson and Paul Kiparsky, ed., pp. 232–286. New York: Holt, Rinehart and Winston.
- Dalrymple, M., ed. 1999, *Semantics and Syntax in Lexical Functional Grammar: The Resource Logic Approach*. Cambridge: MIT Press.
- Dalrymple, M., Lamping, J., Pereira, F., and Saraswat, V.A., 1999, "Quantification, anaphora, and intensionality," in Dalrymple (1999), Chap. 2, pp. 39–89.
- Danvy, O. and Filinski, A., 1989, "A functional abstraction of typed contexts," Tech. Rep. Vol. 89/12, DIKU, University of Copenhagen, Denmark. <http://www.daimi.au.dk/~danvy/Papers/fatc.ps.gz>.
- Dowty, D., 1992, 'Variable-free' syntax, variable-binding syntax, the natural deduction Lambek calculus, and the crossover constraint," in *Proceedings of the 11th West Coast Conference on Formal Linguistics*, Jonathan Mead, ed., Stanford, CA: Center for the Study of Language and Information.
- Dowty, D.R., 1994, "The role of negative polarity and concord marking in natural language reasoning," in *Proceedings from Semantics and Linguistic Theory IV*, Mandy Harvey and Lynn Santelmann, ed., Ithaca: Cornell University Press.
- Felleisen, M., 1987, The calculi of λ_v -CS conversion: A syntactic theory of control and state in imperative higher-order programming languages. Ph.D. thesis, Computer Science Department, Indiana University. Also as Tech. Rep. 226.
- Felleisen, M., 1988, "The theory and practice of first-class prompts," in *POPL '88: Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, pp. 180–190. New York: ACM Press.
- Fischer, M.J., 1972, "Lambda-calculus schemata," in *Proceedings of ACM Conference on Proving Assertions About Programs*, vol. **7**(1) of *ACM SIG-PLAN Notices*, pp. 104–109. New York: ACM Press. Also ACM SIG-ACT News 14.
- Fischer, M.J., 1993, "Lambda-calculus schemata," *Lisp and Symbolic Computation* **6**(3–4), 259–288.
- Fry, J., 1997, "Negative polarity licensing at the syntax-semantics interface," in *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, Philip R. Cohen and Wolfgang Wahlster, ed., pp. 144–150. San Francisco: Morgan Kaufmann.
- Fry, J., 1999, "Proof nets and negative polarity licensing," in Dalrymple (1999), Chap. 3, pp. 91–116.
- Gentzen, G., 1933, Über das Verhältnis zwischen intuitionistischer und klassischer Arithmetik. Galley proof, *Mathematische Annalen*. English translation (Gentzen, 1969c).
- Gentzen, G., 1935, Untersuchungen über das logische Schließen. *Mathematische Zeitschrift* **39**, 176–210, 405–431. English translation Gentzen (1969b).
- Gentzen, G., 1969a, *The Collected Papers of Gerhard Gentzen*, M.E. Szabo, ed., Amsterdam: North-Holland.
- Gentzen, G., 1969b, "Investigations into logical deduction," in Gentzen (1969a), Chap. 3, pp. 68–131.
- Gentzen, G., 1969c, "On the relation between intuitionist and classical arithmetic," in Gentzen (1969a), Chap. 2, pp. 53–67.
- Girard, J.-Y., Taylor, P., and Lafont, Y., 1989, *Proofs and Types*. Cambridge: Cambridge University Press.

- Gödel, K., 1933, "Zur intuitionistischen Arithmetik und Zahlentheorie," *Ergebnisse eines mathematischen Kolloquiums* 4: 34–38. English translation (Gödel, 1965).
- Gödel, K., 1965, "On intuitionistic arithmetic and number theory," in *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvability Problems and Computable Functions*, Martin Davis, ed., pp. 75–81. Hewlett, NY: Raven Press.
- Griffin, T.G., 1990, "A formulae-as-types notion of control," in *POPL '90: Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, pp. 47–58. New York: ACM Press.
- Groenendijk, J. and Stokhof, M., 1991, "Dynamic predicate logic," *Linguistics and Philosophy* 14(1), 39–100.
- de Groote, P., 2001, "Type raising, continuations, and classical logic," in *Proceedings of the 13th Amsterdam Colloquium*, Robert van Rooij and Martin Stokhof, ed., pp. 97–101. Institute for Logic, Language and Computation, Universiteit van Amsterdam.
- Heim, I., 1982, "The semantics of definite and indefinite noun phrases," Ph.D. thesis, Department of Linguistics, University of Massachusetts.
- Hepple, M., 1990, "The grammar and processing of order and dependency: A categorial approach," Ph.D. thesis, Centre for Cognitive Science, University of Edinburgh.
- Hepple, M., 1992, "Command and domain constraints in a categorial theory of binding," in *Proceedings of the 8th Amsterdam Colloquium*, Paul Dekker and Martin Stokhof, ed., pp. 253–270. Institute for Logic, Language and Computation, Universiteit van Amsterdam.
- Hinze, R. and Jeurig, J., 2001, "Weaving a web," *Journal of Functional Programming* 11(6), 681–689.
- Howard, W.A., 1980, "The formulae-as-types notion of construction," in *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Jonathan P. Seldin and J. Roger Hindley, ed., pp. 479–490. San Diego, CA: Academic Press.
- Huang, C.-T.J., 1982, "Logical relations in Chinese and the theory of grammar," Ph.D. thesis, Department of Linguistics and Philosophy, Massachusetts Institute of Technology.
- Huet, G., 1997, "The zipper," *Journal of Functional Programming* 7(5), 549–554.
- Jacobson, P., 1999, "Towards a variable-free semantics," *Linguistics and Philosophy* 22(2), 117–184.
- Jacobson, P., 2000, "Paycheck pronouns, Bach-Peters sentences, and variable-free semantics," *Natural Language Semantics* 8(2), 77–155.
- Jäger, G., 2005, *Anaphora and Type Logical Grammar*, Berlin: Springer-Verlag.
- Kamp, H., 1981, "A theory of truth and semantic representation," in *Formal Methods in the Study of Language: Proceedings of the 3rd Amsterdam Colloquium*, Jeroen A.G. Groenendijk, Theo M.V. Janssen, and Martin B.J. Stokhof, ed., pp. 277–322. Amsterdam: Mathematisch Centrum.
- Kelsey, R., Clinger, W.D., Rees, J., Abelson, H., Dybvig, R.K., Haynes, C.T., Rozas, G.J., Adams, N.I. IV, Friedman, DP., Kohlbecker, E., Steele, G.L., Bartley, D.H., Halstead, R., Oxley, D., Sussman, G.J., Brooks, G., Hanson, C., Pitman, K.M., and Wand, M., 1998, "Revised⁵ report on the algorithmic language Scheme," *Higher-Order and Symbolic Computation* 11(1), 7–105. Also as *ACM SIGPLAN Notices* 33(9):26–76.
- Kolmogorov, A., 1925, "O principe tertium non datur," *Mathematicheskij sbornik* 32, 646–667. English translation (Kolmogorov, 1967).
- Kolmogorov, A., 1967, "On the principle of excluded middle," in *From Frege to Gödel: A source Book in Mathematical Logic, 1879–1931*, Jean van Heijenoort, ed., pp. 414–437. Cambridge: Harvard University Press.
- Krifka, M., 1995, "The semantics and pragmatics of polarity items," *Linguistic Analysis* 25, 209–257.
- Kroch, A.S., 1974, "The semantics of scope in English," Ph.D. thesis, Massachusetts Institute of Technology. Reprinted by New York: Garland, 1979.

- Kuno, S. and Robinson, J.J. 1972, "Multiple wh questions," *Linguistic Inquiry* **3**, 463–487.
- Ladusaw, W.A., 1979, "Polarity sensitivity as inherent scope relations," Ph.D. thesis, Department of Linguistics, University of Massachusetts. Reprinted by New York: Garland, 1980.
- Lambek, J., 1958, "The mathematics of sentence structure," *American Mathematical Monthly* **65**(3), 154–170.
- Landin, P.J., 1966, "The next 700 programming languages," *Communications of the ACM* **9**(3), 157–166.
- Landin, P.J., 1998, "A generalization of jumps and labels," *Higher-Order and Symbolic Computation* **11**(2), 125–143.
- Linebarger, M.C., 1980, "The grammar of negative polarity," Ph.D. thesis, Massachusetts Institute of Technology.
- Meyer, A.R. and Wand, M., 1985, "Continuation semantics in typed lambda-calculi (summary)," in *Logics of Programs*, Rohit Parikh, ed., pp. 219–224. Lecture Notes in Computer Science Vol. 193, Berlin: Springer-Verlag.
- Moortgat, M., 1988, *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus*. Dordrecht: Foris.
- Moortgat, M., 1995, "In situ binding: A modal analysis," in *Proceedings of the 10th Amsterdam Colloquium*, Paul Dekker and Martin Stokhof, ed., pp. 539–549. Institute for Logic, Language and Computation, Universiteit van Amsterdam.
- Moortgat, M., 1996, "Generalized quantification and discontinuous type constructors," in *Discontinuous Constituency*, Harry C. Bunt and Arthur van Horck, ed., pp. 181–207. Berlin: Mouton de Gruyter.
- Moortgat, M., 1997, "Categorial type logics," in *Handbook of Logic and Language*, Johan F.A.K. van Benthem and Alice G.B. ter Meulen, ed., Chap. 2. Amsterdam: Elsevier Science.
- Moortgat, M., 2000, *Computational Semantics: Lab Sessions*. <http://www.let.uu.nl/~Michael.Moortgat/personal/Courses/cslab2000s.pdf>.
- Moot, R., 2002, "Proof nets for linguistic analysis," Ph.D. thesis, Utrecht Institute of Linguistics (OTS), Utrecht University.
- Morrill, G.V., 1994, *Type Logical Grammar: Categorial Logic of Signs*. Dordrecht: Kluwer.
- Morrill, G.V., 2000, "Type-logical anaphora," Report de Recerca Vol. LSI-00-77-R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya.
- Murthy, C.R., 1990, "Extracting constructive content from classical proofs," Ph.D. thesis, Department of Computer Science, Cornell University. Also as Tech. Rep. TR90-1151.
- Papasprou, N.S., 1998, "Denotational semantics of evaluation order in expressions with side effects," in *Recent Advances in Information Science and Technology: 2nd Part of the Proceedings of the 2nd IMACS International Conference on Circuits, Systems and Computers*, Nikos E. Mastorakis, ed., pp. 87–94. Singapore: World Scientific.
- Plotkin, G.D., 1975, "Call-by-name, call-by-value and the λ -calculus," *Theoretical Computer Science* **1**(2), 125–159.
- Postal, P.M., 1971, *Cross-Over Phenomena*. New York: Holt, Rinehart and Winston.
- Reinhart, T., 1983, *Anaphora and Semantic Interpretation*. London: Croom Helm.
- Restall, G., 2000, *An Introduction to Substructural Logics*. London: Routledge.
- Reynolds, J.C., 1998, "Definitional interpreters for higher-order programming languages," *Higher-Order and Symbolic Computation* **11**(4), 363–397.
- Shan, C.-c., 2002, "A continuation semantics of interrogatives that accounts for Baker's ambiguity," in *Proceedings from Semantics and Linguistic Theory XII*, Brendan Jackson, ed., pp. 246–265. Ithaca: Cornell University Press.
- Shan, C.-c., 2004, "Polarity sensitivity and evaluation order in type-logical grammar," in *Proceedings of the 2004 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, Susan Dumais, Daniel Marcu,

- and Salim Roukos, ed., Vol. 2, pp. 129–132. Somerset, NJ: Association for Computational Linguistics.
- Shan, C.-c. and Barker, C., 2006, “Explaining crossover and superiority as left-to-right evaluation,” *Linguistics and Philosophy* **29**(1), 91–134.
- Steedman, M.J., 2000, *The Syntactic Process*. Cambridge: MIT Press.
- Szabolcsi, A., 1989, “Bound variables in syntax (are there any?),” in *Semantics and Contextual Expression*, Renate Bartsch, Johan F.A.K. van Benthem, and Peter van Emde Boas, ed., pp. 295–318. Dordrecht: Foris.
- Szabolcsi, A., 1992, “Combinatory grammar and projection from the lexicon,” in *Lexical Matters*, Ivan A. Sag and Anna Szabolcsi, ed., pp. 241–269. CSLI Lecture Notes Vol. 24, Stanford, CA: Center for the Study of Language and Information.
- Szabolcsi, A., 1997, “Strategies for scope taking,” in *Ways of Scope Taking*, Anna Szabolcsi, ed., Chap. 4, pp. 109–154. Dordrecht: Kluwer.
- Szabolcsi, A., 2004, “Positive polarity–negative polarity,” *Natural Language and Linguistic Theory* **22**(2), 409–452.
- Taha, W. and Nielsen, M.F., 2003, “Environment classifiers,” in *POPL '03: Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, pp. 26–37. New York: ACM Press.