The Logic of Scope (extended abstract)

Chris Barker
New York University, New York NY 10003, USA, chris.barker@nyu.edu,
WWW home page: http://files.nyu.edu/cb125/public/

Abstract. What is the logic of scope? By “scope”, I mean scope-taking in natural languages such as English, as illustrated by the sentence Ann saw everyone. In this sentence, the quantifier denoted by everyone takes scope over the rest of the sentence, that is, it takes the denotation of the rest of the sentence as its semantic argument: everyone(\(\lambda x.\text{saw}(x)(\text{ann})\)). The answer I will give here will be to provide a bi-modal substructural logic whose two implications are related by a structural postulate. This postulate can be interpreted as constituting a kind of lambda-abstraction over structures, where the abstracted structures are interpreted as delimited continuations. I give soundness and completeness results, as well as cut elimination and decidability. I also compare the logic to the standard technique of Quantifier Raising, and mention applications to scope ambiguity and parasitic scope.

Keywords: scope, continuations, substructural logic, Quantifier Raising, parasitic scope, natural language quantification

1 What is the logic of scope?

Just as we might ask “What is the logic of negation?”, we might ask “What is the logic of scope?” And just as the first question has many answers, so too will the second. The answer I will give here will take the form of a substructural logic containing a single structural postulate. I will suggest this logic characterizes a kind of scope-taking that has applications in the analysis of natural language.

1.1 Scope in natural language

Many natural languages have scope-taking expressions, including English:

(1) Ann saw everyone.

In (1), the denotation of the quantifier everyone takes the rest of the sentence in which it occurs as its semantic argument. That is, the denotation of the sentence as a whole is given by everyone(\(\lambda x.\text{saw}(x)(\text{ann})\)).

There are three important properties of scope-taking in natural language that I will discuss here: unbounded scope displacement, embedded scope-taking, and scope ambiguity (see [3] for a more complete discussion).

(2) Ann saw the mother of everyone’s lawyer.
In (2), despite being embedded inside of two layers of possessive constructions, the quantifier still takes scope over the entire sentence. In general, there is no upper limit to the structural distance over which an expression can take scope.

(3) a. Bill thinks [Ann saw everyone].
    b. \( \text{thinks}(\forall x.\text{saw}(x)(\text{ann}))(\text{bill}) \)

In (3a), the quantifier takes scope only over the [bracketed] embedded clause \textit{Ann saw everyone}, which is a proper subpart of the complete sentence. The fact that scopal elements can take embedded scope is what makes undelimited continuations unsuited to modeling scope (see chapter 18 of [4] for discussion); delimited continuations are a better fit.

(4) a. Someone loves everyone.
    b. \( \exists x \forall y.\text{loves}(y)(x) \)
    c. \( \forall y \exists x.\text{loves}(y)(x) \)

Scope ambiguity can arise when there is more than one quantifier in the sentence. There can in general be as many as \( n! \) distinct denotations, where \( n \) is the number of quantifiers.

1.2 Quantifier Raising

By far the dominant way to think about scope-taking is Quantifier Raising (QR), as discussed in detail in [7]. Quantifier Raising accounts for unbounded scope displacement, embedded scope-taking, and scope ambiguity.

From a logical point of view, Quantifier Raising bears a certain resemblance to a structural postulate: Quantifier Raising reconfigures a logical structure by moving the quantifier to adjoin to its scope domain, placing a variable in the original position of the quantifier, and abstracting over the variable at the level of the scope domain.

\[
[\text{Ann [called everyone]}] \xrightarrow{\text{QR}} [\text{everyone}(\lambda x [\text{Ann [called x]])}]
\]

Here, the scope domain of \textit{everyone} is the entire clause. Because the QR operation can target embedded S nodes, embedded scope falls out naturally.

QR easily accounts for scope ambiguity by allowing QR to target quantifiers in any order.

Linear scoping :

\[
[\text{someone [called everyone]}]
\xrightarrow{\text{QR}} [\text{everyone}(\lambda x [\text{someone [called x]])}]
\xrightarrow{\text{QR}} [\text{someone}(\lambda y [\text{everyone}(\lambda x [\text{called x}]))])
\]

Inverse scoping :

\[
[\text{someone [called everyone]}]
\xrightarrow{\text{QR}} [\text{someone}(\lambda y [\text{called everyone}])]
\xrightarrow{\text{QR}} [\text{everyone}(\lambda x [\text{someone}(\lambda y [\text{called x}]))])
\]
Raising the direct object first and then the subject gives linear scope, and raising the subject first and then the direct object gives inverse scope.

So far, so good. We can imagine a substructural logic along the lines of the non-associative Lambek calculus with a structural postulate that implements Quantifier Raising. What would remain unclear is what sort of (logical) semantics would characterize the QR logic. One way to view the results reported in this paper is as giving Quantifier Raising a treatment that resolves these logical questions. Further discussion of Quantifier Raising appears below after introducing the full logic.

1.3 The \( q \) type constructor

[10] extends Lambek grammar with a type constructor \( q \) (‘\( q \)’ for ‘quantification’) which takes three categories as parameters and has the following logical behavior:

\[
\frac{\Gamma \vdash A \quad \Sigma \vdash B \quad C \vdash D}{\Sigma \vdash q(A,B,C) \vdash D}
\]

(5)

An expression in category \( q(A,B,C) \) functions locally (i.e., with respect to \( \Gamma \)) as an \( A \), takes scope over a structure in category \( B \), and functions in the larger context (i.e., with respect to \( \Sigma \)) as an expression of category \( C \). This is exactly what a scope-taking expression needs to do. However, the logical characterization of \( q \) is problematic. For instance, although it is easy to write a left rule (a rule of use) for \( q \), as in (5), a general right rule (a rule of proof) remains elusive. As explained below, in the grammar proposed here, \( q \) can be simulated as a derived inference.

1.4 What this logic for scope will not account for

The account here seeks only to characterize an idealized, unconstrained version of quantifier scope. In any natural language, scope-taking will be constrained by syntactic and lexical factors. See [5] or [8] for formal grammars (also based on delimited continuations) that propose principled constraints on scope-taking.

2 NL\( _\lambda \)

The substructural grammar for characterizing scope discussed here is based on the non-associative Lambek grammar NL (see, e.g., [9], [11]). Since NL rejects all structural rules, including exchange, there will be two versions of implication: \( \setminus \), in which the argument is on the left, and \( / \), in which the argument is on the right.

NL characterizes the logic of function/argument combination when the functor is linearly adjacent to the argument. For scope-taking, linear adjacency is not sufficient. After all, a scope-taker is not adjacent to its argument, it is contained within its argument. What we need is a syntactic notion of ‘surrounding’ and ‘being surrounded by’. Therefore the grammar here will provide two modes: not only a merge mode (already introduced), for ordinary function/argument combination, with implications \( \setminus \) and \( / \); but also a continuation mode, which will govern scope-taking, with implications \( \setminus / \) and \( / \). (The interpretation of the continuation mode will be explained shortly.)
The logical rules for these connectives are identical to the rules given in [11]:129. They constitute the logical core of a two-mode type-logical grammar:

\[ \begin{array}{c}
\text{Axiom} \\
A \vdash A
\end{array} \]

(6)

\[ \begin{array}{c}
\Gamma \vdash A \\
\Sigma[\Gamma \cdot A \setminus B] \vdash C
\end{array} \quad \begin{array}{c}
A \cdot \Gamma \vdash B \\
\Sigma[\Gamma \cdot A \setminus B] \vdash C
\end{array} \quad \begin{array}{c}
\Gamma \vdash A \\
\Sigma[\Gamma \cdot A \setminus B] \vdash C
\end{array} \]

\[ \begin{array}{c}
\Gamma \vdash A \\
\Sigma[B \setminus A \cdot \Gamma] \vdash C
\end{array} \quad \begin{array}{c}
\Gamma \vdash A \setminus B \\
\Sigma[\Gamma \cdot A \setminus B] \vdash C
\end{array} \quad \begin{array}{c}
\Gamma \vdash A \\
\Sigma[B \setminus A \cdot \Gamma] \vdash C
\end{array} \]

\[ \begin{array}{c}
\Gamma \vdash A \\
\Sigma[B \setminus A \cdot \Gamma] \vdash C
\end{array} \quad \begin{array}{c}
\Gamma \vdash A \\
\Sigma[B \setminus A \cdot \Gamma] \vdash C
\end{array} \]

The sequents in the logical rules above have the form \( \Gamma \vdash A \), where \( A \) is a category and \( \Gamma \) is a structure. All categories are structures, and if \( \Gamma \) and \( \Delta \) are structures, then \( \Gamma \cdot \Delta \) (merge mode) and \( \Gamma \circ \Delta \) (continuation mode) are also structures.

In order to allow expressions to combine with material that surrounds it (or that it surrounds), we need to add a structural rule. In order to state this structural rule, we will need to enlarge the set of structures to include gapped structures: if \( \Sigma[\Delta] \) is a structure containing a distinguished substructure \( \Delta \), then \( \lambda \alpha \Sigma[\alpha] \) is also a structure, where \( \alpha \) is a variable taken from the set \( x, y, z, ... \). For instance, \( \lambda xx, \lambda yy, \lambda x \cdot (\text{saw} \cdot x) \), and \( \lambda x \lambda y (y \cdot (\text{saw} \cdot x)) \) are gapped structures.

Although gapped structures have important predecessors, including [12] and [6], they are not standard in discussions of substructural logics. One of the main goals of this paper is to explain how to understand gapped structures. A crucial part of achieving this goal will be to introduce a second substructural logic in the next section, NL_{CLR}, which will be equivalent to (a restricted version of) NL_{\lambda}. NL_{CLR} is a standard substructural logic, and does not involve any gapped structures.

With gapped structures in hand, we can state the following structural inference rule:

\[ \frac{\Gamma[\Sigma[\Delta]] \vdash A}{\Gamma[\Delta \circ \lambda \alpha \Sigma[\alpha]] \vdash A} \]

(7)

In words: if a structure \( \Sigma \) contains within it a structure \( \Delta \), then \( \Delta \) can take scope over the rest of \( \Sigma \), where ‘the rest of \( \Sigma \)’ is represented as the gapped structure \( \lambda \alpha \Sigma[\alpha] \).

Schematically, we have:

\[ \Sigma[\Delta] \equiv \lambda \alpha \Sigma[\alpha] \]

(8)

The postulate says that if \( \Delta \) (the small grey triangle) is some structure embedded within a larger structure \( \Sigma \) (the complete larger triangle), we can view these components in a completely equivalently way by articulating them into a foreground and a background, that is, into a plug and a context—an expression and its continuation. Then \( \Delta \) will
be the foregrounded expression, and the clear notched triangle will be its context, the continuation \( \lambda x \Sigma [\alpha] \).

An expression in a category with the form \( A \setminus B \) will be a *continuation*: something that would be a complete expression of category \( B \), except that it is missing an expression of category \( A \) somewhere inside of it. An expression in a category with the form \( C \setminus (A \setminus B) \) will be something that combines with a continuation of category \( A \setminus B \) surrounding it to form a result expression of category \( C \).

This logic allows for unbounded scope displacement, since there are no constraints on the complexity of the scope host \( \Sigma \). It also allows for embedded scope-taking, since \( \Gamma \) may be non-empty. As for scope ambiguity, we have the following two derivations:

\[
\begin{align*}
DP \cdot (\text{loves} \cdot DP) \vdash S \\
\frac{DP \circ \lambda x(DP \cdot (\text{loves} \cdot x)) \vdash S}{\lambda x(DP \cdot (\text{loves} \cdot x)) \vdash \downarrow^R S \vdash S} \quad /L \\
\frac{S \setminus (DP \setminus S) \circ \lambda x(DP \cdot (\text{loves} \cdot x)) \vdash S}{\text{everyone} \circ \lambda x(DP \cdot (\text{loves} \cdot x)) \vdash S} \quad \text{LEX} \\
\frac{DP \cdot (\text{loves} \cdot \text{everyone}) \vdash S}{\lambda x(\text{loves} \cdot \text{everyone}) \vdash \downarrow^R S} \quad S \vdash S} \quad /L \\
\frac{S \setminus (DP \setminus S) \circ \lambda x(\text{loves} \cdot \text{everyone}) \vdash S}{\text{someone} \circ \lambda x(\text{loves} \cdot \text{everyone}) \vdash S} \quad \text{LEX} \\
\frac{\text{someone} \circ \lambda x(\text{loves} \cdot \text{everyone}) \vdash S}{\text{someone} \cdot (\text{loves} \cdot \text{everyone}) \vdash S} \quad \equiv
\end{align*}
\]

The Curry-Howard labeling for this derivation (see [4]) is \( \exists x \forall y . \text{loves} \ y \ x \). In general, the scope-taker that is focussed (i.e., targeted by the structural postulate) lower in the proof takes wider scope.

\[
\begin{align*}
DP \cdot (\text{loves} \cdot DP) \vdash S \\
\frac{DP \circ \lambda x(x \cdot (\text{loves} \cdot DP)) \vdash S}{\lambda x(x \cdot (\text{loves} \cdot DP)) \vdash \downarrow^R S \vdash S} \quad /L \\
\frac{S \setminus (DP \setminus S) \circ \lambda x(x \cdot (\text{loves} \cdot DP)) \vdash S}{\text{someone} \circ \lambda x(x \cdot (\text{loves} \cdot DP)) \vdash S} \quad \text{LEX} \\
\frac{\text{someone} \circ \lambda x(x \cdot (\text{loves} \cdot DP)) \vdash S}{\text{someone} \cdot (\text{loves} \cdot DP) \vdash S} \quad \equiv \\
\frac{DP \circ \lambda x(\text{loves} \cdot x) \vdash S}{\lambda x(\text{loves} \cdot x) \vdash \downarrow^R S \vdash S} \quad /L \\
\frac{S \setminus (DP \setminus S) \circ \lambda x(\text{loves} \cdot x) \vdash S}{\text{everyone} \circ \lambda x(\text{loves} \cdot x) \vdash S} \quad \text{LEX} \\
\frac{\text{everyone} \circ \lambda x(\text{loves} \cdot x) \vdash S}{\text{someone} \cdot (\text{loves} \cdot x) \vdash S} \quad \equiv
\end{align*}
\]

In this case, the semantic labeling gives the universal wide scope: \( \forall y \exists x . \text{loves} \ y \ x \).
3 Soundness and completeness via NL<sub>CL</sub>

The proofs of soundness and completeness for NL<sub>λ</sub> will proceed by defining NL<sub>CL</sub>, a more standard substructural logic whose soundness and completeness follows from the general results of [13]. I will then give conditions under which NL<sub>λ</sub> and NL<sub>CL</sub> are equivalent.

NL<sub>CL</sub> has the same logical rules as NL<sub>λ</sub>. Instead of the structural postulate λ, however, NL<sub>CL</sub> has the following three structural postulates:

\[
\begin{align*}
  p & \vdash I & p \circ l & \vdash A & (p \circ q) \circ r & \vdash B & (p \circ q) \circ r & \vdash C
\end{align*}
\]

These postulates are identical to the ones given in [2]. [13]:30 considers I (which he writes ‘0’) as “a zero-place punctuation mark,” where punctuation marks (p. 19) “stand to structures in the same way that connectives stand to formulae.” Likewise, B and C are also zero-place punctuation marks. The double horizontal line indicates that these rules are bi-directional, i.e., inference in the top-to-bottom direction and in the bottom-to-top direction are both valid. Restall calls the top-to-bottom inference for the I postulate Push, and the other direction Pop.

In the form of an official inference rule, the I postulate (for instance) is written

\[
\begin{align*}
  \Sigma [p] \vdash A & \quad \Sigma [p \circ l] \vdash A
\end{align*}
\]

and similarly for the other rules.

An example derivation will show how these postulates work together to achieve in-situ quantification for the sentence John saw everyone:

\[
\begin{align*}
  DP & \vdash DP & S & \vdash S & \\text{\(L\)}
  DP & \vdash DP & DP \cdot (DP \circ S) / DP \cdot DP & \vdash S & \\text{\(L\)}
  John \cdot (saw \cdot DP) & \vdash S & \\text{LEX}
  john \cdot (saw \cdot (DP \circ l)) & \vdash S & \\text{B}
  john \cdot (DP \circ ((B \cdot saw) \cdot l)) & \vdash S & \\text{B}
  (B \cdot john) \cdot ((B \cdot saw) \cdot l) & \vdash S & \\text{\(R\)}
  S / (DP \circ (B \cdot john) \cdot ((B \cdot saw) \cdot l)) & \vdash S & \\text{LEX}
  everyone \circ ((B \cdot john) \cdot ((B \cdot saw) \cdot l)) & \vdash S & \\text{B}
  John \cdot (everyone \circ ((B \cdot saw) \cdot l)) & \vdash S & \\text{B}
  john \cdot (saw \cdot (everyone \circ l)) & \vdash S & \\text{I}
  john \cdot (saw \cdot everyone) & \vdash S
\end{align*}
\]
NL\textsubscript{CL} is sound and complete with respect to the usual class of relational models. This follows directly from the proofs given in [13], chapter 11. In particular, [13]:249 provides an algorithm for constructing frame conditions corresponding to the structural postulates.

**Theorem** (Soundness and completeness): $X \vdash A$ is provable in NL\textsubscript{CL} iff for every model $\mathfrak{M} = \langle \mathcal{F}, \models \rangle$ that satisfies the frame conditions, $\forall x \in \mathcal{F}, x \models X \rightarrow x \models A$.

Proof: given in [13], theorems 11.20, 11.37.

Furthermore, NL\textsubscript{CL} is conservative with respect to NL. That is,

**Theorem** (Conservativity): Let an NL sequent be a sequent built up only from the formulas and structures allowed in NL: $\langle /, \\backslash, \cdot \rangle$. An NL sequent is provable in NL\textsubscript{CL} iff it is provable in NL.


### 4 The connection between NL\textsubscript{\lambda} and NL\textsubscript{CL}

This section investigates the conditions under which a derivation in NL\textsubscript{\lambda} has an equivalent derivation in NL\textsubscript{CL}.

I define the following class of structures:

$$
\Gamma [p] ::= p \ | \ p \circ q \ | \ q \cdot \Gamma [p] \ | \ \Gamma [p] \cdot q \ | \ \lambda y. \Gamma [p]
$$

(12)

Given a structure $p$, a $\lceil \rceil$-context will consist either of the empty context, or else the entire left element at the top level of a $\circ$ structure, or else a larger context built up from $\cdot$ and $\lambda$. We can impose these restrictions on NL\textsubscript{\lambda} by replacing the original lambda postulate with one that mentions $\lceil \rceil$-contexts:

$$
\Sigma [\Delta] \equiv \Delta \circ \lambda \alpha \Sigma [\alpha]
$$

(13)

To illustrate, the following (bidirectional) inferences are licensed by (13):

$$
\frac{A}{A \circ \lambda x} \quad \frac{A \circ B}{A \circ \lambda x (x \circ B)} \quad \frac{A \cdot B}{A \circ \lambda x (x \cdot B)} \quad \frac{\lambda x. (x \cdot B)}{B \circ \lambda y \lambda x (x \cdot y)}
$$

(14)

But not these:

$$
\frac{(A \cdot B) \circ C}{A \circ \lambda x ((x \cdot B) \circ C)} \quad \frac{A \circ B}{B \circ \lambda y (A \circ y)}
$$

(15)

The reason these last two inferences are not allowed is that abstraction across $\circ$ is forbidden unless the abstractee is the complete left element connected by $\circ$.

The inspiration for NL\textsubscript{CL} comes from the well-known equivalence between the lambda calculus and Combinatory Logic. More specifically, the postulates of NL\textsubscript{CL} implement a version of Sh"{o}nfinckel’s embedding of $\lambda$-terms into Combinatory Logic. Adapting the presentation in [1]:152, [4] define $\langle \cdot \rangle$, which maps an arbitrary gapped
structure into a NL\textsubscript{CL} structure:

\begin{align*}
\langle x \rangle & \equiv x \\
\langle p \cdot q \rangle & \equiv \langle p \rangle \cdot \langle q \rangle \\
\langle p \circ q \rangle & \equiv \langle p \rangle \circ \langle q \rangle \\
\langle \lambda x. p \rangle & \equiv \lambda (x, \langle p \rangle)
\end{align*}

(16)

\begin{align*}
\lambda (x, x) & \equiv I \\
\lambda (x, p \cdot q) & \equiv (B \cdot p) \cdot \lambda (x, q) \quad \text{(x not free in } p) \\
\lambda (x, p \cdot q) & \equiv (C \cdot \lambda (x, p)) \cdot q \quad \text{(x not free in } q) \\
\lambda (x, x \circ q) & \equiv (C \cdot I) \circ q \quad \text{(x not free in } q)
\end{align*}

With this mapping defined, I can state the following theorem:

**Theorem** (Faithfulness of the \langle \cdot \rangle mapping from \lambda-structures into CL-structures):
For any structure \( p \) and context \( \Gamma \upharpoonright \Gamma \),

\[ \langle p \circ \lambda x \Gamma [x] \rangle \quad \text{CL} \]

(17)

Here, CL schematizes over any series of structural inferences allowable in NL\textsubscript{CL}.

This theorem, in turn, enables me to characterize the equivalence between NL\textsubscript{\lambda} and NL\textsubscript{CL}.

**Theorem** (Embedding of \lambda-free theorems of NL\textsubscript{\lambda} in NL\textsubscript{CL}): For any derivation in NL\textsubscript{\lambda} (with abstraction restricted to \[ \upharpoonright \] -contexts) whose final sequent does not contain any \lambda-structures, there is an equivalent derivation in NL\textsubscript{CL}.

Here, two derivations are equivalent if they differ only in the application of structural rules. They must have the same axiom instances, the same conclusion, and the Curry-Howard labeling must be the same up to \( \alpha \)-equivalence.

The equivalence in the other direction is more straightforward.

**Theorem** (Embedding of IBC-free theorems of NL\textsubscript{CL} in NL\textsubscript{\lambda}): for any derivation in NL\textsubscript{CL} whose conclusion does not contain the structures I, B, or C, there is an equivalent derivation in NL\textsubscript{\lambda}.

The equivalence involves replacing each instance of I, B, and C with instances of the lambda postulate as follows:

\begin{align*}
\frac{p}{p \vdash I} & \quad \sim \quad \frac{p}{p \vdash \lambda x} \\
\frac{p \cdot (q \circ r)}{q \circ ((B \cdot p) \cdot r)} & \quad \sim \quad \frac{p \cdot (q \circ r)}{q \circ \lambda x(p \cdot (x \circ r))} \\
\frac{(p \circ q) \cdot r}{p \circ ((C \cdot q) \cdot r)} & \quad \sim \quad \frac{(p \circ q) \cdot r}{p \circ \lambda x((x \circ q) \cdot r)}
\end{align*}

(18)

Note that each of these applications of the lambda postulate obeys the restriction to \[ \upharpoonright \] -contexts.
Thus NL$_\lambda$ (with the lambda-postulate restricted to [ ]-contexts) and NL$_{CL}$ are equivalent: any sequent containing only structures built from $\cdot$ and $\circ$ will be a theorem of one just in case it is a theorem of the other. Furthermore, for each derivation in one system, there will be a matching derivation in the other that differs only in the application of structural rules, which means that the semantic values of the two derivations will be identical. Since NL$_{CL}$ is conservative with respect to the non-associative Lambek grammar NL, NL$_\lambda$ is too. As a result, NL$_\lambda$ with restricted abstraction contexts can be used with full confidence that it is equivalent to an ordinary and well-behaved substructural grammar.

5 Cut elimination and decidability

5.1 Cut elimination for NL$_\lambda$

Cut elimination is crucial to proving decidability. I will prove cut elimination here and, in the next subsection, decidability for NL$_\lambda$.

The cut rule, repeated here, characterizes transitivity of the logical system:

\[
\frac{\Gamma \vdash A \quad \Sigma[A] \vdash B}{\Sigma[\Gamma] \vdash B} \text{CUT} \tag{19}
\]

The cut rule says that if $\Gamma$ is a proof of $A$, and $\Sigma$ is a proof of $B$ that depends on proving $A$, then we can construct a new proof of $B$ in which $A$ has been replaced with $\Gamma$. The formula $A$ has been ‘cut out’ of the derivation.

The proof strategy, just as it was above for completeness, will be to rely on Restall’s general proof of cut elimination for Gentzen-style sequent systems. This strategy emphasizes the ordinariness and the standardness of the logics here, and how they fit into a larger landscape of substructural logics.

In order for Restall’s proof to apply, we need to demonstrate that the cut rule, the structural rule, and the logical rules conform to certain conditions. This is perfectly straightforward (see [4] for full details). Therefore we have:

**Theorem** (cut elimination): given that the parameter conditions, the eliminability of matching principal constituents, and the regularity condition hold, if $\Gamma \vdash A$ and $\Delta[A] \vdash B$ are provable, then $\Delta[\Gamma] \vdash B$ is also provable.

Proof: see [13]; section 6.3.

5.2 Decidability of NL$_\lambda$

Decidability is a property a logic has if it is always possible to figure out whether a sequent is a theorem (has a proof, has a derivation) in a bounded amount of time, where the bound is some concrete function of the complexity of the sequent to be proved.

The structural postulate given above in (7) is a reversible inference, that is, it is bidirectional. In the discussion that follows, it will be helpful to keep track of the two directions separately:

\[
\frac{\Sigma[\Delta[A]] \vdash B}{\Sigma[A \circ \lambda \Delta [x]] \vdash B} \text{REDUCTION} \quad \frac{\Sigma[A \circ \lambda \Delta [x]] \vdash B}{\Sigma[\Delta[A]] \vdash B} \text{EXPANSION} \tag{20}
\]
Since in proof search we are starting with the conclusion and trying to find appropriate premises, the names ‘reduction’ and ‘expansion’ are relative to the bottom-to-top direction of reading proofs. The problem for decidability is that there is no limit to the opportunities for expansion, since $B ≡ B \circ \lambda_xx ≡ (B \circ \lambda_xx) \circ \lambda_xx ≡ \ldots$.

Nevertheless, we have the following result:

**Theorem (Decidability):** NL$_\lambda$ with abstraction restricted to $\lceil \cdot \rceil$-contexts is decidable.

Proof sketch: we will show that every derivation in NL$_\lambda$ is equivalent to a derivation in which each inference has the subformula property. For our purposes, an inference has the subformula property just in case every formula in the premises corresponds to a unique (part of a) formula in the conclusion.

It is easy to check that every logical rule in NL$_\lambda$ has the subformula property, as does Reduction, so we only need to worry about Expansion.

The first step will be to push each Expansion inference upwards in the proof until one of two things happens: either it encounters a matching Reduction instance, in which case the two rules cancel each other out, and can be eliminated from the proof; or else the expansion is adjacent to a logical rule that introduces the focussed occurrence of $\circ$.

It turns out that the only candidate for such a logical rule is $\lambda R$. It is easy to see that the combination of Expansion with an instance of $\lambda R$ can be viewed as a two-step rule that in aggregate has the desired subformula property.

We can replace the adjacent pair of inferences on the left with the derived inference on the right, which we can call $\lambda L$. By repeated application of this reasoning, every instance of Expansion can either be eliminated, or replaced with an instance of $\lambda L$.

Having eliminated all expansion inferences, we can eliminate Reduction inferences in a similar fashion. Reasoning dually, Reduction inferences can always be pushed downwards until the Reduction encounters an instance of $\lambda R$ that targets the $\circ$ connective introduced by Reduction. And once again, we can replace the combination of the reduction and the instance of $\lambda R$ with a derived rule that captures their net effect:

At this point, we have two derived logical inferences: $\lambda R$, and $\lambda L$. The $\lambda R$ rule says that in-situ elements can take scope directly from embedded positions, without needing to first be abstracted leftwards. Dually, the $\lambda L$ rule says that a context can surround a scope-taker even when the scope-taker is embedded in a still larger surrounding context. Adding the two derived logical rules to the standard logical rules leads to derivations of
in-situ scope-taking, illustrated here for the sentence Ann saw everyone:

\[
\frac{\text{ann} \cdot (\text{saw} \cdot \text{DP}) \vdash S}{\lambda x. \text{ann} \cdot (\text{saw} \cdot x) \vdash \text{DP} \succeq S \quad S \vdash \succeq L_{\lambda}} \\
\frac{\text{ann} \cdot (\text{saw} \cdot \text{S} \div (\text{DP} \succeq S)) \vdash S}{\lambda x. \text{ann} \cdot (\text{saw} \cdot \text{everyone}) \vdash S}
\]

(23)

In effect, we have compiled both parts of the structural rule into the logical rules. If we add these two derived logical rules to the grammar, we can eliminate the structural rules entirely.

Note that each inference rule, including the derived inference rules, eliminates exactly one logical connective. As a result, no part of the proof can have a depth greater than the number of logical connectives in the final sequent. Since there is at most one way to apply each rule to a given occurrence of a logical connective, decidability follows immediately.

5.3 Proof search with gaps

The treatment of scope-taking can be extended to a treatment of overt syntactic movement (see [4]). From the point of view of decidability, gaps corresponding to syntactic movement are a challenge, since they allow us to posit new structure during the course of a proof search, in which case we lose the subformula property. An extension of the technique developed in the previous section allows derivations with gaps without giving up decidability.

\[
\frac{\Gamma[A \cdot B] \vdash C}{\Gamma[A] \vdash B \succeq C \quad R_{\lambda} \quad \Gamma[A \cdot B] \vdash C \quad R_{\lambda}} \quad \frac{\Gamma[A] \vdash B \succeq C \quad R_{\lambda}}{\Gamma[A] \vdash C \quad R_{\lambda}}
\]

(24)

Since each of these inferences has the subformula property, and moreover, eliminates a logical connective, adding them to the logic will not compromise decidability.

To illustrate these logical rules in action, here is a derivation of the wh-question Who did Ann see (with did suppressed for simplicity):

\[
\frac{\text{ann} \cdot (\text{see} \cdot \text{DP}) \vdash S}{\text{ann} \cdot \text{see} \vdash \text{DP} \succeq S \quad Q \vdash Q \quad L_{\lambda}} \quad \frac{\text{ann} \cdot \text{see} \vdash \text{DP} \succeq S \quad Q \vdash Q \quad L_{\lambda}}{\text{who} \cdot (\text{ann} \cdot \text{see}) \vdash Q \quad L_{\lambda}}
\]

(25)

6 Deriving the q type constructor

If we carry this fusion strategy one step further, we derive the rule of use for Moortgat’s q type constructor, given above in (5):

\[
\frac{\Gamma[A] \vdash B}{\lambda x. \Gamma[x] \vdash A \succeq B \quad R_{\lambda} \quad \Sigma[C] \vdash D \quad L_{\lambda}} \quad \frac{\Sigma[C \div (B \succeq A)] \vdash D}{\Sigma[C \div (B \succeq A)] \vdash D \quad L_{\lambda}}
\]

(26)
We now have an explanation for why it was impossible to find a general right rule for the \( q \) type constructor: it is because the \( q \) inference represents the fusion of two logically distinct inferences, each with their own left and right rules.

In support of the usefulness of factoring the \( q \) into independent components, note that the decidable system here extends to parasitic scope, a technique proposed in [2] to account for the scope-taking behavior of adjectives such as same and different. Parasitic scope requires the independent logical components to be interleaved in a way that cannot be duplicated by the \( q \) inference alone:

\[
\begin{align*}
\lambda x.(\text{the} \cdot \lambda y x. (\text{same waiter}) \cdot (\text{served} \cdot \text{everyone})) & \vdash S \\
\lambda x.(\text{the} \cdot \lambda y x. (\text{same waiter}) \cdot (\text{served} \cdot \text{everyone})) & \vdash S \\
\lambda x.(\text{the} \cdot \lambda y x. (\text{same waiter}) \cdot (\text{served} \cdot \text{everyone})) & \vdash S
\end{align*}
\]

Although the innermost pair of \( \lambda L \lambda \) and \( \lambda R \lambda \) could be fused into a single instance of the \( q \) inference, the outermost pair could not.

### 7 Comparison with Quantifier Raising

Here is the structural operation of Quantifier Raising, illustrated with categories borrowed from NL\( \lambda \):

![Diagram of Quantifier Raising]

\[
\begin{align*}
\lambda x.(\text{the} \cdot \lambda y x. (\text{same waiter}) \cdot (\text{served} \cdot \text{everyone})) & \vdash S \\
\lambda x.(\text{the} \cdot \lambda y x. (\text{same waiter}) \cdot (\text{served} \cdot \text{everyone})) & \vdash S \\
\lambda x.(\text{the} \cdot \lambda y x. (\text{same waiter}) \cdot (\text{served} \cdot \text{everyone})) & \vdash S
\end{align*}
\]

In NL\( \lambda \), this derivational step can be simulated closely using the \( \lambda \) postulate (reading the proof from bottom upwards):
everyone ◦ λx(ann ◦ (saw ◦ x)) ⊢ S
ann ◦ (saw ◦ everyone) ⊢ S
\[\lambda\]

So the logic here captures a significant portion of the insight embodied in Quantifier Raising. Is NL₃ then just the logic of Quantifier Raising? In some sense, clearly yes.

However, there are important differences between Quantifier Raising and the \(\lambda\) postulate of NL₃.

For one, the lambda postulate is bidirectional. This reflects the fact that the two structures it relates are fully equivalent logically: they denote the same object in the model. In contrast, in the treatment in, e.g., [7], the pre-QR structure does not have a denotation. Thus the main motivation for executing an instance of Quantifier Raising is to produce a new meaning. In contrast, the lambda postulate here is a structural rule.

Like all structural rules, the effect of the rule on the Curry-Howard labeling is null (no change to the semantic labeling). Quantifier Raising is conceived of as a rule that has a semantic effect but no syntactic effect (it constitutes ‘covert’ movement); the lambda postulate here has a syntactic effect, but no semantic effect. Its role in the logic is purely to characterize the syntactic operation by which a delimited continuation combines with its functor (by begin surrounding by it) or its argument (by surrounding it).

For another difference, Quantifier Raising can create unbound traces.

Unbound trace: [[some [friend [of everyone]]][called]]
QR \[\text{everyone}(\lambda y[[\text{some} \ [\text{friend} \ [\text{of} \ y]]][\text{called}]]]]
QR \[\text{[[some} \ [\text{friend} \ [\text{of} \ y]]][\lambda x[\text{everyone}(\lambda y.x)]]\text{[called]]}}]

If QR targets the embedded quantifier \textit{everyone} first, and then targets the originally enclosing quantifier \textit{some friend of}, the variable introduced by the QR of \textit{everyone} (in this case, \(y\)) will end up unbound (free) in the final Logical Form structure. In a QR system, such derivations must be stipulated to be ill-formed. In the logics developed here, unbound traces cannot arise.

Finally, although I have not emphasized this in the discussion here, the substructural logics given here allow fine-grained control over order of evaluation, allowing accounts of order-sensitive phenomena such as crossover, reconstruction, negative polarity licensing, and more. Evaluation order and its applications in natural language are discussed in detail in [4].

8 Conclusion

What is the logic of scope? Here is my answer: when an expression takes scope, it combines with one of its (delimited) continuations. The substructural logics given here, NL₃ and NLCL, illustrate two (equivalent) ways to implement a concrete continuation-based grammar. These grammars are sound and complete with respect to the usual class of models, they are conservative with respect to NL, and they enjoy cut elimination and decidability.
References

3. Barker, Chris: Scope. In the Handbook of Contemporary Semantics, 2d edition, Shalom Lap-
   pin and Chris Fox (eds), Wiley-Blackwell. (2014)
4. Barker, Chris, Shan, Chung-chieh: Continuations and Natural Language. Oxford University
   Press. (2014)
   Meeting of the Association for Computational Linguistics. Morgan Kaufmann. (2002)
8. Kiselyov, Oleg, Shan, Chung-chieh: Continuation hierarchy and quantifier scope. In Formal
   Approaches to Semantics and Pragmatics, Eric McCready, Katsuhiko Yabushita, Kei Yoshim-
   moto (eds). Springer. (2014)
9. Lambek, Joachim: The mathematics of sentence structure. The American Mathematical
   Monthly 60.3, 154–170 (1958)
10. Moortgat, Michael: Categorial Investigations: Logical and Linguistic aspects of the Lambek
    calculus. Foris. (1988)
11. Moortgat, Michael: Categorial Type Logics. In the Handbook of Logic and Language, J. van
    13th Amsterdam Colloquium, Robert van Rooij and Martin Stokhof (eds). 13th Amsterdam
    Colloquium. ILLC. (2001)