Bracket Induction for Lambek Calculus with Bracket Modalities

Glyn Morrill[1], Stepan Kuznetsov[2,5], Max Kanovich[3,5], and Andre Scedrov[4,5]

[1] Universitat Politécnica de Catalunya, Barcelona, Spain; morrill@cs.upc.edu
[2] Steklov Mathematical Institute of the RAS, Moscow, Russia; sk@mi.ras.ru
[3] University College London, London, UK; m.kanovich@ucl.ac.uk
[4] University of Pennsylvania, Philadelphia, U.S.A.; scedrov@math.upenn.edu
[5] National Research University Higher School of Economics, Moscow, Russia

Abstract. Relativisation involves dependencies which, although unbounded, are constrained with respect to certain island domains. The Lambek calculus **L** can provide a very rudimentary account of relativisation limited to unbounded peripheral extraction; the Lambek calculus with bracket modalities **Lb** can further condition this account according to island domains. However in naïve parsing/theorem-proving by backward chaining sequent proof search for **Lb** the bracketed island domains, which can be indefinitely nested, have to be specified in the linguistic input. In realistic parsing word order is given but such hierarchical bracketing structure cannot be assumed to be given. In this paper we show how parsing can be realised which induces the bracketing structure in backward chaining sequent proof search with **Lb**.

Keywords: Lambek calculus with brackets, bracket induction, categorial grammar

1 Introduction

Relativisation involves dependencies which can be medial as well as peripheral and which, although unbounded, are constrained with respect to certain island domains; furthermore these unbounded dependencies can be multiple, or parasitic, in a way which appears to depend on islands. The Lambek calculus **L** of Lambek 1958 [9] can provide a very rudimentary account of relativisation limited to unbounded peripheral extraction; the Lambek calculus with bracket modalities **Lb** of Morrill 1992 [13] and Moortgat 1995 [11] can further condition this account according to island domains; and the Lambek calculus with bracket modalities and universal subexponential **Lb**! (after Girard 1987 [3]) accommodates furthermore medial and parasitic extraction (Morrill 2017 [14]). However in naïve parsing/theorem-proving by backward chaining sequent proof search for **Lb** and **Lb**! the bracketed island domains, which can be indefinitely nested, have to be specified in the linguistic input. In realistic parsing word order is given but such hierarchical bracketing structure cannot be assumed to be given. In this paper we show how parsing can be realised which induces the bracketing structure in backward chaining sequent proof search for **Lb**.

1.1 Relativisation

Relativisation is an unbounded dependency construction; the distance between a relative pronoun (filler) and the position it binds (gap) can be unboundedly long:

(1) a. the man that$_i$ Mary loves e_i
b. the man that$_i$ John thinks Mary loves e_i
c. the man that$_i$ Suzy knows John thinks Mary loves e_i
$\vdots$

Some domains are islands to relativisation and cannot be penetrated by the filler-gap (extraction) dependency, for example adverbial phrases are weak islands (extraction is semi-acceptable) and relative clauses themselves are strong islands (extraction is unacceptable):

(2) a. ?the paper that$_i$ John laughed [without reading e_i]
b. (?)the paper that$_i$ John went to Paris [without reading e_i]
c. *the waitress that$_i$ John saw the man [that married e_i]

Relativisation can be medial:

(3) the contract that$_i$ John signed e_i yesterday

And although islands block singleton extractions, relativisation can have a parasitic gap in a weak island dependent on a non-island host gap:

(4) the paper that$_i$ John filed e_i [without reading e_i]

Such parasitic gaps can also appear in subjects, which are weak islands:

(5) a. ??the man that$_i$ [the friends of e_i] laughed
b. the man that$_i$ [the friends of e_i] praised e_i

A single host gap can license parasitic gaps in multiple islands; for example:

(6) the paper that$_i$ [the editor of e_i] filed e_i [without reading e_i]

In this paper we give an account in terms of the Lambek calculus with bracket modalities **Lb** of the data of (1–2). We provide a calculus and consider a bracket-inducing parsing/theorem-proving algorithm. We illustrate this algorithm on an example of a lexical grammar for a small fragment of English. The input of this algorithm is just a string (linearly ordered sequence of words), without any bracketing information. The job of the algorithm is to induce (guess) the correct placement of brackets, as well as to derive the resulting **Lb** sequent. The algorithm relies on the assignment of types drawn from the lexical grammar. In Section 6 we discuss the complexity of our algorithm, in comparison with both a naïve approach with brute-force guessing of the correct bracketing and the pseudo-polynomial algorithm for theorem-proving (but not parsing) in **Lb** presented in our earlier paper [5].

As for examples (3–6), their parsing involves **Lb**!, the extension of **Lb** with the universal exponential modality, which is beyond the scope of the present paper. For **Lb**!, the derivability problem in general is algorithmically undecidable [7]. A practically useful fragment, however, guarded by the so-called *bracket non-negative condition,* was shown to be decidable [17] and to belong to the NP complexity class [7]. In other words, despite undecidability of the whole calculus, practical parsing with **Lb**! has the same complexity as for **Lb** without the exponential. We hope to extend the results presented here from **Lb** to **Lb**! (restricted by the bracket non-negative condition) in a subsequent paper.

Our analysis of linguistic examples in this paper follows Morrill 2011 [20] and Morrill 2017 [14]. This paper's purposes are mostly technical. Namely, we present an approach that allows the parsing algorithm for **Lb** to induce open and closed "symbolic" brackets by itself and therefore avoid requesting this information (which is not part of the actual text in natural language) from the user. Therefore, we refrain from deep discussions of the design of the lexical grammar itself and its empirical justification. In particular, we leave beyond the scope of this paper the subtle issues of *semi-grammaticality* [20, Sect. 5.4] of extraction from weak islands; see, for instance, examples (2b) and (5a). For such examples, Morrill 2011 [20, Sect. 5.4] suggests including further structural rules that allow some violation of bracketing, but with a cost for such. The number of applications of such rules is supposed to be counted, and the more times they are used the less grammatical the target sentence is considered. The modification of the algorithm described in this paper to allow such rules is a topic for further investigation.

1.2 Lambek Calculus with Bracket Modalities

The set **Tp** of types of the Lambek calculus with bracket modalities **Lb** is defined in terms of a set $\mathcal{P}$ of primitive types as follows:

$$\mathbf{Tp} ::= \mathcal{P} \mid \langle\rangle\mathbf{Tp} \mid [\,]^{-1}\mathbf{Tp} \mid \mathbf{Tp}{\bullet}\mathbf{Tp} \mid \mathbf{Tp}\backslash\mathbf{Tp} \mid \mathbf{Tp}/\mathbf{Tp}$$

A configuration **Config** is a well-bracketed string of types:

$$\begin{array}{ll}\mathbf{Config} & ::= \mathbf{TreeTerm} \mid \mathbf{Config}, \mathbf{Config} \\ \mathbf{TreeTerm} & ::= \mathbf{Tp} \mid [\mathbf{Config}]\end{array}$$

Note that this definition builds in "Lambek's restriction" whereby configurations (and bracketed configurations) are non-empty. Lambek's restriction, even in the fragment without brackets, is motivated linguistically; otherwise we could derive the sequent $(CN/CN)/(CN/CN), CN \Rightarrow CN$, validating grammatically incorrect phrases like "very book" (compare with "very interesting book", parsed as $(CN/CN)/(CN/CN), CN/CN, CN \Rightarrow CN$). In this paper, Lambek's restriction is crucial for the construction to work (see Section 5).

A sequent is an expression of the form **Config** $\Rightarrow$ **Tp**.

1.3 Sequent Calculus for Lb

Let $\Delta(\Gamma)$ signify a structure Δ with a distinguished substructure Γ. The sequent calculus for **Lb** is as follows:

$$\dfrac{}{P \Rightarrow P}\,id \qquad \dfrac{\Gamma \Rightarrow A \qquad \Delta(A) \Rightarrow B}{\Delta(\Gamma) \Rightarrow B}\,Cut$$

$$\dfrac{\Delta(A) \Rightarrow B}{\Delta([[\,]^{-1}A]) \Rightarrow B}\,[\,]^{-1}L \qquad \dfrac{[\Delta] \Rightarrow B}{\Delta \Rightarrow [\,]^{-1}B}\,[\,]^{-1}R$$

$$\dfrac{\Delta([A]) \Rightarrow B}{\Delta(\langle\rangle A) \Rightarrow B}\,\langle\rangle L \qquad \dfrac{\Delta \Rightarrow B}{[\Delta] \Rightarrow \langle\rangle B}\,\langle\rangle R$$

$$\dfrac{\Delta(A, B) \Rightarrow D}{\Delta(A\bullet B) \Rightarrow D}\,\bullet L \qquad \dfrac{\Delta \Rightarrow A \qquad \Gamma \Rightarrow B}{\Delta, \Gamma \Rightarrow A\bullet B}\,\bullet R$$

$$\dfrac{\Gamma \Rightarrow B \qquad \Delta(C) \Rightarrow D}{\Delta(C/B, \Gamma) \Rightarrow D}\,/L \qquad \dfrac{\Gamma, B \Rightarrow C}{\Gamma \Rightarrow C/B}\,/R$$

$$\dfrac{\Gamma \Rightarrow A \qquad \Delta(C) \Rightarrow B}{\Delta(\Gamma, A\backslash C) \Rightarrow B}\,\backslash L \qquad \dfrac{A, \Gamma \Rightarrow C}{\Gamma \Rightarrow A\backslash C}\,\backslash R$$

Moortgat 1995 [11] shows that this calculus enjoys Cut-elimination: that every theorem has a Cut-free proof. We omit Cut in what follows.

1.4 Grammar

Consider the micro-**Lb** lexical grammar in Figure 1 whereby weak islands are singly bracketed and strong islands doubly bracketed. For example, (1a) is derived as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{N \Rightarrow N} \qquad \dfrac{\dfrac{\dfrac{}{N \Rightarrow N}}{[N] \Rightarrow \langle\rangle N}\,\langle\rangle R \qquad \dfrac{}{S \Rightarrow S}}{[N], \langle\rangle N\backslash S \Rightarrow S}\,\backslash L}{[N], (\langle\rangle N\backslash S)/N, N \Rightarrow S}\,/L}{[N], (\langle\rangle N\backslash S)/N \Rightarrow S/N}\,/R \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{CN \Rightarrow CN} \qquad \dfrac{}{CN \Rightarrow CN}}{CN, CN\backslash CN \Rightarrow CN}\,\backslash L}{CN, [[\,]^{-1}(CN\backslash CN)] \Rightarrow CN}\,[\,]^{-1}L}{CN, [[[\,]^{-1}[\,]^{-1}(CN\backslash CN)]] \Rightarrow CN}\,[\,]^{-1}L}{CN, [[[\,]^{-1}[\,]^{-1}(CN\backslash CN)/(S/N), [N], (\langle\rangle N\backslash S)/N]] \Rightarrow CN}\,/L$$

Examples (2a–c) are blocked because the bracket modalities require the island bracketings indicated below, which prevent the hypothetical subtype of the relative pronoun from associating into the object positions of “reading” and “married” respectively. We discuss example (2a) in detail in Subsection 5.3.

ate: $(\langle\rangle N\backslash S)/N$
built: $(\langle\rangle N\backslash S)/N$
cat: CN
contract: CN
dog: CN
editor: CN/PP
filed: $(\langle\rangle N\backslash S)/N$
friends: CN/PP
house: CN
in: $[\,]^{-1}((\langle\rangle N\backslash S)\backslash(\langle\rangle N\backslash S))/N$
Jack: N
John: N
killed: $(\langle\rangle N\backslash S)/N$
knows: $(\langle\rangle N\backslash S)/S$
laughed: $\langle\rangle N\backslash S$
lay: $\langle\rangle N\backslash S$
loves: $(\langle\rangle N\backslash S)/N$
malt: CN
man: CN
married: $(\langle\rangle N\backslash S)/N$
Mary: N
paper: CN
Paris: N
praised: $(\langle\rangle N\backslash S)/N$
rat: CN
reading: $(\langle\rangle N\backslash S)/N$
saw: $(\langle\rangle N\backslash S)/N$
signed: $(\langle\rangle N\backslash S)/N$
Suzy: N
that: $[\,]^{-1}[\,]^{-1}(CN\backslash CN)/(\langle\rangle N\backslash S)$
that: $[\,]^{-1}[\,]^{-1}(CN\backslash CN)/(S/N)$
the: N/CN
thinks: $(\langle\rangle N\backslash S)/S$
to: PP/N
waitress: CN
went: $(\langle\rangle N\backslash S)/PP$
without: $[\,]^{-1}((\langle\rangle N\backslash S)\backslash(\langle\rangle N\backslash S))/(\langle\rangle N\backslash S)$
worried: $(\langle\rangle N\backslash S)/N$
yesterday: $(\langle\rangle N\backslash S)\backslash(\langle\rangle N\backslash S)$

Fig. 1. Lexical grammar

2 Inducing Brackets

The general strategy will be to represent symbolic half-brackets in the input by variables for which the flow of information is propagation from endsequent to axiom leaves and instantiation from axiom leaves to endsequent. Thus we need a coding of brackets which represents the terminal yield of sequents in terms of open and closed "symbolic" half-brackets which are uninstantiated bottom up.

Such antecedents are of the pattern:

$$(\text{Onset } \mathbf{Tp} \text{ Offset})^{+}$$

where Onset is of the form [*, **Tp** is a type, and Offset is of the form]*. This formulation builds in Lambek's restriction whereby there must be at least one type in an antecedent and within brackets.

We represent Onsets and Offsets by lists of zeros; the length of the list is the number of brackets. For example, [[[is coded $[0, 0, 0]$,] is coded $[0]$ and no brackets are coded by the empty list [] ($\emptyset$).[6]

A symbolically bracketed antecedent $\varDelta$ is *well-bracketed* (is a BI-**Config**), if it is the case that at every point of $\varDelta$, the sum of all Onsets to the left of this point is greater or equal than the sum of all Offsets to the left of this point, and that for the whole $\varDelta$ these two sums are equal.

In the following section we present the bracket inducing rules.

3 Bracket Inducing Rules

Regarding notation, in bracket inducing BI-**Lb** rules, we use $\varDelta$ (and also $\varDelta_1$, $\varDelta_2$) for sequences starting with an Onset F and ending with an Offset G, and $\varOmega$ for sequences starting and ending with types (in particular, just a singleton type is also an $\varOmega$).

All rules have a premise-to-conclusion property by which all sequents generated have well-bracketed antecedents. Fragments of the antecedent, however, are not necessarily well-bracketed: in a sequent of the form $\varDelta_1, \varDelta, \varDelta_2 \Rightarrow B$ an opening bracket in $\varDelta_1$ could have its corresponding closing bracket in $\varDelta_2$, and in this case $\varDelta_1$ and $\varDelta_2$ are not well-bracketed. Also recall that we are working with a Cut-free version of **Lb**, therefore the Cut rule is not included in the BI-**Lb** calculus.

Identity axiom:

$$\frac{}{\emptyset, P, \emptyset \Rightarrow P}\, id$$

[6] We use this notation because it prefigures a planned future extension to exponentials in which onsets are coded by lists of naturals representing the size of so-called stoups; the zeros in the coding of **Lb** essentially mean that here all stoups are empty.

Bracket modalities:

$$\frac{\Delta_1, F, A, G, \Delta_2 \Rightarrow B}{\Delta_1, F\oplus[0], [\,]^{-1}A, [0]\oplus G, \Delta_2 \Rightarrow B}\,[\,]^{-1}L \qquad \frac{[0]\oplus F, \Omega, G\oplus[0] \Rightarrow B}{F, \Omega, G \Rightarrow [\,]^{-1}B}\,[\,]^{-1}R$$

$$\frac{\Delta_1, F\oplus[0], A, [0]\oplus G, \Delta_2 \Rightarrow B}{\Delta_1, F, \langle\,\rangle A, G, \Delta_2 \Rightarrow B}\,\langle\,\rangle L \qquad \frac{F, \Omega, G \Rightarrow B}{[0]\oplus F, \Omega, G\oplus[0] \Rightarrow \langle\,\rangle B}\,\langle\,\rangle R$$

Lambek connectives: the original Lambek rules should also be modified in order to include the new bracket induction mechanism, as shown below.

$$\frac{\Delta_1, F, A, \emptyset, \emptyset, B, G, \Delta_2 \Rightarrow D}{\Delta_1, F, A\bullet B, G, \Delta_2 \Rightarrow D}\,\bullet L \qquad \frac{\Delta_1 \Rightarrow A \qquad \Delta_2 \Rightarrow B}{\Delta_1, \Delta_2 \Rightarrow A\bullet B}\,\bullet R$$

$$\frac{F_1, \Omega, G_1 \Rightarrow B \qquad \Delta_1, F_2, C, G_2, \Delta_2 \Rightarrow D}{\Delta_1, F_2, C/B, \emptyset, F_1, \Omega, G_1\oplus G_2, \Delta_2 \Rightarrow D}\,/L \qquad \frac{\Delta, \emptyset, B, \emptyset \Rightarrow C}{\Delta \Rightarrow C/B}\,/R$$

$$\frac{F_1, \Omega, G_1 \Rightarrow A \qquad \Delta_1, F_2, C, G_2, \Delta_2 \Rightarrow D}{\Delta_1, F_2\oplus F_1, \Omega, G_1, \emptyset, A\backslash C, G_2, \Delta_2 \Rightarrow D}\,\backslash L \qquad \frac{\emptyset, A, \emptyset, \Delta \Rightarrow C}{\Delta \Rightarrow A\backslash C}\,\backslash R$$

4 Correctness

We define a translation $\# : \textbf{Config} \to \text{BI-}\textbf{Config}$ in terms of injective functions $\#(0^*, 0^*) : \textbf{Config} \to \text{BI-}\textbf{Config}$ as follows:

(7) a. $\#(\Delta) = \#(\emptyset, \emptyset)(\Delta)$
b. $\#(F, G)(P) = F, P, G$
$\#(F, G)([\Delta]) = \#(F\oplus[0], [0]\oplus G)(\Delta)$
$\#(F, G)(\Gamma, \Delta) = \#(F, \emptyset)(\Gamma), \#(\emptyset, G)(\Delta)$

Then we have the following:

(8) **Proposition** (*BI completeness*)

$$\vdash_{\mathbf{Lb}} \Gamma \Rightarrow A \quad \Longrightarrow \quad \vdash_{\text{BI-}\mathbf{Lb}} \#(\Gamma) \Rightarrow A$$

Proof. Trivial induction on derivation in **Lb**. Q.E.D.

(9) **Lemma** (*BI soundness*)

$$\vdash_{\mathbf{Lb}} \Gamma \Rightarrow A \quad \Longleftarrow \quad \vdash_{\text{BI-}\mathbf{Lb}} \#(\Gamma) \Rightarrow A$$

Proof. First we show, by induction on derivation in BI-**Lb**, that if $\vdash_{\text{BI-}\mathbf{Lb}} \Delta \Rightarrow A$, then $\Delta = \#(\Gamma)$ for some $\Gamma \in \textbf{Config}$; in particular, that Δ is well-bracketed in the sense of Section 2. In the only non-trivial induction steps, those of $/L$ and $\backslash L$, the well-bracketedness of the antecedent in the conclusion follows from the fact that the antecedent of the minor (left) premise is contiguous in the antecedent of the conclusion. Second, we prove the soundness lemma itself, also by induction on the derivation of $\#(\Gamma) \Rightarrow A$ in BI-**Lb**. Again, the non-trivial case here is the branching rule, $/L$, for example. Here we use the above consideration to establish the fact that both premises are actually #-translations of some **Lb** sequents. After that, the instance of $/L$ in BI-**Lb** transforms into an instance of the corresponding rule in **Lb**. Q.E.D.

(10) **Theorem** (*BI correctness*)

$$\vdash_{\mathbf{Lb}} \Gamma \Rightarrow A \quad \Longleftrightarrow \quad \vdash_{\text{BI-}\mathbf{Lb}} \#(\Gamma) \Rightarrow A$$

Proof. By BI soundness and BI completeness. Q.E.D.

Notice that throughout this section F_i's and G_i's in BI-**Lb** derivations are *ground terms* of the form $[0, 0, \ldots, 0]$ or $\emptyset$ (representing *constant* natural numbers). Thus, BI-**Lb** is actually just an equivalent formulation of **Lb**, and no real bracket induction is taking place yet. In the next section, we treat F_i's and G_i's as *variables,* whose values are not yet known when we start the proof search.

5 Parsing

5.1 Bracket-Inducing Proof-Search and Parsing

The usual parsing procedure using a categorial grammar works as follows. Using a lexical grammar, such as that of Figure 1, we assign types to words of a string, and these types form the left-hand side of the sequent we are going to derive. The right-hand side is a fixed type, usually primitive, like S for "sentence," for example. If the sequent is derivable, the string is considered valid, and, moreover, we can extract some semantic information from the proof via Curry–Howard correspondence (see [12]). This works perfectly well when antecedents are just linearly ordered lists of formulae (for example, with the "pure" Lambek calculus). With brackets, the situation becomes more involved. Now left-hand sides of sequents also include the bracketing structure. In the naïve generalisation of the Lambek-style parsing procedure to **Lb**-grammars, the bracketing structure should be provided to the parsing algorithm along with the input string and the lexical grammar. For example, since in our lexicon transitive verbs have type $(\langle\rangle N\backslash S)/N$ (the subject forms a weak island), one needs to provide "[John] loves Mary" instead of just "John loves Mary" as the input string for the algorithm. The parser/theorem-prover CatLog3 of Morrill 2017 [15] currently depends on being provided with such bracketing structure in the input. A real natural language string, however, does not come with any brackets. Thus, the more appropriate formulation of the parsing problem involves an existential quantifier over possible bracketings. A string is considered valid if the sequent constructed from the corresponding types is derivable in **Lb** for *some* bracketing.

More formally, a string s is of type A according to a grammar if and only if it has a factorization $s = w_1 + w_2 + \cdots + w_n$ such that $w_1\colon A_1, w_2\colon A_2, \ldots,$ and $w_n\colon A_n$ are in the grammar lexicon and the following is derivable in BI-**Lb**:

(11) $F_1, A_1, G_1, F_2, A_2, G_2, \dots, F_n, A_n, G_n \Rightarrow A$

for *some* values of F_1, G_1, F_2, G_2, $\dots$, F_n, G_n. (Recall that F_i and G_i are natural numbers, but written in the form $[0, 0, \dots, 0]$, linearly bounded by the total length of the sequent.)

Notice that Lambek's non-emptiness restriction is crucial here. It guarantees that every pair of brackets has at least one formula inside, thus brackets are well-organised: in the beginning of the sequent there is $[[\dots[$ (corresponding to F_1), in the end $]\dots]]$ (corresponding to G_n), and between two formulae $]\dots][\dots[$ (corresponding to G_i, F_{i+1}). Without Lambek's restriction, a more weird behaviour is possible. For example, the sequent $s/\langle\,\rangle(p/p) \to s$ becomes derivable, but only with the following bracketing: $s/\langle\,\rangle(p/p), [\,] \to s$, which does not map to a BI sequent of the form (11), and thus would not be found by the algorithm we describe below.

The proof search procedure using bracket induction works as follows. We start with a sequent with no brackets placed, $A_1, \dots, A_n \Rightarrow A$, and insert *variables* for symbolic brackets: $F_1, A_1, G_1, F_2, A_2, G_2, \dots, F_n, A_n, G_n \Rightarrow A$. Then we do proof search from the goal sequent to axiom leaves, annotating each rule application with *side effects*, which are equations on F_i's and G_i's. For each new sequent, we introduce new fresh variables in the places where bracketing is altered, or put $\emptyset$, where the rule postulates that there should be no bracketing. Symbolic brackets in the context are just copied upwards. The rules of BI-**Lb** are annotated with side effects as follows (side effect annotations are placed on the right of the rules):

Identity axiom:

$$\frac{}{F, P, G \Rightarrow P}\ \begin{array}{l} F = \emptyset \\ G = \emptyset \end{array}$$

Rules for bracket modalities:

$$\frac{\Delta_1, F', A, G', \Delta_2 \Rightarrow B}{\Delta_1, F, [\,]^{-1}A, G, \Delta_2 \Rightarrow B}\ \begin{array}{l} F = F' \oplus [0] \\ G = G' \oplus [0] \end{array} \qquad \frac{F', \Omega, G' \Rightarrow B}{F, \Omega, G \Rightarrow [\,]^{-1}B}\ \begin{array}{l} F \oplus [0] = F' \\ G \oplus [0] = G' \end{array}$$

$$\frac{\Delta_1, F', A, G', \Delta_2 \Rightarrow B}{\Delta_1, F, \langle\,\rangle A, G, \Delta_2 \Rightarrow B}\ \begin{array}{l} F \oplus [0] = F' \\ G \oplus [0] = G' \end{array} \qquad \frac{F', \Omega, G' \Rightarrow B}{F, \Omega, G \Rightarrow \langle\,\rangle B}\ \begin{array}{l} F = F' \oplus [0] \\ G = G' \oplus [0] \end{array}$$

Lambek rules:

$$\frac{\Delta_1, F, A, \emptyset, \emptyset, B, G, \Delta_2 \Rightarrow D}{\Delta_1, F, A \bullet B, G, \Delta_2 \Rightarrow D} \qquad \frac{\Delta_1 \Rightarrow A \qquad \Delta_2 \Rightarrow B}{\Delta_1, \Delta_2 \Rightarrow A \bullet B}$$

$$\frac{F_1, \Omega, G_1' \Rightarrow B \qquad \Delta_1, F_2, C, G_2', \Delta_2 \Rightarrow D}{\Delta_1, F_2, C/B, G_1, F_1, \Omega, G_2, \Delta_2 \Rightarrow D}\ \begin{array}{l} G_1 = \emptyset \\ G_2 = G_1' \oplus G_2' \end{array} \qquad \frac{\Delta, \emptyset, B, \emptyset \Rightarrow C}{\Delta \Rightarrow C/B}$$

and symmetrically for $\backslash$. The rules without annotations have no side effects.

The proof search procedure yields a tree which we call *pre-derivation.* In the pre-derivation, instead of constant ground terms (natural numbers) we use variables or $\emptyset$'s. On the other hand, the pre-derivation is annotated by side-effect equations that allow computing the ground terms for symbolic brackets.

The side effect equations are actually very simple: on the left-hand side we have either a term with only one occurrence of a symbolic bracket variable from the conclusion of the rule application, or just a ground term, if there was a $\emptyset$. The right-hand side includes variables from the premises. Thus, once the tree is constructed upto axioms, the algorithm tries to resolve side effects going backwards (from axiom leaves to the goal sequent). For axiom leaves, the symbolic bracket variables receive the zero ($\emptyset$) value, and then we recursively go down. At each step we either evaluate the new variable in the conclusion, or, if there was a $\emptyset$, check whether the right-hand side of the equation (which is already computed) is also $\emptyset$.

Sometimes the side effect equations could be non-satisfiable. For example, for the sequent $P \Rightarrow [\,]^{-1}P$, which is not derivable under any bracket assignment, we have $F_1, P, G_1 \Rightarrow [\,]^{-1}P$, and after applying $[\,]^{-1}R$ (which is the only possible rule here) we get $F_2, P, G_2 \Rightarrow P$ with side effects $F_2 = F_1 \oplus [0]$ and $G_2 = F_1 \oplus [0]$. On the other side, we have $F_2 = G_2 = \emptyset$ from the axiom, which gives a non-satisfiable equation $F_1 \oplus [0] = \emptyset$ (recall that F_i and G_i should always be non-negative integers).

Another, more sophisticated and linguistically relevant example is given in Subsection 5.3.

Therefore, even when a pre-derivation is obtained, we still have to resolve the side-effects; fortunately, this can be done in linear time and does not substantially slow down the proof search process. If resolving of side effects succeeds, variables get replaced with ground terms (natural numbers), obtaining a derivation (in the standard sense) in BI-**Lb**. By Theorem (10), this yields derivability of the original sequent, with some brackets assigned, in **Lb**. If resolving side-effects failed, the algorithm continues the proof search.

Correctness of the bracket-inducing proof search algorithm is justified by the following theorem, whose proof is straightforward.

(12) **Theorem.** For a sequent of the form $F_1, A_1, G_1, \ldots, F_n, A_n, G_n \Rightarrow A$, in which F_i and G_j are variables, the algorithm described above yields all BI-**Lb** derivations of all instances of this sequents with ground terms (natural numbers) substituted for these variables. (In particular, it returns "not derivable" if there are no such derivations.)

5.2 A Positive Example

In this subsection we run our bracket-inducing proof search algorithm on the common noun group in (1a), "man that Mary loves." In order to show that it is of type CN, if we prove the following in the BI-**Lb** calculus:

(13) $$\begin{array}{l} F_1, CN, G_1, F_2, [\,]^{-1}[\,]^{-1}(CN\backslash CN)/(S/N), G_2, F_3, N, G_3, \\ \qquad F_4, (\langle\rangle N\backslash S)/N, G_4 \Rightarrow CN \end{array}$$

The following pre-derivation is annotated with side effects at axiom leaves and rules. The goal sequent (13) above is pre-derived using $/L$ from two sequents with the following pre-derivations (we omit side effects that are trivially satisfied, like $\emptyset = \emptyset \oplus \emptyset$):

$$\dfrac{\dfrac{}{\emptyset, N, \emptyset \Rightarrow N} \qquad \dfrac{\dfrac{\dfrac{}{F_{11}, N, G_9 \Rightarrow N}\;{\scriptstyle F_{11}=\emptyset,\ G_9=\emptyset}}{F_5, N, G_3 \Rightarrow \langle\,\rangle N}\;{\scriptstyle F_5=[0]\oplus F_{11},\ G_3=G_9\oplus[0]} \qquad \dfrac{}{F_6, S, \emptyset \Rightarrow S}\;{\scriptstyle F_6=\emptyset}}{F_3, N, G_3, F_4, \langle\,\rangle N\backslash S, \emptyset \Rightarrow S}\;{\scriptstyle F_4=\emptyset,\ F_3=F_6\oplus F_5}}{\dfrac{F_3, N, G_3, F_4, (\langle\,\rangle N\backslash S)/N, G_5, \emptyset, N, \emptyset \Rightarrow S}{F_3, N, G_3, F_4, (\langle\,\rangle N\backslash S)/N, G_5 \Rightarrow S/N}}\;{\scriptstyle G_5=\emptyset}$$

and

$$\dfrac{\dfrac{\dfrac{\dfrac{}{F_{10}, CN, G_1 \Rightarrow CN}\;{\scriptstyle F_{10}=\emptyset,\ G_1=\emptyset} \qquad \dfrac{}{F_9, CN, G_8 \Rightarrow CN}\;{\scriptstyle F_9=\emptyset,\ G_8=\emptyset}}{F_1, CN, G_1, F_8, CN\backslash CN, G_8 \Rightarrow CN}\;{\scriptstyle F_8=\emptyset,\ F_1=F_{10}\oplus F_9}}{F_1, CN, G_1, F_7, [\,]^{-1}(CN\backslash CN), G_7 \Rightarrow CN}\;{\scriptstyle F_7=F_8\oplus[0],\ G_7=[0]\oplus G_8}}{F_1, CN, G_1, F_2, [\,]^{-1}[\,]^{-1}(CN\backslash CN), G_6 \Rightarrow CN}\;{\scriptstyle F_2=F_7\oplus[0],\ G_6=[0]\oplus G_7}$$

The side-effects for the lowermost application of $/L$, which yields the goal sequent (13), are $G_2 = \emptyset$ and $G_4 = G_5 \oplus G_6$.

Using side effects, the algorithm computes the bracketings from leaves to root as follows:

$$\begin{array}{l}
F_{11} = G_9 = F_6 = F_{10} = G_1 = F_9 = G_8 = \emptyset \\
F_5 = [0] \oplus F_{11} = [0] \oplus \emptyset = [0] \\
G_3 = G_9 \oplus [0] = \emptyset \oplus [0] = [0] \\
F_8 = F_4 = G_5 = \emptyset \\
F_1 = F_{10} \oplus F_9 = \emptyset \oplus \emptyset = \emptyset \\
F_3 = F_6 \oplus F_5 = \emptyset \oplus [0] = [0] \\
F_7 = F_8 \oplus [0] = \emptyset \oplus [0] = [0] \\
G_7 = [0] \oplus G_8 = [0] \oplus \emptyset = [0] \\
F_2 = F_7 \oplus [0] = [0] \oplus [0] = [0,0] \\
G_6 = [0] \oplus G_7 = [0] \oplus [0] = [0,0] \\
G_2 = \emptyset \\
G_4 = G_5 \oplus G_6 = \emptyset \oplus [0,0] = [0,0]
\end{array}$$

and establishes the fact that the following sequent (with ground terms substituted for symbolic bracket variables) is derivable in BI-**Lb**:

$$\emptyset, CN, \emptyset, [0,0], [\,]^{-1}[\,]^{-1}(CN\backslash CN)/(S/N), \emptyset, [0], N, [0], \emptyset, (\langle\,\rangle N\backslash S)/N, [0,0] \Rightarrow CN.$$

This sequent corresponds to the following **Lb**-sequent:

$$CN, [[[\,]^{-1}[\,]^{-1}(CN\backslash CN)/(S/N), [N], (\langle\,\rangle N\backslash S)/N]] \Rightarrow CN$$

and the following bracketing of the CN group: "man [[that [Mary] loves]]."

5.3 A Negative Example

In this section we run the bracket-inducing proof search to invalidate (2a), "the paper that John laughed without reading". In order to make the reasoning shorter, we focus on the central part, namely, we show that the dependent clause "John laughed without reading" is not of type S/N. The natural bracketing for this dependent clause would be "[John] laughed [without reading]" (the subject and the without-clause form weak islands), and one can see that the corresponding sequent, according to the lexicon from Figure 1,

$$[N], \langle\rangle N\backslash S, [\,[\,]^{-1}((\langle\rangle N\backslash S)\backslash(\langle\rangle N\backslash S))/(\langle\rangle N\backslash S), (\langle\rangle N\backslash S)/N\,] \Rightarrow S/N$$

is not derivable in **Lb**, since the N which comes from the right-hand side appears outside the bracketed weak island and cannot penetrate the brackets.

Using our bracket-inducing proof search procedure, we establish a stronger fact that there exists *no* bracketing for which the sequent saying that "John laughed without reading" is of type S/N could be derivable in **Lb**.

We shall do proof search in the BI-**Lb** calculus for the following:

(14) $$\begin{array}{l} F_1, N, G_1, F_2, \langle\rangle N\backslash S, G_2, F_3, [\,]^{-1}((\langle\rangle N\backslash S)\backslash(\langle\rangle N\backslash S))/(\langle\rangle N\backslash S), G_3, \\ \qquad F_4, (\langle\rangle N\backslash S)/N, G_4 \Rightarrow S/N \end{array}$$

and show that it yields no derivation.

The basic idea here is that when the proof search comes to the application of $[\,]^{-1}L$, there is always going to be $\emptyset$ on the right, thus the side effect for $[\,]^{-1}L$ would fail, and the pre-derivation will not become a real BI-**Lb** derivation. Accurate justification of this idea requires exhaustive case analysis, which we perform below.

A direct implementation of the algorithm (see Section 6) would do a complete proof search. Here we make some optimisations. First we notice that the $/R$ rule is invertible, therefore we can apply it immediately:

(15) $$\begin{array}{l} F_1, N, G_1, F_2, \langle\rangle N\backslash S, G_2, F_3, [\,]^{-1}((\langle\rangle N\backslash S)\backslash(\langle\rangle N\backslash S))/(\langle\rangle N\backslash S), G_3, \\ \qquad F_4, (\langle\rangle N\backslash S)/N, G_4, \emptyset, N, \emptyset \Rightarrow S \end{array}$$

Second, we are going to use *count invariants* in order to reduce the number of possible cases to be considered. Count invariants for the Lambek calculus were introduced by van Benthem [2] and then extended to the calculi with brackets and additive connectives [23] and universal exponential [8]. Here we use a very weak form of the count invariant:

(16) **Lemma.** If a sequent is provable in **Lb** (or BI-**Lb**), then each primitive type occurs in it an even number of times.

The proof is a straightforward induction on the derivation of the sequent.

On the top-level, there are three connectives, so we have three cases to consider.

Case 1. Apply $/L$ to the long formula in the center of the sequent (15). Notice that $(\langle\rangle N\backslash S)/N$ and the rightmost N should both go to the left premise, since otherwise it would violate the count invariant and therefore be *a priori* not derivable. The side effect here is $G_3 = \emptyset$, and the right premise is as follows

$$F_1, N, G_1, F_2, \langle\rangle N\backslash S, G_2, F_3, [\,]^{-1}((\langle\rangle N\backslash S)\backslash(\langle\rangle N\backslash S)), \emptyset \Rightarrow S.$$

For this very sequent, further proof search fails, since application of $[\,]^{-1}L$ is not possible, neither immediately, nor after applying $\backslash L$ to the formula on the left, due to the rightmost $\emptyset$.

Case 2. Apply $/L$ to the rightmost $/$. This yields a side effect $G_4 = \emptyset$ and the (interesting) right premise is as follows:

$$F_1, N, G_1, F_2, \langle\,\rangle N\backslash S, G_2, F_3, [\,]^{-1}((\langle\,\rangle N\backslash S)\backslash(\langle\,\rangle N\backslash S))/(\langle\,\rangle N\backslash S),\\ G_3, F_4, \langle\,\rangle N\backslash S, \emptyset \Rightarrow S$$

Applying $/L$ to the central formula yields, as the right premise,

$$F_1, N, G_1, F_2, \langle\,\rangle N\backslash S, G_2, F_3, [\,]^{-1}((\langle\,\rangle N\backslash S)\backslash(\langle\,\rangle N\backslash S)), \emptyset \Rightarrow S$$

and again the $\emptyset$ on the right violates the side condition for $[\,]^{-1}L$, whenever it gets applied.

Applying $\backslash L$ to the right formula makes further derivation impossible, since, due to Lambek's restriction, after that there will be no way to decompose the central formula.

Finally, applying $\backslash L$ to the left formula gives

$$F_1', S, G_2, F_3, [\,]^{-1}((\langle\,\rangle N\backslash S)\backslash(\langle\,\rangle N\backslash S))/(\langle\,\rangle N\backslash S), G_3, F_4, \langle\,\rangle N\backslash S, \emptyset \Rightarrow S$$

and we do the same case analysis as above.

Case 3. Apply $\backslash L$ to the leftmost occurrence of $\backslash$. The right premise is as follows:

$$F_1', S, G_2, F_3, [\,]^{-1}((\langle\,\rangle N\backslash S)\backslash(\langle\,\rangle N\backslash S))/(\langle\,\rangle N\backslash S),\\ G_3, F_4, (\langle\,\rangle N\backslash S)/N, G_4, \emptyset, N, \emptyset \Rightarrow S$$

and we proceed similarly to Cases 1 and 2.

This analysis shows that even if the proof search procedure for (14) successfully finishes at axiom links, resolving side effects fails when it comes across the application of $[\,]^{-1}L$ due to $\emptyset$ on the right. Therefore, "the paper that John laughed without reading" does not receive type S/N for any bracketing.

6 Complexity Estimations

6.1 Bracket Induction vs. Generate-and-Test Brackets

The proof search algorithm with side-effects presented in the previous section, still has exponential running time. In general, this is inevitable, due to the NP-hardness of the original Lambek calculus [22]. However, the proof search with bracket induction has a significant advantage in speed compared to a naïve approach where the algorithm searches for all possible bracketings by brute force and does proof search independently for each sequent obtained in this way. More precisely, non-determinism in parsing with **Lb**-based categorial grammar comes from three sources:

1. non-unique type assignment (a lexical item can have several different types),

2. bracketing,
3. proof search.

In the bracket induction approach presented in the present paper, the second source above is handled together with the third one. Thus, our algorithm is still exponential, but is also exponentially faster than the naïve one. This makes bracket induction applicable in practice, while attempts to implement brute force bracket guessing fail to parse even simple sentences in reasonable time. An implementation of our parsing algorithm in Prolog, written by the first author, with an example and runtime log showing execution times are available on GitHub: `https://github.com/skuzn/BI-Lb`

The lower exponential bound on the running time of the bracket-inducing proof search algorithm in the present paper comes from the fact that in the Lambek calculus, even without brackets, there exist examples of sequents with exponentially many derivations: $P/P, \dots, P/P, P, P\backslash P, \dots, P\backslash P \Rightarrow P$. Thus, if we want to yield all possible derivations (parsings) for a sentence, even the output length could be exponential, unless we represent it in a compressed way, as in [21, 5]. The proof search algorithm yields all possible (pre-)derivations in an uncompressed form, and therefore has exponential worst case running time.

6.2 Pseudo-polynomial Approaches

The more subtle question is the comparison of the bracket-inducing parsing procedure presented here with the pseudo-polynomial algorithm for **Lb** presented in our earlier paper [5].

Despite the Lambek calculus being NP-hard [22], Pentus [21] noticed that the complexity essentially comes from complicated types used in the lexicon. He presented a polynomial-time parsing algorithm [21] for Lambek grammars where complexity of all types in the lexicon is bounded. More precisely, the running time of Pentus' algorithm is a polynomial of n and 2^d, where n is the length of the input and d is a complexity measure of types. For simplicity, one can think of d as just the maximal size of a type in the lexicon. Pentus' algorithm is based on proof nets and dynamic programming.

In [5], using the method of Pentus [21], we have presented an algorithm for checking derivability in the Lambek calculus with brackets[7]. However, unlike Pentus' algorithm, our algorithm in [5] is only a theorem-prover, not a parser. That is, the algorithm from [5] does not account for lexical ambiguity, where several types are assigned to one word. Adding this extra level of non-determinism could make running time exponential.

Another, more serious issue is connected with the deep nesting of brackets. The time complexity estimation in [5] is a polynomial of n, 2^d, and n^b, where n is the input length, d is the complexity measure of types in the lexicon, and b is the maximum depth of nested brackets. Thus, the algorithm would run in

[7] In contrast to the present paper, the calculus in [5] allows empty antecedents, but imposing the Lambek's restriction there is quite straightforward.

polynomial time only if both d and b are bound by constants. Unfortunately, in linguistic practice this holds for d, but not for b.

The counter-examples come from well-known phrases with nested dependent clauses, like "the dog that worried the cat that killed the rat that ate the malt that lay in the house that Jack built." The natural bracketing here is as follows (dependent clauses form strong islands, and the subject 'Jack' is a weak one): "the dog [[that worried the cat [[that killed the rat [[that ate the malt [[that lay in the house [[that [Jack] built]]]]]]]]]]." Here b is linear w.r.t. input length ($b = \alpha n$ for some constant α), which yields exponential ($\geq n^{\alpha n}$) running time for the algorithm from [5].

There also exists a shallow bracketing for this phrase: "... the cat [[that killed the rat]] [[that ate the malt]] ..." Parsing with this shallow bracketing, however, yields another reading: "the cat ate the malt" rather than the more natural "the rat ate the malt." Thus, if we restrict our algorithm by imposing a constant bound on the value of b, we can still justify the phrase as grammatically correct, but we lose some of its readings, which is undesired.

The b parameter being linear w.r.t. n, the algorithm from [5] runs exponentially, as does the algorithm presented in the present paper. The advantage of this latter is that it does not require bracketing to be passed as an input along with the sentence itself.

The question whether there exists an algorithm for **Lb** with the running time being a polynomial of n and 2^d (without n^b in the complexity estimation) is an open problem.

7 Future Work

In order to make the presentation as clear as possible, in this paper we have discussed bracket induction on a very small fragment of type-logical grammar, based on the pure Lambek calculus augmented with brackets and bracket modalities. In the future we are planning to extend this approach to broader calculi, including additive connectives [4], discontinuous operations [16, 19], and the (sub)exponential modality for medial and parasitic extraction [17]. For the latter, the whole calculus is undecidable [7], so proof search is possible only in a restricted fragment [17, 7]. Moreover, we are planning to optimise parsing with bracket induction using count invariant heuristics [2, 23, 8] and focusing techniques [1, 10, 18, 6], with necessary modifications for the BI calculi.

Implementing bracket induction in CatLog would allow the system to process raw sentences in natural language, not asking the user for extra structural information (bracketing). Being almost as effective as standard proof search, the proof search procedure with bracket induction would not slow down the parsing workflow. Unfortunately, the running time is still exponential. In the previous section we have discussed why the pseudo-polynomial algorithm for the Lambek calculus with brackets presented in [5] is still not enough to build a polynomial-time version of CatLog. The interesting open question here is whether there is an algorithm for parsing in **Lb** with polynomial runtime for bounded type complexity but unbounded bracket nesting depth, or there is NP-hardness arising from deeply nested brackets even with shallow types.

Acknowledgments

We would like to thank the anonymous referees for their thoughtful comments and questions.

The research of Morrill was supported by the grant TIN2017-89244-R from MINECO (Ministerio de Economia, Industria y Competitividad). Glyn Morrill is also grateful to the University of Pennsylvania for support during his visit in February 2017. Kuznetsov's research towards this paper was supported by the Young Russian Mathematics award, by the Program of the Presidium of the Russian Academy of Sciences No. 01 'Fundamental Mathematics and its Applications' under grant PRAS-18-01, and by the Russian Foundation for Basic Research under grant 18-01-00822. Stepan Kuznetsov is also grateful to the University of Pennsylvania for support during his visit in April–May 2018, when the final version of this paper was prepared. Max Kanovich is grateful to the University of Pennsylvania for support during his visit in February 2017. The participation of Kanovich, Kuznetsov, and Scedrov in the preparation of this article was within the framework of the Basic Research Program at the National Research University Higher School of Economics (HSE) and supported within the framework of a subsidy by the Russian Academic Excellence Project '5-100'.

References

1. Andreoli, J.M.: Logic programming with focusing in linear logic. Journal of Logic and Computation 2(3), 297–347 (1992)
2. van Benthem, J.: Language in Action: Categories, Lambdas, and Dynamic Logic. No. 130 in Studies in Logic and the Foundations of Mathematics, North-Holland, Amsterdam (1991), revised student edition printed in 1995 by the MIT Press
3. Girard, J.Y.: Linear logic. Theoretical Computer Science 50, 1–102 (1987), DOI: 10.1016/0304-3975(87)90045-4
4. Kanazawa, M.: The Lambek calculus enriched with additional connectives. Journal of Logic, Language and Information 1, 141–171 (1992), DOI: 10.1007/BF00171695
5. Kanovich, M., Kuznetsov, S., Morrill, G., Scedrov, A.: A polynomial-time algorithm for the Lambek calculus with brackets of bounded order. In: Miller, D. (ed.) Proceedings of the 2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017). Leibniz International Proceedings in Informatics, LIPIcs, vol. 84, pp. 22:1–22:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany (2017), URL: http://drops.dagstuhl.de/opus/volltexte/2017/7738/, DOI: 10.4230/LIPIcs.FSCD.2017.22
6. Kanovich, M., Kuznetsov, S., Nigam, V., Scedrov, A.: A logical framework with commutative and non-commutative subexponentials. In: Automated Reasoning: Proceedings of IJCAR 2018. Springer (2018), to appear
7. Kanovich, M., Kuznetsov, S., Scedrov, A.: Undecidability of the Lambek calculus extended with subexponential and bracket modalities. In: Klasing, R., Zeitoun, M.

(eds.) FCT 2017: Fundamentals of Computation Theory. LNCS, vol. 10472, pp. 326–340. Springer, Berlin (2017), DOI: 10.1007/978-3-662-55751-8_26

8. Kuznetsov, S., Morrill, G., Valentín, O.: Count-invariance including exponentials. In: Kanazawa, M., de Groote, P., Sadrzadeh, M. (eds.) Proceedings of the 15th Meeting on the Mathematics of Language. pp. 128–139. Association for Computational Linguistics, London (2017), https://aclweb.org/anthology/W/W17/W17-3413.pdf
9. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly 65, 154–170 (1958)
10. Miller, D., Saurin, A.: From proofs to focused proofs: A modular proof of focalization in linear logic. In: CSL 2007: Computer Science Logic. Lecture Notes in Computer Science, vol. 4646, pp. 405–419. Springer (2007), DOI: 10.1007/978-3-540-74915-8_31
11. Moortgat, M.: Multimodal linguistic inference. Journal of Logic, Language, and Information 5(3, 4), 349–385 (1996), DOI: 10.1007/BF00159344
12. Moortgat, M.: Categorial Type Logics. In: van Benthem, J., ter Meulen, A. (eds.) Handbook of Logic and Language, pp. 93–177. Elsevier Science B.V. and the MIT Press, Amsterdam and Cambridge, Massachusetts (1997)
13. Morrill, G.: Categorial Formalisation of Relativisation: Pied Piping, Islands, and Extraction Sites. Tech. Rep. LSI-92-23-R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya (1992)
14. Morrill, G.: Grammar logicised: relativisation. Linguistics and Philosophy 40(2), 119–163 (2017), DOI: 10.1007/s10988-016-9197-0
15. Morrill, G.: Parsing logical grammar: CatLog3. In: Loukanova, R., Liefke, K. (eds.) Proceedings of the Workshop on Logic and Algorithms in Computational Linguistics 2017, LACompLing2017. pp. 107–131. DiVA, Stockholm University (2017), http://su.diva-portal.org/smash/get/diva2:1140018/FULLTEXT03.pdf
16. Morrill, G., Valentín, O.: Displacement Calculus. Linguistic Analysis 36(1–4), 167–192 (2010), http://arxiv.org/abs/1004.4181, special issue Festschrift for J. Lambek
17. Morrill, G., Valentín, O.: Computational Coverage of TLG: Nonlinearity. In: Kanazawa, M., Moss, L., de Paiva, V. (eds.) Proceedings of NLCS'15. Third Workshop on Natural Language and Computer Science. EPiC, vol. 32, pp. 51–63. Kyoto (2015), Workshop affiliated with Automata, Languages and Programming (ICALP) and Logic in Computer Science (LICS)
18. Morrill, G., Valentín, O.: Multiplicative-Additive Focusing for Parsing as Deduction. In: Cervesato, I., Schürmann, C. (eds.) First International Workshop on Focusing, workshop affiliated with LPAR 2015. pp. 29–54. No. 197 in EPTCS, Suva, Fiji (2015)
19. Morrill, G., Valentín, O., Fadda, M.: The Displacement Calculus. Journal of Logic, Language and Information 20(1), 1–48 (2011), DOI: 10.1007/s10849-010-9129-2
20. Morrill, G.V.: Categorial Grammar: Logical Syntax, Semantics, and Processing. Oxford University Press, New York and Oxford (2011)
21. Pentus, M.: A polynomial-time algorithm for Lambek grammars of bounded order. Linguistic Analysis 36(1–4), 44–471 (2010)
22. Pentus, M.: Lambek calculus is NP-complete. Theoretical Computer Science 357(1), 186–201 (2006), DOI: 10.1016/j.tcs.2006.03.018
23. Valentín, O., Serret, D., Morrill, G.: A count invariant for Lambek calculus with additives and bracket modalities. In: Morrill, G., Nederhof, M.J. (eds.) Proceedings of Formal Grammar 2012 and 2013. Springer LNCS, FoLLI Publications in Logic, Language and Information, vol. 8036, pp. 263–276. Springer, Berlin (2013), DOI: 10.1007/978-3-642-39998-5_17