A Polynomial-Time Algorithm for the Lambek Calculus with Brackets of Bounded Order[*][†]

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

1   National Research University Higher School of Economics, Moscow, Russia
     mkanovich@hse.ru
2   Steklov Mathematical Institute of RAS, Moscow, Russia; and
     National Research University Higher School of Economics, Moscow, Russia
     sk@mi.ras.ru
3   Universitat Politècnica de Catalunya, Barcelona, Spain
     morrill@cs.upc.edu
4   University of Pennsylvania, Philadelphia, USA; and
     National Research University Higher School of Economics, Moscow, Russia
     scedrov@math.upenn.edu

―― **Abstract** ――

Lambek calculus is a logical foundation of categorial grammar, a linguistic paradigm of grammar as logic and parsing as deduction. Pentus (2010) gave a polynomial-time algorithm for determining provability of bounded depth formulas in $\mathbf{L}^*$, the Lambek calculus with empty antecedents allowed. Pentus' algorithm is based on tabularisation of proof nets. Lambek calculus with brackets is a conservative extension of Lambek calculus with bracket modalities, suitable for the modeling of syntactical domains. In this paper we give an algorithm for provability in $\mathbf{Lb}^*$, the Lambek calculus with brackets allowing empty antecedents. Our algorithm runs in polynomial time when both the formula depth and the bracket nesting depth are bounded. It combines a Pentus-style tabularisation of proof nets with an automata-theoretic treatment of bracketing.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases Lambek calculus, proof nets, Lambek calculus with brackets, categorial grammar, polynomial algorithm

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.22

1   Introduction

The calculus $\mathbf{L}$ of Lambek [23] is a logic of strings. It is retrospectively recognisable as the multiplicative fragment of non-commutative intuitionistic linear logic without empty antecedents; the calculus $\mathbf{L}^*$ is like $\mathbf{L}$ except that it admits empty antecedents. The Lambek calculus is the foundation of categorial grammar, a linguistic paradigm of grammar as

―――――――

logic and parsing as deduction; see for instance Buszkowski [6], Carpenter [7], Jäger [17], Morrill [28], Moot and Retoré [26]. For example, the sentence "*John knows Mary likes Bill*" can be analysed as grammatical because $N, (N \setminus S) / S, N, (N \setminus S) / N, N \to S$ is a theorem of Lambek calculus. Here N stands for *noun phrase,* S stands for *sentence,* and syntactic categories for other words are built from these two primitive ones using division operations. For example, $(N \setminus S) / N$ takes noun phrases on both sides and yields a sentence, thus being the category of *transitive verb.*

Categorial grammar, that started from works of Ajdukiewicz [4] and Bar-Hillel [5], aspires to practice linguistics to the standards of mathematical logic; for example, Lambek [23] proves cut-elimination, that yields the subformula property, decidability, the finite reading property, and the focalisation property. In a remarkable series of works Mati Pentus has proved the main metatheoretical results for Lambek calculus: equivalence to context free grammars [31]; completeness w.r.t. language models [32][33]; NP-completeness [34]; a polynomial-time algorithm for checking provability of formulae of bounded order in $\mathbf{L}^*$ [35]. The Lambek calculus with only one division operation (and without product) is decidable in polynomial time (Savateev [36]).

The Lambek calculus with brackets **Lb** (Morrill 1992 [27]; Moortgat 1995 [25]) is a logic of bracketed strings which is a conservative extension of Lambek calculus with bracket modalities the rules for which are conditioned on metasyntactic brackets. In this paper we consider a variant of **Lb** that allows empty antecedents, denoted by **Lb***.

The syntax of **Lb*** is more involved than the syntax of the original Lambek calculus. In **L**, the antecedent (left-hand side) of a sequent is just a linearly ordered sequence of formulae. In **Lb***, it is a structure called *configuration,* or *meta-formula.* Meta-formulae are built from formulae, or *types,* as they are called in categorial grammar, using two metasyntactic constructors: comma and brackets. The succedent (right-hand side) of a sequent is one type. Types, in turn, are built from variables, or *primitive types,* $p_1, p_2, \dots$, using the three binary connectives of Lambek, $\setminus$, $/$, and $\cdot$, and two unary ones, $\langle \rangle$ and $[]^{-1}$, that operate brackets. Axioms of **Lb*** are $p_i \to p_i$, and the rules are as follows:

$$\frac{\Pi \to A \quad \Delta(B) \to C}{\Delta(\Pi, A \setminus B) \to C} \; (\setminus \to) \qquad \frac{A, \Pi \to B}{\Pi \to A \setminus B} \; (\to \setminus) \qquad \frac{\Gamma(A, B) \to C}{\Gamma(A \cdot B) \to C} \; (\cdot \to)$$

$$\frac{\Pi \to A \quad \Delta(B) \to C}{\Delta(B / A, \Pi) \to C} \; (/ \to) \qquad \frac{\Pi, A \to B}{\Pi \to B / A} \; (\to /) \qquad \frac{\Gamma \to A \quad \Delta \to B}{\Gamma, \Delta \to A \cdot B} \; (\to \cdot)$$

$$\frac{\Delta([A]) \to C}{\Delta(\langle \rangle A) \to C} \; (\langle \rangle \to) \quad \frac{\Pi \to A}{[\Pi] \to \langle \rangle A} \; (\to \langle \rangle) \quad \frac{\Delta(A) \to C}{\Delta([[]^{-1}A]) \to C} \; ([]^{-1} \to) \quad \frac{[\Pi] \to A}{\Pi \to []^{-1}A} \; (\to []^{-1})$$

Cut-elimination is proved in Moortgat [25]. The Lambek calculus with brackets permits the characterisation of syntactic domains in addition to word order. By way of linguistic example, consider how a relative pronoun type assignment $(CN \setminus CN) / (S / N)$ (here CN is *one* primitive type, corresponding to *common noun:* e.g., "*book*", as opposed to noun phrase "*the book*") allows unbounded relativisation by associative assembly of the body of relative clauses:

   (a)   *man who Mary likes*
   (b)   *man who John knows Mary likes*
   (c)   *man who Mary knows John knows Mary likes*   . . .

Thus, (b) is generated because the following is a theorem in the pure Lambek calculus:

$$CN, (CN \setminus CN) / (S / N), N, (N \setminus S) / S, N, (N \setminus S) / N \to CN$$

Consider also, however, the following example: *book which John laughed without reading,* where * indicates that this example is not grammatical. In the original Lambek calculus this ungrammatical example is generated, but in Lambek calculus with brackets its ungrammaticality can be characterised by assigning the adverbial preposition a type $[]^{-1}((N \backslash S) \backslash (N \backslash S)) / (N \backslash S)$ blocking this phrase because the following is not a theorem in Lambek calculus with brackets:

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

where the $[]^{-1}$ engenders brackets which block the associative assembly of the body of the relative clause. Another example of islands is provided by the "and" ("or") construction: *girl whom John loves Mary and Peter loves.*

Jäger [16] claims to prove the context free equivalence of **Lb** grammar on the basis of a translation from **Lb** to **L** due to Michael Moortgat's student Koen Versmissen [37]. However, contrary to Versmissen the translation is not an embedding translation (Fadda and Morrill [11], p. 124). We present the counter-example in the end of Section 3. Consequently the result of Jäger is in doubt: the context free equivalence theorem might be correct, but the proof of Jäger, resting on the Versmissen translation, is not correct.

Pentus [35], following on Aarts [1], presents an algorithm for provability in $\mathbf{L}^*$ based on tabularisation (memoisation) of proof nets. This algorithm runs in polynomial time, if the order of the sequent is bounded. An algorithm of the same kind was also developed by Fowler [12][13] for the Lambek calculus without product. For the unbounded case, the derivability problem for $\mathbf{L}^*$ is in the NP class and is NP-complete [34]. Also, non-associative Lambek calculus [24] can be embedded into the Lambek calculus with brackets (Kurtonina [19]). De Groote [10], following on Aarts and Trautwein [2], showed polynomial-time decidability of the non-associative Lambek calculus. The Lambek calculus with bracket modalities, $\mathbf{Lb}^*$, includes as subsystems both non-associative and associative Lambek calculi.

In this paper we provide a Pentus-style algorithm for $\mathbf{Lb}^*$ provability using (1) the proof nets for Lambek calculus with brackets of Fadda and Morrill [11] which are based on a correction of the Versmissen translation, and (2) an automata-theoretic argument. Again, for the unbounded case, $\mathbf{Lb}^*$ is NP-hard (since it contains $\mathbf{L}^*$ as a conservative fragment), and also belongs to the NP class, since the size of a cut-free derivation in $\mathbf{Lb}^*$ is linearly bounded by the size of the goal sequent.

The rest of this paper is organised as follows. In Section 2 we define complexity parameters and formulate the main result. Section 3 contains the formulation and proof of a graph-theoretic provability criterion for $\mathbf{Lb}^*$, known as proof nets. In Section 4 we introduce some more convenient complexity parameters and show their polynomial equivalence to the old ones. Section 5 is the central one, containing the description of our algorithm. In order to make this paper self-contained, in Section 6 we give a detailed explanation of Pentus' construction [35], since it is crucial for our algorithm to work. Finally, in Section 7 we discuss directions of future research in this field.

2   **The Main Result**

For a sequent $\Gamma \to C$ we consider the following three *complexity parameters.* The first one is the *size* of the sequent, $\|\Gamma \to C\|$, counted as the total number of variables and logical symbols in it, including brackets.

▶ **Definition 1.** The size of a formula, meta-formula, or sequent in $\mathbf{Lb}^*$ is defined recursively as follows: $\|p_i\| = 0$; $\|A \cdot B\| = \|A \backslash B\| = \|B / A\| = \|A\| + \|B\| + 1$; $\|\langle\rangle A\| = \|[]^{-1} A\| = \|A\| + 1$; $\|\Lambda\| = 0$; $\|\Gamma, \Delta\| = \|\Gamma\| + \|\Delta\|$; $\|[\Gamma]\| = \|\Gamma\| + 2$; $\|\Gamma \to C\| = \|\Gamma\| + \|C\|$.

The second parameter is the *order.*

▶ **Definition 2.** For any formula A let $\mathrm{prod}(A)$ be 1 if A is of the form $A_1 \cdot A_2$ or $\langle\rangle A_1$, and 0 if not. The order of a formula, meta-formula, or sequent in $\mathbf{Lb}^*$ is defined recursively: $\mathrm{ord}(p_i) = 0$; $\mathrm{ord}(A \cdot B) = \max\{\mathrm{ord}(A), \mathrm{ord}(B)\}$; $\mathrm{ord}(A \backslash B) = \mathrm{ord}(B \,/\, A) = \max\{\mathrm{ord}(A) + 1, \mathrm{ord}(B) + \mathrm{prod}(B)\}$; $\mathrm{ord}(\langle\rangle A) = \mathrm{ord}(A)$; $\mathrm{ord}([]^{-1}A) = \max\{\mathrm{ord}(A) + \mathrm{prod}(A), 1\}$; $\mathrm{ord}(\Lambda) = 0$; $\mathrm{ord}(\Gamma, \Delta) = \max\{\mathrm{ord}(\Gamma), \mathrm{ord}(\Delta)\}$; $\mathrm{ord}([\Gamma]) = \mathrm{ord}(\Gamma)$; $\mathrm{ord}(\Gamma \to C) = \max\{\mathrm{ord}(\Gamma) + 1, \mathrm{ord}(C) + \mathrm{prod}(C)\}$.

For sequents without $\cdot$ and $[]^{-1}$, this definition is quite intuitive: the order is the nesting depth of implications ($\backslash$, $/$, and finally $\to$) and $\langle\rangle$ modalities. With $\cdot$, we also count *alternations* between divisions and multiplications: for example, in $p_1 \backslash (p_2 \cdot (p_3 \backslash (p_4 \cdot \ldots p_k) \ldots))$ implications are not nested, but the order grows linearly. On the other hand, the order is always bounded by a simpler complexity parameter, the maximal height of the syntactic tree. Also, linguistic applications make use of syntactic types of small, constantly bounded order.

The third parameter is the *bracket nesting depth*.

▶ **Definition 3.** The bracket nesting depth of a formula, meta-formula, or a sequent in $\mathbf{Lb}^*$ is defined recursively as follows: $\mathrm{b}(p_i) = 0$; $\mathrm{b}(A \,/\, B) = \mathrm{b}(B \backslash A) = \mathrm{b}(A \cdot B) = \max\{\mathrm{b}(A), \mathrm{b}(B)\}$; $\mathrm{b}(\langle\rangle A) = \mathrm{b}([]^{-1}A) = \mathrm{b}(A) + 1$; $\mathrm{b}(\Lambda) = 0$; $\mathrm{b}(\Gamma, \Delta) = \max\{\mathrm{b}(\Gamma), \mathrm{b}(\Delta)\}$; $\mathrm{b}([\Gamma]) = \mathrm{b}(\Gamma) + 1$; $\mathrm{b}(\Gamma \to C) = \max\{\mathrm{b}(\Gamma), \mathrm{b}(C)\}$.

By $\mathrm{poly}(x_1, x_2, \ldots)$ we denote a value that is bounded by a polynomial of x_1, x_2, $\ldots$

▶ **Theorem 4.** *There exists an algorithm that decides whether a sequent $\Gamma \to C$ is derivable in $\mathbf{Lb}^*$ in $\mathrm{poly}(N, 2^R, N^B)$ time, where $N = \|\Gamma \to C\|$, $R = \mathrm{ord}(\Gamma \to C)$, $B = \mathrm{b}(\Gamma \to C)$.*

If the depth parameters, R and B, are fixed, the working time of the algorithm is polynomial w.r.t. N. However, the dependence on the depth parameters is exponential.

3 Proof Nets

In this section we formulate and prove a graph-theoretic criterion for derivability in $\mathbf{Lb}^*$. A sequent is derivable if and only if there exists a *proof net*, that is, a graph satisfying certain *correctness conditions.*

For each variable p_i we introduce two *literals,* p_i and $\bar{p}_i$, and also four literals, $[$, $]$, $\bar{[}$, and $\bar{]}$ for brackets. Next we define two translations (positive, A^+, and negative, A^-) of $\mathbf{Lb}^*$-formulae into expressions built from literals using two connectives, $⅋$ and $\otimes$.

▶ **Definition 5.**

$$p_i^+ = p_i, \qquad\qquad p_i^- = \bar{p}_i,$$
$$(A \cdot B)^+ = A^+ \otimes B^+, \qquad\qquad (A \cdot B)^- = B^- ⅋ A^-,$$
$$(A \backslash B)^+ = A^- ⅋ B^+, \qquad\qquad (A \backslash B)^- = B^- \otimes A^+,$$
$$(B \,/\, A)^+ = B^+ ⅋ A^-, \qquad\qquad (B \,/\, A)^- = A^+ \otimes B^-,$$
$$(\langle\rangle A)^+ =] \otimes A^+ \otimes [, \qquad\qquad (\langle\rangle A)^- = \bar{[} ⅋ A^- ⅋ \bar{]},$$
$$([]^{-1}A)^+ = \bar{]} ⅋ A^+ ⅋ \bar{[}; \qquad\qquad ([]^{-1}A)^- = [\otimes A^- \otimes].$$

For meta-formulae, we need only the negative translation. In this translation we use an extra connective, $\diamond$, which serves as a metasyntactic version of $⅋$ (just as the comma is a metasyntactic product in the sequent calculus for $\mathbf{Lb}^*$).

▶ **Definition 6.** $(\Gamma, \Delta)^- = \Delta^- \diamond \Gamma^-; \quad [\Gamma]^- = \bar{[} \diamond \Gamma^- \diamond \bar{]}.$

Finally, a sequent $\Gamma \to C$ is translated as $\diamond\, \Gamma^- \diamond\, C^+$ (or as $\diamond\, C^+$ if Γ is empty).

Essentially, this is an extension of Pentus' translation of $\mathbf{L}^*$ into cyclic multiplicative linear logic (**CMLL**) [33][35]. In this paper, for the sake of simpicity, we don't introduce an intermediate calculus that extends **CMLL** with brackets, and we formulate the proof net criterion directly for $\mathbf{Lb}^*$.

Denote the set of all literal and connective *occurrences* in this translation by $\Omega_{\Gamma \to C}$. These occurrences are linearly ordered; connectives and literals alternate. The total number of occurrences is $2n$. Denote the literal occurrences (in their order) by $\ell_1, \ldots, \ell_n$ and the connective occurrences by $c_1, \ldots, c_n$.

▶ **Definition 7.** The *dominance* relation on the occurrences of $\wp$ and $\otimes$, denoted by $\prec$, is defined as follows: for a subexpression of the form $A \wp B$ or $A \otimes B$ if the occurrence of the central connective is c_i, then for any c_j inside A or B we declare $c_j \prec c_i$.

We assume that $\wp$'s that come from translations of bracket modalities associate to the left and $\otimes$'s associate to the right (this choice is arbitrary). Thus, in a pair of such $\wp$'s the right one dominates the left one in the syntactic tree, and the left one dominates the principal connective of A (if A has one, *i.e.,* it is not a literal); symmetrically for $\otimes$.

The other two relations are the *sisterhood* relations on bracket literals and connectives, $\mathcal{S}_b$ and $\mathcal{S}_c$ respectively. Both relations are symmetric.

▶ **Definition 8.** The bracket sisterhood relation, $\mathcal{S}_b$, connects pairs of occurrences of [and] or $\bar{[}$ and $\bar{]}$ that come from the same $\langle\rangle A$, $[]^{-1}A$, or $[\Gamma]$. The connective sisterhood relation, $\mathcal{S}_c$, connects pairs of occurrences of $\otimes$, $\wp$, or $\diamond$ that come from the same $\langle\rangle A$, $[]^{-1}A$, or $[\Gamma]$. Occurrences connected by one of the sisterhood relations will be called *sister* occurrences.

▶ **Definition 9.** A *proof structure* $\mathcal{E}$ is a symmetric relation on the set of literal occurrences $(\{\ell_1, \ldots, \ell_n\})$ such that each occurrence is connected by $\mathcal{E}$ to exactly one occurrence, and each occurrence of a literal q, where q is a variable, [, or], is connected to an occurrence of $\bar{q}$.

▶ **Definition 10.** A proof structure $\mathcal{E}$ is *planar,* iff its edges can be drawn in a semiplane without intersection while the literal occurrences are located on the border of this semiplane in their order $(\ell_1, \ldots, \ell_n)$.

Edges of a planar proof structure divide the upper semiplane into *regions.* The number of regions is $\frac{n}{2} + 1$ (the outermost, infinite region also counts).

▶ **Definition 11.** A planar proof structure $\mathcal{E}$ is a *proof net,* iff it satisfies two conditions.
1. On the border of each region there should be exactly one occurrence of $\wp$ or $\diamond$.
2. Define an oriented graph $\mathcal{A}$ that connects each occurrence of $\otimes$ to the unique occurrence of $\wp$ or $\diamond$ located in the same region. The graph $\mathcal{A} \cup \prec$ should be acyclic.

By definition, edges of $\mathcal{E}$ and $\mathcal{A}$ in a proof net do not intersect, in other words, the graph $\mathcal{E} \cup \mathcal{A}$ is also planar. We can also consider proof structures and proof nets on expressions that are not translations of $\mathbf{Lb}^*$ sequents. For example, proof nets allow *cyclic permutations:* a proof net for $\diamond\, \gamma_1 \diamond\, \gamma_2$ can be transformed into a proof net for $\diamond\, \gamma_2 \diamond\, \gamma_1$ (the $\prec$ relation in γ_1 and γ_2 is preserved).

▶ **Definition 12.** A proof structure $\mathcal{E}$ *respects sisterhood,* iff the following condition holds: if $\langle \ell_i, \ell_{i'} \rangle \in \mathcal{E}$, $\langle \ell_i, \ell_j \rangle \in \mathcal{S}_b$, and $\langle \ell_{i'}, \ell_{j'} \rangle \in \mathcal{S}_b$, then $\langle \ell_j, \ell_{j'} \rangle \in \mathcal{E}$ (i.e., sister brackets are connected to sister brackets).

Before continuing, we consider an **example** of a proof net for a sequent with linguistic meaning. According to [28], the sentence *"Mary danced before singing"* gets the following bracketing: *"[Mary] danced [before singing]"* and the following type assignment:

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

This sequent is derivable in **Lb***, and we justify it by presenting a proof net. This is achieved by translating the sequent into a string of literals according to Definition 5 and drawing an appropriate $\mathcal{E}$ graph that satisfies all the conditions for being a proof net (Definition 11):
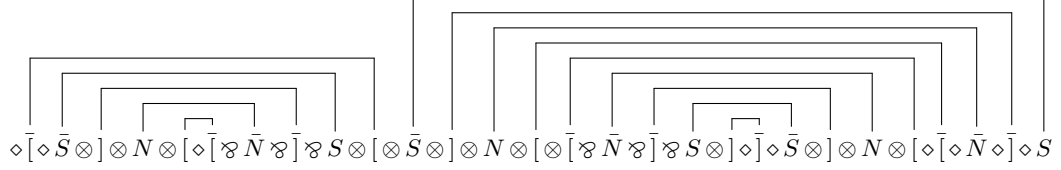
Informally speaking, two literals are connected by $\mathcal{E}$ if they come from the same axiom or bracket rule instance. The next figure depicts the $\mathcal{A}$ relation obtained from $\mathcal{E}$ by Definition 11 and the dominance relation $\prec$ (Definition 7), showing that $\mathcal{A} \cup \prec$ is acyclic:
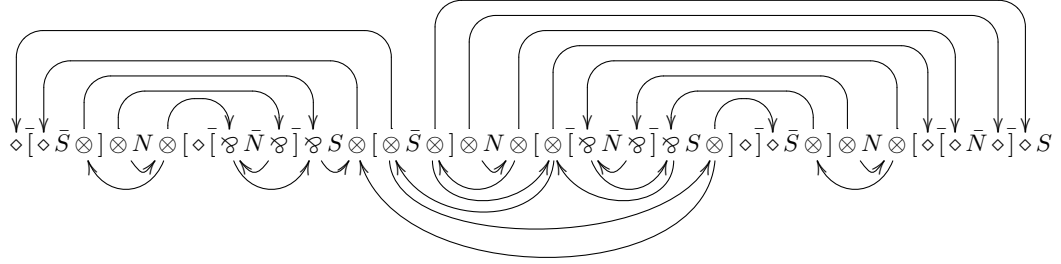
Here $\mathcal{A}$ and $\prec$ are drawn above and below the string respectively.

Finally, as one can see from the figure below, the proof net from our example respects sisterhood:
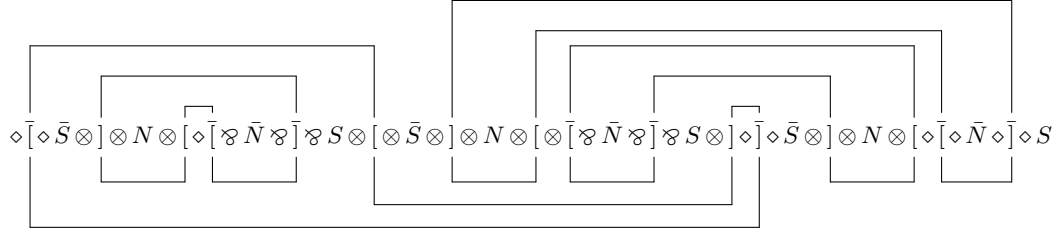
Here $\mathcal{E}$ (we keep only edges connected to brackets) and $\mathcal{S}_b$ are drawn above and below the string respectively. Graphically the sisterhood condition means that these edges form 4-cycles. In the end of this section we show that this sisterhood condition is essential: if it fails, the sequent could be not derivable.

Now we are going to prove that a sequent is derivable in **Lb*** if and only if there exists a proof net that respects sisterhood. First we establish the following technical lemma.

▶ **Lemma 13.** *For an expression of the form $\diamond A_1^- \diamond \ldots \diamond A_m^-$ (which is not a translation of an **Lb*** sequent, since there is no B^+ at the end) there couldn't exist a proof net.*

Proof. Following Pentus [33], for any string γ of literals, $\wp$'s, $\otimes$'s, and $\diamond$'s we define $\natural(\gamma)$ as the number of negative literals (i.e., of the form $\bar{q}$, where q is a variable or a bracket) minus

the number of $\bindnasrepma$'s and $\diamond$'s. Then we establish the following: (1) $\natural(A^+) = 0$ and $\natural(A^-) = 1$ for any formula A; (2) if there exists a proof net for γ, then $\natural(\gamma) = -1$. The first statement is proved by joint induction on A. The second one follows from the fact that the number of regions is greater than the number of $\mathcal{E}$ links exactly by one; links are in one-to-one correspondence with negative literal occurrences and each region holds a unique occurrence of $\bindnasrepma$ or $\diamond$. Since $\natural(\diamond A_1^- \diamond \ldots \diamond A_m^-) = m - m = 0 \neq -1$, there is no proof net.   ◄

▶ **Theorem 14.** *The sequent $\Gamma \to C$ is derivable in* $\mathbf{Lb}^*$ *if and only if there exists a proof net $\mathcal{E}$ over $\Omega_{\Gamma \to C}$ that respects sisterhood.*

Proof. The direction from $\mathbf{Lb}^*$-derivation to proof net is routine: we construct the proof net by induction, maintaining the correctness criterion. For the other direction, we proceed by induction on the number of $\bindnasrepma$ and $\otimes$ occurrences.

If there are no occurrences of $\bindnasrepma$ or $\otimes$, then the total number of $\diamond$ occurrences is, on the one hand, equal to n; on the other hand, it is equal to the number of regions, $\frac{n}{2} + 1$. Therefore, $n = 2$, and the only possible proof net is $\diamond \, \overline{\bar{p} \diamond p}$ that corresponds to the $p \to p$ axiom.

Otherwise consider the set of all occurrences of $\bindnasrepma$ and $\otimes$ with the relation $\mathcal{A} \cup \prec$. Since this relation is acyclic (and the set is not empty), there exists a maximal element, c_i.

Case 1.1: c_i is a $\bindnasrepma$ occurrence that came from $(A \cdot B)^- = B^- \bindnasrepma A^-$. Replacing this $\bindnasrepma$ by $\diamond$ corresponds to applying $(\cdot \to)$.

Case 1.2: c_i is a $\bindnasrepma$ occurrence that came from $(A \backslash B)^+ = A^- \bindnasrepma B^+$. Replacing this $\bindnasrepma$ by $\diamond$ changes $\diamond \Gamma^- \diamond A^- \bindnasrepma B^+$ to $\diamond \Gamma^- \diamond A^- \diamond B^+$, which corresponds to applying $(\to \backslash)$. (In the negative translation, formulae in the left-hand side appear in the inverse order.)

Case 1.3: c_i is a $\bindnasrepma$ occurrence that came from $(B \,/\, A)^+ = B^+ \bindnasrepma A^-$. Again, replace $\bindnasrepma$ with $\diamond$ and cyclically transform the net, yielding $\diamond A^- \diamond \Gamma^- \diamond B^+$. Then apply $(\to /)$.

Case 2.1: c_i is a $\otimes$ occurrence that came from $(A \cdot B)^+ = A^+ \otimes B^+$. Then $\mathcal{A}(c_i)$ is a $\diamond$ occurrence, and the $\mathcal{A}$ link splits the proof net for $\Gamma \to A \cdot B$ into two separate proof nets for $\Gamma_1 \to A$ and $\Gamma_2 \to B$ (Γ_1 and/or Γ_2 could be empty, then two or three $\diamond$'s shrink into one):

$$\diamond \quad \Gamma_2^- \quad \overset{\frown}{\underset{\downarrow}{}} \quad \diamond \quad \Gamma_1^- \quad \diamond \quad A^+ \quad \otimes \quad B^+$$

Note that here the fragments before the $\diamond$ occurrence $\mathcal{A}(c_i)$ and between $\mathcal{A}(c_i)$ and A are negative translations of whole metaformulae (Γ_1 and Γ_2), not just substrings with possibly disbalanced brackets. Indeed, suppose that a pair of sister brackets, $\overline{[}$ and $\overline{]}$, is split between these two fragments. Then, since $\mathcal{E}$ links cannot intersect $\mathcal{A}$, the corresponding pair of $[$ and $]$, connected to the original pair by $\mathcal{E}$, will also be split and therefore belong to translations of different formulae. However, they also form a sister pair (our proof net respects sisterhood), and therefore should belong to one formula (by definition of the translation). Contradiction.

Since for the sequents $\Gamma_1 \to A$ and $\Gamma_2 \to B$ the induction parameter is smaller, they are derivable, and therefore $\Gamma_1, \Gamma_2 \to A \cdot B$ is derivable by application of the $(\to \cdot)$ rule.

Case 2.2: c_i is a $\otimes$ occurrence from $(A \backslash B)^- = B^- \otimes A^+$. Again, the proof net gets split:

$$\diamond \quad \underbrace{\ldots \quad \diamond \quad B^- \quad \otimes \quad A^+ \quad \diamond \quad \Pi^- \quad \diamond \quad \ldots}_{\Delta^-} \quad \diamond \quad C^+$$

(As in the previous case, no pair of sister brackets could be split by $\mathcal{A}$ here, and Π^- is a translation of a whole metaformula.) The outer fragment provides a proof net for $\Delta\langle B\rangle \to C$; applying a cyclic permutation to the inner fragment yields a proof net for $\Pi \to A$. The goal sequent, $\Delta\langle \Pi, A \setminus B\rangle \to C$, is obtained by applying the $(/ \to)$ rule.

The other situation,

$$\diamond \quad \ldots \quad \diamond \quad \Pi^- \quad \diamond \quad B^- \quad \otimes \quad A^+ \quad \diamond \quad \ldots \quad \diamond \quad C^+$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\Delta^-}$$

is impossible by Lemma 13, applied to the inner net.

Case 2.3: c_i is a $\otimes$ occurrence that came from $(B / A)^- = A^+ \otimes B^-$. Symmetric.

Case 3.1: c_i is a $\otimes\hspace{-0.6em}\otimes$ occurrence that came from $([]^{-1}A)^+ = \bar{]} \otimes\hspace{-0.6em}\otimes A^+ \otimes\hspace{-0.6em}\otimes \bar{[}$. Then we replace two $\otimes\hspace{-0.6em}\otimes$'s by $\diamond$'s and cyclically relocate the rightmost $\bar{[}$ with its $\mathcal{E}$ link, obtaining $\diamond \bar{[} \diamond \Gamma^- \diamond \bar{]} \diamond A^+$ from $\diamond \Gamma^- \diamond \bar{]} \otimes\hspace{-0.6em}\otimes A^+ \otimes\hspace{-0.6em}\otimes \bar{[}$. This corresponds to an application of $(\to []^{-1})$.

Case 3.2: c_i is a $\otimes\hspace{-0.6em}\otimes$ occurrence that came from $(\langle\rangle A)^- = \bar{[} \otimes\hspace{-0.6em}\otimes A^- \otimes\hspace{-0.6em}\otimes \bar{]}$. By replacing $\otimes\hspace{-0.6em}\otimes$'s with $\diamond$'s, we change $\langle\rangle A$ into $[A]$. This corresponds to an application of $(\langle\rangle \to)$.

Case 4.1: c_i is a $\otimes$ occurrence that came from $(\langle\rangle A)^+ = \bar{]} \otimes A^+ \otimes \bar{[}$. Consider the $\mathcal{E}$ links that go from these $]$ and $[$. Since $(\langle\rangle A)^+$ is the rightmost formula, they both either go to the left or into A^+. The second situation is impossible, because then $\mathcal{A}(c_i)$ should also be a $\otimes\hspace{-0.6em}\otimes$ occurrence in A^+, that violates the maximality of c_i (and also the acyclicity condition).

In the first situation, the picture is as follows:

$$\gamma_1 \quad \diamond \quad \bar{[} \quad \diamond \quad \Gamma^- \quad \diamond \quad \bar{]} \quad \diamond \quad \gamma_2 \quad \bar{]} \quad \otimes \quad A^+ \quad \otimes \quad \bar{[}$$

(Due to maximality of c_i, $\mathcal{A}(c_i)$ and its sister are $\diamond$'s, not $\otimes\hspace{-0.6em}\otimes$'s.) Clearly, γ_1 and γ_2 are empty: otherwise we have two $\diamond$'s in one region. Then we can remove brackets and transform this proof net into a proof net for $\diamond\Gamma^- \diamond A^+$, i.e., $\Gamma \to A$. Applying $(\to \langle\rangle)$ yields $[\Gamma] \to \langle\rangle A$.

Case 4.2: c_i is a $\otimes$ occurrence that came from $([]^{-1}A)^- = \bar{[} \otimes A^- \otimes \bar{]}$. As in the previous case, consider the $\mathcal{E}$ links going from these bracket occurrences. The good situation is when they go to different sides:

$$\diamond \quad \ldots \quad \diamond \quad \bar{[} \quad \gamma_1 \quad \diamond \quad \bar{[} \quad \otimes \quad A^- \quad \otimes \quad \bar{]} \quad \diamond \quad \gamma_2 \quad \bar{]} \quad \diamond \quad \ldots \quad \diamond \quad C^+$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\Delta^-}$$

(The connectives surrounding $\bar{[} \otimes A^- \otimes \bar{]}$ are $\diamond$'s due to the maximality of c_i.) Again, γ_1 and γ_2 should be empty (the connective after $\bar{[}$ or before $\bar{]}$ here cannot be a $\otimes$, and we get more than one $\otimes\hspace{-0.6em}\otimes$ or $\diamond$ in a region), and removing the bracket corresponds to applying $([]^{-1} \to)$: replace $[[]^{-1}A]$ with A in the context Δ.

Potentially, the $\mathcal{E}$ links from the brackets could also go to one side, but then they end at a pair of sister brackets $\bar{]}$ and $\bar{[}$ (in this order), which can occur only in $\bar{]} \otimes\hspace{-0.6em}\otimes B \otimes\hspace{-0.6em}\otimes \bar{[}$. Then one of these $\otimes\hspace{-0.6em}\otimes$'s is $\mathcal{A}(c_i)$, which contradicts the maximality of c_i. ◄

The idea of proof nets as a representation of derivation in a parallel way comes from Girard's original paper on linear logic [15]. For the non-commutative case, including the Lambek calculus, proof nets were studied by many researchers including Abrusci [3], de Groote [9], Nagayama and Okada [29], Penn [30], Pentus [33], Yetter [38], and others. In our definition

of proof nets for $\mathbf{Lb}^*$ we follow Fadda and Morrill [11], but with the correctness (acyclicity) conditions of Pentus [33][35] rather than Danos and Regnier [8].

The idea of handling brackets similarly to variables is due to Versmissen [37]. If we take a sequent that is derivable in $\mathbf{Lb}^*$, replace brackets with fresh variables, say, r and s, and respectively substitute $r \cdot A \cdot s$ for $\langle\rangle A$ and $r \backslash A / s$ for $[]^{-1}A$, we obtain a sequent that is derivable in $\mathbf{L}^*$ (this follows from our proof net criterion and can also be shown directly). Versmissen, however, claims that the converse is also true. This would make our Theorem 4 a trivial corollary of Pentus' result [35], but Fadda and Morrill [11] present a counter-example to Versmissen's claim. Namely, the sequent $[]^{-1}p, []^{-1}q \to \langle\rangle []^{-1}(p \cdot q)$ is not derivable in $\mathbf{Lb}^*$, but its translation, $r, r \backslash p / s, s, r, r \backslash q / s, s \to r \cdot (r \backslash (p \cdot q) / s) \cdot s$, is derivable in $\mathbf{L}^*$. This example shows the importance of the sisterhood condition in Theorem 14: the only possible proof net for this sequent, shown below, doesn't respect sisterhood.

4 Complexity Parameters for Proof Nets
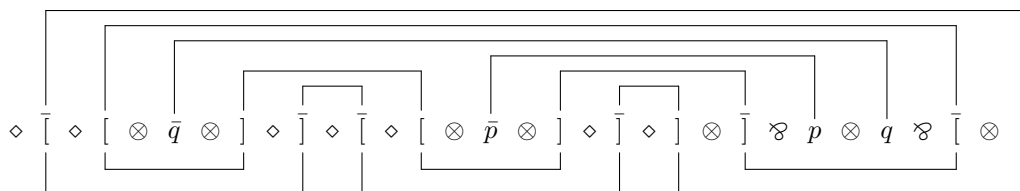
In this section we introduce new complexity parameters that operate with $\Omega_{\Gamma \to C}$ rather than with the original sequent $\Gamma \to C$, and therefore are more handy for complexity estimations. We show that a value is polynomial in terms of the old parameters if it is polynomial in terms of the new ones.

The first parameter, denoted by n, is the number of literals in $\Omega_{\Gamma \to C}$. It is connected to the size of the original sequent by the following inequation: $n \leq 2\|\Gamma \to C\|$. (We have to multiply by 2, since a modality, $\langle\rangle$ or $[]^{-1}$, being counted as one symbol in $\Gamma \to C$, introduces two literals.)

The second parameter, denoted by $d = d(\Omega_{\Gamma \to C})$, is the *connective alternation depth,* and informally it is the maximal number of alternations between $\⅋$ and $\otimes$ on the $\prec$-path from any literal to the root of the parse tree. Formally it is defined by recursion.

▶ **Definition 15.** For an expression γ constructed from literals using $\⅋$, $\otimes$, and $\diamond$, let $\text{prod}(\gamma)$ be 1 if γ is of the form $\gamma_1 \otimes \gamma_2$, and 0 otherwise. Define $d(\gamma)$ by recursion: $d(q) = d(\bar{q}) = 0$ for any literal q; $d(\gamma_1 \⅋ \gamma_2) = d(\gamma_1 \diamond \gamma_2) = \max\{d(\gamma_1) + \text{prod}(\gamma_1), d(\gamma_2) + \text{prod}(\gamma_2)\}$; $d(\gamma_1 \otimes \gamma_2) = \max\{d(\gamma_1), d(\gamma_2)\}$.

The d parameter is connected to the order of the original sequent:

▶ **Lemma 16.** *For any sequent $\Gamma \to C$, the following holds: $d(\Omega_{\Gamma \to C}) \leq \text{ord}(\Gamma \to C)$.*

Proof. We prove the following two inequations for any formula A by simultaneous induction on the construction of A: $d(A^+) \leq \text{ord}(A)$ and $d(A^-) + \text{prod}(A^-) \leq \text{ord}(A)$. The induction is straightforward, the only interesting cases are the negative ones for $A_2 \backslash A_1$, A_1 / A_2, and $([]^{-1}A_1)^-$. In those three cases, we need to branch further into two subcases: whether A_1 is a variable or a complex type. Finally, $d(\Omega_{\Gamma \to C}) = d(\diamond A_1^- \diamond \ldots \diamond A_k^- \diamond C^+) = \max\{d(A_1^-) + \text{prod}(A_1^-), \ldots, d(A_k^-) + \text{prod}(A_k^-), d(C^+) + \text{prod}(C^+)\} \leq \max\{\text{ord}(\Gamma), \text{ord}(C) + \text{prod}(C)\} \leq \max\{\text{ord}(\Gamma) + 1, \text{ord}(C) + \text{prod}(C)\} = \text{ord}(\Gamma \to C)$. ◀

The third parameter is b, the maximal nesting depth of pairs of sister brackets. Clearly, $b = \mathrm{b}(\Gamma \to C)$.

In view of the inequations established in this section, if a value is $\mathrm{poly}(n, 2^d, n^b)$, it is also $\mathrm{poly}(\|\Gamma \to C\|, 2^{\mathrm{ord}(\Gamma \to C)}, \|\Gamma \to C\|^{\mathrm{b}(\Gamma \to C)})$, and for our algorithm we'll establish complexity bounds in terms of n, d, and b.

5   The Algorithm

Our goal is to obtain an efficient algorithm that searches for proof nets that respect sisterhood. We are going to split this task: first find all possible proof nets satisfying Pentus' correctness conditions, and then distill out those which respect sisterhood. One cannot, however, simply yield all the proof nets. The reason is that there exist derivable sequents, even without brackets and of order 2, that have exponentially many proof nets, for example, $p \,/\, p, \ldots, p \,/\, p, p, p \setminus p, \ldots, p \setminus p \to p$. Therefore, instead of generating all the proof nets for a given sequent, Pentus, as a side-effect of his provability verification algorithm, produces a context free grammar that generates a set of words encoding all these proof nets. We filter this set by intersecting it with the set of codes of all proof structures that respect sisterhood. For the latter, we build a finite automaton of polynomial size.

Note that this context free grammar construction is *different* from the translation of Lambek categorial grammars into context free grammars (Pentus [31]). The grammar from Pentus' algorithm that we consider here generates all proof nets for a *fixed* sequent, while in [31] a context free grammar is generated for *all* words that have corresponding derivable Lambek sequents. The latter (global) grammar is of exponential size (though for the case of only one division there also exists a polynomial construction [21]), while the former (local) one is polynomial. For the bracket extension, we present a construction of the local grammar. The context-freeness for the global case is claimed by Jäger [16], but his proof uses the incorrect lemma by Versmissen (see above).

Following Pentus [35], for a given sequent $\Gamma \to C$ we encode proof structures as words of length n over alphabet $\{e_1, \ldots, e_n\}$.

▶ **Definition 17.** The code $c(\mathcal{E})$ of proof structure $\mathcal{E}$ is constructed as follows: if ℓ_i and ℓ_j are connected by $\mathcal{E}$, then the i-th letter of $c(\mathcal{E})$ is e_j and the j-th letter is e_i.

The code of a proof structure is always an involutive permutation of e_1, ..., e_n.

We are going to define two languages, P_1 and P_2, with the following properties:
1. $P_1 = \{c(\mathcal{E}) \mid \mathcal{E}$ is a proof net$\}$;
2. $c(\mathcal{E}) \in P_2$ iff $\mathcal{E}$ respects sisterhood.

Note that in the condition for P_2 we say nothing about words that are not of the form $c(\mathcal{E})$. Some of these words could also belong to P_2. Nevertheless, $w \in P_1 \cap P_2$ iff $w = c(\mathcal{E})$ for some pairing $\mathcal{E}$ that is a proof net and respects sisterhood. Therefore, the sequent is derivable in $\mathbf{Lb}^*$ iff $P_1 \cap P_2 \neq \varnothing$.

Now the algorithm that checks derivability in $\mathbf{Lb}^*$ works as follows: it constructs a context free grammar for $P_1 \cap P_2$ and checks whether the language generated by this grammar is non-empty. Notice that the existence of such a grammar is trivial, since the language is finite. However, it could be of exponential size, and we're going to construct a grammar of size $\mathrm{poly}(n, 2^d, n^b)$, and do it in polynomial time.

For P_1, we use the construction from [35]. As the complexity measure (size) of a context free grammar, $|\mathfrak{G}_1|$, we use the sum of the length of its rules.

▶ **Theorem 18** (M. Pentus 2010). *There exists a context free grammar $\mathfrak{G}_1$ of size $\mathrm{poly}(n, 2^d)$ that generates P_1. Moreover, this grammar can be obtained from the original sequent by an algorithm with working time also bounded by $\mathrm{poly}(n, 2^d)$.*

This theorem is stated as a remark in [35]. We give a full proof of it in Section 6.

Next, we construct a finite automaton for a language that satisfies the condition for P_2.

▶ **Lemma 19.** *There exists a deterministic finite automaton with $\mathrm{poly}(n, n^b)$ states that generates a language P_2 over alphabet $\{e_1, \dots, e_n\}$ such that $c(\mathcal{E}) \in P_2$ iff $\mathcal{E}$ respects sisterhood. Moreover, this finite automaton can be obtained from the original sequent by an algorithm with working time $\mathrm{poly}(n, n^b)$.*

Proof. First we describe this automaton informally. Its memory is organised as follows: it includes a pointer i to the current letter of the word (a number from 1 to $n + 1$) and a stack that can be filled with letters of $\{e_1, \dots, e_n\}$. In the beginning, $i = 1$ and the stack is empty. At each step (while $i \leq n$), the automaton looks at ℓ_i. If it is not a bracket, the automaton increases the pointer and proceeds to the next letter in the word. If it is a bracket, let its sister bracket be ℓ_j. Denote the i-th (currently being read) letter of the word by $e_{i'}$. If $\ell_{i'}$ is not a bracket, yield "no" (bracket is connected to non-bracket). Otherwise let $\ell_{j'}$ be the sister of $\ell_{i'}$ and consider two cases.
1. $j > i$. Then push $e_{j'}$ on top of the stack, increase the pointer and continue.
2. $j < i$. Then pop the letter from the top of the stack and compare it with $e_{i'}$. If they do not coincide, yield "no". Otherwise increase the pointer and continue.
If $i = n + 1$ and the stack is empty, yield "yes".

Since sister brackets are well-nested, on the i-th step we pop from the stack the symbol that was pushed there on the j-th step (if ℓ_i and ℓ_j are sister brackets and $j < i$). Thus, the symbol popped from the stack contains exactly the information that, if the bracket ℓ_j is connected to $\ell_{j'}$, then the bracket ℓ_i should be connected to the sister bracket $\ell_{i'}$, and we verify the fact that $\mathcal{E}$ satisfies this condition by checking that the i-th letter is actually $e_{i'}$.

Note that here we do not check the fact that the word really encodes some proof structure $\mathcal{E}$, since malformed codes will be ruled out by the intersection with P_1.

If the bracket nesting depth is b, we'll never have more than b symbols on the stack. For each symbol we have n possibilities ($e_1, \dots, e_n$). Therefore, the total number of possible states of the stack is $1 + n + n^2 + \dots + n^b \leq (b + 1) \cdot n^b$. The pointer has $(n + 1)$ possible values. Thus, the whole number of possible memory states is $(n + 1) \cdot (b + 1) \cdot n^b + 1$ (the last "+1" is for the "failure" state, in which the automaton stops to yield "no").

Formally, our automaton is a tuple $\mathfrak{A}_2 = \langle Q, \Sigma, \delta, q_0, \{q_F\} \rangle$, where $\Sigma = \{e_1, \dots, e_n\}$ is the alphabet, $Q = \{1, \dots, n + 1\} \times \Sigma^{\leq b} \cup \{\bot\}$, where $\Sigma^{\leq b}$ is the set of all words over Σ of length not greater than b, is the set of possible states ($\bot$ is the "failure" state), $q_0 = \langle 1, \varepsilon \rangle$ is the initial state, $q_F = \langle n + 1, \varepsilon \rangle$ is the final (accepting) state, and $\delta \subset Q \times \Sigma \times Q$ is a set of transitions defined as follows:

$$\delta = \{\langle i, \xi \rangle \xrightarrow{e_{i'}} \langle i + 1, \xi \rangle \mid \ell_i \text{ is not a bracket}\}$$
$$\cup \, \{\langle i, \xi \rangle \xrightarrow{e_{i'}} \bot \mid \ell_i \text{ is a bracket and } \ell_{i'} \text{ is not a bracket}\}$$
$$\cup \, \{\langle i, \xi \rangle \xrightarrow{e_{i'}} \langle i + 1, \xi e_{j'} \rangle \mid \ell_i \text{ is a bracket, its sister bracket is } \ell_j, \, j > i;$$
$$\ell_{i'} \text{ is a bracket, its sister bracket is } \ell_{j'}\}$$
$$\cup \, \{\langle i, \xi e_{i'} \rangle \xrightarrow{e_{i'}} \langle i + 1, \xi \rangle \mid \ell_i \text{ is a bracket, its sister bracket is } \ell_j, \, j < i\}$$
$$\cup \, \{\langle i, \xi e_{i''} \rangle \xrightarrow{e_{i'}} \bot \mid \ell_i \text{ is a bracket, its sister bracket is } \ell_j, \, j < i, \text{ and } i' \neq i''\}.$$

$\mathfrak{A}_2$ is a deterministic finite automaton with not more than $(n+1) \cdot (b+1) \cdot n^b + 1 = \text{poly}(n, n^b)$ states, and it generates a language P_2 such that $c(\mathcal{E}) \in P_2$ iff $\mathcal{E}$ respects sisterhood.

In the RAM model, each transition is computed in constant time, and the total number of transitions is not more than $|Q|^2 \cdot n \leq ((n+1) \cdot (b+1) \cdot n^b + 1)^2 \cdot n$, which is also $\text{poly}(n, n^b)$. ◄

Now we combine Theorem 18 and Lemma 19 to obtain a context free grammar $\mathfrak{G}$ for $P_1 \cap P_2$ of size $\text{poly}(|\mathfrak{G}_1|, |\mathfrak{A}_2|, |\Sigma|)$, where $|\mathfrak{A}_2|$ is the number of states of $\mathfrak{A}_2$. For this we use the following well-known result:

▶ **Theorem 20.** *If a context free grammar $\mathfrak{G}_1$ defines a language P_1 over an alphabet Σ and a deterministic finite automaton $\mathfrak{A}_2$ defines a language P_2 over the same alphabet, then there exists a context free grammar $\mathfrak{G}$ that defines $P_1 \cap P_2$, the size of this grammar is $\text{poly}(|\mathfrak{G}_1|, |\mathfrak{A}_2|, |\Sigma|)$, and, finally, this grammar can be obtained from $\mathfrak{G}_1$ and $\mathfrak{A}_2$ by an algorithm with working time also $\text{poly}(|\mathfrak{G}_1|, |\mathfrak{A}_2|, |\Sigma|)$.*

For this theorem we use the construction from [14, Theorem 3.2.1] that works directly with the context free formalism. This makes the complexity estimation straightforward. Since $|\mathfrak{G}_1| = \text{poly}(n, 2^d)$, $|\mathfrak{A}_2| = \text{poly}(n, n^b)$, and $|\Sigma| = n$, $|\mathfrak{G}_2|$ is $\text{poly}(n, 2^d, n^b)$. Finally, checking derivability of the sequent is equivalent to testing the language $P_1 \cap P_2$ for non-emptiness, which is done using the following theorem [14, Lemma 1.4.3a and Theorem 4.1.2a]:

▶ **Theorem 21.** *There exists an algorithm that checks whether the language generated by a context free grammar $\mathfrak{G}$ is non-empty, with $\text{poly}(|\mathfrak{G}|)$ working time.*

The whole algorithm described in this section works in $\text{poly}(n, 2^d, n^b) = \text{poly}(\|\Gamma \to C\|, 2^{\text{ord}(\Gamma \to C)}, \|\Gamma \to C\|^{\text{b}(\Gamma \to C)})$ time, as required in Theorem 4.

6   Proof of Theorem 18 (Pentus' Construction Revisited)

In our algorithm, described in Section 5, we use Pentus' polynomial-size context free grammar, that generates all proof nets, as a black box: we need only Theorem 18 itself, not the details of the construction in its proof. However, Pentus [35] doesn't explicitly formulate this theorem, but rather gives it as side-effect of the construction for checking *existence* of a proof net (e.g., non-emptiness of the context free language). The latter is, unfortunately, not sufficient for our needs. Moreover, we use slightly different complexity parameters. Therefore, and also in order to make our paper logically self-contained, in this section we redisplay Pentus' construction in more detail. In other words, here we prove Theorem 18.

Pentus' idea for seeking proof nets is based on dynamic programming. In $\Omega_{\Gamma \to C}$, connective and literal occurrences alternate: $c_1, \ell_1, c_2, \ell_2, \ldots, c_n, \ell_n$. Consider triples of the form (i, j, k), where $1 \leq i \leq j \leq k \leq n$.

▶ **Definition 22.** An (i, j, k)-*segment* $\tilde{\mathcal{E}}$ is a planar pairing of literals from $\{\ell_i, \ldots, \ell_{k-1}\}$ such that in every region created by $\tilde{\mathcal{E}}$ there exists a unique $\otimes$ or $\diamond$ occurrence from $\{c_i, \ldots, c_k\}$ that belongs to this region, and, in particular, this occurrence for the outer (infinite) region is c_j, called the *open par*. If $k = j = i$, then $\tilde{\mathcal{E}}$ is empty, and c_i should be a $\otimes$ or $\diamond$ occurrence.

For each $\tilde{\mathcal{E}}$ we construct the corresponding $\tilde{\mathcal{A}}$ that connects each $\otimes$ occurrence to the only $\otimes$ or $\diamond$ in the same region; for the outer region, it uses the open par.

▶ **Definition 23.** An (i, j, k)-segment $\tilde{\mathcal{E}}$ is *correct,* iff the graph $\tilde{\mathcal{A}} \cup \prec$ is acyclic.

For each (i, j, k)-segment, in the non-terminals of the grammar we keep a small amount of information, which we call the *profile* of the segment and that is sufficient to construct bigger segments (and, finally, the whole proof net) from smaller ones.

▶ **Definition 24.** A $\otimes$ occurrence is called *dominant,* iff it is not immediately dominated (in the $\prec$ preorder) by another $\otimes$ occurrence. For each $\otimes$ occurrence c there exists a unique dominant $\otimes$ occurrence $\tau(c)$ such that $\tau(c) \succeq c$ and on the $\prec$ path from c to $\tau(c)$ all occurrences are $\otimes$ occurrences.

Let's call two $\otimes$ occurrences *equivalent,* $c \approx c'$, if $\tau(c) = \tau(c')$. Equivalent $\otimes$ occurrences form *clusters;* from each cluster we pick a unique representative, the $\prec$-maximal occurrence $\tau(c)$. By $\precsim$ we denote the transitive closure of $\prec \cup \approx$: $c \precsim c'$ means that there is a path from c to c' that goes along $\prec$ and also could go in the inverse direction, but only from $\otimes$ to $\otimes$ with no $\bindnasrepma$ or $\diamond$ in between.

▶ **Lemma 25.** *For an (i, j, k)-segment $\tilde{\mathcal{E}}$, the graph $\tilde{\mathcal{A}} \cup \prec$ is acyclic iff any cycle in $\tilde{\mathcal{A}} \cup \precsim$ is a trivial $\approx$-cycle in a cluster, and similarly for a proof structre $\mathcal{E}$ and the graphs $\mathcal{A} \cup \prec$ and $\mathcal{A} \cup \precsim$.*

Proof. Pentus proves this lemma by a topological argument. If $\tilde{\mathcal{A}} \cup \precsim$ has non-trivial cycles, take a simple cycle (i.e. a cycle where no vertex appears twice) that embraces the smallest area. If this cycle includes a link from c to d where $c \approx d$ and $c \succ d$, then consider the $\tilde{\mathcal{A}}$ link that goes from c. This link should go *inside* the cycle, and, continuing by this link, one could construct a new cycle with a smaller area embraced. Contradiction. The other direction is trivial, since every cycle in $\tilde{\mathcal{A}} \cup \prec$ is a non-trivial cycle in $\tilde{\mathcal{A}} \cup \precsim$.   ◄

In view of this lemma we can now use $\precsim$ instead of $\prec$ in the correctness (acyclicity) criteria for proof nets and (i, j, k)-segments.

▶ **Definition 26.** For a connective occurrence c_i let V_i be the set of all dominant $\otimes$ occurrences on the $\prec$ path from c_i to the root of the parse tree.

Since each dominant $\otimes$ marks a point of alternation between $\otimes$ and $\bindnasrepma$ (or $\diamond$, on the top level), and the number of such alternations is bounded by d, we have $|V_i| \leq d$ for any i.

▶ **Definition 27.** The profile of an (i, j, k)-segment $\tilde{\mathcal{E}}$, denoted by R, is the restriction of the transitive closure of $\tilde{\mathcal{A}} \cup \precsim$ to the set $V_i \cup V_j \cup V_k$ that is forced to be irreflexive (in other words, we remove trivial $\approx$-cycles). An (i, j, k)-*profile* is an arbitrary transitive irreflexive relation on $V_i \cup V_j \cup V_k$.

▶ **Lemma 28.** *The number of different (i, j, k)-profiles is $\mathrm{poly}(2^d)$.*

Proof. Let $|V_i| = d_1$, $|V_j| = d_2$, $|V_k| = d_3$ (these three numbers are not greater than d). Each profile includes three chains, Q_i, Q_j, and Q_k, and it remains to count the number of possible connections between them. Due to transitivity, if a vertex in V_i is connected to a vertex in V_j, then it is also connected to all greater vertices. Now we represent elements of V_j as d_2 white balls, and put d_1 black balls between them. The i-th black ball is located in such a place that the i-th vertex of V_i is connected to all vertices of V_j that are greater than the position of the i-th ball, and only to them. Due to transitivity, the order of black balls is the same as Q_i. The number of possible distributions of white and black balls is $\binom{d_1 + d_2}{d_1} < 2^{d_1 + d_2} \leq 2^{2d}$. Doing the same for all 6 pairs of 3 chains, we get the estimation $(2^{2d})^6 = (2^d)^{12} = \mathrm{poly}(2^d)$ for the number of (i, j, k)-profiles.   ◄

Now we define the context free grammar $\mathfrak{G}_1$. Non-terminal symbols of this grammar include the starting symbol S and symbols $F_{i,j,k,R}$ for any triple (i, j, k) $(1 \leq i \leq j \leq k \leq n)$ and any (i, j, k)-profile R. The meaning of these non-terminals is in the following statement, which will be proved by induction after we present the rules of $\mathfrak{G}_1$: a word w is derivable from $F_{i,j,k,R}$ iff $w = c(\tilde{\mathcal{E}})$ for a correct (i, j, k)-segment $\tilde{\mathcal{E}}$ with profile R; a word w is derivable from S iff $w = c(\mathcal{E})$ for some proof net $\mathcal{E}$. (Codes of (i, j, k)-segments are defined in the same way as codes of proof structures, as involutive permutations of $e_i, \ldots, e_{k-1}$.)

For the induction base case, $i = j = k$, we take only those values of i such that c_i is a $\bindnasrepma$ or $\diamond$ occurrence, and denote by Q_i the $\prec$ relation restricted to V_i (this is the trivial profile of an empty (i, i, i)-segment); Q_i is always acyclic, and an isolated $\bindnasrepma$ or $\diamond$ occurrence c_i is always a correct (i, i, i)-segment (with an empty $\tilde{\mathcal{E}}$), and c_i is its open par. Now for each $\bindnasrepma$ or $\diamond$ occurrence c_i we add the following rule to the grammar (this is a ε-rule, the right-hand side is empty):

$$F_{i,i,i,Q_i} \Rightarrow .$$

Next, consider the non-trivial situation, where $i < k$. The difference $k - i$ should be even, otherwise there couldn't exist a literal pairing $\tilde{\mathcal{E}}$. Moreover, if both c_i and c_k are $\bindnasrepma$ or $\diamond$ occurrences, a correct (i, j, k)-segment couldn't exist either, since in the outer region we have at least two $\bindnasrepma$ or $\diamond$ occurrences, namely, c_i and c_k. Therefore, we include rules for $F_{i,j,k,R}$ only if $k - i$ is even and at least one of c_i and c_k should be $\otimes$. Let it be c_i (Pentus' *situation of the first kind*). The c_k case (Pentus' *situation of the second kind*) is handled symmetrically.

We take the leftmost literal occurrence, ℓ_i, and find all possible occurrences among $\ell_{i+1}, \ldots, \ell_{k-1}$ that could be connected to ℓ_i (i.e., if ℓ_i is an occurrence of q, we seek $\bar{q}$, and vice versa). For each such occurrence, ℓ_{h-1}, we consider two triples, $(i + 1, j', h - 1)$ and (h, j, k), and all possible $(i + 1, j', h - 1)$- and (h, j, k)-profiles, R_1 and R_2, respectively. For each such pair, R_1 and R_2, we consider the transitive closure of the following relation: $R_1 \cup R_2 \cup Q_i \cup \{\langle \tau(c_i), d \rangle \mid d \in V_j\}$. If it is irreflexive (acyclic), its restriction to $V_i \cup V_j \cup V_k$, denoted by R, will become a profile of an (i, j, k)-segment. For this, we add the following rule to the grammar:

$$F_{i,j,k,R} \Rightarrow e_{h-1} \, F_{i+1,j',h-1,R_1} \, e_i \, F_{h,j,k,R_2}.$$

▶ **Lemma 29.** *In this grammar, a word w can be derived from $F_{i,j,k,R}$ iff $w = c(\tilde{\mathcal{E}})$ for some (i, j, k)-segment $\mathcal{E}$ with profile R.*

Proof. Proceed by induction on $k - i$. The base case $(i = j = k)$ was considered above.

For **the "only if" part,** let w be derived by a rule for the first kind (the second kind is symmetric). Then $w = e_{h-1}w_1 e_i w_2$, and by induction hypothesis w_1 and w_2 encode $(i + 1, j', h - 1)$- and (h, j, k)-segments with profiles R_1 and R_2 respectively. The word w encodes an (i, j, k)-segment, and it remains to show that this segment is correct and its profile is R. For this new segment, $\tilde{\mathcal{A}} = \tilde{\mathcal{A}}_1 \cup \tilde{\mathcal{A}}_2 \cup \langle c_i, c_j \rangle$. Suppose there is a non-trivial cycle in $\tilde{\mathcal{A}} \cup \precsim$. Since all cycles in $\tilde{\mathcal{A}}_1 \cup \precsim$ and $\tilde{\mathcal{A}}_2 \cup \precsim$ are trivial, this cycle should either include links from both $\tilde{\mathcal{A}}_1$ and $\tilde{\mathcal{A}}_2$ or use the new $\langle c_i, c_j \rangle$ connection (or both). The cycle, however, cannot cross $\tilde{\mathcal{E}}$ links, therefore the only way of "legally crossing the border" between segments is by going through $\otimes$ occurrences that dominate c_i, c_{i+1}, c_{h-1}, c_h, or c_k. We can assume that these "border crossing points" are dominant $\otimes$ occurrences (otherwise we can add a $\approx$-detour to the cycle). Then the cycle is actually a concatenation of parts of the following three kinds: (1) connecting vertices of $V_{i+1} \cup V_{h-1}$; (2) connecting vertices of

$V_h \cup V_k$; (3) connecting $\tau(c_i)$, via c_j, to a vertex d of V_j. In this case, our cycle induces a cycle in $R_1 \cup R_2 \cup Q_i \cup \{\langle \tau(c_i), d \rangle \mid d \in V_j\}$, which is impossible by definition.

It remains to show that R is the profile of the newly constructed segment. Indeed, R is a binary relation on $V_i \cup V_j \cup V_k$ and is included in the transitive closure of $\tilde{\mathcal{A}} \cup \precsim$, therefore R is a subrelation of the profile. On the other hand, if there is a pair $\langle c, d \rangle$ in the profile, then there is a path from c to d and, as shown above, it can be split into parts of kinds (1), (2), and (3). Thus, $\langle c, d \rangle \in R$, and therefore R coincides with the profile.

For **the "if" part,** if c_i in an (i, j, k)-segment is a $\otimes$ occurrence, consider the $\tilde{\mathcal{E}}$ link from the literal occurrence ℓ_i. It splits the segment into two ones. For each of them, by induction hypothesis, we generate their codes from $F_{i+1,j',h-1,R_1}$ and F_{h,j,k,R_2} respectively, and then apply the rule to generate the code of the original segment. Situations of the second kind, where c_k is a $\otimes$ occurrence, are handled symmetrically.   ◀

Finally, we add rules for the starting symbol. These rules are analogous to the rules for situations of the second kind. Take ℓ_n and find all possible occurrences among ℓ_1, ..., ℓ_{n-1} that could be connected to it. For each such occurrence ℓ_h and any pair of $(0, 0, h)$- and $(h + 1, j', n)$-profiles, R_1 and R_2, respectively (in the first segment $j = 0$, since in the whole proof net the open par should be the leftmost occurrence of $\diamond$), consider the transitive closure of $R_1 \cup R_2$. If it is irreflexive, then we add the following rule to the grammar:

$$S \Rightarrow F_{0,0,h,R_1} \, e_n \, F_{h+1,j',n,R_2} \, e_h.$$

▶ **Lemma 30.** *A word w can be derived from S iff $w = c(\tilde{\mathcal{E}})$ for some proof net $\mathcal{E}$.*

Proof. Analogous to the previous lemma.   ◀

This lemma shows that we've constructed a grammar that generates P_1. Now to finish the proof of Theorem 18 it remains to establish complexity bounds. The number of non-terminal symbols is bounded by $n^3 \cdot K + 1$, where K is the maximal number of (i, j, k)-profiles. Since each rule has length at most 5 (1 non-terminal on the left and 4 symbols on the right), $|\mathfrak{G}_1|$ is bounded by $5(n^3 \cdot K + 1)$, and, since K is $\mathrm{poly}(2^d)$ (Lemma 28), $|\mathfrak{G}_1|$ is $\mathrm{poly}(2^d, n)$. Clearly, the procedure that generates $\mathfrak{G}_1$ from the original sequent is also polynomial in running time: acyclicity checks for each rule are performed in $\mathrm{poly}(n)$ time, and the number of rules is $\mathrm{poly}(2^d, n)$.

7   Conclusions and Future Work

In this paper, we've presented an algorithm for provability in the Lambek calculus with brackets. Our algorithm runs in polynomial time w.r.t. the size of the input sequent, if its order and bracket nesting depth are bounded. Our new result for bracket modalities is non-trivial, and we address it with a combination of proof nets and finite automata techniques.

We summarize some questions raised for future research. First, Pentus [35] also presents a parsing procedure for Lambek categorial grammars. In a Lambek grammar, several types can be assigned to one word, which adds an extra level of non-determinism. Our intention is to develop an efficient parsing procedure for grammars with brackets. Second, the problem whether **Lb***-grammars define exactly context free languages is still open (the counter-example by Fadda and Morrill [11] jeopardises Jäger's claim). Third, in our calculus we allow empty antecedents. We are going to modify our algorithm for the bracketed extension of the original Lambek calculus, using a modified notion of proof nets (see for example [22][20]). A more

general question is to extend the algorithm to other enrichments of the Lambek calculus (see, for example, [28]), keeping polynomiality, if possible. Notice that some of these enrichments are generally undecidable [18], so it is interesting to find feasible bounded fragments.

──── **References** ────

1   E. Aarts. Proving theorems of second order Lambek calculus in polynomial time. *Studia Logica*, 53:373–387, 1994.

2   E. Aarts and K. Trautwein. Non-associative Lambek categorial grammar in polynomial time. *Math. Logic Quart.*, 41:476–484, 1995.

3   V. M. Abrusci. Non-commutative proof nets. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*. Cambridge University Press, 1995.

4   K. Ajdukiewicz. Die syntaktische Konnexität. *Studia Philosophica*, 1:1–27, 1935.

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

6   W. Buszkowski. Type logics in grammar. In *Trends in Logic: 50 Years of Studia Logica*, pages 337–382. Springer, 2003.

7   B. Carpenter. *Type-Logical Semantics*. MIT Press, Cambridge, MA, 1997.

8   V. Danos and L. Regnier. The structure of multiplicatives. *Arch. Math. Log.*, 28:181–203, 1989.

9   Ph. de Groote. A dynamic programming approach to categorial deduction. In H. Ganzinger, editor, *Proc. CADE 1999*, volume 1632 of *Lect. Notes Comput. Sci.*, pages 1–15. Springer, 1999.

10   Ph. de Groote. The non-associative Lambek calculus with product in polynomial time. In *Proc. TABLEAUX 1999*, pages 128–139. Springer, 1999.

11   M. Fadda and G. Morrill. The Lambek calculus with brackets. In *Language and Grammar: Studies in Mathematical Linguistics and Natural Language*, pages 113–128. CSLI, Jan 2005.

12   T. Fowler. Efficient parsing with the product-free Lambek calculus. In *Proc. COLING 2008*, 2008.

13   T. Fowler. A polynomial time algorithm for parsing with the bounded order Lambek calculus. In *Proc. MoL 2009*, 2009.

14   S. Ginsburg. *The mathematical theory of context-free languages*. McGraw-Hill, 1966.

15   J.-Y. Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

16   G. Jäger. On the generative capacity of multi-modal categorial grammars. *Research on Language and Computation*, 1(1–2):105–125, 2003.

17   G. Jäger. *Anaphora and Type Logical Grammar*, volume 24 of *Trends in Logic – Studia Logica Library*. Springer, Dordrecht, 2005.

18   M. Kanovich, S. Kuznetsov, and A. Scedrov. Undecidability of the Lambek calculus with a relevant modality. In *Proc. Formal Grammar 2015 and 2016*, volume 9804 of *Lect. Notes Comput. Sci.*, pages 240–256. Springer, 2016.

19   N. Kurtonina. *Frames and labels. A modal analysis of categorial inference*. PhD thesis, Universiteit Utrecht, ILLC, Amsterdam, 1995.

20   S. Kuznetsov. Lambek grammars with one division and one primitive type. *Log. J. IGPL*, 20(1):207–221, 2012.

21   S. L. Kuznetsov. On translating Lambek grammars with one division into context-free grammars. *Proc. Steklov Inst. Math.*, 294:129–138, 2016.

22   F. Lamarche and C. Retoré. Proof nets for the Lambek calculus—an overview. In V. M. Abrusci and C. Casadio, editors, *Proofs and Linguistic Categories, Proc. 1996 Roma Workshop*, pages 241–262. CLUEB, 1996.

23   J. Lambek. The mathematics of sentence structure. *Amer. Math. Mon.*, 65:154–170, 1958.

24   J. Lambek. On the calculus of syntactic types. In *Structure of Language and Its Mathematical Aspects*, volume 12 of *Proc. Symposia Appl. Math.*, pages 166–178. AMS, 1961.

25   M. Moortgat. Multimodal linguistic inference. *J. Log. Lang. Inform.*, 5(3, 4):349–385, 1996.

26   R. Moot and C. Retoré. *The Logic of Categorial Grammars: A Deductive Account of Natural Language Syntax and Semantics.* Springer, Heidelberg, 2012.

27   G. Morrill. Categorial formalisation of relativisation: pied piping, islands, and extraction sites. Technical Report LSI-92-23-R, Universitat Politècnica de Catalunya, 1992.

28   G. V. Morrill. *Categorial Grammar: Logical Syntax, Semantics, and Processing.* Oxford University Press, 2011.

29   M. Nagayama and M. Okada. A graph-theoretic characterization theorem for multiplicative fragment of non-commutative linear logic. *Theor. Comput. Sci.*, 294:551–573, 2003.

30   G. Penn. A graph-theoretic approach to sequent derivability in the Lambek calculus. *Electr. Notes Theor. Comput. Sci.*, 53, 2002.

31   M. Pentus. Lambek grammars are context-free. In *Proc. LICS 1993*, pages 430–433, Montreal, 1993.

32   M. Pentus. Models for the Lambek calculus. *Ann. Pure Appl. Log.*, 75(1–2):179–213, 1995.

33   M. Pentus. *Free monoid completeness of the Lambek calculus allowing empty premises*, volume 12 of *Lecture Notes in Logic*, pages 171–209. Springer-Verlag, Berlin, 1998.

34   M. Pentus. Lambek calculus is NP-complete. *Theor. Comput. Sci.*, 357(1):186–201, 2006.

35   M. Pentus. A polynomial-time algorithm for Lambek grammars of bounded order. *Linguistic Analysis*, 36(1–4):441–471, 2010.

36   Yu. Savateev. Unidirectional Lambek grammars in polynomial time. *Theory Comput. Syst.*, 46(4):662–672, 2010.

37   K. Versmissen. *Grammatical composition: modes, models, modalities.* PhD thesis, OTS Utrecht, 1996.

38   D. N. Yetter. Quantales and (noncommutative) linear logic. *J. Symb. Log.*, 55(1):41–64, 1990.