Chapter 1

Bounded and Ordered Satisfiability: Connecting Recognition with Lambek-style Calculi to Classical Satisfiability Testing

MICHAIL FLOURIS, LAP CHI LAU, TSUYOSHI MORIOKA, PERIKLIS A. PAPAKONSTANTINOU, GERALD PENN

1.1 Introduction

It is well known that the Lambek Grammars are weakly equivalent to the Context-Free Grammars (CFGs, Pentus 1993, 1997), and that testing string membership with a CFG is in P (Earley 1970). Nevertheless, Pentus (2003) has recently proven that sequent derivability in the Lambek Calculus with product is NP-complete. The complexity of the corresponding problem for the product-free fragment remains unknown. This fragment is significant, given the at best limited apparent motivation for products in linguistic applications of the calculus. In this paper, when we mention the Lambek Calculus (LC) or Lambek Grammars (LG), we are referring to the product-free fragment.

Pentus (1997) has presented an algorithm that transforms a product-free Lambek Grammar to a weakly equivalent CFG, but the transformed grammar is exponentially larger in the worst case. Since the grammar is considered a part of the instance for the decision problem of string membership, this does not resolve the open complexity problem for the product-free calculus.

This paper studies the connection between the LG string membership problem and the SAT problem, which was the first problem shown to be NP-complete (Cook 1971). Much of the previous work on parsing with Lambek grammars has derived its inspiration from recognition algorithms for rewriting systems (Hepple 1992), string algebras (Morrill 1996) or graph theory (Moot and Puite 1999; Penn 2002), but fundamentally, LC is a logical framework, like the classical propositional logic upon which SAT is based. The crucial difference is the sensitivity to resources and order that LC incorporates. What we argue here is: **(1)** that a sense of order can be imposed on classical SAT using the polarity that propositional variables already possess (unlike LC), **(2)** that the corresponding *ordered* SAT problem is still NP-complete, **(3)** that this new version of SAT leads to a new and simpler proof of the NP-completeness of the product-free Lambek Calculus with permutation (LP) by an implementation of "locks" and "keys" somewhat reminiscent of the proposed modal extensions of categorial logics (Kurtonina and Moortgat 1996), **(4)** that the problem can be further restricted *bounded-distance* SAT in order to fit into LC, but **(5)** that bounded-distance SAT

Proceedings of Mathematics of Language 8 2003
R. T. Oehrle & J. Rogers (editors).

can be solved in polynomial time. In addition, **(6)** we also prove the first non-trivial hardness result for LC that we are aware of, namely that it is LOGCFL-hard. LOGCFL consists of all languages log-space reducible to a context-free language. We shall use LC and LP to refer both to the respective calculi and to the respective decision problems that determine membership in the set of encodings of pairs (G, x), where G is a Lambek Grammar over an alphabet Σ, and $x \in \Sigma^*$ is a string.

1.2 An NP-complete variation of SAT

Our ordered variation of 3-SAT can be stated as follows:
NFPO-SAT
INSTANCE: Let U be a set of variables and $S = \langle C_1, C_2, \ldots C_n \rangle$ a sequence of clauses, where $|C_i| \leq 3$, such that **(1)** each variable $x \in U$ occurs at most once as a negative literal, **(2)** if $\neg x \in C_i$ there exist no $j < i$ where $x \in C_j$, and **(3)** every clause which contains a negative literal cannot contain any positive literal. Let a formula $\Phi = \bigcup_{i=1\ldots n} C_i$.
QUESTION: Is Φ satisfiable ?

NFPO stands for "Negative First Positive the Others", that is: (a) the clauses are ordered such that the first occurrence of a variable is either negative or positive and all the subsequent occurrences are positive, and (b) every clause contains either all positive or all negative variables. Hence, in this variation we can refer to *positive* and *negative* clauses, with the obvious meaning.

Theorem 1.2.1. NFPO-SAT *is NP-complete.*

Proof. The problem is in NP for the same reason that SAT is in NP. We reduce 3-SAT to NFPO-SAT:
Let $\Phi = C_i, \ldots C_n$ be a 3-SAT instance. For each variable x:

1. Let x occur negatively in one or more clauses.

2. Introduce y such that:

$$\Phi' = \Phi \wedge (y \leftrightarrow \neg x), \text{ i.e.,}$$
$$\Phi' = \Phi \wedge (\neg y \vee \neg x) \wedge (y \vee x)$$

3. Rename the negative occurrences of x by y.

4. Iteratively change each variable, using Φ' instead of Φ, apart from the newly introduced variable.

Finally, order the clauses of Φ' by placing all clauses with negative literals before all clauses with only positive literals.

Given a fixed truth assignment, τ, for which $\tau(\Phi) = T$, let τ' be an extended truth assignment for Φ' such that $\tau'(y) = \neg\tau(x)$. Then $\tau'(\Phi') = T$. Hence, Φ' is satisfiable iff Φ is satisfiable. It is also obvious that the reduction works in quadratic time w.r.t. the input length. Note that the reduction works also in logarithmic space. $\square$

It will also be useful for us to consider a version of NFPO-SAT with an additional condition:
BD-NFPO-SAT (Bounded-Distance NFPO-SAT)
INSTANCE: Let U be a set of n variables and $S = \langle C_1, C_2, \ldots C_m \rangle$ a sequence of clauses where $|C_i| \leq 3$ such that **(1)** each variable $x \in U$ occurs at most once as a negative literal, **(2)** if $\neg x \in C_i$ there exist no $j < i$ where $x \in C_j$, **(3)** every clause which contains a negative literal cannot contain any positive literal, and **(4)** there exists $k = \lceil \log n \rceil$ such that, for each variable $x \in U$, if i is minimum with respect to the occurrence of x in C_i, and j is the maximum j such that $x \in C_j$, then $j - i \leq k$. Let a formula $\Phi = \bigcup_{i=1\ldots n} C_i$.
QUESTION: Is Φ satisfiable ?

This version is not NP-complete. In fact, we can prove that the BD-SAT $\in NL \subseteq P$, where NL stands for the class of languages decided by nondeterministic log-space Turing Machines.
Non-deterministic log space algorithm for BD-NFPO-SAT
(i) Guess values for the variables of the k clauses.
(ii) If the subformula is not satisfied reject
(iii) Keep the k last assignments and make a guess for the next clause
(iv) If the clause cannot be satisfied, or if consistency is lost with the previous assignment reject, otherwise add the new guessed values and repeat (iii) (intuitively: slide the k window by one clause to the right).

Bounded distance satisfiability problems are also of independent interest. We state the following theorem without any proof. In Section 1.6, we prove a stronger result for the hardness of LC.

Theorem 1.2.2. BD-NFPO-SAT *is complete for NL.*

The reduction of BD-NFPO-SAT to LC trivially implies:

Corollary 1.2.3. BD-NFPO-SAT *is hard for NL.*

1.3 Reducing SAT to LP

We will follow the Natural Deduction presentation of the Lambek Calculus. We will use the following deduction quite often, which is valid with or without the rule of permutation:

Lemma 1.3.1. *Let A, B and C categories of the Lambek Calculus. The following deductions can be derived using only elimination and introduction rules:*

$$\frac{A/B \quad B/C}{A/C}, \quad \frac{A/A \quad A/A}{A/A}, \quad \frac{A/A \quad A/A \ldots A/A}{A/A} \tag{1.3.1}$$

That LP is NP-complete is already known, both as a corollary of a more general result for multiplicative Linear Logic (Kanovitch 1991, 1992; Lincoln et al. 1990), or directly (Doerre 1996; Florencio 2002). In addition, not all LP grammars are weakly equivalent to CFGs. We sketch two different proofs for the NP-completeness of LP, both by reducing NFPO-SAT to LP. The order of presentation of the proofs is such that we successively make more use of the ordering constraints imposed by NFPO-SAT.

Proof 1. It is well known (Lincoln et al. 1990) that $LP \in NP$. Assume that we have a formula Φ in (NFPO-)CNF form. For each variable x_i occurring positively in Φ we introduce a basic category X_i, and for each variable occurring negatively, we introduce a new category $\bar{X}_i$. We also have a special basic category A. Assume that we have m clauses $C_i, i = 1, \ldots, m$ and n variables $x_i, i = 1, \ldots, n$. We construct the following string $w = c_1 c_2 \ldots c_m x_1 x'_1 x_2 x'_2 \ldots x_n x'_n$, where c_i, x_i and x'_i are distinct symbols of an alphabet Σ. We construct the mapping f (Lexicon) for each symbol of the alphabet as follows: for each variable x_i occurring positively in the clause C_j we add to the set $f(c_j)$ the elements $((A/A)/X_i)/X_i$ and X_i/X_i, and for each x_i occurring negatively in C_j we add the elements $((A/A)/\bar{X}_i)/\bar{X}_i$ and $\bar{X}_i/\bar{X}_i$. Also, for all $1 \leq i \leq n$, $f(x_i) = f(x'_i) = \{X_i, \bar{X}_i\}$. This algorithm is trivially polynomial time.

The substring $w_1 = c_1 \ldots c_m$ of w, corresponds to the selection of a single literal from each clause that witnesses that clause's truth. In the first clause C_j for which a variable x_i is selected, this corresponds to choosing for c_j the category $((A/A)/X_i)/X_i$, and every other selection for the same variable in a later clause corresponds to choosing X_i/X_i. For any k, if a variable is selected in k clauses then, by lemma 1.3.1, we can still derive two categories for this substring: $((A/A)/X_i)/X_i$ and (after some deductions) X_i/X_i which altogether result in $((A/A)/X_i)/X_i$.

The substring $w_2 = x_1 x'_1 \ldots x_n x'_n$ enforces the consistency of the selected variables. If a variable x_i has been selected together with its negation $\neg x_i$ then after doing several deductions we have both $((A/A)/X_i)/X_i$ and $((A/A)/\bar{X}_i)/\bar{X}_i$, but the $x_i x'_i$ component of the substring w_2 can only deduce either $((A/A)/X_i)/X_i$ or (exclusively) $((A/A)/\bar{X}_i)/\bar{X}_i$ to A/A. So to derive A/A, we can select either x_i or its negation but not both. Thus, the LP grammar constructed has A/A as its distinguished category. Hence, if there is a deduction to A/A the formula is satisfiable.

It is easy to see that if the formula is satisfied, then we can select the categories to derive A/A in a fairly simple way: choose the $((A/A)/X_i)/X_i$ or X_i/X_i if x_i is true and $((A/A)/\bar{X}_i)/\bar{X}_i$ or $\bar{X}_i/\bar{X}_i$ if $\neg x_i$ is true from the corresponding clauses.

Remark 1.3.1. In this reduction, we did not take into account the constraints imposed by the NFPO-SAT and hence the same reduction holds for the unconstrained version of SAT.

Remark 1.3.2. Notice that we could have also ordered the literals in each clause, and replace each c_i by a substring $c_{i,1} c_{i,2} \ldots c_{i,k}$ where k is the number of variables in C_i. Then we need extra basic categories $A_{i,l}$ because now each $c_{i,l}$ corresponds to a literal in clause C_i. If x is the first literal, we would have: $f(c_{i,1}) = \{A_{i,1}, X/X/A_{i,k}/A_{i,k-1}/\ldots/A_{i,2},\ A/A/X/X/A_{i,k}/A_{i,k-1}/\ldots/A_{i,2}\}$, and $f(c_{i,2}) = \{A_{i,2}, A_{i,1}\backslash X/X/A_{i,k}/A_{i,k-1}/\ldots/A_{i,3},\ A_{i,1}\backslash A/A/X/X/A_{i,k}/A_{i,k-1}/\ldots/A_{i,3}\}$, and so on. From among the variables of each clause, one emerges as functor in this part of any successful derivation. This variable is the witness selected to attest to the clause's truth. The same procedure takes place to select one literal from every clause, and the rest of the proof can proceed as previously described.

In the next section, we present a proof that takes into account the sense of ordering inherent to NFPO-SAT. Note that there is no need to do this when reducing to LP, since LP does not have any ordering constraints. It will be useful, however, when it comes to considering LC.

1.4 Enforcing restrictions with locks

We now employ the familiar notion of *locks* and *keys*. The notion of *locks* will be especially useful in LC deductions, where permutation is missing. We introduce the idea only by an example here: $(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.

The notion of a lock prevents us from selecting both a variable and its negation. Thus, if $\neg x_1$ is selected from the first clause then it imposes a *lock*, L_1, which is a special basic category. In the next clause, the only literals that can *unlock* this lock should be every other variable apart from x_1. For this example and the corresponding string $c_{1,x_1}c_{1,x_2}c_{1,x_3}c_{2,\neg x_1}c_{2,x_2}c_{2,x_3}$ we have that $f(c_{1,x_1}) = \{A_1, (A/A)/L_1/A_2/A_3\}$, $f(c_{1,x_2}) = \{A_2, A_1\backslash(A/A)/A_3\}$, $f(c_{1,x_3}) = \{A_3, A_2\backslash A_1\backslash(A/A)\}$, $f(c_{2,\neg x_1}) = \{B_1, (A/A)/B_2/B_3\}$, $f(c_{2,x_2}) = \{B_2, B_1\backslash L_1/B_3, B_1\backslash(A/A)/B_3\}$, and $f(c_{3,x_3}) = \{B_3, B_2\backslash B_1\backslash L_1, B_2\backslash B_1\backslash(A/A)\}$. Notice, that if we had another negative variable in the first clause then we could have easily added keys to the other two variables in the second clause. This task is performed by just adding new elements to the corresponding sets and thus it does not change the time needed to compute these sets to something more than polynomial. Furthermore, in NFPO-SAT, all negative literals occur first, so a c_i with locks comes before any c_j with matching keys. The difficult part is to see what happens if the two above clauses are far enough apart. This means that we have to propagate the locks correspondingly. Notice that we could have only one symbol in the string corresponding to each clause (instead of one symbol for each variable), as in our earlier proof. But now, we will combine the idea of locks and keys with our observation in Remark 1.3.2.

Proof 2. Assume that we have an instance of NFPO-SAT with n variables and m clauses. Construct $w = w_{11}w_{12}w_{13}w'_{11}w'_{12}w'_{13}\ w_{21}w_{22}w_{23}w'_{21}w'_{22}w'_{23}\ \ldots\ w_{m1}w_{m2}w_{m3}w'_{m1}w'_{m2}w'_{m3}$. Intuitively, w_{ik} and w'_{ik} correspond to the k-th literal of the i-th clause. We know that in NFPO-SAT every clause contains either only negative or only positive literals. In addition, we know that all the negative clauses precede the positive clauses in the sequence. Assume that C is a negative clause. Then each literal $\neg x \in C$ sets a lock $A/A/L_x/L_x/\ldots/L_x/L_x$ (L_x appears as many times as the positive occurrences of x). The keys for this variable occur in categories assigned to the other variables in the (positive) clauses where x occurs positively. For some positive $C = (x \vee y \vee z)$, then $f(w_x) = \{B_1, L_y/B_2/B_3, A/A/B_2/B_3\}$, $f(w'_x) = \{B'_1, L_z/B'_2/B'_3, A/A/B'_2/B'_3\}$, where B_i, B'_i are used as described in Remark 1.3.2. That is, literal x holds the keys for the other two literals y and z. $f(w_y), f(w_y)', f(w_z)$ and $f(w_z)'$ are constructed analogously.

1.5 Restricting to LC

This second proof is conceptually less dependent on permutation in the sense that it is only used to combine locks and keys. Also, observe that our constructions involve only first-order categories, where the recognition problem is known to be in P. The main task of adapting the previous reductions to LC (with no permutation) is that we need a way of propagating the locks with no permutation at all. If we have a tuple of clauses we can easily compute the sequence in which the literals appear. The problem is that when we put a lock in some clause we may need to change the order of the previously placed locks. So, we need a sufficient number of rewritings. But we do not know the exact number of locks previously placed, which in the worst case could be $\sum_i^n \left(\begin{array}{c} n \\ i \end{array} \right) = 2^n - 1$, where n is the number of variables with locks placed in a negative clause to the left.

BD-NFPO-SAT places exactly the bound we need to avoid an exponential explosion in this case. The resulting reduction, of course, says nothing about whether recognition in LC is NP-hard. Intuitively, NP-complete problems involve a significant amount of communication among their parts. For example, if we flip the value of a variable x then this has an effect to the whole formula. When we bound this communication in this way, then we fall in the complexity hierarchy from NP-complete to membership in NL. What is needed is a reduction that uses higher-order categories in order to avoid this.

1.5.1 Example of the LC-embedding of BD-NFPO-SAT

For simplicity, assume that the distance bound suffices to cover all of the following portion of a formula:

$$(\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_5 \vee x_1 \vee x_2) \wedge (\neg x_6 \vee \neg x_7 \vee \neg x_8) \wedge (x_6 \vee x_1 \vee x_3) \wedge (x_1 \vee x_7 \vee x_8) \wedge \dots$$

In the clause $(x \vee y \vee z)$ the literals y and z are called *neighbours* of x. Only variables in the first and the third clause can place locks. If some $\neg x$ has placed a lock, then with respect to every other occurrence of x except the last one, the neighbours of x should propagate the lock placed by x. In the clause with the last occurrence of x, the neighbours of x unlock the lock previously imposed by $\neg x$. Below, by "propagate" we refer to the fact that if a positive literal is selected to be true then it should rewrite all locks concerning all variables occurring negatively before itself with the following two constraints: (a) it does not propagate locks placed by its own negation, and (b) it does not propagate locks from variables with instances only before, i.e., to the left of, its clause.

A simple form of this reduction follows: recall from Remark 1.3.2 that we can have a single symbol corresponding to a clause. For the above formula we have the string $w_1 w_2 w_3 w_4 w_5$, where w_1, w_2, w_3, w_4, w_5 are all different symbols of an alphabet. The target category we want to deduce is A/A. We construct the lexicon as follows: $f(w_1) = \{A/A/L_1, A/A/L_2, A/A/L_3\}$, where $A/A/L_1, A/A/L_2, A/A/L_3$ correspond to the locks placed by $\neg x_1, \neg x_2, \neg x_3$ respectively. $f(w_2) = \{L_1/L_1, L_2, L_3/L_3\}$, because x_5 and x_1 can unlock x_2 (L_2) which occurs for the last time in the second clause, x_5 and x_2 can propagate the lock for x_1 (L_1/L_1), and x_5 can propagate the lock for x_3 (L_3/L_3). In the same fashion, we construct the rest of the lexicon, where $L_{i_1 i_2 \dots i_k}$ denotes a composite lock, corresponding to the k locks placed by $x_{i_1}, \dots, x_{i_k}$:
$f(w_3) = \{L_1/L_{16}, L_2/L_{26}, L_3/L_{36}, A/A/L_6,$
$L_1/L_{17}, L_2/L_{27}, L_3/L_{37}, A/A/L_7, L_1/L_{18}, L_2/L_{28},$
$L_3/L_{38}, A/A/L_8\}$, $f(w_4) = \{L_3, L_{36}/L_6, L_{37}/L_7, L_{38}/L_8, L_6, L_{16}/L_1,$
$L_{26}/L_2, L_{36}/L_3, L_3, L_{36}/L_6, L_{37}/L_7, L_{38}/L_8\}$, $f(w_5) = \{L_7, L_8\}$.
Notice that the above formula is satisfiable, e.g., by $x_1 = 1, x_5 = 1, x_6 = 0, x_3 = 1, x_7 = 1$, corresponding to $A/A/L_1\ L_1/L_1\ L_1/L_{16}\ L_{16}/L_1\ L_1$, from which A/A can be derived.

1.6 LOGCFL-Hardness of LC

In the previous sections we developed some machinery, based on ordered satisfiability, in order to show hardness results for LC and LP. A restriction of this machinery, in which the parameter of the longest distance between two appearances of a variable is bounded, exhausts its limits for a logarithm in the number of variables. The reason is that if the distance is more than a logarithm then the reduction is no longer polytime (or log-space). We also saw in Section 1.2 that LC is NL-hard.

In this section we use another approach, which proves a stronger hardness result for LC, namely that it is LOGCFL-hard. All of our reductions belong to L, which is the class of languages characterized by their decidability with deterministic logarithmic space Turing Machines. One characterization of LOGCFL is as the class of all languages log-space reducible to a Context-Free Language (CFL). It contains NL (Sudborough 1978) and is contained in P. Cook (1985) contains more information on LOGCFL. All told, we have the following containments relative to our problem of interest:

$$L \subseteq NL \subseteq LOGCFL \subseteq P \subseteq NP$$

None of these containments is known to be proper, although it is widely conjectured that every one of these containments is proper.

Theorem 1.6.1. *LC is hard for LOGCFL.*

Note that this is not a proof of LOGCFL-completeness. If we knew that LC belonged to LOGCFL, we would know that LC is in polytime.

Proof. Fix an arbitrary language $A \in LOGCFL$. By the definition of LOGCFL, there exists a context-free language L and a log-space computable function f such that, for all x, $x \in A$ iff $f(x) \in L$. We prove that there exists a log-space computable function $g(x)$ such that $x \in A$ if and only if $g(x) \in LC$. We require the following well-known lemma:

Lemma 1.6.2. *If L is a CFL then $L \backslash \{\epsilon\}$ is also context-free, where ϵ is the empty string.*

Since $L \backslash \epsilon$ is context-free, there exists a CFG G in Greibach normal form such that $L \backslash \epsilon = L(G)$. Two cases arise:

1. If $\epsilon \in L$, then $g(x)$ is computed as follows. If $f(x) = \epsilon$ then let $g(x)$ be some fixed accepting instance for LC. Otherwise, from G and $f(x)$, we construct as $g(x)$ an instance of LC with string $f(x)$ and a Lambek Grammar defined as follows: for every Greibach-normal rule, $N_0 \longrightarrow t\, N_1 \ldots N_n$, where t is terminal and the N_i are non-terminals, add to the categories assigned to t in the Lambek Grammar the category $N_0/N_n/\ldots/N_1$.

2. $\epsilon \notin L$. This case is handled similarly to the above except that, if $f(x) = \epsilon$, then let $g(x)$ be some fixed non-accepting instance of LC.

$\square$

Bibliography

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of 3rd Annual IEEE Symposium on the Foundations of Computer Science*, pp. 151–158.

Cook, S. A. (1985). A taxonomy of problems with fast parallel algorithms. *Information and Control*, **64**(1-3):2–22.

Doerre, J. (1996). Parsing for semidirectional lambek grammar is NP-complete. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, **13**(2):90–102.

Florencio, C. C. (2002). A note on the complexity of the associative-commutative lambek calculus. In *Proceedings of the 6th Int'l Workshop on Tree Adjoining Grammar and Related Framework*, pp. 101–106.

Hepple, M. (1992). Chart parsing lambek grammars: Model extensions and incrementatlity. In *Proceedings of the 14th Int'l Conference on Computational Linguistics*.

Kanovitch, M. (1991). The multiplicative fragment of linear logic is NP-complete. Technical Report X-91-13, University of Amsterdam, 1TL1 Prepublication Series.

Kanovitch, M. (1992). Horn-programming in linear logic is NP-complete. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*, pp. 200–210.

Kurtonina, N. and M. Moortgat (1996). Structural control. In P. Blackburn and M. de Rijke, eds., *Specifying Syntactic Structures*. CSLI Publications.

Lincoln, P., J. Mitchell, A. Scedrov, and N. Shankar (1990). Decision problems on propositional linear logic. In *Proceedings of 31st Annual IEEE Symposium on the Foundations of Computer Science*.

Moot, R. and Q. Puite (1999). Proof nets for multimodal categorial grammars. In G.-J. M. Kruijff and R. T. Oehrle, eds., *Proceedings of the Conference on Formal Grammar*.

Morrill, G. (1996). Memoisation of categorial proof nets: parallelism in categorial processing. Technical Report LSI-96-24-R, Dept. de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya.

Penn, G. (2002). A graph-theoretic approach to sequent derivability in the lambek calculus. In *Electronic Notes in Theoretical Computer Science: Proceedings of the Conference on Formal Grammar / 7th Meeting on Mathematics of Language*, volume 53.

Pentus, M. (1993). Lambek grammars are context free. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*, pp. 429–433.

Pentus, M. (1997). Product-free lambek calculus and context-free grammars. *Journal of Symbolic Logic*, **62**(2):648–660.

Pentus, M. (2003). Lambek calculus is np-complete. Technical Report TR-2003005, CUNY Graduate Centre Ph.D. Program in Computer Science.

Sudborough, I. H. (1978). On the tape complexity of deterministic context-free languages. *Journal of the ACM*, **25**(3):405–414.