A Note on Log Space Optimization

C. Àlvarez
B. Jenner

Report LSI–92–30–R

A Note on Log Space Optimization [*]

Carme Àlvarez [†]
Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica Catalunya
Pau Gargallo 5, 08028 Barcelona, Spain
Birgit Jenner [‡]
Fakultät für Informatik
Technische Universität München
Arcisstr. 21, 8000 München 2, Germany

December, 30th 1992

Abstract

We show that computing iterated multiplication of word matrices over $\{0,1\}^*$, using the operations maximum and concatenation, is complete for the class optL of log space optimization functions. The same problem for word matrices over $\{1\}^*$ is complete for the class FNL of nondeterministic log space functions. Improving previously obtained results, we furthermore place the class optL in AC^1, and characterize FNL by restricted log space optimization functions.

[*]Partially supported by DAAD and Spanish Government (Acción Integrada 1992 131–B, 313–AI–e–es/zk).

[†]Research supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM). E-mail address: alvarez@lsi.upc.es

[‡]On leave until March 1995, visiting Dept. Llenguatges i Sistemes Informàtics, Universitat Politècnica Catalunya with a Habilitationsstipendium of Deutsche Forschungsgemeinschaft (DFG-Je 154 2/1). E-mail address: jenner@lsi.upc.es

1 Introduction

The class NC classifies those problems that are solvable rapidly in parallel with a feasible amount of processors. NC has been shown to include a large and interesting variety of problems from different areas (like, e.g., the theory of linear algebra, formal languages, automata, graphs). Structurally, NC can be defined as a hierarchy of function classes NC^k, $k \geq 1$, where NC^k contains all functions computable by a uniform bounded fan-in circuit family C_n of $n^{O(1)}$ size and $O((\log n)^k)$ depth [Co]. With the exception of some few examples, most problems known to be in NC are known to be in NC^2, and can be further classified according to Cook's taxonomy (see Figure 1 below).

$$\mathrm{NC}^1 \subseteq \mathrm{FL} \subseteq \mathrm{FNL} \subseteq \begin{array}{c}\mathrm{FLOGCFL} \subseteq \mathrm{AC}^1 \\ \mathrm{NC}^1(\mathrm{DET})\end{array} \subseteq \mathrm{NC}^2 \qquad [\mathrm{Co}]$$

Figure 1: Cook's taxonomy

NC^1(DET) denotes the class of functions NC^1 Turing reducible to computing the determinant of an integer matrix, AC^1 (respectively, FLOGCFL) the class of functions computable by uniform unbounded fan-in or semi-unbounded fan-in circuits of logarithmic depth, and FNL (respectively, FL) denotes the class of functions computable with nondeterministic (deterministic) logarithmic space. (Note that FNL, FLOGCFL and NC^1(DET) are named NL*, CFL*, and DET*, respectively, by Cook.) None of the inclusions is known to be proper and no relationship between NC^1(DET) and FLOGCFL or AC^1 is known.

Recently, two further subclasses of NC^2 turned out to be interesting: optL, the class of logarithmic space optimization problems and #L, the class of logarithmic space counting problems. These classes were introduced in [AJ] and characterized by complete problems in terms of graphs and automata. Subsequently, a tight relationship between #L and NC^1(DET) has been discovered in [Da], [To], and [Vi]: #L can be characterized by the iterated product of positive integer matrices, and the closure of #L under NC^1 Turing reducibility coincides with the class NC^1(DET) that is characterized by the iterated product of integer matrices.

As a result of a considerable amount of recent research, iterated multiplication problems are now known to be complete for any of the classes of Figure 1 (see [Ba] [BC] [CM] [Co] [BM], and [IL], which gives also an overview).

In this note, we study properties of the class FNL of nondeterministic log space functions and the class optL of log space optimization functions. In [AJ] it was shown that FNL $\subseteq$ optL $\subseteq$ NC^2. The possible "difference" between FNL and optL is emphasized by the fact that computing the maximal word function for deterministic finite-state automata is complete for FNL, whereas for nondeterministic automata it is complete for optL. The maximal word function computes, for an automaton and a word x, the lexicographically greatest word that is smaller than or equal to x and accepted by the automaton. (Subsequent investigations of the complexity of maximal word problems, e.g. for context-free languages, are done in [Vi], [BP], and [ABP].)

In this note, we characterize optL by the problem of computing the iterated product of word matrices over $\{0,1\}^*$, using the operations maximum and concatenation (instead of "+" and "$\cdot$", respectively). We show that hence optL is already included in AC^1. Furthermore, we characterize FNL by optimization functions defined via restricted, so-called "confluent" NL transducers. We show that when the word matrices in the above problem contain only entries from $\{1\}^*$, then a complete problem for the class FNL is obtained. Both facts mirror the possible weakness of FNL as compared to general optL functions.

The characterization of optL by iterated multiplication might be useful for a deeper understanding of the relationships between the classes optL, FLOGCFL, AC^1 and NC^1(DET). As Cook has pointed out, with the exception of very few examples most natural problems in NC^2 are known to be contained in FLOGCFL or NC^1(DET). One example of a natural problem in AC^1 not known to be in FLOGCFL is computing the iterated product of integer matrices using the operations maximum (or minimum) and addition [Co]. Note that the iterated matrix product that is complete for optL is a "mixture" of this matrix product and the one complete for NC^1(DET).

2 Preliminaries

An L *transducer (*NL *transducer)* is a deterministic (nondeterministic) logarithmic space-bounded Turing machine with (unbounded) output tape and accepting and rejecting final states. The output y of a computation of a transducer T on input x is the content of T's output tape when T halts. An output is only considered to be "valid", if T halts in an accepting state. We assume all L and NL transducers to be polynomially time bounded (by imposing a polynomial-time clock). Furthermore, we assume that any transducer has at least one valid output for each input. This is no restriction for NL transducers: Since NL is closed under complementation [Im] [Sz], the absence of a valid output could be checked by a precomputation.)

Note that for an NL transducer T on input of length n the length of an output is always bounded by a polynomial in n, and note that T can have exponentially in n many valid outputs, although the number of reachable configurations of T is bounded by a polynomial in n due to T's space bound. The output is not considered part of a configuration of T.

If a transducer computes exactly one valid value for each input it is called "single-valued". Clearly, any deterministic transducer is single-valued.

For a transducer T, let opt_T denote the function from $\{0,1\}^*$ to $\{0,1\}^*$ such that $opt_T(x)$ is the maximum valid output value of T on input x with respect to lexicographical order. We will consider only the maximum. All results obtained hold with appropriate changes also for the minimum.

We assume all functions to be from $\{0,1\}^*$ to $\{0,1\}^*$.

Definition 2.1

$$\begin{aligned}
\mathrm{FL} &:= \{f \mid f \textit{ is computed by some } \mathrm{L} \textit{ transducer } \};\\
\mathrm{FNL} &:= \{f \mid f \textit{ is computed by some } \mathrm{L} \textit{ transducer}\\
&\qquad\qquad \textit{that has access to an oracle from } \mathrm{NL}\};\\
\mathrm{SVNL} &:= \{f \mid f \textit{ is computed by some single-valued } \mathrm{NL} \textit{ transducer}\};\\
\mathrm{optL} &:= \{f \mid f = opt_T \textit{ for some } \mathrm{NL} \textit{ transducer } T\}.
\end{aligned}$$

For definitions of the uniform circuit classes NC^{k}, AC^{k} and the corresponding (Turing) reducibilities see [Co]. We assume log space uniformity, i.e., the description of the nth circuit C_n can be computed from 1^n with logarithmic space.

3 Properties of the class FNL

The function class FNL can be considered as "the" functional analog of the language class NL. The class FNL has been characterized as the closure of NL (as 0–1 functions) under NC^1, AC^0 or (non-adaptive) log space Turing reducibility [ABJ]. Note that FNL is named NL^* by Cook in [Co].

The following proposition shows that FNL can also be characterized using single-valued NL transducers.

Proposition 3.1 FNL = SVNL.

Proof. For the inclusion from left to right, let f be a function in FNL computed by an L transducer T with oracle $A \in$ NL. Since NL is closed under complementation [Im] [Sz], there are both an NL machine M_A that accepts A and an NL machine $M_{\overline{A}}$ that accepts $\overline{A}$, the complement of A. Construct an NL transducer T' that on input x of length n simulates T and whenever T starts to write a query q saves the current configuration c_q, guesses the answer 1 or 0 of the oracle, and accordingly starts a subroutine that consists in the simulation of M_A (guessed value 1) or $M_{\overline{A}}$ (guessed value 0) on q. Since q might be of size polynomial (in n), it can not be kept on the working tapes of T'. However, each bit i of q can be obtained via simulation of T starting in c_q up to the ith symbol written by T in its oracle tape. If a subroutine halts in an accepting (rejecting) state, the computation of T will be continued (aborted) by T'. Since during subroutines no output is produced, it is obvious that T' is single-valued with valid output $f(x)$.

For the inclusion from right to left, denote by $f_i(x)$ the ith bit of a function $f(x)$. Define the individual bits of f with

$$IB_f := \{x\$i\$b \mid 1 \leq i \leq |f(x)|,\ b \in \{0,1\},\ f_i(x) = b\}.$$

For $f \in$ SVNL, clearly $IB_f \in$ NL. To compute f, an L transducer with oracle IB_f on input x of length n first computes the length l of $f(x)$. l is bounded by a polynomial p in n and is obtained by asking $x\$i\$0$ and $x\$i\$1$ in decreasing order for each i, $1 \leq i \leq p(n)$, until the oracle answers positively. The transducer then outputs each bit $f_i(x)$ for $1 \leq i \leq l$. $f_i(x)$ has value 1 if $x\$i\$1 \in IB_f$, and value 0 otherwise. $\square$

Since for a single-valued transducer, the value and optimal value is the same, Proposition 3.1 implies that FNL is a subclass of optL, a result already obtained in [AJ].

Corollary 3.2 [AJ] FNL $\subseteq$ optL.

We show now a further relationship between these function classes.

Nondeterministic log space functions also can be characterized in terms of optimization functions, if restricted, so-called "confluent" NL transducer are considered. We call a transducer M *confluent*, if for any two computations of M on input x that have the same accepting final configuration, the output of M is the same. (Note that the term "confluent" appears with a different meaning in the context of nondeterministic oracle machines in [SMB].)

For a confluent transducer, we can identify with each accepting configuration a unique output. An NL transducer has due to its logarithmic space bound only polynomially (in n) many configurations for each input x of length n. Hence, any confluent NL transducer is polynomially valued, i.e. the number of different valid outputs on input x is bounded by a polynomial in n. But it is far from obvious whether every polynomially valued NL transducer can be made confluent.

We remark that for NP transducers, "confluence" is no restriction. It is not hard to show that for any NP transducer T, an equivalent confluent NP transducer can be constructed that computes the same set of output values than T.

Theorem 3.3 FNL $= \{f \mid f = opt_T$ *for some confluent* NL *transducer* T $\}$.

Proof. For the inclusion from right to left, let $f = opt_T$ for a confluent NL transducer T. Note that on input x of length n there are at most $p(n)$ different accepting configurations for a polynomial p due to the logarithmic space bound. Since T is confluent, we can identify with each accepting configuration c of T an output o_c. Define the following set OB that provides information about the output bits occuring up to o_c:

$$OB := \{x\$c\$l\$i\$b \mid \text{on input } x, T \text{ produces up to the final accepting configuration } c \text{ an output of length } l \text{ whose } i\text{th bit is } b\}.$$

It is not hard to see that OB is a variant of the graph accessibility problem and therefore contained in NL.

In order to find the final accepting configuration m that corresponds to the maximal output of T, i.e. $o_m = opt_T(x)$, an L transducer T' cycles through all possible configuration in lexicographical order according to the following program.

```
input $x$ of length $n$
$m := 1$
for $c := 1$ to $p(n)$ do
    begin
    if $o_c > o_m$ then $m := c$
    end
output $o_m$
end
```

For the comparisons $o_c > o_m$ and for the output of o_m, T' uses the oracle OB. With queries to OB, T' computes the lengths of both o_c and o_m, and for o_c, o_m of the same length, it can find the first bit (from the left) that (possibly) separates them. Thus, T' can decide $o_c > o_m$. Furthermore, when finally a configuration m is found that outputs the maximum string o_m on input x, again with queries to OB, T' obtains every bit of o_m and produces it as output.

For the inclusion from left to right, a single-valued NL transducer that exists by Proposition 3.1 for functions in FNL can be forced to have just one final accepting configuartion by standard methods. $\square$

It seems unlikely that Theorem 3.3 could be strengthened to hold for arbitrary polynomially valued transducers. In general, a log space machine can not compare more than a constant number of outputs of polynomial size in order to find the maximum one. It is essential for the proof of Theorem 3.3 that the output values of the NL transducer have a short description by an accepting final configuration. In the next section we will show that this is mirrored by the fact that FNL has a complete iterated matrix product over an infinite structure where the values in the matrices are compressible.

4 Complete functions

As mentioned in the introduction, many complexity classes within NC^2 have characterizations by iterated multiplication problems (see [IL] for a survey). We are interested here in the following general case of iterated matrix multiplication.

Let $(D; op_1, op_2)$ be a set D with binary operations op_1, op_2 defined on D. The *general iterated matrix product* $(D; op_1, op_2)$ ITMATPROD is defined as:

> **Input:** m $n \times n$ matrices $A^1, A^2, \ldots, A^m$ with entries from D and an index ij, $1 \le i, j \le n$.
>
> **Compute:** the ijth entry of the iterated matrix product $A^1 \cdot A^2 \cdots A^m$, where "+" and "·" are taken as the operation op_1, op_2.

The *general matrix product* $(D; op_1, op_2)$ MATPROD is defined by setting $m = 2$.

Figure 2 presents an overview about the known complexity results for the iterated product.

ITMATPROD	Class	
$(\mathbf{Z}; +, \cdot)$	$NC^1(DET)$	[Co]
$(\mathbb{N}; +, \cdot)$	#L	[Da] [To] [Vi]
$(\mathbb{N}; MAX, +)$	AC^1	[Co] (completeness not known)
$(\{0,1\}^*; MAX, \circ)$	optL	Theorem 4.1
$(\{1\}^*; MAX, \circ)$	FNL	Theorem 4.2
$(\{0,1\}; \vee, \wedge)$	FNL	[Co]

Figure 2: General Iterated Matrix Products

We show in the following that $(\{0,1\}^*; MAX, \circ)$ ITMATPROD is complete for the class optL, whereas $(\{1\}^*; MAX, \circ)$ ITMATPROD is complete for FNL. Here "MAX" denotes the maximum of two elements and "$\circ$" denotes concatenation. We assume that there is an absorbing element $\perp$ for $\circ$ that is the identity for MAX; i.e. it holds for each $x \in \{0,1\}^* \cup \{\perp\}$, $x \circ \perp = \perp \circ x = \perp$, and $MAX(x, \perp) = x$. The identity for $\circ$ is λ.

The two completeness results differ in the reducibilities used. Completeness for FNL is via NC^1 Turing reducibility, whereas for optL the completeness is obtained for the following stronger *functional log space many-one reducibility*: A function f is log space many-one reducible to a function g, if there exist functions $h_1, h_2 \in$ FL such that for all x it holds: $f(x) = h_2(g(h_1(x)))$.

Theorem 4.1 $(\{0,1\}^*; MAX, \circ)$ *ITMATPROD is complete for* optL.

Proof. For given m $n \times n$ matrices $A^1, \ldots, A^m$ with entries from $\{0,1\}^*$ the $(\{0,1\}^*; \text{MAX}, \circ)$ ITMATPROD results in computing

$$(*)\quad \begin{array}{l} \text{MAX}^n_{k_{m-1}=1}(\cdots \text{MAX}^n_{k_2=1}(\text{MAX}^n_{k_1=1}(A^1_{ik_1} \circ A^2_{k_1k_2}) \circ A^2_{k_2k_3}) \circ \cdots) \circ A^n_{k_{m-1}j}) \\ = \text{MAX}^n_{k_1,k_2,\ldots,k_{m-1}=1}(A^1_{ik_1} \circ A^2_{k_1k_2} \circ \ldots \circ A^n_{k_{m-1}j}). \end{array}$$

An NL transducer T successively guesses the indices k_1 up to k_{m-1} in binary and outputs the corresponding matrix entries $A^1_{ik_1}, \ldots, A^m_{k_{m-1}j}$ according to the following program:

```
input $A^1, A^2, \ldots, A^m; i, j \quad (1 \le i, j \le n)$
$s := i$
for $l := 1$ to $m$ do
    begin
    if $l \neq m$ then guess $t$ $(1 \le t \le n)$ else $t := j$
    output $A^m_{st}$
    if $A^m_{st} = \perp$ then accept else $s := t$
    end
accept
end
```

Here "A^m_{st}" denotes entry st of matrix A^m. Note that T always accepts, but its maximal value is the value of $(*)$.

For the hardness, let f be a function in optL such that for an NL transducer T, $f(x) = opt_T(x)$ for all $x \in \{0,1\}^*$. Let x be an input of length n, and denote by c_0 the start configuration and by c_a the unique accepting configuration of T. W.l.o.g. we can assume that every computation path of T has length exactly $p(n)$ for a polynomial p. Define for T and x the transition matrix A with entries $A_{ij} \in \{0, 1, \lambda, \perp\}$ as follows, where $c_i \xrightarrow{l} c_j$ means that the configuration c_j is reachable from the configuration c_i in one step whereby an output $l \in \{0,1\}$ or, if $l = \lambda$, no output is produced:

$$A_{ij} = \begin{cases} l, & l \in \{0,1,\lambda\},\ c_i \xrightarrow{l} c_j \\ \perp, & \text{else.} \end{cases}$$

A^t_{ij} contains the maximal output that may occur between configuration i and configuration j in a computation of T on input x of length t. Clearly, then $f(x) = A^{p(n)}_{0a}$. The matrix A can be produced easily from the transition table of T with a log space many-one reduction. □

If the alphabet is restricted to a single element, the problem turns out to be FNL complete.

Theorem 4.2 *($\{1\}^*$; MAX, $\circ$) ITMATPROD is complete for* FNL.

Proof. For the containment, a confluent NL transducer T on input of m $n \times n$ matrices $A^1, \ldots, A^m$ just follows the algorithm of Theorem 4.1, but instead of producing an output $\neq \perp$ it counts the number of output symbols (i.e. the number of ones) in binary on its worktape. Before accepting, T deletes everything but this number from its worktapes. As in the proof of Theorem 4.1, the maximal valid output of T computes $(*)$, but now T is confluent, and by Theorem 3.3 $opt_T \in$ FNL.

For the hardness, note that computing the transitive closure of a boolean matrix is complete for FNL (with respect to NC^1 reducibility) as shown in [Co] and can be simulated by ($\{\lambda\}$; MAX, $\circ$) ITMATPROD using, respectively, $\perp, \lambda$, MAX, $\circ$ instead of $0, 1, \vee, \wedge$. □

We show now that ($\{0,1\}^*$; MAX, $\circ$) ITMATPROD can be computed by AC^1 circuits. This follows from the fact that the product of two matrices ($\{0,1\}^*$; MAX, $\circ$) MATPROD can be computed in AC^0. The proof generalizes to the statement that if $(D; op_1, op_2)$ MATPROD $\in$ AC^0, then $(D; op_1, op_2)$ ITMATPROD $\in$ AC^1.

Theorem 4.3 $(\{0,1\}^*; MAX, \circ)$ *ITMATPROD* $\in \mathrm{AC}^1$.

Proof. First, we claim that $(\{0,1\}^*; \mathrm{MAX}, \circ)$ MATPROD is contained in log space uniform AC^0. In [CSV] it was shown that the maximum of a list of n strings of n bits each can be computed by a non-uniform AC^0circuit family C_n. It is not hard to see that these circuits in fact can be made log space uniform, i.e. that there exists a function in FL that computes the description of circuit C_n on input 1^n. Furthermore, the concatenation of two strings x, y of variable size l_1, l_2 can be computed by a log space uniform AC^0 circuit family D_n, provided that x, y are coded appropriately padded up to size m with $m \geq l_1, l_2$. Then, $MAX_{k=1}^{n} A_{ik} \circ A_{kj}$ can be computed by using the circuits for the maximum and concatenation as subcircuits.

Second, we use the AC^0 circuits for $(\{0,1\}^*; \mathrm{MAX}, \circ)$ MATPROD to compute the iterated product in $O(\log k)$ levels of suitable matrix products. For given matrices $A^1, \ldots, A^m$, let 2^k be the smallest power of 2 such that $m \leq 2^k$, and if $m \leq 2^k$, let any of the matrices $A^{m+1}, \ldots, A^{2^k}$ be the $n \times n$ identity matrix. Then compute $A^{2i-1} \cdot A^{2i}$ for $1 \leq i \leq 2^{k-1}$ in parallel, and repeat the process for the resulting 2^{k-1} matrices $k = \log m$ times. This yields an AC^1 circuit for $(\{0,1\}^*; \mathrm{MAX}, \circ)$ ITMATPROD. □

Since the reduction of optL functions to $(\{0,1\}^*; \mathrm{MAX}, \circ)$ ITMATPROD described in the proof of Theorem 4.1 can be computed easily in log space and AC^1 is closed under log space reductions, we can improve the upper bound NC^2 of optL given in [AJ]. This result has been subsequently proved in [ABP] with different methods.

Corollary 4.4 $\mathrm{optL} \subseteq \mathrm{AC}^1$.

Acknowledgement

The results of this note were presented at the first Dagstuhl-Seminar on "Structure and Complexity Theory", February 1992. We thank the organizers Klaus-Ambos-Spies, Steve Homer, and Uwe Schöning. We also thank José Balcázar and Jacobo Torán for helpful comments on an earlier version of this note.

References

[ABP] E. ALLENDER, D. BRUSCHI AND G. PIGHIZZINI, The complexity of computing maximal word functions, Manuscript, 1992.

[AJ] C. ÀLVAREZ AND B. JENNER, A Very Hard Log-Space Counting Class, *Theoretical Computer Science* **107**,1 (1993), pp. 3–30. (preliminary version in Proc. 5th IEEE *Structure in Complexity Theory Conference* (1990), pp. 154–168.)

[ABJ] C. ÀLVAREZ, J.L. BALCÁZAR AND B. JENNER, Functional Oracle Queries As a Measure of Parallel Time, Proc. 8th STACS Conference, *Lecture Notes in Computer Science* **480** (Springer, Berlin, 1991), pp. 422–433. (to appear in *Mathematical Systems Theory* under the title "Adaptive logspace reducibility and parallel time")

[Ba] D.A. BARRINGTON, Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1, *J. Comput. Syst. Sci.* **38** (1989), pp. 150–164.

[BM] M. BEAUDRY AND P. MCKENZIE, Circuits, Matrices, and Nonassociative Computation, Proc. 7th IEEE *Structure in Complexity Theory Conference* (1992), pp. 94–106.

[BC] M. BEN-OR AND R. CLEVE, Computing Algebraic Formuls Using a Constant Number of Registers, *SIAM Journ. of Comput.* **21** (1992), pp. 54–58.

[BP] D. BRUSCHI AND G. PIGHIZZINI, The complexity of computing maximal word functions, Proc. 8th FCT Conference, *Lecture Notes in Computer Science* **529**, (Springer, Berlin, 1991), pp. 157–167.

[CSV] A.K. CHANDRA, L. STOCKMEYER AND U. VISHKIN, Constant Depth Reducibility, *SIAM Journ. of Comput.* **13**, 2 (1984), pp. 423–439.

[Co] S.A. COOK, A taxonomy of problems with fast parallel algorithms, *Information and Control,* **64** (1985), pp. 2–22.

[CM] S.A. Cook and P. McKenzie, Problems Complete for Deterministic Logarithmic Space, *Journ. of Algorithms* **8** (1987), pp. 385–394.

[Da] C. Damm, DET= $L^{\#L}$, Manuscript, 1991.

[Im] N. Immerman, Nondeterministic space is closed under complement, *SIAM Journ. of Comput.* **17**,5 (1988), 935–938.

[IL] N. Immerman and S. Landau, The Complexity of Iterated Multiplication, Manuscript, 1992. (preliminary version in Proc. 4th IEEE *Structure in Complexity Theory Conference*, 1989, pp. 104-111.)

[SMB] A. Selman, X. Mei-rui, and R. Book, Positive relativizations of complexity classes, *SIAM J. Comput.* **12** (1983), pp. 565–579.

[Sz] R. Szelepcsényi, The method of forced enumeration for nondeterministic automata, *Acta Informatica* **26** (1988), pp. 279–284.

[To] S. Toda, Counting Problems Computationally Equivalent to Computing the Determinant, Technical Report CSIM 91-07, Dept. Comp. Sci. and Inf. Math., Univ. of Electro-Communications, Tokyo, May 1991.

[Vi] V. Vinay, Counting Auxiliary Pushdown Automata and Semi-Unbounded Arithmetic Circuits. Proc. 6th IEEE *Structure in Complexity Theory Conference*, 1991, pp. 270–284.