Logarithmic space counting classes

Carme Àlvarez
Birgit Jenner

Report LSI–90–13

Resum: En aquest article definim les classes de comptatge per a espai logarítmic #L, opt-L i span-L, anàlogament a les classes de comptatge definides per temps polinòmic. Les funcions de #L compten el nombre de còmputs acceptadors de màquines de Turing indeterministes que treballen amb espai logarítmic. Les funcions de span-L compten el nombre dels diferents valors de sortida que pot calcular una màquina indeterminista amb cinta de sortida i que treballa en espai logarítmic. I opt-L es defineix com la classe de funcions que calculen el màxim valor de computat per màquines del tipus ja esmentat. Les funcions completes que presentem per aquestes classes estan definides en termes d'autòmats finits i grafs.

Demostrem que les classes #L i opt-L estan contingudes en NC^2, però que si opt-L estés continguda en #L aleshores tots els llenguatges en NL serien acceptats per màquines indeterministes i inambígües. Sorprenentment la classe span-L sembla que sigui molt més complexa que les classes #L i opt-L. Per exemple, es pot demostrar que si les funcions de span-L poguessin ser calculades amb una màquina indeterminista que treballa en temps polinòmic llavors la Jerarquia Polinòmica colapsaria, P = NP = PH = P(#P). Aquest resultat s'obté gràcies a que span-L i #P són bastant semblants: span-L $\subseteq$ #P, i qualsevol funció de #P es pot representar com una diferència entre dues funcions de span-L. En canvi tenim que si #P $\subseteq$ span-L llavors NL = P = NP.

Finalment, proposem versions restringides de les classes de comptatge span-L i opt-L i demostrem, per exemple, que si opt-L coincidís amb una de les seves versions restringides llavors L = NL.

Resumen: En este artículo definimos las clases de funciones de conteo para espacio logarítmico análogas a las clases de conteo para tiempo polinómico. Las funciones de #L cuentan el número de cómputos acceptadores de máquinas de Turing indeterministas que trabajan en espacio logarítmico. Las funciones de span-L cuentan el número de valores distintos que puede escribir una máquina indeterminista con espacio de trabajo acotado por un logaritmo. Y la clase opt-L calcula el máximo de los valores de salida calculados por una máquina del mismo tipo que el anterior. Las funciones completas que presentamos para estas clases estan definidas en términos de autómatas finitos y máquinas de Turing.

Demostramos que las clases #L y opt-L estan contenidas en NC^2, pero si opt-L estuviera contenida en #L tendríamos que los lenguajes en NL serían aceptados por máquinas indeterministas con epacio de trabajo acotado por un logaritmo y inambíguas, es decir con un único computo aceptador en caso de que exista. Sorprendentemente la clase span-L parece mucho más compleja que las dos anteriores. Se puede demostrar, por ejemplo, que si las funciones de span-L se pudieran calcular en tiempo polinómico entonces la Jerarquia Polinómica colapasaría, P = NP = PH = P(#P). Esto se debe a que las clases span-L y #P son bastante parecidas: span-L $\subseteq$ #P, y cualquier función en #P se puede obtener de una substracción entre dos funciones span-L. Pero en cambio si #P $\subseteq$ span-L entonces NL = P = NP.

También proponemos versiones restringidas de las clases de conteo span-L y opt-L y demostramos, por ejemplo, que si opt-L coincidiera con una de sus versiones restringidas entonces L $\subseteq$ NL.

Zusammenfassung: Wir untersuchen die logarithmischen Funktionen-Platzklassen #L, opt-L, and span-L, die wir analog zu den entsprechenden polynomiellen Zeitklassen definieren: Funktionen in #L zählen die Anzahl der akzeptierenden Berechnungen logarithmisch platzbeschränkter Turingmaschinen; Funktionen in span-L zählen die Anzahl der verschienden Ausgabewerte solcher Maschinen mit zusätzlichem Ausgabeband; und Funktionen in opt-L berechnen den maximalen Ausgabewert derartiger Maschinen. Wir konstruieren funktionale Varianten NL-vollständiger Graphen- und Automatenprobleme, die vollständig für diese drei Klassen sind.

Wir zeigen, daß sowohl #L als auch opt-L in NC^2 und damit in FP enthalten sind. span-L jedoch scheint überraschenderweise eine sehr viel "schwierigere" Zählklasse zu sein als #L und opt-L. Wir zeigen, daß span-L-Funktionen genau dann in polynomieller Zeit berechenbar sind, wenn P = NP = PH = P(#P), d.h. wenn P(#P) und alle Klassen der Polynomiellen Hierarchie in P enthalten sind. Dieses Resultat folgt, weil span-L und #P sehr ähnlich sind: span-L $\subseteq$ #P, und jede Funktion in #P kann als Subtraktion zweier Funktionen in span-L dargestellt werden. #P $\subseteq$ span-L jedoch würde NL = P = NP implizieren.

Wir untersuchen ferner verschiedene Restriktionen der Klassen opt-L und span-L, und zeigen z.B., daß L = NL, wenn die Klasse opt-L mit einer ihrer eingeschränkten Versionen zusammenfallen sollte.

Logarithmic Space Counting Classes†

(April 4, 1990)

Carme Àlvarez *
Departament L.S.I.
Universitat Politècnica de Catalunya
Pau Gargallo 5
08028 Barcelona
Spain

Birgit Jenner **
Fachbereich Informatik
Universität Hamburg
Rothenbaumchaussee 67/69
2000 Hamburg 13
West-Germany

Abstract: We consider the logarithmic space counting classes #L, opt-L, and span-L, which are defined analogously to their polynomial time counterparts. We obtain complete functions for these three classes in terms of graphs and finite automata. We show that #L and opt-L are both contained in NC^2, but that, surprisingly, span-L seems to be a much harder counting class than #L and opt-L. We demonstrate that span-L-functions can be computed in polynomial time if and only if P = NP = PH = P(#P), i.e. iff the class P(#P) and all the classes of the polynomial time hierarchy are contained in P. This result follows from the fact that span-L and #P are very similar: span-L $\subseteq$ #P, and any function in #P can be represented as a subtraction of two functions in span-L. Nevertheless, #P $\subseteq$ span-L would imply NL = P = NP. We furthermore investigate various restrictions of the classes opt-L and span-L, and show, e.g., that if opt-L coincides with one of its restricted versions, then L = NL follows.

1. Introduction

During the past several years the topic of "counting" appeared in many different settings in complexity theory. In the case of logarithmic space, for example, powerful counting techniques revealed the *intrinsic* computational power of various machine models to achieve counting, above all NL, and LOGCFL (see [Im 88] [Sze 88] [Bo et al. 89]). On the other hand, counting was used to *increase* the computational power of polynomial time machines by defining counting variants of NP, like the function

† A preliminary version of this report appears in the proceedings of the international conference STRUCTURE IN COMPLEXITY THEORY '90.

* Work partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM).

** Work done while visiting the L.S.I. Department of UPC in Barcelona supported by Universität Hamburg and Deutsche Forschungsgemeinschaft (DFG-Forschungsstipendium Je 154/1-1).

classes opt-P (see [Kre 86]), #P (see [Va 79a] [Va 79b]), and span-P ([Kö Schö Tor 89], see also [Schö 88] [Kö 89]). These classes have been shown to contain interesting functional counting variants of NP-complete problems. Recently, Toda pointed out the enormous power that such classes can have: the whole polynomial time hierarchy is contained in P(#P) [To 89].

This raises the question, whether the phenomenon of increasing the computational power by counting is a general one and shows up by logarithmic space classes as well, and if so to what extent, or whether e.g. due to the intrinsic counting power of NL here no more power is gained. Stated alternatively, this question concerns the complexity of functional counting variants of NL-complete problems. For example, it is well-known that the non-emptiness problem $L(M) \neq \emptyset$? for a given finite automaton M is NL-complete with respect to logarithmic space reductions. This holds irrespectively of whether the automaton M is deterministic (DFA) [Jo 75] or nondeterministic (NFA). Consider the following functional counting variants of this problem: computing the number of words accepted by a DFA or NFA which are smaller than a given word x, the "ranking functions" for DFA or NFA, and the problem of computing the maximal word smaller than or equal to a given word x accepted by a NFA, the "maximal word function" for NFA. How difficult are these functions to compute? Can they, e.g., be computed by a deterministic log space Turing machine with oracle in NL?

We show that these functions are, respectively, many-one complete for the three logarithmic space counting classes #L, span-L, and opt-L, that we define analogously of their polynomial time counterparts: Functions in #L count the number of accepting computations of a nondeterministic logarithmic space bounded Turing machine, functions in span-L count the number of different output values of such a machine with additional output tape, and functions in opt-L compute the maximum of all output values.

Although it seems at first sight that these three classes have similar computational power, we will show that this is unlikely. span-L turns out to be a very much harder counting class than both #L and opt-L.

The paper is organized as follows. Section 2 contains all the necessary preliminaries and the definitions of the classes #L, span-L, and opt-L, and in Section 3 we show various functional counting variants of automata and graph problems to be complete for these classes.

Section 4 reveals the difference in the computational power of #L and opt-L on the one hand, and span-L on the other. First, we show that both #L and opt-L are contained in NC^2, and hence in polynomial time. The inclusion opt-L $\subseteq$ #L, however, implies unambNL = NL, i.e., that all languages in NL can be accepted unambiguously, with an unique accepting computation. In contrast, span-L seems to be a very hard log space counting class. In the remaining part of Section 4 we demonstrate that span-L is contained in #P, and that any function in #P is metric reducible to a function in span-L. The latter result is obtained by showing that span-L is powerful enough to compute the number of non-satisfying assignments of a Boolean formula. Thus, although the equality span-L = #P is unlikely, since it implies

NP $\subseteq$ NL, the classes span-L and #P are nevertheless very similar, when used as oracle. In particular, they share the ranking function for NFA as a complete function with respect to metric reducibility. The similarity of span-L and #P furthermore provides us with some information about the computational power of span-L and thus of the difficulty of computing the "ranking function" even of such a simple device as a NFA. Our results yield a new characterization of the class P(#P) as P(span-L), and as a consequence of the result by Toda [To 89] that P(#P) contains the whole polynomial time hierarchy (PH), we obtain that the "ranking function" of a NFA can be computed in polynomial time if and only if P = NP = PH = P(#P).

Similar results about the particular difficulty of computing the ranking function of languages in various other "small" (ambiguous) complexity classes have been published before, e.g., in [Go Si 85], [Be Go Sa 87], [He 87], and [Hu 88]. Our result extends this list by an even simpler case, i.e. the ranking of languages presented by NFAs, and furthermore by completeness results.

In Section 5 we investigate various restrictions of the classes opt-L and span-L defined via NL-machines with output size bounded logarithmically in the size of the input, and via NL-machines that produce their output deterministically. These restrictions are shown to lead to alternative characterizations of function classes defined by deterministic log space bounded Turing machines with oracle in NL and logarithmically bounded number of oracle queries. We prove, e.g., that if functions in opt-L could be computed by NL-machines which write their output deterministically, then L = NL would follow. Furthermore, we demonstrate that logarithmic restrictions on the size of the values of functions in span-L yield a class contained in NC^2.

2. The Log Space Counting Classes #L, span-L, and opt-L

In the following we will define function classes by considering certain counting operators defined over the computation tree of nondeterministic logarithmic space Turing machines with output (NL-transducer) or without output (NL-machine). To this effect, the computation graph of the machines should not contain cycles which lead to infinitely many accepting computations. This will be achieved by imposing additionally a polynomial time bound onto the NL-machines by attaching a clock. It is well known that as far as the corresponding language class NL is concerned the attachment of a polynomial time clock leads to the same class of accepted languages. This is also true for the one-way restriction of this class, 1NL, defined by (off-line) machines which read their input by moving the input head just from left to right (see e.g. [Ha Ma 81). In the following we will thus understand that any of the mentioned logarithmic space Turing machines (with or without output) are polynomial time bounded.

We understand a configuration of a NL-machine or NL-transducer to contain the input and work tape(s) head position, the work tape(s) content, the clock, the state z, and in the case of a transducer a further symbol denoting whether the transition table of the transducer specified that there occurs an output 0 or 1 or no output in z. (For simplicity, we only consider outputs in $\{0,1\}^*$.) In the initial

and (unique) final configuration of M, w.l.o.g. no output occurs. To write down a complete configuration thus $O(\log n)$ space is sufficient for inputs of length n. The NL-transducer have accepting and rejecting final states, and an output of a computation is only considered to be "valid", if the machine stops in an accepting state. Note that whereas the number of reachable configurations is bounded by a polynomial, the number of accepting computation paths and/or valid outputs is not in general. This number can become exponential.

All the sets and functions we consider are defined over the alphabet $\{0,1\}$. For a set A, its complement is denoted by CoA; and for a language class A (always in roman), the class of all complements of languages in A by CoA. We will use the prefix "F" to denote the class of functions (e.g. FL, FP) instead of the language class (L, P). The cardinality of a set A is represented by $\|A\|$. The set of natural numbers is denoted by N, and the length of a word x by $|x|$. For words x, y we denote by $x \leq y$ that x is smaller or equal than y with respect to *lexicographical order*. (Recall that this means that words are ordered according to length and for the same length according to alphabetical order.) The ranking function $rank_A : \{0,1\}^* \to \mathrm{N}$ of a set $A \subseteq \{0,1\}^*$ is defined with $rank_A(x) = \|\{w \in A \mid w \leq x\}\|$. The census function $cens_A$ of a set A is the function $cens_A : 1^* \to \mathrm{N}$ such that $cens_A(1^n) = \|\{w \in A \mid |w| \leq n\}\|$ (i.e. the restriction of $rank_A$ to 1^*).

By counting the number of different accepting computations of a NL-machine, we obtain the class #L, the logarithmic space analogon of the class #P, introduced by Valiant [Va 79a] [Va 79b].

Definition 2.1. For a NL-machine M define the function $acc_M : \{0,1\}^* \to \mathrm{N}$ such that $acc_M(x)$ is the number of accepting computations of M on x. Then

$$\#\mathrm{L} := \{ f \mid f = acc_M \text{ for some NL-machine } M \}.$$

The class opt-L is defined in terms of the maximum of all possible valid output values of a NL-transducer. This class is the logarithmic space analogon of the function class opt-P introduced by Krentel in [Kre 86]. (Note that, for simplicity, opt-L contains only maximization functions. The results obtained hold for the case of minimization functions, too.)

Definition 2.2. Let M be a NL-transducer. Define the function $opt_M : \{0,1\}^* \to \mathrm{N}$ such that $opt_M(x)$ is the maximum valid output value of M on input x with respect to lexicographical order, or if there is no valid output, then $opt_M(x)$ equals $\perp$. Define

$$\text{opt-L} := \{ f \mid f = opt_M \text{ for some NL-transducer } M \}.$$

The class span-L is defined in terms of the number of different valid outputs that occur in a computation tree of a NL-transducer (the "span" of the tree). The name "span-L" is chosen here to indicate the closeness to the analogue class span-P,

a class introduced in [Kö Schö Tor 89] (see also [Schö 88], [Kö 89]). Note that in [To Wa 89] the class span-P is named "#·NP".

Definition 2.3. Let M be a NL-transducer. Define the function $span_M : \{0,1\}^* \to$ N, where $span_M(x)$ denotes the number of different valid outputs that occur in the nondeterministic computation tree induced by M on input x, and $span_M(x) = 0$ if there are no valid outputs. Define

$$\text{span-L} := \{ f \mid f = span_M \text{ for some NL-transducer } M \}.$$

(Note that all the functions in the preceding definitions are specified as functions into N, to keep the consistency with earlier publications [Kre 86] [Kö Schö Tor 89]. We consider the output of our functions being codified as a binary string in a natural way.)

In [Kö Schö Tor 89] it was shown that opt-P $\cup$ #P $\subseteq$ span-P $\subseteq$ #NP, where #NP denotes the class of such functions that witness the number of accepting computations of NP-machines with oracles in NP. The corresponding relationships are easily shown to hold for the log space counting classes, too. (Note that in constrast to the polynomial time case, where opt-P $\subseteq$ span-P is known to hold for maximization functions, but is unknown for minimization functions (see [Kö 89]), in the case of logarithmic space the inclusion opt-L $\subseteq$ span-L holds for both kinds of optimization functions, since NL is closed under complementation [Im 88] [Sze 88]).

With the inclusion span-L $\subseteq$ #P, shown in Section 4 (Theorem 4.5.), we get the following inclusion diagram:

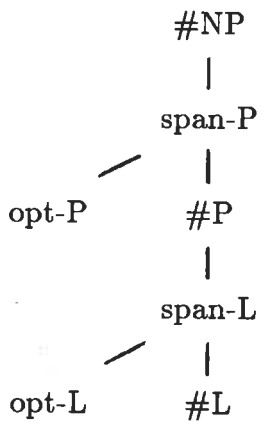

Figure 1: Inclusion Relations Among Log Space and Polynomial Time Counting Classes

Furthermore, it is easily verified that all of the three classes #L, span-L, and opt-L contain the class FL of functions computable with logarithmic space and that

they are closed under *logarithmic space functional many-one reducibility*, defined as follows: Let f, g: $\{0,1\}^* \to \mathrm{N}$. A log space functional many-one reduction from f to g is a function $h : \{0,1\}^* \to \{0,1\}^*$ with $h \in \mathrm{FL}$, such that for all $x \in \{0,1\}^*$ we have $f(x) = g(h(x))$. The class of functions which are many-one reducible to a class of functions F is denoted by $\mathrm{FL}_m(\mathrm{F})$.

We also consider the more powerful reducibility computed by a deterministic logarithmic space transducer with additional functional oracle, i.e., by a machine that has besides the work tape(s) and the output tape additional oracle query and oracle answer tapes. Depending whether the number of oracle queries is 1 or unbounded, we distinguish between *metric reducibility* $\mathrm{FL}_1(\cdot)$ and *functional Turing reducibility* $\mathrm{FL}(\cdot)$.

The polynomial time concepts $\mathrm{FP}_m(\cdot)$, $\mathrm{FP}_1(\cdot)$, and $\mathrm{FP}(\cdot)$ are obtained by considering polynomial time bounded machines instead of log space bounded machines. Note that the $\mathrm{FP}_1(\cdot)$-reducibility is equivalent to the metric reducibility defined in [Kre 86].

We denote by $\mathrm{P}(\cdot)$ the closure of language or function classes under deterministic polynomial time Turing reducibility and by $\mathrm{L}(\cdot)$ the closure under deterministic logarithmic space Turing reducibility.

3. Complete Functions

Complete functions for the classes #L, span-L, and opt-L can be obtained by "counting" variants of NL-complete automata and graph problems. For example, it is known that the emptiness problem $L(M) \stackrel{?}{=} \emptyset$ for a given finite automata M is NL-complete for deterministic or nondeterministic finite automata [Jo 75] [Jo Li La 76]. Consider the following functional counting versions of this problem:

- the *census function*: "Given an automaton M and 1^n, how many words of length n are accepted by M ?",

or its generalization

- the *ranking function*: "Given an automaton M and x, how many words lexicographically smaller than or equal to x are accepted by M ?"

and

- the *maximal word function*: "Given an automaton M and x, which is the lexicographically greatest word smaller than or equal to x accepted by M ?",

or its generalization

- the *maximal relative word function*: "Given an automaton M, where $L(M) \subseteq \Sigma^*$, a subalphabet $\Sigma' \subseteq \Sigma$, and $x \in \Sigma'^*$, which is the lexicographically greatest word $h(w) \leq x$ with $w \in L(M)$, when $h : \Sigma \to \Sigma' \cup \{\lambda\}$ is the homomorphism $h(a) := a$, if $a \in \Sigma'$, and $h(a) = \lambda$, otherwise ?",

To make the meaning of the maximal relative word function clearer, we will state an example:

Let $L(M) \subseteq \{0,1,b,c\}^*$ be the (finite) set $\{ c1b0c11bbc0, b11c1, 0001, 0cc1b0c1 \}$. Let $\Sigma' = \{0,1\}$, and let h be defined as above. Then the (lexicographically) greatest word $h(w)$ less or equal than $x = 0111$ equals 0101.

The results we will obtain concerning the complexity of these functions are summarized in the following table:

	DFA	NFA
ranking function	#L-complete	span-L-complete
census function	#L-complete	span-L-complete
maximal relative word function	opt-L-complete	opt-L-complete
maximal word function	$\in$ FL(NL)	opt-L-complete

Figure 2: Complete Functions for Log Space Counting Classes

Complete for #L is the ranking function for DFA, $f_{\#DFA}$. This is true even for the special case of the census function, or for the more general case of an unambiguous NFA, i.e., a NFA which accepts each word with an unique accepting computation.

$f_{\#DFA}$:
Input: an encoding of a DFA M and a string $x \in \{0,1\}^*$
Output: number of words lexicographically smaller than x accepted by M .

Complete for span-L is the ranking function for NFA, $f_{\#NFA}$. Here again, even computing the census function is already complete.

$f_{\#NFA}$:
Input: an encoding of a NFA M and a string $x \in \{0,1\}^*$
Output: number of words lexicographically smaller than x accepted by M .

Theorem 3.1.
(i) $f_{\#DFA}$ is many-one complete for #L;
(ii) $f_{\#NFA}$ is many-one complete for span-L.

Proof: (i) $f_{\#DFA} \in$ #L can be seen as follows. Construct a NL-machine N which, given $x \in \{0,1\}^*$ and a DFA M as input, "guesses" a word $y \leq x$ bit-by-bit and records for every new guessed bit the corresponding state of M. N accepts iff a final state is ever reached. Since the automaton M is unambiguous, there is exactly one computation path of N for any word smaller than or equal to x. Furthermore, the number of *valid* computation paths of N corresponds to the number of such words accepted by M.

For the hardness #L $\subseteq$ FL$_m(f_{\#DFA})$ let $f \in$ #L be given such that $f(x)$ equals the number of accepting computations of a NL-machine N for any input $x \in \{0,1\}^*$. Let $p(|x|)$ be the polynomial which bounds the running time of N. We have to

show that there exists a function h computable with logarithmic space such that $f(x) = f_{\#DFA}(h(x))$. This function will be the function $h(x) := (<N_x>, 1^{p(|x|)})$ where $<N_x>$ denotes the encoding of a complete state transition graph of a DFA which is constructed as follows: Let $C_{(N,x)}$ denote the set of all configurations of N on input x, let $c_{(start,x)}$ be the start configuration and $C_{(acc,x)}$ the set of all accepting configurations. Let furthermore $sink$ denote an element not contained in $C_{(N,x)}$. Then the following 5-tuple denotes the set of states, the alphabet, the transition function, the (unique) initial state, and the set of final states: $N_x := (\ C_{(N,x)} \cup \{sink\},\ C_{(N,x)} \cup \{sink\},\ \delta,\ c_{(start,x)},\ C_{(acc,x)}\ )$, where for all $c_i,\ c_j \in C_{(N,x)} \cup \{sink\}$:

$$\delta(c_i, c_j) := \begin{cases} c_j, & c_i \text{ reaches } c_j \text{ in one step in a computation of } N \text{ on } x \\ & (c_i \xrightarrow{c_j} c_j); \\ sink, & \text{otherwise} \quad (c_i \xrightarrow{c_j} sink). \end{cases}$$

The automaton N_x is deterministic and can obviously be constructed easily with logarithmic space. Furthermore it is clear that the construction ensures that the number of different computation paths of N on input x equals the number of different words accepted by N_x smaller than $1^{p(|x|)}$. Note that with a little more effort the construction can even be made such that the alphabet $C_{(N,x)} \cup \{sink\}$ of N_x equals $\{0,1\}$.

(ii) To see $f_{\#NFA} \in$ span-L construct a NL-machine N with output, which given a NFA M and $x \in \{0,1\}^*$ as input, does the following. At the beginning N "guesses" an initial state of M. N then "guesses" and outputs a word $y \leq x$ bit-by-bit, "guessing" and recording a new state of M consistent with each guessed bit and the transition table of M. N accepts iff a final state is ever reached. Thus only in this case the output is valid. Since the machine is ambiguous there may be more than one valid computation with the same output, but $span_N(<<M>, x>)$, the number of different valid outputs of N, corresponds to the number of words accepted by M.

For the hardness span-L $\subseteq \mathrm{FL}_m(f_{\#NFA})$ consider an arbitrary NL-transducer N which witnesses $f(x)$ via its span. We have to show that there exists a function h computable with logarithmic space such that $f(x) = f_{\#NFA}(h(x))$. h will be the function $h(x) := (<N_x>, 1^{p(|x|)})$, where $p(|x|)$ denotes the polynomial which bounds the running time of N, and $<N_x>$ denotes the encoding of a state transition graph of the NFA $N_x := (\ C_{(N,x)},\ \{0,1\},\ \delta,\ c_{(start,x)},\ c_{(acc,x)}\ )$, where $C_{(N,x)}$ denotes the set of all configurations of N on input x, $c_{(start,x)}$ the start configuration, $c_{(acc,x)}$ the unique accepting configuration, in which w.l.o.g. no output occurs, and for all $c_i,\ c_j \in C_{(N,x)}$, such that c_i reaches c_j in one step in a computation of N on x, and $b \in \{0,1,\lambda\}$ we define $(c_i \xrightarrow{b} c_j)$:

$$\begin{aligned} \delta(c_i, b) &= c_j, \quad \text{if the output in } c_j \text{ is } b \in \{0,1\}, \\ \delta(c_i, \lambda) &= c_j, \quad \text{if in } c_j \text{ no output occurs.} \end{aligned}$$

Note that since M is polynomially clocked this graph contains no cycles. Then it is ensured that the number of different words up to length $p(|x|)$, i.e., words smaller

than or equal to $1^{p(|x|)}$, which are accepted by N_x, is exactly the number of valid outputs the given NL-machine N can produce on input x. Furthermore it is obvious, that the construction of N_x given x can be done with logarithmic space. □

Complete for opt-L are the maximal relative word function for both DFA and NFA:

$f_{maxrelDFA}$:
Input: an encoding of a DFA M with $L(M) \subseteq \Sigma^*$, a finite set $\Sigma' \subseteq \Sigma$, and a string $x \in \Sigma'^*$
Output: lexicographically greatest word $h(w) \leq x$ such that $w \in L(M)$, where $h : \Sigma \to \Sigma' \cup \{\lambda\}$ is the homomorphism $h(a) := \begin{cases} a, & \text{if } a \in \Sigma', \\ \lambda, & \text{otherwise.} \end{cases}$
(If no such word exists, the output is $\perp$.)

$f_{maxrelNFA}$ is defined accordingly for a given NFA.

Note that the maximal word function

f_{maxDFA} (resp., f_{maxNFA}):
Input: an encoding of a DFA (resp., NFA) M and a string $x \in \{0,1\}^*$,
Output: lexicographically maximal word $y \in L(M)$ with $y \leq x$
(If no such word exists, the output is $\perp$)

is the special case of $f_{maxrelDFA}$ (resp., $f_{maxrelNFA}$), where $\Sigma' = \Sigma = \{0,1\}$. The function f_{maxNFA} is already complete for opt-L. On the other hand, f_{maxDFA} can be shown to be contained in FL(NL), a subclass of opt-L (see Theorem 4.2.).

Theorem 3.2.
(i) $f_{maxrelDFA}$ and $f_{maxrelNFA}$ are many-one complete for opt-L.
(ii) f_{maxNFA} is many-one complete for opt-L.
(iii) $f_{maxDFA} \in$ FL(NL).

Proof: (i) It suffices to show that $f_{maxrelNFA} \in$ opt-L, and that $f_{maxrelDFA}$ is hard for opt-L.

For $f_{maxrelNFA} \in$ opt-L, construct a NL-transducer N, which on input of an encoding of a nondeterministic automata M with $L(M) \subseteq \Sigma^*$, a finite set $\Sigma' \subseteq \Sigma$, and a string $x \in \Sigma'^*$ "guesses" and outputs a string $y = y_1 y_2 \ldots y_n \in \Sigma'^*$ with $y \leq x$ symbolwise, thereby recording the state p of M so far reached. For any "guessed" symbol $y_i \in \Sigma'$, N "guesses" two states q, q' of M, and verifies that there is a transition from q to q' on which the symbol is read, and that there is a path from p to q, on which only symbols in $\Sigma - \Sigma'$ are read. q' then becomes the new state so far reached. N accepts when q' is a final state of M. Since N has "guessed" an arbitrary word smaller than or equal to x, clearly, the maximal valid output of N, $opt_N(<M>, \Sigma', x)$, equals the (lexicographically) greatest word $h(w) \leq x$ such that $w \in L(M)$, where h is the homomorphism, which deletes all symbols in $\Sigma - \Sigma'$.

For the hardness opt-L $\subseteq \mathrm{FL}_m(f_{maxrelDFA})$ let f be an arbitrary function in opt-L. Let N be the NL-transducer such that $opt_N(x) = f(x)$ for all inputs x. We will construct a function h computable with logarithmic space such that $f(x) = f_{maxrelDFA}(h(x))$. h is defined with

$$h(x) := (<N_x>, \{0,1\}, 1^{p(|x|)}),$$

where $p(|x|)$ denotes the polynomial, which bounds the running time of N, $\{0,1\}$ denotes the specified subalphabet, and $<N_x>$ denotes the encoding of a state transition graph of the NFA $N_x := (\ C \cup C',\ \{0,1\} \cup C,\ \delta,\ c_{start},\ c_{acc}\ )$, with C the set of all configurations of N on input x, C' a (distinct) set of all configurations, in which an output 0 or 1 occured, c_{start} the start configuration, and c_{acc} the unique accepting configuration, in which w.l.o.g. no output occurs. $\delta : C \cup C' \cup \{sink\} \times C \cup \{0,1\} \rightarrow C \cup C' \cup \{sink\}$ is defined for all configurations c_i, c_j, such that c_i reaches c_j in one step in a computation of N on x with:

$$\delta(c_i, c_j) = c_j, \quad \text{if no output occurs in } c_j;$$
$$\delta(c_i, c_j) = c'_j, \quad \text{if in } c_j \text{ an output occurs;}$$

and

$$\delta(c'_j, b) = c_j, \quad \text{if in } c_j \text{ occurs the output } b \in \{0,1\}.$$

To make the automaton complete lead all the remaining labels into the state $sink$.

It is easy to see that the automaton is deterministic. Note that for any path of N on input x, there is a word in $L(N_x)$, consisting of the sequence of configurations of the path, each of which is followed by the possible output 0 or 1 of the configuration, (e.g., $c_1 0 c_2 c_3 1 c_4 \ldots c_{p(|x|)}$). But then, clearly, $f(x)$, the maximal output produced by N on input x, equals the maximal word $h(w) \leq 1^{p(|x|)}$ such that $w \in L(N_x)$, where h is the homomorphism that deletes all symbols in C of w. Thus, $f_{maxrelDFA}$ is hard for opt-L, too.

(ii) We have $f_{maxNFA} \in$ opt-L with (i), since f_{maxNFA} is a special case of $f_{maxrelNFA}$.

For the proof of the hardness opt-L $\subseteq \mathrm{FL}_m(f_{maxNFA})$, the proof for the hardness in Theorem 3.1.(ii) carries over. Let $f \in$ opt-L be arbitrarily chosen and let N be the NL-transducer with time bound $p(|x|)$ such that $f = opt_N$. For given N and $x \in \{0,1\}^*$ construct N_x exactly as for the proof of Theorem 3.1.(ii). It it easily verified that $f(x) = f_{maxNFA}(h(x))$ with h as defined there.

(iii) Given an encoding of a DFA M and a string $x \in \{0,1\}^*$, a deterministic log space transducer N computes $f_{maxDFA}(<M>, x)$ with the help of the following two oracles $A, B \in$ NL:

$$A := \{\ <M>\$x\$0^m \mid m \geq 1,\ \exists w \in \{0,1\}^m \text{ such that } w \leq x \text{ and } w \in L(M)\ \};$$
$$B := \{\ <M>\$q\$0^n \mid n \geq 1,\ q \text{ is a state of } M, \text{ such that there exists a path of length } n \text{ from } q \text{ to a final state of } M\ \}.$$

With (maximal $|x|$) queries to A, N can find the length l_{max} of the maximal word smaller than or equal to x accepted by M. Since the given automata is deterministic, with (maximal $2 \cdot l_{max}$) queries to B, N can find longer and longer prefixes of this word, outputting each found bit. □

Note that it will not be easy to improve the upper bound of FL(NL) for f_{maxDFA}, since already $f_{maxDFA} \in \mathrm{FL}_{\log}(\mathrm{NL})$ implies L = NL and thus FL = FL(NL) (see Section 5, the remark after Proposition 5.5.). Hence f_{maxDFA} is clearly an example of a "hardest" function in FL(NL).

Other complete functions for the classes #L, span-L, and opt-L can be obtained by defining "counting" versions of the graph accessibility problem, GAP, which is well-known to be NL-complete (see [Jo 75]). Consider the functional graph problems:

$f_{\#path}$:
Input: a directed graph $G = (V, E)$ with vertex set $V = \{1, 2, \ldots, n\}$
Output: number of different paths of length at most n from vertex 1 to vertex n.

$f_{\#specialpath}$:
Input: a directed labelled graph $G = (V, E)$ with vertex set $V = \{1, 2, \ldots, n\}$ and edges labelled over L, and $L' \subseteq L$
Output: number of lexicographically different paths from vertex 1 to vertex n of length at most n, with the labels in L' deleted.

$f_{maxpath}$:
Input: a directed labelled graph $G = (V, E)$ with vertex set $V = \{1, 2, \ldots, n\}$
Output: lexicographically maximal path from vertex 1 to vertex n of length at most n. (If no such path exists, the output is $\perp$.)

It is not hard to show that these three functions are contained in #L, span-L, and opt-L, respectively. By (many-one) reducing the complete problems of Theorem 3.1.(i),(ii) and Theorem 3.2.(ii) to the corresponding graph functions it can furthermore be shown that these functions are even complete for these classes.

Corollary 3.2.
(i) $f_{\#path}$ is many-one complete for #L;
(ii) $f_{\#specialpath}$ is many-one complete for span-L;
(iii) $f_{maxpath}$ is many-one complete for opt-L. □

4. #L and opt-L are Easy, but span-L is Hard (for the Polynomial Hierarchy)

Since there are only polynomially many different configurations possible for a NL-machine or a NL-transducer, one expects that the definition of counting classes with the help of logarithmic space bounded Turing machines leads to classes contained in

FP. This is indeed the case for the classes #L and opt-L. With the following two theorems we show that both classes are even contained in NC^2.

Recall that $FL \subseteq NC^2 \subseteq FP$, where NC^2 denotes the class of Boolean functions computable by (log space) uniform circuits of polynomial size and $(\log n)^2$ depth with n being the length of the input. These classes are closed under NC^1-reducibility (for a formal definition of these concepts see e.g. [Ru 81] or [Co 85]).

Theorem 4.1. $\#L \subseteq NC^2$.

Proof: Let f be in #L. Then there exists a NL-machine M such that $f(x) = acc_M(x)$. Let $p(|x|)$ be the polynomial time bound of M. Thus there are $q(|x|)$ different configurations of M for a polynomial q with $q > p$.

We reduce the problem of computing the number of accepting computation paths of M on input x to the problem of computing a matrix powering. Let $A = (a_{ij})$ be the $q(|x|) \times q(|x|)$ integer matrix, which represents the adjacency matrix of the computation graph of M on x:

$$a_{ij} := \begin{cases} 1, & \text{if } i = j \text{ or configuration } j \text{ is accessible from configuration } i \\ & \text{in one step in a computation of } M \text{ on input } x; \\ 0, & \text{otherwise.} \end{cases}$$

Then the element $a_{ij}^{p(|x|)}$ of the matrix $A^{p(|x|)}$, $p(|x|)$ times the product of A with itself, contains the number of paths from vertex i to vertex j with maximal length $p(|x|)$, if the matrix product of two $n \times n$ integer matrices $B = (b_{ij})$ and $C = (c_{ij})$ is slightly changed as follows:

$$(b * c)_{ij} := \begin{cases} 1, & \text{if } i = j; \\ \sum_{\substack{i \neq k \\ j \neq k}} a_{ik} * b_{kj} + \frac{1}{2} * (a_{ii} * b_{ij} + a_{ij} * b_{jj}), & \text{otherwise.} \end{cases}$$

Clearly, this modified product is not more difficult to compute than the (normal) integer matrix product, which is computable in NC^1 [Co 85], and computing the matrix powering thus can be done in NC^2. Furthermore, the function h with $h(<M>, x) = (A, s, t, p(|x|))$ that computes the matrix A, the initial and accepting configuration s and t (we assume that there is an unique accepting configuration), and the value $p(|x|)$ is contained in NC^1. Thus we conclude that $\#L \subseteq NC^2$.

Note that it even holds $\#L \subseteq DET \subseteq NC^2$, where DET is the class of functions (NC^1-)reducible to computing integer matrix determinants [Co 85]. In [Co 85] it is proved that the problem of computing integer matrix powering is complete for DET. By starting with the adjacency matrix above with entries 0 in the diagonal and using the normal matrix product and powering, the entry a_{ij}^k of the matrix A^k contains the number of paths from i to j with *exact* length k. Computing the number of all paths with length up to $p(|x|)$ then can be achieved by computing A^k for $1 \leq k \leq p(|x|)$ and summing up all entries a_{ij}^k. The summing up can be done with a NC^1-algorithm. □

For opt-L, inclusion in FP follows, since due to the polynomial bound on the number of configurations of a NL-transducer, the optimal output value can be constructed bit-by-bit by a breadth-first search method. But with a little more effort, we can even show that opt-L is contained in NC^2.

Theorem 4.2. $FL(NL) \subseteq \text{opt-L} \subseteq NC^2$.

Proof: Since NL is closed under complementation [Im 88] [Sze 88], clearly it holds $FL(NL) \subseteq \text{opt-L}$.

For the inclusion $\text{opt-L} \subseteq NC^2$ let f be in opt-L. Then there exists a NL-transducer M such that $f(x) = opt_M(x)$. We will reduce the problem of computing $opt_M(x)$, the maximum output of M on input x, to computing a special matrix powering, the entries of the matrices being words from the set $\{0,1\}^* \cup \{\perp\}$.

Define a special concatenation $*$ for words $w, v \in \{0,1\}^* \cup \{\perp\}$ as follows:

$$w * v := \begin{cases} wv, & \text{if } w, v \in \{0,1\}^*, \\ \perp, & \text{otherwise.} \end{cases}$$

Now let $\perp$ be lexicographically smaller than any other word in $\{0,1\}^* \cup \{\perp\}$. Then the special "matrix product" $A * B = ((a * b)_{ij})$ of two word matrices $A = (a_{ij})$ and $B = (b_{ij})$ of dimension n is defined with the operation $*$ and the maximum with respect to lexicographical ordering MAX as follows.
For all $1 \leq i, j \leq n$ let

$$(a * b)_{ij} := MAX_{k=1}^{n} \{ a_{ik} * b_{kj} \}.$$

Let s, t with $s \neq t$ denote the initial and (unique) final configuration of M, resp., in both of which w.l.o.g. no output occurs. Let $p(|x|)$ denote the time bound of M. Then $A^{p(|x|)} = (a_{ij}^{p(|x|)})$ is the matrix obtained by multiplying the following matrix $A = (a_{ij})$ $p(|x|)$ times with itself. It can be proved by induction that $opt_M(x)$ equals the element $a_{st}^{p(|x|)}$, if $a_{st}^{p(|x|)}$ contains a word in $\{0,1\}^+$ and equals 0 otherwise.

$$a_{ij} := \begin{cases} b, & \text{if } i \xrightarrow{1} j \text{ and in the configuration } j \text{ the output bit is } b \in \{0,1\}; \\ \perp, & \text{otherwise,} \end{cases}$$

where $i \xrightarrow{1} j$ denotes that the configuration j is accessible from configuration i in one step of M in a computation on x.

The matrix A, s, t and $p(|x|)$ can be obtained from M and x with a NC^1-function, and the problem of computing the special matrix powering $A^{p(|x|)}$ is contained in NC^2, since it is not harder to compute than computing matrix powering in the case of integer matrix multiplication, which is contained in NC^2 [Co 85]. □

The relationship between #L and opt-L seems to be difficult to solve. The containment of opt-L in #L implies that nondeterministic computation can be made unambigous in the case of logarithmic space. Let unambNL denote the class of languages acceptable by NL-machines, which in the case of acceptance have an unique accepting computation. Then we have:

Proposition 4.3. FL(NL) $\subseteq$ #L if and only if unambNL = NL.

Proof: Clearly, we have for any c_A, the characteristic function of a language $A \in$ NL, $c_A \in$ FL(NL). But $c_A \in$ #L implies that there exists a NL-transducer, which on input x has exactly one computation path if $x \in A$ and none if not, and thus $A \in$ unambNL.

Conversely, assume that unambNL = NL. Since NL is closed under complementation [Im 88][Sze 88], a L-transducer M with oracle $A \in$ NL computing a function f can be simulated by a NL-transducer N, which uses subroutines for A and CoA to solve the oracle queries: For any oracle query w, the answer is "guessed" and correspondingly either the subroutine for A or CoA is started. Since $|w|$ can be up to a polynomial in the length of x for this the configuration c_w, in which M starts to write w on its oracle tape, is saved, and each symbol of w is produced anew by starting M in c_w. $A, CoA \in$ NL, and by assumption, the subroutines A and CoA compute their solution with an unique accepting path. Thus, clearly, $acc_N(x) = f(x)$ can be simply achieved by guessing the length of $f(x)$ and arranging trivial accepting paths according to the output bits of M. □

With Proposition 4.2. it follows:

Corollary 4.4. If opt-L $\subseteq$ #L, then unambNL = NL. □

The corresponding result for polynomial time functions classes opt-P $\subseteq$ #P if and only if UP = NP was shown in [Kö Schö Tor 89]. Note that the other direction in Corollary 4.4. can be obtained, if one-way classes are considered (see Theorem 4.14.).

The preceding results show that in contrast to opt-L, the class #L cannot be shown easily to contain the class FL(NL). When used as oracle to L-transducers (or L-machines), however, we have parallel to the case of polynomial time, where P(PP) = P(#P) is well-known (see [Ba Bo Schö 86]):

$$\text{L(NL)} \subseteq \text{L(PLP)} \subseteq \text{L(\#L)} \subseteq \text{L(DET)},$$

where PLP denotes probabilistic logarithmic space (with unbounded two-sided error) and polynomial worst case time, and DET is the set of languages reducible to computing integer matrix determinants (see [Bo et al. 89]).

Though #L and opt-L both are contained in FP, surprisingly, span-L turns out to be a hard log space counting class. In the following we will show that if span-L is contained in FP then the polynomial time hierarchy collapses to P. This follows from the fact that the complexity of span-L is closely tied to the complexity of #P. The two classes are logarithmic space metric reducible to each other, i.e., $\text{FL}_1(\#\text{P}) = \text{FL}_1(\text{span-L})$, as shown in the remaining part of this section.

Theorem 4.5. span-L $\subseteq$ #P.

Proof: Let f_M be a function in span-L which given x computes the number of different valid output values of an arbitrarily chosen NL-transducer M. Let the time bound of M be $p(|x|)$ for a polynomial p, i.e., any output value y of M on input x satisfies $y \leq p(|x|)$. Since the set

$$A := \{\, x\$y \mid x, y \in \{0,1\}^*,\ y \text{ is a valid output of } M \text{ on input } x \,\},$$

is in NL, there exists a P-algorithm for deciding A. Construct an NP-machine N which given x "guesses" an y of length less or equal than $p(|x|)$ and then verifies that $x\$y \in A$ by executing the P-algorithm for A on $x\$y$. Then N has exactly one accepting computation for each y which is an output value of M on input x, i.e. N has exactly $f_M(x)$ accepting computations. Thus, $f_M \in$ #P. □

To show that, conversely, #P is log space metric reducible to span-L, we first prove a lemma. Consider the set of pairs of Boolean formulas and assignments, where the formulas are in conjunctive normal form with at most three variables per clause (3-CNF):

$$\text{EVAL3-SAT} := \{\, F\,\$b_1 b_2 \ldots b_n \mid F \text{ is a Boolean formula in 3-CNF with } n \text{ variables and } b_i \in \{0,1\} \text{ is a satisfying assignment for } F \,\}.$$

As shown by Lynch [Ly 77], EVAL3-SAT is contained in L (see also [Bu 87]). It is well known that, consequently, NP = NL(L) where NL(·) denotes the closure under logarithmic space nondeterministic Turing reducibility defined by Ladner and Lynch [La Ly 76] (see [Ru Si To 84]).

But, NP can be characterized even as NLOG(L), where NLOG(·) denotes the closure under logarithmic space *nondeterministic many-one* reducibility (see [Lan 86]), i.e., the set of all languages A for which there exist a language $B \in$ L and a NL-machine N with output tape such that for all x it holds:

$$x \in A \iff \text{one of the computation paths of } N \text{ on } x \text{ produces a word } y \in B.$$

The following lemma shows that EVAL3-SAT is also contained in Co1NL. Consequently, NP equals NLOG(Co1NL), since one easily verifies that

$$\text{3-SAT} \in \text{NLOG(EVAL3-SAT)}.$$

For this, given a Boolean formula, a NLOG-transducer simply copies the formula onto its output tape, writes the separation symbol \$ and "guesses" and writes bitwise an assignment.

Note that in contrast to NP = NLOG(Co1NL), the class NLOG(1NL) equals NL: Given a NLOG-transducer T and a 1NL-machine M construct a NL-machine that takes turns in simulating T and M. First, T is simulated as long as the first output

occurs; then M is simulated on this output as long as it asks for the following input bit (M reads the input one-way), which is the following output bit of T and can be obtained by simulating T further, and so on. For taking turns in the simulation, only the current configuration of either T or M has to be stored. Thus NLOG(1NL) $\subseteq$ NL. The inclusion in the other direction is trivial.

Lemma 4.6. EVAL3-SAT $\in$ Co1NL.

Proof: A Co1NL-machine can be understood as a one-way nondeterministic logarithmic space machine with universal states. Thus given a formula F in 3-CNF and an assignment

$$(x_{11}, x_{12}, x_{13}), \ldots, (x_{m1}, x_{m2}, x_{m3})\$b_1 b_2 \ldots b_n,$$

a clause X_i can be "guessed" universally, the head of the input tape can be moved to the guessed clause and all of its three literals x_{i1}, x_{i2}, and x_{i3} can be stored on the working tape. By moving the head to the assignment the three relevant bits can be picked up and it can be checked whether the clause is satisfied. The well-formedness of the input (a well-formed formula in 3-CNF plus appropriate assignment) can be checked by an different universal tree from the start configuration.

(The reader who prefers to "guess" existentially easily verifies conversely, that the complement of EVAL3-SAT is contained in 1NL: a clause can be "guessed" and verified that it is not satisfied by the assignment or it can be "guessed" and verified that the input is not well-formed. Clearly, in both cases the input has to be read only once from left to right for the verification. In the former case first the (maximal 3) variables of the guessed clause will have to be picked up and stored and then by moving the input head further to the right the corresponding three variable assignments must be made out to accept in the case that all three make its literal have value 0.) □

Note the similarity with the proof that has been used in [Je Ki 89] to show that polynomial time and logarithmic space bounded auxiliary pushdown automata with one alternation accept 3-SAT by first guessing an assignment onto the pushdown store and then *universally* checking that it was correct.

The idea of NP = NLOG(EVAL3-SAT) yields that the function that computes for a Boolean formula F in 3-CNF the number of non-satisfying assignments of F is contained in span-L. As shown by Valiant [Va 79b], the "complement" of this function, the function which counts the number of satisfying assignments of F:

$f_{\#sat}$:

Input: a Boolean formula F in 3-CNF
Output: number of satisfying assignments of F,

is an example of a function complete for #P even with respect to logarithmic space (functional) many-one reducibility. Thus we get:

Theorem 4.7. $\#P \subseteq FL_1(\text{span-L})$.

Proof: Since $f_{\#sat}$ is log space many-one complete for #P, it suffices to show $f_{\#sat} \in FL_1(\text{span-L})$.

Construct a NL-machine M such that given a well-formed formula F in 3-CNF with n variables as input to M, $span_M(F)$ equals the number of its *non-satisfying* assignments. Call this function $f_{\#unsat}$. Using $f_{\#unsat}$ as oracle $f_{\#sat}(F)$ can be computed by a deterministic logarithmic space transducer M' as follows. Given an arbritary input F, M' first checks that F is a well-formed Boolean formula in 3-CNF. If this is not the case, M' outputs 0. If F is well-formed, M' copies F onto the oracle query tape and queries its oracle $f_{\#unsat}$ once. Then it just subtracts $f_{\#unsat}(F)$, the content of the oracle answer tape, from 2^n. Since n can be computed before reading the oracle answer tape and 2^n can be stored with logarithmic space (by storing n), this subtraction can be done with an log space machine.

The construction of M can be done as follows. M guesses an assignment for all the n variables of F —symbol by symbol— and verifies that it does *not* satisfy F by simulating the 1NL-machine for the complement of EVAL3-SAT, which exists by Lemma 4.6. Every symbol (0 or 1) guessed will be written by M on the output tape. Thus the number of different valid outputs of M corresponds to the number of different unsatisfying assignments for F, i.e. the number of different possible assignments, 2^n, minus the number of satisfying assignments. □

Corollary 4.8. Every function in #P is logarithmic space metric reducible to a function in span-L and vice versa, i.e. $FL_1(\text{span-L}) = FL_1(\#P)$. □

Thus, span-L and #P are classes very similar in computation power, when used as oracles. More precisely, we could say that any function in #P can be computed by the subtraction of two functions in span-L. This parallels the statement in [Kö Schö Tor 89] that any function in #NP can be computed by subtracting two functions in span-P.

But although span-L and #P are very similar, they do not seem to be the same function class. Since $f_{\#sat}$ is complete for #P, the inclusion $\#P \subseteq$ span-L implies that there exist a NL-transducer M such that $span_M(F)$ equals the number of different *satisfying* assignments of a Boolean formula F. Then, clearly, by simulating this machine and checking that it has a valid output on input F, a NL-machine could accept 3-SAT. Thus, it holds:

Proposition 4.9. If span-L = #P, then NL = P = NP. □

Since $P(NP) \subseteq P(\#P)$, Corollary 4.8. furthermore implies that span-L is Turing hard for $\Delta_2^p = P(NP)$, the second deterministic level of the polynomial time hierarchy, and thus span-L $\subseteq$ FP implies P = NP. But the implications of this inclusion are even stronger. As recently shown by Toda [To 89], the class PP (see [Gi 77]) is hard for the polynomial time hierarchy (PH) with respect to Turing reducibility. Since P(#P) = P(PP) (see [Ba Bo Schö 86]), this class can now be also characterized by

span-L. Furthermore, results obtained by Toda and Watanabe [To Wa 89] imply P(#P) = P(span-P), and P(span-P) = P(#NP). Thus we have:

Corollary 4.10. PH $\subseteq$ P(span-L) = P(#P) = P(span-P) = P(#NP) = P(PP).

Corollary 4.11. span-L $\subseteq$ FP if and only if P = NP = PH = P(#P).

Corollary 4.8. has shown that span-L and #P share the same complete languages with respect to metric reducibility $FL_1(\cdot)$. Thus, with Theorem 3.1.(ii) and 3.2.(ii), we can add to the list of functions complete for #P the two functions $f_{\#NFA}$ and $f_{\#specialpath}$. Note that these functions are not mere counting versions of NP-complete problems, like, e.g., $f_{\#sat}$ and most of the so far known #P-complete functions, but are rather counting versions of NL-complete problems.

Corollary 4.12. $f_{\#NFA}$ and $f_{\#specialpath}$ are complete for #P with respect to metric reducibility. □

Consequently, we know that computing the census or ranking functions of regular languages or, e.g., languages in 1NL can be done in polynomial time if and only if FP = #P. Furthermore, Corollary 4.12. implies that computing the ranking or census function for languages in 1NL (and various other classes) is even hard for #P or span-L with respect to metric reducibility.

In fact, we can say even more about the relationship between span-L and #P in terms of automata problems. As shown in Section 3 computing the ranking function or the census function for a NFA is logarithmic space *many-one* complete for span-L. The "complement" of the census function

$f_{\#co-NFA}$

Input: an encoding of a NFA M and 1^n
Output: number of words of length n that are not accepted by M

is logarithmic space *many-one* complete for #P.

To see this, we refer to the proof given in [Ha Hu 74] for the result that non-universality of regular expressions is complete for PSPACE. There it was shown how the non-accepting computations of a nondeterministic polynomial space bounded Turing machine on input x can be described by a regular expression obtainable with logarithmic space in $|x|$. Since regular expressions can be easily transformed to NFA this holds for NFA also. Thus we can construct for a nondeterministic $p(|x|)$ space bounded Turing machine N with input x a NFA M such that for a suitably chosen polynomial q any word

$$w \notin L(M) \text{ with } |w| \leq 2^{q(|x|)}$$

codifies an accepting computation of N on input x. Therefore, the function which, given a NFA and n (in binary), computes the number of words of length n that are *not* accepted by the NFA is (many-one) hard for #PSPACE (compare [La 89]). (It is

even complete.) This function is a coding variant of $f_{\#co-NFA}$. It is easily seen that in the case of polynomial time bounded machines the length of the computations can be described in unary. Thus $f_{\#co-NFA}$ is hard for #P. Since the word problem for NFA is decidable in NL, it is easy to see that $f_{\#co-NFA}$ is contained in #P.

Note that the classes #P and #PSPACE can trivially be separated, since functions f in #PSPACE = FPSPACE can have values $f(x)$ of length up to $2^{(p(|x|))}$, whereas a value of a function in #P (or FP(#P)) is always polynomially bounded in $|x|$ (see [La 89], where investigations of #PSPACE and restrictions of this class can be found).

An overview of the complete automata functions for #L, span-L, and #P is given in Figure 3, where "complete" means (functional) logarithmic space many-one complete.

input $<<M>, 1^n>$, compute	DFA	NFA
non-emptiness $L(M) \neq \emptyset$?	NL-complete [Jo 75]	NL-complete [Jo 75]
non-universality $L(M) \neq \Sigma^*$?	NL-complete [Jo 75]	PSPACE-complete [Ha Hu 74]
number of members $\| L(M)^{\leq n} \|$	#L-complete	span-L-complete
number of non-members $\| CoL(M)^{\leq n} \|$	#L-complete	#P-complete

Figure 3: Overview about the Complexity of some Automata Problems and Functions

The last two lines of the table reflect a second time that any function in #P can be computed by a subtraction of a function in FP (or FL) and a function in span-L (compare the proof of Theorem 4.7.). (This time we have an example where the function in span-L even has been shown to be (many-one) complete for this class.) This parallels the relationship between #NP and span-P (see [Kö Schö Tor 89]), where any function in #NP can be computed by a subtraction of a function in FP and a function in span-P. In fact, we could just substitute "L" for "P" in this statement, and say: any function in #NL can be computed by a subtraction of a function in FL and a function in span-L, since with the considerations after Theorem 4.5. it is not hard to show that

$$\#\mathrm{NL} = \#\mathrm{P}, \qquad (*)$$

when #NL denotes the class of functions that witness the number of accepting computations of NL-machines with oracles in NL, where the oracle tape is attached as defined by Ladner and Lynch [La Ly 76].

Corollary 4.11. shows that the inclusion of span-L in any of the two subpolynomial counting classes #L or opt-L is unlikely.

Corollary 4.13. If either span-L $\subseteq$ #L or span-L $\subseteq$ opt-L, then P = NP = P(#P) = P(span-L). □

We obtain two further time-space downward separations by considering the relationships between the one-way classes 1NL, unamb1NL, #1L, opt-1L, and span-1L. (These classes are defined by restricting the underlying NL-machines or NL-transducers to read their input one-way.) Note that opt-1L $\cup$ #1L $\subseteq$ span-1L.

Since the difference between counting classes and span classes exactly corresponds to the difference of unambiguous to ambiguous computation, the proof of NP $\subseteq$ UP $\iff$ #P = span-P given in [Kö Schö Tor 89] can be translated to logarithmic space counting classes, if one-way machines are considered. In the case of 2-way classes the proof technique cannot be used to show the implication from right to left.

Theorem 4.14. The following propositions are equivalent:
(i) unamb1NL = 1NL;
(ii) span-1L = #1L;
(iii) opt-1L $\subseteq$ #1L.

Proof: (i) $\Rightarrow$ (ii) Assume 1NL $\subseteq$ unamb1NL and let $f = span_M$, where M is an 1NL-machine. Consider the set

$$L_M = \{\, x_1\#y_1\$\ldots\$x_n\#y_n\$ \mid x_i \in \{0,1\},\ y_i \in \{0,1\}^*, \text{ and on input } x_1x_2\ldots x_n \text{ there is a computation path on which } M \text{ outputs } y_1\ldots y_n, \text{ such that output } y_i \text{ occurs after reading bit } x_i \text{ and before reading input bit } x_{i+1} \,\}.$$

As M is a one-way machine, it can be shown that $L_M \in$ 1NL, and hence $L_M \in$ unamb1NL by assumption. Let M' be the 1NL-machine for L_M that if it accepts an input, it has an unique accepting computation. Now, construct a 1NL-machine M'', which fulfills $f(x) = acc_{M''}(x)$. M'', on input x does the following bitwise: M'' "guesses" $x_1\#y_1\$x_2\#y_2\$\ldots x_n\#y_n\$$ bitwise, simulates M' on this word, thereby checking that $x = x_1\ldots x_n$. The existence of M'' implies $f \in$ #1L.

(ii) $\Rightarrow$ (iii) obvious.

(iii) $\Rightarrow$ (i) Assume opt-1L $\subseteq$ #1L and let $A \in$ 1NL. Then there is a 1NL-machine M such that $A = L(M)$. Define M' such that it outputs "1" if M accepts and nothing otherwise. Then $opt_{M'}$ is the characteristic function of A which by assumption is in #1L. That is, there is a 1NL-machine which has one accepting computation on inputs $x \in L$, and none on $x \notin L$. Thus $L \in$ unamb1NL. □

Note that the one-way restrictions of all the classes contain complete problems or functions for the 2-way classes, and therefore span-L = FL_m(span-1L), #L = FL_m(#1L), and opt-L = FL_m(opt-1L). Thus Theorem 4.14. together with Corollary 4.11. yields:

Corollary 4.15. The following statements imply P = NP = P(#P) = P(span-L):
(i) unambl1NL = 1NL;
(ii) span-1L = #1L;
(iii) opt-1L $\subseteq$ #1L. □

5. Some Restrictions

In this section we consider some interesting restrictions of the classes opt-L and span-L. In [Kö Schö Tor 89] the name of the class span-P refers back to the possible different outputs which occur at the *leaves* of computation trees of NP-transducers, i.e., in the last configuration of a computation path, rather than distributed on the path. Since in the case of polynomial time the whole output can be stored on the work tapes, there is clearly room for alternative definitions. In the case of logarithmic space however, the output of the NL-transducer may be longer than logarithmic (in the length of the input), and thus cannot be stored on the work tape. Here, a definition specifying that the output occurs at the leaves of the computation tree leads to *two* interesting restrictions of the classes defined by logarithmic space bounded transducers. We will show that both restrictions lead to alternative characterizations of the following two classes: $\mathrm{FL}_{\log}$(NL), the class of functions f, computable by deterministic logarithmic space transducers with oracle in NL, which make only $O(\log|x|)$ many oracle queries on any input x, and its subclass $\mathrm{FL}_{\log}(\mathrm{NL})[\log n]$, the class of functions $f \in \mathrm{FL}_{\log}(\mathrm{NL})$, such that $|f(x)| \leq c \log|x|$ for a constant c.

The corresponding function classes without restriction on the number of oracle queries are FL(NL) and, resp., FL(NL)[log n].

The first restriction is obtained as follows:

Definition 5.1. By opt-L{log n} and span-L{log n} we denote the classes of functions that are defined as the classes opt-L or, resp., span-L (see Definition 2.2. and 2.3.), with the restriction that all the outputs of the underlying NL-transducers on input x are bounded in length by $c \log|x|$ for a constant c.

This restriction is strong enough that an output produced on a computation path of a NL-transducer can be completely reconstructed by a L-transducer with oracle in NL. The following proposition shows that both opt-L{log n} and span-L{log n} coincide with FL(NL)[log n].

The second restriction concerns the output *mode* rather than the output *size* of the NL-transducers. We restrict the transducer in a Ruzzo-Simon-Tompa fashion to make only *deterministic* use of the extra, the output tape (compare [Ru Si To 84], where such a restriction was considered for the use of the oracle tape). The restriction

demands that the printing of the whole output has to be done deterministically, i.e., that there is no further nondeterministic choice possible after the first output occurs on a path.

Definition 5.2. Define opt-L{*determ*} and span-L{*determ*} as span-L or, resp., opt-L, with the restriction that all the outputs of the underlying NL-transducers on input x are produced deterministically.

Clearly, printing deterministically means that what is printed along one path is completely determined by the first configuration on the path in which an output occurred. Since there are only polynomially different configurations, the number of different valid outputs therefore is bounded by a polynomial, too. Not surprising then is the observation that the so specified function class span-L{*determ*} coincides with the class span-L{$\log n$}. As furthermore noted, the functions in FL(NL)[$\log n$] are already computable with a logarithmic number of questions to an oracle in NL, i.e., are contained in the class $\mathrm{FL}_{\log}(\mathrm{NL})[\log n]$. Clearly, we need only one oracle question for each output bit.

Proposition 5.3. The following are different characterizations of the same class:
(i) opt-L{$\log n$},
(ii) span-L{$\log n$},
(iii) span-L{*determ*},
(iv) FL(NL)[$\log n$],
(v) $\mathrm{FL}_{\log}(\mathrm{NL})[\log n]$.

Proof: (i) $\Rightarrow$ (ii) For opt-L{$\log n$} $\subseteq$ span-L{$\log n$} let $f \in$ opt-L{$\log n$}. Then $f = opt_M$ such that M is a NL-transducer all of which outputs y on input x satisfy $|y| \leq c \log |x|$ for a constant c. Define a NL-transducer M' which computes $opt_M(x)$, the maximal output of M on input x, by cycling through all possible output values of M, i.e., all strings v with $|v| \leq c \log |x|$. Since the language

$$L_M = \{\, x\$v \mid v \text{ with } |v| \leq c \log |x| \text{ is a valid output of } M \text{ on input } x \,\}$$

is contained in NL, and NL is closed under complementation [Im 88] [Sze 88], this can be done by M'. After computing $opt_M(x)$, M' simply "guesses" a word v such that $v \leq opt_M(x)$, and outputs v. Then $span_{M'}(x) = opt_M(x)$.

(ii) $\Rightarrow$ (iii) The inclusion span-L{$\log n$} $\subseteq$ span-L{*determ*} is obvious, since the output on any path can be stored on the work tape.

(iii) $\Rightarrow$ (iv) For span-L{*determ*} $\subseteq$ FL(NL)[$\log n$] let $f \in$ span-L{*determ*}. Let M be the NL transducer which witnesses f. There are maximally polynomial many different configurations in which M starts to write (deterministically) its output. Some of these configurations correspond to a certain valid output value. Define

$$\begin{aligned} L_M := \{ x\$c \mid\ & c \text{ is a configuration, in which } M \text{ starts to write a valid output } w_c, \\ & \text{and for all configurations } c' \text{ with } c' > c, \text{ in which } M \text{ starts to write} \\ & \text{a valid output } w_{c'}, \text{ it holds } w_{c'} \neq w_c \,\}. \end{aligned}$$

Since $span_M(x)$, the number of different output values, equals the number of different c which satisfy $x\$c \in L_M$, a L-transducer with oracle L_M can compute this value on input x by cycling through all the possible configurations c of M in, e.g., lexicographical order. $L_M \in$ NL holds, since NL is closed under complementation: $w_{c'} \neq w_c$ is a L-property, and the conditions specified for L_M yield one alternation (with logarithmic space).

(iv) $\Rightarrow$ (v) For the inclusion $\mathrm{FL(NL)}[\log n] \subseteq \mathrm{FL_{log}(NL)}[\log n]$, let $f \in \mathrm{FL(NL)}[\log n]$. Then there is a NL-transducer M with oracle $A \in$ NL which outputs on input x, $f(x) = y$ with $|y| \leq c \log |x|$ for a constant c. Define

$$L_{(M,A)} := \{x\$i \mid \text{the } i\text{th output bit of } M \text{ with oracle } A \text{ on input } x \text{ is } 1\}.$$

Since NL is closed under complementation, it is easily seen that $L_{(M,A)}$ is contained in NL. By using $L_{(M,A)}$ as oracle then each of the $c \log |x|$ output bits of M can be found by using only a logarithmic number of questions.

(v) $\Rightarrow$ (i) The remaining inclusion $\mathrm{FL_{log}(NL)}[\log n] \subseteq$ opt-L$\{\log n\}$ is obvious. □

Surprisingly, as shown in the following proposition, opt-L$\{determ\}$, too, has a characterization in terms of deterministic transducers with oracle in NL. Clearly, the output of functions f in opt-L$\{determ\}$ no longer is logarithmically bounded. But, and this seems to be critical for the proof, it can be "compressed" to a string of length logarithmic: a configuration in which the NL-transducer M with $opt_M(x) = f(x)$ starts to write its optimal output.

Proposition 5.4. opt-L$\{determ\} = \mathrm{FL_{log}(NL)}$.

Proof: For the inclusion from left to right let $f \in$ opt-L$\{determ\}$. Let M be the NL-transducer such that $f(x) = opt_M(x)$. On input x there are maximally polynomial many different configurations of M, in which M starts to write (deterministically) a valid output. All configurations have length $c \log |x|$ for a constant c. Thus a deterministic transducer M', which first constructs a configuration of M on its work tape, in which M starts to write deterministically (!) its maximal valid output, and then simulates M from this configuration, computes f. M' can find such a configuration with $O(\log |x|)$ questions to the following oracle:

$$\begin{aligned} A := \{x\$z \mid\ & z \text{ is a prefix of a configuration of } M \text{ of length } c \log |x|, \\ & \text{in which } M \text{ starts to write the maximal of its valid outputs} \\ & \text{on input } x\}. \end{aligned}$$

We claim that $A \in$ NL. Construct a NL-machine N as follows: Given a word $x\$z$, N "guesses" a string v of length maximal $c \log |x|$ such that zv codifies a configuration of M, in which M starts to write the valid output w_{zv}. N then verifies that for all configurations c of M, in which M starts to write a valid output w_c, it holds $w_c \leq w_{zv}$. Clearly, by cycling through all the possible configurations c of M in,

e.g., lexicographical order, and simulating M from the start, N can verify that c is a configuration, in which M starts to write a valid output. Furthermore, comparing the output M produces started in c with the output started in zv can be achieved by simulating M simultaneously started in c and started in zv. M accepts, if for all c the output produced in c is smaller than or equal to the output produced in zv. Since NL is closed under complementation, $A \in \mathrm{NL}$.

For the inclusion $\mathrm{FL}_{\log}(\mathrm{NL}) \subseteq \text{opt-L}\{determ\}$ let M be the deterministic transducer with oracle $A \in \mathrm{NL}$ computing $f \in \mathrm{FL}_{\log}(\mathrm{NL})$. Then the sequence of oracle answers of M on input x is a string $s_x \in \{0,1\}^*$ such that $s_x \leq c \log |x|$ for a constant c. But, given x, s_x can be precomputed by a NL-transducer M', which simulates M on input x ignoring any output, thereby using two subroutines for A and CoA to compute and save the sequence of oracle answers s_x. With s_x then, M' can simulate M completely deterministically to produce $f(x)$. □

Note that

$$\text{opt-L}\{determ\} = \mathrm{FL}_{\log}(\mathrm{NL}) = \mathrm{FL}_1(\mathrm{FL}_{\log}(\mathrm{NL})[\log n]) = \mathrm{FL}_1(\text{opt-L}\{\log n\}).$$

Thus the classes opt-L$\{\log n\}$ and opt-L$\{determ\}$ behave similarily, and the former class can be considered as the log space analogon of the class opt-P$\{\log n\}$ investigated in [Kre 86]. In this article, it was shown that $\mathrm{FP}_{\log}(\mathrm{NP}) = \mathrm{FP}_1(\text{opt-P}\{\log n\})$.

The unrestricted class, opt-L, however, seems to be more powerful than its restriction to deterministic writing. With Theorem 4.2. and Proposition 5.4. we have:

$$\mathrm{FL} \subseteq \text{opt-L}\{determ\} = \mathrm{FL}_{\log}(\mathrm{NL}) \subseteq \mathrm{FL}(\mathrm{NL}) \subseteq \text{opt-L} \subseteq \mathrm{NC}^2.$$

But already the inclusion $\mathrm{FL}(\mathrm{NL}) \subseteq \mathrm{FL}_{\log}(\mathrm{NL})$ has strong implications, since Krentel's idea of the proof for the statement $\mathrm{FP}_{\log}(\mathrm{NP}) = \mathrm{FP}(\mathrm{NP})$ if and only if $\mathrm{P} = \mathrm{NP}$ [Kre 86], can be translated to logarithmic space classes.

Proposition 5.5. $\mathrm{FL}_{\log}(\mathrm{NL}) = \mathrm{FL}(\mathrm{NL})$ if and only if $\mathrm{L} = \mathrm{NL}$.

Proof: The implication from right to left is obvious. For the implication from left to right, recall that the graph accessibility problem, GAP, is complete for NL [Jo 75] [Jo Li La 76]. Clearly, a deterministic transducer M with oracle GAP can be constructed, which produces, given as input an instance of GAP, i.e., a graph and two specified nodes s and t, a path from s to t in the graph, if a path exists, and 0 otherwise. Call the function computed by M f. By assumption, we have: $f \in \mathrm{FL}_{\log}(\mathrm{NL})$. Let M' be the transducer that computes on input x $f(x)$ with the sequence of oracle answers s_x with $s_x \leq c \log |x|$ for a constant c. Construct a L-machine N that recognizes GAP as follows: N cycles through all binary strings $v \leq c \log |x|$. For each string v simulates M' interpreting v as the sequence of answers of the oracle M' and checks whether the output of M' corresponds to a path from s to t in the graph. Clearly, if N finds a correct path, then $x \in \mathrm{GAP}$, and if there exist such a path, N will find it with the help of the correct oracle answer sequence s_x of M'. □

Remark: With a similar demonstration, even the result

$$f_{maxDFA} \in \mathrm{FL}_{\log}(\mathrm{NL}) \quad \text{if and only if} \quad \mathrm{L} = \mathrm{NL}$$

can be obtained. For this note that $f_{maxDFA} \in \mathrm{FL}_{\log}(\mathrm{NL})$ implies that the NL-complete non-emptiness problem for nondeterministic finite automata could be solved deterministically. Hence f_{maxDFA} is an example of a "hardest" function in FL(NL) (see also Section 3).

With the preceding proposition we obtain the surprising implication:

Corollary 5.6. If $\text{opt-L} \subseteq \text{opt-L}\{determ\}$, then $\mathrm{L} = \mathrm{NL}$. □

Since it is not at all clear whether the result $\text{opt-L} \subseteq \mathrm{NC}^2$ can be improved to $\text{opt-L} \subseteq \mathrm{FL(NL)}$, the implication from right to left remains open.

Besides the restrictions of the classes opt-L and span-L obtainable via restricting the *power of the NL-transducers*, other kinds of restrictions can be achieved by restricting the *size of the values of the functions* in opt-L and span-L to a logarithm. Define opt-L[log n] (resp., span-L[log n]) as the class of functions $f \in$ opt-L (resp., span-L), for which exists a constant c such that for every x in $\{0,1\}^*$, $|f(x)| \leq c \log |x|$.

Such a restriction has been considered in [Kre 86] for opt-P and in [Kö Schö Tor 89] for span-P, too. In the latter article it was shown that although the inclusion span-P $\subseteq$ opt-P in the unrestricted case is unlikely (because it implies NP = CoNP) the two classes coincide in the restricted case: span-P[log n] = opt-P[log n].

In Section 4 we have shown that in the case of logarithmic space counting classes, too, the inclusion span-L $\subseteq$ opt-L, is very unlikely, since it implies P = NP (see Corollary 4.13.). As far as the [log n] restriction of these classes is concerned, clearly, in the case of opt-L this restriction coincides with the {log n} restriction. Hence with Proposition 5.3. we know opt-L[log n] = FL(NL)[log n] $\subseteq$ span-L[log n].

A result like span-L[log n] = opt-L[log n] = FL(NL)[log n], however, corresponding to the result in the case of polynomial time classes, seems unlikely. Even though the restriction [log n] ensures that there are only polynomially different output values, these values do not seem to be "compressible", and thus a log space transducer has not enough space to store the information of even one complete output value. But we have:

$$\text{span-L}[\log n] \subseteq \mathrm{FL}(\text{opt-L}), \qquad (*)$$

where FL(opt-L) denotes the class of functions computable by deterministic log space transducers with functional oracle in opt-L. With $(*)$ we know, e.g., that $optL \subseteq$ FL(NL) is a sufficient condition for span-L[log n] $\subseteq$ opt-L[log n]. Furthermore, with $\text{opt-L} \subseteq \mathrm{NC}^2$ (Theorem 4.2.) we could conclude the following theorem, which we prove directly.

Theorem 5.7. span-L[$\log n$] $\subseteq$ NC^2.

Proof: Let M be a NL-transducer witnessing a function in span-L[$\log n$]. W.l.o.g. M has an unique final accepting state in which no output occurs, and each computation path of M on input x is exactly $p(|x|)$ steps long for a suitable polynomial p, and for all inputs x, the number of output values is polynomially bounded.

We can use the technique of computing the output values via iterated matrix multiplication by defining a special matrix product, if we do some precomputation. Clearly, in general there are exponentially many output values possible between two arbitrary configurations c and c' of M. Consequently, the output values cannot be constructed non-adaptively, starting with the full adjacency matrix of the configurations as, e.g., in the proofs of Theorem 4.1. or 4.2. But a simple observation shows that there are only exponentially many output values possible between two arbitrary configurations c and c' of M, if either c or c' does not reach the final configuration or is not reachable from the start configuration. On the other hand, any configuration on a computation path on which a valid output is computed, clearly is reachable from the start configuration and reaches a final configuration.

We start with the following matrix $A = (a_{ij})$ of configurations of M on input x, where t is the (unique) final accepting configuration, and s the start configuration:

$$a_{ij} = \begin{cases} \{b\}, & \text{if } s \xrightarrow{*} i \xrightarrow{1} j \xrightarrow{*} t, \\ & \text{and in configuration } j, M \text{ outputs } b \in \{0,1\}; \\ \{\lambda\}, & \text{if } i = j \text{ or } s \xrightarrow{*} i \xrightarrow{1} j \xrightarrow{*} t, \\ & \text{and } M \text{ makes no output in configuration } j \\ \bot, & \text{otherwise;} \end{cases}$$

where $i \xrightarrow{1} j$ (resp., $i \xrightarrow{*} j$) means that the configuration is accessible from configuration i in one step (resp., in an arbitrary number of steps) of M in a computation on x. The matrix A can be obtained with an NC^2-algorithm.

After computing the following "special matrix product" of A $p(|x|)$ times with itself, the entry $a_{st}^{p(|x|)}$ of the resulting matrix will contain all the output values ($\neq \lambda$) of M on input x. To compute $span_M(x)$ these values have only to be summed up.

Denote the complex product of two sets S and S' by $S \cdot S'$. Define:

$$S * S' := \begin{cases} S \cdot S', & \text{if } S, S' \subseteq \{0,1\}^*, \\ \{\bot\}, & \text{otherwise.} \end{cases}$$

Extend this definition to the special complex product $B*C$ of $n \times n$ matrices $B = (B_{ij})$ and $C = (C_{ij})$ which entries are subsets of $\{0,1,\bot\}^*$ as follows:

$$(B * C)_{ij} := \bigcup_{k=1}^{n} B_{ik} * C_{kj}.$$

It is not hard to show that this "special matrix complex product" can be computed in NC^1, like the product of two integer matrices. Thus, computing $A^{p(|x|)}$ can be done in NC^2. □

Acknowledgment

The authors would like to thank José Balcázar and Jacobo Torán for reading various earlier versions of this article, for many helpful comments, and above all, for their encouragement to write this article.

References

[Ba Bo Schö 86] J.L. Balcázar, R.V. Book, U. Schöning: The polynomial-time hierarchy and sparse oracles, Journ. ACM 33 (1986), 603–617.

[Be Go Sa 87] A. Bertoni, M. Goldwurm, N. Sabadini: Computing the Counting function of context-free languages, Proc. 4th STACS, 1987, LNCS 247, 169–179.

[Bo et al. 89] A. Borodin, S.A. Cook, P.W. Dymond, W.L. Ruzzo, M. Tompa: Two applications of inductive counting for complementation problems, SIAM Journ. of Comput. 18,3 (1989), 559–578; and Erratum (to this article), SIAM Journ. of Comput. 18,6 (1989), 1283.

[Bu 87] S. Buss: The Boolean formula value problem is in ALOGTIME, Proc. 19th STOC, 1987, 123–131.

[Co 85] S.A. Cook: A taxonomy of problems with fast parallel algorithms, Information and Computation 64 (1985), 2–22.

[Gi 77] J. Gill: Computational complexity of probabilistic Turing machines, SIAM Journ. of Comput. 6 (1977), 675–695.

[Go Si 85] A.V. Goldberg, M. Sipser: Compression and ranking, Proc. 17th STOC, 1985, 59–68.

[Ha Hu 74] J. Hartmanis, H.B. Hunt: The LBA problem and its importance in the theory of computing, in: R.M. Karp (ed.), Complexity of Computation, SIAM-AMS Proceedings, vol. 7, 1974, 1–26.

[Ha Ma 81 J. Hartmanis, S. Mahaney: Languages simultaneously complete for one-way and two-way automata, SIAM Journ. of Comput. 10 (1981), 383–390.

[He 87] L. Hemachandra: On ranking, Proc. 2nd Structure in Complexity Theory Conf., 1987, 103–117.

[Hu 88] D.T. Huynh: The complexity of ranking, Proc. 3rd Structure in Complexity Theory Conf., 1988, 204–212.

[Im 88] N. Immerman: Nondeterministic space is closed under complement, SIAM Journ. of Comput. 17,5 (1988), 935–938.

[Je Ki 89] B. Jenner, B. Kirsig: Characterizing the polynomial hierarchy by alternating auxiliary pushdown automata (extended version), Informatique théorique et Applications 23 (1989), 87–99.

[Jo 75] N.D. Jones: Space-bounded reducibility among combinatorial problems, Journ. of Comput. System Sci. 1975, 68–85.

[Jo Li La 76] N.D. Jones, E. Lien, W.T. Laaser: New problems complete for nondeterministic log space, Math. Systems Theory 10 (1976), 1–17.

[Kö 89] J. Köbler: Strukturelle Komplexität von Anzahlproblemen, Doctoral dissertation (in German), University Stuttgart, 1989.

[Kö Schö Tor 89] J. Köbler, U. Schöning, J. Torán: On counting and approximation, Acta Informatica 26 (1989), 363–379.

[Kre 86] M.W. Krentel: The complexity of optimization problems, Proc. 18th STOC, 1986, 69–76.

[La 89] R. Ladner: Polynomial space counting problems, SIAM Journ. of Comput. 18,6 (1989), 1087–1097.

[La Ly 76] R. Ladner, N. Lynch: Relativization of questions about log space computability, Math. Systems Theory 10 (1976), 19–32.

[Lan 86] K.J. Lange: Decompositions of nondeterministic reductions, Proc. 13th ICALP, 1986, LNCS 226, 206–213.

[Ly 77] N. Lynch: Log space recognition and translation of parenthesis languages, Journ. of the ACM 24 (1977), 583–590.

[Ru 81] W. Ruzzo: On uniform circuit complexity, Journ. of Computer and System Sciences 22 (1981), 365–383.

[Ru Si To 84] W. Ruzzo, J. Simon, M. Tompa: Space-bounded hierarchies and probabilistic computation, Journ. of Comput. and System Sciences 28 (1984), 216–230.

[Schö 88] U. Schöning: The power of counting, Proc. 3rd Structure in Complexity Theory Conf., 1988, 2–9.

[Sze 88] R. Szelepcsényi: The method of forced enumeration for nondeterministic automata, Acta Informatica 26 (1988), 279–284.

[To 89] S. Toda: On the computational power of PP and $\oplus$P, Proc. 30th FOCS, 1989, 514–519.

[To Wa 89] S. Toda, O. Watanabe: On the power of #P functions, manuscript, 1989.

[Va 79a] L.G. Valiant: The complexity of computing the permanent, Theoret. Comp. Sci. 8 (1979), 189–201.

[Va 79b] L.G. Valiant: The complexity of enumeration and reliability problems, SIAM J. Comput. 8 (1979) 410–421.