Theoretical Computer Science 86 (1991) 325-342
Elsevier

# The complexity of computing the number of strings of given length in context-free languages* **

Alberto Bertoni, Massimiliano Goldwurm and Nicoletta Sabadini

*Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, 20133 Milano, Italy*

Communicated by A. Salomaa
Received June 1988

*Abstract*

Bertoni, A., M. Goldwurm and N. Sabadini, The complexity of computing the number of strings of given length in context-free languages, Theoretical Computer Science 86 (1991) 325-342.

Computing the number of strings of given length contained in a language is related to classical problems of combinatorics, formal languages and computational complexity. Here we study the complexity of this problem in the case of context-free languages. It is shown that, for unambiguous context-free languages such a computation is "easy" and can be carried out by efficient parallel algorithms. On the contrary, for some context-free languages of ambiguity degree two, the problem becomes intractable. These results are related to other classical subjects concerning counting problems, exponential time recognizable languages and sparse sets.

## 1. Introduction and summary

Given a language $L \subseteq \Sigma^*$, let us consider the function $f_L$ defined on the set of the unary strings such that, for every nonnegative integer $n$, $f_L(1^n)$ is the number of strings of length $n$ in $L$. We say that $f_L$ is the counting function of the language $L$. The sequence $\{f_L(1^n)\}_n$ can be represented by the analytic function

$$F_L(z) = \sum_{n=0}^{+\infty} f_L(1^n) z^n$$

* Some partial results of this research were presented in the paper "Computing the counting functions of context-free languages", that appeared in *Proc. 4th Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Comput. Sci., Vol. 247 (Springer, Berlin, 1987) 167-179.

** This research has been supported by Ministero della Pubblica Istruzione in the frame of the Project 40% "Progetto e Analisi di Algoritmi".

which is called the generating function of $L$. Note that $F_L$ is analytic in a neighbourhood of 0 and its convergence ratio is between $1/\#\Sigma$ and 1.

The properties of $f_L$ and $F_L$ are particularly interesting when $L$ is context-free for two main reasons. First of all, it is known from an old result due to Chomsky and Schützenberger that, for every unambiguous context-free language $L$, $F_L$ is algebraic [7]. In the literature this property is used to study the inherent ambiguity of context-free languages [28, 1, 14]. In particular, Flajolet has developed some analytic techniques for proving the inherent ambiguity of context-free languages avoiding the traditional (and usually involved) combinatorial approach. Some open problems on the ambiguity of context-free languages have been solved by applying these techniques [14].

In the second place, the functions $f_L$ and $F_L$ are related to a classical combinatorial method proposed by Schützenberger to solve enumeration problems. Roughly speaking, given a class of finite sets $\{A_n\}$, the idea is to associate it with a suitable unambiguous context-free language $L$, so that for every integer $n$, $\#A_n = f_L(1^n)$. The generating function of $L$ is the solution of an algebraic system of equations induced by the unambiguous grammar generating $L$; then its Taylor coefficients are estimated by means of standard analytic tools.

In the literature this approach is used to enumerate classical combinatorial objects as rooted planar trees, random walks and planar graphs (see for instance [21, 22, 11, 12, 16, 13]). For some of these enumeration problems no other combinatorial technique is known. In particular in [13] the method is used to solve an open problem on the enumeration of polyominoes.

These results suggest that computing the counting function of unambiguous context-free languages is "easy" from a computational point of view. Moreover, there are two natural questions arising from this context. The first one is whether the generating functions of context-free languages lie in a particular class of transcendental functions (like, for instance, the class of exponential functions [14]). The second question is whether Schützenberger's method can be extended to inherently ambiguous languages and, in this case, whether the method remains computationally efficient.

In this work we study the complexity of computing the counting functions of polynomially recognizable languages using traditional tools of computational complexity. It is easy to show that, for every language $L$ in P, $f_L$ belongs to the class $\#P_1$ introduced by Valiant in [34]. $\#P_1$ is the class of all the enumeration problems which correspond to NP computations over a single-letter input alphabet. It contains many interesting counting problems on graphs such as, for example, determining how many graphs of size $n$ admit an NP-complete substructure.

We prove that the counting function of every unambiguous context-free language belongs to the class $NC^2$, that is the second level of the hierarchy of efficiently parallelizable problems introduced in [27] and [10]. This result essentially explains why Schützenberger's method is computationally efficient.

On the other hand we prove that there are context-free languages of ambiguity degree two, whose counting function is $\#P_1$-complete. This fact suggests that computing $f_L$ for inherently ambiguous context-free languages is an intractable problem. It gives also evidence that both questions mentioned above admit a negative answer; in particular, if the generating functions of context-free languages were exponential, their Taylor coefficients would be easily computable.

In Section 2 we show that the counting functions of unambiguous context-free languages lie in $NC^2$. In particular, we prove that they can be computed by families of log-space uniform Boolean circuits of polynomial size and depth $O(\log n \log\log n)$. Moreover, we show that the problem is $NC^1$ reducible to computing the determinant of an $n \times n$ matrix with $n$-bit integer entries. This result is a particular case of a more general analysis on the complexity of computing the Taylor coefficients of algebraic functions. The techniques we use are based on classical properties of algebraic functions and the Chinese Remainder Theorem.

In Section 3 we study the relationship between counting functions and enumeration problems; after recalling the definition of the class $\#P_1$ and the corresponding reducibility (called *#1-reducibility*), we prove that all the counting functions of languages in P are in $\#P_1$ and that every function in $\#P_1$ is #1-reducible to the counting function of a context-free language of ambiguity degree two.

In Section 4 we consider the class #EXPTIME of enumeration problems that correspond to nondeterministic exponential time computations; we show a correspondence between functions in #EXPTIME and counting functions of languages in P, proving that $\#P_1$ is included in the class of functions computable in deterministic polynomial time if and only if every function in #EXPTIME is computable in deterministic exponential time.

In Section 5 we present a language $L$ in P whose counting function is $\#P_1$-complete; such a language is related to the regular expressions only containing the operations of union, concatenation and squaring [24, 35]. Then, applying a result due to [19], it follows that, for some context-free language $L$ of ambiguity degree two, if $f_L$ were computable in polynomial time then there would be no sparse sets in $NP - P$.

In this work we make use of the following notation:

$\mathbb{C}, \mathbb{Z}, \mathbb{N}$ denote the set of complex, integer and nonnegative integer numbers respectively;

$\mathbb{C}[x]$ and $\mathbb{Z}[x]$ are the corresponding rings of polynomials;

$\#A$ is the cardinality of the set $A$;

$|z|$ is the length of the string $z$;

$A_k$ denotes the class of context-free languages of ambiguity degree $k$;

DTM denotes deterministic Turing machine;

NDTM denotes nondeterministic Turing machine;

FP denotes the class of functions computable in polynomial time by a DTM;

$\lfloor x \rfloor = \max\{n \in \mathbb{N} \mid n \leq x\}$, $\lceil x \rceil = \min\{n \in \mathbb{N} \mid x \leq n\}$;

For every integer $a$ and every $p \in \mathbb{N}$, $\langle a \rangle_p$ denotes the integer $b \in \{0, 1, \ldots, p-1\}$ such that $b \equiv a \pmod{p}$;

Similarly, for every matrix $B$ with integer components $b_{ij}$, $\langle B\rangle_p$ denotes a matrix $C=[c_{ij}]$ of the same size as $B$ such that $\langle b_{ij}\rangle_p = c_{ij}$ for every $i, j$;
Given two matrices $B=[b_{ij}]$, $C=[c_{ij}]$ of the same size, $B\equiv C \pmod p$ means that for every $i, j$, $b_{ij}\equiv c_{ij} \pmod p$.

## 2. Counting functions of unambiguous context-free languages

Here we are interested in the algebraic analytic functions. We recall that a complex function $f(z)$ is called algebraic if there exists a finite sequence of polynomials $q_0(x), q_1(x), \ldots, q_d(x) \in \mathbb{C}[x]$ such that, for every $z \in \mathbb{C}$,

$$\sum_{j=0}^{d} q_j(z)(f(z))^j = 0.$$

The degree of the algebraic function is the smallest integer $d$ such that the above relation holds.

The following theorem states the main property of the generating functions of unambiguous context-free languages.

**Proposition 2.1** (Chomsky et al. [7]). *If $L$ is an unambiguous context-free language then $F_L(z)$ is algebraic.*

In the following we use this proposition and the properties of algebraic functions to design fast parallel algorithms for $f_L$ when $L$ is unambiguous context-free. In particular we use the following result.

**Proposition 2.2** (Comtet [8]). *Let $f(z)$ be a function analytic in a neighbourhood of 0,*

$$f(z) = \sum_{n=0}^{+\infty} c_n z^n,$$

*and let us assume that $f(z)$ is algebraic of degree $d$, implicitly defined by an equation of the form*

$$\sum_{j=0}^{d} q_j(z)(f(z))^j = 0,$$

*where $q_j \in \mathbb{Z}[x]$ for every $j=0,\ldots,d$. Then there exists an integer $n^0>0$ and a finite sequence of polynomials $p_0(x), p_1(x), \ldots, p_k(x) \in \mathbb{Z}[x]$ such that*

(1) $p_k(x) \neq 0$;
(2) *the degree of $p_j(x)$ is strictly less than $d$ for every $j$;*
(3) *for every $n \geq n^0$, $p_0(n)\cdot c_n + p_1(n)\cdot c_{n-1} + \cdots + p_k(n)\cdot c_{n-k} = 0$.*

To classify the function $f_L$ with respect to parallel complexity, we recall the definitions of uniform Boolean circuit and of the hierarchy $\{\mathrm{NC}^k\}$.

Uniform Boolean circuits are considered a standard model of parallel computation and are widely studied in the literature [5, 27, 9]. Informally, a family of uniform Boolean circuits is a sequence of combinational circuits satisfying a suitable condition of uniformity which allows a DTM (or other abstract machine) to generate "easily" a description of each circuit. Here we consider the "log-space uniformity", introduced by [5]: a sequence of Boolean circuits $\{c_n\}$ is a family of log-space uniform Boolean circuits if some DTM generates, for any input $n \in \mathbb{N}$, a description of the circuit $c_n$ using $\log n$ work space. We also recall that size and depth of a circuit are respectively the number of nodes and the length of the longest path from an input node to an output node; these measures correspond respectively to hardware and time costs in parallel computations.

**Definition 2.3.** For every integer $k$, $\mathrm{NC}^k$ is the set of problems computable by a family of log-space uniform Boolean circuits of depth $O(\log^k n)$ and size $n^{O(1)}$.

The class NC, defined as

$$\mathrm{NC} = \bigcup_{k=1}^{+\infty} \mathrm{NC}^k,$$

has been proposed as the representative class of problems which admit fast parallel algorithms. NC is robust with respect to other formalisms: for instance, it can be defined as the class of problems solvable on parallel RAM in polylog time with a polynomial number of processors [25]. Clearly, all the problems in NC can be solved in polynomial time on DTM.

In [10] many problems between $\mathrm{NC}^1$ and $\mathrm{NC}^2$ are classified using a special notion of reducibility called $\mathrm{NC}^1$-reducibility. Informally a problem $f$ is $\mathrm{NC}^1$-reducible to another problem $g$ if $f$ can be computed in $\mathrm{NC}^1$ except for requiring the computation of $g$ (for a more precise definition see [10]). An interesting class is $\mathrm{DET} \subseteq \mathrm{NC}^2$ defined as the class of problems $\mathrm{NC}^1$-reducible to computing the determinant of an $n \times n$ matrix with $n$-bit integer entries; for example, we recall the problems of computing the inverse and the first $n$ powers of an $n \times n$ matrix with $n$-bit integer entries.

The main result of this section concerns the complexity of computing the Taylor coefficients of algebraic functions.

**Proposition 2.4.** *Let $f(z)$ be an algebraic function as defined in Proposition* 2.2, *$f(z) = \sum_{n=0}^{+\infty} c_n z^n$ and assume that, for every $n$, $c_n \in \mathbb{N}$. Then, the following problem*:

*Input*: $1^n$, *Output*: *$c_n$ in binary notation,*

*can be solved by a family of log-space uniform Boolean circuits of size polynomial in $n$ and depth* $O(\log n \log\log n)$.

**Proof.** By Proposition 2.2, we can determine an integer $n^0 > 0$ and a finite set of polynomials $p_0(x), p_1(x), \ldots, p_k(x) \in \mathbb{Z}[x]$ such that, for every $n \geq n^0$, $p_0(n) \cdot c_n +$

$p_1(n)\cdot c_{n-1}+\cdots+p_k(n)\cdot c_{n-k}=0$. From this relation we deduce that, $\forall n\geqslant n^0$,

$$c_n=r_1(n)\cdot c_{n-1}+\cdots+r_k(n)\cdot c_{n-k} \qquad (*)$$

where, for every $j=1,\ldots,k$, $r_j(x)=-p_j(x)/p_0(x)$. Now let us define the sequence of column arrays $\{\underline{z}(n)\}$, where

$$\underline{z}(n)=(c_{n-1},c_{n-2},\ldots,c_{n-k})^{\mathrm{T}}.$$

So, by relation $(*)$, it is easily shown that, for every $n\geqslant n^0$,

$$\underline{z}(n+1)=A[n]\cdot\underline{z}(n)$$

where $\cdot$ is the matrix product and $A[n]$ is the following $k\times k$ matrix:

$$\begin{bmatrix} r_1(n) & r_2(n) & r_3(n) & \cdots & r_{k-1}(n) & r_k(n)\\ 1 & 0 & 0 & \cdots & 0 & 0\\ 0 & 1 & 0 & \cdots & 0 & 0\\ 0 & 0 & 1 & \cdots & 0 & 0\\ \vdots & \vdots & \vdots & & \vdots & \vdots\\ 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$$

Therefore, the calculus of $c_n$ is reduced to computing the upper-leftmost component of the matrix

$$\left(\prod_{j=n^0}^{n} A[j]\right)\cdot Z,$$

where $Z$ is the $k\times k$ matrix whose first column is $\underline{z}(n^0)$ and all the other components are null. Moreover, for every matrix $B$ and every integer $p$, let $B/p$ denote the matrix obtained dividing each component of $B$ by $p$. Hence we can write the matrix $A[n]$ in the form $B[n]/p_0(n)$, where $B[n]$ is a $k\times k$ matrix whose components are polynomials with integer coefficients. It follows that $c_n$ is the upper-leftmost component of the matrix $A/b$, where $A=(\prod_{j=n^0}^{n}B[j])\cdot Z$, and $b=\prod_{j=n^0}^{n}p_0(j)$.

We can now summarize the tasks of the Boolean circuit computing $c_n$ in the following steps:

(1) Determine the binary representation of $n$ and then compute in parallel the matrices $B[n^0], B[n^0+1],\ldots,B[n]$, and the $O(\log n)$-bit integers $p_0(n^0), p_0(n^0+1),\ldots,p_0(n)$;

(2) Compute the array $\underline{z}(n^0)$ and the matrix $Z$;

(3) Compute the integer

$$b=\prod_{j=n^0}^{n}p_0(j);$$

(4) Compute the $k\times k$ matrix

$$A=\left(\prod_{j=n^0}^{n}B[j]\right)\cdot Z$$

and determine its upper-leftmost component $a_{11}$;

(5) Evaluate $a_{11}/b = c_n$.
The computation of Step (1) belongs to $\mathrm{NC}^1$ since sum and product of two $\mathrm{O}(n)$-bit integers lie in $\mathrm{NC}^1$ [10]. The array $\underline{z}(n^0)$ does not depend on the input since $n^0$ is fixed and only requires constant size and depth.

As regards Steps (3) and (5), we recall that the product of $n$ integers of $n$ bits each and the division of two $n$-bit integers can be carried out by log-space uniform Boolean circuits of polynomial size and $\mathrm{O}(\log n \log\log n)$ depth [26]; since both $b$ and $a_{11}$ are $\mathrm{O}(n \log n)$-bit integers, also Step (3) and Step (5) have circuit complexity $\mathrm{O}(\log n \log\log n)$.

Finally, by Lemma 2.8 proved below, the matrix $A$ can be computed by log-space uniform Boolean circuits of polynomial size and $\mathrm{O}(\log n \log\log n)$ depth. □

**Corollary 2.5.** *For every unambiguous context-free language L, the following problem:*

*Input*: $1^n$, *Output*: $f_L(1^n)$,

*belongs to* DET.

**Proof.** Since the division of two $n$-bit integers is $\mathrm{NC}^1$ equivalent to the iterated product of $n$ $n$-bit integers [2], we have that computing Steps (3) and (5) of the algorithm described in the previous proposition is in DET. By a result due to Cook and Berkowitz, it is well-known that the iterated product of $n$ $n \times n$ matrices with $n$-bit integer entries belongs to DET [10, 3]. So, also Step (4) is in DET and hence the problem of computing the coefficients $\{c_n\}$ is in DET. Therefore the result follows from Proposition 2.1. □

To complete the proof of Proposition 2.4 we use the Chinese Remainder Theorem which is a classical tool in analysis of algorithms for algebraic operations [29, 23].

**Lemma 2.6.** *Let $p_1, p_2, \ldots, p_r$ be pairwise relatively prime integers (i.e.* $\gcd(p_i, p_j) = 1$ $\forall i \neq j$*) and let $p$ be the product of all the $p_i$, $i = 1, \ldots, r$. Given $r$ matrices $B_1, \ldots, B_r$ of size $k \times k$ with integer components, there exists a unique matrix $B$ with integer components such that $B \equiv B_i \pmod{p_i}$ for every $i = 1, \ldots, r$, and*

$$B = \left\langle \sum_{i=1}^{r} m_i s_i B_i \right\rangle_p ,$$

*where $m_i = p/p_i$ and $s_i = (m_i)^{-1} \pmod{p_i}$.*

**Proof.** Applying the Chinese Remainder Theorem to each component of $B$ we obtain the result. □

**Lemma 2.7.** *Let $B$ be a matrix $k \times k$ with integer components of* $\mathrm{O}(\log n)$ *bits each and let $p$ be an integer $p = \mathrm{O}(n \log^2 n)$. Then, the problem of computing $\langle B \rangle_p$ is in* $\mathrm{NC}^1$.

**Proof.** Let $b$ be a component of $B$. We have

$$b = \sum_{i=0}^{r} b_i 2^i,$$

where $b_i \in \{0, 1\}$ for every $i$ and $r = O(\log n)$. We can compute in parallel all the values $\langle 2^i \rangle_p$, $(i = 0, \ldots, r)$, by determining for every index $i$ the integer $k_i$ such that $0 \leq 2^i - k_i p \leq p - 1$. Such a computation requires depth $O(\log n)$ and polynomial size. Recalling that the sum of $n$ integers of $n$ bits each belongs to $NC^1$ [10], we have that the sum

$$s = \sum_{i=0}^{r} b_i \langle 2^i \rangle_p$$

can be computed in depth $O(\log \log n)$ and polynomial size. Since $0 \leq s < O(\log n)p$, also computing $\langle s \rangle_p$ requires $O(\log \log n)$ depth and the lemma is proved. Note that only the computation of $\langle 2^i \rangle_p$, $(i = 0, \ldots, r)$, requires $O(\log n)$ depth, while all the other steps can be carried out in $O(\log \log n)$ depth. □

**Lemma 2.8.** *Let $B_1, B_2, \ldots, B_n$ be $k \times k$ matrices with integer components of* $O(\log n)$ *bits each and assume that all the components of the matrix*

$$A = \prod_{i=1}^{n} B_i$$

*are nonnegative integers. Then the matrix $A$ can be computed by a family of log-space uniform Boolean circuits of polynomial size and* $O(\log n \log \log n)$ *depth.*

**Proof.** Let $a_{ij}$ be the $i, j$th component of the matrix $A$. From the hypotheses it is easy to verify that there exist two positive integers $\alpha, \beta$ such that the absolute value of all the components of the matrices $B_1, B_2, \ldots, B_n$ is lower than $\alpha n^{\beta}$; this implies that, for every $i, j$,

$$0 \leq a_{ij} \leq (\alpha n^{\beta})^n k^n = \delta^n n^{\gamma n} \quad \text{for suitable constants } \delta \text{ and } \gamma.$$

Let $p_1, p_2, \ldots, p_r$ be the smallest $r$ primes where $r = \gamma n \lceil \log_2 n \rceil + n \lceil \log_2 \delta \rceil$. Since the sequence $p_1, p_2, \ldots, p_r$ does not depend on the input, we can compute it in advance and weave the results into the circuit. Note that such an operation requires only logarithmic space. Moreover, we have

$$p = \prod_{i=1}^{r} p_i > \delta^n n^{\gamma n} \geq a_{ij}$$

for every $i, j$. The last equality implies $\langle A \rangle_p = A$.

Hence the computation of $A$ can be carried out via the following steps:

(1) Compute

$$p = \prod_{i=1}^{r} p_i;$$

(2) Compute in parallel all the matrices $C_{ij} = \langle B_i \rangle_{p_j}$, for every $j = 1, \ldots, r$ and $i = 1, \ldots, n$;

(3) Compute in parallel all the values $\langle 2^i \rangle_{p_j}$, for every $j = 1, \ldots, r$ and $i = 0, \ldots, \lfloor 2 \log_2 p_j \rfloor$;

(4) Compute in parallel all the matrices

$$H_j = \left\langle \prod_{i=1}^{n} C_{ij} \right\rangle_{p_j},$$

where $j = 1, \ldots, r$.

(5) Using Lemma 2.6, compute $\langle A \rangle_p$ from $H_1, H_2, \ldots, H_r$ and $p_1, p_2, \ldots, p_r$.

By the Prime Number Theorem we know that $p_r = \mathrm{O}(r \log r)$ and hence the integer $p$ can be computed by log-space uniform Boolean circuits of polynomial size and depth $\mathrm{O}(\log n \log\log n)$. Applying Lemma 2.7, we have that the computation of Steps (2) and (3) belongs to $\mathrm{NC}^1$.

Now, given two $k \times k$ matrices $B$ and $C$ with components in $\{0, 1, \ldots, p_j\}$, since all the coefficients $\langle 2^i \rangle_{p_j}$ have already been computed, by the remark at the end of the proof of Lemma 2.7, the computation of $\langle B \cdot C \rangle_{p_j}$ requires polynomial size and depth $\mathrm{O}(\log\log p_j) = \mathrm{O}(\log\log n)$.

Then, using a standard divide and conquer technique, Step (4) can be executed by Boolean circuits of depth $\mathrm{O}(\log n \log\log n)$ and polynomial size.

Step (5) can be carried out by computing the matrix

$$H = \sum_{i=1}^{r} m_i s_i H_i,$$

where $m_i = p/p_i$ and $s_i = (m_i)^{-1} \pmod{p_i}$. Since the prime numbers $p_1, \ldots, p_r$ have $\mathrm{O}(\log n)$ bits, the coefficients $m_i$ and $s_i$ can be computed in polynomial size and depth $\mathrm{O}(\log n)$ ([26] and [2, Lemma 4.1]). The same circuit complexity is required to compute each product $m_i s_i H_i$ and their sum $H$. Each component $h$ of the matrix $H$ satisfies the relation $0 \leqslant h \leqslant \mathrm{O}(n^2 \log^3 n)p$. Since it is easy to verify that $p$ is a $\mathrm{O}(n \log^3 n)$-bit integer, reasoning as in the proof of Lemma 2.7, $\langle h \rangle_p$ can be computed in depth $\mathrm{O}(\log n)$ and polynomial size. By Lemma 2.6 $A = \langle A \rangle_p$ is equal to $\langle H \rangle_p$ and hence also Step (5) requires polynomial size and depth $\mathrm{O}(\log n)$.

Finally, note that all the circuits considered above are log-space uniform. □

## 3. Counting functions and the class $\#\mathrm{P}_1$

In this section we relate the counting functions of languages in P to the complexity of enumeration problems studied in [33, 34]. First of all we recall the notion of counting Turing machine.

**Definition 3.1.** A *counting Turing machine* (CTM) is a NDTM with an auxiliary output device that prints on a special tape the number of accepting computations induced by the input. We say that a function $f:\{0,1\}^* \to \mathbb{N}$ is computable by a CTM $M$ if, for every $x \in \{0,1\}^*$, $f(x)$ is the number of accepting computations of $M$ on input $x$. A CTM $M$ has time complexity $T(n)$ if, regarding $M$ as a standard NDTM, the longest accepting computation induced by any input of size $n$ takes at most $T(n)$ steps.

The class #P of the functions computable in polynomial time by a counting Turing machine has been widely studied in [33, 34]. Also the class #PSPACE of the functions computable by CTM in polynomial space has been considered in the literature [4]. Another interesting class of counting problems is the class $\#\mathrm{P}_1$ of all the functions $f:\{1\}^* \to \mathbb{N}$ computable in polynomial time by a CTM which has unary input alphabet [34].

To introduce a natural notion of reduction in $\#\mathrm{P}_1$, we define an *oracle Turing machine* (OTM) as a Turing machine $M$ with an oracle function $h_M:\{1\}^* \to \mathbb{N}$ and unary input alphabet; it has a query tape, an answer tape and a work tape. To consult the oracle, the machine $M$ prints a unary string $1^m$ on the query tape, enters a special query state and returns in unit time the binary representation of $h_M(1^m)$ on the answer tape.

**Definition 3.2.** Given two functions $f, g:\{1\}^* \to \mathbb{N}$, we say that $f$ is *#1-reducible to g* if there exists an oracle Turing machine $M$ computing $f$ in polynomial time which calls an oracle for $g$ only once. Moreover, for every input $1^n$, $M$ produces the binary representation of $f(1^n)$ in polynomial time.

It is easy to verify that #1-reducibility is transitive. A function $f$ is said to be $\#\mathrm{P}_1$-complete if $f \in \#\mathrm{P}_1$ and for every $g \in \#\mathrm{P}_1$, $g$ is $\#_1$-reducible to $f$. The class $\#\mathrm{P}_1$ has been introduced in [34] where a notion of reducibility is defined which is equivalent to ours except for not requiring one oracle call only. We note that our $\#\mathrm{P}_1$-complete functions are complete also with respect to Valiant's reduction.

**Proposition 3.3.** *For every language $L$ in* P, *$f_L$ belongs to* $\#\mathrm{P}_1$.

**Proof.** Let $M$ be a DTM recognizing a language $L \subseteq \Sigma^*$ in polynomial time. We can determine a NDTM $M'$ such that, on input $1^n$, $M'$ generates nondeterministically all the strings of length $n$ in $\Sigma^*$. Then, having a string $y$ of length $n$ on its tape, $M'$ simulates the machine $M$ on input $y$. Clearly, $M'$ works in polynomial time and the number of accepting computations is $f_L(1^n)$. □

Now we want to prove that the functions in $\#\mathrm{P}_1$ are at least as difficult as counting functions of languages in $A_2$. To prove it we recall a well-known result due to Hartmanis which relates context-free languages and Turing machine computations.

Given a NDTM $M$, we can assume without loss of generality that all the halting computations are accepting and consist of an odd number of transitions. So, every accepting computation of $M$ is a sequence of instantaneous descriptions $w_1$, $w_2, \ldots, w_{2n}$ where $w_1$ is an initial configuration, for every $i = 1, \ldots, 2n-1$ $w_i \vdash_M w_{i+1}$ and $w_{2n}$ is an accepting configuration; such a computation can be represented by a string of the form

$$w_1 m w_2^{\mathrm{T}} m w_3 m \ldots m w_{2n}^{\mathrm{T}} m m \qquad (*)$$

where $m$ is a special mark which does not denote a tape or a state symbol of $M$ and, for every $i = 1, \ldots, n$, $w_{2i}^{\mathrm{T}}$ is the reversal of the string $w_{2i}$. We say that a string of the form $(*)$ is an alternating description of accepting computation of $M$. Moreover, let $q_0$ be the initial state of $M$ and let $S$ and $H$ be respectively the set of all the initial and halting instantaneous descriptions of $M$; let us denote by $H^{\mathrm{T}}$ the set of all the reversals of strings belonging to $H$. Then we can define the following languages:

$$L_0(M) = \{wmu^{\mathrm{T}}m \mid w \vdash_M u\}^*.m,$$

$$L_1(M) = S.m.\{w^{\mathrm{T}}mum \mid w \vdash_M u\}^*.H^{\mathrm{T}}.mm.$$

(For the sake of simplicity we omit the brackets $\{\ldots\}$ in the representation of singletons and we denote by . the product between languages.)

The following lemma states the main properties of $L_0(M)$ and $L_1(M)$.

**Lemma 3.4** (Hartmanis [18]). *For every* NDTM $M$, *we have*

(1) $L_0(M)$ *and* $L_1(M)$ *are deterministic context-free languages, and*

(2) $L_0(M) \cap L_1(M)$ *is the set of all the alternating descriptions of accepting computations of* $M$.

**Proposition 3.5.** *For every function* $f \in \#\mathrm{P}_1$, *there exists a language* $L \in A_2$ *such that* $f$ *is* $\#1$*-reducible to* $f_L$.

**Proof.** Given a function $f \in \#\mathrm{P}_1$, let $M$ be a CTM computing $f$ in time $cn^c$ for some constant $c > 0$. By a standard padding technique we can determine a CTM $M'$ computing $f$ in polynomial time such that, for every positive integer $n$, all the alternating descriptions of accepting computations on input $1^n$ have the same length. To be more precise we describe the machine $M'$ in detail.

Such a machine works in two phases: in the first one, given an input $1^n$, $M'$ marks the leftmost $cn^c + 1$ cells of the tape as work space. In the second phase $M'$ simulates $M$ on input $1^n$ as follows: each step of $M$ is simulated by scanning the workspace from its leftmost cell to its rightmost one and back again; so every step of $M$ corresponds to a "cycle" of $2cn^c$ steps of $M'$. During this phase $M'$ takes also account of the current number of $M$-steps that have been simulated; to count this number, for each cycle, $M'$ marks a cell of the work space by an extra symbol preserving the previous information. Such an operation only requires an extension of the work alphabet. If $M$ halts on input $1^n$ in $t < cn^c$ steps, then $M'$, after simulating all the steps of $M$, performs other $cn^c - t$ cycles to mark all the cells of the work space and then halts. In any other case $M'$ halts when all the cells of the work space have been marked. A given computation is accepting if and only if, at some point of the computation, $M'$ has simulated an accepting configuration of $M$. Clearly, $f(1^n)$ is the number of accepting computations of $M'$.

Note that all the computations of $M'$ on input $1^n$ run for $q(n)$ steps, where $q$ is a suitable polynomial; moreover, for every $i = 1, \ldots, q(n)$, the $i$th instantaneous descriptions of all these computations have the same length (in particular during the second phase they have length $cn^c + 1$). It follows that there exists a polynomial $p$ such that for every $n \in \mathbb{N}$ all the alternating descriptions of accepting computations of $M'$ on input $1^n$ have length $p(n)$. Note also that, for every $n, m \in \mathbb{N}$, $n \neq m$ implies $p(n) \neq p(m)$.

Now, let us consider the languages $A = L_0(M')$, $B = L_1(M')$ as in Lemma 3.4. Then, by Lemma 3.4, we have

(a) $\forall n \in \mathbb{N}\, f(1^n) = f_{A\cap B}(1^{p(n)})$,

(c) $f_{A\cup B} = f_A + f_B - f_{A\cap B}$.

By Proposition 2.4, since $A$ and $B$ are unambiguous, $f_{A\cup B} \in \mathrm{FP}$ if and only if $f_{A\cap B} \in \mathrm{FP}$. Hence, a polynomial OTM for $f$ can be derived which uses the function $f_{A\cup B}$ as oracle. □

## 4. Counting functions and the class #EXPTIME

In this section we consider the class of functions $f : \{0, 1\}^* \to \mathbb{N}$ computable in exponential time by counting Turing machines.

For every constant $c$, let $\#\mathrm{TIME}(2^{cn})$ be the class of functions $f : \{0, 1\}^* \to \mathbb{N}$ computable by CTM in time $O(2^{cn})$. Then we define

$$\#\mathrm{EXPTIME} = \bigcup_{c=1}^{+\infty} \#\mathrm{TIME}(2^{cn}).$$

Analogously, we denote by F-EXPTIME the class of functions $f : \{0, 1\}^* \to \mathbb{N}$ computable by DTM in time $2^{cn}$ for some constant $c$.

In order to obtain a more precise comparison between $f_L(1^n)$ for $L \in \mathrm{P}$ and complexity classes of counting problems, we slightly modify the definition of $f_L$. Let us consider the bijective function $B : \mathbb{N} \to \{1\}.\{0, 1\}^* \cup \{0\}$ where, for every integer $n$, $B(n)$ is the binary representation of $n$. Given a language $L \subseteq \Sigma^*$, we define the function $f_L^b : \{0, 1\}^* \to \mathbb{N}$ as follows:

$$f_L^b(x) = \#\{z \in L \mid |z| = n \text{ and } B(n) = 1x\}.$$

Now, the following theorems allow us to characterize the class of counting problems associated with $f_L^b$, where $L$ belongs to P.

**Proposition 4.1.** *For every language $L$ in* P, *$f_L^b$ belongs to* #EXPTIME.

**Proof.** Similar to the proof of Proposition 3.3. □

**Proposition 4.2.** *Let $f:\{0,1\}^*\to\mathbb{N}$ be a function in* #EXPTIME. *There exists a language $L\in$* P *and a polynomial $p$ such that*

$$f(x)=\#\{z\in L\,|\,|z|=p(n) \text{ and } B(n)=1x\}.$$

**Proof.** Since $f\in$ #EXPTIME, there exists a NDTM $M$ such that all the accepting computations of $M$ on input $x\in\{0,1\}^*$ run for $\leqslant 2^{c|x|}$ steps for a suitable constant $c$; moreover, there are exactly $f(x)$ accepting computations.

Applying the padding technique described in the proof of Proposition 3.5, we can determine a NDTM $M'$ simulating $M$ such that, for every input $x$, all the alternating descriptions of accepting computations of $M'$ on $x$ have the same length and, for every pair of distinct inputs $x$, $y$, the corresponding alternating descriptions have different length. As in Proposition 3.5, given an input $x$, $M'$ first marks the leftmost $n^c$ cells as work space, where $B(n)=1x$; then it simulates each step of $M$ by scanning the work space in both directions and counting in parallel the number of simulated moves. $M'$ halts after scanning the work space $n^c$ times and accepts the input if and only if it has simulated an accepting configuration of $M$. Clearly, for every input $x$, $M'$ has $f(x)$ many accepting computations running for $q(n^c)$ steps for a suitable polynomial $q$.

Let $L$ be the language such that $L=L_0(M')\cap L_1(M')$. Then, by Lemma 3.4 it is easy to verify that, for a suitable polynomial $p$,

$$f(x)=\#\{z\in L\,|\,|z|=p(n) \text{ and } B(n)=1x\}.$$

Furthermore, we have that $L$ belongs to P, because the alternating descriptions of accepting computations of a NDTM are easily recognized by a DTM in polynomial time. □

**Definition 4.3.** For every function $f\in$ #EXPTIME, we denote by $L_f$ the language in P built up in Proposition 4.2 and satisfying the relation

$$f(x)=\#\{z\in L\,|\,|z|=p(n) \text{ and } B(n)=1x\}.$$

**Corollary 4.4.** $\#\mathrm{P}_1\subseteq\mathrm{FP}$ *if and only if* #EXPTIME = F-EXPTIME.

**Proof.** Immediate from Propositions 3.5, 4.1, 4.2. □

## 5. Counting functions complete in $\#\mathrm{P}_1$

In this section we exhibit a language $L\in$ P whose counting function is $\#\mathrm{P}_1$-complete. The proof is based on a slight extension of a well-known result due to

Meyer and Stockmeyer concerning the complexity of word problems requiring exponential time [24]. It follows that there exists a context-free language of ambiguity degree two whose counting function is also $\#P_1$-complete. As a consequence, for some language $L \in A_2$, if $f_L$ belongs to FP then there are no sparse sets in NP − P.

**Definition 5.1.** Given a finite set $S$, we denote by $\mathrm{RE}(S, \cup, ., ^2)$ the set of all the correctly parenthesized regular expressions involving all the symbols $\sigma \in S$, the empty word $\varepsilon$, the binary operations $\cup$ and . of union and concatenation, and the unary squaring operation $^2$. Similarly, $\mathrm{RE}(S, \cup, .) \subseteq \mathrm{RE}(S, \cup, ., ^2)$ is the subset of all regular expressions containing only the operations of union and concatenation.

We recall that every expression $e \in \mathrm{RE}(S, \cup, ., ^2)$ represents a language $L(e) \subseteq S^*$ defined in the usual way: $L(\varepsilon) = \{\varepsilon\}$; $\forall \sigma \in S$, $L(\sigma) = \{\sigma\}$; $\forall e_1, e_2 \in \mathrm{RE}(S, \cup\ ., ^2)$,

$$L(e_1 \cup e_2) = L(e_1) \cup L(e_2), \qquad L(e_1.e_2) = L(e_1).L(e_2),$$

$$L(e_1^2) = L(e_1).L(e_1).$$

Note that, for every $e \in \mathrm{RE}(S, \cup, ., ^2)$, $\#L(e) < +\infty$.

**Lemma 5.2.** *There exists a* DTM $M$ *such that, for every input* $(exp, x)$ *where* $exp \in \mathrm{RE}(\{0, 1\}, \cup, .)$ *and* $x \in \{0, 1\}^*$, $M$ *checks whether* $x \in L(exp)$ *within time polynomial in* $|exp|$.

**Proof.** Note that $|x| > |exp|$ implies $x \notin L(exp)$, and hence we consider only the case $|x| \leq |exp|$. First the DTM $M$ computes a nondeterministic finite states automaton $A_{exp}$ which recognizes the language $L(exp)$. Such a computation can be carried out as in [32, Proposition 4.11] and requires time polynomial in $|exp|$. Let $Q$, $q_0$, $F$ and $\delta$ be respectively the set of states, the initial state, the final states and the transition relation of $A_{exp}$. Then, given the string $x = x_1x_2 \ldots x_m$, the DTM $M$ implements the following procedure and checks whether $x \in L(exp)$.

```
begin
  $A := \{q_0\}$
  for every $i = 1, \ldots, m$ do
  begin
    $B := \{q \in Q \mid \exists q' \in A$ and $q \in \delta(q', x_i) \cup \delta(q', \varepsilon)\}$
    $A := B$
  end
  if $A \cap F = \emptyset$ then reject
                 else accept
end
```

It is easy to check that the previous procedure is polynomial and hence the lemma is proved. □

Given a bijective binary encoding of the expressions in $\mathrm{RE}(\{0,1\}, \cup, ., ^2)$ (see, for instance, [15]), for every $exp \in \mathrm{RE}(\{0,1\}, \cup, ., ^2)$ let $\langle exp\rangle$ denote the encoding of *exp*. Note that $|\langle exp\rangle| = \mathrm{O}(|exp|)$.

**Definition 5.3.** Let us define the function $\mathrm{Rex}: \{0,1\}^* \to \mathbb{N}$ such that

$$\forall y \in \{0,1\}^* \quad \mathrm{Rex}(y) = \# L(exp), \quad \text{where } y = \langle exp\rangle.$$

**Proposition 5.4.** $\mathrm{Rex} \in \#\mathrm{EXPTIME}$.

**Proof.** Given an input string $y = \langle exp\rangle$, the CTM $M'$ computing $\mathrm{Rex}(y)$ works in two phases. In the first one, replacing each subexpression $(s)^2$ by $(s.s)$, $M'$ eliminates the square operation in *exp* and produces an expression $exp' \in \mathrm{RE}(\{0,1\}, \cup, .)$. Note that $L(exp) = L(exp')$ and $|exp'| \leq 2^{|exp|}$. Then, $M'$ generates nondeterministically all the strings $x \in \{0,1\}^*$ such that $|x| \leq |exp'|$. This computation requires $2^{\mathrm{O}(|exp|)}$ nondeterministic time.

In the second phase, for every string $x \in \{0,1\}^*$ of length less or equal to $|exp'|$, $M'$ checks deterministically whether $x \in L(exp')$ and, in the affirmative case, accepts the input; by Lemma 5.2 this can be performed within time $2^{\mathrm{O}(|exp|)}$.

Hence the longest accepting computation runs for $2^{\mathrm{O}(|y|)}$ steps. Moreover, since $L(exp)$ does not contain any string $x$ of length larger than $2^{|exp|}$, we conclude that $\# L(exp)$ is exactly the number of accepting computations of $M'$. □

Now we recall a classical result due to Meyer and Stockmeyer concerning exponential time complexity classes.

**Proposition 5.5** (Meyer et al. [24, Theorem 3.1]). *Given a* CTM *$M$ of time complexity $2^{cn}$, let us consider a binary encoding of the computations of $M$. Then, for every input $x$, there exists a regular expression $e_{M,x} \in \mathrm{RE}(\{0,1\}, \cup, ., ^2)$ such that*

(1) $L(e_{M,x}) = \{z \in \{0,1\}^* \mid |z| \leq b(|x|)$, *$z$ is not the binary encoding of an accepting computation of $M$ on input $x$*$\}$, *where $b$ is an integer-valued function such that, for a suitable polynomial $p$, $b(n)$ is computable by a* DTM *in time $p(n)$*;

(2) *$e_{M,x}$ is computable by a* DTM *in space* $\log(|x|)$;

(3) $|e_{M,x}| = \mathrm{O}(|x|)$.

**Proposition 5.6.** *The counting function of the language $L_{\mathrm{Rex}}$ is* $\#\mathrm{P}_1$-complete.

**Proof.** For the sake of simplicity, let us denote $L_{\mathrm{Rex}}$ by $L'$. So we have to prove that $L' \in \mathrm{P}$ and, for every $L \in \mathrm{P}$, $f_L$ is $\#1$-reducible to $f_{L'}$. Since Rex belongs to $\#\mathrm{EXPTIME}$, by Proposition 4.2 we know that $L'$ belongs to P and there exists a polynomial $p$ such that

$$\forall y \in \{0,1\}^* \quad \mathrm{Rex}(y) = \#\{z \in L' \mid |z| = p(n),\ B(n) = 1y\} \qquad (1)$$

By Proposition 4.1, for every $L \in \mathrm{P}$, $f_L^b \in \#\mathrm{EXPTIME}$ and hence there exists a CTM $M$ computing $f_L^b$ which works in time $2^{c|x|}$ for some constant $c$ and every input $x$. Moreover, by Proposition 5.5, there exists a log-space computable function $h_M : \{0,1\}^* \to \{0,1\}^*$ such that $h_M(x) = \langle e_{M,x} \rangle$ and $e_{M,x}$ satisfies conditions (1)-(3) of Proposition 5.5. It follows that, for every input $x$,

$$f_L^b(x) = \#\{z \in \{0,1\}^* \mid |z| \leq b(|x|)\} - \mathrm{Rex}(h_M(x))$$

where the function $b(n)$, defined as in Proposition 5.5, is computable in polynomial time with respect to $n$. For each $n \in \mathbb{N}$, let $x$ and $m$ be given by $1x = B(n)$ and $B(m) = 1h_M(x)$ respectively. Then, using the last equality together with condition (1) above, we have

$$\begin{aligned} f_L(1^n) &= \#\{z \in \{0,1\}^* \mid |z| \leq b(\lfloor \log n \rfloor)\} - \#\{z \in L' \mid |z| = p(m)\} \\ &= (2^{b(\lfloor \log n \rfloor)+1} - 1) - f_{L'}(1^{p(m)}). \end{aligned} \tag{2}$$

Clearly the first term of (2) and the integer $m$ are computable in polynomial time with respect to $n$. So, to prove that $f_L$ is $\#_1$-reducible to $f_{L'}$, we have only to show that $p(m)$ is polynomially bounded by $n$. Since, by Proposition 5.5, $|h_M(x)| = \mathrm{O}(|x|)$, we have $m \leq 2^{\mathrm{O}(|x|)}$ and hence $p(m) \leq q(n)$ for a suitable polynomial $q$. $\square$

From Propositions 3.5 and 5.6 and Corollary 4.4, the following statements are immediate.

**Proposition 5.7.** *There exists a language $L \in A_2$ whose counting function is $\#\mathrm{P}_1$-complete.*

**Proposition 5.8.** *There exists a language $L \in A_2$ such that*

$$f_L \in \mathrm{FP} \quad \textit{if and only if} \quad \#\mathrm{EXPTIME} = \text{F-EXPTIME}.$$

Now we recall that, for a constant $c$, $\mathrm{TIME}(2^{cn})$ (resp. $\mathrm{NTIME}(2^{cn})$) denotes the class of languages recognized by a DTM (resp. NDTM) in time $\mathrm{O}(2^{cn})$. Then the classes EXPTIME and NEXPTIME are defined as follows:

$$\mathrm{EXPTIME} = \bigcup_{c=1}^{+\infty} \mathrm{TIME}(2^{cn}), \qquad \mathrm{NEXPTIME} = \bigcup_{c=1}^{+\infty} \mathrm{NTIME}(2^{cn}).$$

Although the problem whether EXPTIME = NEXPTIME is still open, it is conjectured that the two classes are different.

The properties of EXPTIME and NEXPTIME are related to the properties of sparse sets in NP − P. We recall that a language $S \subseteq \Sigma^*$ is called a sparse set if there exists a polynomial $p$ such that $\#\{x \in S \mid |x| \leq n\} \leq p(n)$. Sparse sets are widely studied in literature; in particular, in [19] it is proved that EXPTIME = NEXPTIME if and only if there are no sparse sets in NP − P.

Since #EXPTIME = F-EXPTIME implies NEXPTIME = EXPTIME, from the previous propositions we obtain the following result.

**Proposition 5.9.** *There is a language $L \in A_2$ such that, if $f_L$ is computable in polynomial time, then there are no sparse sets in* NP − P.

## 6. Conclusions and open problems

In Section 2 we have shown that computing $f_L$ for unambiguous context-free languages belongs to the class DET of problems $NC^1$-reducible to computing the determinant of an $n \times n$ matrix with $n$-bit integer entries. Clearly, computing $f_L$ for unambiguous $L$ does not seem to be complete for DET, since the most "difficult" operation used to solve the problem is the iterated product of matrices of fixed size (Step (4) in the proof of Proposition 2.4); such an operation looks easier than the iterated product of $n$ matrices of size $n \times n$ (itmatprod), which is complete for DET [10]. Actually, we have proved a stronger result: $f_L$ for unambiguous context-free $L$ can be computed by a log-space uniform family of Boolean circuits with depth $O(\log n \log \log n)$ and polynomial size. It would be interesting to know whether this problem belongs to $NC^1$. Anyway, such a question does not appear to be easily solvable, since it is still an open problem whether integer division belongs to $NC^1$, and computing $f_L$ for unambiguous $L$ seems to be as difficult as integer division.

In Section 5 we have shown that there is a context-free language $L$ of ambiguity degree two such that computing $f_L$ is difficult unless EXPTIME = NEXPTIME. This means that Schützenberger's method cannot be extended to the whole class of context-free languages. Nevertheless, it has been observed that the generating functions of many inherently ambiguous context-free languages have easily computable Taylor coefficients [14]. It would be interesting to characterize the class of these languages (or significant subclasses), since it represents those ambiguous languages to which Schützenberger's method can be extended.

Finally, considering the language $\#P_1$-complete mentioned in Proposition 5.7, we note that, although its counting function is "difficult", it is possible to prove that the computation of approximate values is easy. Hence, a natural question is whether there exist context-free languages whose counting functions cannot be easily approximated.

## References

[1] G. Baron and W. Kuich, The characterization of nonexpansive grammars by rational power series, *Inform. and Control* **48** (1981) 109–118.

[2] P.W. Beame, S.A. Cook and H.J. Hoover, Log depth circuits for division and related problems, *SIAM J. Comput.* **15** (1986) 994–1003.

[3] S.J. Berkowitz, On computing the determinant in small parallel time using a small number of processors, *Inform. Process. Lett.* **18** (1984) 147–150.

[4] A. Bertoni, G. Mauri and N. Sabadini, A characterization of the class of functions computable in polynomial time by Random Access Machine, in: *Proc. 13th ACM Symp. on Theory of Computing* (1981) 168–176.

[5] A. Borodin, On relating time and space to size and depth, *SIAM J. Comput.* **6** (1977) 733–744.

[6] K. Chandrasekharan, *Introduction to Analytic Number Theory* (Springer, Berlin, 1968).

[7] N. Chomsky and M.P. Schützenberger, The algebraic theory of context free languages, in: *Comp. Prog. and Formal Systems* (North-Holland, Amsterdam, 1963), 118–161.

[8] L. Comtet, Calcul pratique des coefficients de Taylor d'une fonction algébrique, *Enseign. Math.* **10** (1964) 267–270.

[9] S.A. Cook, Towards a complexity theory of synchronous parallel computation, *Enseign. Math.* **27** (1981) 99–124.

[10] S.A. Cook, A taxonomy of problems with fast parallel algorithms, *Inform. and Control* **64** (1985) 2–22.

[11] R. Cori and J. Richard, Enumération des graphes planaires à l'aide des séries formelles en variables non commutatives, *Discrete Math.* **2** (1971) 115–162.

[12] R. Cori and B. Vauquelin, Planar maps are well labeled trees, *Canad. J. Math.* **33** (1981) 1023–1042.

[13] M.P. Delest and G. Viennot, Algebraic languages and polyominoes enumeration, *Theoret. Comput. Sci.* **34** (1984) 169–206.

[14] P. Flajolet, Analytic models and ambiguity of context-free languages, *Theoret. Comput. Sci.* **49** (1987) 283–309.

[15] A.V. Goldberg and M. Sipser, Compression and ranking, in: *Proc. 7th ACM Symp. on Theory of Computing* (1985) 440–448.

[16] J.R. Goldman, Formal languages and enumeration, *J. Combin. Theory Ser. A* **24** (1978) 318–338.

[17] M. Goldwurm, Formal series methods in algorithm analysis for problems on languages, Doctoral Thesis, Dip. di Scienze dell'Informazione, Università di Milano, May 1988.

[18] J. Hartmanis, Context-free languages and Turing machine computations, *Proc. Sympos. Appl. Math.* **19** (1967) 42–51.

[19] J. Hartmanis, N. Immerman and V. Sewelson, Sparse sets in NP – P: EXPTIME versus NEXPTIME, *Inform. and Control* **65** (1985) 158–181.

[20] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation* (Addison-Wesley, Reading, MA, 1979).

[21] W. Kuich, Languages and enumeration of planted plane trees, *Indag. Math.* **32** (1970) 268–280.

[22] W. Kuich, A context-free language and enumeration problems in infinite trees and digraphs, *J. Combin. Theory* **10** (1971) 135–142.

[23] P. McKenzie and S.A. Cook, The parallel complexity of Abelian permutation group problems, Tech. Rep. 181/85, Dept. of Computer Science, Univ. of Toronto, 1985.

[24] A.R. Meyer and L.J. Stockmeyer, Word problems requiring exponential time, in: *Proc. 5th ACM Symp. on Theory of Computing* (1973) 1–9.

[25] N. Pippenger, On simultaneous resource bounds, in: *Proc. 20th IEEE Symp. on Foundation of Computer Science* (1979), 307–311.

[26] J.H. Reif, Logarithmic depth circuits for algebraic functions, *SIAM J. Comput.* **15** (1986) 231–242.

[27] W.L. Ruzzo, On uniform circuit complexity, *J. Comput. System Sci.* **22** (1981) 365–383.

[28] A. Salomaa and M. Soittola, *Automata Theoretic Aspects of Formal Power Series* (Springer, New York, 1978).

[29] A. Schönhage and V. Strassen, Schnelle Multiplikation grosser Zahlen, *Computing* **7** (1971) 281–292.

[30] M.P. Schützenberger, Certain elementary families of automata, in: *Proc. Symp. on Mathematical Theory of Automata* (1962) 139–153.

[31] M.P. Schützenberger, Context-free languages and pushdown automata, *Inform. and Control* **6** (1963) 246–264.

[32] L.J. Stockmeyer, The complexity of decision problems in automata theory and logic, Ph.D. Thesis, MIT Project MAC TR-133, 1974.

[33] L.G. Valiant, The complexity of computing the permanent, *Theoret. Comput. Sci.* **8** (1979) 189–202.

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

[35] K. Wagner and G. Wechsung, *Computational Complexity* (VEB Deutscher Verlag der Wissenschaften, Berlin, 1986).

[36] I. Wegener, *The Complexity of Boolean Functions* (Wiley, New York and Teubner, Stuttgart, 1987).