Semantical Counting Circuits *

Fabrice Noilhan† Miklos Santha‡

Abstract

Counting functions can be defined syntactically or semantically depending on whether they count the number of witnesses in a non-deterministic or in a deterministic computation on the input. In the Turing machine based model, these two ways of defining counting were proven to be equivalent for many important complexity classes. In the circuit based model, it was done for #P , but for low-level complexity classes such as $\#AC^0$ and $\#NC^1$ only the syntactical definitions were considered. We give appropriate semantical definitions for these two classes and prove them to be equivalent to the syntactical ones. We also consider semantically defined probabilistic complexity classes corresponding to AC^0 and NC^1 and prove that in the case of unbounded error, they are identical to their syntactical counterparts.

1 Introduction

Counting is one of the basic questions considered in complexity theory. It is a natural generalization of non-determinism: computing the number of solutions for a problem is certainly not easier than just deciding if there is a solution at all. Counting has been extensively investigated both in the machine based and in the circuit based models of computation.

Historically, the first counting classes were defined in Turing machine based complexity theory. Let us call a non-deterministic Turing machine an NP-machine if it works in polynomial time, and an NL-machine if it works in logarithmic space. In the case of a non-deterministic machine, an accepting path in its computation tree on a string x certifies that x is accepted. We will call such a path a *witness* for x. The very first, and still the most famous, counting class called #P was introduced by Valiant [Val79] as the set of counting functions that map a string x to the number of witnesses for x of some NP-machine. An analogous definition was later made by Alvarez and Jenner [AJ93] for the class #L: it contains the set of counting functions that map x to the number of witnesses for x of some NL-machine. These classes contain several natural complete problems: for example computing the permanent of a matrix is complete in #P, whereas computing the number of paths in a directed graph between two specified vertices is complete in #L.

The so-called gap classes were defined subsequently to include functions taking also negative values into the above model. GapP was introduced by Fenner, Fortnow and Kurtz [FFK94] as the difference of two functions in #P. The analogous definition for GapL was made independently by Vinay [Vin91], Toda [Tod91], Damm [Dam91] and Valiant [Val92]. This later class has received considerable attention, mostly because it characterizes the complexity of computing the determinant of a matrix [AO96, ST98, MV97].

Still in the Turing machine model, there is an alternative way of defining the classes #P and #L, based on the computation of deterministic machines. In the following discussion let us consider deterministic Turing machines acting on pairs of strings (x, y) where for some polynomial $p(n)$, the length of y is $p(|x|)$. In that setting we will say that the string y is a *witness* for x when the machine accepts (x, y). We will call a deterministic Turing machine a P-machine if it works in polynomial time, and an L-machine if it works in logarithmic space and it has only one-way access to y. Then #P (respectively #L) can be defined as the set of functions f for which there exists a P-machine (respectively L-machine) such that $f(x)$ is the number of witnesses for x. The equivalence between these definitions can be established if we interpret the above

*This research was supported by the EC thematic network RAND-APX IST-1999-14036 and by the CNRS/STIC grants No. 01N80/0502 and 01N80/0607. Part of the work of the second author was done while visiting MSRI, Berkeley.

†Université Paris-Sud, LRI, Bât. 490, 91405 Orsay, France. Fabrice.Noilhan@lri.fr

‡CNRS, UMR 8623, Université Paris-Sud, LRI, Bât. 490, 91405 Orsay, France. Miklos.Santha@lri.fr

deterministic Turing machines as a normal form, with simple witness structure, for the corresponding non-deterministic machines, where the string y describes the sequence of choices made during the computation on x. Nonetheless this latter way of looking at counting has at least two advantages over the previous one.

The first advantage is that this definition is more robust in the following sense. Two non-deterministic machines, even if they compute the same relation $R(x)$, might define different counting functions depending on their syntactical properties. On the other hand, if the definition is based on deterministic machines, only the relation they compute is playing a role. Indeed, two deterministic machines computing the same relation $R(x, y)$ will necessarily define the same counting function independently from the syntactical properties of their computation. Therefore, from now on, we will refer to the non-deterministic machine based definition of counting as *syntactical*, and to the deterministic machine based definition as *semantical*.

The second advantage of the semantical definition of counting is that probabilistic complexity classes can be defined more naturally in that setting. For example PP(respectively PL) is just the set of languages for which there exists a P-machine (respectively L-machine) such that a string x is in the language exactly when there are more witnesses for x than non-witnesses. In the case of the syntactical definition, these probabilistic classes are usually defined via probabilistic Turing machines.

The above duality in the definition of counting exists of course in other models where determinism and non-determinism are meaningful concepts. This is the case of the circuit based model of computation. Still, in this model syntactical counting has received considerably more attention than semantical counting. Before we discuss the reason for that, let us make clear what do we mean here by these notions.

The syntactical notion of a witness for a string x in a circuit family was defined by Venkateswaran [Ven92] as an accepting subtree of the corresponding circuit on x, which is a smallest sub-circuit certifying that the circuit's output is 1 on x. It is easy to show that the number of such witnesses is equal to the value of the arithmetized version of the circuit on x. Let us stress again that this number, and therefore the counting function defined by a circuit, depends heavily on the specific structure of the circuit and not only on the function computed by it. For example if we consider circuit C_1 which is just the variable x, and circuit C_2 which consists of an OR gate whose both inputs are the same variable x, then clearly these two circuits compute the same function. On the other hand, on input $x = 1$, the counting function defined by C_1 will take the value 1, whereas the counting function defined by the circuit C_2 will take the value 2.

For the semantical notion of a witness we consider again families whose inputs are pairs of strings of polynomially related lengths. As in the case of Turing machines, y is a witness for x if the corresponding circuit outputs 1 on (x, y).

Venkateswaran was able to give a characterization of #P and #L in the circuit model based on the syntactical definition of counting. His results rely on a circuit based characterization of NP and NL. He has shown that #P is equal to the set of counting functions computed by uniform semi-unbounded circuits of exponential size and of polynomial algebraic degree; and #L is equal to the set of counting functions computed by uniform skew-symmetric circuits of polynomial size. Semantically #P can be characterized as the set of counting functions computed by uniform polynomial size circuits.

In recent years several low level counting classes were defined in the circuit based model, all in the syntactical setting. Caussinus et al.[CMTV98] have defined $\#\mathrm{NC}^1$, and Agrawal et al. [AAD00] have defined $\#\mathrm{AC}^0$ as the set of functions counting the number of accepting subtrees in the respective circuit families. In subsequent works, many important properties of these classes were established [ABL98, AAB$^+$99]. Although some attempts were made [Yam96], no satisfactory characterization of these classes was obtained in the semantical setting. The main reason for that is that by simply adding "counting" bits to AC^0 or NC^1 circuits, we fall to the all too powerful counting class #P [SST95, VW96], and it is not at all clear what type of restrictions should be made in order to obtain $\#\mathrm{AC}^0$ and $\#\mathrm{NC}^1$.

The main result of this paper is such a semantical characterization of these two counting classes. Indeed, we will define semantically the classes $\#\mathrm{AC}^0_{\mathrm{CO}}$ and $\#\mathrm{NC}^1_{\mathrm{CO}}$ by putting some relatively simple restrictions on the structure of AC^0 and NC^1 circuits, and by restricting the way they access counting variables. Our main result is that this definition is equivalent to the syntactical definition; that is, we have

Theorem 1 $\#\mathrm{AC}^0 = \#\mathrm{AC}^0_{\mathrm{CO}}$ *and* $\#\mathrm{NC}^1 = \#\mathrm{NC}^1_{\mathrm{CO}}$.

Put it another way, if standard AC^0 and NC^1 are seen as "non-deterministic" circuit families in the syntactical definition of the corresponding counting classes, we are able to characterize their "deterministic" counterparts which define the same counting classes semantically.

Semantically defined counting classes give rise naturally to the corresponding probabilistic classes in the three usually considered cases: in the unbounded, in the bounded and in the one sided error model. Indeed, we will define the probabilistic classes $\mathrm{PAC}^0_{\mathrm{CO}}$, $\mathrm{PNC}^1_{\mathrm{CO}}$, $\mathrm{BPAC}^0_{\mathrm{CO}}$, $\mathrm{BPNC}^1_{\mathrm{CO}}$, $\mathrm{RAC}^0_{\mathrm{CO}}$ and $\mathrm{RNC}^1_{\mathrm{CO}}$. PAC^0 and PNC^1 were already defined syntactically via $\#\mathrm{AC}^0$ and $\#\mathrm{NC}^1$, and we will prove for this model that our definitions coincide with previous ones:

Theorem 2 $\mathrm{PAC}^0_{\mathrm{CO}} = \mathrm{PAC}^0$ *and* $\mathrm{PNC}^1_{\mathrm{CO}} = \mathrm{PNC}^1$.

Other authors have previously studied bounded-error and one-sided error probabilistic circuits [Joh90]. The definitions that were used in that work are also semantical, but they differ from our definitions in that they impose no restrictions on the way circuits access counting variables. Those classes seem more closely related to BPP; it is not known if those circuits can be simulated in deterministic polynomial time. In contrast, the classes we consider in Theorem 2 define small subclasses of P. Still, it remains an open question if those classes coincide with ours, and we think that this question is worthy of further investigations.

The paper is organized as follows: Section 2 contains the definitions for semantical circuit based counting. Section 3 exhibits the mutual simulations of syntactical and semantical counting for the circuit classes AC^0 and NC^1. Theorem 1 is a direct consequence of Theorems 4 and 5 proven here. Finally in section 4 we discuss the gap and random classes which are derived from semantical counting circuits. Theorem 6 relating gap classes and counting circuits will imply Theorem 2.

2 Definitions

In this chapter we define *counting circuit families* which will be used for the semantical definition of a counting function. Counting circuits have two types of input variables: standard and counting ones. They are in fact restricted Boolean circuits, where the restriction is put on the way the gates and the counting variables can be used in the circuits. First we will define the usual Boolean circuit families and the way they are used to define (syntactically) counting functions, and then we do the same for counting circuit families. The names "circuit" versus "counting circuit" will be used systematically this way in the rest of the paper.

A *bounded fan-in circuit* with n input variables is a directed acyclic graph with vertices of in-degree 0 or 2. The vertices of in-degree 0 are called inputs, and they are labeled with an element of the set $\{0, 1, x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$. The vertices of in-degree 2 are labeled with a bounded AND or OR gate. There is a distinguished vertex of outdegree 0, this is the output of the circuit. An *unbounded fan-in circuit* is defined similarly with the only difference that non input vertices can have arbitrary in-degree, and they are labeled with unbounded AND or OR gates. A *circuit family* is a sequence $(C_n)_{n=1}^{\infty}$ of circuits where C_n has n input variables. It is *uniform* if its extended connection language [Vol99] is computed in DLOGTIME. An NC^1 circuit family is a uniform, bounded fan-in circuit family of polynomial size and logarithmic depth. An AC^0 circuit family is a uniform, unbounded fan-in circuit family of polynomial size and constant depth. In fact, in the case of AC^0 circuits an equivalent condition of uniformity is to require that the direct connection language be computable in DLOGTIME [Vol99, Theorem 4.31].

A circuit C is a *tree circuit* if all its vertices have out-degree 1. A *proof tree* in C on input x is a connected subtree which contains its output, has one edge into each OR gate, has all the edges into the AND gates, and which evaluates to 1 on x. The number of proof trees in C on x will be denoted by $\#\mathrm{PT}_C(x)$. A Boolean tree circuit family $(C_n)_{n=1}^{\infty}$ computes a function $f : \{0,1\}^* \to \mathbb{N}$ if for every x, we have $f(x) = \#\mathrm{PT}_{C_{|x|}}(x)$. We denote by $\#\mathrm{AC}^0$ (respectively by $\#\mathrm{NC}^1$) the class of functions computed by a uniform AC^0 (respectively NC^1) tree circuit family.

In order to introduce counting variables into counting circuits and to carry out the syntactical restrictions, we will use a new type of gate, which we call a SELECT gate. This is actually a small circuit which is built in some specific way from AND and OR gates. The SELECT gates which use a counting variable to choose a branch of the circuit will actually replace OR gates which will be prohibited in their general form.

We now define formally the new gate. In the following we will denote single Boolean variables with a subscript such as v_0. Boolean vector variables will be denoted without a subscript, such as v. We will also identify an integer $0 \leq s \leq 2^k - 1$ with its binary representation $(s_0, \ldots, s_{k-1})$.

The bounded fan-in SELECT gate will have 3 arguments. It is defined by $\mathrm{SELECT}^1(x_0, x_1, u_*) = x_{u_*}$, and represented by $\mathrm{OR}(\mathrm{AND}(x_0, \overline{u_*}), \mathrm{AND}(x_1, u_*))$. For every k, the unbounded fan-in SELECT^k gate has

$2^k + k$ arguments and is defined by $\mathrm{SELECT}^k(x_0, \ldots, x_{2^k-1}, u_0, \ldots, u_{k-1}) = x_u$. This gate is represented by the circuit $\mathrm{OR}_{i=0}^{2^k-1}(\mathrm{AND}(x_i, u = i))$ where $u = i$ stands for the circuit $\mathrm{AND}_{j=0}^{k-1}(\mathrm{OR}(\mathrm{AND}(\overline{u_j}, \overline{i_j}), \mathrm{AND}(u_j, i_j)))$. The last gate can easily be extended to $m+k$ arguments for $m < 2^k$ as $\mathrm{SELECT}^k(x_0, \ldots, x_{m-1}, u_0, \ldots, u_{k-1}) = \mathrm{SELECT}^k(x_0, \ldots, x_m, 0, \ldots, 0, u_0, \ldots, u_{k-1})$. Clearly, SELECT^k can be simulated by a circuit of depth $O(k)$ containing only SELECT^1 gates.

We will define recursively unbounded fan-in counting circuits. There will be two types of input variables: "standard" and "counting" ones.

Definition 1 (Counting circuit)

- *If C is a Boolean tree circuit, then C is a counting circuit. All its variables are standard.*
- *If $C_0, \ldots, C_{2^k-1}$ are counting circuits and $u_0, \ldots, u_{k-1}$ are input variables which are not appearing in them, then $\mathrm{SELECT}(C_0, \ldots, C_{2^k-1}, u_0, \ldots, u_{k-1})$ is a counting circuit. The variables $u_0, \ldots, u_{k-1}$ are counting variables.*
- *If $C_0, \ldots, C_k$ are counting circuits and they do not have any common counting variables, then $\mathrm{AND}(C_0, \ldots, C_k)$ is a counting circuit.*

Moreover, we require that no input variable can be counting and standard at the same time.

Bounded counting circuits are defined analogously, with $k = 1$ in all the construction steps.

The set of all standard (respectively counting) variables of a circuit C will be denoted $\mathsf{SV}(C)$ (respectively $\mathsf{CV}(C)$. Let C be a counting circuit with n standard variables. The *counting function* $\#\mathrm{CO}_C : \{0,1\}^n \mapsto \mathbb{N}$ associated with C is defined as:

$$\#\mathrm{CO}_C(x) = \begin{cases} C(x) & \text{if } \mathsf{CV}(C) = \emptyset, \\ \#\{u \mid C(x,u) = 1\} & \text{if } \mathsf{CV}(C) \neq \emptyset. \end{cases}$$

A sequence $(C_n)_{n=1}^{\infty}$ of counting circuits is a *counting family* if there exists a polynomial p such that for all n, C_n has n standard variables and at most $p(n)$ counting variables. A family is *uniform* if its extended connection language is computed in DLOGTIME. The *counting function* computed by a circuit family is defined as $\#\mathrm{CO}_{C_{|x|}}(x)$. Finally, the *semantical counting classes* are defined as follows: $\#\mathrm{AC}^0_{\mathrm{CO}}$ (respectively $\#\mathrm{NC}^1_{\mathrm{CO}}$) is the set of functions computed by a uniform AC^0 (NC^1) family of counting circuits. Again, for the uniformity of $\#\mathrm{AC}^0_{\mathrm{CO}}$ we could have asked equivalently that the direct connection language be computable in DLOGTIME.

Before addressing the power of counting circuits we would like first to point out that they can be extended to contain counting variables and standard OR and negation gates without increasing their power. Since this additional possibility will be helpful in our constructions, we will make here this statement precise.

An unbounded fan-in *extended counting circuit* is a counting circuit with the following additional construction steps to the Definition 1.

- If u is a counting variable then u and $\overline{u}$ are extended counting circuits.
- If $C_0, \ldots, C_k$ are extended counting circuits and they do not have any common counting variable then $\mathrm{OR}(C_0, \ldots, C_k)$ is an extended counting circuit.
- If C is an extended counting circuit, $\overline{C}$ is an extended counting circuit.

We obtain the definition for the bounded fan-in case by taking again $k = 1$. We will denote by $\#\mathrm{AC}^0_{\mathrm{ECO}}$ (respectively $\#\mathrm{NC}^1_{\mathrm{ECO}}$) the set of functions computed by a uniform AC^0 (respectively NC^1) family of extended counting circuits. The following theorem shows that extended counting circuits are no more powerful than regular ones.

Theorem 3 *$\#\mathrm{AC}^0_{\mathrm{ECO}} = \#\mathrm{AC}^0_{\mathrm{CO}}$ and $\#\mathrm{NC}^1_{\mathrm{ECO}} = \#\mathrm{AC}^0_{\mathrm{CO}}$.*

Proof: Let C be an extended counting circuit. We will show that there exists a counting circuit computing $\#\mathrm{CO}_C$ whose size and depth is of the same order of magnitude. First observe that if u is a counting variable then $u = \mathrm{SELECT}(u, 0, 1)$ and $\overline{u} = \mathrm{SELECT}(u, 1, 0)$. Then one can get rid of the OR gates by recursively replacing $\mathrm{OR}(C_0, \ldots, C_{2^k-1})$ with $\mathrm{SELECT}(C_0, \ldots, C_{2^k-1}, u_0, \ldots, u_{k-1})$ where $\{u_0, \ldots, u_{k-1}\} \cap (\mathsf{CV}(C_0) \cup \ldots \cup \mathsf{CV}(C_{2^k-1})) = \emptyset$. Since $C_0, \ldots, C_{2^k-1}$ do not have any common counting variables, this does not change the counting function computed by the counting circuit. Finally observe that negation gates in C can be pushed down to the literals. For this, besides the standard de Morgan laws, one can use the following equality whose verification is straightforward:

$$\overline{\mathrm{SELECT}(C_0, \ldots, C_{2^k-1}, u_0, \ldots, u_{k-1})} = \mathrm{SELECT}(\overline{C_0}, \ldots, \overline{C_{2^k-1}}, u_0, \ldots, u_{k-1}).$$

Since all these transformations may increase the size or the depth only by a constant factor, the statement follows. □

3 Circuits and counting circuits

3.1 Simulating circuits by counting circuits

We will use a step-by-step simulation. We will define a function ϕ which maps circuits into counting circuits by structural recursion on the output gate G of the circuit. The definition will be done for the unbounded case from which the bounded case can be obtained by replacing the parameter k with 1 in all the construction steps, and unbounded gates by bounded ones.

Definition 2 (the ϕ function) *If G is a literal, then $\phi(G) = G$ and the corresponding variable is standard. If G is an* AND *gate whose entries are the circuits $C_0, \ldots, C_k$, then let the circuits C_i' be obtained from $\phi(C_i)$ by renaming counting variables so that $\forall i \neq j$, $\mathsf{CV}(C_i') \cap \mathsf{CV}(C_j') = \mathsf{CV}(C_i') \cap \mathsf{SV}(C_j') = \emptyset$. Then $\phi(C) = \mathrm{AND}(C_0', \ldots, C_k')$. If G is an* OR *gate whose entries are the circuits $C_0, \ldots, C_{2^k-1}$, then the circuits C_i' are obtained from $\phi(C_i)$ by renaming counting variables so that $\forall i \neq j$, $\mathsf{CV}(C_i') \cap \mathsf{CV}(C_j') = \mathsf{SV}(C_i') \cap \mathsf{SV}(C_j') = \emptyset$. Let $\mathcal{V} = \mathsf{CV}(C_0') \cup \ldots \cup \mathsf{CV}(C_{2^k-1}')$ and $\mathcal{V}_i = \mathcal{V} - \mathsf{CV}(C_i')$. Let C_i'' be defined as $\mathrm{AND}(C_i', \mathcal{V}_i)$, and let $u_0, \ldots, u_{k-1}$ be counting variables such that $\{u_0, \ldots, u_{k-1}\} \cap \mathcal{V} = \emptyset$. Then $\phi(C) = \mathrm{SELECT}(C_0'', \ldots, C_{2^k-1}'', u_0, \ldots, u_{k-1})$.*

The next two lemmas will prove that the definition of ϕ is correct and that the functions computed by the corresponding circuit families are equal.

Lemma 1 *If (C_n) is a uniform AC^0 (respectively NC^1) family of circuits, then $(\phi(C_n))$ is a uniform AC^0 (resp. NC^1) family of counting circuits.*

Proof: Throughout the construction, we assured that the entry circuits of an AND gate do not have common counting variables. Clearly, no input variable can be counting and standard at the same time. Since there are a polynomial number of gates and for each gate, we introduced a polynomial number of counting variables, the number of counting variables is bounded by a polynomial.

The uniformity of $(\phi(C_n))$ in the unbounded case follows easily from the uniformity of (C_n) since we can work with the uniformity condition requiring that the direct connection language be in DLOGTIME. In the bounded case, let us suppose that we have an admissible encoding scheme for (C_n) such that the extended connection language of (C_n) with respect to the encoding is in DLOGTIME. To create an admissible encoding scheme for $(\phi(C_n))$ with a similar property, we choose gate numbers for the counting variables so that there exists a DLOGTIME computable bijection between them and the numbers of the OR gates in (C_n). Whenever a new counting variable has to be introduced at a SELECT gate in $(\phi(C_n))$ we will choose the one which is in bijection with corresponding OR gate in (C_n). This ensures that no renaming of counting variables is ever necessary. Let us suppose now that the entries of an OR gate are the circuits C_0 and C_1. Then for $i = 0, 1$, in the circuit C_i'' we will create a subcircuit structurally similar to C_{1-i} for computing the conjunction of the counting variables of C_{1-i}. We substitute the constant 1 for the standard variables,

and recursively replace the OR gates by AND gates of fan-in three whose third entry is the corresponding counting variable. The uniformity of $(\phi(C_n))$ now follows from the uniformity of (C_n).

To finish the proof, we should consider the depth of the counting circuits. In the unbounded case, $(\phi(C_n))$ is of constant depth since the SELECT gates which replace the OR gates of the original circuit are of constant depth. In the bounded case, let k be such that there are at most n^k variables in C_n. The depth of C_n is $O(\log n)$. Let us define $d_i = \max\{\text{depth}(\phi(D))\}$ where D is a subcircuit of C_n of depth i. Then we have

$$d_{i+1} \leq 2 + \max(d_i, \lceil k \cdot \log n \rceil)$$

since the depth increases only when the output gate is an OR. Therefore, $(\phi(C_n))$ is of logarithmic depth. □

Lemma 2 *For every circuit C, $\#\mathrm{PT}_C(x) = \#\mathrm{CO}_{\phi(C)}(x)$.*

Proof: We will prove this by structural recursion on the output gate G of C. If G is a literal, then by definition, circuits and counting circuits define the same counting function. If G is an AND gate then since for $i = 0, \ldots, k$ the variables in $\mathsf{CV}(C'_i)$ are distinct, $\#\mathrm{CO}_{\phi(C)}(x) = \prod \#\mathrm{CO}_{C'_i}(x)$, which is the same as $\prod \#\mathrm{CO}_{\phi(C_i)}(x)$ because C'_i was obtained from $\phi(C_i)$ by renaming the variables. By the inductive hypothesis and the definition of the proof tree model, this is equal to $\#\mathrm{PT}_C(x)$. If G is an OR gate then since the counting variables $u_0, \ldots, u_{k-1}$ are distinct from the counting variables of the subcircuits, $\#\mathrm{CO}_{\phi(C)}(x) = \sum \#\mathrm{CO}_{C''_i}(x)$. For every i, $\#\mathrm{CO}_{C''_i}(x) = \#\mathrm{CO}_{C'_i}(x)$ since the AND gate fixes all the counting variables outside $\mathcal{V}_i$. This is the same value as $\#\mathrm{CO}_{\phi(C_i)}(x)$ since C'_i was obtained from $\phi(C_i)$ by renaming the variables. The statement follows from the inductive hypothesis. □

The two lemmas imply

Theorem 4 *$\#\mathrm{AC}^0 \subseteq \#\mathrm{AC}^0_{\mathrm{CO}}$ and $\#\mathrm{NC}^1 \subseteq \#\mathrm{NC}^1_{\mathrm{CO}}$.*

3.2 Simulating counting circuits by circuits

We will use in the construction circuits computing fixed integers which are powers of 2. For $l \geq 0$ the circuit A_{2^l} computing the integer 2^l is defined as follows. A_1 is the constant 1 and A_2 is $\mathrm{OR}(1,1)$. For $l \geq 2$, in the unbounded case, A_{2^l} has a topmost unbounded AND gate with l subcircuits A_2. In the bounded case, we replace the unbounded AND gate by its standard bounded simulation consisting of a balanced binary tree of bounded AND gates. Clearly, the depth of A_{2^l} in the bounded case is $\lceil \log l \rceil + 1$.

We now define a function ψ which maps counting circuits into circuits by structural recursion on the output gate G of the counting circuit. Again, the definition will be done for the unbounded case, from which the bounded case can be obtained by replacing the parameter k with 1.

Definition 3 (the ψ function) *If G is a literal, then $\psi(G) = G$. If G is an* AND *gate whose entries are $C_0, \ldots, C_k$, then $\psi(C) = \mathrm{AND}(\psi(C_0), \ldots, \psi(C_k))$. If G is a* SELECT *gate whose entries are $C_0, \ldots, C_{2^k-1}, u_0, \ldots, u_{k-1}$ then set $\mathcal{V} = \mathsf{CV}(C_0) \cup \ldots \cup \mathsf{CV}(C_{2^k-1})$ and $\mathcal{V}_i = \mathcal{V} - \mathsf{CV}(C_i)$. We let $C'_i = \mathrm{AND}(\psi(C_i), A_{2^{|\mathcal{V}_i|}})$ and $\psi(C) = \mathrm{OR}(C'_0, \ldots, C'_{2^k-1})$.*

Again, we proceed with two lemmas to prove the correctness of the simulation.

Lemma 3 *If (C_n) is a uniform AC^0 (respectively NC^1) family of counting circuits, then $(\psi(C_n))$ is a uniform AC^0 (resp. NC^1) family of circuits.*

Proof: In the construction, we get rid of the SELECT gates and of the counting variables. We do not modify the number of standard variables. The only step where we increase the size or the depth of the circuit is when SELECT gates are replaced. Each replacement introduces at most a polynomial number of gates. Therefore the size remains polynomial. The uniformity of $(\psi(C_n))$ follows from the uniformity of (C_n) using the ideas of Lemma 1 and the fact that circuits A_{2^l} are uniform.

In the unbounded case, the depth remains constant since the circuits A_{2^l} have constant depth. In the bounded case, we claim that every replacement of a SELECT gate increases the depth by a constant. This follows from the fact that $\text{depth}(A_{2^{|\mathcal{V}_i|}}) \leq \text{depth}(C_{1-i}) + 1$ for $i = 0, 1$ since a bounded counting circuit with m counting variables has depth at least $\lceil \log(m+1) \rceil$. Therefore the whole circuit remains of logarithmic depth. □

Lemma 4 *For every counting circuit C, $\#\mathrm{PT}_{\psi(C)}(x) = \#\mathrm{CO}_C(x)$.*

Proof: We will prove this by structural recursion on the output gate G of the counting circuit. In the proof, we will use the notation of definition 3. If G is a literal, then by definition, C and $\psi(C)$ define the same counting function. If G is an AND gate then by definition $\#\mathrm{PT}_{\psi(C)}(x) = \prod \#\mathrm{PT}_{\psi(C_i)}(x)$. Since for $i = 0, \ldots, k$ the subcircuits C_i do not share common counting variables, using the inductive hypothesis this is equal to $\#\mathrm{CO}_C(x)$. If G is a SELECT gate then $\#\mathrm{PT}_{\psi(C)}(x) = \sum 2^{|\mathcal{V}_i|} \cdot \#\mathrm{PT}_{\psi(C_i)}(x)$. Also, $\#\mathrm{CO}_C(x) = \sum 2^{|\mathcal{V}_i|} \cdot \#\mathrm{CO}_{C_i}(x)$ since the value of variables in $\mathcal{V}_i$ do not influence the value of the circuit C_i. The result follows from the inductive hypothesis. □

Theorem 5 $\#\mathrm{AC}^0_{\mathrm{CO}} \subseteq \#\mathrm{AC}^0$ *and* $\#\mathrm{NC}^1_{\mathrm{CO}} \subseteq \#\mathrm{NC}^1$.

4 Gap and random classes via semantical counting

In this section, we will point out another similarity between semantical counting circuits and deterministic Turing machine based counting: we will define probabilistic classes by counting the fraction of assignments for the counting variables which make the circuit accept. We will prove that in the unbounded error case, our definitions coincide with the syntactical definition via gap classes. In the bounded error and one sided error models we could not determine if our definitions and the syntactical ones are identical.

4.1 Gap classes

As usual, for any counting class #C, we define the associated gap class GapC. A function f is in GapC iff there are two functions f_1 and f_2 in #C such that $f = f_1 - f_2$. The following theorem will be useful for discussing probabilistic complexity classes.

Theorem 6 *Let f be a function in* GapAC^0 *(respectively* GapNC^1*) . Then there is an* AC^0 *(resp.* NC^1*) uniform family of extended counting circuits (C_n) such that $2 \cdot f(x) = \#\mathrm{CO}_{C_{|x|}}(x) - \#\mathrm{CO}_{\overline{C}_{|x|}}(x)$.*

Proof: Let #C be one of the classes $\#\mathrm{AC}^0$ or $\#\mathrm{NC}^1$, and let f be a function in GapC. Then there exist two functions in #C, f_1 and f_2 such that $f = f_1 - f_2$. Fix an entry x of length n. Let us take two uniform #C families of counting circuits which compute f_1 and f_2, and let respectively D_1 and D_2 be the counting circuits in these families which have n input variables.

Let $\mathcal{V}_1 = \mathsf{CV}(D_1)$, $\mathcal{V}_2 = \mathsf{CV}(D_2)$ and $m =\mid \mathcal{V}_1 \mid + \mid \mathcal{V}_2 \mid$. We will suppose without loss of generality that $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$ (by renaming the variables if necessary). We define $D'_1 = \mathrm{AND}(D_1, \mathcal{V}_2)$ and $D'_2 = \mathrm{AND}(D_2, \mathcal{V}_1)$. Let t be a counting variable such that $t \notin \mathcal{V}_1 \cup \mathcal{V}_2$ and define $C_n = \mathrm{SELECT}(D'_1, \overline{D'_2}, t)$. The counting circuit family (C_n) is uniform and is in C since its depth and size are changed only up to a constant with respect to the families computing f_1 and f_2.

We first claim that the counting function associated with C_n, on entry x, computes $f_1(x) + (2^m - f_2(x))$. First, let us observe that $\#\mathrm{CO}_{\overline{D'_2}}(x) = 2^m - \#\mathrm{CO}_{D'_2}(x)$. Therefore, since t does not appear in D'_1 and D'_2, we have $\#\mathrm{CO}_{C_n}(x) = \#\mathrm{CO}_{D'_1}(x) + (2^m - \#\mathrm{CO}_{D'_2}(x))$. By the definition of the AND gate, the claim follows.

The number of variables in C_n is $m + 1$. Therefore, $\#\mathrm{CO}_{C_n}(x) - \#\mathrm{CO}_{\overline{C_n}}(x) = 2 \cdot \#\mathrm{CO}_{C_n}(x) - 2^{m+1}$. By the previous claim, this is $2f(x)$. □

4.2 Randomized classes via semantical counting

Another advantage of semantical counting circuits is that probabilistic complexity classes can easily be defined in this model. The definition is analogous to the definition of probabilistic classes based on Turing machines' computation: for a given input, we will count the fraction of all settings for the counting variables which make the circuit accept. We will define now the usual types of probabilistic counting classes in our model and compare them to the existing definitions. For a counting circuit C we define $\mathrm{Pr}_{\mathrm{CO}}(C(x))$ by:

$$\mathrm{Pr}_{\mathrm{CO}}(C(x)) = \begin{cases} C(x) & \text{if } \mathsf{CV}(C) = \emptyset, \\ \#\{v \mid C(x, v) = 1\}/2^{|\mathsf{CV}(C)|} & \text{if } \mathsf{CV}(C) \neq \emptyset. \end{cases}$$

Let now C be one of the classes AC^0 or NC^1. Then PC_{CO} is the family of languages for which there exists a uniform C family of counting circuits (C_n) such that $x \in L$ iff $\Pr_{CO}(C_{|x|}(x)) > 1/2$. Similarly BPC_{CO} is the family of languages for which there exists a uniform C family of counting circuits (C_n) such that $x \in L$ iff $\Pr_{CO}(C_{|x|}(x)) > 1/2 + \epsilon$ for some constant $\epsilon > 0$. Finally, RC_{CO} is the family of languages for which there exists a uniform C family of counting circuits (C_n) such that if $x \in L$ then $\Pr_{CO}(C_{|x|}(x)) \geq 1/2$ and if $x \notin L$ then $\Pr_{CO}(C_{|x|}(x)) = 0$.

Let us recall that PC was defined [AAD00, CMTV98] as the family of languages L for which there exists a function f in GapC such that $x \in L$ iff $f(x) > 0$. Theorems 6 and 3 immediately imply that these definitions coincide with ours, which is exactly the statement of Theorem 2.

Bounded error and one sided error circuit based probabilistic complexity classes were defined in the literature for the classes in the AC and NC hierarchy [Weg87, Joh90, Coo85]. These are semantical definitions in our terminology, but unlike in our case, no special restriction is put on the way counting variables are introduced. To be more precise, let a *probabilistic circuit family* (C_n) be defined as a uniform family of circuits where the circuits have standard and probabilistic input variables and the number of probabilistic input variables is polynomially related to the number of input variables. For any input x, the probability that such a family accepts x is the fraction of assignments for the probabilistic variables which make the circuit $C_{|x|}$ accept x. Then the usual definition of BPC and RC is similar to that of BPC_{CO} and RC_{CO} except that probabilistic circuit families and not counting circuit families are used in the definition.

Branching problems constitute another model for defining low level counting and probabilistic complexity classes. This possibility was first explored by Caussinus et al. [CMTV98], and further studied by Agrawal and al. [AAD00]. If we denote by #BR the family of counting functions computed by DLOGTIME-uniform polynomial size and constant width counting branching programs, then their results can be summarized as $\#AC^0 \subset \#BR \subseteq \#NC^1$.

The robustness of our definitions of randomized classes is underlined by the fact that the bounded error (respectively one-sided error) probabilistic class defined via constant depth and polynomial size branching programs lies between the classes $BPAC^0_{CO}$ and $BPNC^1_{CO}$ (respectively RAC^0_{CO} and RNC^1_{CO}). This follows from the inclusions $\#AC^0_{CO} \subseteq \#BR \subseteq \#NC^1_{CO}$, and from the fact that counting branching programs are also defined semantically.

As we mentioned already, it is known [SST95, VW96] that if PAC^0 is defined via probabilistic and not counting circuit families, then it is equal to PP. Therefore, it is natural to ask what happens in the other two error models: is $BPC_{CO} = BPC$ and is $RC_{CO} = RC$? If not, then we think that since branching programs form a natural model for defining low level probabilistic complexity classes, the above result indicates that counting circuits might constitute the basis of the "right" definition.

5 Acknowledgments

We would like to thank the anonymous referees for several remarks and corrections which greatly improved the presentation of the paper.

References

[AAB+99] E. Allender, A. Ambainis, D.M. Barrington, S. Datta, and H. LêThanh. Bounded depth arithmetic circuits: Counting and closure. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming*, pages 149–158, 1999.

[AAD00] M. Agrawal, E. Allender, and S. Datta. On TC^0, AC^0, and arithmetic circuits. *Journal of Computer and System Sciences*, 60(2):395–421, 2000.

[ABL98] A. Ambainis, D.M. Barrington, and H. LêThanh. On counting AC^0 circuits with negated constants. In *Proceedings of the 23th ACM Symposium on Mathematical Foundations of Computer Science*, pages 409–417, 1998.

[AJ93] Alvarez and Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107:3–30, 1993.

[AO96] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *RAIRO*, 30:1–21, 1996.

[Bar89] D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[CMTV98] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57:200–212, 1998.

[Coo85] S.A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.

[Dam91] C. Damm. $DET = L^{\#L}$. *Informatik-Preprint, Fachbereich Informatik der Humboldt-Universität zu Berlin 8*, 1991.

[FFK94] S. A. Fenner, L. J. Fortnow, and S. A. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.

[Joh90] D.S. Johnson. A catalog of complexity classes. *Handbook of Theoretical Computer Science*, A:67–161, 1990.

[MV97] M. Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago Journal of Theoretical Computer Science*, December 1997.

[SST95] S. Saluja, K. V. Subrahmanyam, and M. N. Thakur. Descriptive complexity of #P functions. *Journal of Computer and System Sciences*, 50(3):493–505, 1995.

[ST98] M. Santha and S. Tan. Verifying the determinant in parallel. *Computational Complexity*, 7:128–151, 1998.

[Tod91] S. Toda. Counting problems computationally equivalent to computing the determinant. *Tech. Rep. CSIM 91-07*, 1991.

[Val79] L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Val92] L.G. Valiant. Why is boolean complexity theory difficult? *London Mathematical Society Lecture Notes Series*, 16(9), 1992.

[Ven92] H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. *SIAM Journal on Computing*, 21:665–670, 1992.

[Vin91] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proceedings of the 6th Annual IEEE Conference on Structure in Complexity Theory*, pages 270–284, 1991.

[Vol99] H. Vollmer. *Introduction to Circuit Complexity*. Springer Verlag, 1999.

[VW96] H. Vollmer and K.W. Wagner. Recursion theoretic characterizations of complexity classes of counting functions. *Theoretical Computer Science*, 163(1–2):245–258, 1996.

[Weg87] I. Wegener. *The complexity of boolean functions*. Wiley - Teubner series in computer science, 1987.

[Yam96] T. Yamakami. Uniform AC^0 counting circuits. *Manuscript*, 1996.