Journal of Universal Computer Science, vol. 7, no. 9 (2001), 816-825
submitted: 13/9/01, accepted: 4/6/01, appeared: 28/9/01 © Springer Pub. Co.

Determinism, Nondeterminism, Alternation, and Counting

Sanjay Gupta
(Department of Computer Science
Virginia Polytechnic Institute and State University
sgupta@vt.edu)

Abstract: Toda proved a remarkable connection between the polynomial hierarchy and the counting classes. Tarui improved Toda's result to show the connection to a weak form of counting and provided an elegant proof. This paper shows that a key step in Tarui's proof can be done uniformly using the depth-first traversal and provides the algorithm that generalizes Toda's result to arbitrary alternating Turing machines (ATMs). Tarui's proof is carefully dissected to obtain an interesting relationship between the running time of the constructed counting machine and the different parameters of the original ATM: the number of alternation blocks, the number of nondeterministic steps, and the number of deterministic steps.

Keywords: Theory of computation, Computational complexity, alternating Turing machines, counting classes

Category: F.1

1 Introduction

Toda [7] proves a remarkable connection between the polynomial hierarchy and the counting classes: PH $\subseteq$ BP $\cdot$ PP. This result can be viewed as stating that languages accepted by an alternating Turing machine (ATM) [2] that has $O(1)$ alternation blocks can be accepted by a probabilistic counting machine. A natural question is whether this result can be extended to ATMs where the number of alternation blocks changes with the size of the input, even if it involves using stronger counting classes. An analysis of Toda's constructive proof reveals two problems. First, the proof uses induction on the number of alternation blocks, and hence, cannot be used as is when the number of alternation blocks can change with the input. Second, the running time of the counting machine is doubly exponential in the number of alternation blocks of the original ATM.

Tarui [6] improves Toda's result to show the connection to a weak form of counting, PH $\subseteq$ BP $\cdot$ SPP, and provides an elegant proof which improves the bounds on Toda's construction to exponential in the number of alternation blocks. An inspection of Tarui's proof shows that an intermediate step in the construction of the counting machine involves repeated substitutions of a polynomial with an exponential number of product terms for $O(1)$ steps, corresponding to the number of alternation blocks of the original ATM. While this step can be easily done when the number of alternation blocks is fixed, it is not clear if the step can be performed efficiently when the number of alternation blocks changes with the input, especially since random access is not available on Turing machines.

This paper makes three contributions. First, we show that the above step in Tarui's proof can be performed uniformly and efficiently using the depth-first traversal without needing the repeated substitution done by Tarui. Second, we explicitly provide the algorithm that generalizes Toda's result to arbitrary ATMs. Third, we carefully dissect Tarui's proof to show how the running time of the constructed counting machine relates to the various parameters of the original ATM: the number of alternation blocks, the number of nondeterministic bits guessed, and the bounds on the deterministic steps. This dissection reveals that the running time of the new counting machine has the following interesting form in terms of the bounds of the original ATM:

$$(\text{Total nondeterministic bits guessed})^{\text{Alternation blocks}} \cdot \text{Deterministic steps}\ .$$

The analysis of the deterministic steps of the counting machine using the sequential tapes of a Turing machine is non-trivial. The dissection also identifies an interesting hurdle to improving the running time of the counting machine from its current exponential bounds. Not only does the counting machine guess an exponential number of nondeterministic bits, it simulates the deterministic steps of the original ATM on an exponential number of computation paths. This naturally leads to the question: Are these exponential steps necessary? In [5], we construct a different counting machine that makes only a polynomial number of nondeterministic steps, but needs an exponential number of random bits, a polynomial number of independent random bits along different computation paths. Furthermore, each computation path of the new machine simulates only a constant number of deterministic steps of the original ATM. These results show that there is no a priori reason to believe that the bounds on the counting machine cannot be further improved, at least based on our current knowledge.

2 Preliminaries

We assume the standard notation used in computational complexity theory. The input is a string over some fixed alphabet Σ. The length of the input string is denoted by n. The length of a string y is denoted by $|y|$. For two n-bit binary strings $u = u_1u_2 \dots u_n$ and $v = v_1v_2 \dots v_n$, let $u \cdot v$ denote $\sum_{i=1}^{n} u_iv_i \pmod 2$. The set of integers is denoted by Z.

Our model of computation is nondeterministic Turing machine (NTM). Deterministic Turing machines are NTMs that do not make any nondeterministic guesses. Alternating Turing machines (ATMs) [2] are NTMs that make nondeterministic steps in one of the two categories: existential ($\exists$) or universal ($\forall$). Every block of consecutive nondeterministic guesses in the same category is called an alternation block. In a generic sense, counting machines are nondeterministic machines, where the acceptance or rejection depends in some way on the number of computation paths. Let a nondeterministic machine M output 1 on accepting paths and -1 on rejecting paths. For input x, let $\mathrm{AR}(M,x) \in Z$ denote the summation of outputs on all the computation paths of M. AR functions are intimately tied to the counting classes [1, 3, 4]. A language $L \in \mathrm{PP}$ if there exists a nondeterministic machine such that $x \in L \Leftrightarrow \mathrm{AR}(M,x) \geq 0$. A language $L \in \mathrm{SPP}$ if there exists a nondeterministic machine such that $x \in L \Rightarrow \mathrm{AR}(M,x) = 1$ and $x \notin L \Rightarrow \mathrm{AR}(M,x) = 0$.

A nondeterministic machine can behave probabilistically by requiring random bits as additional input. In this case, the output of the machine depends on both the input and random bits. For input x, and random bits r, let $\mathrm{AR}(M,x,r) \in Z$ denote the summation of outputs on all the computation paths of M. Toda's theorem proves that any language L accepted by an ATM that has a constant number of alternation blocks can be accepted by a NTM M probabilistically such that $x \in L \Leftrightarrow \mathrm{AR}(M,x,r) \geq 0$ with probability at least $3/4$, where r denotes a polynomial number of random bits chosen uniformly and independently.

3 Main Result

In this section, we generalize Toda's theorem to arbitrary ATMs as follows.

Theorem 3.1 *Let a language L be accepted by an alternating Turing machine M that has $a(n)$ alternation blocks, guesses $p(n)$ bits during each alternation block, and executes $t(n) \geq p(n)a(n)$ deterministic steps. Then, there exists a nondeterministic machine M' such that for all input x*

$$Prob \begin{pmatrix} x \in L \Rightarrow \mathrm{AR}(M',x,r) = 1 \\ x \notin L \Rightarrow \mathrm{AR}(M',x,r) = 0 \end{pmatrix} \geq \frac{3}{4}\,,$$

where r is chosen uniformly and independently from $\{0,1\}^{O(p^3(n)a(n))}$. M' guesses $(p(n)a(n))^{O(a(n))}$ bits and makes at most $(p(n)a(n))^{O(a(n))} \cdot t(n)$ deterministic steps.

Remarks:

1. Note that the running time of the new counting machine has the following interesting form in terms of the steps of the original ATM:

$$(\text{Total nondeterministic bits})^{\text{Alternation blocks}} \cdot \text{Deterministic steps}\,.$$

2. If $t(n) \geq (p(n)a(n))^{O(a(n))}$, then the running time of the counting machine is the same as that of the original machine. Thus, all the alternation blocks

and nondeterministic steps can be subsumed by one weak counting operator. For example, if $a(n) = O(\log n)$, $p(n) = n^{O(1)}$, and $t(n) = n^{O(\log n)}$, then M' guesses $n^{O(\log n)}$ bits and makes $n^{O(\log n)}$ deterministic steps.

Proof of Theorem 3.1

For the sake of clarity, we write p, a, and t to denote $p(n)$, $a(n)$, and $t(n)$ respectively. Let the $O(p^3 a)$ random bits be given on a separate tape, called random bit tape, and grouped in $O(pa)$ blocks of p^2 bits each: $w_1, w_2, \dots, w_p$, $|w_i| = p, 1 \le i \le p$.

1. Using the random reduction given by Valiant and Vazirani [8], Tarui constructed a random polynomial $q_1(x_1, \dots, x_{2^p})$ over Z to approximate $\mathrm{NOR}(x_1, x_2, \dots, x_{2^p})$ as follows. Pick $w_1, \dots, w_p$ uniformly and independently from $\{0,1\}^p$, and for each $0 \le i \le p$ and each $1 \le j \le 2^p$, let

$$r_j^i = \begin{cases} 1 \text{ if } i = 0 \\ 1 \text{ if } i \ge 1 \text{ and binary}(j-1) \cdot w_1 = \cdots = \text{binary}(j-1) \cdot w_i = 0 \\ 0 \text{ otherwise,} \end{cases}$$

where binary$(j-1)$ denotes the value of $j-1$ written as a p-bit binary vector. Let

$$q_1(x_1, \dots, x_{2^p}) = \prod_{i=0}^{p} (r_1^i x_1 + \cdots + r_{2^p}^i x_{2^p} - 1)^2.$$

If $\mathrm{NOR}(x_1, \dots, x_{2^p}) = 1$, then $q_1(x_1, \dots, x_{2^p}) = 1$, and if $\mathrm{NOR}(x_1, \dots, x_{2^p}) = 0$, then $q_1(x_1, \dots, x_{2^p}) = 0$ with probability at least $\frac{1}{4}$. There are $(2^p + 1)^{2(p+1)}$ *product terms* [1] if we *expand* q_1, each of which can be indexed using $2(p+1)(p+1)$ bits: ith block of $(p+1)$ bits picks a term $r_j^i x_j$ (or -1, when $j = 2^p + 1$) from ith factor. The polynomial has degree $\mathrm{O}(p)$ and uses $\mathrm{O}(p^2)$ random bits. Given $w_1, \dots, w_p$, calculating each r_j^i takes $\mathrm{O}(p^2)$ deterministic steps.

Construct $O(pa)$ polynomials $q_1(x_1, \dots, x_n)$ using independent random bits and let $q(x_1, \dots, x_{2^p})$ denote their product. The new polynomial approximates NOR with one-sided error probability $1 - (3/4)^{O(pa)}$, has degree $O(p^2 a)$, and uses $O(p^3 a)$ random bits. Note that the constant in $O(pa)$ can be used to improve the probability of error to arbitrarily small values. The polynomial has the following form:

$$q(x_1, x_2, \dots, x_{2^p}) = \underbrace{\underbrace{[\ \underbrace{(\qquad)^2}_{2^p+1 \text{ terms}} \cdots (\qquad)^2]}_{2(p+1) \text{ factors for } 0 \le i \le p} \cdots [(\qquad)^2 \cdots (\qquad)^2]}_{O(pa) \text{ random bit blocks using independent bits}}$$

[1] A polynomial in *expanded form* is written as a sum of *product terms* $c_i x_{i_1} x_{i_2} \dots x_{i_k}$.

There are $(2^p+1)^{2(p+1)O(pa)}$ product terms in the expanded polynomial q, which can be indexed using $2(p+1)(p+1)O(pa)$ bits by using $(p+1)$ bits to pick a term from each factor.

2. Convert the ATM M into a clean form so that all the nondeterministic bits are guessed before making any deterministic steps; that is, on input x, $|x| = n$, the machine guesses $p(n)$ nondeterministic bits j_l in the lth alternating block, $1 \leq l \leq a$, and accepts if $\phi(x, j_1, j_2, \ldots, j_a)$, where ϕ is a deterministic predicate that needs $t(n)$ steps.

   This conversion increases the number of computation paths without affecting the worst case bounds for the machine. The computation tree of the clean ATM M can be viewed as a "stratified" circuit of depth a with alternating AND/OR gates, where the fan-in of each gate is 2^p. Introduce a pair of NOT gates in sequence at each input of all AND gates or, equivalently, at the output of all OR gates. By combining one NOT gate with the AND gate and the second with the OR gate, the "stratified" circuit of depth a can be viewed as a circuit with only NOR gates. (We assume that the top gate is an AND gate. Otherwise, negate the output; the output of the counting machine we construct can be easily negated.) Note that the polynomial $q(x_1, x_2, \ldots, x_{2^p})$ constructed in Step 1 approximates all the $O(2^{pa})$ NOR gates in the computation tree with error probability less than $O(2^{pa})(3/4)^{O(pa)} \leq 1/2^{O(1)}$.

3. Replace [2] each NOR gate with the polynomial $q(x_1, x_2, \ldots, x_{2^p})$ constructed in Step 1 as shown in Figure 1. Note that all the polynomials are identical and constructed using the same set of random bits; only their variables are different, depending on the position in the tree.

   As written in the syntactic form shown in Step 1, each q has several occurrences of the same variable: total number of factors in which each x_j appears is $(p+1)O(pa)$. For each occurrence of x_j, add a corresponding child node with polynomial q.

   The tree defines a new polynomial Q that is obtained by substituting q in each child node into the corresponding x_j in the polynomial q for the parent node. We want to simulate the resultant polynomial Q using a nondeterministic machine so that each product term of the expanded polynomial Q is simulated along a computation path of the nondeterministic machine. Since computing the explicit description of Q seems expensive, we do the simulation without it.

[2] Our proof differs from Tarui's from now on. Tarui repeatedly substitutes the polynomials in child nodes into the corresponding term in the parent node to obtain one polynomial as a sum of products. Then nondeterministic bits are used to index each product term. This step is easily done non-uniformly, which is sufficient if the number of alternation blocks does not change with the length of the input.

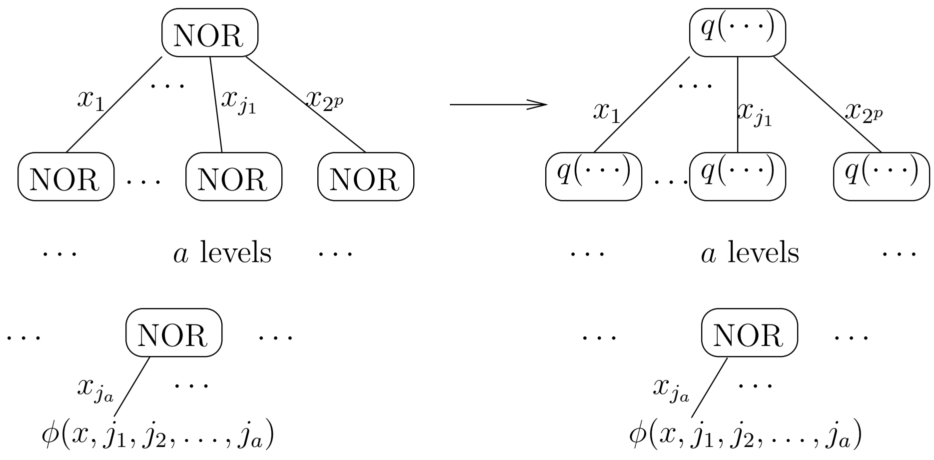

Figure 1: Computation tree of the ATM and the tree obtained by replacing each NOR gate by the polynomial $q(x_1, x_2, \ldots, x_{2^p})$.

In the next step, we nondeterministically guess $2(p+1)^2O(pa)$ bits to index each product term of the expanded polynomial q. This gives a term $\prod_{k=1}^{2(p+1)O(pa)} \alpha_k$, where α_k is picked from each of the $2(p+1)$ factors of the $O(pa)$ random bit blocks of q. Thus, $\alpha_k = r_{j_k}^{i_k} x_{j_k}$ or -1.

When $\alpha_k \neq -1$ in a node, we replace it by a product term of the expanded polynomial q that corresponds to the kth child of the node. The above process of picking up a product term of the expanded polynomial q is repeated for all α_k's and for all the a levels. Using $2(p+1)^2O(pa)$ additional nondeterministic bits for each variable at each level, we guess a product term of the expanded polynomial Q. Thus, we need a total of $(2(p+1)^2O(pa))^a = O((p^3a)^a)$ nondeterministic bits. The product term of the expanded polynomial Q so selected is represented in Figure 2. Each node has $2(p+1)O(pa)$ terms α_k, and when $\alpha_k \neq -1$, then we have a child node.

Each product term of the expanded polynomial Q represented by Figure 2 evaluates to one of the values $\{0, 1, -1\}$. If the value is 0, then the corresponding computation path guesses a bit and accepts on 1 and rejects on 0, so that the path contributes 0 to $\mathrm{AR}(M', x, r)$. Otherwise, accept or reject depending on the sign so that the computation path contributes the appropriate value to $\mathrm{AR}(M', x, r)$.

4. Guess the $O((p^2a)^a)$ nondeterministic words $j_k, 1 \leq k \leq O((p^2a)^a)$, $|j_k| = p+1$, to index a product term of the expanded polynomial Q. The product term, obtained by picking one term $r_{j_k}^{i_k} x_{j_k}$ (or -1) in each factor within

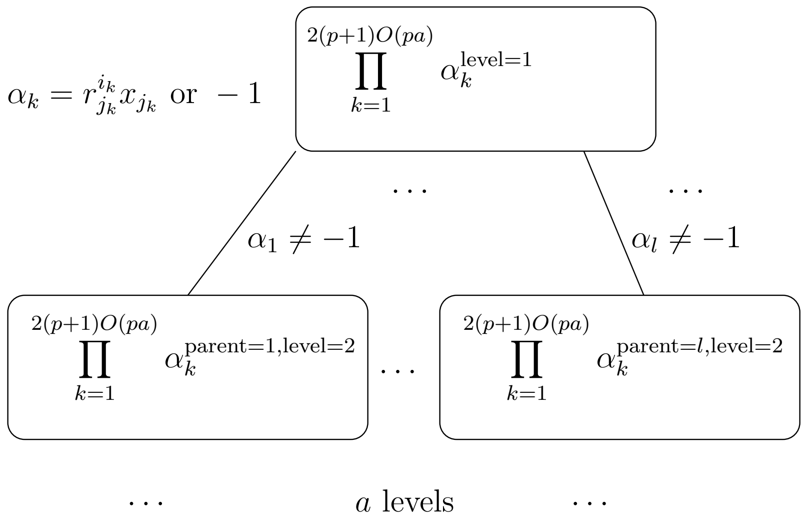

Figure 2: Tree representing one product term of the expanded polynomial Q; each node has one product term of the expanded polynomial q.

each random bit block of q, is represented by Figure 2. We assume that the nondeterministic bits are ordered in the depth-first order of the tree traversal. Within a node j_k's are ordered according to the order of random bit blocks, and within each random bit block according to the value of i_k. Therefore, as we read j_k's from the tape on which the nondeterministic bits are stored, we traverse the tree in Figure 2 in depth-first order.

For each iteration of the depth-first traversal, which accesses a new α_k, $1 \leq k \leq O((p^2a)^a)$, do the following.

(a) Read the next $p+1$ nondeterministic bits j_k. If $j_k = 2^p+1$, it indexes the term -1 of the polynomial q. Toggle the sign bit. (Use one toggle bit to keep track of the current sign of the product term of Q.) If $j_k > 2^p + 1$, guess a bit and reject on 0 and accept on 1.

(b) Calculate $r^{i_k}_{j_k}$ by accessing the correct random bit block from the tape. If $r^{i_k}_{j_k} = 0$, then guess a bit and reject on 0 and accept on 1.

We have to be careful in ensuring that this step is done efficiently because the random bits needed to calculate $r^{i_k}_{j_k}$ are stored on a sequential tape. The analysis below shows that we can access the random bits in parallel with the previous iteration of depth-first traversal.

(c) Every time the depth-first traversal reaches the leaf level of the tree, we have computation paths $j_{k_1}, j_{k_2}, \ldots, j_{k_a}$, corresponding to each of the a alternation blocks of the original ATM M, given by j_k's at each level. Simulate the deterministic predicate $\phi(x, j_{k_1}, j_{k_2}, \ldots, j_{k_a})$ and if it rejects guess a bit and reject on 0 and accept on 1. We do not need to perform the simulation if any j_{k_l} is $2^p + 1$ and the indexed term is -1; just toggle the sign bit.

(d) At the end of the depth-first traversal, all the $O((p^2a)^a)$ computation paths of the original ATM M accept and all the $r^{i_k}_{j_k}$'s are 1. If the sign of the product term is positive, then accept, otherwise reject.

Analysis:
It is easy to see that the counting machine needs $O(p^3a)$ random bits in Step 1 and guesses $O(p(p^2a)^a)$ nondeterministic bits in Step 4. Step 4a simply moves the tape head forward to access the $O(p(p^2a)^a)$ nondeterministic bits in sequence. Step 4c simulates the deterministic steps of ATM M on $O((p^2a)^a)$ computation paths and takes $O(t(p^2a)^a)$ steps.

Calculating $O((p^2a)^a)$ $r^{i_k}_{j_k}$'s in Step 4b requires $O(p^2(p^2a)^a)$ deterministic steps. However, we need to be careful in the analysis of deterministic steps required to access the random bits needed to calculate $r^{i_k}_{j_k}$'s. We show that accessing the appropriate random bits can be done in parallel with the $O((p^2 + t)(p^2a)^a)$ deterministic steps needed in Steps 4b and 4c during the previous depth-first iteration.

At every step of depth-first traversal, the following tuple is pushed—popped, when depth-first traversal retreats to an earlier level of the tree—on a separate tape used as a stack: $\langle$Current *level*, current *factor*, current *block*$\rangle$, where $1 \leq$ *level* $\leq a$, $1 \leq$ *factor* $\leq 2(p+1)$, and $1 \leq$ *block* $\leq O(pa)$. The *factor* and *block* respectively are the indices of the current factor and random bit blocks of the polynomial q, and *level* is the current level of the tree. The depth-first starts with $(1, 1, 1)$, *level* is incremented the fastest, followed by *factor*, followed by *block*. The value of i_k needed to calculate $r^{i_k}_{j_k}$ is given by *factor*, and *block* defines the random bit block needed to calculate $r^{i_k}_{j_k}$.

When the depth-first search accesses a child node for the first time, we need to access the first random bit block. This can be done in $O(1)$ steps by keeping a copy of the first random bit block on a separate tape.

When the depth-first search moves forward within a node at the leaf level, we need to access the random bits in their given order on the tape.

However, when the depth-first search retreats from a leaf node to one of the ancestors, we need to access an intermediate blocks of random bits. This step can be done in parallel by pre-checking the retreats while Steps 4b and 4c are executed at the leaf level during the previous iteration of depth-first traversal. When the last $\alpha_k \neq -1$ of a node is being simulated at the leaf level in a depth-first iteration, then the next iteration will retreat to an ancestor node that is

1. The input is given on two tapes. The first tape contains (M, x) and the second tape contains $O(pa)$ independently chosen blocks of random bits $w_1, \ldots, w_p$, where each $|w_i| = p$, for a total of $O(p^3 a)$ random bits.

2. On a separate tape guess $O((p^2 a)^a)$ words of nondeterministic bits, where each word is of size $p+1$. These identify $r_j^i x_j$'s in each node and at each level of the tree of Figure 2 in depth-first order. Note that each $j, 1 \leq j \leq 2^p + 1$, requires $p+1$ nondeterministic bits and each node has $2(p+1)O(pa)$ such x_j's and children for exactly a levels.

3. Traverse the tree of Figure 2 in the depth-first order, calculating r_j^i using the random bits and nondeterministic bits. Use one bit to keep track of the current sign of the term represented by the tree. When $j = 2^p + 1$ corresponding to the term -1, the sign bit toggles. If any $r_j^i = 0$, guess a bit and reject on 0 and accept on 1.

   Every time the depth-first search reaches the leaf level of the tree, we have computation paths $j_1, \ldots, j_a$, corresponding to each of the a alternation blocks of the ATM M, given by j_k's at each level. Simulate the deterministic steps of the ATM M on this computation path and if it rejects guess a bit and reject on 0 and accept on 1.

   At the end of the depth-first traversal, all the $O((p^2 a)^a)$ computation paths of ATM M accept and all the r_j^i's are 1. If the sign of the term is positive, then accept, otherwise reject.

Table 1: Algorithm for the counting machine

---

not at the last factor in the last random bit block. Search the stack tape backwards to access the first tuple $\langle \mathit{level}, \mathit{factor}, \mathit{block} \rangle$ such that *factor* is not equal to $2(p+1)$ or $\mathit{block} \neq O(pa)$. If $\mathit{factor} < 2(p+1)$, then we need to access the same random bit block in the next iteration. Otherwise, retrieve the random bit block given by $\mathit{block} + 1$ from the random bit tape and copy it on a separate work tape. Since the deterministic simulation and calculation of $r_{j_k}^{i_k}$'s requires $O((p^2 + t)p^2 a)$ steps, we have ample time to access the correct random bit block among $O(p^3 a)$ random bits and to copy it on to a separate tape. $\Box$

The algorithm for the counting machine is summarized in Table 1. It is clear from the proof that the error probability of 1/4 can be decreased to $1/2^e$ using $e + O(pa)$ independent polynomials in Step 1.

4 Conclusion and Future Research

We have shown that a key step in Tarui's proof can be performed uniformly and efficiently on a Turing machine, which is needed to generalize Toda's theorem to ATMs that can have an arbitrary number of alternation blocks. We have provided an explicit algorithm that generalizes Toda's result to arbitrary ATMs. We have also shown how the running time of the counting machine relates to the bounds on the various parameters of the original ATM. As to the question whether the exponential number of nondeterministic and deterministic steps are really necessary, in [5], we construct a different counting machine that makes only a polynomial number of nondeterministic steps, but needs an exponential number of random bits, a polynomial number of independent random bits along different computation paths. Furthermore, each computation path of the new machine simulates a constant number of deterministic steps of the original ATM. These results show that there is no a priori reason to believe that the bounds cannot be further improved, at least based on our current knowledge.

References

1. R. Beigel, N. Reingold, and D. Spielman. PP is closed under intersection. *Journal of Computer and System Sciences*, 50(2):191–202, 1995.
2. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *JACM*, 8(1):114–133, 1981.
3. S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.
4. S. Gupta. Closure properties and witness reduction. *Journal of Computer and System Sciences*, 50(3):412–432, 1995.
5. S. Gupta. PSPACE and the power of counting. Manuscript, 2001.
6. J. Tarui. Randomized polynomials, threshold circuits, and the polynomial hierarchy. *Theoretical computer science*, 113:167–183, 1993.
7. S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20:865–877, 1991.
8. L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.