A Superpolynomial Lower Bound on the Size of Uniform Non-constant-depth Threshold Circuits for the Permanent

Pascal Koiran
LIP
École Normale Supérieure de Lyon
Lyon, France
pascal.koiran@ens-lyon.fr

Sylvain Perifel
LIAFA
Université Paris Diderot – Paris 7
Paris, France
sylvain.perifel@liafa.jussieu.fr

***Abstract*—We show that the permanent cannot be computed by DLOGTIME-uniform threshold or arithmetic circuits of depth $o(\log\log n)$ and polynomial size.**

***Keywords*-permanent; lower bound; threshold circuits; uniform circuits; non-constant depth circuits; arithmetic circuits**

I. Introduction

Both in Boolean and algebraic complexity, the permanent has proven to be a central problem and showing lower bounds on its complexity has become a major challenge. This central position certainly comes, among others, from its $\sharp$P-completeness [15], its VNP-completeness [14], and from Toda's theorem stating that the permanent is as powerful as the whole polynomial hierarchy [13]. More recently, it played a role in the celebrated and subtle result of Kabanets and Impagliazzo [6]: either $\mathsf{NEXP}^{\mathsf{RP}}$ does not have Boolean circuits of polynomial size, or the permanent does not have arithmetic circuits of polynomial size.

However little is known about the circuit complexity of the permanent in the general case. Indeed, the best lower bound so far on its circuit size is no more than the trivial $\Omega(n^2)$ (remember that PER_n has n^2 variables). Despite this rather dark state of affairs, some progress has been made on restricted classes of circuits. For instance, we know lower bounds on *monotone* circuits (such circuits for the permanent must have exponential size, see [5], [11]), and recently, lower bounds on *multilinear* circuits were obtained (see e.g. [8], [9], [10]).

A lot of work has also been done on *constant-depth* circuits, in which gates have unbounded fan-in. This line of research has been quite successful on Boolean circuits and gave deep insights into circuit complexity: see e.g. [3], [12]. However, pushing the limit of lower bounds beyond constant depth for polynomial-size circuits has remained elusive so far.

Another restriction worth studying is *uniformity*: circuits are not arbitrary any more but are required to be described by a Turing machine. If this description is very efficient (running in time logartihmic in the size of the circuit, we speak of DLOGTIME-uniformity), Allender [1] (see also similar results on circuits with modulo gates in [2]) has shown that the permanent does not have threshold circuits of *constant* depth and "sub-subexponential" size. In this paper, we obtain a tradeoff between size and depth: instead of sub-subexponential size, we only prove a superpolynomial lower bound on the size of the circuits, but now the depth is no more constant. More precisely, we show the following theorem.

Theorem 1: The permanent does not have DLOGTIME-uniform polynomial-size threshold circuits of depth $o(\log\log n)$.

It seems to be the first superpolynomial lower bound on the size of *non-constant-depth* threshold circuits for the permanent (though a lower bound is proved in [10] on *multilinear* arithmetic circuits of depth $o(\log n/\log\log n)$). Admittedly, the depth $o(\log\log n)$ is still small but until now the known techniques were only able to prove lower bounds on constant-depth circuits. Note that our techniques are not really very different from those of [1]—but seeing how to step beyond O(1) depth does require a different way of looking at the problem. It is also worth remarking that our technique can readily be adapted to an easier proof of Allender's result [1] (see Remark 2 at the end of the paper).

Let us very briefly describe our proof technique. In contrast with [1], we do not use the relation between threshold circuits and the counting hierarchy, which implied to consider only constant-depth circuits. Also, the diagonalization in [1] is a variant on the nondeterministic time hierarchy theorem. Here, we use the usual deterministic time hierarchy theorem as an indirect diagonalization : under the assumption that the permanent has DLOGTIME-uniform circuits of polynomial size and depth $o(\log\log n)$, we show

1) the value of a threshold circuit of size s and depth d can be computed in time $(\log s)^{2^{O(d)}}$ (Lemma 3 combined with Lemma 1);
2) every language in E has uniform threshold circuits of size $2^{O(n)}$ and depth $o(\log n)$ (Corollary 4).

These two points together imply that every language in E can be computed in subexponential time, a contradiction with the time hierarchy theorem.

Since threshold circuits can simulate arithmetic circuits, we also obtain a superpolynomial lower bound on the size of uniform arithmetic circuits of depth $o(\log \log n)$ for the permanent (Corollary 7).

Organization of the paper — The next section is devoted to the definition of the notions in use: circuits (Boolean, threshold, arithmetic), uniformity and some complexity classes. Then Section III is dedicated to the proof of Theorem 1 by showing a series of results along the way suggested above.

II. Preliminaries

The notions we use are very standard but, for completeness, we still recall them in this section.

A. Boolean circuits

A Boolean circuit on n variables is a directed acyclic graph, whose vertices are labeled either by a variable among $\{x_1, \dots, x_n\}$ or by an operation among $\{\vee, \wedge, \neg\}$. Vertices of indegree[1] 0 are called inputs, the others are called gates. A gate labeled by $\neg$ is required to have indegree 1, whereas gates labeled by $\vee$ or $\wedge$ have indegree 2. A single gate has outdegree 0 and is called the output gate.

The value computed by a vertex is defined recursively: an input x_i has for value the value of the variable $x_i \in \{0, 1\}$. A $\neg$ gate $g = \neg h$ has for value the negation of the value of h. An $\vee$ gate $g = h_1 \vee h_2$ (respectively an $\wedge$ gate $g = h_1 \wedge h_2$) has for value the disjunction (resp. conjunction) of the values of h_1 and h_2. The value of the circuit is by definition the value of its output gate.

The size of the circuit is the number of vertices and the depth is the length of the longest path from an input vertex to the output gate.

Remark that in order to recognize a language, one needs not only one but a whole family (that is, an infinite sequence) of circuits (C_n), as explained below. There is also a variant in which gates $\vee$ and $\wedge$ have unbounded fan-in: this is useful when defining classes of circuits of constant depth.

B. Threshold circuits

A threshold circuit is similar to a Boolean circuit with $\vee$ and $\wedge$ gates of arbitrary fan-in, but another type of gate is allowed: threshold gates (also known as majority gates). A threshold gate is also of arbitrary fan-in, and its value is 1 if at least half of its inputs have value 1, and 0 otherwise.

Again, in order to recognize a language, a whole family of circuits is needed. Note that it makes sense to consider families of bounded depth threshold circuits since gates are allowed to have arbitrary fan-in.

[1]Indegree and outdegree are also called fan-in and fan-out, respectively.

C. Arithmetic circuits

An arithmetic circuit is similar to a Boolean circuit but with other kinds of gates. It has $+$, $-$ and $\times$ gates, all of fan-in 2, and besides variables, another input is labeled by the constant 1. The variables are not considered to have Boolean values anymore, but instead they are symbolic and the circuit computes a polynomial (over the ring $\mathbb{Z}$) in the obvious way: the value of the input gate labeled by 1 is the constant polynomial 1, the value of an input gate labeled by x_i is the polynomial x_i, the value of a $+$ gate (respectively $-$ gate, $\times$ gate) is the sum (resp. difference, product) of the values of its inputs.

An arithmetic circuit C with n input gates computes a multivariate polynomial over $\mathbb{Z}$ with n variables. Circuit families (C_n) are used to compute families of polynomials. The permanent family (also called permanent for short) is the family (PER_n) of polynomials defined as follows:

$$\mathrm{PER}_n(x_{1,1}, x_{1,2}, \dots, x_{1,n}, x_{2,1}, \dots, x_{n,n}) = \sum_{\sigma} \prod_{i=1}^{n} x_{i,\sigma(i)}$$

where the sum is taken over all the permutations σ of $\{1, \dots, n\}$. The n^2 variables $x_{i,j}$ can be viewed as the coefficients of an $n \times n$ matrix, allowing us to speak of the permanent of a matrix.

D. Uniformity

Circuits, be they Boolean, threshold or arithmetic, are finite objects easily encoded in binary (e.g. by the list of their vertices and edges). Hence they can be handled by Turing machines.

As already mentioned, we are interested in sequences (C_n) of circuits in order to recognize languages. In the most general setting, no assumption is made on the structure of these circuits: in particular, the Boolean encodings of the circuits of a family may be uncomputable. However, if a single Turing machine is able to produce the Boolean encoding of all the circuits of the family, then we speak of uniformity. The degree of uniformity depends on the ressources needed by the machine.

A family of circuits (C_n) is said P-uniform if there exists a deterministic Turing machine which, on input (n, i) given in binary, outputs the i-th bit of the encoding of C_n in time polynomial in n (that is, in time exponential in the size of the input). Similarly, a family of circuits (C_n) is said $\mathsf{DLOGTIME}$-uniform if there exists a deterministic Turing machine which, on input (n, i) given in binary, outputs the i-th bit of the encoding of C_n in time logarithmic in n (that is, in time linear in the size of the input). Of course, $\mathsf{DLOGTIME}$-uniformity implies P-uniformity. It can be argued that $\mathsf{DLOGTIME}$-uniformity is the right notion of uniformity for small-depth circuits, see [7].

Remark 1: In the remainder of the paper, we shall work with $\mathsf{DLOGTIME}$-uniformity, but everything remains valid if replaced by "polylogtime" uniformity.

E. Complexity classes

Here we present definitions of the complexity classes that concern us. Let $\mathsf{DTIME}(t(n))$ denote the set of languages recognized in time $t(n)$ by a deterministic Turing machine. Then P is the class $\mathsf{DTIME}(n^{O(1)}) = \cup_{k>0}\mathsf{DTIME}(n^k)$ (that is, deterministic polynomial time) and E is the class $\mathsf{DTIME}(2^{O(n)}) = \cup_{k>0}\mathsf{DTIME}(2^{k.n})$ (that is, deterministic exponential time with linear exponent).

Recall the time hierarchy theorem [4]: for time-constructible functions f and g, if $f(n)/g(n) = o(1/\log(g(n)))$ then $\mathsf{DTIME}(g(n)) \not\subset \mathsf{DTIME}(f(n))$. In particular, we will use the following consequence: $\mathsf{E} \not\subset \mathsf{DTIME}(n^{2^{o(\log n)}})$.

The class $\sharp\mathsf{P}$ is the set of functions $f : \{0,1\}^* \to \mathbb{N}$ defined as follows: there exist a polynomial p and a language $A \in \mathsf{P}$ such that $f(x) = \#\{y \in \{0,1\}^{p(|x|)} : (x,y) \in A\}$. Computing the permanent of a 0-1 matrix is $\sharp\mathsf{P}$-complete (Valiant [14]). Then PP is the set of languages B such that there is $f \in \sharp\mathsf{P}$ satisfying $[x \in B \iff f(x) \geq 2^{p(|x|)-1}]$. The class PP can also be viewed as the languages B such that there exists a polynomial-time nondeterministic Turing machine N satisfying [$x \in B$ iff at least half of the computation paths of N are accepting]. Remark that if every function in $\sharp\mathsf{P}$ can be computed in polynomial time, then $\mathsf{PP} = \mathsf{P}$.

Complexity classes can also be defined in terms of circuits (either Boolean or threshold). An input x is accepted by a circuit C if the value of C on x, denoted by $C(x)$, is 1. In order to recognize languages, families (C_n) of circuits are considered: circuit C_n will recognize inputs of size n, hence we make the assumption that C_n has n input gates. Now, a language A is recognized by a family (C_n) of circuits if $A = \{x \in \{0,1\}^* : C_{|x|}(x) = 1\}$.

We shall use the well known characterization of P in terms of circuits: P is the set of languages recognized by P-uniform families of polynomial-size Boolean circuits. The class AC^0 is the set of languages recognized by a family of constant-depth Boolean circuits of polynomial size, where the gates $\vee$ and $\wedge$ have unbounded fan-in. The class TC^0 is the set of languages recognized by a family of constant-depth threshold circuits of polynomial size. Uniform versions of these classes, $\mathsf{DLOGTIME}$-AC^0 and $\mathsf{DLOGTIME}$-TC^0 respectively, are defined by requiring $\mathsf{DLOGTIME}$-uniformity on the circuit family.

III. Technical developments

This series of results is devoted to the proof of Theorem 1.

Lemma 1: If the permanent has P-uniform polynomial-size threshold circuits then PP=P.

Proof: First turn the threshold circuits into Boolean circuits. To this end, every $\wedge$ or $\vee$ gate of unbounded fan-in is replaced by trees of $\wedge$ or $\vee$ gates of fan-in 2 (which clearly remains P-uniform and of polynomial size), and every threshold gate with $N = n^{O(1)}$ inputs is replaced by the addition of the inputs followed by a comparison of the result with $N/2$. This iterative addition can easily be carried out by a P-uniform circuit of size polynomial in N, hence polynomial in n. This proves that the permanent has P-uniform polynomial-size Boolean circuits.

Thus, by $\sharp\mathsf{P}$-completeness of the permanent every function in $\sharp\mathsf{P}$ can be computed in polynomial time. This implies that $\mathsf{PP} = \mathsf{P}$. ■

As a preparation to the proof of Lemma 3, let us first rephrase the hypothesis $\mathsf{PP} = \mathsf{P}$ in a convenient way.

Lemma 2: Let A be a language with a (deterministic) algorithm $\mathcal{A}$ running in time at most $t(n) \geq n$, where $t : n \mapsto t(n)$ is a function computable in time $O(t(n))$ by an algorithm $\mathcal{T}$. Consider the following problem B: given a word x, a length n and an integer $N \leq 2^n$, decide whether at least N words $y \in \{0,1\}^n$ satisfy $(x,y) \in A$.
If $\mathsf{PP} = \mathsf{P}$ then B has an algorithm $\phi(\mathcal{A},\mathcal{T})$ running in time $p(t(n+|x|))$ for a fixed polynomial p (independent of A). Furthermore, the transformation ϕ computing from $(\mathcal{A},\mathcal{T})$ the code of the algorithm for B is computable in linear time.

Proof: Remark that this is not a completely obvious consequence of $\mathsf{PP} = \mathsf{P}$ since the polynomial p is required to be independent of A. In fact this comes from the existence of a complete problem for PP. Take indeed the canonical PP-complete language

$$H = \{(M,x,1^n) : \begin{array}{l}\text{at least half of the}\\ \text{computation paths of } M(x)\\ \text{are accepting in time } n\},\end{array}$$

where M is a nondeterministic Turing machine. The hypothesis $\mathsf{PP} = \mathsf{P}$ implies that H is decidable in time $p(n)$ for some polynomial p.

To the problem B is associated a language $\tilde{B} = \{(x,n,N,1^{t(n+|x|)}) : \#\{y \in \{0,1\}^n : (x,y) \in A\} \geq N\}$. Then $\tilde{B}$ is in PP and a reduction from $\tilde{B}$ to H is the mapping $(x,n,N,1^{t(n+|x|)}) \mapsto (M,(n,x,N),1^{t(n+|x|)})$, where $M(n,x,N)$ has the following behaviour: it first guesses $b \in \{0,1\}$; if $b = 0$, then it creates $2^n - N$ accepting paths among 2^n paths; if $b = 1$, then it guesses $y \in \{0,1\}^n$ and decides whether $(x,y) \in A$ by running the algorithm for A in time $t(n+|x|)$. Therefore $M(n,x,N)$ runs in time $O(t(n+|x|))$, has 2^{n+1} paths, and among them $2^n - N + \#\{y \in \{0,1\}^n : (x,y) \in A\}$ are accepting. This is more than half of the total number of paths iff $\#\{y \in \{0,1\}^n : (x,y) \in A\} \geq N$. Since $t(n+|x|)$ is computable in time $O(t(n+|x|))$ by the algorithm $\mathcal{T}$, this reduction shows that $\tilde{B}$ is decidable in time $p(t(n+|x|))$.

Furthermore, note that the algorithm for $\tilde{B}$ just consists in the computation of $t(n+|x|)$ thanks to the algorithm $\mathcal{T}$, and then in the execution of the algorithm for H in which the algorithm $\mathcal{A}$ is plugged. This proves the claim on ϕ. ■

Similar to succinct representations used for exponential-time-complete languages, threshold circuits can be succinctly given, not by their binary encoding but rather by a description of their gates. That is, instead of giving the threshold circuit C directly, a Boolean circuit B is given, whose value $B(i)$ on input i is the i-th bit of the encoding of C. This may enable much shorter representations of the circuit. Circuits given in that way will be called "succinctly given".

Lemma 3: Let $A_{\leq d}$ be the problem of deciding the value of a succinctly given threshold circuit of depth $\leq d$, that is,

$$A_{\leq d} = \{(B,x) : \begin{array}{l} B \text{ represents a threshold circuit } C \\ \text{of depth} \leq d, \text{ and } C(x)=1\} \end{array}$$

where B is a Boolean circuit and x is a Boolean input to C of appropriate size. The size of the input (B,x) is denoted by m.

If $\mathsf{PP} = \mathsf{P}$, then $A_{\leq d}$ has an algorithm of running time $m^{2^{O(d)}}$. Furthermore, the code of this algorithm (that is, the identifier of a Turing machine for it) is computable from d in time $2^{O(d)}$ by a fixed Turing machine.

Proof: The idea is to recursively evaluate the values of the gates at each depth of the circuit, using Lemma 2 for threshold gates. In order to apply Lemma 2, one has to consider all the inputs of a particular gate, leading us to define the language A_k corresponding to the gates being inputs of the i-th gate of C, whose depth is $\leq k$, as follows:

$$A_k = \{(B,x,i,j) : \begin{array}{l} B \text{ represents a threshold circuit } C \\ \text{in which gate } i \text{ is at depth} \leq k, \\ \text{gate } j \text{ is an input of gate } i, \text{ and} \\ \text{the value of gate } j \text{ is 1 in the} \\ \text{computation } C(x)\} \end{array}$$

Note that one can artificially add to C a final "identity gate" taking as input the output of C, in which case deciding $A_{\leq d}$ reduces to deciding A_{d+1}.

Let us call $T(k,d,m')$ the time needed to decide A_k as a function of the depth d of C, and of the size m' of the input (B,x,i,j). Note that $m' = \Theta(m)$ since the size of i and j is less than or equal to that of B (indeed, B takes as input an integer representing a gate number, thus of the order of i and j). The language A_2 consists in cheching that gate j is an input of gate i, and in evaluating gate j on level 1 (an input gate), that is, deciding to which bit of x it corresponds: this can be done in polynomial time, hence in time $m'^{O(1)}$. Therefore $T(2,d,m') = m'^{O(1)}$.

The purpose is now to decide A_{k+1} by using the algorithm for A_k. It can be done easily since we can decide the value of a gate at depth k if we know the values of the gates at depth $\leq k-1$. Indeed, let us decide whether $(B,x,i,j) \in A_{k+1}$, supposing gate i is at depth $k+1$ and has gate j as input: we want to compute the value of gate j. Since gate j is at depth $\leq k$, the algorithm for A_k provides the value of all the inputs of gate j, which are used in turn to compute the value of gate j itself. More precisely we proceed inductively:

- If gate j is a $\neg$ gate, that is, $f = \neg g$, then the value of f is the negation of the value of g.
- If gate j is an $\vee$ or an $\wedge$ gate, that is, $f = g \diamond h$ with $\diamond \in \{\vee, \wedge\}$, then we perform the corresponding Boolean operation on the values of g and h.
- Finally, if gate j is a threshold gate, we decide whether at least half of its inputs evaluate to 1.

Let us bound the execution time $T(k+1,d,m')$ of this algorithm for A_{k+1} as a function of $T(k,d,m')$ (the execution time of the algorithm for A_k). In the first case, we take the negation of one request of the form $(B,x,j,g) \in A_k$, therefore we have the following relation: $T(k+1,d,m') = T(k,d,m') + O(1)$. Similarly, in the second case we make a Boolean combination of two requests (one for each input), hence $T(k+1,d,m') \leq 2T(k,d,m') + O(1)$. Finally in the third case, the task is to decide whether more than half of the inputs y of gate j evaluate to 1. Applying Lemma 2 to the language A_k, with N being the number of inputs to gate j and with requests of the form $(B,x,j,y) \in A_k$ for all gates y, yields $T(k+1,d,m') \leq p(T(k,d,m'))$ for some fixed polynomial p.

As a whole, we have the following relation, for a fixed polynomial p:

$$T(k+1,d,m') \leq p(T(k,d,m')).$$

In other words, there exists an exponent $\alpha \in \mathbb{N}$ such that $T(k+1,d,m') \leq T(k,d,m')^{\alpha}$, hence $T(k,d,m') \leq T(2,d,m')^{\alpha^k}$. Since $T(2,d,m') = m'^{O(1)}$ and deciding A requires to go up to $k = d+1$, there is an algorithm for $A_{\leq d}$ running in time

$$T(d+1,d,m') \leq m'^{2^{O(d)}} \leq m^{2^{O(d)}}.$$

Note that this upper bound for T is easily computable in time $O(T(d+1,d,m'))$ by a fixed algorithm $\mathcal{T}$.

Furthermore, the algorithm for A_{k+1} consists in one or two calls to the algorithm for A_k in the two first cases, and in one call to the algorithm for $\phi(\mathcal{A}_k, \mathcal{T})$ in the third case, where ϕ is the transformation given by Lemma 2 and $\mathcal{A}_k$ is the algorithm for A_k. Therefore the algorithm for A_{k+1} is computable from the algorithms for A_k and T in linear time. This proves that the algorithm for $A_{\leq d}$ is computable by a fixed Turing machine in time $2^{O(d)}$. ■

The limitation on the depth can now be removed. Note that if we worked at constant depth, Lemma 3 would be enough (see Remark 2); the requirement on the constructibility of the algorithm for $A_{\leq d}$ would even be useless. However, the following lemma is needed for nonconstant depth.

Lemma 4: Let A be the problem of deciding the value of a succinctly given threshold circuit, that is,

$$A = \{(B,x) : \begin{array}{l} B \text{ represents a threshold circuit } C \\ \text{and } C(x)=1\} \end{array}$$

where B is a Boolean circuit and x is a Boolean input to C of appropriate size. The depth of the threshold circuit C is denoted by d and the size of the input (B,x) by m.

If $\mathsf{PP} = \mathsf{P}$, then A has an algorithm of running time $m^{2^{O(d)}}$.

Proof: It is enough to show that the language

$$\tilde{A} = \{(B, x, 1^d) : (B, x) \in A_{\leq d}\},$$

where $A_{\leq d}$ is the language defined in Lemma 3, has an algorithm of running time $m^{2^{O(d)}}$.

In order to decide $\tilde{A}$, one first computes in time $2^{O(d)}$ the code of the algorithm for $A_{\leq d}$ thanks to Lemma 3, and then executes it in time $m^{2^{O(d)}}$ on input (B, x). ∎

Lemma 4 concerns the evaluation of succinctly given threshold circuits; the direct consequence for languages is the following corollary. Remark that the hypothesis of polynomial-time constructibility of (C_n) in its statement implies that the size of C_n is at most $2^{n^{O(1)}}$ (in order to access to the gates of C_n by numbers of polynomial bitsize).

Corollary 1: Suppose a language A has threshold circuits of depth $d(n)$ and constructible in polynomial time (that is, the i-th bit of C_n is computable in time $n^{O(1)}$).

If $\mathsf{PP} = \mathsf{P}$, then A has an algorithm of running time $n^{2^{O(d(n))}}$.

Let us now see how to relate the hypothesis on the permanent to decision languages. We need the following result concerning the completeness of the permanent under a very strong notion of reduction. This result appears in [2] as a careful analysis of the usual reduction of Valiant [15] (see also [16] for many-one reductions), which can in fact be carried out in a much more efficient way than just polynomial time.

Proposition 1: The permanent of 0-1 matrices is hard for $\sharp\mathsf{P}$ under $\mathsf{DLOGTIME}$-uniform AC^0 many-one reductions, that is, the reduction is computed by $\mathsf{DLOGTIME}$-uniform AC^0 circuits.

Corollary 2: Every language $A \in \mathsf{P}$ can be expressed as a function of the permanent of a 0-1 matrix M of size $n^{O(1)}$, computed by $\mathsf{DLOGTIME}$-uniform AC^0 circuits. More precisely, there are functions M and α computed by $\mathsf{DLOGTIME}$-uniform AC^0 circuits such that $x \in A \Rightarrow \alpha(\mathrm{PER}(M(x))) = 1$ and $x \notin A \Rightarrow \alpha(\mathrm{PER}(M(x))) = 0$.

Scaling up this result to exponential time yields the following corollary.

Corollary 3: For every language $A \in \mathsf{E}$, there are two functions M and α computable by size $2^{O(n)}$, constant-depth Boolean circuits constructible in linear time (that is, the i-th bit C_n is computable in time $O(n)$), such that $x \in A \Rightarrow \alpha(\mathrm{PER}(M(x))) = 1$ and $x \notin A \Rightarrow \alpha(\mathrm{PER}(M(x))) = 0$.

This implies the following result.

Corollary 4: If the permanent has $\mathsf{DLOGTIME}$-uniform polynomial-size threshold circuits of depth $d(n)$, then every language A in E has threshold circuits of size $2^{O(n)}$ and depth $O(d(2^{O(n)}))$, these circuits being constructible in linear time (that is, the i-th bit of C_n is computable in time $O(n)$).

Proof: By Corollary 3, membership to A is decided by the permanent of a matrix $M(x)$ of size $2^{O(n)}$. It is enough to compute the matrix $M(x)$ by constant-depth uniform circuits, then to plug the result into the uniform threshold circuits of depth $d(2^{O(n)})$ for the permanent of matrices of size $2^{O(n)}$, and finally to apply function α computed by constant-depth uniform circuits. The remaining circuits are again uniform threshold circuits of depth $O(d(2^{O(n)}))$. ∎

Combining Corollary 4, Lemma 1 and Corollary 1 yields the following.

Corollary 5: If the permanent has $\mathsf{DLOGTIME}$-uniform polynomial-size threshold circuits of depth $d(n)$, then $\mathsf{E} \subseteq \mathsf{DTIME}(n^{2^{O(d(2^{O(n)}))}})$.

This is in contradiction with the time hierarchy theorem as soon as $d(n) = o(\log\log n)$, hence we have proved our main result:

Corollary 6 (Theorem 1): The permanent does not have $\mathsf{DLOGTIME}$-uniform polynomial-size threshold circuits of depth $o(\log\log n)$.

Since an arithmetic circuit can be simulated by a threshold circuit (addition and multiplication are indeed in $\mathsf{DLOGTIME}$-uniform TC^0), we obtain the following corollary.

Corollary 7: The permanent does not have $\mathsf{DLOGTIME}$-uniform polynomial-size arithmetic circuits of depth $o(\log\log n)$.

Remark 2: The same proof with minor modifications yields the following result of Allender [1]: the permanent does not have $\mathsf{DLOGTIME}$-uniform constant-depth threshold circuits of size $S(n)$, as soon as $\forall k, S^{(k)}(n) = o(2^n)$ (the k-th iterate of S on n is $o(2^n)$). This includes for instance functions such as $S(n) = n^{\log^i n}$ or $S(n) = n^{(\log n)^{\log\log n}}$.

To see this, the following modifications should be adopted: Lemma 1 should concern circuits of size $S(n)$ (as everywhere else in the paper) and have the conclusion $\mathsf{PP}(S(n)) = \mathsf{DTIME}(S(n))$ (this equality should replace $\mathsf{PP} = \mathsf{P}$ everywhere). The running time of the algorithm of Lemma 2 should be changed to $S(t(n + |x|))$, and in Lemma 3 to $S^{(O(d))}(m)$. Lemma 4 is no more needed since Lemma 3 for constant d already provides an algorithm of fixed length. For Corollary 1, the time should be $S^{(O(d))}(\log s(n))$, where $s(n)$ is the size of the circuits. In Corollary 4, the size should be replaced by $S(2^n)$ and the depth by $O(1)$. Finally, in Corollary 5, E would be included in $\mathsf{DTIME}(S^{(O(1))}(n))$.

Acknowledgments — The authors are indebted to Eric Allender and Dieter van Melkebeek for their comments and corrections and want to thank them warmly, as well as the (anonymous) referees.

References

[1] E. Allender. The permanent requires large uniform threshold circuits. *Chicago J. Theor. Comput. Sci.*, (7), August 1999.

[2] E. Allender and V. Gore. A uniform circuit lower bound for the permanent. *SIAM J. Comput.*, 23(5):1026–1049, 1994.

[3] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

[4] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–305, 1965.

[5] M. Jerrum and M. Snir. Some exact complexity results for straight-line computations over semirings. *J. ACM*, 29(3):874–897, 1982.

[6] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

[7] D. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *J. Comput. Syst. Sci.*, 41(3):274–306, 1990.

[8] R. Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. In *STOC*, pages 633–641, 2004.

[9] R. Raz, A. Shpilka, and A. Yehudayoff. A lower bound for the size of syntactically multilinear arithmetic circuits. In *FOCS*, pages 438–448, 2007.

[10] R. Raz and A. Yehudayoff. Lower bounds and separations for constant depth multilinear circuits. In *IEEE Conference on Computational Complexity*, pages 128–139, 2008.

[11] R. Sengupta and H. Venkateswaran. A lower bound for monotone arithmetic circuits computing 0-1 permanent. *Theor. Comput. Sci.*, 209(1-2):389–398, 1998.

[12] R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *STOC*, pages 77–82, 1987.

[13] S. Toda. On the computational power of PP and $\oplus$P. In *Proc. 30th IEEE Symposium on the Foundations of Computer Science*, pages 514–519, 1989.

[14] L. G. Valiant. Completeness classes in algebra. In *Proc. 11th ACM Symposium on Theory of Computing*, pages 249–261, 1979.

[15] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:181–201, 1979.

[16] V. Zankó. #P-completeness via many-one reductions. *Int. J. Found. Comput. Sci.*, 2(1):77–82, 1991.