Functional oracle queries as a measure of parallel time

Carme Àlvarez
José L. Balcázar
Birgit Jenner

Report LSI–90–24

Functional oracle queries as a measure of parallel time

Carme Àlvarez, José L. Balcázar, and Birgit Jenner

Abstract

We discuss two notions of functional oracle for logarithmic space-bounded machines, which differ in whether there is only one oracle tape for both the query and the answer or a separate tape for the answer, which can still be read on while the following query is already being constructed.

The first notion turns out to be basically non-adaptive, behaving like access to an oracle set. The second notion, on the other hand, is adaptive. By imposing appropriate bounds on the number of functional oracle queries made in this computation model, we obtain new characterizations of the NC and AC hierarchies; thus the number of oracle queries can be considered as a measure of parallel time. Using this characterization of parallel classes, we solve open questions of Wilson.

Resum

Discutim dues nocions d'oracle funcional per màquines amb espai fitat logarítmicament, que difereixen en si es disposa o no de una cinta apart per les respostes obtingudes que es pugui llegir al mateix temps que es construeix la pregunta següent.

La primera noció és bàsicament "no adaptiva", i es comporta com si es tingués accés a un conjunt oracle. La segona, en canvi, és "adaptiva". Imposant fites sobre el nombre de preguntes amb aquest model, obtenim noves caracteritzacions de les jerarquies NC and AC, demostrant-se que el nombre de preguntes es pot interpretar com una mesura del temps paral·lel. Fem servir aquesta caracterització de classes de complexitat paral·lela per resoldre problemes oberts de Wilson.

Functional oracle queries as a measure of parallel time

Carme Àlvarez*, José L. Balcázar*, and Birgit Jenner**

(Preliminary version, August 1, 1990)

Abstract

We discuss two notions of functional oracle for logarithmic space-bounded machines, which differ in whether there is only one oracle tape for both the query and the answer or a separate tape for the answer, which can still be read on while the following query is already being constructed.

The first notion turns out to be basically non-adaptive, behaving like access to an oracle set. The second notion, on the other hand, is adaptive. By imposing appropriate bounds on the number of functional oracle queries made in this computation model, we obtain new characterizations of the NC and AC hierarchies; thus the number of oracle queries can be considered as a measure of parallel time. Using this characterization of parallel classes, we solve open questions of Wilson.

1. Introduction

As a part of the study of the structural properties of the parallel complexity classes in the hierarchies NC and AC, Wilson [18] has studied reducibilities based on these classes. An analog of the Turing reducibility can be obtained by allowing oracle nodes in the circuits, charging these nodes with a certain depth depending on the context: for NC classes each oracle node counts as a depth logarithmic in the number of its input wires, while for AC classes these nodes count depth 1. The naturalness of this definition is argued in [18] and in [17].

Natural questions solved by Wilson compare the power of the AC^k reducibility with that of NC^{k+1} reducibility for certain interesting oracle classes. In particular, he has shown that AC^k and NC^{k+1} reducibilities coincide if applied to classes in the NC or AC hierarchy. Indeed, it holds for all $k \geq 0$, $i \geq 1$:

$$AC^k(NC^i) = NC^{k+1}(NC^i) = NC^{k+i};$$
$$AC^k(AC^i) = NC^{k+1}(AC^i) = AC^{k+i}.$$

* Departament L.S.I., Universitat Politècnica de Catalunya, Pau Gargallo 5, 08028 Barcelona, Spain. Work partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM).

** Institut für Informatik, Technische Universität München, Arcisstraße 21, 8000 München 2, FRG. Work done while visiting the L.S.I. Department of UPC in Barcelona supported by Deutsche Forschungsgemeinschaft (DFG-Forschungsstipendium Je 154/1-1).

His proof of the right to left inclusions in these equalities relies on a "leveling" technique which is only applicable to NC circuits without oracle nodes. Therefore, he left open the question of whether these reducibilities coincide for other oracle classes; in particular, whether they are the same if applied to the classes L and NL. Answering such questions, he argued, may be helpful in understanding the relationship between AC^k and NC^{k+1}.

We will in the following provide (affirmative) answers to these open questions in [18], indeed finding characterizations that explain in part these relationships for relativized classes. We get these results as a side effect of introducing a new sequential model which allows us to build up both the AC hierarchy from AC^1 up as well as the NC hierarchy from NC^2 up.

The model is defined by logarithmic space bounded Turing machines querying a functional oracle a bounded number of times (dependent on n, the length of the input). If the bound is $f(n)$ and the oracles are taken from a function class $\mathcal{F}$ closed under log space Turing reductions, then it turns out that this model essentially corresponds to log space uniform unbounded fan-in circuits of depth $f(n)$ provided that oracle nodes for functions in $\mathcal{F}$ are allowed in the circuit.

2. Preliminaries

Throughout this paper $\log n$ means the function $\max(1, \lceil \log_2 n \rceil)$. We treat sets of words over a finite, fixed alphabet which when required we will identify with the set $\{0,1\}$. Functions map $\{0,1\}^*$ into $\{0,1\}^*$, and they satisfy that $f(x)$ has the same length for all x of length n (a condition implicitly given if a function is contained in a circuit class).

Many of our complexity classes can be defined in a completely standard way by time-bounded or space-bounded multitape Turing machines; L denotes deterministic logarithmic space, NL denotes nondeterministic logarithmic space, P denotes deterministic polynomial time, and an F prefixing a complexity class name will correspond to functions computed by Turing machines with unbounded output tape within the corresponding resource bounds. See [3] for definitions and basic facts about these classes.

Oracle Turing machines usually query oracle sets. We also use Turing machines that query oracle functions. We will use two kind of oracle devices: in the first one there is an unbounded oracle tape, which is used to write down the query and where in place of it the oracle gives its answer; this answer is erased before writing a new query. In the second model there are unbounded separate tapes for queries and answers, so that the previous answer can be read on while writing the new query. We will see that these models are substantially different.

There are many complexity classes below the class P. We concentrate on classes corresponding to very efficient parallel computation on a feasible amount of hardware: these are the NC and AC hierarchies. There are various characterizations of these classes; we will define them in terms of Boolean circuits. Basic facts about these classes can be found in [4]. Since we employ them to compute functions and allow oracle gates in them, we review concisely the model.

Circuits with bounded fan-in are finite directed acyclic graphs with nodes or gates up to indegree 2 with a certain label or type. Nodes of indegree zero are the input nodes $x_1, x_2, \ldots, x_n$ or nodes labelled 0 or 1; nodes with indegree one are labelled $\neg$,

and nodes with indegree two are labelled $\wedge$ or $\vee$. Some of the nodes are specified as output nodes $y_1, y_2, \ldots, y_m$.

In circuits with unbounded fan-in there is no restriction on the indegree of the $\vee$ and $\wedge$ nodes. These nodes are also called existential and universal, respectively.

A circuit family $\{\alpha_n\} := (\alpha_1, \alpha_2, \ldots)$ computes a function f, if the output of α_n on input x of length n is the same as $f(x)$ for all x. If $f(x) \in \{0,1\}$ for all x, i.e., there is only one output node in each circuit of the family, then we can also say that $\{\alpha_n\}$ accepts the set of all x for which $f(x) = 1$.

The size of a circuit is the number of nodes that it contains. The depth of a circuit is the length of the longest path from the input nodes to an output node. The NC and AC hierarchies contain all those functions which are computable by bounded fan-in, respectively unbounded fan-in circuits of polynomial size and polylogarithmic depth which satisfy a certain "uniformity" condition. Throughout this paper we will assume all circuits to be log space uniform, i.e., the description of α_n can be constructed by a deterministic $O(\log n)$ space bounded Turing machine on input 1^n (we fix a description below). Observe that this uniformity condition already implies that the size of our circuits is polynomial. Each gate i of a circuit is described by a tuple $\langle i, t, p_1, p_2, \ldots p_m \rangle$ specifying the name i of the gate, the type t, and the name p_j of the jth input to gate i. This is valid for both NC and AC circuits, and also for circuits with oracle gates as described later in this section. This description is a natural extension of the direct connection languages proposed in [16], compare [9], for describing NC circuits; the extension just handles unbounded fan-in (or oracle) gates.

We have chosen this form of uniformity for simplicity. For all our circuits the weaker form of log time uniformity (see [8], [5]) would already be sufficient, but is somehow uninteresting since most of the oracles considered already have the computational power to compute functions in deterministic logarithmic space.

For $k \geq 0$ we denote by NC^k, resp. AC^k the class of functions computed by log space uniform NC, resp. AC circuits of depth $O(\log^k n)$.

Observe that some references use the notation NC and AC to denote classes of sets; these correspond in our definitions to the subclasses formed by 0-1 valued functions. But following e.g. [9] we extend the definition to arbitrary functions. The length of the output is anyway polynomially bounded due to the bound on the size of the circuit.

We will allow the NC and AC circuits to have access to oracle gates, which compute the value $f(x)$ of x for a functional oracle $f : \{0,1\}^l \to \{0,1\}^m$ (see [9]). Again, note that usually oracle nodes determine the membership of a string x in an oracle set; this corresponds in our approach to taking $m = 1$, i.e. using a 0-1 valued function f instead of the set $L_f := \{x \mid f(x) = 1\}$ (see e.g. [18], [17]).

For AC circuits, oracle nodes have depth 1. In the case of NC circuits with oracle nodes we want to charge in a fair manner the use of the unbounded fan-in oracle gate, in the same way as we should charge, say, an unbounded fan-in existential gate. Such a gate can only be simulated by an NC circuit of depth logarithmic in the number of its inputs. Thus, by definition, in a NC circuit an oracle gate with k inputs contributes $\log k$ to the depth of the circuit. This is the standard way of counting the depth of oracle nodes (see e.g. [18]). This decision furthermore is motivated by the fact that

a charge of 1 for each oracle gate would yield reductions that are not appropriate for completeness proofs for the class FL or classes within the NC or AC hierarchies. Indeed, in general these classes are closed under NC^1 reducibility but are not known to be closed under such a reducibility.

For each circuit with depth at least logarithmic, each functional oracle node for a function $g: \{0,1\}^n \to \{0,1\}^m$ for $m \geq 1$ can be replaced by a circuit family with the same depth that uses as oracle a set defining the individual bits of g:

$$L_g := \{x\$i\$b \mid |g(x)| \geq i, \text{ and the } i\text{th bit of } g(x) \text{ equals } b\}.$$

(a construction often seen, e.g., in [9], [18]). The new circuit again is log space uniform. Thus, for most function classes we can restrict ourselves to 0-1 oracle functions. This is also the reason why we do not charge extra depth for multiple outputs in functional oracle gates.

Let $\mathcal{F}$ be a function class. Then $NC^k(\mathcal{F})$ and $AC^k(\mathcal{F})$ denote the class of functions computed by log space uniform NC, resp., AC circuits of depth $O(\log^k n)$ which contain functional oracle nodes for a function $f \in \mathcal{F}$. Note that a class of oracle sets $\mathcal{A}$ is equivalent to the class of oracle functions consisting of all the characteristic functions c_A for languages A contained in the language class $\mathcal{A}$.

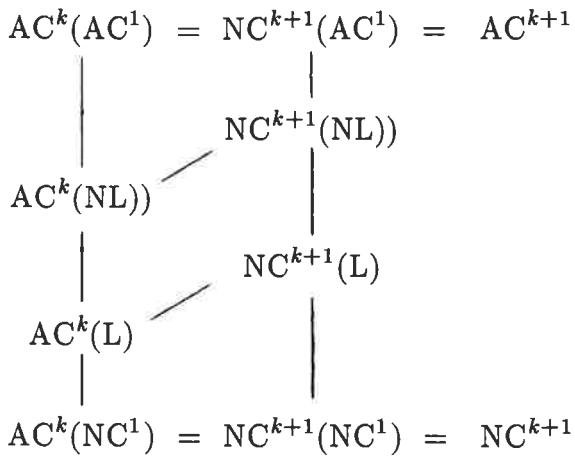
As soon as we allow oracle gates to appear in the circuits, we can consider the relativized classes NC^k and AC^k as reducibilities. Furthermore, the nonadaptive version of both NC^1 and AC^0, denoted by NC^1_1 and AC^0_1, are of special interest. They are defined as the class of functions computed by (uniform) AC^0 or, respectively, NC^1 circuits that have *at most* one oracle node on each path from an input node to an output node.

The study conducted by Wilson of the structural properties of the NC and AC hierarchies includes an analysis of these reducibilities. As stated in the introduction, Wilson has shown that AC^k and NC^{k+1} reducibilities coincide if applied to classes in the NC or AC hierarchy. This can be now formally stated as follows:

Theorem 1. [18] It holds for all $k \geq 0$, $i \geq 1$:

$$AC^k(NC^i) = NC^{k+1}(NC^i) = NC^{k+i};$$
$$AC^k(AC^i) = NC^{k+1}(AC^i) = AC^{k+i}.$$

Wilson has left open whether AC^k and NC^{k+1} reducibility coincide for other classes as well, for example, for classes lying "between" NC^1 and AC^1, like L and NL. With respect to these classes only the following inclusion relations were known to hold for all $k \geq 1$:



We will later refine this picture, by showing the equalities $NC^{k+1}(L) = AC^k(L)$ and $NC^{k+1}(NL) = AC^k(NL)$, and by furthermore characterizing these two classes in terms of log space bounded Turing machines with bounded number of "adaptive" queries to a functional oracle.

3. Nonadaptiveness

In this section we treat one of the two functional oracle computational models described in the preliminaries: the case in which the oracle answer is provided in the oracle tape, overwriting completely the query. This model will be shown to bear close resemblance to the set oracle model, and therefore we will use the same notation for the classes defined in both ways, the difference lying only in whether the class of oracles is a class of functions or a class of sets.

Using that model, we define the class $FL(\mathcal{F})$ as the class of functions computable within logarithmic work space, using an oracle function from $\mathcal{F}$; we insist that the same tape is used for creating the oracle queries and reading the oracle answers. Thus after a query, the answer can be read as many times as desired but must be completely erased before starting to write a new query.

Note that this model yields the same class, whether the answers are read once (from left to right) or more often. This is because the base machine before going to the next query can ask each query q polynomially often by simply storing the configuration which "induced" q.

Neither the oracle tape nor the output tape are affected by the space bound. The functions in a class $FL(\mathcal{F})$ can be considered reducible to functions in $\mathcal{F}$; we will refer to this kind of reducibility as *functional log space Turing reducibility.*

The fact that during the construction of a query the only information available is that of the work tapes implies a certain weakness of this model; more precisely, it is non-adaptive, in the sense that the answers obtained for previous queries cannot be relevant for the construction of forthcoming ones; this lack of adaptiveness makes this model weak. We will show this by proving that the queries might be asked simultaneously, and still get the same computational power. A fact analogous to this one arises when logarithmic space is used to compute Turing reductions between languages, since in this

case the Turing and truth-table reducibilities coincide [14]. However, the analogy is not complete since the equivalence between truth-table reducibility and Turing reducibility with logarithmically many queries that sometimes holds for sets does not seem to hold for functions (see [13], [2]).

Define the class $\mathrm{FL}_{\|}(\mathcal{F})$ as follows: The oracle is called only once, but given an input x of length n, a polynomial number $p(n)$ of arguments $x_1, \ldots, x_{p(n)}$ are passed to it simultaneously, and as answer a concatenation of the values of the function on each argument is obtained. The mode of operation can be formalized by using as functional oracle a specially structured function $f : (\{0,1\}^*\$)^n \to (\{0,1\}^*\$)^n$, and corresponds to the view of truth-table reducibility between languages as one round of queries made in parallel to an oracle set. This reducibility is clearly nonadaptive, since the answers of the first queries are not known when the next queries are prepared.

Note that like the model for $\mathrm{FL}(\cdot)$, the model for $\mathrm{FL}_{\|}(\cdot)$ also yields the same class of functions, whether the answers (essentially given as a polynomially long 0-1 string) are read once or more often, because the sequence of queries can be replicated a polynomial number of times.

Another clearly nonadaptive restriction of functional log space Turing reducibility $\mathrm{FL}(\cdot)$ is given by restricting the number of questions to 1 only. This reducibility, denoted by $\mathrm{FL}_1(\cdot)$, is similar to the "metric" reducibility defined for polynomial time in [13], since it holds:

$$f \in \mathrm{FL}_1(g) \iff \exists h_1, h_2 \in \mathrm{FL} \quad \text{such that} \quad f(x) = h_1(x, g(h_2(x))) \quad \text{for all } x.$$

Since no query depends of previous answers, the reducibilities $\mathrm{AC}^0_1(\cdot)$ and $\mathrm{NC}^1_1(\cdot)$ defined in the preliminaries are also nonadaptive.

The following result proves the nonadaptiveness of $\mathrm{FL}(\cdot)$.

Proposition 2. Let $\mathcal{F}$ be a class of functions. Then the following statements are equivalent:

(i) $f \in \mathrm{FL}(\mathcal{F})$;
(ii) $f \in \mathrm{FL}_{\|}(\mathcal{F})$;
(iii) $f \in \mathrm{FL}_1(\mathrm{AC}^0_1(\mathcal{F}))$,
i.e., $f(x) = h_1(x, g(h_2(x)))$ for some $h_1, h_2 \in \mathrm{FL}$, and $g \in \mathrm{AC}^0_1(\mathcal{F})$;
(iv) $f \in \mathrm{FL}_1(\mathrm{NC}^1_1(\mathcal{F}))$,
(v) $f \in \mathrm{FL}(\mathrm{FL}(\mathcal{F}))$.

Proof. Let a "query inducing configuration" of a machine M computing a functional Turing reduction, be a configuration in which M starts to write the first symbol of a new query. W.l.o.g. we can assume that such configurations are specially marked, e.g. by requiring them to be in a special "start-query" state, and are thus easily generable with a logarithmic resource bound. Note that for a given query inducing configuration, its position in the lexicographic order of all the query inducing configurations can be computed with logarithmic space.

(i) $\Longrightarrow$ (ii) Let $f \in \mathrm{FL}(g)$ with $g \in \mathcal{F}$ be computed by the machine M. Then construct in lexicographic order all query inducing configurations and, for each one, the corresponding induced queries; then ask them all in parallel. Clearly, with the

information of all answers to possible queries, the computation of M on x can now be completely simulated by referring to the ith answer if M queries a string w induced by the ith query configuration. Thus, $f \in \mathrm{FL}_{\|}(g)$.

(ii) $\Longrightarrow$ (iii) Let $f \in \mathrm{FL}_{\|}(\mathcal{F})$ be computed by a machine M with parallel functional oracle $g \in \mathcal{F}$. We assume for the moment that each of the parallel queries has size $p(|x|)$ for a polynomial p and that there are exactly $p(|x|)$ many of them, given an input of length n. Then clearly, M is a machine which poses only 1 query, but to a functional oracle g' of the form

$$g' : (\{0,1\}^{p(n)})^{p(n)} \to (\{0,1\}^{q(n)})^{p(n)}$$

with $g'(w_1, w_2, \ldots, w_{p(n)}) = g(w_1), g(w_2), \ldots, g(w_{p(n)})$ such that each $g(w_i)$ has the same length $q(n)$ for a polynomial q.

Clearly, a log space uniform AC^0_1 circuit with functional oracle nodes for g can be constructed that computes g'. The circuit is trivially structured; for an input of length n^2 each consecutive group i of n inputs are the inputs to oracle node i, $1 \leq i \leq n$, the m outputs of oracle node i are the outputs $m(i-1)+1$ up to $m(i-1)+m = m \cdot i$ of the circuit. Thus, $g' \in \mathrm{AC}^0_1(\mathcal{F})$. In the case of the parallel queries not having the assumed format, the base machine can "pad up" each query to length $p(n)$ and pose additional "dummy" queries to match the format. The circuit then provides in parallel oracle nodes for each length up to $p(n)$, filters out the relevant parts of the input, and attaches them to the right oracle gate. The output again will be padded up to the format.

By decomposing the computation of a $\mathrm{FL}_1(\cdot)$ machine in three phases "computation before the oracle query", "the oracle query", "the computation after the oracle query", it can be shown generally that $f \in \mathrm{FL}_1(\mathcal{G})$ for any function class $\mathcal{G}$, if and only if there exist $h_1, h_2 \in \mathrm{FL}$, and $g \in \mathcal{G}$ such that $f(x) = h_1(x, g(h_2(x)))$. The functions h_1 and h_2 describe the computation of M from the start configuration up to the query (h_2), and from the query answer up to the end (h_1). Clearly, h_1 and h_2 are functions in FL.

(iii) $\Longrightarrow$ (iv) Immediate.

(iv) $\Longrightarrow$ (v) This holds, since for every function h, being $h \in \mathrm{NC}^1_1(\mathcal{F})$ implies $h \in \mathrm{FL}(\mathcal{F})$. To see this, consider a log space uniform NC^1 circuit with oracle nodes for $f \in \mathcal{F}$, which satisfies that there is at most one oracle node on each path from an input node to an output node. Then, clearly, a deterministic log space Turing machine M can evaluate each output gate of the circuit in a depth-first manner. Whenever an oracle gate, whose up to polynomially many input bits have to be evaluated, is encountered in the evaluation, M does so using its oracle tape to store these bits, and finally solves this gate by querying its oracle. Note that it seems critical here that there is at most one oracle gate on each path.

(iv) $\Longrightarrow$ (i) The idea of the proof is similar to that in the proof of the transitivity of log space reductions. It consists of a (quasi) step-by-step simulation of a machine M_f computing a function $f \in \mathrm{FL}(g)$ for $g \in \mathrm{FL}(h)$ and $h \in \mathcal{F}$, by a machine M' which queries directly h.

Let the machine computing g with the help of the functional oracle be M_g. The simulating machine M' starts to simulate M_f step-by-step, so that each output of M_f

will be output of M'. When M_f is not constructing a query, the simulation proceeds normally. But whenever M_f enters a query inducing configuration c, M' saves this configuration on its worktape, in order to be able to generate the coming query q_c arbitrarily many times, and goes on with the simulation. The induced query q_c is discarded.

The only moment in which the simulation cannot proceed is when M_f wants to read the ith bit of its oracle answer tape. Let us see how to find it: it is part of the answer of M_g on q_c, so M' starts simulating M_g. But q_c is never fully constructed; instead, whenever in the simulation M_g wants to read the jth bit of q_c, M' repeats again the relevant part of the simulation of M_f from the saved query inducing configuration c on, to find the jth bit of the induced query q_c. This process is iterated as many times as M_g needs an input bit before writing down its ith output bit. Then the computation of M_g can be aborted and the computation of M_f resumed, knowing which is the needed ith bit of the oracle answer. ∎

From this Proposition 2 it follows immediately that a function class is closed under log space functional Turing reducibility if and only if it is closed under log space many-one reducibility and $\mathrm{AC}^0_1(\cdot)$ reducibility. Actually, later in this paper we point out that a weaker, more technical condition can be substituted for the $\mathrm{AC}^0_1(\cdot)$ reducibility.

These closure properties are known to hold for many function classes contained in FP that contain FL.

Corollary 3. The following function classes are closed under $\mathrm{FL}(\cdot)$:

$$\mathrm{FL}, \quad \mathrm{FNL}, \quad \mathrm{NC}^i \quad \text{for } i \geq 2, \quad \text{and} \quad \mathrm{AC}^j \quad \text{for } j \geq 1.$$ ∎

This discussion allows us to trace an analogy between language classes and function classes, in the following sense. Assume that the language class $\mathcal{A}$ contains L, and take its closure under $\mathrm{FL}(\cdot)$. The resulting class $\mathrm{FL}(\mathcal{A})$ is in a sense "the" function class corresponding to the language class $\mathcal{A}$. We mean that endowing any of the classes mentioned in Corollary 3 with either oracle sets from $\mathcal{A}$ or with oracle functions from $\mathrm{FL}(\mathcal{A})$ yields the same function class.

Also, note that if $\mathcal{F}$ is taken to be the class of characteristic functions c_A for languages A in a language class $\mathcal{A}$, then Proposition 2 yields $\mathrm{FL}(\mathcal{A}) = \mathrm{FL}_{\|}(\mathcal{A})$. Therefore, these function classes with oracle sets behave in a manner similar to that of the analogous language classes.

The closure of the class NL under this functional reducibility $\mathrm{FL}(\cdot)$ yields a very interesting class of functions, the class FL(NL), which will play a very important role later on in this paper. To ease the reference to it we will use the shorthand FNL.

Indeed, this class is exactly the class of (single-valued) functions computable by nondeterministic log space Turing transducers which have for each input x at least one accepting computation and compute on all accepting computations the same output; see also [12], where this concept is used to define a notion of NL-printability. This is an easy consequence of the closure of NL under complementation, which allows us to replace direct nondeterministic simulations for the oracle calls in FL(NL) computations.

An interesting point is that the class of single-valued functions computed by nondeterministic polynomial time transducers does not seem to be so well-behaved, and

seems to lack an effective enumeration of nondeterministic polynomial time machines computing them. On the contrary, such an enumeration can be obtained for FNL using the characterization as FL(NL).

A second point in which FNL seems to differ from one of its natural polynomial time counterparts, namely FP(NP), is the adaptiveness property. FP(NP) is an adaptive class, but FNL is not, since in general the answers of the oracle cannot all be recorded (see Proposition 4 below).

The function class $\mathrm{NL}^* = \mathrm{NC}^1(\mathrm{NL})$ (see [9]) has been suggested also as a function class closely related to NL. Indeed, its restriction to 0-1 functions equals the class NL, as stated e.g. in [7], because an NL machine can evaluate the NC^1 circuit by computing the reduction in a depth first manner from the output gate. On the other hand, as discussed above, we advocate $\mathrm{FL}(\mathcal{A})$ as a kind of "functional analog" of a language class $\mathcal{A}$; in this sense, FNL is the generalization of the 0-1 function in NL to arbitrary functions. There is no conflict, as shown by the following:

Proposition 4. $\mathrm{FNL} = \mathrm{FL}(\mathrm{FNL}) = \mathrm{FL}_{\|}(\mathrm{NL}) = \mathrm{NC}^1_1(\mathrm{NL}) = \mathrm{NL}^*$.

Proof . The inclusion $\mathrm{FL}_{\|}(\mathrm{NL}) \subseteq \mathrm{NC}^1_1(\mathrm{NL})$ follows by the fact that $\mathrm{NC}^1_1(\mathrm{FNL}) = \mathrm{NC}^1_1(\mathrm{NL})$, since we can compute the individual bits of a function $g \in \mathrm{FNL}$ or $g \in \mathrm{FL}_{\|}(\mathrm{NL})$ in parallel using a set in NL. The inclusion $\mathrm{NL}^* \subseteq \mathrm{FNL}$ follows from the same argument that shows that $\mathrm{NL}^* \subseteq \mathrm{NL}$ for 0-1 functions, by letting the FNL machine evaluate each of the output bits in sequence. The other inclusions are immediate or direct consequences of Proposition 4. ∎

4. Adaptiveness

This section continues the study of functional oracles for space-bounded machines considering the second model presented in the preliminaries. In order to distinguish this model of reducibility from the functional log space Turing reducibility, we introduce a different notation.

Let $\mathrm{FL}_{\mathcal{G}}[\![\mathcal{F}]\!]$ be the class of functions computable by log space machines with separate oracle query and answer tapes and querying an oracle function from $\mathcal{F}$ with the number of queries bounded by $g(n)$ for a function $g \in \mathcal{G}$ (n is the length of the input). Here, then, the machine can create a query w on the question tape, go in a query state and will receive the answer $f(w)$ on the oracle answer tape; the oracle query tape will be blank again. While the machine is reading the answer of its last query, it can now already create the following query. This query will thus not only depend on the work tape configuration of the transducer, but possibly also on the last answer; and hence using this model we obtain an adaptive reducibility, in which each query may depend essentially on the answer obtained from the previous query. We will be mainly interested in $\mathcal{F}$ being one of the classes FL, FNL, NC^k, AC^k for $k \geq 0$, and the class $\{id\}$ containing just the identity function id, and $\mathcal{G}$ the class of polylogarithmic (i.e., $O(\log^k n)$) or polynomial functions.

Observe that in this model the logarithmic space bound no longer enforces a (polynomial) time bound; e.g., without a time bound and already with the identity function as functional oracle, any recursive enumerable function could be computed by this model.

Simply, because the oracle query and answer tape function as an additional (unlimited) storage.

To principally make the model able of characterizing complexity classes we have two choices: we could bound the size of the oracle tape by a function in the length of the input (an option considered in the last part of this article; it basically yields space classes), or we could bound the number of queries by a function $g(n)$ in the length of the input, which gives us at the same time an implicit bound on the size of the oracle tapes. Because of the logarithmic work tape bound, any query has length of only up to $p(m)$ when m is the length of the last query answer. This notion basically yields time classes.

With a polynomial number of questions, we can simulate any log space uniform (and hence polynomially sized) circuit computing a function:

Proposition 5. $\mathrm{FL}_{poly}[\![\{id\}]\!] = \mathrm{FL}_{poly}[\![\mathrm{FL}]\!] = \mathrm{FL}_{poly}[\![\mathrm{FNL}]\!] = \mathrm{FL}_{poly}[\![\mathrm{FP}]\!] = \mathrm{FP}$.

Proof . The inclusion $\mathrm{FL}_{poly}[\![\mathrm{FP}]\!] \subseteq \mathrm{FP}$ can be proved by a straightforward simulation. By the considerations above the length of each new query remains polynomial in the length of the input. For the inclusion $\mathrm{FP} \subseteq \mathrm{FL}_{poly}[\![\{id\}]\!]$ simulate a given FP transducer M step by step on a given input x with as many queries as the time bound of M. Each configuration of M of the computation on x can be constructed on the query tape given the last configuration on the answer tape. ∎

More generally, a polynomial number of questions is equivalent to functional polynomial time Turing reducibility $\mathrm{FP}(\cdot)$.

Proposition 6. Let $\mathcal{F}$ be a function class. Then it holds: $\mathrm{FL}_{poly}[\![\mathrm{FL}(\mathcal{F})]\!] = \mathrm{FP}(\mathcal{F})$.

In fact, the model turns out to be only interesting to distinguish functions classes *within* FP by looking at subpolynomial query bounds; for the case of polylogarithmic many queries we will get characterizations of circuit classes.

Several characterizations of parallel complexity classes by in a sense "sequential" models are based on the idea of *phases* of a computation: fragments of computations within which the operations are "nearly independent" of each other, but depend only on results obtained in previous phases (see [11]). Frequently this idea of phase indicates parts of the computations that can be parallelized, so that the number of phases in a computation corresponds to the parallel time required to simulate this computation (and vice-versa).

We will show that the adaptive queries to functional oracles can be understood as dividing the computation process in a kind of phases, in the sense that an unbounded fan-in circuit can be evaluated with as many queries as its depth (which measures the parallel time). Conversely, to simulate a logspace machine with functional oracle, a circuit with oracle gates needs a depth equivalent to the number of oracle queries provided that the oracle class has a certain computational power.

Thus, here we extend the idea that number of phases corresponds to parallel time, so that it encompasses as well, for certain oracles, the "number of queries" resource. We obtain new characterizations of the AC and NC classes that are very well suited to the work with oracles. These characterizations allow us to complete the work of [18], where the techniques did not work for oracle classes other than circuit classes.

We prove now how the depth of unbounded fan-in circuits corresponds precisely to number of functional queries under this model.

Theorem 7. $\mathrm{AC}^k(\,\mathrm{FL}(\mathcal{F})\,) = \mathrm{FL}_{\log^k}[\![\,\mathrm{FL}(\mathcal{F})\,]\!]$ for all $k \geq 0$.

Proof. For the inclusion from left to right, let the function h be computed by the log space uniform family $\{\alpha_n\}$ of AC^k circuits with oracle nodes for a function $f \in \mathrm{FL}(\mathcal{F})$. Then the circuit α_n for inputs of length n has depth $c \cdot \log^k n$ for a constant c. Recall that in the case of unbounded fan-in circuits, each oracle gate contributes 1 to the depth of the circuit.

We will show how α_n can be evaluated in exactly as many phases as its depth, using the oracle query and answer tapes of a log space Turing machine M with functional oracle g as temporary storage means to keep provisional results, i.e. the partially evaluated circuit. Let the *depth* of a gate be its distance to the input gates. Then, in phase i all the gates of depth at most i will be evaluated.

For this, the functional oracle g is chosen such that it reproduces the complete description of the circuit α_n, supplemented by the values of those oracle gates which are directly evaluable with the information so far obtained, i.e., those oracle gates, whose direct predecessors are already evaluated.

Thus the oracle g is the following function:

Input: $\langle\alpha_n\rangle\$\langle o_1, o_2, \ldots, o_m\rangle\$\langle x_1, x_2, \ldots, x_m\rangle$, where $\langle\alpha_n\rangle$ is a description of a circuit α_n with oracle nodes in f, where all of the gates up to a certain depth are evaluated, i.e., have their value attached, $\langle o_1, o_2, \ldots, o_m\rangle$ is a list of oracle gates of the circuit, and $x_1, x_2, \ldots, x_m \in \{0,1\}^*$ is a list of inputs to the oracle gates in the list, x_i being input for gate o_i;

Output: $\langle\alpha_n\rangle\$\langle o_1, o_2, \ldots, o_m\rangle\$\langle y_1, y_2, \ldots, y_m\rangle$, where $y_j = f(x_j)$ for all $1 \leq j \leq m$.

Note that $g \in \mathrm{FL}(\mathcal{F})$, since it has just to copy the first two parts and evaluate m many times the function f.

The base machine M does the following for a given input x of length n:

First, M constructs the complete circuit description $\langle\alpha_n\rangle$ on its oracle query tape by simulating the log space machine that produces α_n. It thereby attaches to each input gate its respective value by reading it from the input tape. Then M repeats the following basic computation phases for $i := 1$ to $c \cdot \log^k n$:

(i) It queries its functional oracle to provide on its answer tape the description of α_n partially evaluated up to depth $i-1$, and the output of all oracle gates of depth i.

(ii) It then produces a new description of α_n on the oracle query tape, where *all* gates of depth i are evaluated, and attaches a list of the oracle gates of depth $i+1$, followed by a list of the inputs to theses gates.

For (ii) M has to

- "update" the old circuit description with the values of the oracle gates of depth i, by appending this information at the end of the new description of the respective oracle gate;
- evaluate all "normal" gates of depth i, and append this information at the end of the new description of the respective gate;

- find those oracle gates whose inputs are now known, and create a list of all of them and a list of their inputs.

It is easy to see that log space suffices to perform these computations, when the oracle answer tape provides the information of the circuit already evaluated up to depth $i-1$ for all gates and up to depth i for the oracle gates. Note that the oracle answer tape has to be read more than once to obtain all the needed information.

The process of evaluation is repeated until $i = c \cdot \log^k n$. Then all gates are evaluated. The value $h(x)$ then can be produced on the output tape by simply reading out the values of the respective output gates in the appropriate order.

For the inclusion from right to left, let f be a function in $\mathrm{FL}_{\log^k}[\![\,\mathrm{FL}(\mathcal{F})\,]\!]$ computed by a log space Turing transducer M with $c \cdot \log^k n$ queries to a functional oracle $g \in \mathrm{FL}(\mathcal{F})$, for a constant c. Let the length of $f(x)$ for all x of length n be $q(n)$. W.l.o.g. M outputs nothing before the first oracle query.

Then a computation of M is (roughly speaking) composed of $c \cdot \log^k n$ computation phases between two oracle answers, which can be described by a function $h \in \mathrm{FL}(g)$ with

$$h(\langle x, v_i, out_i, a_i\rangle) := \langle x, v_{i+1}, out_{i+1}, a_{i+1}\rangle$$

where

- x is the input of length n;
- v_i and v_{i+1} are configurations of size $\log n$ (they include the work tape contents and input head position, but not the oracle tape contents or output tape contents); v_i is the configuration in which M is in when receiving the ith oracle answer; v_0 is the start configuration, and $v_{c\cdot\log^k n+1}$ is the (unique) halt configuration;
- out_i is the output of M up to configuration v_{i+1} (not v_i), (which is the same as the configuration M is in directly before query v_{i+1}), and out_{i+1} is the output produced by M up to configuration v_{i+2}, which has out_i as prefix; $out_0 = \lambda$; and
- a_i is the ith oracle answer.

Note that all of the parameters can be padded up such that they have a fixed length for each n, with the length of each out_i be always the length $q(n)$ of $f(x)$.

It is easily verified that $h \in \mathrm{FL}(g)$; and thus $h \in \mathrm{FL}(\mathcal{F})$.

Since the number of oracle queries of M is bounded by $c \cdot \log^k n$, we can construct an (essentially trivially structured) AC^k circuit consisting of a line of $c \cdot \log^k n$ functional oracle gates $o_1, o_2, \ldots, o_{c\cdot\log^k n}$ for the function h. Here the circuit input gates are the input gates of o_1, all the output gates of oracle gate o_i are input gates of oracle gate o_{i+1}, and some specified $p(n)$ outputs of the last oracle gate $o_{c\cdot\log^k n}$ are $f(x)$, the output of M in a computation on x. It is easy to see that the circuit is log space uniform. ∎

Observe that the proof shows that any function computable by an $\mathrm{AC}^k(\mathrm{FL}(\mathcal{F}))$ circuit of depth $c \cdot \log^k n$ for a constant c can be computed by a log space machine with $c \cdot \log^k n$ questions to an oracle in $\mathrm{FL}(\mathcal{F})$.

Theorem 7 implies that for any function class $\mathcal{F}$ closed under functional Turing reducibility $\mathrm{FL}(\cdot)$ it holds: $\mathrm{AC}^k(\mathcal{F}) = \mathrm{FL}_{\log^k}[\![\,\mathcal{F}\,]\!]$. In particular, we get:

Corollary 8. For all $k \geq 0$ it holds:
(i) $\mathrm{AC}^k(\mathrm{FL}) = \mathrm{FL}_{\log^k}[\![\,\mathrm{FL}\,]\!] = \mathrm{FL}_{\log^k}[\![\,\{id\}\,]\!]$;
(ii) $\mathrm{AC}^k(\mathrm{FNL}) = \mathrm{FL}_{\log^k}[\![\,\mathrm{FNL}\,]\!]$;
(iii) $\mathrm{AC}^k(\mathrm{NC}^i) = \mathrm{FL}_{\log^k}[\![\,\mathrm{NC}^i\,]\!]$ for $i \geq 2$;
(iv) $\mathrm{AC}^k(\mathrm{AC}^j) = \mathrm{FL}_{\log^k}[\![\,\mathrm{AC}^j\,]\!]$ for $j \geq 1$.

Proof. The first equalities in (i) to (iv) all follow from Theorem 7 with Corollary 3. The second equality in (i) holds, since any of the oracle gates of a $\mathrm{AC}^k(\mathrm{FL})$ circuit can be evaluated by the log space base machine (recall the proof of Theorem 7 for the inclusion $\mathrm{AC}^k(\mathrm{FL}) \subseteq \mathrm{FL}_{\log^k}[\![\,\mathrm{FL}\,]\!]$). Thus, here we even have $\mathrm{AC}^k(\mathrm{FL}) \subseteq \mathrm{FL}_{\log^k}[\![\,\{id\}\,]\!]$. ∎

To complete the characterizations, we show now that we can also characterize parallel classes defined by bounded fan-in circuits. As a corollary we will obtain a positive solution to Wilson's open problems mentioned in the introduction.

Theorem 9. For any $k \geq 0$ it holds:
(i) $\mathrm{NC}^{k+1}(\mathrm{FL}) \subseteq \mathrm{FL}_{\log^k}[\![\,\mathrm{FL}\,]\!]$;
(ii) $\mathrm{NC}^{k+1}(\mathrm{FNL}) \subseteq \mathrm{FL}_{\log^k}[\![\,\mathrm{FNL}\,]\!]$.

Proof. We will show (i). The proof for (ii) can be obtained analogously.

Let $f \in \mathrm{NC}^{k+1}(\mathrm{FL})$ be a function that is computed by the family $\{\alpha_n\}$ of circuits with oracle nodes for a function $g \in \mathrm{FL}$. As said in the preliminaries, we can assume with no loss of generality that the oracle g is a 0-1 function in FL. Each α_n has depth $c \cdot (\log n)^{k+1}$ for a constant c, and is generated by the log space Turing machine M_α on input 1^n. Suppose that M_α outputs the specification of each gate i as $(i, t_i, p_{i_1}, p_{i_2}, \ldots, p_{i_m})$, where t_i indicates the type of gate i, and the sequence $p_{i_1}, p_{i_2}, \ldots, p_{i_m}$ specifies the inputs of gate i. For normal gates, this sequence contains two values, the left and right inputs; for oracle gates, this can be up to polynomially (in n) many inputs, say n^d. Thus for any gate, each predecessor gate of any gate i can be recorded with $d \cdot \log n$ bits by simply referring to its index in the specification of the gate i produced by M_α.

Recall that the contribution of an oracle node i to the depth of the circuit is logarithmic on the number m of its inputs $p_{i_1}, p_{i_2}, \ldots, p_{i_m}$. Define the *weight* of a node i with m inputs to be $\log m$. Then any gate contributes to the depth of the circuit with its weight.

Define the *weighted depth* of a node i relative to a list of gates $l = (i_0, i_1, \ldots, i_n)$ as the maximal sum of the weights of all nodes on a path from a node of l to i that does not pass through any other node of l, counting the weight of i but not that of the node of l. Then, clearly, the weighted depth relative to the list of input nodes of all nodes in α_n is smaller than or equal to $c \cdot \log^{k+1} n$. Note that any path from l to i can be recorded with as many bits as the weighted depth of i relative to l by simply recording descriptions of all its gates *relative* to the description of i (see also [6]). This fact is crucial for the proof.

We will construct a log space bounded Turing transducer T with a functional oracle $h \in \mathrm{FL}$ that evaluates the $\mathrm{NC}^{k+1}(g)$ circuit in pieces of depth more or less logarithmic,

taking into account the contribution of the oracle nodes to the total depth.

The evaluation principle is similar to that of Theorem 4, where we described the evaluation of an unbounded fan-in circuit of depth $\log^k n$ with functional oracle nodes using as many phases as the depth. But now we are confronted with a bounded fan-in circuit of depth $\log^{k+1} n$. Thus we have to evaluate more than one "level" of gates in each phase.

T will evaluate α_n for a given input x using again its oracle query and answer tapes as temporary storage means. T will query $\log^k n$ times the functional oracle described below, always satisfying that after phase j all gates with weighted depth up to $j \cdot c \cdot \log n$ relative to the input gates are evaluated (and accessible as a partial result on the oracle answer tape).

The oracle query and answer tapes will contain during each phase a description of the partially evaluated circuit α_n, with gates of fan-in 1 and 2, and oracle nodes, which describes each gate i by $(i, t_i, p_{i_1}, p_{i_2}, \ldots, p_{i_m}, v_i)$, where i, t_i, and $p_{i_1}, p_{i_2}, \ldots, p_{i_m}$ are as above and the additional entry $v_i \in \{0, 1, +, ¿\}$ specifies a value 0 or 1 of the gate, marks the gate by + or labels it "not yet evaluable" by means of ¿. Note that for an input gate i the list $p_{i_1}, p_{i_2}, \ldots, p_{i_m}$ is empty, and v_i equals the ith bit x_i of the input $x = x_1 x_2 \ldots x_n$ of the circuit.

The functional oracle h is the following:

Input: A description of the partially evaluated circuit α_n such that the gates marked with $v_i = +$ build a subcircuit (with oracle nodes) of depth maximally $c \cdot \log n$;

Output: Another description of α_n where, for all gates i that were marked with $v_i = +$ in the input description, now v_i has the correct value, and the other labels do not change.

Initially, T produces on the oracle query tape the circuit description of α_n by simulating M, integrating the values of the input nodes for the input x, and marking all other gates with ¿. Since no gates are marked +, a query to the oracle just transfers this description to the answer tape.

Then T performs $\log^k n$ computation phases, each consisting of the following two processes:

(i) marking process:
given the last partially evaluated circuit description, found on the oracle answer tape, T plready evaluated;

(ii) evaluation:
a query to the oracle h yields another description of α_n, where all the marks + set during the marking process are substituted by the appropriate values of the gates.

Thus, each query furnishes T with the values of an additional set of gates: those receiving a + during the marking process, i.e. all gates with weighted depth up to $c \cdot \log n$ relative to the list of the gates that are already evaluated. Therefore, after j oracle queries, all gates with weighted depth up to $j \cdot c \cdot \log n$ relative to the input gates are evaluated in the description found on the oracle answer tape.

Hence the last description after $\log^k n$ phases contains the values of all gates, and T can now simply collect the values of the output gates to produce $f(x)$.

We claim the following:

1. The oracle function h belongs to FL. This can be verified with the same argument that proves that FL is closed under NC^1 reductions (see [9]).
2. The marking process can be performed within logarithmic space. This is shown by an argument that is implicit in the proof of $\mathrm{NC}^1(\mathrm{FL}) = \mathrm{FL}$ in [9] (or [6]). For each gate i, it has to be checked whether i has weighted depth $c \cdot \log n$ relative to a list of gates l. A sufficient and necessary condition for this is that *all* of the paths that can be specified relative to i by at most $c \cdot \log n$ bits reach a gate on the list l. This can be checked by marking out $c \cdot \log n$ bits on the tape and constructing for each gate i marked ¿ in the description each path upwards relative to i. If the stack overflows for one of the paths without reaching a gate in l then we leave the specification of i unchanged; if no overflow occurs, then we mark i substituting + for ¿. ∎

Observe that the statement of this theorem differs from Theorem 7 in the level of generality. A natural question is whether other classes can be substituted for FL and FNL in Theorem 9. Most of the proof would remain valid for an arbitrary class, but some hypothesis on $\mathcal{F}$ must hold in order to guarantee that the function h employed in the proof remains in the class of oracle functions, and that the marking process can be achieved. Closure under $\mathrm{FL}(\cdot)$ does not seem to suffice, since there may be several oracle gates of nonconstant fan-in linked together in the marked part of the circuit, and $\mathrm{FL}(\cdot)$ would be unable to perform the evaluating process. Closure under $\mathrm{NC}^1(\cdot)$ does not suffice since the marked part of the circuit has logarithmic depth but might not be uniformly generated in logspace. If however, as it seems, h can be computed by an log space uniform NC^1 family of circuits with oracle nodes for g, this theorem might hold for all classes $\mathcal{F}$ simultaneously closed under $\mathrm{FL}(\cdot)$ and under $\mathrm{NC}^1(\cdot)$.

From Theorems 7 and 9 we get several interesting corollaries. The first one is the affirmative answer to Wilson questions: indeed the reducibilities AC^k and NC^{k+1} coincide on the oracle classes L and NL. Recall from the preliminaries that $\mathrm{AC}^k(\mathrm{L}) = \mathrm{AC}^k(\mathrm{FL})$ and similarly for NC^{k+1} and/or FNL.

Corollary 10. Let *id* denote the identity function. Then it holds for all $k \geq 0$:
(i) $\mathrm{AC}^k(\mathrm{FL}) = \mathrm{NC}^{k+1}(\mathrm{FL}) = \mathrm{FL}_{\log^k}[\![\,\mathrm{FL}\,]\!] = \mathrm{FL}_{\log^k}[\![\,\{id\}\,]\!]$;
(ii) $\mathrm{AC}^k(\mathrm{FL}(\mathrm{NL})) = \mathrm{NC}^{k+1}(\mathrm{FL}(\mathrm{NL})) = \mathrm{FL}_{\log^k}[\![\,\mathrm{FNL}\,]\!]$. ∎

Note that for $k = 0$ we have:

$$\mathrm{AC}^0(\mathrm{FL}) = \mathrm{NC}^1(\mathrm{FL}) = \bigcup_c \mathrm{FL}_c[\![\mathrm{FL}]\!] = \mathrm{FL};$$

$$\mathrm{AC}^0(\mathrm{FNL}) = \mathrm{NC}^1(\mathrm{FNL}) = \mathrm{NL}^* = \bigcup_c \mathrm{FL}_c[\![\mathrm{FNL}]\!] = \mathrm{FNL}.$$

Another interesting consequence is that the $\mathrm{FL}_{\log}[\![\,\cdot\,]\!]$ operator is able to provide another analogy between the NC and the AC hierarchies, since both are built up in

exactly the same manner using this operator, and we obtain one or the other just depending on the starting class.

Corollary 11. For all $k \geq 0$ it holds:

$$\mathrm{NC}^{k+2} = \mathrm{FL}_{\log^k}[\![\,\mathrm{NC}^2\,]\!] = \mathrm{FL}_{\log^i}[\![\,\mathrm{NC}^j\,]\!], \quad \text{with } j \geq 2,\ i+j = k+2;$$
$$\mathrm{AC}^{k+1} = \mathrm{FL}_{\log^k}[\![\,\mathrm{AC}^1\,]\!] = \mathrm{FL}_{\log^i}[\![\,\mathrm{AC}^j\,]\!], \quad \text{with } j \geq 1,\ i+j = k+1;$$

and thus

$$\mathrm{NC} = \mathrm{AC} = \bigcup_k \mathrm{FL}_{\log^k}[\![\,\{id\}\,]\!].$$

In particular, we have

$$\mathrm{AC}^{k+1} = \mathrm{FL}_{\log}[\![\,\mathrm{AC}^k\,]\!] \quad \text{for all } k \geq 1, \text{ and}$$
$$\mathrm{NC}^{k+1} = \mathrm{FL}_{\log}[\![\,\mathrm{NC}^k\,]\!] \quad \text{for all } k \geq 2.$$

Proof. This follows with Theorem 7 and the results by Wilson [18] that for all $k \geq 0$ it holds $\mathrm{AC}^{k+1} = \mathrm{AC}^k(\mathrm{AC}^1)$ and $\mathrm{NC}^{k+2} = \mathrm{AC}^k(\mathrm{NC}^2)$. ∎

These results refine the picture presented before as follows:

$$\begin{array}{ccccccc}
\mathrm{AC}^k(\mathrm{AC}^1) & = & \mathrm{NC}^{k+1}(\mathrm{AC}^1) & = & \mathrm{FL}_{\log^k}[\![\,\mathrm{AC}^1\,]\!] & = & \mathrm{AC}^{k+1} \\
\vert & & \vert & & & & \\
\mathrm{AC}^k(\mathrm{FNL}) & = & \mathrm{NC}^{k+1}(\mathrm{FNL})) & = & \mathrm{FL}_{\log^k}[\![\,\mathrm{FNL}\,]\!] & & \\
\vert & & \vert & & & & \\
\mathrm{AC}^k(\mathrm{FL}) & = & \mathrm{NC}^{k+1}(\mathrm{FL}) & = & \mathrm{FL}_{\log^k}[\![\,\mathrm{FL}\,]\!] & = & \mathrm{FL}_{\log^k}[\![\,\{id\}\,]\!] \\
\vert & & \vert & & & & \\
\mathrm{AC}^k(\mathrm{NC}^1) & = & \mathrm{NC}^{k+1}(\mathrm{NC}^1) & = & \mathrm{NC}^{k+1} & &
\end{array}$$

Observe that although e.g., both the classes $\mathrm{NL}^* = \mathrm{NC}^1(\mathrm{NL})$ and the class $\mathrm{NC}^2(\mathrm{NL})$ have an characterization in our model, constant versus logarithmic number of oracle questions, the class NC^2 does not.

5. Variations of the adaptive model

In the previous sections we have contrasted two notions of functional oracle for logarithmic space-bounded machines, giving rise to the reducibilities $\mathrm{FL}(\cdot)$ and $\mathrm{FL}_{\log^k}[\![\,\cdot\,]\!]$. Among other things we have shown that a lot of the function classes $\mathcal{F}$ contained in FP, like e.g. $\mathcal{F} \in \{\mathrm{FL},\ \mathrm{FL}(\mathrm{NL}),\ \mathrm{AC}^1,\ \mathrm{NC}^2,\ \ldots\}$ are closed under the nonadaptive reducibility $\mathrm{FL}(\cdot)$, whereas the closure of these classes under the adaptive reducibility $\mathrm{FL}_{\log^k}[\![\,\cdot\,]\!]$ gives us new characterization of the circuit classes $\mathrm{AC}^1(\mathrm{FL})$, $\mathrm{AC}^1(\mathrm{FNL})$, AC^2, NC^3, ... as $\mathrm{FL}_{\log}[\![\,\mathrm{FL}\,]\!] = \mathrm{FL}_{\log}[\![\,\{id\}\,]\!]$, $\mathrm{FL}_{\log}[\![\,\mathrm{FNL}\,]\!]$, $\mathrm{FL}_{\log}[\![\,\mathrm{AC}^1\,]\!]$, $\mathrm{FL}_{\log}[\![\,\mathrm{NC}^2\,]\!]$, ... In this model the number of functional oracle questions can be considered a measure of parallel time.

We have stated all theorems for the functional case. An inspection of the proof techniques shows that all theorems of the last section also hold for the respective languages classes, $\mathrm{L}_{\log^k}[\![\cdot]\!]$. We want to discuss two variations of this latter model.

Explicit space bound on the oracle tapes

In the model $\mathrm{L}_{\log^k}[\![\cdot]\!]$ the oracle query and answer tape were not explicitly space bounded, but an implicit space bound was given by the fact that there is a polynomially bounded increase of the length of an oracle query to the next due to the logarithmic space bound of the base machine. Thus, for subpolynomial (e.g., polylogarithmic) bounds on the number of queries, and oracle queries to polynomial time computable functions, restricting the length of the oracle query and answer tape to a polynomial is no restriction as far as the class of functions computable is concerned. Let us denote a bound $f(|x|)$ for $f \in \mathcal{F}$ on the *length* of the oracle query and answer tape in a $\mathrm{L}_{\mathcal{G}}[\![\cdot]\!]$ reducibility (i.e., a bound $g(|x|)$ on the number of queries for a function $g \in \mathcal{G}$) by $\mathrm{L}_{\mathcal{G}}[\![\cdot]\!]_f$ or $\mathrm{L}_{\mathcal{G}}[\![\cdot]\!]_{\mathcal{F}}$. Denote by *poly* the class of all polynomials. Then we have shown e.g. that for all $k \geq 0$ it holds:

$$\mathrm{AC}^k(\mathrm{FL}) = \mathrm{NC}^{k+1}(\mathrm{FL}) = \mathrm{L}_{\log^k}[\![\mathrm{FL}]\!]_{poly} = \mathrm{L}_{\log^k}[\![\{id\}]\!]_{poly};$$

$$\text{and thus} \qquad \mathrm{AC} = \mathrm{NC} = \bigcup_k \mathrm{L}_{\log^k}[\![\mathrm{FL}]\!]_{poly} = \bigcup_k \mathrm{L}_{\log^k}[\![\{id\}]\!]_{poly}.$$

By switching the bound $\log^k$ on the number of questions and the bound *poly* of the length of the oracle tapes, we get the classes $\mathrm{L}_{poly}[\![\mathrm{FL}]\!]_{\log^k}$. These classes can be shown to be exactly the classes $\mathrm{SC}^k := DSPACE, DTIME(\log^k n, poly)$ of languages deterministically computable with $O(\log^k n)$ space and polynomial time; $\mathrm{SC} := \bigcup_k \mathrm{SC}^k$. Thus, variants of one and the same model allow us both to characterize the class NC as well as the class SC

Theorem 12. For all $k \geq 1$ it holds

$$\mathrm{SC}^k = \mathrm{L}_{poly}[\![\mathrm{FL}]\!]_{\log^k} = \mathrm{L}_{poly}[\![\{id\}]\!]_{\log^k}; \quad \text{and thus} \quad \mathrm{SC} = \bigcup_k \mathrm{L}_{poly}[\![\{id\}]\!]_{\log^k}.$$

Proof. For the inclusion from left to right, simulate an SC^k machine M on input x step by step using the identity function as oracle. The ith query posed then will be the ith configuration (input head position, work tape content, etc.) of size $\log^k |x|$ in M's computation on x.

The inclusion from right to left can be obtained by a straightforward simulation of the oracle machine. Clearly, with the bound $O(\log^k n)$ on the length of the oracle query and answer tapes and $O(\log n)$ space bound suffices here. ∎

Nondeterministic base machine

What computational power and properties has the nondeterministic variant $\mathrm{NL}_{\log^k}[\![\cdot]\!]$ of the adaptive model $\mathrm{L}_{\log^k}[\![\cdot]\!]$? This topic has been studied in [1]. Here, we want to

sketch some of the results. Two ways of attaching the oracle tape are usually distinguished in the case of nondeterministic base machines,

(i) attachment via Ladner-Lynch: the oracle tape can be written on nondeterministically, and the oracle machine is required to be polynomial time bounded; or

(ii) attachment via Ruzzo-Simon-Tompa: the oracle tape can be written on only deterministically (which implicitly yields a polynomial time bound).

It is known that these two kinds of oracle attachments, which we distinguish by the subscripts "ll" and "rst", in the case of language oracles seemingly show quite different behaviour. It holds e.g., $\mathrm{NL}(\mathrm{NL})_{ll} = \mathrm{NP}$, and $\mathrm{NL}(\mathrm{NL})_{rst} = \mathrm{NL}$.

For functional oracles, too, there seems to be a difference: Attaching an arbitrary function class contained in FP that contains the identity function via Ladner-Lynch yields exactly NP.

On the other hand, attaching function classes contained in FP as functional oracle via Ruzzo-Simon-Tompa yields subclasses of NP which neither seem to be contained in P nor to be as strong as NP. These subclasses turn out to be characterizable by other concepts, e.g., bounded amount of nondeterminism (see [10]).

6. Discussion

We have restricted our investigations to the circuit classes NC^k and AC^k and the respective reducibilities. Note that our results also are interesting for many circuit classes "lying in power between" AC^k and NC^{k+1}, e.g., the classes TC^k of functions computed by threshold circuits (see [15]). Counting the depth of oracle nodes in such circuits 1, as in AC circuits, Theorem 7 implies:

$$\mathrm{AC}^k(\mathrm{FL}(\mathcal{F})) = \mathrm{TC}^k(\mathrm{FL}(\mathcal{F})) = \mathrm{FL}_{\log^k}[\![\,\mathrm{FL}(\mathcal{F})\,]\!];$$

and with Theorem 9 we get for $k \geq 0$:

$$\mathrm{AC}^k(\mathrm{FL}) = \mathrm{TC}^k(\mathrm{FL}) = \mathrm{NC}^{k+1}(\mathrm{FL}) = \mathrm{FL}_{\log^k}[\![\,\mathrm{FL}\,]\!] = \mathrm{FL}_{\log^k}[\![\,\{id\}\,]\!]; \quad \text{and}$$
$$\mathrm{AC}^k(\mathrm{NL}) = \mathrm{TC}^k(\mathrm{NL}) = \mathrm{NC}^{k+1}(\mathrm{FL}) = \mathrm{FL}_{\log^k}[\![\,\mathrm{FNL}\,]\!].$$

Furthermore, some of the results presented here can be strengthened somewhat by using more technical conditions on the classes.

Say that a function class $\mathcal{F}$ is closed under *marked replication* if, for all $f \in \mathcal{F}$, the function $f_{repl} : (\{0,1\}^*\$)^+ \to (\{0,1\}^*\$)^+$ with

$$f_{repl}(w_1\$w_2\$\ldots\$w_m\$) := f(w_1)\$f(w_2)\$\ldots\$f(w_m)\$$$

is contained in $\mathcal{F}$. Similarly, closure under *marked concatenation* for a function class $\mathcal{F}$ holds if, for all $f_1, f_2 \in \mathcal{F}$, the function $conc_{f_1,f_2} : (\{0,1\}^*\$)^2 \to (\{0,1\}^*\$)^2$ with

$$conc_{f_1,f_2}(w_1\$w_2\$) := f_1(w_1)\$f_2(w_2)\$$$

is contained in $\mathcal{F}$. Let the *join* of two functions $f_1,\ f_2 : \{0,1\}^* \to \{0,1\}^*$ be the function $join_{f_1,f_2} : \{0,1\}^+ \to \{0,1\}^+$ with

$$join_{f_1,f_2}(w) := \begin{cases} f_1(w'), & \text{if } w = 1w', \\ f_2(w') & \text{if } w = 0w'. \end{cases}$$

A function class $\mathcal{F}$ is closed under *join*, if for all $f_1, f_2 \in \mathcal{F}$ the function $join_{f_1,f_2}$: $\{0,1\}^+ \to \{0,1\}^+$ is contained in $\mathcal{F}$.

Then it can be seen that closure under marked replication can be substituted for the closure under $\mathrm{AC}_1^0(\cdot)$ in Proposition 2.

Also, using these definitions, a slightly stronger version of Theorem 7 can be obtained as follows: Let $\mathcal{F}$ be a function class that contains FL and is closed under join, and under marked replication and marked concatenation. Then it holds:

$$\mathrm{AC}^k(\mathcal{F}) = \mathrm{FL}_{\log^k}[\![\mathcal{F}]\!] \qquad \text{for all } k \geq 0.$$

Clearly, any "reasonable" functional complexity class is closed under marked replication, marked concatenation, and join, and will, if it has some "minimal" computational power, include all functions computable with logarithmic space. However, we feel that the theorems as presented in the body of the paper cover all the interesting cases and have more natural hypothesis.

Acknowledgment

The authors are grateful to Prof. L. Ruzzo for his immediate and detailed explanation of certain closure results for NC^1 reducibility, and to Chris Wilson for helpful comments on an earlier version of this paper and stimulating discussions. The second author is indebted to Ron Book, for discussions in which he contributed insights on the relationship between queries and phases. All three authors are very grateful to the Deutsche Forschungsgemeinschaft, who supported the visit of the third author to Barcelona.

References

[1] C. Àlvarez; in preparation.

[2] C. Àlvarez; B. Jenner; A very hard log space counting class; Proc. of the 5th Structure in Complexity Conference, 1990, pp. 154–168.

[3] J.L. Balcazar, J. Díaz, J. Gabarró; Structural Complexity I; Springer-Verlag, Berlin, 1988.

[4] J.L. Balcazar, J. Díaz, J. Gabarró; Structural Complexity II; Springer-Verlag, Berlin, 1990.

[5] D.A. Mix Barrington, N. Immerman, H. Straubing; On uniformity within NC^1; Proc. of the 3rd Structure in Complexity Theory Conference, pp. 47–59.

[6] A. Borodin; On relating time and space to size and depth; SIAM Journal of Computing 6,4 (1977), pp. 733–744.

[7] A. Borodin, S.A. Cook., P. Dymond, W.L. Ruzzo, M. Tompa; Two applications of complementation via inductive counting; SIAM Journal of Computing 18,3 (1989), pp. 559–578.

[8] S. Buss; The Boolean formula value problem is in ALOGTIME; Proc. of the 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 123–131.

[9] S.A. Cook; A taxonomy of problems with fast parallel algorithms; Information and Control 64 (1985), pp. 2–22.

[10] J. Díaz; J. Torán; Classes of bounded nondeterminism; Math. Systems Theory 23 (1990), pp. 21–32.

[11] J-W. Hong; Computation: Computability, Similarity and Duality; Pitman, London, 1986.

[12] B. Jenner; B. Kirsig; Alternierung und Logarithmischer Platz; Dissertation (in German), Universität Hamburg, 1989.

[13] M.W. Krentel; The complexity of optimization problems; Proc. of the 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 69–76.

[14] R. Ladner, N. Lynch; Relativization of questions about log space computability; Math. Systems Theory 10 (1976), pp. 19–32.

[15] I. Parberry, G. Schnitger; Parallel computation with threshold functions; J. of Computer and System Sciences 36 (1988), pp. 278–302.

[16] W. Ruzzo; On uniform circuit complexity; J. of Computer and System Sciences 22 (1981),pp. 365–383.

[17] C.B. Wilson; Relativized NC; Math. Systems Theory 20 (1987), pp. 13–29.

[18] C.B. Wilson; Decomposing NC and AC; SIAM Journal of Computing 19,2 (1990), pp. 384–396. (preliminary version in 4th Structure in Complexity Theory Conference, 1989)