Math. Systems Theory 20, 13-29 (1987)

Mathematical Systems Theory

Relativized *NC*

Christopher B. Wilson

Department of Computer and Information Science, University of Oregon, Eugene, OR 97403, USA

Abstract. This paper introduces a notion of relativized depth for circuit families and discusses issues regarding uniform families of relativized circuits. This allows us to define a version of relativized *NC* and compare it under various oracles with relativized *L*, *NL*, and *P*. We see that NC_1 is properly contained in *L* if and only if there exists an oracle *A* such that NC_1^A is properly contained in L^A. There is an oracle *A* where the hierarchy collapses, $NC_1^A = NC^A$, and another where $NC_1^A \subset NC_2^A \subset \cdots \subset NC^A \subset P^A$. We then construct an *A* so that, for any *k*, NC_1^A contains a set not in $NSPACE^A(O(n^k))$, suggesting that the notion of relativized space is too weak or that of relativized depth is too strong.

1. Introduction

As with the notion of sequential computation, the study of parallel computation naturally leads to the study of the inherent intractability of problems with respect to their parallel solutions. In this paper we will look at complexity classes defined in terms of the classical measures on both sequential and parallel models of computation. By examining the structure of these classes, we can gain some insight regarding the nature of these types of computation.

Our parallel model is uniform Boolean circuit families, and in requiring fast parallel time we restrict the circuits to have poly-log depth. In addition, to ensure that the circuits in a family do not get too large, we restrict them to polynomial size. These restrictions yield the well-known complexity class *NC* (see [14] and [9]). The following containments are known:

$$NC_1 \subseteq L \subseteq NL \subseteq NC_2 \subseteq NC \subseteq P.$$

None of these containments is known to be strict. A goal of this paper is to use

the method of relativization to see what relationships are possible. Also, we will investigate what one could prove with a relativizable proof technique. For example, a proof of $L = NL$ will relativize, but $L \neq NL$ and several other relationships cannot be exhibited with relativizable proof techniques. In Section 2 we provide the reader with more complete definitions. There we will define formally relativized parallel computation, where the circuits have access to an arbitrary oracle set. The notion of *relativized depth* introduced is an interesting and natural complement to that of relativized size, found in [23]. Sections 3 and 4 discuss uniformity and what it means for a circuit family to be uniform in the presence of an oracle.

Section 5 shows to what extent known results will relativize. For example, the NC^A hierarchy is seen to be in P^A for any oracle A. Also, nondeterministic log-space relative to A, NL^A, is contained in NC_2^A for any A. We show that NC_1 is properly contained in log-space, L, if and only if there exists an oracle A such that NC_1^A is properly contained in L^A.

The results of Section 6 are mainly concerned with questions of circuit depth. There is an oracle A so that the NC^A hierarchy collapses—all levels are equal. On the other hand, there is an A so that NC_k^A is properly contained in NC_{k+1}^A for all k. As a corollary, one obtains that NC^A is properly in P^A. Surprisingly, we can construct an oracle so that, for any k, NC_1^A contains a set not in $NSPACE^A(O(n^k))$. A corollary of this, clearly, is an A where NL^A is properly in NC_2^A.

In summary, the main results to be covered are as follows:

(i) $NC_1 \subset L$ iff there is an A such that $NC_1^A \subset L^A$.
(ii) $\exists A,\ NC_1^A = NC_2^A = \cdots = NC^A = P^A$.
(iii) $\exists A,\ NC_1^A \subset NC_2^A \subset \cdots \subset NC^A \subset P^A$.
(iv) $\exists A, \forall k,\ NC_1^A - NSPACE^A(O(n^k)) \neq \emptyset$.

2. Circuts and Oracles

Our model of parallel computation will be the Boolean (or logical) circuit. We can think of a Boolean circuit as being an acyclic directed graph with labeled nodes representing gates of the type *and*, *or*, and *not*, computing the appropriate unary or binary function of the inputs. These gates have fan-in at most two. As the circuits will be used to accept *sets*, rather than to compute *functions*, they may have some n inputs, but only one output. The output will indicate whether the circuit accepts the given input string of n characters over $\{0, 1\}$.

The *size* of a circuit is the number of gates it contains. Alternatively, one may wish to count the number of edges, as we do later. This would at most double the size measure. The *depth* of a circuit is the length of the longest directed path from an input edge to the output. In general, we view the size as a measure of the hardware required and the depth as the parallel time needed for the answer to trickle out. One will agree that a circuit having depth which is a fixed power of the logarithm of the length of the input string is indeed very fast.

A set L is defined to be in $SIZE(s(n))$ if and only if there exists a circuit family $\{\alpha_n\}$ such that for all n, α_n accepts only those strings in L of length n and the size of α_n is bounded above by $s(n)$. Similarly, L is in $DEPTH(d(n))$ if and only if there is a circuit family as above, but the size restriction becomes a depth restriction, limiting each α_n to have depth bounded above by $d(n)$.

The way we compare complexity classes is through the use of oracles. Let A be some subset of $\{0, 1\}^*$. A computation is *relative to the oracle A* if we allow it to be determined in a single step whether an arbitrary string x is a member of the set A. This is referred to as *querying* the oracle. (Think of having a black box which can answer any oracle question.) In a sense, these oracles provide a sort of generalized computational setting. If one can show that a particular relationship between two complexity classes holds relative to some oracle, then one gains intuition as to the structure of those classes. Furthermore, a proof that the negation of that relationship is in fact the case without oracles must satisfy certain special properties. In particular, that proof must not relativize, that is, hold in the presence of an arbitrary oracle.

Thus, we will want to allow the circuits access to an oracle set. To accomplish this, we use the notion of an *oracle gate* or *node*. An oracle gate is a k-input, one-output gate which, on an input x of length k, will produce the value 1 on its output edge if and only if x is in the specified oracle set. The contribution of this node to the *depth* of the path on which it lies is $\lceil \log_2 k \rceil$. (A similar notion can be found, see [9], in an NC_1 reduction.) The *size* of the relativized circuit is the number of *edges* in the circuit. A motivation for these measures derives from a comparison with Turing machines. Consider a Turing machine writing down a string of length k. It takes k steps do so so and, if the machine is to keep track of its tape head, requires $\lceil \log_2 k \rceil$ workspace. This allows the correspondence of size with time and depth with space [8] to be valid for an oracle query as well. Some relativized comparisons of circuit size to sequential (Turing machine) time have been covered in [23]. Another type of relativized parallel computation will be found in [11]. Given the discussion above, it now makes sense to define the relativized classes $SIZE^A(s(n))$ and $DEPTH^A(d(n))$. $SIZE\text{-}DEPTH^A(s(n), d(n))$ is also clear, as in [14].

The sequential classes $TIME^A(t(n))$ and $NTIME^A(t(n))$ have been used many places, and are well established (see especially [3]). For these, a Turing machine has a separate write-only query tape upon which it writes the string to be queried to the oracle. After the query, the tape contents are erased. Measuring space is somewhat trickier. One can require the query tape to be subject to the space bound or allow it to be excluded. In both cases, undesirable things occur: in the former we can have a set not be accepted in log-space relative to itself (see also [2]); in the latter we might have nondeterministic log-space not contained in P [10]. Therefore we adopt the convention introduced in [17]. The oracle tape is not subject to the space bound, but a nondeterministic Turing machine must act deterministically while there is anything on the query tape. This eliminates the mentioned anomalies. We refer to this restriction as the RST restriction. We define $SPACE^A(s(n))$ and $NSPACE^A(s(n))$ as relativized space under the RST restriction.

The following can be shown, where L is $SPACE(\log n)$ and NL is $NSPACE(\log n)$.

Fact. *$L = NL$ if and only if, for all oracles A, $L^A = NL^A$.*

This result, from [22], is a slight generalization of a result appearing in [20] and later in [15].

In some proofs to follow, we let $\langle x, y\rangle$ denote an encoding of the strings x and y so that x and y are easily recoverable. Also, $|\langle x, y\rangle|$ should be polynomial in $|x|$ and $|y|$. In fact, it is easy to see that it can be linear in $|x|$ and $|y|$. A simple example would be to encode the bit 0 as 00, the bit 1 as 01, and the mark # as 11. Then let $\langle x, y\rangle = x \# y$. Hence, encoding and decoding are in NC_0. The encoding can be generalized to handle more than two strings, for example, $\langle x, y, z\rangle = \langle\langle x, y\rangle, z\rangle$. The symbol $\subseteq$ will denote set containment, while $\subset$ will denote *proper* set containment.

An *instantaneous description*, or *id*, of a Turing machine is a tuple giving the current state of the machine, the locations of the input and worktape heads, and the contents of the worktapes. When a Turing machine needs to make a query of an oracle, it enters a *query-id*. In a single step, it then enters an *answer-id* depending on what had been on the query tape. Thus, there can be two reachable answer-ids for each query-id. In any computation, the point at which the machine writes a character on an empty query tape is described by an id referred to as a *beg-write-id*. Under the RST restriction, the computation is deterministic until an answer-id is reached, at which point the query tape will again be empty.

3. Uniformity

As defined so far, there are no constraints on the complexity of building the circuit families. If the nth circuit in a family takes exponential time to construct, one really has not gained much. Furthermore, the fact that the nth circuit may be independent of the $(n-1)$th circuit would allow a linear size log-depth family to accept a nonrecursive set. It was this nonuniformity that was exploited in [23] to allow such a family to accept any set in $\Delta_2^{P,A}$ relative to A.

In order to avoid the uniformity issues from being decisive, we choose to make the classes we compare essentially equally uniform. One approach would be to make the Turing machine classes nonuniform [19], [14]. The approach we use is to make the circuit classes uniform [8], [14], [16]. The standard method to force this is to require that the transformation $1^n \to \overline{\alpha_n}$ be easy to compute on a Turing machine, where $\overline{\alpha_n}$ is an encoding over $\{0, 1\}$ of the circuit α_n. A reasonable encoding of a circuit α can be found in [16]. If α has k gates, including inputs, then $\bar{\alpha}$ is the k concatenated tuples (g, t, g_L, g_R), where gate number g is of type t and has left input from gate g_L and right input from gate g_R.

Definition. A circuit family $\{\alpha_n\}$ is *$s(n)$-uniform* if the transformation $1^n \to \overline{\alpha_n}$ (where $\overline{\alpha_n}$ is a string describing the circuit) can be performed in space $O(s(n))$ on a deterministic Turing machine.

A $U(s(n))$ preceding the name of a class indicates that the circuit families involved in defining that class must be $s(n)$-uniform.

A widely studied hierarchy of classes of uniform circuits, *NC* [14], involves simultaneous resource bounds, measuring the size and depth at the same time. This hierarchy is especially interesting not only because it contains a wide class of natural and interesting problems [9] but because it characterizes that which can be computed on fast, feasibly sized, constructible parallel models. It is most simply defined in terms of circuits for our applications here, but can just as well be defined on several other formal versions of parallel computation, such as alternating Turing machines [16].

Definition.

$$NC_k = \bigcup_{i \geq 0} U(\log n)\text{-}SIZE\text{-}DEPTH(O(n^i), O((\log n)^k)),$$

$$NC = \bigcup_{k \geq 0} NC_k.$$

(Note: There is a slight unorthodoxy in our notation here. We will write NC_k rather than the more commonly used NC^k to leave room for an oracle superscript.)

Often, these classes are defined as classes of functions rather than of sets, as is done here.

4. Uniformity with Oracles

If we wish to examine what can happen to *NC* under relativization, we must decide on what uniformity means in the presence of an oracle. This superficially seems a bit unusual as the introduction of relativization generally makes complexity measures nonuniform, since the oracle can be possible nonrecursive. But the issue is how the computational devices attain access to the oracle. Relativized Turing machines access the oracle in a uniform manner. Relativized circuits are still a nonuniform measure with respect to the oracle.

In computing the transformation $1^n \to \overline{\alpha_n}$, there are two obvious choices. We can allow the use of the oracle in the computation or forbid such use. From one point of view, we might want to exclude the use of the oracle. Uniformity is a notion independent of any oracle set. It is a property of the description of the circuits, not having anything to do with the set to which they are relative. On the other hand, though, we like to think of an oracle as providing a generalized computational setting. In this sense, any computation should have access to the oracle.

We will define uniformity both ways, preferring the exclusion of oracle calls in the computation of the transformation. We will refer to the allowance such use as being weakly uniform. It turns out that here the distinction does not really seem to matter. We generally choose the type of uniformity that will phrase the results in the strongest form.

Definition. A relativized circuit family $\{\alpha_n\}$ is $s(n)$-*uniform*, $U(s(n))$, if the transformation $1^n \to \overline{\alpha_n}$ can be done in $O(s(n))$-space by a deterministic Turing machine without the use of the oracle.

The family $\{\alpha_n\}$ is *weakly $c(n)$-uniform, $wU(c(n))$*, if the transformation $1^n \to \overline{\alpha_n}$ can be done by a deterministic Turing machine relative to the oracle with at most $O(c(n))$ calls to the oracle.

Notice that this weak version of uniformity is indeed not very uniform. There are no restrictions on the time or space required. Weak uniformity is also needed to allow certain step-by-step arguments to succeed. Here, it is not so much the uniformity of the circuit families that is required, but some limitation on how many of them there are. Typically, our oracles are constructed in a countable number of stages. Without any uniformity on the families, there would be uncountably many of them, ruling out a diagonalization. Moreover, the Turing machine constructor gives us an index to the family it constructs. Let us now define a relativized version of *NC*.

Definition. Let A be an oracle set.

$$wNC_k^A = \bigcup_{i,j\geq 0} wU(n^i)\text{-}SIZE\text{-}DEPTH^A(O(n^j), O((\log n)^k)),$$

$$NC_k^A = \bigcup_{i\geq 0} U(\log n)\text{-}SIZE\text{-}DEPTH^A(O(n^i), O((\log n)^k)),$$

$$wNC^A = \bigcup_{k\geq 0} wNC_k^A, \qquad NC^A = \bigcup_{k\geq 0} NC_k^A.$$

Notice that wNC^A stands for the weakly uniform version of NC^A. The polynomial query bound in the definition of wNC_k^A arises, intuitively, from the fact that if an $O(\log n)$ space bounded constructor were allowed to query an oracle, then it could ask polynomially many questions. Otherwise, no time or space bounds apply. Importantly, this means that there is not necessarily an effective enumeration of these machines and therefore of the weakly uniform circuit families. The enumeration could be made effective, however, by the addition of a recursive time or space bound to the constructors. This would allow the oracle constructed in the proof of Theorem 7 to be recursive, with only a slight weakening of the result.

5. Space Versus Depth

Now we are able to generalize a result by Borodin [8]. This is a straightforward adaptation of the earlier result.

Lemma 1. *Let $s(n), s: \mathbf{N} \to \mathbf{N}$, be a constructible monotonic function satisfying $s(n) \geq \log n$. Then for any oracle A,*

$$SPACE^A(s(n)) \subseteq NSPACE^A(s(n))$$
$$\subseteq U(s(n))\text{-}SIZE\text{-}DEPTH^A(2^{O(s(n))}, O(s(n)^2)).$$

Proof. The first inclusion follows directly from the definitions. The second follows from an adaptation of the proof by Borodin [8].

Think of the id's of a nondeterministic computation as the nodes of a directed graph and the state transition function as inducing edges. Unrelativized, there would be $N = 2^{O(s(n))}$ id's. The transitive closure could then be computed in uniform depth $\log^2 N$ in polynomial in N size.

Under the RST restriction, the segment from any beg-write-id to the unique query-if following it is deterministic, and thus can be viewed as an $s(n)$-space computable function. Since the $s(n)$-space deterministic functions are already known to be in $U(s(n))$-$SIZE$-$DEPTH(2^{O(s(n))}, O(s^2(n)))$ by [8], we can precompute this segment. That is, for each beg-write-id, we can compute with an appropriately sized circuit the next query-id and the query, make the query, giving us the appropriate answer-id. Now the situation is as in the unrelativized case. The possible computation can be viewed as a directed graph. And we have precomputed the single edge from each possible beg-write-id to the appropriate answer-id. □

Hence, we have a generalization of a result mentioned in [9].

Corollary 2. *For any oracle A,* $NL^A \subseteq NC_2^A$.

It is not necessarily true that $U(s(n))$-$DEPTH^A(s(n))$ is contained in $SPACE^A(s(n))$ as is known to be true in the unrelativized case. Theorem 8 will illustrate how this can occur. For this model, the best one can do is bound a circuit family's size by Turing machine time, independent of the family's depth.

Lemma 3. *Let* $t(n)$, $t: \mathbf{N} \to \mathbf{N}$, *be a constructible monotonic function satisfying* $t(n) \geq n$. *Then for all oracles A,*

$$U(\log t(n))\text{-}SIZE^A(t(n)) \subseteq TIME^A(t(n)^{O(1)}).$$

The proof of this is straightforward after [8], depending on the fact that the circuit value problem is in P, and remains so in the presence of an oracle. The result looks weak given what we know of the unrelativized case, but it cannot be stated any more favorably. We cannot improve the space bound below $t(n)^{O(1)}$, as a generalization of Theorem 8 would show. This is because one query may depend on $t(n)^{O(1)}$ other queries in the circuit: for example, if a query is constructed by concatenating the answer bits of $t(n)$ other queries. These outcomes must be written down on a worktape before finally being transferred to the query tape, thus forcing the bound.

This does at least show that the *NC* hierarchy cannot be too powerful.

Corollary 4. *For all* $k \geq 1$ *and oracles A,*

$$NC_k^A \subseteq U(\log n)\text{-}SIZE^A(n^k) \subseteq P^A.$$

What we would like to do now is exhibit the relativized separation of various complexity classes. For some, this is not very likely, as an oracle separating two classes might prove that they are indeed not equal. We can get such a result for NC_1 and L. Unfortunately, the fact that $NC_1 \subseteq L$ does not relativize, as we shall see in Theorem 8. But we can show the following.

Theorem 5. *$NC_1 = L$ if and only if, for all oracles A, $L^A \subseteq NC_1^A$.*

Proof. One direction is trivial if we let A be the empty oracle. So assume that $NC_1 = L$ and let A be an arbitrary oracle. We let S be a set in L^A and will show that $S \in NC_1^A$.

Since $S \in L^A$, it is accepted by some deterministic Turing machine M which operates in space $O(\log n)$. If we were to run M on an input x of length n, because M operates in log-space, there could be at most $O(n^k)$ answer-id's, where an answer-id is an id of the machine after a query has been made and the response received. Note that in an answer-id, the oracle tape is empty and the machine is in a *yes* or *no* state. Any id α may spawn a next query Q. We also known, again due to the $O(\log n)$ space restriction, that there is a polynomial $p(n)$ which bounds the length of the longest query on any input of length n. Our representation of Q will be padded out to $2p(n)$ bits. So the languages

$\{\langle x, \alpha, i\rangle$: the *length* of the query caused by α on input x is $\geq i\}$

and

$\{\langle x, \alpha, i, d\rangle$: the length of the query caused by α on input x is $\geq i$ and the ith bit is $d\}$

are in L, and, by assumption, in NC_1. So we could hook up $p(n)$ circuits to get an NC_1 circuit β computing the mapping $(x, \alpha) \to Q$. As a technical point, each bit of Q will be represented by two bits in the circuit: the first bit indicates whether the second is part of the query. For example, $Q = 010$ would be represented by $0 \cdots 0101110$.

Now consider the $O(n^k)$ answer-id's $\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{cn^k}$ (α_0 is the initial id, which generates the first query). For each of these α_i, we can construct an NC_1 circuit which, with x, will output the next query Q_i that will be made after the answer-id α_i. This we do by fixing the appropriate inputs x and α_i to β.

Let us define a nonrelativized version of S, $\hat{S} = \{\langle x, y\rangle | M$ accepts x using y as an oracle string—if answer-id α leads to a query Q, then the answer to $Q \in A$ is interpreted as the αth bit of $y(|y| = c|x|^k)\}$. Since $\hat{S}$ is in L, it is also, by assumption, in NC_1. So there is an NC_1 circuit γ accepting it. If the string y is appropriately fixed, then γ can be used to accept S.

To construct an NC_1^A circuit for S, we describe three levels. The first level is cn^k copies of β, each with one of the answer-id's $\alpha_1, \ldots, \alpha_{cn^k}$ fixed as input and all having x as an argument. This level has depth $O(\log n)$ as β is an NC_1 circuit.

Intuitively, the second level simply consists of queries to the oracle of each of the outputs of β from the first level. As each of those has length at most polynomial in n, the depth of this level is $O(\log n)$. However, the query does not have a fixed length, so there will have to be $p(n)$ query nodes set up in parallel. The rightmost k second bits get sent to the query node of size k. The first bits of the pairs tell us the length and, thus, from which query node to take the answer. This extra processing takes only log n depth. For the sake of convenience, we may assume that the query nodes have the ability to handle queries of varying length.

The third and bottom level is the circuit γ with input x and y restricted to the outputs of the queries made at the second level. That is, the αth bit of y is restricted to be the answer to the αth query at the second level, which in turn is the answer to the next query M would make when in answer-id α on input x. And as we know that γ is an NC_1 circuit, we have that this level has depth $O(\log n)$. □

The circuit layout is as shown in Figure 1.

The total depth of this circuit accepting S is $O(\log n)$, and it also has polynomial size. Uniformity is seen in the fact that to construct it, all one must do is generate β and γ—both can be done uniformly—generate all possible answer-id's $\alpha_1, \ldots, \alpha_{cn^k}$, and describe some fairly trivial connections.

Corollary 5.1. *$NC_1 \subset L$ if and only if there exists an oracle A such that $L^A - NC_1^A \neq \emptyset$.*

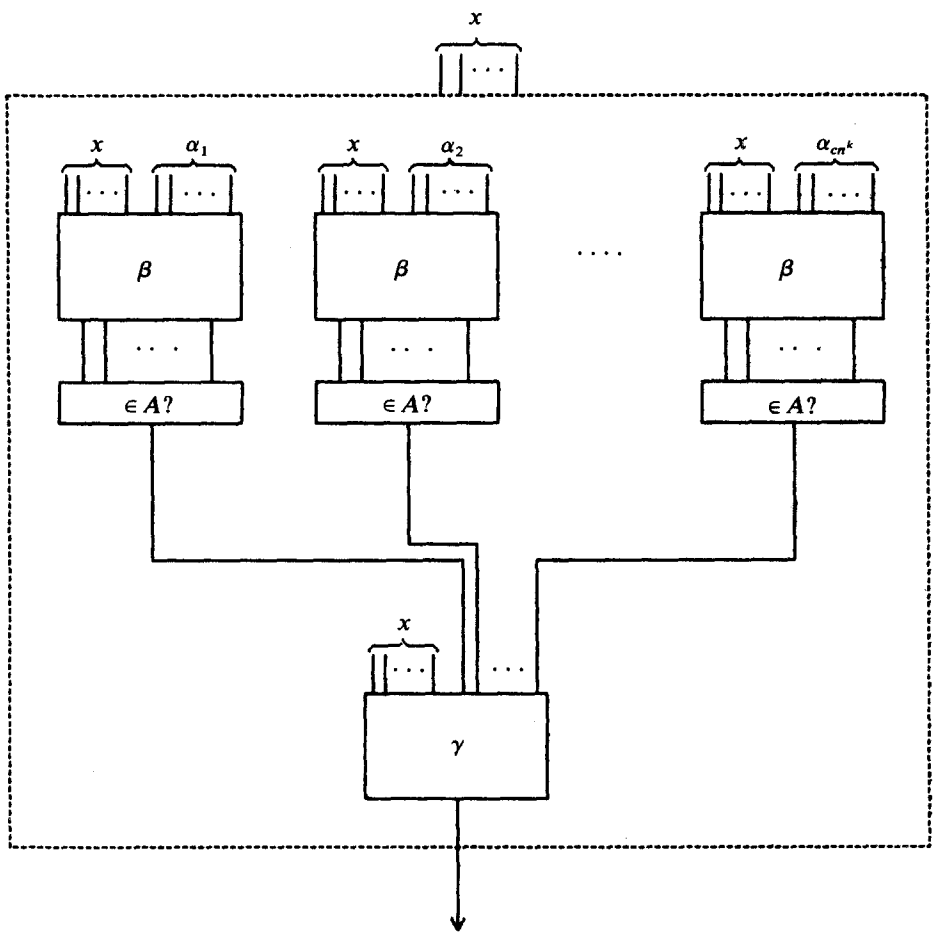

Fig. 1

Corollary 5.2. *$NC_1 \subset L$ if and only if there exists an oracle A such that $NC_1^A \subset L^A$.*

The second corollary is in a more pleasing form. The first points out an interesting approach to separating NC_1 from L. If we could construct a single oracle A for which L^A contains one set not in NC_1^A, then we have a proof that $NC_1 \subset L$. This situation is similar to that mentioned earlier regarding L and NL. There we saw that if one had an oracle A such that $L^A \subset NL^A$, then one had a proof that $L \subset NL$. In both these cases, NC_1 versus L and L versus NL, a proof of equality can relativize, because in that case no oracle witnessing inequality can exist. Notice that these results are similar to the approach of positive relativization taken in [6] and [7].

A natural oracle one would want to build is an A such that $NC_1^A \subset NL^A$. This, however, would be a proof that $NC_1 \subset L$ or $L \subset NL$, due to Theorem 5 and the similar relationship of L and NL.

6. Depth

The NC hierarchy is not known either to collapse or to have full structure. In this section we will examine what relationships are possible and see what we could hope to prove with standard techniques. We can, for example, show a relativized collapse of this hierarchy, implying that any proof of separation must be unrelativizable.

Theorem 6. *There is an oracle A such that $NC_1^A = L^A = NP^A$.*

Proof. The proof of this is straightforward. Let $M_1, M_2, \ldots$ be an enumeration of the nondeterministic Turing machines with polynomial time bounds. We define a complete set for NP:

$$K(A) = \{\langle i, x, 0^n\rangle : M_i^A \text{ accepts } x \text{ in at most } n \text{ steps}\}.$$

As in [3], A can be constructed so that

$$y \in K(A) \Leftrightarrow y0^{|y|} \in A.$$

For this A, then, $S \in NP^A$ implies that both $S \in L^A$ and $S \in NC_1^A$. Let us briefly argue the latter conclusion. A string x is in S if and only if $\langle i, x, 0^{p_i(|x|)}\rangle 0^m \in A$, $m = |\langle i, x, 0^{p_i(|x|)}\rangle|$, where M_i accepts S in polynomial $p_i(n)$ time. The evaluation of the polynomial $p_i(n)$ will only need to be done in $O(\log n)$ space by the machine which constructs the circuit α_n. The circuit α_n consists of the single query $\langle i, x, 0^{p_i(|x|)}\rangle 0^m$ as above, and has both polynomial size and $\log n$ depth. □

We can also exhibit full structure in the NC hierarchy, as has recently been shown for the polynomial hierarchy [24].

Theorem 7. *There exists an oracle A such that*

$$NC_{k+1}^A - wNC_k^A \neq \varnothing, \qquad \forall k \geq 1.$$

Proof. First we describe a language $L_{k+1}(A)$ which for all A is in NC^A_{k+1}. What follows is an algorithm to accept it, after which we can convince ourselves that a circuit family of the appropriate size and depth will accept it as well.

Input x, $|x| = n$
  $K \leftarrow \lceil \log^{k+1} n \rceil$
  do $i \leftarrow 1$ to $K-1$
    if $x0^{n-i}1b_{i-1}\cdots b_1 \in A$ then $b_i \leftarrow 1$
      else $b_i \leftarrow 0$
    end
  if $x0^{n-K}1b_{K-1}\cdots b_1 \in A$
    then accept
    else reject

If we build the obvious (but wrong) circuit which copies the given sequential algorithm, we would find that each of the $\lceil \log^{k+1} n \rceil$ queries would incur $\log(2n)$ depth, resulting in an $O(\log^{k+2} n)$ depth circuit. But a circuit could determine $\lceil \log n \rceil$ bits b_i at a time by testing all, polynomial in number, possibilities in $O(\log n)$ depth. Only $O(\log^k n)$ such levels would be required. The construction of the circuit is both uniform and independent of the oracle.

As usual, the oracle will be constructed in stages. A stage will be indicated by a pair $\langle k, i \rangle$. At stage $\langle k, i \rangle$ we will ensure that the ith wNC^A_k circuit family cannot accept $L_{k+1}(A)$. The ith wNC^A_k circuit family will be the one described by the deterministic Turing machine $M^A_{\langle k,i \rangle}$ (the enumeration of which, we recall, is not necessarily effective without some sort of additional constraint). Let us consider the circuit α constructed by $M^A_{\langle k,i \rangle}$ on an arbitrary 1^n—recall that it makes at most n^c oracle calls, for some integer c.

We need to partition α into levels. For some string x, fixed later, let us consider only those queries made by α of the form xz, where $|z| = n$. The circuit α is partitioned into *independent query levels* as follows: queries at the first level are all queries that depend on no other queries, queries at the second level each depend on some query at the first level, and so on. In general, then, any query at level m will depend on some query at level $m-1$ (and possibly lower numbered levels as well), but it will not depend on any query at levels numbered m or higher.

Notice that each query node in level m can be affected by a node in level $m-1$, for otherwise it would be in level $m-1$ by definition. So there is a path to each m level node from some $m-1$ level node. Hence, α can have at most $O(\log^{k-1} n)$ levels. Otherwise there would be a path containing $\omega(\log^{k-1} n)$ query nodes, each of which incurs depth $\lceil \log n \rceil$. The depth of α would then be $\omega(\log^k n)$, contradicting the fact that it is a wNC^A_k circuit. So, for some d, α can have at most $d \log^{k-1} n$ levels.

During the construction, we assume that if a string is queried which has not been specifically assigned to either A or $\bar{A}$, it will be assumed to be in $\bar{A}$. We need not go to the effort of assigning these strings to $\bar{A}$ since any subsequent stage will only add to A strings which are longer than anything queried at the current stage. We should also note that in the level decomposition of a circuit α, strings can be queried between levels, before level 1, and after level m. But

we are concerned with strings only of a particular form, xz, so these other strings queried will be assumed to be in $\bar{A}$. They will never be added to A.

Now we can describe the construction. Make n large enough so that it is larger than the length of any string queried at any earlier stage and $2^{\log^2 n/d}$ is larger than the sum of the size of α and n^c, the largest number of strings queried in the construction of α. Also ensure that 2^n is larger than n^c. Finally, let us assume that n is of the form 2^{dl} for some l.

Choose an x of length n such that for no z of length n was xz queried during the construction of α. This is possible since $2^n > n^c$. In the construction, no string relevant to the membership of x in $L_{k+1}(A)$ was queried. So establishing whether x is to be in $L_{k+1}(A)$ by adding strings of the form xz to A will not affect the behavior of the machine constructing α. Therefore, α will be unaffected by any later decision that may put x into $L_{k+1}(A)$.

At this point we proceed in steps. Set the input to α to be x. At each of the $d \log^{k-1} n$ steps, one step for each level, we will fix

$$\frac{\log^{k+1} n}{d \log^{k-1} n} = \frac{\log^2 n}{d}$$

bits of the final string put in A. Step m proceeds as follows:

Let $i = (m-1) \log^2 n/d$.
(Note: In the construction we will have ensured that no string of the form $xzb_i \cdots b_1$, $|z| = n - i$, is queried at any level 1 through $m-1$.)
There are $2^{\log^2 n/d}$ strings $x0^{n-|y|-i}yb_i \cdots b_1$, where $|y| = \log^2 n/d$. For some choice of y, the string is not queried at this or any previous level. Furthermore, it can have the property that no string of the form $xzyb_i \cdots b_1$, $|z| = n - |y| - i$, is queried at this or any previous level. Pick such a y and for $j = 1$ to $\log^2 n/d$, where $y = y_{\log^2 n/d} \cdots y_1$, put $x0^{n-j-i}1y_{j-1} \cdots y_1 b_i \cdots b_1$ into A if and only if $y_j = 1$.

Consider the strings placed into A at step m. By the note, which forms an inductive invariant, none of these strings will have been queries at any earlier level. We already know that the machine which constructed α will not have queried any of them. Possibly some of these strings will be queried at this level, but placing these strings into A here will affect only later levels, not other queries made at this level. The step of the construction proceed as described up through the last bit, at which point $x0^{n-K}1b_{K-1} \cdots b_1$, $K = \log^{k+1} n$, is put into A if and only if α with the oracle A rejects x.

It remains to show that placing this last string into A will have no effect on the construction of A so far. This we do by showing that it could not have been queried earlier. Suppose it was queried at level j. Where $i_{j-1} = (j-1) \log^2 n/d$ and $i_j = j \log^2 n/d$, let $y = b_{i_j} \cdots b_{i_{j-1}+1}$ and $z = 0^{n-K}1b_{K-1} \cdots b_{i_j+1}$. Then the string $xzyb_{i_{j-1}} \cdots b_1$ is queried at level j. This contradicts the way in which y was chosen at step j. No string of this form is queried at this level, for any z.

So at stage $\langle k, i \rangle$ we have ensured that the ith wNC_k^A family will not accept $L_{k+1}(A) \in NC_{k+1}^A$. Hence, the diagonalization is complete. □

Notice the strong use of uniformity here. There is a set in each uniform level not in the next lower weakly uniform level. Hence, both the uniform and weakly uniform hierarchies can have full structure. This allows us to change the definition of uniformity and know that there can still be full structure.

We could almost change Theorem 7 to separate NC_{k+1}^A from nonuniform NC_k^A. The uniformity of the lower level was hardly used. The problem is one of countability—there is an uncountable number of nonuniform polynomial-sized poly-log depth circuit families. So in a countable number of steps we would not be able to ensure that each family does not accept a particular language. The weak uniformity was used here only to guarantee that there would be a countable number of circuit families of concern.

Lemma 7.1. *Let A be the oracle constructed in the proof of Theorem 7. Then $P^A - wNC^A \neq \emptyset$.*

Proof. Suppose that $P^A \subseteq wNC^A$ for the A from the previous proof. In that proof, recall the series of sets $L_k(A)$ having the property that for all k, $L_{k+1}(A) \notin wNC_k^A$. Each $L_k(A)$ was seen to be in NC_k^A, so there exists a series of circuit families $\{\alpha_n^k\}$, the kth one being an NC_k^A family accepting $L_k(A)$. Let us define the language

$$S = \{\langle k, x\rangle : \alpha_{|x|}^k \text{ accepts } x \text{ relative to } A\}.$$

There is a polynomial time algorithm for S given in the proof of Theorem 7. However, the running time of this algorithm is $O(|x|\log^k|x|)$, which is not necessarily polynomial in the length of $\langle k, x\rangle$. It is, however, polynomial if $k = c\log|x|/\log\log|x|$, for then $O(|x|\log^k|x|)$ is $O(|x|^{c+1})$. So let

$$\hat{S} = \left\{\langle k, x\rangle \,|\, k \leq \frac{\log|x|}{\log\log|x|}, \langle k, x\rangle \in S\right\}.$$

The fact that $\hat{S} \in P$ follows. By our initial supposition, $\hat{S} \in wNC^A$, so there exists a k such that $\hat{S} \in wNC_k^A$. But, by using a wNC_k^A circuit family for $\hat{S}$, one can construct a wNC_k^A circuit family for $L_{k+1}(A)$ almost everywhere. This in turn, by means of building in a finite table, yields a wNC_k^A circuit family for $L_{k+1}(A)$, which we have seen to be impossible. Hence, $\hat{S}$ cannot be in wNC_k^A for any k. □

Corollary 7.2. *There is an oracle A such that*

$$NC_1^A \subset NC_2^A \subset \cdots \subset NC^A \subset P^A.$$

This follows from Theorem 7 and Lemma 7.1 taken with Corollary 4.

Corollary 7.3. *There is an oracle A such that*

$$wNC_1^A \subset wNC_2^A \subset \cdots \subset wNC^A.$$

There is a version of *NC* whose uniformity restriction is intermediate between $O(\log n)$-space and weakly uniform: polynomial time uniform *NC*, or *PUNC* (this notation was introduced by Allender [1]; see also [4]). The definition of *PUNC* and $PUNC^A$ can be taken as the same as for *NC* and NC^A with the modification that a description of the nth circuit α_n be constructible from 1^n in time polynomial in n. Theorem 7 and Corollary 7.1 apply to the *PUNC* hierarchy as well.

Corollary 7.4. *There is an oracle A such that*

$$PUNC_1^A \subset PUNC_2^A \subset \cdots \subset PUNC^A \subset P^A.$$

As mentioned earlier, the fact that $NC_1 \subseteq L$ is not always true in the presence of an oracle.

Theorem 8. *There is an oracle A such that for all* $k \geq 1$,

$$NC_1^A - NSPACE^A(O(n^k)) \neq \emptyset.$$

Proof. First we described a series of languages $S_k(A)$ which for all A are in NC_1^A. To determine if $x \in S_k(A)$, $|x| = n$, and we will assume that $n = 2^i$, look at all strings of the form $0^m y$ of length n where $|y| = (k+1)\log n = (k+1)i$. Query all $2^{(k+1)i} = n^{k+1}$ such strings and let z, $|z| = n^{k+1}$, be the bit string of answers. At the second level of the circuit, query z. If this string is in A, output 1 (accept), otherwise output 0 (reject). The size of the circuit is $n^{k+2} + n^{k+1} + 1$ and its depth is $(k+2)\log n$. That the uniformity condition is satisfied is clear.

Figure 2 shows an outline of a circuit accepting $S_k(A)$.

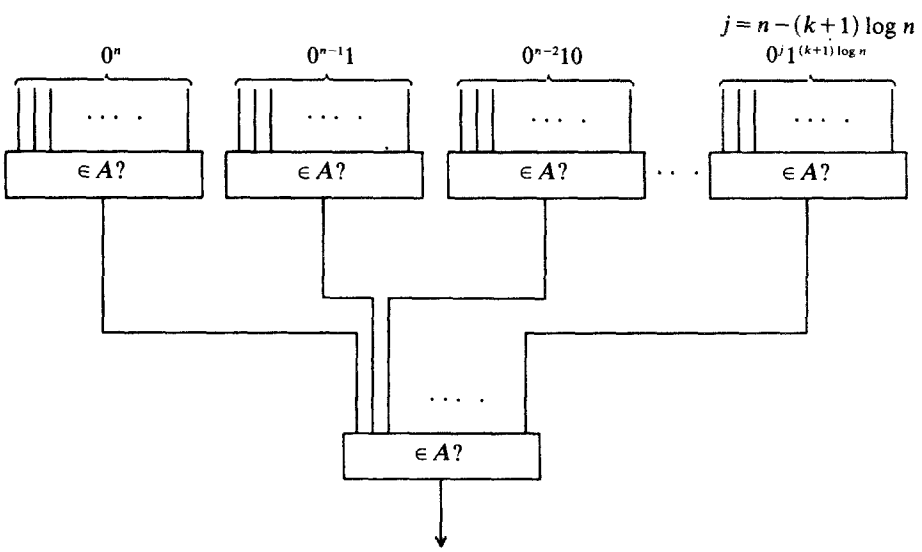

Fig. 2

We will construct an A so that $S_k(A)$ is not contained in $NSPACE^A(O(n^k))$ for each k. This is done is stages. Let $\{M_i\}$ be an enumeration of the nondeterministic Turing machines which operate under the RST restriction. Now at stage $\langle i, c, k\rangle$ of the construction, we ensure that M_i^A does not accept $S_k(A)$ within cn^k space. Without loss of generality, we may assume that $M_{\langle i,c,k\rangle}$ is the machine M_i restricted to cn^k space. In fact, we could choose that as the definition of the encoding $\langle i, c, k\rangle$.

Let us examine a machine M_i operating in space cn^k. Now there can be only $2^{O(n^k)}$ id's. So, in particular, there are at most 2^{dn^k}, for some d depending on i and c, *beg-write-id's*, id's where the machine begins to write on the query tape, and thus starts to act deterministically. This is true for all oracles. Each beg-write-id uniquely determines a query. So we can say, and this is a key point, that for a fixed x of length n there is some d such that

$$card\left(\bigcup_A \{y \mid M^A_{\langle i,c,k\rangle} \text{ queries } y \text{ on } x\}\right) \leq 2^{dn^k}.$$

Thus, there are only 2^{dn^k} possible queries over all oracles.

Construction of A

Stage 0: $A \leftarrow \varnothing$.

Stage $\langle i, c, k\rangle$: Let d be the constant such that on an input of length n there are at most 2^{dn^k} possible queries by $M_{\langle i,c,k\rangle}$. Choose n of the form 2^j large enough so that it is larger than the length of any string earlier queried and $2^{n^{k+1}} > 2^{dn^k}$. So there must exist a string z, $|z| = n^{k+1}$, which is not queried by M_i on 0^n for any oracle. Choose such a z. Put $0^m y$ of length n, $|y| = (k+1)\log n$, into A if and only if the yth bit of z is 1. Now run M_i^A on 0^n within space cn^k. Put z into A if and only if it rejects.

end construction.

Since M_i could not have queried z for any A, placing it into A cannot change the behavior of M_i^A on 0^n. Thus we have shown, for all i, c, and k, that there is an n such that $0^n \in S_k(A)$ if and only if M_i^A rejects 0^n within space cn^k. Notice that not only have we shown that $NC_1^A - NSPACE^A(O(n^k))$ can be nonempty, but that it can contain a tally set, which is a set of words over a single character. □

This implies that our model is not quite as natural as we had hoped. An open issue would be to devise a natural relativized model for which $NC_1 \subseteq L$ holds. However, Theorem 8 together with Corollary 2 does give us the following.

Corollary 8.1. *There is an oracle A such that $NL^A \subset NC_2^A$.*

7. Concluding Remarks

We have exhibited a number of new relativized relationships which involve circuit depth. These results imply specific properties about proofs which would purport to show their negation. In particular, we now see that questions involving parallel time may be difficult to answer in the same way that the classic complexity classes are difficult: both may require proofs which do not relativize.

A general open issue is a characterization of proof techniques and their ability to relativize. For example, allowing a particular type of access to the oracle may allow one to relativize some relationships but not others. The access allowed may allow a certain class of proof techniques to generalize to all oracles. Then we would know that this class of proof techniques would be insufficient to show the negation of the relativized relationships. A general theory would give some intuition into the techniques that might be required to prove interesting things, and would help us apply the many relativized results in a formal manner.

Fortunately, we do know that not all our proof techniques relativize. Well-known examples of this are found in [12], [5], and [13].

A *faithful* relativization can be considered a method of oracle access under which known results will generalize to all oracles. We know that using the RST restriction is faithful when considering relativized space alone, but saw that we encounter problems when comparing space with depth. In particular, the depth restricted devices were noticeably more powerful than their corresponding space restricted devices. To overcome this, we might either weaken the power of circuit depth, or probably easier, allow slightly more power to relativized space. One possibility might be to use an oracle stack, mentioned in [22], to store partially constructed queries.

An open question is whether we can, for example, get a partial collapse of the NC hierarchy. Does there exist an oracle A such that $NC_1^A \neq NC_2^A = NC^A$? Other oracles we would like to see are ones witnessing any of the following:

$NC_1 \subset NC_k = NC$ for some $k > 1$;

$NC_1 \neq NC_k \neq NC \neq P \neq PSPACE$ for all $k > 1$;

$NC = P \neq NP$;

$NC \neq P = NP$.

References

[1] E. Allender, The complexity of sparse sets in P, *Proceedings of the Structure in Complexity Theory Conference*, Springer-Verlag, New York, 1986, pp. 1-11.

[2] D. Angluin, On relativizing auxiliary pushdown machines, *Math. Systems Theory*, **13** (1980), 283-299.

[3] T. Baker, J. Gill, and R. Solovay, Relativizations of the $P = ?NP$ question, *SIAM J. Comput.*, **4** (1975), 431-452.

[4] P. Beame, S. Cook, and J. Hoover, Log depth circuits for division and related problems, *Proceedings of the 25th Symposium on Foundations of Computer Science*, 1984, pp. 1-6.

[5] N. Blum, A Boolean function requiring $3n$ network size, *Theoret. Comput. Sci.*, **28** (1984), 337-345.

[6] R. V. Book, T. J. Long, and A. Selman, Quantitative relativizations of complexity classes, *SIAM J. Comput.*, **13** (1984), 461-486.

[7] R. V. Book, T. J. Long, and A. Selman, Qualitative relativizations of complexity classes, *J. Comput. System Sci.*, **30** (1985), 395-413.

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

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

[10] R. Lander and N. Lynch, Relativizations of questions about log-space reducibility, *Math. Systems Theory*, **10** (1976), 19-32.
[11] P. Orponen, General nonrelativizability results for parallel models of computation, *Proceedings of the Winter School on Theoretical Computer Science*, 1984, pp. 194-205.
[12] W. Paul, A $2.5n$ lower bound on the combinatorial complexity of Boolean functions, *SIAM J. Comput.*, **6** (1977), 427-443.
[13] W. Paul, N. Pippenger, E. Szemeredi, and W. Trotter, On nondeterminism versus determinism and related problems, *Proceedings of the 24th Symposium on Foundations of Computer Science*, 1983, pp. 429-438.
[14] N. Pippenger, On simultaneous resource bounds (prelinary version), *Proceedings of the 20th Symposium on Foundations of Computer Science*, (1979), pp. 307-311.
[15] C. W. Rackoff and J. I. Seiferas, Limitations on separating nondeterministic complexity classes, *SIAM J. Comput.*, **10** (1981), 742-745.
[16] W. L. Ruzzo, On uniform circuit complexity, *J. Comput. System Sci.*, **22** (1981), 365-383.
[17] W. L. Ruzzo, J. Simon, and M. Tompa, Space-bounded hierarchies and probabilistic computations, *J. Comput. System Sci.*, **28** (1984), 216-230.
[18] W. Savitch, Relationships between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.*, **4** (1970), 177-192.
[19] C. P. Schnorr, The network complexity and the Turing machine complexity of finite functions, *Acta Inform.*, 7 (1976), 95-107.
[20] I. Simon, On some subrecursive reducibilities, Ph.D. dissertation, Stanford University, 1977.
[21] L. Stockmeyer, The polynomial time hierarchy, *Theoret. Comput. Sci.*, **3** (1977), 1-22.
[22] C. Wilson, Relativized circuit size and depth, Technical Report #179/85, Department of Computer Science, University of Toronto, 1985.
[23] C. Wilson, Relativized circuit complexity, *J. Comput. System Sci.*, **31** (1985), 169-181.
[24] A. Yao, Separating the polynomial-time hierarchy by oracles, *Proceedings of the 26th Symposium on Foundations of Computer Science*, 1985, pp. 1-10.

Received August 14, 1985, and in revised form September 1, 1986, and February 9, 1987, and in final form March 11, 1987.