CIS-TR 86-01
Parallel Computation and the NC Hierarchy Relativized[1]

Christopher B. Wilson

Department of Computer and Information Science
University of Oregon

[1]An earlier version of this paper will be presented at the *Structure in Complexity Theory Conference*, 1986.

Parallel Computation and the NC Hierarchy Relativized

Christopher B. Wilson

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

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. Our approach is to 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 a uniform bounded fan-in Boolean circuit family, and in requiring fast parallel time we restrict the circuits to have polynomial size and poly-log depth. These restrictions yield the well known complexity class NC (see [Pi79] and [Co83]). In section 2 we provide the reader with a more complete definition. And there we will formally define 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 [Wi85b]. 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. And 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 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 A 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.

Although the method of relativization has been applied quite successfully to classes defined in terms of Turing machine time ([BGS75], [BS76], [Ya85]), an appropriate definition of relativized Turing machine space has been difficult to derive. I. Simon [Si77] examines many different definitions, but the two methods that initially attracted the most attention were characterized by how the space bound was applied to the query tape. One could either subject it to the space bound or exclude it. The former may initially appear the most natural, though for certain A, $A \notin L^A$, certainly not a natural situation. On the other hand, if the query tape is excluded from the space bound, then for some A, $NL^A \not\subseteq P^A$, and Savitch's theorem [Sa70] will fail to hold [LL76]. Our goal is to find a measure of relativized space which retains as many known containments as possible.

Our inability to relativize statements such as $NC_1 \subseteq L$ while retaining $NL \subseteq NC_2$ is somewhat unappealing. One way to avoid this is to provide a more sophisticated definition of relativized space. Section 7 provides the definition of oracle stacks, upon which a space bounded TM can push partially constructed queries. sL and sNL become the oracle stack versions of relativized L and NL. Section 8 provides convincing evidence that this is, with respect to depth, a more reasonable measure of relativized space. We see that for any oracle set A, $NC_1^A \subseteq sL^A$ and $sNL^A \subseteq NC_3^A$. Proving that $NL^A \subseteq NC_2^A$ remains open.

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) $\exists A\ \forall k,\ NC_1^A - NSPACE^A(O(n^k)) \neq \emptyset$
(v) $\forall A,\ NC_1^A \subseteq sL^A$
(vi) $\forall A,\ sNL^A \subseteq NC_3^A$

2. Circuits 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 labelled 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 a measure of the parallel time. 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 one 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 [Co83], in an NC_1 reduction.) The *size* of the relativized circuit is the number of *edges* in the circuit. Some relativized comparisons of circuit size to sequential (Turing machine) time have been covered in [Wi85b]. Another type of relativized parallel computation will be found in [Or84]. More recently, Buss [Bu86] has devised a notion of relativized alternation. As alternating Turing machines can be used to simulate parallel processes [Ru81], this also yields an intriguing approach to relativized parallelism.

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 [Pi79]. The sequential classes $TIME^A(t(n))$ and $NTIME^A(t(n))$ have been used many places, and are well established (see especially [BGS75]). 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 [An80]); in the latter we might have nondeterministic log-space not contained in P ([LL76]). Therefore we adopt the convention introduced in [RST84]. 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.

So 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 2.1 $L = NL$ if and only if, for all oracles A, $L^A = NL^A$.

This result, from [Wi85a], is a slight generalization of a result appearing in [Si77] and later in [RS81].

In some proofs to follow, we let $<x,y>$ denote an encoding of the strings x and y so that x and y are easily recoverable. Also, $|<x,y>|$ should be polynomial in $|x|$ and $|y|$. In fact, it is easy to see that it can be linear in $|x|$ and $|y|$. The encoding can be generalized to handle more than two strings - for example, $<x,y,z> = <<x,y>,z>$.

3. Uniformity

As defined so far, there are no constraints on the complexity of building the circuit families. If the n^{th} circuit in a family takes exponential time to construct, one really has not gained much. Furthermore, the fact that the n^{th} circuit may be independent of the $(n-1)^{th}$ circuit would allow a linear size log-depth family accept a nonrecursive set. It was this non-uniformity that was exploited in [Wi85b] 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 non-uniform ([Sc76],[Pi79]). The approach we use is to make the circuit classes uniform ([Bo77],[Pi79],[Ru81]). The standard method to force this is to require that the transformation $1^n \rightarrow \overline{\alpha_n}$ be easy to compute on a Turing machine, where $\overline{\alpha_n}$ is an encoding over $\{0,1\}$ of the circuit α_n.

Definition 3.1 A circuit family $\{\alpha_n\}$ is $s(n)$- *uniform* if the transformation $1^n \rightarrow \overline{\alpha_n}$ (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 ([Pi79]), 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 ([Co83]) 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 ([Ru81]).

Definition 3.2

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

$$NC = \bigcup_{k \geq 1} NC_k$$

(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. As classes of sets, the following containments are known (see [Co83]).

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

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. The fact mentioned earlier, for example, showed that a proof of $L = NL$ would relativize. We will see here that $L \neq NL$ and several other relationships cannot be exhibited with relativizable proof techniques.

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 non-uniform, since the oracle can be possibly non-recursive. 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 non-uniform measure with respect to the oracle.

In computing the transformation $1^n \rightarrow \overline{\alpha_n}$, 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. Allowing such use we will refer to 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 4.1 A relativized circuit family $\{\alpha_n\}$ is $s(n)$- *uniform*, $U(s(n))$, if the transformation $1^n \rightarrow \overline{\alpha_n}$ 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 $1^n \rightarrow \overline{\alpha_n}$ 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. This assumption of the computability (relative to some oracle) of the transformation would allow some of the oracles constructed below to be recursive. Let us now define a relativized version of NC.

Definition 4.2 Let A be an oracle set.

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

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

$$wNC^A = \bigcup_{k \geq 1} wNC_k^A \qquad NC^A = \bigcup_{k \geq 1} NC_k^A$$

Notice that wNC^A stands for the weakly uniform version of NC^A.

5. Space versus Depth

Now we are able to generalize a result by Borodin [Bo77]. This is a straightforward adaptation of the earlier result.

Lemma 5.1 Let $s(n)$, s: $\mathbf{N}\rightarrow\mathbf{N}$, 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 [Bo77].

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-id following it is deterministic, and thus can be viewed as a $s(n)$-space computable function. Since the $s(n)$-space deterministic functions are already known to be in $U(s(n))\text{-}SIZE\text{-}DEPTH(2^{O(s(n))}, O(s^2(n)))$, 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.

QED

Hence, we have a generalization of a result mentioned in [Co83].

Corollary 5.2 For any oracle A, $NL^A \subseteq NC_2^A$.

With this model, it is not necessarily true that $U(s(n))\text{-}DEPTH^A(s(n))$ is contained in $SPACE^A(s(n))$ as is known to be true in the unrelativized case. This is because a circuit can use the outcomes of up to $2^{s(n)}$ queries to formulate a next query, whereas an $s(n)$-space bounded Turing machine does not have the space to remember any more than $s(n)$ such outcomes. For this model, the best we can do

is bound a circuit family's size by Turing machine time, independent of the family's depth.

Lemma 5.3 Let $t(n)$, $s(n)$, t,s: N→N, be constructible monotonic functions satisfying $t(n) \geq s(n) \geq \log t(n) \geq \log n$. Then for all oracles A,

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

The proof of this is straightforward, 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 6.4 would show, since one query may depend on $t(n)^{O(1)}$ other queries in the circuit. 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 5.4 For all $k \geq 1$ and oracles A, $NC_k^A \subseteq U(\log n)\text{-}SIZE^A(n^k) \subseteq P^A$.

The next step would be to exhibit separation of various classes. For some, this is not always possible as an oracle separating two classes might prove that they are indeed not equal. We can get a similar result for NC_1 and L. Unfortunately, the fact that $NC_1 \subseteq L$ does not relativize, as we shall see in theorem 6.4. But we can show the following.

Theorem 5.5 $NC_1 = L$ 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 a response received. An answer-id α spawns a next query Q. So the language

$$\{ <x,\alpha,i,d> : \text{the } i^{th} \text{ bit of the query caused by } \alpha \text{ on input } x \text{ is } d \}$$

is in L, and, by assumption, in NC_1. So we could hook up $|Q|$ circuits to get an NC_1 circuit β computing the mapping $(x,\alpha) \rightarrow Q$. Remember that $|Q|$ can be at most polynomial in n.

Now consider the $O(n^k)$ answer-id's $\alpha_1,\alpha_2,\cdots,\alpha_{cn^k}$. 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 input α_i to β.

Let us define a non-relativized version of S, $\hat{S} = \{ x\#y \mid M$ accepts x using y as an oracle string - if answer-id α leads to a query Q, then $Q \in A$ is interpreted as the α^{th} 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 $\alpha_1,\cdots,\alpha_{cn^k}$ fixed as input and all having x as an argument. This level has depth $O(\log n)$ as β is an NC_1 circuit.

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 $O(\log n)$.

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 α^{th} bit of y is restricted to be the answer to the α^{th} 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 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 $\alpha_1,\cdots,\alpha_{cn^k}$, and describe some fairly trivial connections. See figure 1 for a diagram of the circuit layout.

QED

As a notational convention, we allow $\subset$ to denote *proper* set containment.

Corollary 5.5.1 $NC_1 \subset L$ if and only if there exists an oracle A such that $L^A - NC_1^A \neq \emptyset$.

Corollary 5.5.2 $NC_1 \subset L$ 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 [BLS84] and [BLS85].

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 5.5 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.

First, we will show a certain structural analogy to the polynomial hierarchy ([St77]). If the polynomial hierarchy is equal at level k, that is, if $\Sigma_k^P = \Pi_k^P$, then the hierarchy collapses to level k. NC has a similar structural property.

Lemma 6.1 For $k \geq 1$, if $NC_k = NC_{k+1}$, then $NC = NC_k$.

Proof

Assume that $NC_k = NC_{k+1}$. It suffices to show that $NC_{k+2} = NC_{k+1}$, for the desired result will then follow by induction. Let $L \in NC_{k+2}$. L^n is accepted by α_n, an NC_{k+2} circuit. Break α_n into $\lceil \log n \rceil$ levels, each of depth $O(\log^{k+1} n)$. We will want each of these levels to be a uniform circuit. To show uniformity, we note that transitive closure is in NC_2. So we could build a uniform circuit which, on input i and α_n, would output a description of an $O(\log^{k+1} n)$ depth circuit for the i^{th} level, which could then be simulated by another circuit. So each level can be viewed as an NC_{k+1} circuit computing a function f_i, $i = 1, \ldots, \lceil \log n \rceil$. The final value is the composition of these functions -- $f_{\lceil \log n \rceil}$ can be viewed as 0-1 valued for set acceptance. So the language $\hat{L} = \{ <d, i, 0^n, y> \mid$ the d^{th} bit of $f_i(y)$, f_i induced by α_n, is 1 $\}$ is in NC_{k+1}. By assumption, then, $\hat{L}$ is in NC_k. So we can uniformly construct an

NC_k circuit for each f_i. By attaching the outputs of the circuit for f_i to the inputs of the circuit for f_{i+1}, clearly a uniform operation, we obtain an NC_{k+1} circuit for L.
QED

It is generally believed that the hierarchy exists with full structure, or at least that $NC_1 \neq NC_2 \neq P$. We can show a relativized collapse of this hierarchy, implying that any proof of separation must be unrelativizable.

Theorem 6.2 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:

$$K(A) = \{ <i,x,0^n> : M_i^A \textit{ accepts } x \textit{ in at most } n \textit{ steps} \}.$$

As in [BGS75], A can be constructed so that

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

For this A, then, $S \in NP^A$ implies that both $S \in L^A$ and $S \in NC_1^A$.
QED

For other classes, showing separation is not quite as onerous. We can exhibit full structure in the NC hierarchy, as has recently been shown for the polynomial hierarchy [Ya85].

Theorem 6.3 There exists an oracle A such that

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

Proof

First we describe a language $L_{k+1}(A)$ which for all A is in NC_{k+1}^A. 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 sequentially, 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 $<k,i>$. At stage $<k,i>$ we will ensure that the i^{th} wNC_k^A circuit family cannot accept $L_{k+1}(A)$. The i^{th} wNC_k^A circuit family will be the one described by the deterministic Turing machine $M^A_{<k,i>}$. Let us consider the circuit α constructed by $M^A_{<k,i>}$ 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_k^A circuit. So, for some d, α can have at most $d \log^{k-1} n$ levels.

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 maximum 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 $2^{d\,l}$ for some l.

Choose an x of length n such that for no z of length n was xz queried in 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 unchanged by any such placement of x.

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

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

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. So 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 level m. By the note, which is an inductive invariant, none of these strings will have been queried 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 steps 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 a string of the form $xzyb_{i_{j-1}} \cdots b_1$ 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 $<k,i>$ we have ensured that the i^{th} wNC_k^A family will not accept $L_{k+1}(A) \in NC_{k+1}^A$. Hence, the diagonalization is complete.

QED

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 6.3 to separate NC^A_{k+1} from non-uniform NC^A_k. Notice that the uniformity of the lower level was hardly used. The problem is one of countability - there is an uncountable number of non-uniform 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 6.3.1 Let A be the oracle constructed in the proof of theorem 6.3. 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, $L_{k+1}(A) \notin wNC^A_k$. Each $L_k(A)$ was seen to be in NC^A_k, so there exists a series of circuit families $\{\alpha^k_n\}$, the k^{th} one being an NC^A_k family accepting $L_k(A)$. Let us define the language

$$S = \{ <k,x> : \alpha^k_{|x|} \text{ accepts } x \text{ relative to } A \}.$$

S is easily seen to be in P: there is an algorithm for it given in the proof of theorem 6.3. By our initial supposition, $S \in wNC^A$, so there exists a k such that $S \in wNC^A_k$. But by using a wNC^A_k circuit family for S, one can easily construct a wNC^A_k circuit family for $L_{k+1}(A)$, which we know to be impossible. Hence, S cannot be in wNC^A_k for any k.

QED

Corollary 6.3.2 There is an oracle A such that

$$NC^A_1 \subset NC^A_2 \subset \cdots \subset NC^A \subset P^A.$$

This follows from theorem 6.3 and lemma 6.3.1 taken with corollary 5.4.

Corollary 6.3.3 There is an oracle A such that

$$wNC^A_1 \subset wNC^A_2 \subset \cdots \subset wNC^A.$$

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

Theorem 6.4 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 describe 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.

See figure 2 for an outline of a circuit accepting $S_k(A)$.

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 in stages. Let $\{M_i\}$ be an enumeration of the nondeterministic Turing machines which operate under the RST restriction. Now at stage $<i,c,k>$ of the construction, we ensure that M_i^A does not accept $S_k(A)$ within cn^k space. Let us assume that $M_{<i,c,k>}$ is the machine M_i restricted to cn^k space. In fact, we could choose that as the definition of the encoding $<i,c,k>$.

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 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

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

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

Construction of A

Stage 0: $A \leftarrow \emptyset$

Stage $<i,c,k>$: Let d be the constant such that on an input of length n there are at most 2^{dn^k} possible queries by $M_{<i,c,k>}$. 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 y^{th} 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.

QED

This implies that our model is not quite as natural as we had hoped. Below we propose a natural relativized model for which $NC_1 \subseteq L$ holds. Theorem 6.4 together with corollary 5.2 does give us the following.

Corollary 6.4.1 There is an oracle A such that $NL^A \subset NC_2^A$.

7. Oracle Stacks

Theorem 6.4 shows that the containments $NC_1 \subseteq L$ and, in general, $U(s(n)) - DEPTH(s(n)) \subseteq SPACE(s^k(n))$ do not relativize. The fact that $NC_1 \subseteq L$ did not relativize was due to the fact that an NC_1 circuit has a kind of built in memory of the outcomes of all the queries: the output lines from each of the query nodes provide this. So in such a circuit, one query can depend on the outcomes of $\omega(\log n)$ previous queries. No log-space machine has the memory to accommodate this information.

The remedy, then, is to allow the space bounded machines to have more than one query tape, on which partially constructed queries can be stored. We would have to settle on a structure for the collection of tapes and a method, if desired, to measure the space usage. A model along these lines is found in [Or84], where random access is allowed to the tapes through an index tape. Only the index tape was included in the space bound. This approach is adopted in [Bu86] when discussing a variant (called the m variant) of relativized space, with the modification that the space will be the sum of the logarithms of the lengths of the queries. This is essentially the measure we adopt, but the query structure will be different. Here, we will put the partially constructed queries on a stack, as suggested by Cook [Co84] and introduced in [Wi85a]. This structure takes advantage of the essentially tree-like nature of the circuits.

Definition 7.1 An *oracle stack* is a push down stack replacing the query tape, each entry can be thought of as an independent query tape. The machine is allowed to write onto the top entry, possibly adding to what is already there, but is not allowed to read it. When the machine is finished writing, it can either **push**, at which point the tape is pushed down and the top tape becomes empty, or it can **query**, which causes the contents of the top tape to be queried and erased, and the stack to be popped. If the contents of the stack are the partial queries $q_1, q_2, \cdots, q_k$, the *space used* by the stack is

$$\sum_{i=1}^{k} max(\log|q_i|,1).$$

Notice that a machine whose stack space is bounded by $s(n)$ can query strings of length at most $2^{s(n)}$.

Defining the use of the stack is not a problem for deterministic machines, but for nondeterministic ones there may be several possibilities, especially for the restricted versions. The unrestricted version is straightforward: where $s(n)$ is the space bound, for the nondeterministic Turing machine to accept an input of length n, there must be an accepting computation which uses at most $s(n)$ work space and stack space. Under the RST restriction, we must decide at which points the machine must act deterministically. If it is to be between starting to write on the stack until a push or a query, then this turns out to be no restriction at all. The machine could write $s(n)$ nondeterministically chosen bits onto a work tape, copy them onto the stack, push, query a dummy string (causing a pop), copy more nondeterministically chosen bits onto the stack, and so on. So we will force the machine to act deterministically from the point at which it writes onto an empty stack (entering a beg-write-id) to a point at which the stack is again empty.

Definition 7.2 L is in $SPACE_s^A(s(n))$ if and only if it is accepted by a Turing machine relative to A with an oracle stack which uses space at most $O(s(n))$. L is in $NSPACE_s^A(s(n))$ if and only if it is accepted by a nondeterministic Turing machine using an oracle stack whose space usage is at most $O(s(n))$ under the restriction that the machine may make nondeterministic moves only when the stack is empty.

Definition 7.3

$$sL^A = \bigcup_{c\geq 1} SPACE_s^A(c\ \log n) \qquad sNL^A = \bigcup_{c\geq 1} NSPACE_s^A(c\ \log n)$$

8. Faithful Containments

The major appealing feature of this model is that when simulating a circuit family by a Turing machine, the latter's space does not blow up relative to the former's depth.

Theorem 8.1 For any oracle A and space bound $s(n)$,

$$U(s(n))\text{-}DEPTH^A(O(s(n))) \subseteq SPACE_s^A(O(s(n))).$$

Proof

The proof of this is fairly simple. The Turing machine can derive the relevant parts of the circuit in $O(s(n))$ space due to the uniformity condition. And as in the unrelativized case in [Bo77], it will perform a depth first search of the circuit starting at the circuit's output edge. As bits of a query are computed, they are written on a query tape. If determining further bits of the query involves evaluating a subsection of the circuit, then the partial query is pushed and stored while the evaluation continues. Since the sum of the logarithms of the size of the queries on any directed path from an input edge to the output edge is bounded by $O(s(n))$, so bounded will the space used by the stack be.

QED

Corollary 8.2 For any oracle A, $NC_1^A \subseteq sL^A$.

So by strengthening the space bounded Turing machines, we have ensured that $NC_1^A \subseteq sL^A$. What is not clear is whether sNL^A will be contained in NC_2^A, or even NC^A. In the proof of theorem 6.3 there were constructed some sets $L_k(A)$ which, for some A, are not in NC_{k-1}^A (but are always in NC_k^A). These $L_k(A)$ fortunately do not seem to be in sL^A or sNL^A, but they would be if the machines could make multiple copies in the stack. This they could do if random access were allowed, if the query tape were not erased, or if the top query tape was copied into the stack rather than pushed. The similar notion of relativized space used in [Or84] would allow that any $L_k(A)$ be computed in log-space relative to A.

Under certain circumstances, it is easy to see that $sNL^A \subseteq NC_2^A$. Suppose that all queries were of length at most $O(\log n)$. Then there would be at most polynomially many of them, and they could all be queried simultaneously at the top level of the circuit. An NC_2 circuit below this could complete the computation. On the other hand, suppose that the length of the smallest query on the stack is of length n. Since the space used is $O(\log n)$, there will be at most a constant number of strings

on the stack. So there would only need to be a constant number of query levels, with an NC_2 circuit between each level. A problem, however, arises when the items on the stack are of intermediate length. In that case, we can at least show that sNL will be in NC, and in particular it will be contained in NC_3.

Theorem 8.3 For any oracle A, $sNL^A \subseteq NC_3^A$.

Proof

For an arbitrary A, let $S \in sNL^A$. Then S is accepted by a nondeterministic Turing machine M with an oracle stack which uses work space and stack space bounded by $O(\log n)$. We will show that there is an NC_3^A circuit which, on an arbitrary input x of length n, simulates M on x. This circuit, incidentally, is independent of A. By the definition of sNL, the machine M will act deterministically when there are items on the stack. Furthermore, the stack height is at most $c\log n$ for some constant c and the queries are at most polynomial in length. (In fact, we could restrict things a bit more - certainly not all $c\log n$ queries are of polynomial length. Taking advantage of this fact should allow us to show that the language is in NC_2.)

The main idea of the proof is to compute the reachability graph on the id's first for stack height 0 (trivial), then for stack height 1, then stack height 2, and so on. Computing this graph for stack height $k+1$ from the one for stack height k will be seen to be an NC_2 circuit. Thus, the total depth will be $c\log n\ O(\log^2 n) = O(\log^3 n)$ to compute the connectivity of the id's when something is on the stack. The nondeterministic closure can then be computed by an NC_2 circuit.

We define the following function:
on inputs α, β, and k, where α and β are id's, $|\alpha|, |\beta| \leq O(\log n)$, and k is an integer $0 \leq k \leq c\log n$,

$u(\alpha,\beta,k)=1$ iff β is reachable from α in such a way that
the stack is empty at id α and β and
for all intermediate id's, the stack height is at least one but never more than k.

That is, α begins a write on a tape and β is the direct result of the first pop (query) which empties the stack, which in between never holds more than k partially constructed queries at a time. Also note that $u(\alpha,\beta,0)=1$ if and only if α yields β in one step, without recourse to the oracle.

Define U_k to be a bit string whose $<\alpha,\beta>^{th}$ bit is $u(\alpha,\beta,k)$, where $<.,.>$ is a suitable bijection of $\mathbf{N}^2$ to $\mathbf{N}$. Since the id's have length bounded by $c\log n$, there are polynomially many of them, and so U_k also has polynomial length. It is easy to see that U_0 can be computed by an NC_1 circuit.

Claim There is an NC_2^A circuit computing U_{k+1} from U_k.

Since the length of each U_k is at most polynomial, the claim will follow from the following subclaim.

Subclaim For each id pair α, β, there is an NC_2^A circuit computing the $<\alpha,\beta>^{th}$ bit of U_{k+1} from U_k.

This we show by

(i) Determining $u(\alpha,\beta,k+1)$ from U_k assuming the answer to the query yielding β to be both yes and no: that is, compute both $u(\alpha,\beta,k+1)$/yes and $u(\alpha,\beta,k+1)$/no
(ii) calculate the last query made
(iii) make that query, returning $ans \in$ {yes,no}
(iv) $u(\alpha,\beta,k+1)$ will be $[u(\alpha,\beta,k+1)$/yes & ans =yes] or $[u(\alpha,\beta,k+1)$/no & ans =no]

We will see that both steps (i) and (ii) can be computed in logarithmic space, hence both can be calculated by an NC_2 circuit. Step (iii) is clearly in NC_1 - a single polynomial size, and hence log depth, query will suffice. Also, step (iv) is constant depth.

We now show that given α, β, U_k, and the original input x, in log-space we can compute $u(\alpha,\beta,k+1)$ with the assumption *assmp* as the response to the last query.

Algorithm

0. If $u(\alpha,\beta,k)=1$, then output 1 and halt.
1. Ensure that α begins to write to the query tape. If not, then output 0 and halt.
2. Simulate M starting from id α until an id α' is reached which causes either a query (pop) or a push (new begin-write id). During this simulation, ignore any writing to the query tape.
3. If id α' causes a push, then search for a β' such that bit $<\alpha',\beta'>$ of U_k is 1 (that is, $u(\alpha',\beta',k)=1$). Let α be this β' and return to step 2.
4. [Here, α' causes a query.]
   Test if α' yields β with *assmp* as the oracle response.
5. If so, then output 1. Otherwise, output 0.

The algorithm is $O(\log n)$ space, so can be performed by an NC_2 circuit. By a similar argument, we can see that step (ii) of our agenda can also be done in log-space. In the algorithm above, in step 2 of the simulation, instead of ignoring any writing to the query tape, direct them to an output tape. Note that for both steps (i) and (ii), step 2 of the algorithm will be deterministic. This behavior of M is guaranteed by the definition of our oracle stack model.

Thus, the subclaim and therefore the claim hold. So given the input x, we can compute U_K, where $K = c\log|x|$ is the maximum stack height, on an NC_3^A circuit. To complete the proof, we show that given x and U_K, the answer to $x \in S$ can now be computed by a NC_2 circuit. To show this, we illustrate how to do so in nondeterministic log-space, since it is known that $NL \subseteq NC_2$.

> Starting from the initial id, until a final id is encountered, simulate M on x. When a beg-write id α is encountered, nondeterministically guess a β of length $O(\log n)$. If the $<\alpha,\beta>^{th}$ bit of U_K is 1, then let α be β and continue the simulation.

All id's α and β are at most $O(\log n)$ in length, so this is an NL operation. To recap, the string U_K can be computed from x by a circuit of depth $O(\log^3 n)$. From U_K and x, a circuit of depth $O(\log^2 n)$ can determine if $x \in S$. Therefore, $S \in NC_3^A$.
QED

The proof has actually shown something slightly stronger. If the *sNSPACE* machine being simulated has stack height bounded by $O(\log n)$ and questions of at most polynomial length - thus using $O(\log^2 n)$ space - then the language it accepts is in NC_3^A. As mentioned earlier, if the *sNL* machine has constant stack height and asks polynomial length questions or has $O(\log n)$ stack height and asks constant length questions, then the language will be in NC_2^A. However, the circuit must be set up to possibly accommodate many short questions on one input and few long questions on another.

9. Related Results

As further positive evidence of the applicability of this model, we see that Savitch's theorem relativizes.

Theorem 9.1 Let $f(n)$ be a space bound. Then for all oracles A,

$$NSPACE_s^A(f(n)) \subseteq SPACE_s^A(f^2(n)).$$

Proof

For an arbitrary $L \in NSPACE_s^A(f(n))$, let M be the nondeterministic Turing machine with an oracle stack operating under the RST restriction accepting L. And let x, $|x| = n$, be some input. Consider all id's of M on x for which the stack is empty. As in the unrelativized case, there are only $2^{O(f(n))}$ such id's. Suppose I_1

and I_2 are two such id's. If I_1 is a beg-write-id which deterministically leads to I_2, the next id where the stack is empty, we will view this as a *single* nondeterministic step. Note that it is only in such segments of the computation that the use of the oracle stack takes place.

So for two id's I_1 and I_2, I_1 can lead to I_2 in one nondeterministic step in two ways: either by a deterministic oracle query sequence as under the RST restriction or by I_1 causing a nondeterministic choice, one choice of which leads to I_2 in a single move (or, thirdly, I_1 could equal I_2). In any case, the claim that I_1 leads to I_2 in one nondeterministic step could be verified in $f(n)$ space deterministically, possibly using the oracle stack.

To check if I_1 leads to I_2 in at most 2^i nondeterministic steps, we would test, for all empty stack id's I_{int}, whether I_1 leads to I_{int} and I_{int} leads to I_2, each within 2^{i-1} nondeterministic steps. This suggests the standard recursive algorithm ([Sa70], see also [HU79]), the only modification being to the testing done at the base case (i=0), where a slightly more complicated test may require some oracle calls, though still this can be done in $f(n)$ space. The depth of the recursion would be $O(f(n))$, and the number of bits needed to be saved at each level would be $f(n)$, the length of the description of an id. Hence, the deterministic space requirement would be $O(\ f^2(n)\ )$.

QED

The oracle stack still does not allow one to easily separate sL from sNL relative to some oracle. In this sense, they behave much like relativized L and NL. This extends a result from [Si77].

Theorem 9.2 $L = NL$ if and only if, for all oracles A, $sL^A = sNL^A$.

Proof

We can adapt the proof of fact 2.1 here. It suffices to prove that if $L = NL$, then for arbitrary A, $sNL^A \subseteq sL^A$. Noticing that any machine M accepting a set $S \in sNL^A$ has at most cn^k, for some c and k, empty stack id's on inputs of length n, we define $\hat{M}$ as follows. $\hat{M}$ takes as input $x\#y_1\#y_2\# \cdots \#y_{cn^k}$, $|y_i| = O(\log n)$. $\hat{M}$ simulates M on input x, but when M, in id α, starts to write to the oracle stack, the computation resumes from id y_α. The set $\hat{S}$ accepted by $\hat{M}$ is in $NL = L$. So there is a deterministic Turing machine M_0 accepting this set. To accept the same set of strings as M, simulate M_0. When it needs a y_α, simulate the deterministic portion of M starting from id α. This will use $O(\log n)$ space on the oracle stack. When the stack is empty, pass the resultant id back to M_0.

QED

10. 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 those of the classic complexity classes are difficult: both will 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 [Pa77], [Bl82], and [PPST83].

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 to depth. In particular, the depth restricted devices were noticeably more powerful than their desired corresponding space restricted devices. The oracle stack was introduced, and we were able to show that this model is faithful to $NC_1 \subseteq L$ and $NL \subseteq NC$. The problem of whether $NL \subseteq NC_2$ can relativize remains open.

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 $NC_1^A \neq NC_2^A = NC^A$? By lemma 6, it would suffice to show $NC_1^A \subset NC_2^A$ with $NC_2^A = NC_3^A$. Other oracles we would like to see are ones witnessing any of the following:

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

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

$NC = P \neq NP$;

$NC \neq P = NP$.

References

[An80] D. Angluin, "On Relativizing Auxiliary Pushdown Machines," *Math. Systems Theory* 13(1980), pp 283-299

[BGS75] T. Baker, J. Gill, and R. Solovay, "Relativizations of the P=?NP Question," *SIAM Journal on Computing, vol. 4, no. 4,* Dec. 1975, pp 431-452

[BS76] T. Baker and A. Selman, "A Second Step toward the Polynomial Hierarchy," *Proc. 17th FOCS* (1976), pp 71-75

[Bl82] N. Blum, "A Boolean Function Requiring 3n Network Size," *Tech. Report A82/13* (June 1982), Universität des Saarlandes

[BLS84] R. V. Book, T. J. Long, and A. Selman, "Quantitative Relativizations of Complexity Classes," *SIAM Journal on Computing* 13(1984), pp 461-486

[BLS85] R. V. Book, T. J. Long, and A. Selman, "Qualitative Relativizations of Complexity Classes," to appear in the *Journal of Computer and System Sciences* (1985)

[Bo77] A. Borodin, "On Relating Time and Space to Size and Depth," *SIAM Journal on Computing, vol. 6, no. 4,* Dec. 1977, pp 733-744

[Bu86] J. Buss, "Relativized Alternation", to appear in the *Proceedings of the Structure in Complexity Theory Conference,* June, 1986

[Co83] S. Cook, "The Classification of Problems which Have Fast Parallel Algorithms," *Technical Report #164/83* University of Toronto, Department of Computer Science (1983)

[Co84] S. Cook, *private communication*

[LL76] R. Ladner and N. Lynch, "Relativizations of Questions about Log-Space Reducibility," *Math. Systems Theory* 10(1976), pp 19-32

[Or84] P. Orponen, "General Nonrelativizability Results for Parallel Models of Computation," *Proceedings of the Winter School on Theoretical Computer Science,* 1984, pp 194-205

[Pa77] W. Paul, "A 2.5n Lower Bound on the Combinatorial Complexity of Boolean Functions," *SIAM Journal on Computing, vol. 6, no. 3,* Sept. 1977, pp 427-443

[PPST83] W.Paul, N. Pippenger, E. Szemeredi, and W. Trotter, "On Nondeterminism versus Determinism and Related Problems," *Proceedings of the 24th FOCS,* 1983, pp 429-438

[Pi79] N. Pippenger, "On Simultaneous Resource Bounds (Preliminary Version)," *Proceedings of the 20th FOCS* (1979), pp 307-311

[RS81] C. W. Rackoff and J. I. Seiferas, "Limitations on Separating Nondeterministic Complexity Classes," *SIAM Journal on Computing, vol. 10, no. 4*, Nov. 1981, pp 742-745

[RST84] W. L. Ruzzo, J. Simon, and M. Tompa, "Space- Bounded Hierarchies and Probabilistic Computations," *Journal of Computer and System Sciences* 28, 2(April 1984), pp 216-230

[Ru81] W. L. Ruzzo, "On Uniform Circuit Complexity," *Journal of Computer and System Sciences* 22, 3(1981), pp 365-383

[Sa70] W. Savitch, "Relationships between Nondeterministic and Deterministic Tape Complexities," *Journal of Computer and System Sciences* 4(1970), pp 177-192

[Sc76] C. P. Schnorr, "The Network Complexity and the Turing Machine Complexity of Finite Functions," *Acta Informatica* 7(1976), pp 95-107

[Si77] I. Simon, "On Some Subrecursive Reducibilities," *Ph.D. Dissertation, Stanford Univ.*, March 1977

[St77] L. Stockmeyer, "The Polynomial Time Hierarchy," *Theoretical Computer Science* 3(1977), pp 1-22

[Wi85a] C. Wilson, "Relativized Circuit Size and Depth," *Technical Report #179/85* University of Toronto, Department of Computer Science (1985)

[Wi85b] C. Wilson, "Relativized Circuit Complexity," *Journal of Computer and System Sciences* 31, 2(October 1985), pp 169-181

[Ya85] A. Yao, "Separating the Polynomial-Time Hierarchy by Oracles," *Proc. 26th FOCS,* (1985), pp 1-10

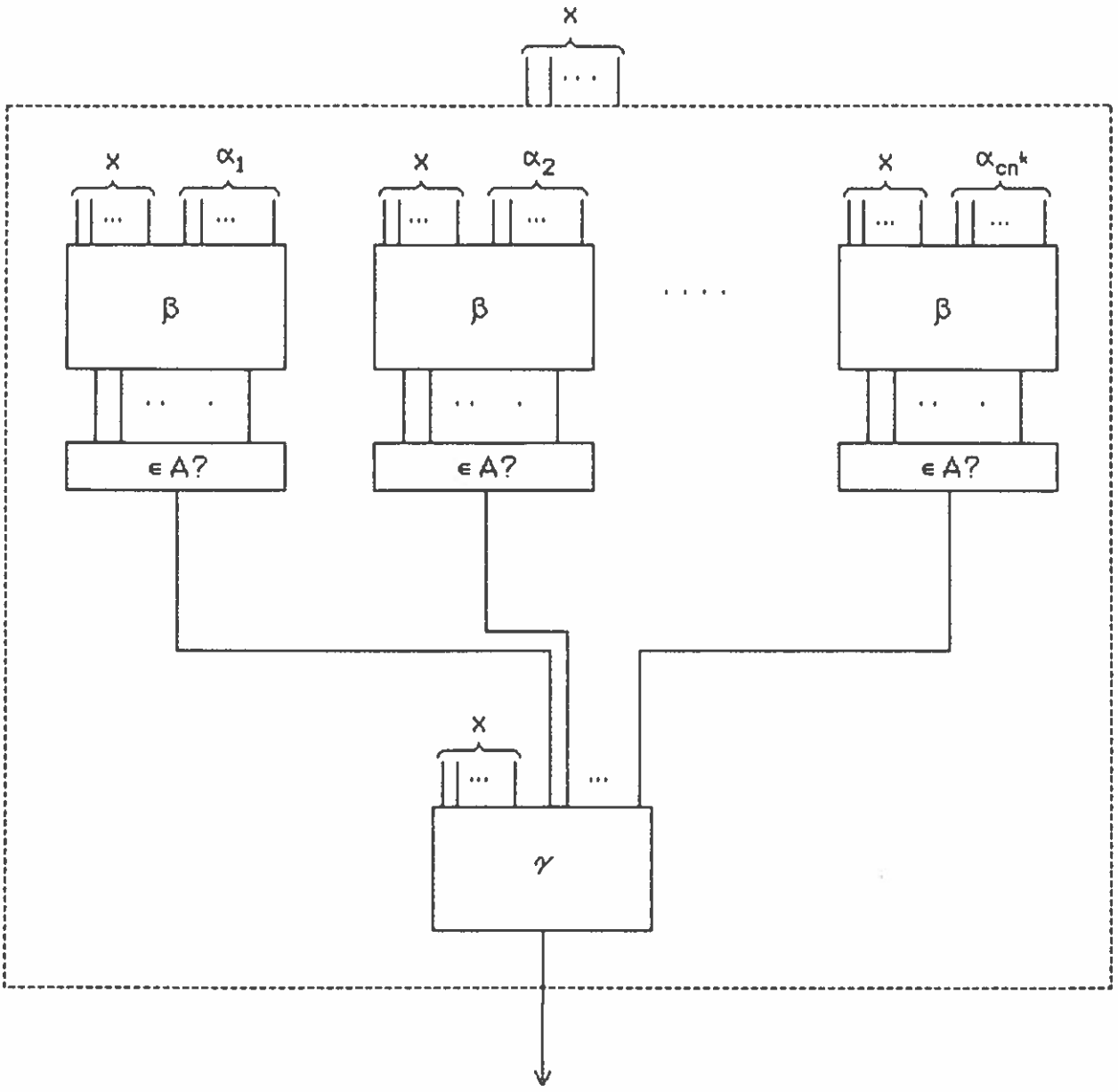

Figure 1
Theorem 5.5

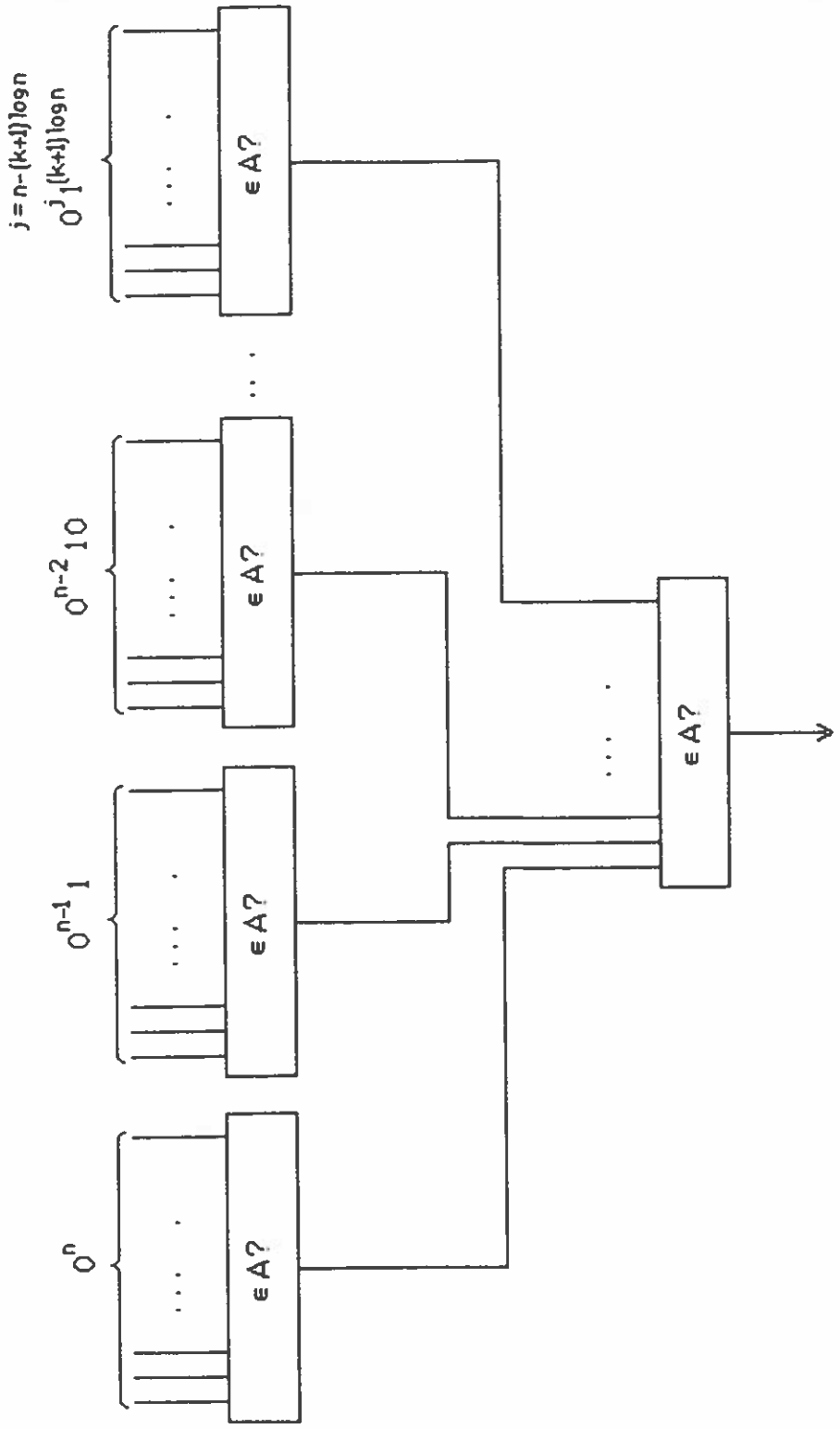

Figure 2
Theorem 6.4