INFORMATION AND COMPUTATION 96, 65–76 (1992)

# Generic Oracles, Uniform Machines, and Codes

MARTIN DOWD

*2532 Orange Avenue, Costa Mesa, California 92627*

The basic properties of generic oracles are reviewed, and proofs given that they separate $\mathscr{P}$ and $\mathscr{NP}$ and are weakly incompressible. A new notion of generic oracle, called t-generic, is defined. It is shown that t-generic oracles do not exist, and consequently a nondeterministic oracle machine which for any oracle $X$ accepts the tautologies relativized to $X$ when running with oracle $X$ does not run in polynomial time at any oracle. A weak form of t-generic oracle, called $r$-generic, is shown to exist, and it is shown that if there exists an $r$-generic oracle $X$ at which the $r$-query relativized tautologies are not in co $\mathscr{NP}^X$ then $\mathscr{NP} \neq$ co $\mathscr{NP}$. The notion of a code for the Boolean functions is defined, and it is shown that generic oracles do not have short codes in any code. Universal circuits of size $O(n \log^4 n)$ are shown to exist, and it is shown that increasing the number of $\wedge$, $\vee$ gates from $g$ to $2g + 1$ allows the computation of new Boolean functions. © 1992 Academic Press, Inc.

## 1. INTRODUCTION

A great deal of attention has been focused on the behavior of oracle machines which run in some complexity bound. The results obtained comporise a branch of complexity theory in their own right, and contribute to an understanding of fundamental open questions of complexity theory. For example the set of oracles separating $\mathscr{P}$ and $\mathscr{NP}$ has measure 1 (Bennett and Gill, 1981), is co-meager (Mehlhorn, 1973), and contains all generic oracles (Dowd, 1982; Regan, 1984; Poizat, 1986; Blum and Impagliazzo, 1987). This suggests that $\mathscr{P}$ and $\mathscr{NP}$ are unequal; but since there are also oracles for which they are equal, the unrelativized case must be settled by new methods.

This paper presents primarily the theorem that a nondeterministic oracle machine which uniformly accepts the relativized tautologies, that is, which regardless of $X$ accepts the tautologies relativized to $X$ when running with oracle $X$, never accepts in polynomial time, that is, does not accept in polynomial time at any oracle $X$. That it does not accept in polynomial time at any oracle $X$ where $\mathscr{NP}^X \neq$ co $\mathscr{NP}^X$ is a triviality. Letting $\mathscr{P}^X$ etc. denote the class of functionals, certainly co $\mathscr{NP}^X \neq \mathscr{NP}^X$; by Lemma 2.2 of (Blum and Impagliazzo, 1987), if $\mathscr{NP}^X \cap$ co $\mathscr{NP}^X \neq \mathscr{P}^X$ then $\mathscr{P} \neq \mathscr{NP}$.

The proof of the theorem involves a notion related to forcing. A notion



0890-5401/92 $3.00
Copyright © 1992 by Academic Press, Inc.
All rights of reproduction in any form reserved.

of generic oracle, called t-generic, is defined; this notion is so strong that t-generic oracles can be shown not to exist, and the theorem follows. If the definition of t-generic oracle is sufficiently restricted, to what we call $r$-generic for an integer $r$, then such oracles do exist. These have the "genericity" property that if $X$ is $r$-generic, and no nondeterministic machine with oracle $X$ accepts the $r$-query tautologies relativized to $X$ in polynomial time, then $\mathcal{NP} \neq \text{co}\,\mathcal{NP}$.

We review the standard notion of generic oracle also, and consider the complexity of these and $r$-generic oracles as families of Boolean functions and as strings, i.e., their Kolmogorov complexity. Also, some properties of circuits as a model of computation of the Boolean functions are given.

## 2. Generic Oracles

If $X$ and $Y$ are partial functions from the natural numbers $\mathcal{N}$ to $\{0, 1\}$, $Y$ is said to be an approximation to $X$, written $Y \sqsubseteq X$, if the domain of $Y$ is a subset of the domain of $X$, and $Y(n) = X(n)$ for all $n$ in the domain of $Y$. An oracle may be viewed as a total function from $\mathcal{N}$ to $\{0, 1\}$, or "bit string." A partial function is called finite if its domain is. Often the domain is $\{i: 0 \leqslant i < n\}$, resulting in a finite bit string. If $X$ is a bit string we let $X_n$ denote the finite bit string consisting of the first $n$ bits of $X$.

We use $M^X$ to denote either an accepting–rejecting oracle machine, the recursive functional it computes, or the predicate at some particular oracle $X$; the context makes clear which is intended. For $Y$ a partial function, $X$ an oracle, and $M$ an accepting–rejecting total deterministic oracle machine, say that $Y$ forces $\forall x M^X(x)$ if $\forall x M^X(x)$ is true whenever $Y \sqsubseteq X$. Define an oracle $G$ to be m-generic if for any total deterministic oracle machine $M$, if $\forall x M^G(x)$ is true, then it is forced by some finite approximation to $G$.

Oracles generic for arithmetic or fragments thereof are of importance in recursive function theory (Feferman, 1965; Hinman, 1969), and have more recently been considered in complexity theory. The definition using machines defines a slightly smaller class than the oracles generic for the $\Pi_1$ fragment of arithmetic with an extra unary predicate; this is so because bounded quantifiers are no longer redundant. The DMPR theorem (see Davis, 1973) implies that the $\Sigma_1$ predicates are the same whether or not bounded quantifiers are present, when the language is that of arithmetic. Weiss (1980) has shown that this need not hold with an oracle. Certainly m-generic oracles are generic for the $\Pi_1$ formulas of relativised arithmetic, but the converse does not follow, and indeed is false.

Theorem 1. *There is an oracle $X$ which is $\Pi_1$-generic but not m-generic.*

*Proof.* Let $E_n$, $n \geqslant 1$, be an enumeration of the open formulas in the language of arithmetic with an oracle, such that $E_n$ makes fewer than $n$ queries. The usual nonconstructive procedure can be used to construct a $\Pi_1$-generic $X$, namely, at stage $n$, do nothing unless for some extension of $X$ and some $\mathbf{y}$ $\neg E_n^X(\mathbf{y})$; in this case extend $X$ to guarantee that this holds. To ensure that $X$ is not m-generic, simply ensure at stage $n$ that there is an $m \leqslant n$ such that $\langle n, m \rangle \in X$, where $\langle n, m \rangle$ is the Gödel pairing function.

The following theorem states the usual properties of $m$-generic oracles; a proof may be found in (Dowd, 1982). By Cantor space is meant the space of oracles, equipped with the usual topology and measure (see Rogers, 1967).

THEOREM 2. *The m-generic oracles form a co-meager, measure* 0 *subset of Cantor space, closed under complementation and finite changes. They do not include r.e. or co-r.e. sets, but do include* $\Delta_1$*-sets.*

Say that a subset $S$ of Cantor space is $\Pi_1(X)$ if there is a recursive predicate $M^X(x)$ such that $S(X) \Leftrightarrow \forall x M^X(x)$; similarly $S$ is $\Sigma_1(X)$ if there is a recursive predicate $M^X(x)$ such that $S(X) \Leftrightarrow \exists x M^X(x)$.

THEOREM 3. *If $S$ is a $\Sigma_1(X)$ predicate then $S$ is true of every m-generic oracle iff it is true in every open interval, that is, is true at some $X$ in the interval.*

*Proof.* One direction follows since the m-generic oracles are a dense subset of Cantor space. Conversely if $\exists x M^G(x)$ is false for $G$ m-generic, then $\forall x \neg M^G(x)$ is true and hence forced by some finite $Y \sqsubseteq G$. $\exists x M^G(x)$ is thus false for every $X$ in the interval determined by $Y$.

COROLLARY 4. *For any m-generic $G$, $\mathscr{P}^G \neq \mathscr{NP}^G$.*

*Proof.* The statement that a particular polynomial time deterministic oracle machine fails to accept the satisfiable formulas at $X$ is $\Sigma_1(X)$. Further it is true in any interval.

This corollary is well known (for various notions of generic oracles); the proof here is short and useful for other questions. For example, consider the question of whether m-generic sets are Kolmogorov incompressible. To review the basic definitions, a name for an integer is a Turing machine with no input which outputs the integer. The Kolmogorov complexity $K(x)$ of the integer $x$ is the length of the shortest name. The system of names may be required to be prefix-free or "instantaneous"; we denote this complexity by $K_I(x)$.

We will say that a bit string $S$ is incompressible if there are $m$, $c$ such

that $K(S_n) \geqslant n - c$ for $n \geqslant m$. Say that $f$: $\mathcal{N} \mapsto \mathcal{N}$ is restricting if for any $c$ there is an $n > c$ such that $f(n) < n - c$; $S$ is incompressible iff there is no restricting function bounding $K(S_n)$. Say that $S$ is weakly incompressible if there is no recursive restricting $f$ bounding $K(S_n)$.

There are nonrecursive restricting $g$ which are not bounded by any recursive restricting $f$, so it is not obvious that weak incompressibility implies compressibility. To see this, let $\{f_1, \ldots\}$ be a list of recursive restricting functions such that $f_i(x) \leqslant f_{i+1}(x)$ for all $x$, and every recursive restricting $f$ is dominated by some $f_i$ (i.e., $f(x) < f_i(x)$ almost everywhere). Then $g$ may be defined by considering those values where $f_i$ crosses $n - i$ for the last time; details are left to the reader.

If a system of names is restricted to be prefix-free, then the incompressible $S$ form a subset of Cantor space of measure 1 (Chaitin, 1975). This is not so if an arbitrary system of names is allowed (Chaitin, 1969). To the author's knowledge it is open whether for an arbitrary system of names there is an incompressible bit string. As we will show in the next corollary, m-generic bit strings are weakly incompressible. Another obvious question, which we leave open, is whether for a prefix-free system of names there is an incompressible m-generic bit string.

COROLLARY 5. *An m-generic $G$ is weakly incompressible.*

*Proof.* The statement that a particular recursive restricting $f$ is a bound on the names is $\Pi_1(X)$. Further it is false in any interval.

## 3. UNIFORM MACHINES

The relativised propositional calculus adds to the language of the propositional calculus an $n$-ary connective for each $n$; we will use $X^n$ to denote this. Given a set $X$, the bit string $X_{2^n}$ determines an $n$-ary Boolean function, by numbering the bit positions, from 0 on the left, and considering the bit number as an $n$-tuple of bits. We let $X^n$ denote this Boolean function also; whether the Boolean function or the connective is intended will be clear from context.

Let $\mathrm{TAUT}^X(x)$ be the functional which is true if $x$ is the Gödel number of a relativized formula which is a tautology when the $X^n$ are interpreted according to $X$ (i.e., when the interpretation of the connective $X^n$ is the Boolean function $X^n$). $\mathrm{TAUT}^X$ is "uniformly" complete for co $\mathcal{NP}^X$, meaning that given any co $\mathcal{NP}$ functional $M^X$, there is a polynomial time computable function $f$ such that for any oracle $X$, $M^X(x)$ iff $\mathrm{TAUT}^X(f(x))$. This is well known, and follows by easy modifications to standard arguments. We say that an oracle machine accepts $\mathrm{TAUT}^X$ uniformly if for each $X$, operating with respect to $X$ it accepts exactly $\mathrm{TAUT}^X$.

Define $G$ to be t-generic if there is a polynomial $p$ such that for any (Gödel number of a) relativized formula $x$, if $\mathrm{TAUT}^G(x)$ then $\mathrm{TAUT}^G(x)$ is forced by at most $p(|x|)$ queries to $G$, where $|x|$ denotes the length of the binary notation for $x$. By this we mean that there is a finite partial function $F$ whose domain has size at most $p(|x|)$, where $F \sqsubseteq G$ and $\mathrm{TAUT}^X(x)$ holds whenever $F \sqsubseteq X$.

LEMMA 6. *If a deterministic polynomial time oracle machine $M^X$ accepts all its inputs with respect to a t-generic oracle $G$, then it is forced to do so by a sparse set of queries. That is, there is a partial function $Y$ from $\mathcal{N}$ to $\{0, 1\}$ satisfying $Y \subseteq G$ whose domain is sparse, which forces $\forall x M^X(x)$.*

*Proof.* The relativized formula asserting that "on all inputs of length $\leqslant n$ the machine $M$ accepts" is a tautology with respect to the oracle $G$ for every $n$, and its length is bounded by a polynomial in $n$. Therefore the $n$th is forced by a set $W_n$ of queries to $G$ of size polynomial in $n$. Let $W = \bigcup \{W_n : n \text{ is a power of } 2\}$. Then $W$ is sparse, and forces the statement.

The polynomial time restriction may be removed; for any total deterministic oracle machine $T$ there is a deterministic polynomial time oracle machine $U$ such that for each $X$, $\forall x T^X(x)$ iff $\forall x U^X(x)$.

LEMMA 7. *There does not exist a t-generic oracle.*

*Proof.* The statement that $X$ is t-generic with a given polynomial $p$ is $\Pi_1(X)$, and if true of $X$ is forced by a sparse set of queries. Thus if there is a t-generic set there is a sparse t-generic set. However, the statement that $X$ is sparse, with a particular polynomial $p$, is $\Pi_1(X)$, so if $X$ were sparse and t-generic it would be forced to be sparse by a sparse set of queries, which is clearly absurd.

THEOREM 8. *If $M$ is a nondeterministic oracle machine uniformly accepting $\mathrm{TAUT}^X$, then for no $X$ are the members of $\mathrm{TAUT}^X$ accepted in polynomial time.*

*Proof.* Suppose the members of $\mathrm{TAUT}^X$ are accepted in polynomial time for some $X$; then $X$ is t-generic.

The argument is easily modified to show that the members of $\mathrm{TAUT}^X$ are not accepted in time $t(n)$ for any recursive subexponential function $t$. The analogous result for $\mathrm{SAT}^X$ and deterministic oracle machines is false. Consider the deterministic oracle machine which assumes $X$ is QVAL, the quantified validities, and on input $F$, replaces queries by quantified

formulas and using self-reducibility computes a satisfying truth assignment. If this truth assignment satisfies $F$ accept, otherwise run an arbitrary deterministic oracle machine accepting $\mathrm{SAT}^X$ uniformly.

## 4. $r$-Generic Oracles

A formula of the form

$$(p_1 \Leftrightarrow X_{i_1}(\mathbf{q}_1) \wedge \cdots \wedge p_r \Leftrightarrow X_{i_r}(\mathbf{q}_r)) \Rightarrow F$$

for $F$ query free is called an $r$-query formula. Restricting the formula to this form is merely a convenience; the important property is that it has at most $r$ queries. If in the definition of t-generic oracle, the relativized formulas are restricted to be $r$-query formulas the oracle is called $r$-generic.

LEMMA 9. *If $F$ is a 1-query formula which is a tautology with respect to some $X$ then there is a unique minimal set $S$ of queries which force $F$ to be a tautology (we say $F$ specifies $S$).*

*Proof.* Suppose $F$ is $(p \Leftrightarrow X_n(q_1, \ldots, q_n)) \Rightarrow G$, and consider any truth assignment to the $q_i$. For at least one value of $p$ all extensions to the remaining atoms must satisfy $G$. If there is only one value of $p$, $X$ is determined at this place. With respect to any $X_n$ having the correct value at the determined places the formula is a tautology.

THEOREM 10. *For any integer $r$, $r$-generic oracles exist.*

*Proof.* By induction on $r$. Define $l(F)$ for a formula $F$ to be the number of occurrences of atoms, 0's, and 1's. For the basis $r=1$ let $k$ be sufficiently large. Assume $X^n$ has been constructed deciding the 1-query formulas involving $X^i$ with $i \leqslant n$. An $X^{n+1}$ extending $X^n$ will be ruled out if there is some formula $F$ involving $X^{n+1}$, such that $F$ specifies a partial function $S$ consistent with $X^{n+1}$ and $|S| > l^k$. If $F$ specifies $S$, let $S_i$, $i=0, 1$, be those queries of $S$ where the first atom $q_1$ is $i$. If $S_0$ is inconsistent with $X^n$ $F$ is irrelevant. Letting $N$ denote $2^n$, if $|S_1| = s_1$ there are $2^{N-s_1}$ $X^{n+1}$ consistent with $X^n$ and $S_1$. Also $S_0$ is specified by $F0/q_1$, so (letting $l$ denote $l(F)$) $|S_0| \leqslant (l-1)^k$, and so if $|S| > l^k$ then $s_1 > l^k - (l-1)^k$. Thus among the $2^n$ $X^{n+1}$ consistent with $X^n$ at most

$$\sum_{l=n+S}^{(2N)^{1/k}} 2^l 2^{N-(l^k-(l-1)^k)}$$

are ruled out. Since this is $o(2^N)$ for $k$ sufficiently large the required $X^{n+1}$ exists. For the induction step, define $k_1 = k$, $k_{s+1} = k \cdot k_s + 1$, and suppose

$X^n$ has been constructed deciding the $s$-query formulas involving $X^i$ with $i \leqslant n$, with polynomial $l^{k_s}$ for $1 \leqslant s \leqslant r$. Proceeding as above, the number of $X^{n+1}$ ruled out by 1-query formulas is determined as above. Now suppose $F$ is $(p \Leftrightarrow X^{n+1}(q_1, ..., q_{n+1})) \Rightarrow G$, where $G$ is an $(s-1)$-query formula. We may further suppose that if $G$ involves $X^i$ then $i \leqslant n$, so that with respect to $X^n$, $F$ is either never a tautology or specifies some set $S$ of queries. In the latter case let $S_1$ be the queries in $S$ with $q_1 = 1$, and let $\hat{S}_1$ be the disjunction of the formulas

$$(q_1 \Leftrightarrow 1 \wedge q_2 \Leftrightarrow t_{i2} \wedge \cdots \wedge q_{n+1} \Leftrightarrow t_{i,n+1}) \Rightarrow (p \Leftrightarrow u),$$

where $X^{n+1}(1, t_{i2}, ..., t_{i,n+1}) \Leftrightarrow u$ are the queries in $S_1$. Then $\hat{S}_1 \Rightarrow G$ is a tautology with respect to $X^n$, and is forced by some query set $T$, which we take as small as possible. $S_1 \cup T$ forces $F$; and if $|S_1| \leqslant l^k$ then for $l$ sufficiently large $|T| \leqslant (l + (2n+4)\,|S_1|)^{k_{s-1}}$ and so $|S_1| + |T| \leqslant l^{k_s}$. The remainder of the proof is similar to the case $r = 1$.

The sums in the above proof can be estimated using geometric series, and it follows that the $r$-generic oracles have positive measure, and hence since they are closed under finite changes measure 1. Thus, the $r$-generic oracles separating $\mathcal{NP}$ and co $\mathcal{NP}$ have measure 1. Let $r\,\mathrm{TAUT}^X$ be the $r$-query formulas which are tautologies with respect to $X$.

THEOREM 11. *If $\mathcal{NP} = \mathrm{co}\,\mathcal{NP}$ then there is a nondeterministic oracle machine $M$ uniformly accepting* $\mathrm{TAUT}^X$, *such that with respect to any $r$-generic $X$, $M$ accepts the members of $r\,\mathrm{TAUT}^X$ in polynomial time.*

*Proof.* Let TAUTX be the formulas which are tautologies with respect to any oracle: TAUTX is in co $\mathcal{NP}$, indeed is complete. For one proof, given a formula

$$(p_1 \Leftrightarrow X^{i_1}(\mathbf{q}_1) \wedge \cdots \wedge p_r \Leftrightarrow X^{i_r}(\mathbf{q}_r)) \Rightarrow G,$$

replace the hypothesis with one requiring that the $p_i$ be consistent, i.e., have the same value when the corresponding $\mathbf{q}_i$ are equal. Thus, by hypothesis $\mathrm{TAUTX} \in \mathcal{NP}$. Let $M$ be the nondeterministic oracle machine which, with the relativized formula $F$ as input, where all queries by $F$ involve $X^i$ with $i \leqslant n$, does the following:

(1) Guesses queries $Q_1, ..., Q_t$ involving $X^n$.

(2) Checks if the $Q_i$ agree with $X$.

(3) If not, rejects.

(4) If so, runs a nondeterministic machine for TAUTX on $Q_1 \wedge \cdots \wedge Q_t \Rightarrow F$.

As a consequence, if there is an $r$-generic $X$ such that $r\mathrm{TAUT}^X \notin \mathcal{NP}^X$ then $\mathcal{NP} \neq \mathrm{co}\,\mathcal{NP}$. The statements about oracles which are forced by sparse sets of queries when true with respect to an $r$-generic oracle are those which can be verified by oblivious deterministic polynomial time oracle machines, which make at most $r$ queries on any input, since the representing formulas for such machines are $r$-query formulas. Direct arguments show that the $r$-generic oracles are closed under complementation and finite changes, and it follows as in (Dowd, 1982) that an $r$-generic oracle is not r.e.

THEOREM 12. *An m-generic oracle is not* 1*-generic.*

*Proof.* The statement that $X$ is 1-generic with polynomial $p$ is $\Pi_1(X)$, and false in any interval.

## 5. CODES FOR THE BOOLEAN FUNCTIONS

The notion of a code for the Boolean functions provides a link between Kolmogorov complexity and the complexity of Boolean functions. Also, some obvious facts about complexity classes can be stated. A code for the integers might be defined as a total recursive function whose range is $\mathcal{N}$. For a code for the Boolean functions, however, we want the arity to be not much bigger than the length of the name.

Define a pre-code for the Boolean functions to be a recursive function $c$: $\mathcal{N} \times \mathcal{N} \to \{0, 1\}$ and a log space computable function $a$: $\mathcal{N} \to \mathcal{N}$. If $|a(x)| = n$ then $x$ is said to be a name for the $n$-ary Boolean function $f$ given by $f(d_{n-1}, ..., d_0) = c(x, d_{n-1} \cdots d_0)$. If each Boolean function has a name, the pre-code is called a code. The complexity of a code is defined to be the complexity of $c$. Code $(c, a)$ is said to simulate code $(d, b)$ if there is a function $s(x)$ computable in log space such that $a(s(x)) = b(x)$ and $|y| \leqslant |b(x)| \Rightarrow c(s(x), y) = d(x, y)$.

The complexity restriction on $s$ is necessary; is $s$ were allowed to be an arbitrary recursive function then any code simulates any other. Indeed define a code to be regular if for some $a$ every $n$-ary Boolean function has a code of length $\leqslant 2^{an}$. Regularity is not a severe restriction, in that any code can be easily modified to yield a regular code simulating it. Between any two regular codes, an exponential space simulation exists.

THEOREM 13. *There is no optimum code, i.e., code which simulates every other code.*

*Proof.* Given a code $(c, a)$ let $\{f_n\}$ be a recursive set of Boolean functions, where $f_n$ is $n$-ary and the shortest code for $f_n$ is of length $\geqslant 2n$. Define

$d(2x, y) = c(x, y)$, $d(2x+1, y) = f_n(y)$, and $b(2x) = a(x)$, $b(2x+1, y) = n$, where $n = |x|$. Then $(c, a)$ does not simulate $(d, b)$.

THEOREM 14. *Circuits are an optimum polynomial time code.*

*Proof.* If $c(x, y)$ is a polynomial time code let $p(n)$ be a polynomial which bounds $|a(x)|$ for $|x| \leqslant n$. Let $C_n$ be the circuit which computes $c(x, y)$ for $|x| \leqslant n$ and $|y| \leqslant p(n)$. Then $C_n$ can be computed from $x$ in log space, and the bits of $x$ plugged in for the $x$ inputs, to obtain a circuit for the Boolean function coded by $x$.

For any of the usual complexity classes, there are sets whose completeness follows by coding computations on inputs of length $n$ and then "plugging in" the bits of $x$, such as the circuit value problem for $\mathscr{P}$. Such constructions yield optimum codes, as in the theorem. For example, quantified Boolean formulas are an optimum polynomial space code, and branching programs an optimum log space code. We give a proof of the latter, for the reader unfamiliar with branching programs.

A branching program is a rooted dag, whose interior vertices are of outdegree 2 and are labelled with atoms, and whose leaves are labelled with $\{0, 1\}$. A truth assignment determines a path from the root to a leaf, by taking the left branch out of a node labelled $p$ if $p$ is assigned 0, else the right branch. A Boolean function is thus determined, where the value at a truth assignment is the label of the leaf of the path determined by the assignment.

THEOREM 15. *The branching programs are an optimum log space code.*

*Proof.* Consider a log space machine restricted to inputs of length $n$, where the machine is assumed to halt in polynomial time on every input. Associate a node with every configuration of the form $(w, p)$ where $w$ is the work tape contents and $p$ is the input tape position. The left edge out of a configuration is the successor configuration when the input bit is 0, and similarly for the right edge.

The formula value problem is known to be complete for log space by alternating log time transformation, provided the formulas are suitably coded (for example, by annotated adjacency matrices of their graphs); see below for a proof. Branching programs simulate formulas with any reasonable coding convention, since the formula value problem is in log space (Lynch, 1977). The converse simulation seems not to hold; thus, it is likely that the value problem for a code can be complete without the code being optimum.

The above fact has been observed by the author and R. Statman, and independently by M. Tompa. For a proof, assume each configuration is

entered at most once, and has at most two possible antecedents. Assuming "true," "false," and "or" are in the connective set, there is a vertex for each configuration. If the configuration has no antecedents, it is a "false" gate, unless it is the input configuration, in which case it is a "true" gate. Otherwise, the left input is the left going antecedent, or "false" if none; and similarly for the right input. If the nodes are "directly addressable" in the representation of the formula than the transformation is in alternating log time.

## 6. Codes and Generic Sets

The following theorem is a relative of Corollary 5.

THEOREM 16. *If $G$ is m-generic then for any code $c$, the Boolean functions $G^n$ do not have names bounded by any recursive subexponential function.*

*Proof.* The statement that the $G^n$ have $c$ codes bounded by a particular recursive subexponential function is $\Pi_1(X)$. Further it is false in any interval, since there are $2^n$ extensions $X^{n+1}$ of $X^n$.

THEOREM 17. *If $G$ is 1-generic then the Boolean functions $G^n$ do not have formulas bounded by any subexponential function.*

*Proof.* Let $F_n$ be a formula for $G^n$, and consider the formula $X^n \Leftrightarrow F_n$. All $2^n$ queries are required to force this to be true. Thus $2^n$ must be polynomial in the length of $F_n$, so the latter is exponential.

## 7. Some Properties of Circuits

Circuits are an important code; they have been extensively studied, and many important questions remain open. This section includes two simple theorems on circuits. The first is an instance of a general notion for codes, and the second an improvement to Lemma 0 of (Kannan, 1981). We assume that in a circuit, $\neg$'s occur only at the inputs; the complexity is the number of $\wedge$, $\vee$ gates. Conventions are such that this is 0 for 0, 1, $x_i$, or $\neg x_i$.

Given a code $(c, a)$ define $u$ to be $n$-universal if $r = |a(u)| - n \geqslant \max\{|a(y)|: |y| \leqslant n\}$; and if $x$ is an $n$-digit binary number (with leading 0's allowed), then $c(x, y) = c(u, yx)$, where $yx$ denotes $y$ with $x$ appended. Since the required Boolean function must have a code, $n$-universal codes exist for all $n$ in any code.

THEOREM 18. *There are n-universal circuits of size* $O(n \log^4 n)$.

*Proof.* A circuit is coded as a list of triples $a = b \circ c$, where $a$, $b$, $c$ are bit strings of length $O(\log g)$ for $g$ the number of gates; the assignments to $b$ and $c$ occur previously in the list (or they are atoms or negated atoms); and the $a$ are increasing in the lexicographic order. The circuit value problem can be solved on a multitape Turing machine in time $O(n \log^2 n)$ as follows. Recursively label the gates of the first half with their output values. Sort the triples of the second half into order by the left input and label the left inputs with their values; do the same for the right input. Sort the second half into original order and recursively label the second half. The overhead is $O(n \log n)$, so the procedure runs in time $O(n \log^2 n)$. The representing circuit has $O(n \log^3 n)$ gates (Pippinger and Fischer, 1979), and so $O(n \log^4 n)$ bits.

THEOREM 19. *Let* $F(n, g)$ *be the n-ary Boolean functions computed by the circuits with n inputs and g gates. If* $|F(n, g)| < 2^{2^n}$ *then* $F(n, g)$ *is a proper subset of* $F(n, 2g + 1)$.

*Proof.* By assumption $F(n, g)$ cannot be closed under conjunction and disjunction. However conjunctions and disjunctions of functions in $F(n, g)$ are in $F(n, 2g + 1)$.

Without further restrictions this is the best possible result, as the example $F(2, 2) = F(2, 1)$ shows.

## 8. CONCLUSION

Theorem 8 can be seen as adding to the evidence that $\mathcal{NP} \neq \text{co}\,\mathcal{NP}$. Uniform machines provide yet another relativized version of unrelativized problems. This theorem is perhaps hardest of all to reconcile with the existence of an $\mathcal{NP}$ algorithm for TAUT. Theorem 11 further substantiates this feeling; one can to some extent adapt unrelativized algorithms to the uniform case. Also, at oracles where $\mathcal{NP} = \text{co}\,\mathcal{NP}$, the machine witnessing this is particular to the oracle. One possible direction for further research might be whether more can be said about this.

RECEIVED August 19, 1985; FINAL MANUSCRIPT RECEIVED May 22, 1990

## REFERENCES

BENNETT, C., AND GILL, J. (1981), Relative to a random oracle $P^A \neq NP^A \neq \text{co} - NP^A$ with probability 1, *SIAM J. Comput.* **10**, 96.

BAKER, T., GILL, J., AND SOLAVAY, R. (1975), Relativizations of the $P = NP$ question, *SIAM J. Comput.* **4** (1975), 431.

BLUM, M., AND IMPAGLIAZZO, R. (1987), Generic oracles and oracle classes, *in* "Proc. 19th Symp. FOCS," pp. 118–126.

CHAITIN, G. (1969), On the length of programs for computing finite binary sequences, *J. Assoc. Comput. Mach.* **16**, 145.

CHAITIN, G. (1975), A theory of program size formally identical to information theory, *J. Assoc. Comput. Mach.* **22**, 329.

DAVIS, M. (1973), Hilbert's tenth problem is undecidable, *Ann. of Math.* **80**, 233.

DOWD, M. (1982), "Forcing and the *P* Hierarchy," Rutgers University Laboratory for Computer Science Research Technical Report No. LCSR-TR-35.

FEFERMAN, S. (1965), Some applications of the notion of forcing and generic sets, *Fund. Math.* **56**, 325.

HINMAN (1969), Some applications of forcing to hierarchy problems in arithmetic, *Z. Math. Logik Grundlag. Math.* **15**, 341.

KANNAN, R. (1981), A circuit size lower bound, *in* "Proc. 22nd FOCS," pp. 304–309.

LYNCH, N. (1977), Log space representation and translation of parenthesis languages, *J. Assoc. Comput. Mach.* **24**, 583.

MEHLHORN, K. (1973), On the size of computable functions, *in* "Proc. 14th IEEE *Symp. Switching and Automata Theory*," pp. 190–196.

PIPPINGER, N., AND FISCHER, M. J. (1979), Relations between complexity measures, *J. Assoc. Comput. Mach.* **26**, 361.

POIZAT, B. (1986), $Q = NQ?$, *J. Symbolic Logic* **51**, 22.

REGAN, K. (1984), personal communication.

ROGERS, H. JR. (1967), "Theory of Recursive Functions and Effective Computability," McGraw–Hill, New York.

WEISS, M. (1980), "Arithmetic with a Distinguished Predicate: Diophantine Relations and Existentially Complete Models," Ph.D. thesis, Mathematics Department, Rutgers University.