SOME CONNECTIONS BETWEEN NONUNIFORM AND UNIFORM COMPLEXITY CLASSES

Richard M. Karp[1] and Richard J. Lipton[2]

University of California, Berkeley

1. *Introduction*

With any subset S of $\{0,1\}^*$ we may associate a sequence of boolean functions defined by

$$S_n : \{0,1\}^n \to \{0,1\}$$

where $S_n(x_1,x_2,\cdots,x_n) = 1$ if and only if $x_1x_2\cdots x_n$ is in S. As usual, let $L(S_n)$ denote the minimum number of gates in a boolean circuit realizing S_n. Then we say that S *has small circuits* if the function $L(S_n)$ is bounded by a polynomial in n.

It is well known that every set in P has small circuits [13]. Adleman [1] has recently proved the stronger result that every set accepted in polynomial time by a randomized Turing machine has small circuits. Both these results are typical of the known relationships between uniform and nonuniform complexity bounds. They obtain a nonuniform upper bound as a consequence of a uniform upper bound.

The central theme here is an attempt to explore the converse direction. That is, we wish to understand when nonuniform upper bounds can be used to obtain uniform upper bounds. The immediate answer is "not always:" clearly there are sets with small circuits that are not even recursive. The very trivial nature of such "counter examples" suggests, however, that a more careful investigation may still yield insight. Indeed, as we will show, if one considers not arbitrary sets but rather "well behaved ones" it is possible to achieve our goal. For example, we will show that

> if SAT has small circuits, then the Meyer-Stockmeyer [10] hierarchy collapses.

Thus, here is an example of a nonuniform upper bound that has uniform consequences. The proof, of course, will depend on the fact that SAT is not a "pathological" set, but is rather well behaved.

2. *Nonuniform Complexity Measures*

In this section we will define our basic notion of nonuniform complexity. Then we will show how to relate it to more common notions.

Let S be a subset of $\{0,1\}^*$. Let $h : N \to \{0,1\}^*$ where N is the set of natural numbers. Define $S : h = \{wx : x \in S \text{ and } w = h(|x|)\}$. Next let S be any collection of subsets of $\{0,1\}^*$ and let F be any collection of functions from N to N. The key definition is

$$S/F = \{S : \exists h \text{ with } \lambda n.|h(n)| \text{ in } F \text{ and } S : h \text{ in } S\}$$

Intuitively, S/F is the collection of subsets of $\{0,1\}^*$ that can be accepted by S with F "advice." The idea behind this definition is foreshadowed in papers by Pippenger [11] and Plaisted [12].

We are mainly interested in *poly* the collection of all polynomially-bounded functions and *log* the collection of all functions that are $O(\log n)$. Indeed, many of our results will concern the classes P/poly and P/log.

If f is a function, S/f is synonymous with $S/\{f\}$. Some preliminary facts are:

(1) for all S, $S/0 = S$;

(2) any subset of $\{0,1\}^*$ is in $P/2^n$;

(3) if f is infinitely often nonzero, then P/f contains nonrecursive sets;

(4) if $g(n) < f(n) \leq 2^n$ (i.o.) then $P/f \not\subseteq P/g$

The classes P/poly and P/log can be characterized in terms of classic circuit complexity.

An n-input m-gate *combinatorial circuit* C is a function $C : \{n+1,\cdots,n+m\} \to \{0,1\}^4 \times \{1,\cdots,n+m\}^2$ satisfying: if $C(i) = \langle B,j,k\rangle$ then $j < i$ and $k < i$. The interpretation of C is that gate i uses the truth table B on its inputs j and k to produce its output. In the usual way we define what it means for a circuit C to *realize* the boolean function f. Then let $L(f)$ denote the minimum number of gates in a combinatorial circuit realizing the boolean function f. Next, as in the introduction, if S is a subset of $\{0,1\}^*$, then $S_n : \{0,1\}^n \to \{0,1\}$ is defined by

This research was supported in part by NSF grants [1]MCS77-09906 and [2]MCS79-20409.

$$S_n(x_1,x_2,\cdots x_n) = \begin{cases} 1, & \text{if } x_1x_2\cdots x_n \in S \\ 0, & \text{otherwise} \end{cases}$$

Finally, recall that a set S has *small circuits* if $L(S_n)$ is bounded by a polynomial in n.

The following simple theorem, which is given in [11], characterizes P/poly.

Theorem 2.1: Let S be a subset of $\{0,1\}^*$. Then the following are equivalent:

(1) S has small circuits.

(2) S is in P/poly.

We now turn to the more difficult task of characterizing P/log. In order to do this we need to introduce a further notion.

Let C be a n-input m-gate combinatorial circuit. Define d to be $\lceil \log_2(n+m) \rceil$. We will then define function $\hat{C} : \{0,1\}^{3d+4^2} \to \{0,1\}$ by

$$\hat{C}(\hat{k},B,\hat{j},\hat{k}) = \begin{cases} 1, & \text{if } C(i) = \langle B,k,j \rangle \text{ ;} \\ 0, & \text{otherwise.} \end{cases}$$

(Here $\hat{x}$ is the binary representation of x.) Note, i,j, and k range over $1,\ldots n+m$ and so can be represented in d bits each. Intuitively, if C is a combinatorial circuit, then $\hat{C}$ is the boolean function that describes the gate connections of C. A set S *has small circuits with easy descriptions* if there are sequences of circuits C_n and D_n such that

(1) C_n realizes S_n and D_n realizes $\hat{C}_n$;

(2) C_n has at most n^{ℓ_1} gates and D_n has at most $\ell_2 \log n$ gates (ℓ_1, ℓ_2 constants).

Then,

Theorem 2.2 Let S be a subset of $\{0,1\}^*$. Then S in P/log implies that S has small circuits with easy descriptions.

The proof of this is a simple refinement of the standard arguments that connect Turing machine complexity and circuit complexity. It is possible to obtain a converse to the above theorem at the cost of a further refinement of the notion of "small circuits." We defer such a statement to the final version of this paper. We do note for now the following partial converse:

Theorem 2.3: Let S be a subset of $\{0,1\}^*$. Then if S has small circuits with easy descriptions, then S is in P/F where F is the collection of $(\log n \log\log n)$ bounded functions.

Another way we can gain insight into our classes S/F is to use them to restate other known results. For example, the result in [2] that there are short universal traversal sequences for undirected graphs can be restated as

UGAP is in DSPACE(log n)/poly .

Here UGAP is the undirected maze problem. As another example, we have Adleman's [1] result that R has small circuits which can be restated as

R is a subset of P/poly .

It may be interesting that both these results use the probabilistic method of Erdös to prove the existence of the required advice bits.

3. Summary of Main Results

We will study a variety of complexity classes. These include the basic time and space classes DTIME(t), DSPACE(s) and NSPACE(s) and the classes:

P = the set of languages accepted in deterministic polynomial time,

R = the set of languages accepted in random polynomial time [1],

NP = the set of languages accepted in nondeterministic polynomial time,

PSPACE = the set of languages accepted in polynomial space,

EXPTIME = $\bigcup_{i>0}$ DTIME(2^{n^i}) .

Also important is the P-time hierarchy of Meyer and Stockmeyer [10]. We let $\sum_i^p$ (respectively π_i^p) denote those languages accepted in polynomial time by Turing machines that make i alternations starting from an existential (respectively universal) state. Note that P is $\sum_0^p$ and π_0^p and NP is $\sum_1^p$. Finally, note that P, PSPACE and EXPTIME can be viewed as complexity classes associated with alternating Turing machines; specifically, P = ASPACE(log n), PSPACE = AP and EXPTIME = APSPACE [3,8].

The organization of the remaining sections is according to proof methods. The hypothesis of each theorem tells us that $K = S : h$, where S is in a given complexity class and a bound on $|h(|x|)|$ is known. A uniform algorithm to test whether $x \in K$ is not given $h(|x|)$; however, it can exploit the fact that $h(|x|)$ is consistent; i.e., for all strings y of the same length as x, $y \in K \Leftrightarrow h(|x|) \cdot y \in S$. The algorithm must somehow filter through all the strings that might be $h(|x|)$, and come up with the right decision about x. The method of doing so depends on the structure of K. The following section treats the case where K is a "game." Section 5 considers the case where K is self-reducible. Finally, Section 6 deals with the case where K has a simple recursive definition.

The main results of this paper are summarized in Figure 1. The rest of this paper is devoted to supplying proofs and additional comments on these main results. As promised in the introduction each result demonstrates that a nonuniform hypothesis can have uniform consequences.

4. The Round-Robin Tournament Method

Insight into the nature of a complexity class can often be gained by identifying "hardest" problems in the class; i.e., problems that are complete in the class with respect to an appropriate definition of reducibility. For complexity classes defined in terms of time and space on alternating Turing machines, these complete problems often take the form of games ([3,4]). In this section we explain and apply a proof technique called "the round-robin tournament method," which enables us to relate

the nonuniform complexity of a game to its uniform complexity. The specific complexity classes we consider are PSPACE, P and EXPTIME (alias AP, ASPACE($\log n$) and APSPACE, respectively ([3,8])).

A *game* G is specified by

(i) a set $W \subseteq \{0,1\}^*$ and

(ii) a pair of length-preserving functions F_0 and F_1, each mapping $\{0,1\}^* - W$ into $\{0,1\}^*$.

There is a straightforward interpretation of this structure as a game of perfect information. Each string $x \in \{0,1\}^*$ is a possible position in the game. Starting in an initial position, the players move alternately until a position in W is reached. When a player is to move in position x, he may move either to $F_0(x)$ or to $F_1(x)$. When a position in W is reached, the player to move is declared the winner. Note that all the positions arising in a single play of the game have the same length.

We further require that our games be *terminating;* i.e.,

(iii) there is no sequence of moves leading from a position x back to itself.

Given a game G, let G denote the set of positions from which the first player can force a win. The set G is specified recursively by

$$G = W \cup \{x \mid F_0(x) \notin G\} \cup \{x \mid F_1(x) \notin G\}$$

This specification of G suggests the following method of selecting an optimal move in any position $x \in W$: move to $F_0(x)$ if $F_0(x) \notin G$; otherwise move to $F_1(x)$. If $x \in G$, then this method of move selection will force a win against any choice of moves by the opponent.

Let us now apply nonuniform complexity to games. Suppose $G = S : h$, where $S \subseteq \{0,1\}^*$ and h is a function from N into $\{0,1\}^*$. Then

$$x \in G \iff h(|x|) \cdot x \in S .$$

The optimal move selection rule can be restated as follows:

in any position $x \notin W$, move to $F_0(x)$ if $h(|x|) \cdot F_0(x) \notin S$, and otherwise move to $F_1(x)$.

We would like to consider situations in which $G = S : h$, but $h(|x|)$ is not known. If we guess that $h(|x|) = w$, then the following move selection rule is indicated:

in any position $x \notin W$, move to $F_0(x)$ if $w \cdot x \notin S$, and otherwise move to $F_1(x)$. Call this rule Strat(w).

Given strings w,w' and x, the predicate Win(w,w',x) is defined as follows: play out position x with the first player choosing his moves according to Strat(w), and the second player using Strat(w'); Win(w,w',x) is true if the first player wins.

The following easy lemma is the basis of the round-robin tournament proof technique.

Lemma 4.1: Let G be a game, S a subset of $\{0,1\}^*$ and h a function from N into $\{0,1\}^*$, such that $G = S : h$. Let w and w' range over some set of strings T(x) which includes $h(|x|)$. Then,

$$x \in G \iff$$

(1) $\exists w \forall w'$ Win(w,w',x) $\iff$

(2) $\forall w' \exists w$ Win(w,w',x).

Proof: If $x \in G$ then the sentence $\forall w'$(Win($h(|x|)$,w',x) is true. Hence (1) and (2) are true. If $x \notin G$ then, for all w, Win(w,$h(|x|)$,x) is false; hence (1) and (2) are false. □

Lemma 4.1 suggests how to decide if $x \in G$ when $h(|x|)$ is not known but a set T(x) containing $h(|x|)$ is known. Simply play a round-robin tournament among the strategies associated with all the strings in T(x), starting each game in position x. Then $x \in G$ if and only if some strategy emerges undefeated. A subtle point is that the round-robin tournament method determines whether $x \in G$ without necessarily identifying $h(|x|)$.

To prepare for the applications of the round-robin tournament method, we assert the existence of games with certain properties.

Fact 1: There is a game G such that the associated set G is complete in EXPTIME with respect to many-one polynomial-time reducibility. Moreover, the set W is in P, and the functions F_0 and F_1 are computable in polynomial time.

Fact 2: There is a game G such that G is complete in PSPACE with respect to many-one polynomial-time reducibility. For this game, the set W is in P, and the functions F_0 and F_1 are computable in polynomial time. Moreover, there is a polynomial $p(\cdot)$ such that, for every position x, every play of G starting at x terminates within $p(|x|)$ moves.

Fact 3: There is a game G such that G is complete in P with respect to many-one logspace reducibility. For this game W is in logspace and the functions F_0 and F_1 are computable in logspace. Moreover, each position $x \in W$ consists of the concatenation of a *fixed part* x_1 with a *variable part* x_2, such that $F_0(x)$ and $F_1(x)$ have the same fixed part as x does. Also, if $|x_1| = n$, then $|x_2| = f(n)$, where $f(n)$ is a nondecreasing function which is $\leq 3\log_2 n$.

Facts 1 and 3 may be derived by simple modifications and encodings of games described in [3]. One of the modifications is to encode into each nonterminal position a "clock" which is decremented at each move; this is done to ensure termination. Similarly, a game of the type referred to in Fact 2 can be derived from any of several PSPACE-complete games derived in [14].

We are now ready to give the main theorems of this section.

Theorem 4.2: If PSPACE $\subseteq$ P/poly then PSPACE = $\Sigma_2^P \cap \Pi_2^P$.

Proof: Since $\Sigma_2^p \cap \Pi_2^p \subseteq$ PSPACE, it suffices to prove PSPACE $\subseteq$ P/poly $\Rightarrow$ PSPACE $\subseteq \Sigma_2^p \cap \Pi_2^p$. For this it is sufficient to show

$$G \in P/poly \Rightarrow G \in \Sigma_2^p \cap \Pi_2^p ,$$

where G is the PSPACE-complete set described in Fact 2. Suppose $G \in P/poly$. Then there is a set $S \in P$, a positive constant k, and a function $h : N \to \{0,1\}^*$ such that $|h(n)| \leq k + n^k$, so that $G = S : h$. By lemma 4.1, $x \in G \iff \exists w \forall w' Win(w,w',x)$. Here each of w and w' ranges over all strings of length $\leq k + |x|^k$. Since F_0 and F_1 are polynomial-time computable, W is polynomial-time recognizable and play from x terminates within $p(|x|)$ moves, the predicate Win(w,w',x) is computable in polynomial time. Thus $G \in \Sigma_2^p$. Similarly, since $x \in G \iff \forall w' \exists w Win(w,w',x)$, it follows that $G \in \Pi_2^p$. □

Theorem 4.3: PSPACE $\subseteq$ P/log $\iff$ PSPACE = P.

Proof: Since P $\subseteq$ PSPACE, and since PSPACE = P implies PSPACE $\subseteq$ P/log, it suffices to prove

$$PSPACE \subseteq P/log \Rightarrow PSPACE \subseteq P .$$

For this it suffices to show $G \in P/log \Rightarrow G \in P$, where G is the PSPACE-complete set described in Fact 2. Again, the round-robin tournament method yields the proof. Suppose $G \in P/log$. Then there is a set $S \in P$, a positive constant k, and a function $h : N \to \{0,1\}^*$ such that $h(n) \leq k\log_2 n$, so that $G = S : h$. Then $x \in G \iff \exists w \forall w' Win(w,w',x)$, where w and w' range over the $O(|x|^k)$ strings of length $\leq k\log_2|x|$. Since Win(w,w',x) can be computed in polynomial time, we can decide in polynomial time whether $x \in G$ by actually enumerating the polynomially-many pairs (w,w'), computing Win(w,w',x) for each pair, and determining whether some strategy Strat(w) indeed wins from x against all the competing strategies. Thus $G \in P$. □

Theorem 4.4: EXPTIME $\subseteq$ PSPACE/poly $\iff$ EXPTIME = PSPACE.

Proof: The proof is almost a carbon copy of the proof of theorem 4.3. It suffices to show that

$$G \in PSPACE/poly \Rightarrow G \in PSPACE,$$

where G is the game referred to in Fact 1. Suppose $G \in$ PSPACE/poly. Then $G = S : h$, where $S \in$ PSPACE and $|h(|x|)| \leq k + |x|^k$, for some k. Then $x \in G \iff \exists w \forall w' Win(w,w',x)$ where w and w' range over all strings of length $\leq k + |x|^k$. Since w, F_0 and F_1 are computable in polynomial time (and thus certainly in polynomial space) the predicate Win(w,w',x) is computable in polynomial space; it suffices to play out the game from x, alternately using Strat(w) and Strat(w') for move selection; this simulation requires repeated calls on the polynomial-space recognizer for S. Thus the truth of the formula

$$\exists w \forall w' Win(w,w',x)$$

can be decided in polynomial-space by simply running through the pairs (w,w'), and evaluating Win(w,w',x) for each pair. It follows that $G \in$ PSPACE. □

The last in our clone of four theorems proved by the round-robin tournament method is the following.

Theorem 4.5: For any positive integer ℓ, $P \subseteq DSPACE((\log n)^\ell)/\log n \iff P \subseteq DSPACE((\log n)^\ell)$.

Proof: It suffices to prove $G \in DSPACE((\log n)^\ell)/\log n \Rightarrow G \in DSPACE((\log n)^\ell)$, where G is the set described in Fact 3. Suppose $G \in DSPACE((\log n)^\ell)/\log n$. Then $G = S : h$ where $S \in DSPACE((\log n)^\ell)$ and $|h(x)| \leq k\log_2|x|$, for some k. Then $x \in G \iff \exists w \forall w' Win(w,w',x)$, where w and w' range over all strings of length $\leq k\log_2|x|$. Clearly space $O((\log n)^\ell)$ suffices to deterministically enumerate all pairs (w,w') and, for each, to play out Strat(w) against Strat (w') from position x, with the help of repeated calls on a deterministic space-$(\log n)^\ell$ recognizer for S. It follows that $G \in DSPACE((\log n)^\ell)$. □

5. *The Self-Reducibility Method*

The "hardest" problems in complexity classes defined by bounds on nondeterministic time or space can usually be described conveniently in terms of existential quantification; i.e., the problem is specified as a set $K \subseteq \{0,1\}^*$, where

$$x \in K \iff \exists y \mathcal{R}(x,y) .$$

Here x and y denote strings, and $\mathcal{R}$ is an easy-to-compute predicate.

For example, in the satisfiability problem, whichis NP-complete, x is the encoding of a propositional formula, y is the encoding of a truth assignment, and $\mathcal{R}(x,y)$ means "y satisfies x." In the directed graph reachability problem, which is logspace complete in NSPACE(log n), x is the encoding of a directed graph with distinguished vertices s and t, y is the encoding of a sequence of edges, and $\mathcal{R}(x,y)$ means "y is a path from s to t."

Such problems often enjoy a "self-reducibility" property. Various formal definitions of self-reducibility can be found in the literature ([9, 15, 17]). Here is one version of the idea: a *self-reducibility structure* for K is specified by a well-founded partial ordering α of $\{0,1\}^*$ such that

(i) A, the set of minimal elements in α, is recursive and

(ii) if $x \in A$, then $x \in K \iff \mathcal{R}(x,\lambda)$ is true,

together with a pair of computable functions G_0 and G_1 mapping $\{0,1\}^*$-A into $\{0,1\}^*$, such that

(iii) for all x, $G_0(x)\alpha x$, $G_1(x)\alpha x$, and $|G_0(x)| = |G_1(x)| = |x|$;

(iv) for all x and z,

$$\mathcal{R}(x,0z) \equiv \mathcal{R}(G_0(x),z) \quad \text{and}$$
$$\mathcal{R}(x,1z) \equiv \mathcal{R}(G_1(x),z)$$

To illustrate the concept, we give self-reducibility structures for two important examples. The first example is the satisfiability problem for propositional formulas, encoded so that the following property holds: Let $F(t_1,t_2,\cdots,t_n)$ be a formula in which the variables $t_1,t_2,\cdots,t_n$ appear, and let $F(a,t_2,\cdots,t_n)$ be the same formula with the boolean constant a substituted for t_1. Let $\langle F(t_1,t_2,\cdots,t_n)\rangle$ and $\langle F(a,t_2,\cdots,t_n)\rangle$ denote the encodings of these two formulas as strings. Then $|\langle F(t_1,t_2,\cdots,t_n)\rangle| = |\langle F(a,t_2,\cdots,t_n)\rangle|$. Let SAT denote this version of the satisfiability problem. Then $x \in SAT \Leftrightarrow \mathcal{R}(x,y)$, where $\mathcal{R}(x,y)$ is true iff

$$x = \langle F(t_1,t_2,\cdots,t_n)\rangle\ ,\ y \in \{0,1\}^n\ ,$$

and the substitution $t_i = y_i$, $i = 1,2,\cdots,n$ makes F true. The set SAT has a self-reducibility structure in which A is the set of propositional formulas containing no variables,

$$G_0(\langle F(t_1,t_2,\cdots,t_n)\rangle) = \langle F(0,t_2,\cdots,t_n)\rangle \quad \text{and}$$
$$G_1(\langle F(t_1,t_2,\cdots,t_n)\rangle) = \langle F(1,t_2,\cdots,t_n)\rangle\ .$$

As a second example, let DAG denote the set of encodings of triples $(\Psi,s,t,)$ such that

(i) Ψ is a directed acyclic graph in which the out-degree of each vertex is either 0 or 2; if v has out-degree 2 then its successor vertices are denoted v_0 and v_1;

(ii) s is a vertex and t is a vertex of out-degree 0;

(iii) there exists a directed path from s to t.

It is clear that DAG is self-reducible. Take $\mathcal{R}(x,y)$ to be true if x encodes a triple (Ψ,s,t) and the sequence of 0's and 1's y indicates a path from s to t. Let A be the set of triples (Ψ,s,t) such that s is of out-degree 0; and let $G_0((\Psi,s,t)) = (\Psi,s_0t)$ and $G_1((\Psi,s,t)) = (\Psi,s_1,t)$.

When a set K has a self-reducibility structure, any algorithm to test membership in K can be used in a recursive manner to compute a function witness(x) with the property that $x \in K$ if and only if $\mathcal{R}(x,\text{witness}(x))$ is true. The recursive definition of this function is:

$$\begin{aligned}\text{witness}(x) = \text{if } & x \in A \text{ then } \lambda \text{ else}\\ & \text{if } G_0(x) \in K \text{ then}\\ & 0\cdot\text{witness}(G_0(x)) \text{ else}\\ & 1\cdot\text{witness}(G_1(x)).\end{aligned}$$

The self-reducibility of K also enables connections to be made between the nonuniform complexity of K and its uniform complexity. These connections are based on the following lemma.

Lemma 5.1: Suppose K has a self-reducibility structure (α,A,G_0,G_1) and $K = S : h$. For any string w define witness(w,x) by the following recursive procedure:

$$\begin{aligned}\text{witness}(w,x) = \text{if } & x \in A \text{ then } \lambda \text{ else}\\ & \text{if } w\cdot G_0(x) \in S \text{ then}\\ & 0\cdot\text{witness}(w,G_0(x))\\ & \text{else } 1\cdot\text{witness}(w,G_1(x)).\end{aligned}$$

Then $x \in K \Leftrightarrow \exists w\, \mathcal{R}(x,\text{witness}(w,x))$, provided w ranges over some set which includes $h(|x|)$.

Proof: It suffices to note that

$$x \in K \Leftrightarrow \mathcal{R}(x,\text{witness}(h(|x|),x)) \text{ is true.}$$

Lemma 5.1 suggests a uniform way of testing membership in K: for each w in a suitable set, compute witness(w,x) and test whether $\mathcal{R}(x,\text{witness}(w,x))$ is true. The complexity of this algorithm will depend on the time and space needed to test membership in A, S and $\mathcal{R}$, on the lengths of chains in the partial ordering α, and on the number of strings w that need to be considered.

Now we are ready to give some applications of self-reducibility.

Theorem 5.2: $P = NP \Leftrightarrow NP \subseteq P/\log$.

Proof: The implication $P = NP \Rightarrow NP \subseteq P/\log$ is immediate. Since SAT is NP-complete, the reverse implication will follow once we prove that

$$SAT \in P/\log \Rightarrow SAT \in P\ .$$

Assume that $SAT \in P/\log$. Then $SAT = S : h$, where $S \in P$ and, for some k, $|h(n)| \leq k\log_2 n$.

Using the self-reducibility structure for SAT given above, coupled with the method of lemma 5.1, we can test whether a string x is in SAT. It is necessary to compute witness(w,x) for each of the polynomially-many strings w of length $\leq k\log_2 n$ and, for each, to test whether $\mathcal{R}(x,\text{witness}(w,x))$ is true. Each such computation can be done in polynomial time. Hence we conclude that $SAT \in P$. □

By similar methods we can relate the nonuniform and uniform complexities of other self-reducible problems. For example, we can state the following result.

Theorem 5.3: Let Factor denote the set of triples of integers $\langle x,y,z\rangle$ such that x has a factor between y and z. Then

$$\text{Factor} \in P/\log \Leftrightarrow \text{Factor} \in P.$$

As another application of the self-reducibility method, we give the following theorem.

Theorem 5.4: $NSPACE(\log n) \subseteq DSPACE(\log n)/\log \Leftrightarrow NSPACE(\log n) = DSPACE(\log n)$.

Proof: It is sufficient to prove $NSPACE(\log n) \subseteq DSPACE(\log n)/\log \Rightarrow NSPACE(\log n) = DSPACE(\log n)$.

Since DAG is logspace complete in $NSPACE(\log n)$, it suffices to show that

$$DAG \in DSPACE(\log n)/\log \Rightarrow DAG \in DSPACE(\log n)\ .$$

Suppose that $DAG = S : h$, where $S \in DSPACE(\log n)$ and $|h(n)| \leq k\log_2 n$. Then, guided by the self-

reducibility of DAG, we can test whether $(\Psi,s,t) \in$ DAG by performing the following computation for each string w of length $\leq k\log_2 n$:

v:=s;
while v has out-degree 2 do
v:= if $w\cdot(\Psi,v_0,t) \in S$ then v_0 else v_1 .

If v is ever set equal to t then accept (Ψ,s,t); otherwise, reject it. It is clear that this method recognizes DAG deterministically within space $O(\log n)$. □

6. *The Method of Recursive Definition*

Let K be a subset of $\{0,1\}^*$, and let $C_K : \{0,1\}^* \to \{0,1\}$ be the characteristic function of K. By a recursive definition of C_K we mean a rule that specifies C_K on a "basis set" $A \subseteq \{0,1\}^*$, and uniquely determines C_K on the rest of $\{0,1\}^*$ by a recurrence formula of the form

$$C_K(x) = F(x,C_K(F_1(x)),C_K(f_2(x)),\cdots,C_K(f_t(x)),$$
$$x \in \{0,1\}^* - A .$$

Example 1: Let G be a game, as defined in Section 4 , and let G be the set of positions from which the player to move can force a win. Then G is uniquely determined by

(i) if $x \in W$ then $x \in G$
(ii) if $x \in \{0,1\}^* - W$ then
$x \in G \iff F_0(x) \notin G$ or $F_1(x) \notin G$.

Example 2: Let (α,A,G_0,G_1) be a self-reducibility structure for the set $K \subseteq \{0,1\}^*$. Then K is determined uniquely by its intersection with A, together with the recurrence

for $x \in K$, $x \in K \iff G_0(x) \in K \cup G_1(x) \in K$.

Example 3: If A is a $n\times n$ matrix of 0's and 1's, then the *permanent* of A is defined by $\mathrm{Perm}(A) = \sum_{\sigma} \prod_{i=1}^{n} a_{i,\sigma(i)}$, where σ ranges over all the permutations of $\{1,2,\cdots,n\}$. The function Perm(·) is uniquely defined by the recurrence formula

Perm(A) = 0 if A is a matrix of 0's;
Perm(A) = $(A)_{11}$ if A is a 1×1 matrix; else
Perm(A) = Perm(A') + Perm(A'') ,

where: (i,j) is the lexicographically first position in A containing a 1, A' is the matrix obtained from A by setting the (i,j) entry to 0, and A'' is the matrix obtained from A by crossing out the i^{th} row and j^{th} column of A.

Let us define a set $\mathrm{PER} \subseteq \{0,1\}^*$ as follows: PER is the set of strings which encode an $n\times n$ matrix A of 0's and 1's, and an integer i, such that the i^{th} digit of the binary representation of Perm(A) is a 1. Then the above recurrence formula for the function Perm(·) translates into a simple recursive definition for the characteristic function of the set PER.

The theme of the present section is that, when C_K has a simple enough recursive definition, bounds on the nonuniform complexity of K yield bounds on its uniform complexity. The idea is as follows. Suppose $K = S : h$, and C_K is determined by its values on A, together with the recurrence formula

$$C_K(x) = F(x,C_K(f_1(x)),C_K(f_2(x)),\cdots,C_K(f_t(x)),$$
$$x \in \{0,1\}^* - A,$$

where $|f_1(x)| = |f_2(x)| = \cdots = |f_t(x)| = |x|$.
For any string w, define $K_w = \{x \mid wx \in S\}$. Then, for $x \in A$, we can make the following assertion:

$$x \in K \iff \exists w\,[x \in K_w \wedge \forall y\,[C_{K_w}(y) = F(y,C_{K_w}(f_1(y)),\cdots,C_{K_w}(f_t(y))] .$$

Here, w ranges over all strings of length $|h(|x|)|$, and y ranges over all strings of the same length as x. The above formula suggests a uniform algorithm to test membership in K.

As an illustration of this approach, we prove that, if NP has small circuits, then $\bigcup_i \sum_i^p = \sum_2^p$, i.e., the polynomial-time hierarchy collapses. Originally we have proved this with $\sum_2^p$ replaced by $\sum_3^p$. This improvement is due to M. Sipser.

Theorem 6.1: If $NP \subseteq P/poly$ then $\sum_2^p = \bigcup_{i=1}^{\infty} \sum_i^p$.

The proof of this theorem requires the following lemma.

Lemma 6.2: If $NP \subseteq P/poly$ then $\bigcup_{i=1}^{\infty} \sum_i^p \subseteq P/poly$.

Proof: Let E_i be the set of encodings of true sentences of the form

(*) $Q_1\vec{x}_1Q_2\vec{x}_2\cdots Q_i\vec{x}_i\ F(\vec{x}_1,\vec{x}_2,\cdots,\vec{x}_i)$

where $Q_1 = \exists$, the Q_j are alternately $\exists$ and $\forall$, x_j is shorthand for the tuple $x_{j_1},x_{j_2},\cdots x_{j,r_j}$ of boolean variables, and F is a propositional formula. Let A_i be defined in the same way, except that $Q_1 = \forall$. It is known that E_i is logspace complete in $\sum_i^p$, and A_i is logspace complete in Π_i^p . Also, it is clear that $A_i \in P/poly \iff E_i \in P/poly$. It suffices for the lemma to prove that $E_i \in P/poly$ for all i.

By hypothesis, $E_1 \in P/poly$. We proceed by induction on i. Assume $E_{i-1} \in P/poly$; then $A_{i-1} \in P/poly$. Thus there exists a set $S \in P$, a constant k and a function $h : N \to \{0,1\}^*$ such that $|h(n)| \leq k + n^k$ and $x \in A_{i-1} \iff h(|x|)\cdot x \in S$.

If y is the encoding of a sentence of the form (*), and $\vec{a}$ is a r_1-tuple of boolean variables, let $y_{\vec{a}}$ denote the encoding of the sentence

that results from y by deleting the quantifier Q_1 and substituting $\vec{a}$ for $\vec{x}_i$ in $F(\vec{x}_1,\vec{x}_2,\cdots\vec{x}_i)$. We choose our encoding conventions and method of substitution so that the length of $y_{\vec{a}}$ is equal to the length of y.

Since $S \in P$, the following set T is in NP:

$$T = \{wy \mid \text{for some } \vec{a},\ w \cdot y_{\vec{a}} \in S\} .$$

By hypothesis $T \in P/poly$, so there exists $S' \in P$, $k' \in N$ and $h' : N \to \{0,1\}^*$ so that $|h'(n)| \leq k' + n^{k'}$ and $x \in T \iff h'(|x|)\cdot x \in S'$. Then $y \in A_1 \iff$ for some $\vec{a}$, $y_{\vec{a}} \in E_{i-1} \iff$ for some a, $h(|y_{\vec{a}}|)\cdot y_{\vec{a}} \in S \iff h(|y_{\vec{a}}|)\cdot y \in T \iff h'(|h(|y_{\vec{a}}|)\cdot y|)\cdot h(|y_{\vec{a}}|)\cdot y \in S'$. But the prefix $h'(|h(|y_{\vec{a}}|)\cdot y|)\cdot h(|y_{\vec{a}}|)$ is a polynomial-bounded function of $|y|$; also $S' \in P$. These two facts together establish that $A_i \in P/poly$. □

Proof of Theorem 6.1: It suffices to prove that $NP \subseteq P/poly \Rightarrow \Pi_3^p \subseteq \Sigma_2^p$; for this it is sufficient to prove that the set A_3 is in Σ_2^p. Our proof is based on the fact that A_3 has an easy-to-evaluate recursive definition of the form $C_{A_3}(y) = \mathcal{R}(y, C_{A_3}(y'), C_{A_3}(y''))$. Consider a sentence y of the form

$$Q_1 x_1 Q_2 x_2 \cdots Q_n x_n \ F(x_1, x_2, \cdots, x_n)$$

where the string of quantifiers $Q_1 Q_2 \cdots Q_n$ is contained in $\forall^* \exists^* \forall^*$. Let $y' = Q_2 x_2 \cdots Q_n x_n \ F(0, x_2, \cdots x_n)$ and $y'' = Q_2 x_2 \cdots Q_n x_n \ F(1, x_2, \cdots, x_n)$. Then $C_{A_3}(y)$ = if $Q_1 = \forall$ then $C_{A_3}(y') \wedge C_{A_3}(y'')$ else $C_{A_3}(y') \vee C_{A_3}(y'')$. C_{A_3} is uniquely determined by this recursive definition which is of the form $C_{A_3}(y) = \mathcal{R}(y, C_{A_3}(y'), C_{A_3}(y''))$, coupled with its values on the "basis set" consisting of sentences without quantifiers.

By lemma 6.2, $A_3 \in P/poly$. Thus $A_3 = S : h$ where $S \in P$ and $|h(n)| \leq k + n^k$. For each $w \in \{0,1\}^*$ define $f_w : \{0,1\}^* \to \{0,1\}$ by $f_w(x) = 1 \iff wx \in S$. Then membership of y in A_3, in the case where y contains at least one quantifier, is expressed by the following formula:

$$(**)\ \exists w \forall z [f_w(y) = 1 \wedge f_w(z) = \mathcal{R}(z, f_w(z'), f(z''))] .$$

Here w ranges over all strings of length $\leq x + |y|^k$, and z ranges over all strings of length $|y|$. Also, with the help of a polynomial-time algorithm to test membership in S, the property $f_w(y) = 1$ and $f_w(z) = \mathcal{R}(z, f_w(z_1), f_w(z_2))$ can be tested in polynomial time. Thus the $\exists \forall$ form of (**) establishes that $A_3 \in \Sigma_2^p$. □

There are a number of corollaries of theorem 6.1.

Corollary 6.3: If $R = NP$ then $\bigcup_i \Sigma_i^p = \Sigma_2^p$.

This follows immediately from the observation [1] that every set in R has small circuits.

The next corollary concerns sparse sets. A set S is *sparse* [5,6] if

$$\exists c \forall n \geq 2 \ |S \cap \{0,1\}^n| \leq n^c .$$

Corollary 6.4: If there is a sparse set S that is complete in NP with respect to polynomial-time Turing reducibility (cf. Cook [4]), then

$$\bigcup_i \Sigma_i^p = \Sigma_2^p .$$

This corollary follows immediately from theorem 6.1 once it is noted (cf. Meyer [6]) that the existence of such an S implies that every set in NP has small circuits. Corollary 6.4 should be compared with a result of Fortune [5] which shows that, if there exists a sparse set whose complement is complete in NP with respect to many-one polynomial-time reducibility (Karp [7]) then $P = NP$. Note that corollary 6.4 has a weaker conclusion than Fortune's result, but also a weaker hypothesis.

Let ZEROS denote the following decision problem: given a prime q and a set $\{p_1(x), p_2(x), \cdots, p_n(x)\}$ of sparse polynomials with integer coefficients, to determine whether there exists an integer x such that, for $i = 1,2,\cdots,n$, $p_i(x) \equiv 0 \bmod q$.

Corollary 6.5: If $ZEROS \in P/poly$, then $\bigcup_i \Sigma_i^p = \Sigma_2^p$.

This is based on a result of Plaisted [12].

Theorem 6.6: (Meyer) $EXPTIME \subseteq P/poly \iff EXPTIME = \Sigma_2^p$.

Corollary 6.7: $EXPTIME \subseteq P/poly \Rightarrow P \neq NP$.

Proof: Assume for contradiction that $EXPTIME \subseteq P/poly$ and $P = NP$. The first hypothesis implies that $EXPTIME = \Sigma_2^p$, and the second implies that $P = \Sigma_2^p$. Hence $P = EXPTIME$. But this contradicts the result that $P \subsetneq EXPTIME$, which is easily proved by diagonalization.

Let PER be the recognition problem associated with the function $Perm(\cdot)$ (cf. example 3 of this section).

Theorem 6.8: $PER \in P/poly \Rightarrow PER \in \Sigma_2^p$

Theorem 6.9: $PER \in P/\log \Rightarrow PER \in co\text{-}NP$.

Note, PER is in PSPACE but is not known to be in any Σ_i^p. It is known to be NP-hard ([18]).

FIGURE 1: MAIN RESULTS

$$PSPACE \subseteq P/poly \Rightarrow PSPACE = \Sigma_2^p \cap \Pi_2^p$$

$$PSPACE \subseteq P/log \iff PSPACE = P$$

$$EXPTIME \subseteq PSPACE/poly \iff EXPTIME = PSPACE$$

$$P \subseteq DSPACE((\log n)^{\ell})/log \iff P \subseteq DSPACE((\log n)^{\ell}) \quad (\ell > 0)$$

$$NSPACE(\log n) \subseteq DSPACE(\log n)/log \iff NSPACE(\log n) = DSPACE(\log n)$$

$$NP \subseteq P/log \iff P = NP \quad (1)$$

$$NP \subseteq P/poly \Rightarrow \cup \Sigma_i^p = \Sigma_2^p \quad (2)$$

$$EXPTIME \subseteq P/poly \Rightarrow EXPTIME = \Sigma_2^p \Rightarrow P \neq NP \quad (3)$$

(1) obtained jointly with Ravindran Kannan

(2) an improvement by Michael Sipser of an early result of ours

(3) due to Albert Meyer

References:

[1] L. Adleman, "Two Theorems on Random Polynomial Time," *Proc. 19th IEEE Symp. on Foundations of Computer Science,* pp. 75-83 (1978).

[2] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovasz C. Rackoff, "Random Walks, Universal Sequences, and the Complexity of Maze Problems," *Proc. 20th IEEE Symp. on Foundations of Computer Science,* (1979).

[3] A.K. Chandra and L.J. Stockmeyer, "Alternation," *Proc. 17th IEEE Symp. on Foundations of Computer Science,* (1976).

[4] S.A. Cook, "The Complexity of Theorem-Proving Procedures," *Proc. 3rd ACM Symp. on Theory of Computing,* pp. 151-158 (1971).

[5] S. Fortune, "A Note on Sparse Complete Sets," *SIAM J. Computing 8,* pp. 431-433 (1979).

[6] J. Hartmanis and L. Berman, "On Isomorphisms and Density of NP and Other Complete Sets," *Proc. 8th ACM Symp. on Theory of Computing,* pp. 30-40 (1976).

[7] R.M. Karp, "Reducibility Among Combinatorial Problems," in Complexity of Computer Computations (R.E. Miller and J.W. Thatcher, eds.), Plenum, New York (1972).

[8] D. Kozen, "On Parallelism in Turing Machines," *Proc. IEEE Symp. Foundations of Computer Science,* (1976).

[9] A.R. Meyer and M.S. Paterson, "With What Frequency are Apparently Intractable Problems Difficult", *M.I.T. Tech. Report,* Feb. 1979.

[10] A.R. Meyer and L.J. Stockmeyer, "The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space," *Proc. 13th IEEE Symp. on Switching and Automata Theory,* pp.]25-]29 (1972).

[11] N. Pippenger,"On Simultaneous Resource Bounds," *Proc. 20th IEEE Symp. on Foundations of Computer Science,* (1979)

[12] D.A. Plaisted, "New NP-hard and NP-complete Polynomial and Integer Divisibility Problems," *Proc. 18th IEEE Symp. FOCS,* pp. 241-253 (1977).

[13] S.E. Savage, "Computational Work and Time on Finite Machines," *JACM 19,* (4), pp. 660-674 (1972).

[14] T.S. Schaefer, "On the Complexity of Some Two-Person Perfect-Information Games," *JCSS 16,* pp.]85-225 (1978).

[15] C.P. Schnorr, "Optimal Algorithms for Self-Reducible Problems, *3rd Int. Coll. on Automata, Lang. and Programming,* Edinburgh (1976).

[16] L.J. Stockmeyer, "The Polynomial-Time Heirarchy," *Theoretical Computer Science 3,* p. 1-22 (1977).

[17] L.G. Valiant, "Relative Complexity of Checking and Evaluating, *U. of Leeds Tech. Report,* 1974.

[18] L.G. Valiant, "The Complexity of Computing the Permanent," *TCS,* (1979).