TURING MACHINES THAT TAKE ADVICE *

by Richard M. KARP [1]) and Richard J. LIPTON [2])

1. INTRODUCTION

Turing machines, random access machines and most of the other abstract computing devices studied in computational complexity theory represent *uniform algorithms,* which can receive arbitrarily long strings of symbols as input. The time and space needed by such devices to recognize a set $S \subseteq \{0, 1\}^*$ are examples of *uniform measures* of the complexity of S. In contrast, Boolean circuits, as well as certain types of decision trees and straight-line programs, compute functions with a finite domain. To study the complexity of recognizing the set $S \subseteq \{0, 1\}^*$ using such computational devices, we can view S as determining an infinite sequence of finite functions. For example, we can introduce, for each n, the Boolean function $S_n : \{0, 1\}^n \to \{0, 1\}$ defined as follows: $S_n(x_1, x_2, ..., x_n) = 1$ if and only if $x_1 x_2 ... x_n \in S$. If $L(S_n)$ denotes the minimum size of a Boolean circuit realizing S_n, then the growth rate of $L(S_n)$ as $n \to \infty$ is a measure of the nonuniform complexity of S.

Let us say that S has *small circuits* if $L(S_n)$ is bounded by a polynomial in n. It is well known that every set in P has small circuits [16]. Adleman [1] has recently proved the stronger result that every set accepted in polynomial time by a randomizing 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

* This article has already been published in *Logic and Algorithmic*, an international Symposium in honour of Ernst Specker, Zürich, February 1980. Monographie de L'Enseignement Mathématique N° 30, Genève 1982.

This research was supported in part by NSF grant [1]) MCS77-09906 and [2]) MCS79-20409. An earlier version of this paper was presented at the Twelfth Annual ACM Symposium on Theory of Computing, 1980 [9].

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 [19] 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.

Our results also serve to rule out some plausible speculations about the complexity of problems in NP. For example, one might imagine that $P \neq NP$, but SAT is tractable in the following sense: for every l there is a very short program that runs in time n^2 and correctly treats all instances of length l. Theorem 5.2 shows that, if "very short" means "of length c log 1", then this speculation is false.

Finally, we mention that the proof techniques presented here were put to use by S. Mahaney in his proof that $P \neq NP$ implies the nonexistence of sparse NP-complete problems [11], and by S. A. Cook in his proof that

$$P \subseteq HARDWARE\ (log\ n) \Rightarrow P \subseteq DSPACE\ (log\ n\ log\ log\ n)$$

[5].

2. Nonuniform Complexity Measures

In this section we will define our basic notion of nonuniform complexity and relate it to circuit complexity.

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

$$V/F = \{S : h \mid S \varepsilon V \text{ and } h \varepsilon F\}$$

Intuitively, V/F is the collection of subsets of $\{0, 1\}^*$ that can be accepted by V with an amount of advice bounded by F. The idea behind this definition is foreshadowed in papers by Pippenger [14] and Plaisted [15].

We are mainly interested in *poly*, the collection of all polynomially-bounded functions, and *log*, the collection of all functions that are $0\ (log\ n)$. Indeed, many of our results will concern the classes $P/poly$ and P/log.

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

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

(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 $g(n) < f(n) \leqslant 2^n$ (*i.o.*) then $P/f \subseteq P/g$.

The class *P/poly* can be characterized in terms of classic circuit complexity. An n-input m-gate *Boolean circuit* C is a function

$$C: \{n+1, ..., n+m\} \to \{0, 1\}^4 \times \{1, ..., n+m\}^2$$

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 inputs j and k to produce its output. If $1 \leqslant j \leqslant n$ then input j is simply the input variable x_j; otherwise, input j is the output of gate j. 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 Boolean 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

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

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 [14], 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*.

Another way we can gain insight into our classes V/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

$$UGAP \text{ is in } DSPACE(\log n)/poly.$$

Here *UGAP* is the undirected maze problem. As another example, we have Adleman's [1] result that R (the set of languages accepted in polynomial time by randomizing Turing machines) has small circuits, which can be restated as

$$R \text{ is a subset of } P/poly\,.$$

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 discuss a variety of complexity classes. These include the basic time and space classes $DTIME\,(T\,(n))$, $DSPACE\,(S\,(n))$ and $NSPACE\,(S\,(n))$ and the classes:

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

R = the set of languages accepted in polynomial time by randomizing Turing machines [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 polynomial-time hierarchy of Meyer and Stockmeyer [19]. For $i \geqslant 1$ we let $\sum_i^p$ (respectively $\prod_i^p$) denote those languages accepted in polynomial time by Turing machines that make i alternations starting from an existential (respectively universal) state. Note that $NP = \sum_1^p$ and co- $NP = \prod_1^p$. 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, 10].

Many of the following theorems take the form

$$L \subseteq S\,/\,F \Rightarrow L \subseteq S'$$

where L and S' are uniform complexity classes and V/F is a nonuniform complexity class. The proof usually consists of showing that

$$K \in V/F \Rightarrow K \in S'$$

where the set of strings K is complete in L with respect to an appropriate reducibility. The hypothesis tells us that K is of the form S : h where S is a language in V and a bound on $|h(|x|)|$ is known. The proof that $K \in S'$ consists of giving an appropriate uniform algorithm to recognize K. The function $h(|x|)$ is not available to this uniform algorithm, but the algorithm 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 the 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, 10])).

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

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

This specification of G suggests the following method of selecting an optimal move in any position $x \notin \mathrm{W}$: move to $F_0(x)$ if $F_0(x) \notin \mathrm{G}$; otherwise move to $F_1(x)$. If $x \in \mathrm{G}$, 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 $\mathrm{G} = \mathrm{S} : h$, where $\mathrm{S} \subseteq \{0, 1\}^*$ and h is a function from N into $\{0, 1\}^*$. Then

$$x \in \mathrm{G} \Leftrightarrow h(|x|) \cdot x \in \mathrm{S} .$$

The optimal move selection rule can be restated as follows:

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

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

in any position $x \notin \mathrm{W}$,
if $w \cdot x \notin \mathrm{S}$, then move to $F_0(x)$,
else 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, G, the associated set of strings, S a subset of $\{0, 1\}^*$ and h a function from N to $\{0, 1\}^*$, such that $\mathrm{G} = \mathrm{S} : h$. Let w and w' range over some set of strings $T(x)$ which includes $h(|x|)$. Then the following are equivalent:

(1) $x \in G$

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

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

Proof. If $x \in \mathrm{G}$ then the sentence $\forall\, w'\,(Win\,(h(|x|), w', x)$ is true. Hence (2) and (3) are true. If $x \notin \mathrm{G}$ then, for all w, $Win\,(w, h(|x|), x)$ is false; hence (2) and (3) are false. ■

Lemma 4.1 suggests how to decide if $x \in \mathrm{G}$ 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 $x \in \mathrm{G}$ if and only if some strategy emerges undefeated. A subtle point is that the round-robin tournament method determines whether $x \in \mathrm{G}$ 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 $x \notin \mathrm{W}$ 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 $\leqslant 3 \log_2 n$.

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 non-terminal 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 [17].

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

THEOREM 4.2. If $PSPACE \subseteq P/poly$ then $PSPACE = \sum_2^p \cap \prod_2^p$.

Proof. Since $\sum_2^p \cap \prod_2^p \subseteq PSPACE$, it suffices to prove

$$PSPACE \subseteq P/poly \Rightarrow PSPACE \subseteq \textstyle\sum_2^p \cap \prod_2^p .$$

For this it is sufficient to show

$$\mathrm{G} \in P/poly \Rightarrow \mathrm{G} \in \textstyle\sum_2^p \cap \prod_2^p$$

where G is the *PSPACE*-complete set described in Fact 2. Suppose $\mathrm{G} \in P/poly$. Then there is a set $\mathrm{S} \in P$, a positive constant k, and a function $h: N \to \{0, 1\}^*$ such that $|h(n)| \leqslant k + n^k$, so that $\mathrm{G} = \mathrm{S}: h$. By lemma 4.1,

$$x \in G \Leftrightarrow \exists w \forall w' \; Win(w, w', x) .$$

Here each of w and w' ranges over all strings of length $\leqslant k + |x|^k$. 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 $\mathrm{G} \in \sum_2^p$. Similarly, since

$$x \in \mathrm{G} \Leftrightarrow \forall w' \exists w \; Win(w, w', x) .$$

it follows that $\mathrm{G} \in \prod_2^p$. ∎

THEOREM 4.3. $PSPACE \subseteq P/log \Leftrightarrow PSPACE = P$.

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 $\mathrm{G} \in P/log \Rightarrow \mathrm{G} \in P$, where G is the *PSPACE*-complete set described in Fact 2. Again, the round-robin tournament method yields the proof. Suppose $\mathrm{G} \in P/log$. Then there is a set $\mathrm{S} \in P$, a positive constant k, and a function $h: N \to \{0, 1\}^*$ such that $h(n) \leqslant k \log_2 n$ so that $\mathrm{G} = \mathrm{S}: h$. Then $x \in \mathrm{G} \Leftrightarrow \exists w \forall w' \; Win(w, w', x)$, where w and w' range over the $0(|x|)^k$ strings of length $\leqslant k \log_2 |x|$. 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. *$EXPTIME \subseteq PSPACE/poly \Leftrightarrow EXPTIME = PSPACE$.*

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 $|h(|x|)| \leqslant k + |x|^k$, for some k. Then

$$x \in G \Leftrightarrow \exists w \forall w'\ Win\,(w, w', x)$$

where w and w' range over all strings of length $\leqslant k + |x|^k$. Since W, F_0 and F_1 are 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,

$$P \subseteq DSPACE\,((log\ n)^l)\,/\,log\ n \Leftrightarrow P \subseteq DSPACE\,((log\ n)^l)\,.$$

Proof. It suffices to prove

$$G \in DSPACE\,((log\ n)^l(/log) \Rightarrow G \in DSPACE\,((log\ n)^l)\,,$$

where G is the set described in Fact 3. Suppose

$$G \in DSPACE\,((log\ n))^l/log\,.$$

Then $G = S : h$ where $S \in DSPACE\left((log\ n)^l\right)$ and $|h(x)| \leqslant k \log_2 |x|$, for some k. Then $x \in G \Leftrightarrow \exists w\, \forall w'\ Win\,(w, w', x)$, where w and w' range over all strings of length $\leqslant k \log_2 |x|$. Clearly space $0\left((log\ n)^l\right)$ 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 $(\log n)^l$ recognizer for S. It follows that

$$G \in DSPACE\left((log\ n)^l\right). \qquad \blacksquare$$

5. The Self-Reducibility Method

The "hardest" problems in complexity classes defined by bounds on nondeterministic time or space often possess a structural property called *self-reducibility*. Various formal definitions of self-reducibility can be found in the literature ([12, 18, 20]). Here is one version of the idea. Let K be a subset of $\{0, 1\}^*$. A *self-reducibility structure* for K is specified by a partial ordering $<$ of $\{0, 1\}^*$ such that

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

(ii) $A \cap K$ is recursive

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

(iii) $G_0(x) < x$, $G_1(x) < x$, $|G_0(x)| = |G_1(x)| = |x|$ and $x \in K \Leftrightarrow G_0(x) \in K$ *or* $G_1(x) \in K$.

If K has a self-reducibility structure, then K is called *self-reducible*.

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 $F(t_1, t_2, \ldots, t_n)$ be a formula in which the variables $t_1, t_1, \ldots, t_n$ appear, and let $F(a, t_2, \ldots, t_n)$ be the same formula with the Boolean constant a substituted for t_1. Let $< F(t_1, t_2, \ldots, t_n) >$ and $< F(a, t_2, \ldots, t_n) >$ denote the encodings of these two formulas as strings. Then

$$|< F(t_1, t_2, \ldots, t_n) >| = |< F(a, t_2, \ldots, t_n) >|.$$

Let SAT denote this version of the satisfiability problem. The set SAT has a self-reducibility structure in which A is the set of propositional formulas containing no variables,

$$G_0(<F(t_1, t_2, \ldots, t_n)>) = <F(0, t_2, \ldots, t_n)> \text{ and}$$
$$G_1(<F(t_1, t_2, \ldots, t_n)>) = <F(1, t_2, \ldots, t_n)> .$$

As a second example, let DAG denote the set of encodings of triples (Ψ, 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 $\sigma_0(v)$ and $\sigma_1(v)$;

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

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

Assume that, for any directed acyclic graph G, any vertex t of out-degree 0, and any two vertices v and w, the encodings of (Ψ, v, t) and (Ψ, w, t) are of the same length. Then DAG is clearly self-reducible. Let A be the set of triples (Ψ, s, t) such that s is of out-degree 0, and let $G_0((\Psi, s, t)) = (\Psi, \sigma_0(s), t)$ and $G_1((\Psi, s, t)) = (\Psi, \sigma_1(s), t)$.

It is possible to relate the uniform complexity of a self-reducible set K to its nonuniform complexity. Suppose K has a self-reducibility structure $(<, \mathrm{A}, G_0, G_1)$ and $\mathrm{K} = \mathrm{S} : h$. For each $w \in \{0, 1\}^*$ define $reduct_w$, a total function over $\{0, 1\{^*$, by the following recursive definition:

$$\begin{aligned} reduct_w(x) = {} & \text{if } x \in \mathrm{A} \text{ then } x \text{ else} \\ & \text{if } w \cdot G_0(x) \in S \text{ then } reduct_w(G_0(x)) \text{ else} \\ & reduct_w(G_1(x)). \end{aligned}$$

Then, for all w, $reduct_w(x) \in \mathrm{A}$. Also, $reduct_w(x) \in \mathrm{K} \Rightarrow x \in \mathrm{K}$ and $x \in \mathrm{K} \Leftrightarrow reduct_{h(|x|)}(x) \in \mathrm{K}$. These observations imply the following lemma.

LEMMA 5.1. Let w range over some set which includes $h(|x|)$. Then

$$x \,\varepsilon\, K \Leftrightarrow \exists w\, [reduct_w(x) \,\varepsilon\, K] .$$

Lemma 5.1 suggests a uniform way of testing membership in K: for each w in a suitable set, compute $reduct_w(x)$ and test whether

$$reduct_w(x) \in \mathrm{A} \cap \mathrm{K} .$$

The complexity of this algorithm will depend on the time and space needed to test membership in A, and in $\mathrm{A} \cap \mathrm{K}$, on the lengths of chains in the partial ordering $<$, 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

$$\text{SAT} \in P/log \Rightarrow \text{SAT} \in P\ .$$

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

Using the self-reducibility structure for SAT given above, coupled with the method of lemma 5.1, we can test whether string x is in SAT. It is necessary to compute $reduct_w(x)$ for each of the polynomially-many strings w of length $\leqslant k \log_2 n$ and, for each, to test whether

$$reduct_w(x) \in \mathrm{A} \cap \mathrm{K}\ .$$

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 $<x, y, z>$ such that x has a factor between y and z. Then

$$Factor \in P\,/\,log \Leftrightarrow Factor \in P\ .$$

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

THEOREM 5.4.

$$NSPACE\,(log\,n)\,/\,log \subseteq DSPACE\,(log\,n)\,/\,log$$
$$\Leftrightarrow NSPACE\,(log\,n) = DSPACE\,(log\,n)\,.$$

Proof. It is sufficient to prove

$$NSPACE\,(log\,n)/log \subseteq DSPACE\,(log\,n)/log$$
$$\Rightarrow NSPACE\,(log\,n) = DSPACE\,(log\,n)\ .$$

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

$$\text{DAG} \in DSPACE\,(log\, n)/log \Rightarrow \text{DAG} \in DSPACE\,(log\, n)\,.$$

Suppose that DAG $=$ S : *h*, where S $\in$ *DSPACE* (*log n*) and

$$|h(n)| \leqslant k \log_2 n\,.$$

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 $\leqslant k \log_2 n$:

$v := s$;

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 0 (*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

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

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 $\Leftrightarrow F_0(x) \notin$ G or $F_1(x) \notin$ G.

Example 2. Let $(<, \text{A}, G_0, G_1)$ 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

$$\text{for} \quad x \notin \text{A},\ x \in \text{K} \Leftrightarrow G_0(x) \in \text{K} \cup G_1(x) \in \text{K}\,.$$

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

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

where

$$|f_1(x)| = |f_t(x)| = |x|\,.$$

For any string w, define $K_w = \{x \mid wx \in S\}$. Then, for $x \in A$, we can make the following assertion:

$$\begin{aligned} x \in K \Leftrightarrow \exists w[x \in K_w] \wedge \forall y\big[C_{K_w}(y) \\ = F(y, C_{K_w}(f_1(y)), \ldots, C_{K_w}(f_t(y))\big]\,. \end{aligned}$$

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 by searching through all choices of w and y. Further, the quantifier structure of the formula allows us to conclude that K lies in $\sum_2^p$, provided that $|h(n))|$ is bounded by a polynomial in n, S is in P, and F is computable in polynomial time.

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 proved this with $\sum_2^p$ replaced by $\sum_3^p$. The 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

$$(*)\quad Q_1 \vec{x}_1\, Q_2 \vec{x}_2 \ldots Q_i \vec{x}_i\, F(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_i)$$

where $Q_1 = \exists$, the Q_j are alternately $\exists$ and $\forall$, $\vec{x}_j$ is shorthand for the triple $x_{j_1}, x_{j_2}, \ldots, x_{j,r_j}$ 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

$\prod_i^p$. Also, it is clear that $A_i \in P/poly \Leftrightarrow E_i \in P/poly$. 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 $|h(n)| \leqslant k + n^k$ and $x \in A_{i=1} \Leftrightarrow h(|x|) \cdot x \in S$.

If y is the encoding of a sentence of the form (*), and $\vec{a}$ is a t_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 $F(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_i)$. 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 exist $S' \in P$, $k' \in N$ and $h': N \to \{0, 1\}^*$ so that $|h'(n)| \leqslant k' + n^{k'}$ and $x \in T \Leftrightarrow h'(|x|) \cdot x \in S$. Then $y \in A_i \Leftrightarrow$ for some $\vec{a}$, $y_{\vec{a}} \in E_{i-1} \Leftrightarrow$ for some a,

$$h(|y_{\vec{a}}) \cdot y_{\vec{a}} \in S \Leftrightarrow h(|y_{\vec{a}}|) \cdot y \in T \Leftrightarrow h'\big(|h(|y_{\vec{a}}|) \cdot y|(\cdot h(|y_{\vec{a}}|) \cdot y \in S'.$$

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 $NP \subseteq P/poly \Rightarrow \prod_3^p \subseteq \sum_2^p$; for this it is sufficient to prove that the set A_3 is in $\sum_2^p$. Our proof is based on the fact that A_3 has an easty-to-evaluate recursive definition of the form $C_{A_3}(y) = R(y, C_{A_3}(y'), C_{A_3}(y''))$. Consider a sentence y of the form

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

where the string of quantifiers $Q_1 Q_2 \ldots Q_n$ is contained in $\forall^* \exists^* \forall^*$.

Let

$$y' = Q_2 x_2 \ldots Q_n x_n F(0, x_2, \ldots, x_n)$$

and

$$y'' = Q_2 x_2 \ldots Q_n x_n F(1, x_2, \ldots, x_n).$$

Then

$$C_{A3}(y) = (\text{if } Q_1 = \forall \text{ then } C_{A3}(y') \wedge C_{A3}(y'') \text{ else } C_{A3}(y') \cup C_{A3}(y'')).$$

C_{A_3} is uniquely determined by this recursive definition which is of the form $C_{A_3}(y) = R((y, C_{A_3}(y'), C_{A_3}(y''))$, 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 $|h(n)| \leqslant k + n^k$. For each $w \in \{0,1\}^*$ define $f_w : \{0,1\}^* \to \{0,1\}$ by $f_w(x) = 1 \Leftrightarrow wx \in S$. Then membership of y in A_3, in the case where y contains at least one quantifier, is expressed by the following formula:

$$(**) \qquad \exists w \forall z \left[f_w(y) = 1 \wedge f_w(z) = R\left(z, f_w(z'), f_w(z'')\right)\right] .$$

Here w ranges over all strings of length $\leqslant k + |y|^k$, 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

$$f_w(z) = R\left(z, f_w(z'), f_w(z'')\right)$$

can be tested in polynomial time. Thus the $\exists \forall$ form of (**) establishes that $A_3 \in \sum_2^p$. ■

Theorem 6.1 has a number of corollaries.

COROLLARY 6.3. If $R = NP$ then $\bigcup_i \sum_i^p = \sum_2^p$.

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* [6, 7] if

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

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

$$\bigcup_i \sum_i^p = \sum_2^p .$$

This corollary follows immediately from Theorem 6.1 once it is noted that the existence of such an S implies that every set in *NP* has small circuits. Corollary 6.4 should be compared with results of Mahaney [11] and Fortune [6] which show that, if there exists a sparse or co-sparse set which is complete in *NP* with respect to many-one polynomial-time reducibility (Karp [8]) then $P = NP$. Note that Corollary 6.4 has a weaker conclusion than the results of Mahaney and Fortune, but also a weaker hypothesis.

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

COROLLARY 6.5. If $ZEROS \in P/poly$, then $\cup \sum_i^p = \sum_2^p$.

This is based on Plaisted's result [15] that every problem in NP can be solved in polynomial time with the help of an oracle for $ZEROS$ together with a polynomial-bounded number of advice bits. Thus $NP \subseteq P/poly$ if $ZEROS \in P/poly$.

THEOREM 6.6. (Meyer) $EXPTIME \subseteq P/poly \Leftrightarrow EXPTIME = \sum_2^p$.

Proof. Let G be the set of strings representing positions from which the first player can win in the *EXPTIME*-complete game mentioned in FACT 1. It suffices to prove that

$$G \in P\,/\,poly \Rightarrow G \in \textstyle\sum_2^p .$$

Suppose G $=$ S $: h$ where S $\in P$ and h is polynomial-bounded. Then

$$x \in \mathrm{G} \Leftrightarrow \exists w \forall z \left[x \in W \cup z \in W \cup \left(wz \in \mathrm{S} \Leftrightarrow wF_0(z) \notin \mathrm{S} \cup wF_1(z) \notin \mathrm{S}\right)\right]$$

Here w ranges over all strings of length $\mid h(|x|)$ and z ranges over all strings of the same length as x. Since membership in S or membership in W can be tested in polynomial time, it tollows that G $\in \sum_2^p$. ■

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 $EXPTIME = \sum_2^p$, and the second implies that $P = \sum_2^p$. Hence $P = EXPTIME$. But this contradicts the result that $P \subsetneqq EXPTIME$, which is easily proved by diagonalization. ■

Figure 1. MAIN RESULTS

$PSPACE \subseteq P\,/\,poly \Rightarrow PSPACE = \sum_2^p \cap \sum_2^p$

$PSPACE \subseteq P\,/\,log \Leftrightarrow PSPACE = P$

$EXPTIME \subseteq PSPACE\,/\,poly \Leftrightarrow EXPTIME = PSPACE$

$P \subseteq DSPACE\,((log\, n)^l)\,/\,log \Leftrightarrow P \subseteq DSPACE\,((log\, n)^l)$

$NSPACE\,(log\, n) \subseteq DSPACE\,(log\, n)\,/\,log$

$\Leftrightarrow NSPACE\,(log\, n) = DSPACE\,(log\, n)$

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

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

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

REFERENCES

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

[2] Aleliunas, R., R. M. Karp, R. J. Lipton, L. Lovász and C. Rackoff. Random Walks, Universal Sequences, and the Complexity of Maze Problems. *Proc. 20th IEEE Symp. on Foundations of Computer Science*, pp. 218-223 (1979).

[3] Chandra, A. K. and L. J. Stockmeyer. Alternation. *Proc. 17th IEEE Symp. on Foundations of Computer Science*, pp. 98-108 (1976).

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

[5] —— Towards a Complexity Theory of Synchronous Parallel Computation. *Technical Report 141/80*, Computer Science Department, University of Toronto (1980).

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

[7] Hartmanis, J. 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).

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

[9] Karp, R. M. and R. J. Lipton. Some Connections Between Nonuniform and Uniform Complexity Classes. *Proc. 12th Annual ACM Symposium on Theory of Computing*, pp. 302-309 (1980).

[10] Kozen, D. On Parallelism in Turing Machines. *Proc. IEEE Symp. on Foundations of Computer Science*, pp. 89-97 (1976).

[11] Mahaney, S. R. Sparse Complete Sets for NP: Solution of a Conjecture of Berman and Hartmanis. *Proc. 21st IEEE Symp. on Foundations of Computer Science*, pp. 54-60 (1980).

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

[13] Meyer, A. R. 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. 125-129 (1972).

[14] Pippenger, N. On Simultaneous Resource Bounds. *Proc. 20th IEEE Symp. on Foundations of Computer Science*, pp. 307-311 (1979).

[15] Plaisted, D. A. New NP-hard and NP-complete Polynomial and Integer Divisibility Problems. *Proc. 18th IEEE Symp. on Foundations of Computer Science*, pp. 241-253 (1977).

---

(1) Obtained jointly with Ravindran Kannan.

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

(3) Due to Albert Meyer.

[16] SAVAGE, J. E. Computational Work and Time on Finite Machines. *JACM 19, (4)*, pp. 660-674 (1972).
[17] SCHAEFER, T. S. On the Complexity of Some Two-Person Perfect-Information Games. *JCSS 16*, pp. 185-225 (1978).
[18] SCHNORR, C. P. Optimal Algorithms for Self-Reducible Problems. *3rd Int. Coll. on Automata, Language and Programming*, Edinburgh (1976).
[19] STOCKMEYER, L. J. The Polynomial-Time Hierarchy. *Theoretical Computer Science 33*, pp. 1-22 (1977).
[20] VALIANT, L. G. Relative Complexity of Checking and Evaluating. *Univ. of Leeds Tech. Report*, (1974).

(Reçu le 16 mai 1981)

Richard M. Karp

Computer Science Division
University of California
Berkeley, CA 93720
USA

Richard J. Lipton

Department of Electrical
Engineering and Computer Science
Princeton University
Princeton, N.J. 08540
USA

vide-leer-empty