JOURNAL OF COMPUTER AND SYSTEM SCIENCES 13, 100–118 (1976)

Complexity-Class-Encoding Sets

NANCY LYNCH

University of Southern California, Los Angeles, California 90007

Received August 20, 1974; revised March 19, 1976

Properties of sets which are complex because they encode complexity classes are explored. It is shown that not all sets with inherent complexity are of this type, although this is the only type of set for which well-developed techniques exist for *proving* inherent complexity.

Possibilities for the complexity of encoding sets are discussed, first with reference to an "almost everywhere" vs. "infinitely many arguments" classification, and later with reference to the density of the set of arguments on which the problem is complex.

It is shown that relative complexity relationships among sets of this type are highly structured, in contrast to the wide variation possible among arbitrary recursive sets.

1. INTRODUCTION

In proving decision procedures for logical theories, as well as other types of problems, to be inherently complex [3; 13; and others] the general method used has been to show that the problems are sufficiently expressive to encode a complexity class of sets (i.e., all problems which are computable within a certain time or space bound). Because that complexity class must contain diagonalizing sets which are known to be complex, the decision problem itself is shown to be complex.

It is not difficult to show that there are complex problems which do not owe their complexity to the fact that they encode complexity classes; thus, most work on proving the complexity of problems has focused on a proper subclass of all complex problems. We call members of this subclass, informally, "complexity-class-encoding sets" or simply "encoding sets." In this paper, interesting properties of this type of set are explored, with particular attention paid to differences between encoding sets and arbitrary complex sets.

In Section 2, we define our notions of encoding. A definition of encoding for a complexity class of problems must arise from a definition of efficient reducibility between problems; we restrict ourselves to encoding by polynomial-time-bounded reducibilities as studied in [9].

In Section 3, elementary results are presented. To summarize the basic important property of encoding sets, we state lower and upper bounds on the complexity which

may be inferred from the fact that a set is an encoding set. We then show that some provably complex sets cannot be proved to be complex by an encoding argument, indicating the need for new proof techniques for inherent complexity.

In Section 4, we examine complexity possibilities for encoding sets, as far as an "almost everywhere" vs. "infinitely often" classification. Results in Section 3 show that encoding sets have a certain minimal complexity, at least on an infinite set of arguments. Also, encoding sets need not exceed that minimal complexity by very much. Here we show that encoding sets may be very simple on infinitely many arguments, or arbitrarily complex on infinitely many arguments. In addition, for some definitions of encoding, encoding sets may be arbitrarily complex a.e., but for others this is not the case.

In Section 5, we again examine complexity possibilities for encoding sets, this time by a finer classification that before; we see what possibilities may exist for the *density* of the sets of arguments on which encoding sets are complex. Unlike before, there is not a wide variation in results among different definitions of encoding. In fact, for all our definitions, sets which encode nontrivial complexity classes have a recursive bound on their sparsity.

In Section 6, relative complexity relationships among encoding sets are discussed. In [10] several results are presented showing the existence of arbitrarily complex sets having an independence property: even with the aid of one of the sets as an oracle, the other is still approximately as difficult to compute as without the oracle. We wonder whether this independence property is possessed by any pairs of "natural" problems. So far, all examples of natural problems known to have inherent complexity i.o. have been shown to be encoding sets. For encoding sets, as we show in this section, the independence property cannot hold, since helping relationships among encoding sets are very strictly limited. Thus, the existence of pairs of natural, independent sets apparently cannot be shown until further techniques for proving lower bounds are developed.

Finally, in Section 7, we make explicit the distinctness of the encoding relations arising from different efficient reducibilities.

Additional details for some proofs, related results and examples may be found in a longer version of the current paper [7].

2. Notation and Definitions

All sets involved in encodings in this paper will be sets of finite strings over $\Sigma = \{0, 1\}$. If $x \in \Sigma^*$, $| x |$ will represent the length of string x. At times, we will wish to interpret a string as an integer. For a string $x \in \Sigma^*$, we will write $\hat{x}$ for the integer whose binary representation is the concatenation $1x$. This association induces a natural ordering on strings.

For a set $A \subseteq \Sigma^*$, we will write $| A |$ for the cardinality of set A, and C_A for the characteristic function.

λ is the empty string. (λ will also be used as in Church's lambda-notation.)

We write $\exists^\infty x$ or i.o. (x) to denote "for infinitely many x," and $\forall^\infty x$ or a.e. (x) to denote "for all except possibly finitely many x." When no confusion is likely, we write simply i.o. or a.e.

If A is a set, $t: \Sigma^* \to N$, we write Comp $A \leqslant t$ if there is a multitape Turing machine with separate input and output tapes, computing C_A, which uses not more than $t(x)$ steps on any input string x. Similarly, we write Comp $A \leqslant t$ a.e. if such a Turing machine exists, using not more than $t(x)$ steps a.e. (x). And finally, we write Comp $A \leqslant t$ i.o. if such a machine exists, using not more than $t(x)$ steps i.o. (x). Comp $A > t$ a.e. will denote $\neg$Comp $A \leqslant t$ i.o., and Comp $A > t$ i.e. will denote $\neg$Comp $A \leqslant t$ a.e.

Analogously, Comp$^{(B)}$ $A \leqslant t$ for t, A as above, and set B means there is a multitape oracle Turing machine C_A (when used with a B-oracle), using not more that $t(x)$ steps on any x. Again, the other definitions are extended analogously.

Definitions of encoding arise from definitions of efficient reducibility. In this paper we use two definitions of encoding arising from Cook's [2] and Karp's [6] reducibilities.

DEFINITION. $A \leqslant_T^{\mathscr{P}} B$ (A is polynomial-time Turing reducible to B) if there exists an oracle Turing machine M and a polynomial p such that:

$x \in A \Leftrightarrow M$ with input x and oracle B accepts within $p(| x |)$ steps. $A \leqslant_m^{\mathscr{P}} B$ (A is polynomial-time many-one reducible to B) if there exists a polynomial-time computable function $f: \Sigma^* \to \Sigma^*$ such that:

$$x \in A \Leftrightarrow f(x) \in B.$$

We will say $A \leqslant_m^{\mathscr{P}} B$ via f in this case.

These two reducibilities are time-bounded analogs to Turing and many-one reducibilities in recursive function theory [14], and are explored and compared in [9].

DEFINITION. *A T-encodes t* for a recursive set A and recursive function $t: \Sigma^* \to N$, provided:

$$(\forall B)[\text{Comp } B \leqslant t \Rightarrow B \leqslant_T^{\mathscr{P}} A].$$

A m-encodes t for A and t as above, if:

$$(\forall B)[\text{Comp } B \leqslant t \Rightarrow B \leqslant_m^{\mathscr{P}} A].$$

Note. (a) For any A, t, A m-encodes $t \Rightarrow A$ T-encodes t. (b) If for some polynomial p, and for all x, $t(x) \leqslant p(| x |)$, then:

$$(\forall A)[A \text{ } T\text{-encodes } t],$$

and

$$(\forall A \neq \varnothing, \Sigma^*)[A \text{ } m\text{-encodes } t].$$

The following definition will be useful.

DEFINITION. For any recursive $t: \Sigma^* \to N$, t is *honest* if there is a multitape Turing machine M with separate input and output tapes and a polynomial p such that M computes t, giving the answer in binary, and on every input string x, M requires not more than $p(t(x))$ steps.

In particular, if t is a total function, and for some Turing machine (of any type), $t(x)$ is the number of steps used on input x, then t is honest.

We will also require pairing and projection functions, both for binary strings and for integers: for 2 binary strings x and y. We let $\langle x, y\rangle$ denote the string $x'11y'$ where x' is the string formed by inserting a 0 after each bit of x, and y' is similarly formed from y. Also,

$$\pi_1(z) = \begin{cases} x & \text{if } z = \langle x, y\rangle, \\ \text{undefined otherwise,} \end{cases}$$

and

$$\pi_2(z) = \begin{cases} y & \text{if } z = \langle x, y\rangle, \\ \text{undefined otherwise,} \end{cases}$$

define partial recursive projection functions: $\Sigma^* \to \Sigma^*$. We also use the same notation $\langle\ \rangle$, π_1 and π_2 for functions on positive integers. When we do so, they will represent the usual integer pairing and projection functions in [14], modified to exclude 0.

3. BASIC RESULTS

In this section, we note some basic facts about encoding sets.

The key fact about encoding sets used in the literature is that they are necessarily complex, at least on an infinite set of arguments, with a lower bound on their complexity related to the function whose class they encode. More specifically,

PROPOSITION 1. *There exists a recursive operator $\mathcal{F}$ such that:*

$$(\forall t: \Sigma^* \Rightarrow N, \textit{honest})(\forall A, \textit{recursive})$$
$$[A \textit{ T-encodes } \mathcal{F} \circ t \Rightarrow \operatorname{Comp} A > t \textit{ i.o.}].$$

Proof. By direct simulation of the A-oracle and diagonalization techniques of Hartmanis and Stearns [5]. ∎

Of course, versions used in applications are very sharp. This is possible because more restrictive coding definitions are generally used, and because only well-behaved functions t (such as nested exponentials) are generally used.

On the other hand, an upper limit on the inferred complexity of encoding sets follows from the following result, which gives the existence of m-encoding sets of fairly small complexity relative to an honest name for the complexity class encoded:

THEOREM 2. *If* $t: \Sigma^* \to N$ *is honest,* $t(x) \geqslant |x|$ *for all* x, t *is monotone nondecreasing in the length of its input, then there exists a recursive set* A *and a polynomial* p, *such that*:

$$A \text{ } m\text{-encodes } t \text{ and } \mathrm{Comp}\, A \leqslant \lambda x[p(t(x))].$$

Proof. We let $\{M_i\}$ be an enumeration of all multitape Turing machine transducers, $\{\phi_i\}$ the corresponding enumeration of partial functions computed, and $\{T_i\}$ the corresponding enumeration of step-counting functions. We assume that these enumerations have the property that simulation is efficient: we assume that it is possible, when given the binary representation of any integer i, to generate a binary description of M_i in a natural form, in at most $p(\log_2 i)$ steps, for some polynomial p. We also assume, for example, the existence of a Turing machine which takes as inputs the binary representation of an integer i, and a string x, and outputs $\phi_i(x)$; we assume that this can be done in time $p(\log_2 i, T_i(x))$ for some polynomial p of three variables.

Now define A by:

$$x \in A \text{ iff } x = \langle i, y \rangle \quad \text{and} \quad T_i(y) \leqslant t(y) \quad \text{and} \quad \phi_i(y) = 1.$$

It is clear that A m-encodes t.

The efficiency of the simulation, together with the monotonicity, honesty, and lower bound for t, gives:

$$\mathrm{Comp}\, A \leqslant \lambda x[p(t(x))],$$

for some polynomial p, as required. ∎

A result that shows that some provably complex sets cannot be shown to be complex by an encoding argument is the following.

THEOREM 3. *If* $t: \Sigma^* \to N$ *is honest, and for all polynomials* p, $t(x) \geqslant p(|x|)$ *a.e., then*:

$$(\exists p, \textit{ a polynomial})(\forall s: \Sigma^* \to N, \textit{ recursive})(\exists A, \textit{ recursive})$$
$$[\mathrm{Comp}\, A > s \textit{ a.e. and } \neg A \textit{ T-encodes } \lambda x[p(t(x))]].$$

Proof. We use two lemmas:

LEMMA 4. *If* $t: \Sigma^* \to N$ *is recursive, and for all polynomials* p, $t(x) > p(|\, x\,|)$ *a.e., then there exists* $t': \Sigma^* \to N$, *recursive, such that*:

(i) *for all polynomials* p, $t'(x) > p(|\, x\,|)$ *a.e., and*

(ii) *for all polynomials* p, $t(x) > p(t'(x))$ *a.e.*

Proof of Lemma 4. Assume t is given. Define t' by

$$t'(x) = \begin{cases} 1 & \text{if } |\, x\,| \geqslant t(x), \\ |\, x\,|^{k_x}, & \text{where } k_x \text{ is the largest integer such that} \\ & |\, x\,|^{(k_x)^2} < t(x), \quad \text{otherwise.} \end{cases}$$

The hypothesis on t shows that for any integer k, $k_x > k$ a.e. (x). But then if $k_x > k$ and $|\, x\,| \geqslant 2$, we have

$$t'(x) = |\, x\,|^{k_x} > |\, x\,|^k.$$

This shows (i). Also, if $k_x > k$ and $|\, x\,| \geqslant 2$, we have

$$(t'(x))^k = (|\, x\,|^{k_x})^k < (|\, x\,|^{k_x})^{k_x} = |\, x\,|^{(k_x)^2} < t(x).$$

This shows (ii). ∎

The next lemma says that there exist some sets B which are not very complex, and some sets A which *are* very complex, such that B is not Turing reducible to A is polynomial time:

LEMMA 5. *If* $t: \Sigma^* \to N$ *is honest, and for all polynomials* p, $t(x) > p(|\, x\,|)$ *a.e., then*:

$$(\exists B)(\exists p, \textit{ a polynomial})(\forall s: \Sigma^* \to N, \textit{ recursive})(\exists A, \textit{ recursive})$$

$$[\text{Comp } B \leqslant \lambda x[p(t(x))] \textit{ and } \text{Comp } A > s \textit{ a.e. and } B \not\leqslant_T^{\mathscr{P}} A].$$

Proof of Lemma 5. Given t, we apply Lemma 4 to obtain t'. Since t is honest, we may apply Hartmanis and Stearns diagonalization techniques to obtain a recursive set B with:

$$\text{Comp } B \leqslant \lambda x[p_1(t(x))] \text{ for some polynomial } p_1, \text{ and: } (\forall p, \text{ a polynomial})$$

$$[\text{Comp } B > \lambda x[p(t'(x))] \text{ i.o.}].$$

We than apply the construction in Theorem 5.3 of [10] to set B and lower bound $\lambda x[p_2(t'(x))]$ for a sufficiently large polynomial p_2. We obtain a set A, with Comp $A > s$ a.e., for which any oracle Turing machine with one worktape, using oracle A and computing C_B, requires at least $t'(x)$ steps i.o. (x). Thus, by polynomial invariance of different Turing machine models, $B \not\leqslant_T^{\mathscr{P}} A$. ∎

Proof of Theorem 3 (continued). Immediate from Lemma 5.

Thus, we see that some sets have inherent a.e. complexity, but with none of their complexity due to encoding of a complexity class. Whether there are *natural* problems (i.e., problems not defined by diagonalization constructions) with this property remains open. If so, additional proof techniques for i.o. complexity will be needed. This need will be discussed again later in the paper when we examine helping relations among encoding sets.

4. Complexity Possibilities for Encoding Sets

Proposition 1 allows us to conclude that encoding sets have a certain minimum complexity, at least on an infinite set of arguments. Also, by Theorem 2, encoding sets need not exceed that minimum complexity by very much. In this section, we analyze the remaining possibities for the complexity of encoding sets, as far as an "a.e." and "i.o." classification. First, it is easy to show that the complexity of encoding sets may be very small or very large on an infinite set of arguments.

Theorem 6. (a) *There exists a polynomial p such that for any t: $\Sigma^* \to N$, recursive*:

$$(\exists A, \textit{recursive})[A \textit{ m-encodes } t \textit{ and } \text{Comp } A \leqslant \lambda x[p(|x|)] \textit{ i.o.}].$$

(b) *For any s, t: $\Sigma^* \to N$, recursive*:

$$(\exists A, \textit{recursive})[A \textit{ m-encodes } t \textit{ and } \text{Comp } A > s \textit{ i.o.}].$$

Proof. Not all arguments are needed to effect the encoding. For (b), diagonalization techniques as in [4], for example, may be used to build the desired i.o. complexity into A. ∎

All results so far have been the same for $\leqslant_m^{\mathscr{P}}$ and $\leqslant_T^{\mathscr{P}}$; thus, we have stated the stronger result in every case. Theorem 7 and Corollary 12 will now distinguish between the two reducibilities by showing that T-encoding sets may be arbitrarily complex a.e., but the same is not true for m-encoding sets.

Theorem 7. *For any s, t: $\Sigma^* \to N$ recursive,*

$$(\exists A, \textit{recursive})[A \textit{ T-encodes } t \textit{ and } \text{Comp } A > s \textit{ a.e.}].$$

Proof. In [10], the following is shown.

Lemma 8. *For any recursive set B, any s: $\Sigma^* \to N$, recursive,*

$$(\exists A, \textit{recursive})[B \leqslant_T^{\mathscr{P}} A \textit{ and } \text{Comp } A > s \textit{ a.e.}].$$

The theorem then follows from Lemma 8, Theorem 2 and transitivity of $\leqslant_T^{\mathscr{P}}$. ∎

The situation is somewhat different for $\leqslant_m^{\mathscr{P}}$. First, it is obvious that polynomial-computable sets are $\leqslant_m^{\mathscr{P}}$-reducible to all nontrivial sets. The next theorem is a partial converse to this fact, stating that any set $\leqslant_m^{\mathscr{P}}$-reducible to arbitrarily complex sets is polynomial-computable, at least i.o.

(We note that, in the lengthy interval between the initial writing of this paper and its appearance, a stronger version of this theorem has been proved [8]. However, since the proof of the stronger version requires a lemma from the present proof, we retain the proof of this weaker version.)

THEOREM 9. *Assume B is such that for all s: $\Sigma^* \to N$, recursive*:

$$(\exists A, \textit{recursive})[B \leqslant_m^{\mathscr{P}} A \textit{ and } \mathrm{Comp}\, A > s \textit{ a.e.}].$$

Then: $(\exists p, \textit{a polynomial})[\mathrm{Comp}\, B \leqslant \lambda x[p(|\, x\,|)] \textit{ i.o.}]$.

Proof. We use 2 lemmas:

LEMMA 10. *Assume B is such that for all s: $\Sigma^* \to N$, recursive*:

$$(\exists A, \textit{recursive})[B \leqslant_m^{\mathscr{P}} A \textit{ and } \mathrm{Comp}\, A > s \textit{ a.e.}].$$

Then for any r: $\Sigma^* \to N$, recursive,

$$(\exists A, \textit{recursive})(\exists f\colon \Sigma^* \to \Sigma^*, \textit{recursive})[B \leqslant_m^{\mathscr{P}} A \textit{ via } f \textit{ and } |\, x\,| > r(f(x)) \textit{ a.e.}].$$

Proof of Lemma 10. Assume the hypothesis is true for B and the conclusion false. Let r: $\Sigma^* \to N$, recursive, be such that

$$(\forall A, \text{recursive})(\forall f\colon \Sigma^* \to \Sigma^*, \text{recursive})$$
$$[B \leqslant_m^{\mathscr{P}} A \text{ via } f \Rightarrow (\exists^\infty x)[|\, x\,| \leqslant r(f(x))]].$$

We assume without loss of generality that $r(x) \geqslant |\, x\,|$ for all x. But this easily implies

$$(\forall A, \text{recursive})(\forall f\colon \Sigma^* \to \Sigma^*, \text{recursive})$$
$$[B \leqslant_m^{\mathscr{P}} A \text{ via } f \Rightarrow (\exists^\infty y)(\exists x)[y = f(x) \text{ and } |\, x\,| \leqslant r(y)]].$$

Consider the following Algorithm $\mathscr{A}$. With r as above, and A, f such that $B \leqslant_m^{\mathscr{P}} A$ via f, Algorithm $\mathscr{A}$ may be used to compute C_A.

ALGORITHM $\mathscr{A}$. Given input y, compute in order $f(\lambda), f(0), f(1), f(00), \ldots$. If for any x with $|\, x\,| \leqslant r(y)$, we discover that $f(x) = y$, we compute and output $C_B(x)$. Otherwise, we use some alternative procedure to compute $C_A(y)$.

END

But then for any such A, f, we have the following: an a.e. upper bound of $\lambda x[2^{|x|}]$ on f's complexity, together with the function r and any fixed upper bound on B's complexity, combine to provide an upper bound on A's complexity i.o. (namely, on sufficiently long members of the infinite set of strings y such that

$$(\exists x)[y = f(x) \text{ and } | x | \leqslant r(y)]).$$

This upper bound depends on the bounds mentioned above, but is independent of A and f, contradicting the hypothesized existence of arbitrary complex A such that $B \leqslant_m^{\mathscr{P}} A$. ∎

LEMMA 11. *Assume B is such that for all $s: \Sigma^* \to N$, recursive:*

$$(\exists A, \textit{recursive})[B \leqslant_m^{\mathscr{P}} A \textit{ and } \text{Comp } A > s \textit{ a.e.}].$$

Then for any $r: \Sigma^* \to N$, recursive,

$$(\exists A, \textit{recursive})(\exists f: \Sigma^* \to \Sigma^*, \textit{recursive})$$

$$[B \leqslant_m^{\mathscr{P}} A \textit{ via } f \textit{ and } (\exists^\infty x)(\exists y)[r(y) \leqslant | x | \textit{ and } f(x) = f(y)]].$$

Proof of Lemma 11. We may assume without loss of generality that r is monotone increasing in the length of its argument. We assume B satisfies the hypothesis, and assume that r is given. In terms of r, we define a recursive function $r_1 : N \to N$ as follows:

$$r_1(1) = 1$$

$$r_1(n+1) = r(0^{r_1(n)}) \qquad \text{for all } n > 1.$$

We then transform r_1 into a recursive function $r_2 : \Sigma^* \to N$ by:

$$r_2(y) = r_1(2\hat{y}) \qquad \text{for all strings } y.$$

We then apply Lemma 10 to B and r_2, and obtain A, f with A recursive, $B \leqslant_m^{\mathscr{P}} A$ via f, and $| x | > r_2(f(x))$ a.e. (x). It follows that:

$$(\forall^\infty \langle x, y\rangle)[| x | \leqslant r_2(y) \Rightarrow \widehat{f(x)} < \hat{y}]. \tag{1}$$

(This is because if $| x | > r_2(f(x))$, then if $\widehat{f(x)} \geqslant \hat{y}$, we would have $r_2(f(x)) \geqslant r_2(y)$, so $| x | > r_2(y)$. Also, if x is one of the finite set of exceptions to $[| x | > r_2(f(x))]$, there can be only finitely many y such that $\langle x, y\rangle$ provides an exception to $[| x | \leqslant r_2(y) \Rightarrow \widehat{f(x)} < \hat{y}]$.)

Now by the definition of r_2,

$$(\forall w \in \Sigma^*)(\exists C_w \subseteq \Sigma^*, | C_w | = 2^{\hat{w}}) \tag{2}$$

$$[(\forall x \in C_w)[| x | \leqslant r_2(w)] \text{ and } (\forall x, y \in C_w)[\hat{y} < \hat{x} \Rightarrow r(y) \leqslant | x |]].$$

(Namely, $C_w = \{0, 0^{r(0)}, 0^{r(0^{r(0)})}, \ldots\}$ will suffice, where the dots indicate we continue until we have exactly $2\hat{w}$ elements. The monotonicity of r assures that the elements of C_w are distinct.)

But then by (1), there is a constant k independent of w such that:

$$(\forall w)[|\{x \in C_w \mid \widehat{f(w)} < \hat{w}\}| \geqslant 2\hat{w} - k].$$

Thus, by the Pigeonhole Principle,

$$(\forall w)[|\{\langle x, y\rangle \mid x, y \in C_w, \hat{y} < \hat{x} \text{ and } f(x) = f(y)\}| \geqslant \hat{w} - k + 1].$$

But then (2) yields the conclusion of the lemma. ∎

Proof of Theorem 9 (continued). We can now prove the theorem. Given B as in the hypothesis, we choose a recursive function r: $\Sigma^* \to N$ which satisfies:

$$\text{Comp } B \leqslant r \text{ and } r(x) \geqslant 2^{|x|}, \text{ for all } x,$$

and r is increasing in its argument.

We then apply Lemma 11 to B and r, to obtain A and f such that:

$$(\exists^{\infty} x)(\exists y)[r(y) \leqslant |x| \text{ and } f(x) = f(y)]. \tag{3}$$

We use B and this function f to define the following Algorithm $\mathscr{B}$ for computing C_B.

ALGORITHM $\mathscr{B}$. Given input x, compute $f(x)$. Then compute, in order, $f(\lambda)$, $f(0), f(1), f(00), \ldots$. If for any y with $\hat{y} < \hat{x}$, we discover that $f(y) = f(x)$, we compute and output $C_B(y)$. Otherwise, we compute $C_B(x)$ by some alternative method.

END

By (3), there is an infinite set of arguments x for which the first alternative in Algorithm $\mathscr{B}$ will hold, and for which the number of steps required by Algorithm $\mathscr{B}$ may be bounded above by

$$p_1(|x|, r(y)), \text{ for some polynomial } p_1,$$

where y represents the first argument found by the search in Algorithm $\mathscr{B}$. The lower bound on r is used here, as well as the condition [Comp $B \leqslant r$]. But by (3) and the monotonicity of r, the number of steps on argument x is bounded by $p(|x|)$ for some polynomial p. Thus,

$$\text{Comp } B \leqslant \lambda x[p(|x|)] \text{ i.o., as needed.} \quad ∎$$

As a corollary to Theorem 9, we obtain a result for $\leqslant_m^{\mathscr{P}}$ which corresponds to Theorem 7. That is, we interpret Theorem 9 in terms of encodings:

COROLLARY 12. *There exist* $s, t: \Sigma^* \to N$, *recursive, such that* $(\forall A$, *recursive*$)$ $[A$ *m-encodes* $t \Rightarrow$ Comp $A \leqslant s$ *i.o.*$]$.

Proof. By an appropriate version of the Compression Theorem of Blum [1] and Theorem 9. ∎

An interesting interpretation for Corollary 12 is that for sets which are sufficiently complex a.e., conventional m-encoding techniques are useless even for inferring moderate i.o. complexity.

5. DENSITY PROPERTIES

Proposition 1, given a lower bound on the complexity of encoding sets, is proved by a contradiction. Thus, it gives no information about the density of the infinite set of arguments on which the set is complex.

For natural problems, it is clear that some classification sharper than "a.e. vs. i.o." is desirable. In this section, we consider conclusions about density that can be drawn for encoding sets in general. We see that there is a bound on the sparsity of encoding sets, but that this bound is too large to be of practical interest. The implication is that much more information about the specific problems than simply the fact that they are encoding sets, is needed to obtain interesting density results.

The first result deals with individual sets rather than complexity classes. It states that there exist arbitrarily complex sets $\leqslant_m^{\mathcal{P}}$-reducible to arbitrarily sparse sets.

DEFINITION. If $s: \Sigma^* \to N$ is recursive, and A is recursive, we say A *is s-sparse* if there is a Turing machine M computing C_A, and a polynomial p such that for any string x, M runs in time greater that $\lambda y[\, p(|\, y\,|)]$ for at most $\hat{x}$ strings of length $\leqslant s(x)$.

THEOREM 13. *For any* $s, t: \Sigma^* \to N$, *recursive, there exist* A, B, *recursive, such that*:

(i) Comp $B > t$ *a.e.*,

(ii) A *is s-sparse, and*

(iii) $B \leqslant_m^{\mathcal{P}} A$.

Proof. We assume without loss of generality that s is honest, and $(\forall x, y)[\hat{x} < \hat{y} \Rightarrow s(x) < s(y)]$. We also assume $(\forall x, y)[\hat{x} < \hat{y} \Rightarrow t(x) \leqslant t(y)]$, and $t(x) \geqslant 2^{|x|}$ for all x.

We define a new recursive $u: \Sigma^* \to N$, intended to be very large relative to s and t:

$$u(x) = \begin{cases} 0 & \text{if } x \neq 0^{s(i)} \text{ for any } i, \\ 2^{t(0^{s(j)})} & \text{if } x = 0^{s(i)}, \text{ and } j = i + 1. \end{cases}$$

Now in terms of s and u, we define sets A and B. First, we define $C_A(x) = 0$ for

strings x not of the form $0^{s(i)}$ for any i. Then on arguments of the form $0^{s(i)}$, we carry out a Rabin-type diagonal construction (similar to the Compression Theorem proof in [1]), to ensure that Comp $A > u$ a.e. on $\{0^{s(i)} \mid i \in \Sigma^*\}$.

B is then defined by

$$C_B(x) = \begin{cases} 0 & \text{if } |x| < s(\lambda), \\ C_A(0^{s(i)}) & \text{if } s(i) \leqslant |x| \text{ and } s(j) > |x| \text{ and } j = i + 1. \end{cases}$$

We claim that A and B satisfy the theorem:

(i) Comp $B > t$ *a.e.*

Assume Comp $B \leqslant t$ i.o. Consider the following Algorithm $\mathcal{A}$, which depends on B and s. We will see that it provides a contradiction to the lower bound on A's complexity arising from the Rabin diagonal construction:

ALGORITHM $\mathcal{A}$. Given input x, output 0 if $x \neq 0^{s(i)}$ for some i. Otherwise, dovetail the computations of:

$$C_B(0^{s(i)}), \qquad C_B(0^{s(i)-1}1), \ldots, C_B(1^{s(j)-1}),$$

using a fast algorithm for B, until one of these computations converges. When it does, output the answer.

END

If Comp $B \leqslant t$ i.o., as assumed, then Algorithm $\mathcal{A}$ yields infinitely many arguments $x = 0^{s(i)}$ for some i, such that the number of steps required to compute $C_A(x)$ may be bounded by $p(t(0^{s(j)}))$, $j = i + 1$, for some polynomial p. (The monotonicity and lower bound for t are used here.) But this contradicts the lower bound on A's complexity.

(ii) A *is* s*-sparse*

The following simple algorithm suffices.

ALGORITHM $\mathcal{B}$. Given input x, see if x is of the form $0^{s(i)}$ for any string i. If so, compute $C_A(x)$ by some fixed procedure. If not, output 0.

END

This procedure requires not more than $p(|x|)$ steps for some polynomial p, on all x not of the form $0^{s(i)}$, because of the monotonicity and honesty of s. (We must be careful to bound the length of time we spend on computing s on any argument x to $p_1(|x|)$, for p_1 a (monotone) polynomial arising from the definition of the honesty of s.)

Moreover, for any x, there are at most $\hat{x}$ strings of the form $0^{s(i)}$, of length $s(x)$.

(iii) $B \leqslant_m^{\mathscr{P}} A$

This is straightforward by the definition of B from A, provided we use the bound p_1 as above. ∎

Although we have just shown that there exist arbitrarily complex sets encoded by arbitrarily sparse sets, we will now note that no *single* nontrivial set is reducible to arbitrarily sparse sets. Thus, no nontrivial complexity class is encoded by arbitrarily sparse sets. In [8] is proved:

PROPOSITION 14. *($\forall s$, recursive)($\exists A$, recursive)[$B \leqslant_m^{\mathscr{P}} A$ and A is s-sparse] iff B is polynomial computable.*

This easily implies

COROLLARY 15. *There exist $s, t: \Sigma^* \to N$, recursive, such that: ($\forall A$, recursive) [A m-encodes $t \Rightarrow A$ is not s-sparse].*

Even more strongly, in [15] is proved:

PROPOSITION 16. *($\forall s$, recursive)($\exists A$, recursive)[$B \leqslant_T^{\mathscr{P}} A$ and A is s-sparse] iff B is polynomial computable.*

Thus,

COROLLARY 17. *There exist $s, t: \Sigma^* \to N$, recursive, such that: ($\forall A$, recursive) [A T-encodes $t \Rightarrow A$ is not s-sparse].*

Corollaries 15 and 17 yield minimum density results for the set of arguments on which an encoding set is complex. However, the functions s which arise via the proofs of Propositions 14 and 16 are probably too large to be of practical interest; this is especially true in the latter case.

We may look at these results in another way. It is not difficult to see that there exist sets which are both arbitrarily sparse and arbitrarily complex i.o. However, Corollaries 15 and 17 say that, for sets which are sufficiently sparse, encoding techniques are useless even for inferring moderate i.o. complexity.

At this point, an interesting related question is naturally suggested. "Sparsity" in the literature commonly refers to the number of elements in the set rather than the number of elements on which the set is complex. McCreight and Meyer [12] have shown that there are arbitrarily a.e. complex sets which are sparse in the usual sense. We ask whether encoding sets can be sparse in the same sense.

DEFINITION. If $s: \Sigma^* \to N$ is recursive and A is recursive, we say A *is s-thin* if for any string x,

$$| A \cap \{y \mid | y | \leqslant s(x)\}| \leqslant \hat{x}.$$

Question. Do there exist $s, t: \Sigma^* \to N$, recursive, such that: $(\forall A$, recursive$)$ [A T-encodes $t \Rightarrow A$ is not s-thin]?

6. Helping Relationships Among Encoding Sets

In [10] are presented several results showing the existence of arbitrarily complex (a.e.) sets having various types of "helping" relationships; for instance, there are pairs of arbitrarily complex sets with an independence property-even with the aid of one of the sets as an oracle, the other is still approximately as difficult to compute as without the oracle:

Proposition 18. *For all t_1, $t_2 : \Sigma^* \to N$, honest, with $t_1(x) \geqslant |x|$ and $t_2(x) \geqslant |x|$ for all x, there exist recursive sets A and B with:*

(i) $\text{Comp } A \leqslant \lambda x[p(2^{t_1(x)})]$
(ii) $\text{Comp } B \leqslant \lambda x[p(2^{t_2(x)})]$ } for some polynomial p,
(iii) $\text{Comp}^{(A)} B > t_2$ *a.e.*,
(iv) $\text{Comp}^{(B)} A > t_1$ *a.e.*

Proof. The arguments in [10, Sect. 4] may be redone for the Turing machine time measure. ∎

The two sets in Proposition 18 have the property that they cannot "help" each other's computation on any infinite set of arguments; that is, for any Turing machine using an oracle for the other set, there is another Turing machine not using *any* oracle, computing the same function with at most exponential loss of efficiency.

Intuitively, it seems very likely that there are pairs of natural problems that have this kind of independence property; after all, it is improbable that values of *every* interesting recursive function could be useful in computing every other interesting recursive function. However, known examples of natural problems having inherent complexity i.o. are encoding sets, and for this type of set, the independence property cannot hold, as shown below. Thus, the existence of pairs of natural, independent sets apparently cannot be shown until further techniques for proving lower bounds are developed.

To show that encoding sets are not independent, we need some (reasonable) conditions on the complexity class encoded and on the complexity of the encoding sets. We use the following:

Definition. If $r: \Sigma^* \to \Sigma^*$, $s, t: \Sigma^* \to N$ are recursive, we say *r spreads s to t* if:

(i) r is monotone increasing in its argument,

(ii) there exists a polynomial p and a Turing machine computing r in $\lambda x[p(|r(x)|)]$ steps, and

(iii) $s(x) \leqslant t(r(x))$ for all x.

Thus, for example, the function r defined by:

$$r(x) = \underbrace{x0 \cdots 0}_{\text{length } = 2^{2^{|x|}}}$$

has the property that r spreads

$$\lambda x \left[2^{2^{2^{2^{2^{|x|}}}}} \right] \quad \text{to} \quad \lambda x \left[2^{2^{2^{|x|}}} \right].$$

The next theorem shows how, with compatibility requirements as in the definition above, an encoding set lowers a known upper bound for the complexity of other sets, when it is used as an oracle in their computation. Bounds of the sort actually proved so far do satisfy the compatibility requirements. The basic ideas in this theorem were noted by Meyer [*M*].

THEOREM 19. *If* $r: \Sigma^* \to \Sigma^*$, $s, t: \Sigma^* \to N$ *are recursive functions, with* $t(x) \geqslant 2^{|x|}$ *for all* x, *and we have*:

(i) *r spreads s to t,*

(ii) Comp $A \leqslant s$ *a.e., and*

(iii) *B T-encodes* $\lambda x[p(t(x))]$, *for all polynomials p, then* $\text{Comp}^{(B)} A \leqslant \lambda x[p(|r(x)|)]$ *for some polynomial p.*

Proof. Assume r, s, t, A, and B are given. We define a new "spread-out" version A' of A:

$$x \in A' \Leftrightarrow (\exists y \in A)[x = r(y)].$$

We claim first that Comp $A' \leqslant \lambda x[p_1(t(x))]$ for some polynomial p_1.

Given input string x, we compute $r(\lambda)$, $r(0)$, $r(1), \ldots, r(x)$ to see if any equal x; property (i) of the above definition guarantees that this search is sufficient. Property (ii) of the same definition bounds the amount of time we must spend on each computation. If none of these computations gives output x in the required time, we output 0. Otherwise, assume we have found string y with $r(y) = x$. We compute and output $C_A(y)$. In the worst case, the total time needed may be bounded by $p_2(2^{|x|}, s(y))$, for some polynomial p_2 (by hypothesis (ii) of the theorem). By the lower bound on t and by property (iii) of the definition, the time may be bounded by $p_1(t(x))$, as needed.

By the definition of A', it is clear that $\mathrm{Comp}^{(A')} A \leqslant \lambda x[p_3(|r(x)|)]$ for some polynomial p_3. Also, by hypothesis (iii) of the theorem, $A' \leqslant_T^{\mathscr{P}} B$. By combining these two facts, it is easy to show that $\mathrm{Comp}^{(B)} A \leqslant \lambda x[p(|r(x)|)]$ for some polynomial, as needed. ▮

Thus, if A has an i.o. lower bound on its complexity much above $\lambda x(|r(x)|)$, (shown, say, by an encoding argument) then having the help of B as an oracle must lower the complexity of A i.o. We conclude that if A and B are two sets with large i.o. complexity demonstrated by encoding arguments, then they cannot have an independence property if their complexity bounds are reasonably compatible (i.e., of the "same shape").

For example, consider a set A such that

$$\mathrm{Comp}\, A > \lambda x \left[p\left(2^{2^{2^{|x|}}} \right) \right] \quad \text{i.o. for all polynomials } p$$

and

$$\mathrm{Comp}\, A \leqslant \lambda x \left[2^{2^{2^{2^{|x|}}}} \right] \quad \text{a.e.,}$$

and any recursive set B such that

$$B\ T\text{-encodes } \lambda x \left[p\left(2^{2^{2^{|x|}}} \right) \right] \quad \text{for all polynomials } p.$$

Defining

$$r(x) = \underbrace{x0 \cdots 0}_{\text{length}} = 2^{2^{|x|}}$$

allows us to apply Theorem 19 and conclude:

$$\mathrm{Comp}^{(B)} A \leqslant \lambda x \left[p\left(2^{2^{|x|}} \right) \right] \quad \text{for some polynomial } p.$$

Thus, B "helps" A by two exponentials.

7. Differences Between Encoding Definitions

In Theorem 7 and Corollary 12, we have a proof that T-encoding and m-encoding are distinct relations. In this section, we make this distinction explicit. We then suggest the possibility of definitions of encoding other than T-encoding and m-encoding, such as "truth-table-encoding."

Theorem 20. *If $t: \Sigma^* \to N$ is honest, monotone nondecreasing in the length of its argument, and sufficiently large, then there exist a recursive set A and a polynomial p such that:*

(i) *A T-encodes t,*

(ii) *$\neg A$ m-encodes t, and*

(iii) $\text{Comp } A \leqslant \lambda x[p(t(x))]$.

Proof. We use a lemma of interest in itself:

Lemma 21. $(\exists B$, *recursive*$)(\forall C$, *recursive*$)(\exists A$, *recursive*$)[B \leqslant_m^{\mathscr{P}} A$ *and* $C \leqslant_T^{\mathscr{P}} A]$. *Furthermore, if $t: \Sigma^* \to N$ is recursive, monotone nondecreasing in the length of its input, and sufficiently large, then for any recursive C with* $\text{Comp } C \leqslant t$, *the A produced above has* $\text{Comp } A \leqslant \lambda x[p(t(x))]$ *for some polynomial p independent of C and t.*

Proof of Lemma 21. B, along with a partial determination of A, is defined in stages, beginning with stage 1. At stage n, C_B is determined on one or several consecutive strings, and C_A may also be fixed at 0 on either one or two arguments. We will arrange the construction so that by the end of stage $\hat{x}$, exactly one of $C_A(x0)$ and $C_A(x1)$ will have been determined to equal 0. The strings $x0$ and $x1$ will be called "mates."

(We will eventually define $x \in C \Leftrightarrow (x0 \in A$ or $x1 \in A)$.) Define $C_A(\lambda) = 0$.

Stage n.

Substage 1. Let y be the first string such that $C_B(y)$ is as yet undefined. See if $T_{\pi_1(n)}(y) \leqslant 2^{|y|}$. If not, let $C_B(y) = 0$ and go on to substage 2. If so, we will diagonalize over $\phi_{\pi_1(n)}$ as a many-one procedure:

(a) If $C_A(\phi_{\pi_1(n)}(y)$ is determined to equal 0, we let $C_B(y) = 1$, and go to substage 2.

(b) If $C_A(\phi_{\pi_1(n)}(y))$ is not determined to equal 0, and also C_A is not already equal to 0 for the *mate* of $\phi_{\pi_1(n)}(y)$, we let $C_B(y) = 1$ and $C_A(\phi_{\pi_1(n)}(y)) = 0$, and go to substage 2.

(c) Otherwise, $C_A(\phi_{\pi_1(n)}(y))$ is not already defined to be 0, but C_A is defined to be 0 for the mate of $\phi_{\pi_1(n)}(y)$. We then simulate substage 1, parts (a) and (b), for successively higher strings y. Since only finitely many values of C_A are determined at any stage, two eventual outcomes are possible: either

(i) some string y would cause an exit to substage 2, or

(ii) we will discover two strings y', y'' with $\widehat{y'} < \widehat{y''}$ and $\phi_{\pi_1(n)}(y') = \phi_{\pi_1(n)}(y'')$.

If the former occurs, then before exiting to substage 2, we augment the required definition of C_B in such a way as to ensure that C_B is defined on an initial segment.

If the latter occurs, then define $C_B(y') = 0$ and $C_B(y'') = 1$, augment the definition of C_B to keep the initial segment definition, and go to substage 2.

Substage 2. If $\hat{x} = n$, and if both $C_A(x0)$ and $C_A(x1)$ are still undefined, let $C_A(x0) = 0$.

END

This succession of stages completely defines B. Clearly, for any set A satisfying the partial determination above, $B \not\leqslant_m^{\mathscr{P}} A$. If C is any recursive set, define A so that:

$$x \in C \Leftrightarrow (x0 \in A \text{ or } x1 \in A).$$

This insures $C \leqslant_T^{\mathscr{P}} A$.

Finally, if t is as hypothesized, then the time to compute either $C_A(x0)$ or $C_A(x1)$ is just the time needed to complete $\hat{x}$ stages in the construction of C_B, then possibly to compute $C_C(x)$, which may be bounded as in the statement of the lemma. ▮

Proof of Theorem 20 (continued). Obtain B as in the lemma, and let $t: \Sigma^* \to N$ be honest, $t(x) \geqslant |x|$ for all x, t monotone nondecreasing in the length of its input, t sufficiently large as required for Lemma 21, and Comp $B \leqslant t$.

By Theorem 2, there exists a recursive set C such that C m-encodes t and Comp $C \leqslant \lambda x[p(t(x))]$ for some polynomial p.

By Lemma 21, we obtain A recursive such that $B \not\leqslant_m^{\mathscr{P}} A$, $C \leqslant_T^{\mathscr{P}} A$, and Comp $A \leqslant \lambda x[p(t(x))]$ for some polynomial p.

Since C m-encodes t, A T-encodes t.

Since $B \not\leqslant_m^{\mathscr{P}} A$ and Comp $B \leqslant t$, $\neg A$ m-encodes t. ▮

Just as in the proof in [9] that $\leqslant_T^{\mathscr{P}} \neq \leqslant_m^{\mathscr{P}}$, Theorem 20 actually proves a somewhat stronger result than stated. Specifically we have not used the full power of T-encoding; actually, A encodes t by a very simple 2-question disjunctive truth-table procedure. The same remark is true, for example, for Lemma 8. It would seem reasonable, therefore, to consider other definitions of encoding, corresponding to the various polynomial truth-table reducibilities studied in [9]. For instance,

DEFINITION. *A tt-encodes t* for a recursive set A and recursive function $t: \Sigma^* \to N$, provided

$$(\forall B)[\text{Comp } B \leqslant t \Rightarrow B \leqslant_{tt}^{\mathscr{P}} A].$$

(For the formal definition of $\leqslant_{tt}^{\mathscr{P}}$, we refer to [9].)

Most questions about tt-encoding seem to have straightforward answers, given the answers to the corresponding questions for m-encoding and T-encoding. However,

one interesting open question is whether a result analogous to Theorem 20 can be proved, distinguishing between T-encoding and tt-encoding with a set A of small complexity. Part of the difficulty in proving such a result is that we do not know whether the appropriate analog for Lemma 21 is true. A somewhat weaker result, sufficient to show that T-encoding and tt-encoding are distinct relations, appears in [7].

Note. As mentioned earlier in the paper, an unusually long time has elapsed between the initial writing of this paper and its appearance. In the interim, some of the results have been superseded and therefore eliminated from the final version. In addition, some results possibly lesser interest, some examples and some details of proofs have been omitted. The complete original version of the paper appears as [7].

References

1. M. Blum, A machine-independent theory of the complexity of recursive functions, *J. Assoc. Comput. Mach.* (1967), 322–336.
2. S. A. Cook, The complexity of theorem-proving procedures, *in* "Conf. Rec. Third ACM Symposium on Theory of Computing," pp. 151–158, 1971.
3. M. Fischer and M. Rabin, Super-exponential complexity of Presburger arithmetic, *in* "Project Mac TM," Vol. **43**, 1974.
4. J. Hartmanis and J. Hopcroft, An overview of the theory of computational complexity, *J. Assoc. Comput. Mach.* (1971), 444–473.
5. J. Hopcroft and J. Ullman, "Formal Languages and Their Relation to Automata," Addison–Wesley, Reading, Mass., 1969.
6. R. Karp, Reducibility among combinatorial problems, *in* "Complexity of Computer Computations" (R. Miller and J. Thatcher, Eds.), Plenum Press, New York, 1972.
7. N. Lynch, Complexity-class-encoding sets, USC Department of Mathematics, Preprint No. 57, 1974.
8. N. Lynch, On reducibility to complex or sparse sets, *J. Assoc. Comput. Mach.* (1975), 341–345.
9. R. Ladner, N. Lynch, and A. Selman, A comparison of polynomial-time reducibilites, *TCS*, to appear.
10. N. Lynch, A. Meyer, and M. Fischer, Relativization of the theory of computational complexity, *Trans. Amer. Math. Soc.*, to appear.
11. A. Meyer, Personal communication.
12. E. McCreight and M. Meyer, Computationally complex and pseudo-random zero–one valued functions.
13. A. Meyer and L. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, *in* "SWAT Conference Record," 1972.
14. H. Rogers, "Theory of Recursive Functions and Effective and Effective Computability," McGraw–Hill, New York, 1967.
15. R. Solovay, On sets Cook-reducible to sparse sets, IBM RC 5215, 1975.

Printed in Belgium