JOURNAL OF COMPUTER AND SYSTEM SCIENCES: 10, 183–199 (1975)

On the Density of Honest Subrecursive Classes

MICHAEL MACHTEY*,†

Purdue University, Department of Mathematics and Computer Sciences, Lafayette, Indiana 47907

Received April 11, 1973

The relation of honest subrecursive classes to the computational complexity of the functions they contain is briefly reviewed. It is shown that the honest subrecursive classes are dense under the partial ordering of set inclusion. In fact, any countable partial ordering can be embedded in the gap between an effective increasing sequence of honest subrecursive classes and an honest subrecursive class which is properly above the sequence (or in the gap between an effective decreasing sequence and a class which is properly below the sequence). Information is obtained about the possible existence of least upper bounds (greatest lower bounds) of increasing (decreasing) sequences of honest subrecursive classes. Finally it is shown that for any two honest subrecursive classes, one properly containing the other, there exists a pair of incomparable honest subrecursive classes such that the greatest lower bound of the pair is the smaller of the first two classes, and the least upper bound of the pair is the larger of the first two classes.

1. INTRODUCTION

There has been much work on classifying computable functions into hierarchies. A noted example is the Grzegorczyk hierarchy [5]. The fact that these hierarchies do not classify all of the computable functions has led to work on classifying computable functions in subrecursive classes [7, 8, 9] with the same closure properties as the classes in the hierarchies. These systems of subrecursive classes include those of the elementary classes and the primitive recursive classes of computable functions. The classes in many hierarchies bear a close relation to the computational complexity of the functions they contain. This has led to study of honest subrecursive classes which enjoy the same relation to the computational complexity of the functions they contain.

* Supported in part by Air Force Office of Scientific Research, Grant No. 71-2205A, and National Science Foundation, Grant No. GJ27172A1.

† The author wishes to thank the referee for several helpful suggestions for improving the exposition of this paper.

The main result of this paper is a strong density theorem for the honest subrecursive classes. It is shown that any countable partial ordering can be embedded in the gap between an effective increasing sequence of classes and a class which is properly above it. As a corollary, we deduce that no effective properly increasing sequence of classes has a least upper bound. It is further shown that a slightly non-effective properly increasing sequence of honest classes may have a least upper bound among the honest classes. These results settle two conjectures and an open problem of Meyer and Ritchie [9] raised in the context of elementary-honest classes. Finally, it is shown that for any two honest classes, one properly containing the other, there exists a pair of incomparable honest classes such that the greatest lower bound (intersection) of the pair is the smaller of the first two classes, and the least upper bound of the pair is the larger of the first two classes.

2. Preliminaries

We shall be studying several systems of subrecursive classes of computable functions. One such system is that of the primitive recursive classes [7], and another is that of the elementary classes of Meyer and Ritchie [9]. Other systems to which the work of this paper applies are those of the doubly recursive, triply recursive, . . . , or multiply recursive classes defined analogously (see [11]). Since the work below could be referring equally well to any of these systems of subrecursive classes, $\mathbf{C}(f)$ will be used to denote the subrecursive class generated by the computable function f. Also, for each of the systems considered there is an effective (in fact, elementary) list C_0, C_1, C_2,..., of recursive operators such that

$$\mathbf{C}(f) = \{C_i(f): i \in N\},$$

where N stands for the natural numbers. The reader is referred to Rogers [13] for the terminology and notation of recursive function theory.

The significance of the honest subrecursive classes lies in their relation to the computational complexity of the functions they contain. Although this relation is invariant over a wide range of "natural" computational complexity measures, for the sake of concreteness and simplicity a specific measure will be used in this paper. The measure is that of Turing machine space based on the definitions and conventions for Turing machines introduced in Davis [4] along with the Gödel numbering used there. Specifically, if i is the Gödel number of a Turing machine then the ith Turing machine M_i will be that Turing machine, and if i is not the Gödel number of a Turing machine then M_i will be the Turing machine $\{q_0\, 1Lq_0, q_0BLq_0\}$ (i.e., a Turing machine which computes the totally undefined function). If $\mathbf{x}$ denotes an arbitrary string x_1, x_2,..., x_n of natural number arguments then $\varphi_i(\mathbf{x})$ will denote the result of running the ith Turing machine M_i on inputs $\mathbf{x}$; $\varphi_i(\mathbf{x})$ will be undefined (or divergent) if this computation fails to halt. The *space functions* are defined as follows. $S_i(\mathbf{x})$ is the number of squares of tape needed for the input and computation of $\varphi_i(\mathbf{x})$ by M_i on input $\mathbf{x}$ if $\varphi_i(\mathbf{x})$ is convergent, and $S_i(\mathbf{x})$ is divergent if $\varphi_i(\mathbf{x})$ is divergent. It is well known that this Gödel numbering is an acceptable Gödel numbering in the sense of Rogers [14], and that the space functions give a computational complexity measure in the sense of Blum [2]. It will be assumed that the reader is familiar with many of the standard "programming" techniques for Turing machines such as the use of special markers and the use of large alphabets to simulate multitrack tapes in no additional space. Hopcroft and Ullman [6] is one source of such techniques.

We now proceed to sketch some fairly well-known material on Turing machines, space functions, and subrecursive classes, which will be needed later in the paper. Proofs appear in [8].

DEFINITION 2.1. A partial computable function ψ is *tape constructible* if there is a Turing machine M_i such that

$$\psi = \varphi_i = S_i .$$

PROPOSITION 2.2. *For all i, S_i is tape constructible; thus the tape constructible functions are exactly the space functions.*

PROPOSITION 2.3. *The tape constructible functions are closed under the operations of summation, minimalization, maximalization, and summation and maximalization of a single function up to the given argument. That is, if S_i and S_j are tape constructible functions then so are $S_i + S_j$, $\min(S_i, S_j)$, $\max(S_i, S_j)$, as well as φ and ψ such that*

$$\varphi(x) = \sum_{y \leqslant x} S_i(y) \qquad \textit{and} \qquad \psi(x) = \max_{y \leqslant x} S_i(y).$$

Moreover, there are elementary functions $\overline{\min}$, $\overline{\max}$, etc., such that $\min(S_i, S_j) = S_{\overline{\min}(i,j)}$, $\max(S_i, S_j) = S_{\overline{\max}(i,j)}$, *etc.*

PROPOSITION 2.4. *If $\varphi_i = \varphi_j$ then there is a k such that $\varphi_i = \varphi_j = \varphi_k$ and $S_k =$* $\min(S_i, S_j)$. *This is sometimes called the parallel computation property.*

The following two propositions represent key insights into the relation between subrecursive classes and computational complexity, insights which appear to have been arrived at independently by several authors, notably Ritchie [12] and Cobham [3]. First, though, we introduce some useful notation.

Definition 2.5. If f and g are total functions then $f < g$ will mean that $f(\mathbf{x}) < g(\mathbf{x})$ for all $\mathbf{x}$, and $f < g$ a.e. will mean that $f(\mathbf{x}) < g(\mathbf{x})$ for all but finitely many $\mathbf{x}$; $f < \mathbf{C}(g)$ will mean that $f < C_i(g)$ for some i. Similarly for $\leqslant$.

Proposition 2.6. *If $S_i < \mathbf{C}(g)$ then $\varphi_i \in \mathbf{C}(g)$.*

Proposition 2.7. *Let f and g be total computable functions with $f \in \mathbf{C}(g)$. For each i such that $\varphi_i = g$ there is a j such that $\varphi_j = f$ and $S_j \in \mathbf{C}(S_i)$. That is for example, if f is primitive recursive in g then for every running space for g there is a running space for f which is primitive recursive in it.*

Definition 2.8. If f is a total function then let $G_f(\mathbf{x}, y) = 1$ if $f(\mathbf{x}) = y$ and let $G_f(\mathbf{x}, y) = 0$ otherwise; G_f is the *graph* of the function f. Let $\mathbf{C}(0)$ be the subrecursive class generated by the zero function (i.e., $\mathbf{C}(0)$ is the class of elementary functions, or the class of primitive recursive functions, etc.). Let g be a total computable function, then $\mathbf{C}(g)$ is honest if $\mathbf{C}(g) = \mathbf{C}(f)$ for some function f such that $G_f \in \mathbf{C}(0)$; g is *honest* if $\mathbf{C}(g)$ is honest. $\mathbf{C}(g)$ and g are *dishonest* if they are not honest.

Note that if $\mathbf{C}(g)$ is the class of functions elementary in g then the honest $\mathbf{C}$-classes are the elementary-honest classes studied by Meyer and Ritchie [9]; if $\mathbf{C}(g)$ is the class of functions primitive recursive in g then the honest $\mathbf{C}$-classes are the honest Pr-classes studied in [7].

Proposition 2.9. *For all i, if S_i is total then S_i is honest.*

Proposition 2.10. *Let g be a total computable function. The following are equivalent.*

(a) *g is honest;*
(b) *there is an i such that $\varphi_i = g$ and $S_i \in \mathbf{C}(g)$;*
(c) *there is a j such that φ_j is total and $\mathbf{C}(g) = \mathbf{C}(S_j)$.*

If t is a total computable function then the *computational complexity class* determined by t is defined as follows.

$$C_t = \{\varphi_i \colon \varphi_i \text{ is total and } S_i < t \text{ a.e.}\}.$$

C_t is the class of functions which can be computed in space t.

Proposition 2.11. *Let g be a total computable function. $\mathbf{C}(g)$ is a computational complexity class if and only if g is honest.*

The following proposition is a slight extension of one proved by Axt [1] for the notion of relative primitive recursiveness, but it holds and the same proof techniques work for all of the subrecursive reducibilities beings studied here.

PROPOSITION 2.12. *There is an elementary function k such that if f and g are total computable functions such that $f < g$ then for all j, $C_j(f) < C_{k(j)}(g)$.*

Note that if f is honest and $f \leqslant g$ then there is an i such that $\mathbf{C}(f) = \mathbf{C}(S_i)$. By Proposition 2.12, $S_i < \mathbf{C}(g)$; thus by Propositions 2.2 and 2.6, $S_i \in \mathbf{C}(g)$. Therefore $\mathbf{C}(f) \subseteq \mathbf{C}(g)$.

At this point a comment about the simulation of Turing machines by other Turing machines is in order. If we wish to build a Turing machine to simulate another Turing machine (or a fixed finite set of other Turing machines), then we may do so with the machine we build using no more space for the simulation than the original machine used for its computation. On the other hand, suppose we are building a Turing machine to perform simulations of computations by some infinite set of Turing machines. Then because some of the machines being simulated may have much larger alphabets than the machine being built, the simulations cannot always be performed in the same amount of space as the original computations. However, for each Turing machine in the set being simulated there will be a constant such that simulations of computations by that machine can be performed by the built machine in space at most equal to that constant times the space of the original computation. Moreover, the constant is an elementary function of the Gödel number of the machine being simulated, and in any case is no greater than the Gödel number of the machine being simulated. This observation will be used later in the paper.

3. DENSITY

The purpose of this section is to prove the strong density results for the honest $\mathbf{C}$-classes. These will establish Conjectures 2 and 3 of Meyer and Ritchie [9, p. 81] made in the context of elementary-honest classes. We begin with a simple density theorem. This theorem is a special case of Theorem 3.2, and therefore its proof will only be sketched as a helpful warm-up enabling the introduction in a simpler context of some of the techniques to be used in the proofs of the stronger density results.

THEOREM 3.1. *Let f and g be honest functions such that $\mathbf{C}(f) \subset \mathbf{C}(g)$ (where $\subset$ stands for proper containment). Then there is an honest function h such that $\mathbf{C}(f) \subset \mathbf{C}(h) \subset \mathbf{C}(g)$.*

Proof. Without loss of generality we may assume that f and g are tape constructible functions of one argument such that $f < g$ and such that for all x, $f(x) + 1 < f(x + 1)$ and $g(x) + 1 < g(x + 1)$. We shall produce a tape constructible (hence, honest) function h such that $f \leqslant h \leqslant g$ yielding $\mathbf{C}(f) \subseteq \mathbf{C}(h) \subseteq \mathbf{C}(g)$. It remains to build h subject to these constraints in a manner which will make the containments proper.

The basic idea is for h to alternate between following f and following g, doing each for enough arguments each time to guarantee that $C_i(h) \neq g$ or $C_i(f) \neq h$, respectively, for one new i. The only problem is that in order to keep h honest we may not have enough space to discover such a diagonalization as soon as it has occurred. Since f and g are increasing, eventually there will be enough space to discover that the diagonalization took place at a previous argument and our patience will be amply rewarded.

Let M_f be a Turing machine which computes f such that $S_f = f$, and let M_g be a Turing machine which computes g such that $S_g = g$. We use a multitrack Turing machine M to compute h. On input $x + 1$, M recapitulates its computations on inputs $0, \ldots, x$ to find $h(x)$ plus a current index n and a current mode (f, g, or c). In the mode f, h follows f until it is discovered that $C_n(h) \neq g$. M simulates M_f on input $x + 1$ and marks off $f(x + 1)$ tape squares (on all tracks). M then uses this space to compute as many as possible of the values $g(0), g(1), g(2), \ldots, h(0), h(1), h(2), \ldots$, and $C_n(h)(0)$, $C_n(h)(1)$, $C_n(h)(2), \ldots$. If this computation discloses a y such that $C_n(h)(y) \neq g(y)$ then the new mode is g. If no such y is found then the mode is not changed. In either case $h(x + 1) = f(x + 1)$ and the index is not changed. In the mode g, h follows g until it is discovered that $C_n(f) \neq h$. M marks off $g(x + 1)$ tape squares and uses this space to look for a y such that $C_n(f)(y) \neq h(y)$. If such a y is found, the new mode is c; if no such y is found the mode is not changed. In either case $h(x + 1) = g(x + 1)$ and the index is not changed. In the mode c, h is *coasting* from g down to f to keep h increasing and honest. If $h(x) < f(x + 1)$ then $h(x + 1) = f(x + 1)$, the new mode is f and the new index is $n + 1$; otherwise $h(x + 1) = h(x) + 1$ and the mode and index are not changed.

The function h is strictly increasing and can be seen by induction on the argument x to be tape constructible; since h is increasing the recapitulation can be done without interfering with tape constructibility. Also $f \leqslant h \leqslant g$ as required. If M stays in mode f for infinitely many consecutive arguments then $h = f$ a.e. Therefore $\mathbf{C}(h) = \mathbf{C}(f) \subset \mathbf{C}(g)$ and hence for the appropriate index n, $C_n(h)(y) \neq g(y)$ for some y. Since f is increasing, for sufficiently large arguments there is enough space to discover this fact in the computation of $h(x)$. Thus M can stay in mode f for only finitely many consecutive arguments. Similarly, M can stay in mode g for only finitely many consecutive arguments. Clearly, M can stay in mode c for only finitely many consecutive arguments, and while the index does not change, M cannot return to a mode once it has left that mode. Therefore the index increases without bound as the argument x increases. Once the index is greater than i, $C_i(f) \neq h$ and $C_i(h) \neq g$, yielding $\mathbf{C}(f) \subset \mathbf{C}(h) \subset \mathbf{C}(g)$.

In the next theorem we shall prove that in fact any countable partial ordering can be embedded in the honest $\mathbf{C}$-classes between $\mathbf{C}(f)$ and $\mathbf{C}(g)$. It was shown in [8] that there is a primitive recursive partial ordering of the natural numbers in which every countable partial ordering can be embedded. The same techniques can be used to get an elementary partial ordering of the natural numbers in which every countable partial ordering can be embedded, but for the sake of completeness the construction of such an ordering will be sketched here. Mostowski [10] has given a recursive partial ordering R in which every countable partial ordering can be embedded; we shall show how to embed R in an elementary partial ordering of the natural numbers.

Let M be a Turing machine which computes the partial ordering R; that is on inputs x and y, M gives output 1 if $x \leqslant_R y$ and M gives output 0 if $x \not\leqslant_R y$. Let S be the space function for M and let

$$s(x) = \max_{y,z \leqslant x} S(y, z);$$

that is, $s(x)$ is the space required for M to compute R on $\{0,..., x\}$. Then let i be such that $s \leqslant S_i$ and S_i is strictly increasing.

We now define the elementary partial ordering P of the natural numbers in which R (and hence every countable partial ordering) can be embedded. If x and y are both not in the range of S_i then $x \leqslant_P y$ if and only if $x \leqslant y$; if x is not in the range of S_i and y is in the range of S_i then $x \leqslant_P y$ and $y \not\leqslant_P x$; if $S_i(j) = x$ and $S_i(k) = y$ then $x \leqslant_P y$ if and only if $j \leqslant_R k$. Note that a Turing machine can, on input x, determine whether x is in the range of S_i, and if x is in the range of S_i can find j such that $S_i(j) = x$, using only the space needed for the input x; this is because S_i is strictly increasing and tape constructible. Therefore a Turing machine can, on inputs x and y, compute the partial ordering P on x and y using only the space required for the inputs, and so the partial ordering P is certainly elementary. Note that S_i gives an effective embedding of the ordering R into the ordering P. There are elementary functions s and t such that for all i, $s(i) \not\leqslant_P t(i)$ and such that if x and y are such that $x \not\leqslant_P y$ then there are infinitely many i such that $s(i) = x$ and $t(i) = y$. Finally, for convenience we shall assume that $i \leqslant_P 0$ for all i.

THEOREM 3.2. *Let f and g be honest functions such that $\mathbf{C}(f) \subset \mathbf{C}(g)$. Then any countable partial ordering can be isomorphically embedded in the honest* **C**-*classes between* $\mathbf{C}(f)$ *and* $\mathbf{C}(g)$.

Proof. Without loss of generality we may assume that f and g are tape constructible functions of one argument such that $f < g$ and such that for all x, $f(x) + 1 < f(x + 1)$ and $g(x) + 1 < g(x + 1)$. Moreover we shall assume that $S_s, S_t < f$ where S_s and S_t are space functions for computing the elementary functions s and t. Let M_f be a Turing machine which computes f such that $S_f = f$, and let M_g be a Turing machine which computes g such that $S_g = g$. We shall construct an embedding h of the partial ordering P into the honest **C**-classes between $\mathbf{C}(f)$ and $\mathbf{C}(g)$; the construction will give multitrack Turing machines $M_{h(i)}$ such that $\mathbf{C}(\varphi_{h(i)})$ is the image of i. For notational convenience we shall use M_i to denote $M_{h(i)}$ and h_i to denote $\varphi_{h(i)}$. The computation of M_i uses an index n and a mode c, d, f, g, or o; for any argument x, the index and mode will actually be independent of i. As in the proof of Theorem 3.1, on input $x + 1$, M_i first recapitulates its computations on inputs $0,..., x$ in order to find $h_i(x)$ and the current index and mode. The index is used to insure that the construction deals properly with all of the operators C_j; the purpose of the modes will be explained in the following cases.

Case 1

The mode is f. This mode guarantees that $\mathbf{C}(h_i) \subset \mathbf{C}(g)$ by having h_i follow f until it is discovered that $C_n(h_i) \neq g$. M_i simulates M_f on input $x + 1$ using end markers, and when this is done marks off $f(x + 1)$ tape squares on all tracks (this portion will eventually be filled with "1"s to give M_i's output). M_i then uses the space that has been marked off to compute as many as possible of the values $g(0)$, $g(1)$, $g(2)$,..., $h_i(0)$, $h_i(1)$, $h_i(2)$,..., and $C_n(h_i)(0)$, $C_n(h_i)(1)$, $C_n(h_i)(2)$,... . If this computation discloses a y such that $C_n(h_i)(y) \neq g(y)$ then the new mode is g. If no such y is found, the mode is not changed. In either case $h_i(x + 1) = f(x + 1)$ and the index is not changed.

Case 2

The mode is g. This mode guarantees that $\mathbf{C}(f) \subset \mathbf{C}(h_i)$ by having h_i follow g. Similarly to Case 1, M_i marks off $g(x + 1)$ tape squares and uses that space to look for a y such that $f(y) \neq C_n(h_i)(y)$. If such a y is discovered, the new mode is c; otherwise the mode is not changed. In either case $h_i(x + 1) = g(x + 1)$ and the index is not changed.

Case 3

The mode is c. In this mode h_i is coasting from g down to f. M_i simulates M_f on input $x + 1$ to obtain $f(x + 1)$. If $h_i(x) < f(x + 1)$ then $h_i(x + 1) = f(x + 1)$ and the new mode is o. Otherwise $h_i(x + 1) = h_i(x) + 1$ and the mode is not changed. In either case the index is not changed.

Case 4

The mode is o. This mode is to get $\mathbf{C}(h_i)$ to occupy its proper place in the ordering. This case has two subcases.

Subcase A. $s(n) \leqslant_P i$. M_i marks off $f(x + 1)$ tape squares and in that space looks for a y such that $h_{s(n)}(y) > C_j(h_{t(n)})(y)$ for all $j \leqslant n$. If such a y is discovered, the new mode is d; otherwise the mode is not changed. In either case $h_i(x + 1) = g(x + 1)$ and the index is not changed.

Subcase B. Otherwise, M_i behaves the same as in Subcase A except that $h_i(x+1) = f(x+1)$.

Case 5

The mode is d. This is another coasting mode to get the h_i's together and down to f. If $h_i(x) < f(x+1)$ then $h_i(x+1) = f(x+1)$; otherwise $h_i(x+1) = h_i(x) + 1$. If $f(x+1) < h_0(x) + 1$ then the mode and index are not changed; otherwise the new mode is f and the new index is $n+1$.

This completes the description of the computation of h_i. Clearly h_i is strictly increasing and $f \leqslant h_i \leqslant g$ for all i. The rest of the demonstration that this construction accomplishes the required embedding will be broken down into the demonstration of the following four facts.

Fact 1. For any argument x, the index n and the mode (i.e., case) are independent of i, and $n \leqslant x$.

The independence is seen by induction on x; assume independence through argument x. If the computation on argument $x+1$ is in modes f, g, or c (Cases 1, 2, or 3) then the index and mode will clearly be independent of i at the end of the computation. If the computation on argument $x+1$ is in mode o then an examination of the two subcases of Case 4 shows that in each subcase the same space-bounded computation determines what the next index and mode will be; therefore the index and mode will be independent of i at the end of the computation. If the computation on argument $x+1$ is in mode d then an examination of Case 5 shows that the determination of whether to change the index and mode is independent of i. That $n \leqslant x$ is obvious.

Fact 2. As the argument x increases, the index n increases without bound. Therefore $\mathbf{C}(f) \subset \mathbf{C}(h_i) \subset \mathbf{C}(g)$ for all i.

Assume for the sake of a contradiction that the index attains a maximum. Then from the nature of the construction it follows that the index n and the mode remain fixed for all sufficiently large arguments. Assume that the final mode is f. Then $h_i = f$ a.e. and so $\mathbf{C}(h_i) = \mathbf{C}(f) \subset \mathbf{C}(g)$. Therefore there is a y such that $C_n(h_i)(y) \neq g(y)$, and since f is strictly increasing, at some sufficiently large argument the computation in Case 1 will have enough space to find such a y and the mode will be changed. If the final mode is g the argument for a contradiction is similar to that for mode f. If the final mode is c, since $f(x) + 1 < f(x+1)$ for all x, for a sufficiently large argument the computation of Case 3 will change the mode. Assume the final mode is o. By Fact 1 we have that $h_{s(n)} = g$ a.e. and $h_{t(n)} = f$ a.e. But since $\mathbf{C}(f) \subset \mathbf{C}(g)$ there is a y such that $h_{s(n)}(y) > C_j(h_{t(n)})(y)$ for all $j \leqslant n$. Since f is strictly increasing, at some sufficiently large argument the computation in Case 4 will have enough space to find such a y and the mode will be changed. If the final mode is d, then as with mode c, there will eventually be a change of mode. Therefore, in any case our assumption that the index attains a maximum leads to a contradiction. Once the index is larger than j the construction guarantees that $C_j(f) \neq h_i$ and $C_j(h_i) \neq g$ yielding $\mathbf{C}(f) \subset \mathbf{C}(h_i) \subset \mathbf{C}(g)$ for all i.

Fact 3. For all i, h_i is honest.

The functions h_i are not quite tape constructible, but $S_{h(i)} \leqslant h_i + i$. Since h_i is strictly increasing, the recapitulation of earlier computations can be done without interfering with this inequality. Cases 1, 2, and 3 pose no problems. Since whether $j \leqslant_P k$ can be determined in space $j + k$ and since $S_s < f$, the determination of which subcase of Case 4 to use can be done in space bounded by $f(x + 1) + i$. For Case 5, note that whether $f(x + 1) < h_0(x) + 1$ can be determined in space $f(x + 1)$. Therefore in any case, the inequality $S_{h(i)} \leqslant h_i + i$ is preserved.

Fact 4. For any i and j, $\mathbf{C}(h_i) \subseteq \mathbf{C}(h_j)$ if and only if $i \leqslant_P j$.

Assume that $i \leqslant_P j$; we shall show that $h_i \leqslant h_j$. If the computation is in mode f, g, or c then the two functions are equal, and if the inequality holds on entering mode d then it will be preserved while in that mode. For the case of mode o, assume that $h_i(x + 1)$ is computed in mode o and that $h_i(x) \leqslant h_j(x)$. If $s(n) \leqslant_P i$ then $s(n) \leqslant_P j$; thus if $h_i(x + 1)$ is computed by Subcase A then so is $h_j(x + 1)$, and they are equal. If $h_i(x + 1)$ is computed by Subcase B then whichever subcase is used to compute $h_j(x + 1)$ it is clear that $h_i(x + 1) \leqslant h_j(x + 1)$.

Assume that $i \nleqslant_P j$. To see that $\mathbf{C}(h_i) \nsubseteq \mathbf{C}(h_j)$ we shall show that for each k there is a y such that $h_i(y) > C_k(h_j)(y)$. Let x be large enough such that $n \geqslant k$, $s(n) = i$, $t(n) = j$, and such that during the computation of $h_i(x + 1)$ the mode changes from o to d. Then $h_i(x + 1)$ is computed by Subcase A of Case 4, and an examination of that subcase shows that there is a y such that $h_i(y) > C_k(h_j)(y)$ as claimed. This completes the proof of the theorem.

The following *naming* lemma will be needed for the proof of the next density theorem; it is proved in [8, Lemma 3.5].

LEMMA 3.3. *There is an elementary function r such that if φ_i is a total function of one argument then $\varphi_{r(i)}$ is total, and if φ_i is also honest then $\varphi_i = \varphi_{r(i)}$* a.e. *and $S_{r(i)} \in \mathbf{C}(\varphi_i)$.*

We now introduce some terminology needed to state the next density theorem. $\mathbf{O}'$ is the Turing jump of the empty set (i.e., the Turing degree of the diagonal set K), and a sequence of computable functions $f_0, f_1, f_2, \ldots$, is said to be *recursive* in $\mathbf{O}'$ if there is a function k recursive in $\mathbf{O}'$ such that $f_i = \varphi_{k(i)}$ for all i. It is a routine exercise to construct an elementary function k of two arguments such that for all i, $\lim_n k(i, n) = k(i)$ and such that for all i and n, $\varphi_{k(i,n)}$ is total. For example a Turing machine to compute $k(i, n)$ might use the space for the inputs to compute as much as possible of the diagonal set K and to try to compute $k(i)$ from that. Let j be the approximation to $k(i)$ produced by this; if no approximation to $k(i)$ is produced the machine uses the Gödel number of some fixed total Turing machine for j. In either case the machine gives as output the Gödel number of a Turing machine which simulates M_j in parallel with searching for a change in the approximation to $k(i)$, and which halts when either the simulation halts or the approximation changes.

THEOREM 3.4. *Let $f_0, f_1, f_2, \ldots,$ be a sequence of honest functions recursive in $\mathbf{O}'$ and let g be an honest function such that for all i, $\mathbf{C}(f_i) \subseteq \mathbf{C}(f_{i+1})$ and $\mathbf{C}(f_i) \subset \mathbf{C}(g)$. Then there is an honest function h such that $\mathbf{C}(f_i) \subset \mathbf{C}(h) \subset \mathbf{C}(g)$ for all i.*

Proof. Without loss of generality we may assume that g is a tape constructible function of one argument such that for all x, $g(x) + 1 < g(x + 1)$; let M_g be a Turing machine which computes g such that $S_g = g$. We begin the proof with a technical lemma on the naming of the classes $\mathbf{C}(f_i)$.

LEMMA 3.5. *There exists an elementary function p of two arguments such that for all i, $\lim_n p(i, n) = p(i)$ exists and $\mathbf{C}(S_{p(i)}) = \mathbf{C}(f_i)$, and such that if $j \leqslant i$ then for all n and x, $S_{p(j,n)}(x) \leqslant S_{p(i,n)}(x) \leqslant S_g(x)$ and $S_{p(i,n)}(x) + 1 < S_{p(i,n)}(x + 1)$.*

Proof of lemma. Let $p'(i, n)$ be such that

$$S_{p'(i,n)}(x) = \max_{j \leqslant i} 2 \sum_{y \leqslant x} (S_{r(k(j,n))}(y) + 1),$$

and let $p(i, n) = \overline{\min}(p'(i, n), g)$, where r is from Lemma 3.3, k is from the discussion following Lemma 3.3, and $\overline{\min}$ is from Proposition 2.3.

Since for all i, $\lim_n k(i, n) = k(i)$, we have that $\lim_n p'(i, n)$ exists and therefore that $\lim_n p(i, n)$ exists. Also since for all i, $\sum S_{r(k(i))} \in \mathbf{C}(f_i)$, and for all i and j, $\mathbf{C}(\max(S_i, S_j)) = \text{lub}(\mathbf{C}(S_i), \mathbf{C}(S_j))$ and $\mathbf{C}(\min(S_i, S_j)) = \text{glb}(\mathbf{C}(S_i), \mathbf{C}(S_j))$ if S_i and S_j are increasing (as was shown in [8]), it follows that $\mathbf{C}(S_{p(i)}) = \mathbf{C}(f_i)$ for all i. The inequalities are clear from the definition of the function p.

We now return to the proof of the theorem. We shall give a multitrack Turing machine M to compute h; M will use an index n, modes f, g, and c, and an e-value e. On input $x + 1$, M first recapitulates its computations on inputs $0, \ldots, x$ in order to find $h(x)$ and the current mode, index, and e-value. The index is used to insure that the construction deals properly with all of the functions f_j and all of the operators C_j, and the e-value is used as a guess at a point at which the function p may have become constant for the given index. The purpose of the modes will be explained in the following cases.

Case 1

The mode is f. This mode guarantees that $\mathbf{C}(h) \subset \mathbf{C}(g)$ by having h follow $S_{p(n,e)}$. If $p(n, e) \neq p(n, x + 1)$ then the mode and index are not changed but $x + 1$ is the new e-value and $h(x+1)$ is determined as follows. If $h(x) < S_{p(n,x+1)}(x+1)$ then $h(x+1) = S_{p(n,x+1)}(x+1)$; otherwise $h(x+1) = h(x)+1$. If $p(n, e) = p(n, x+1)$ then M marks off $S_{p(n,e)}(x+1)$ tape squares and in that space looks for a y such that $C_n(h)(y) \neq g(y)$. If such a y is discovered then the new mode is g, the new e-value is $x+1$, and the index is not changed. If no such y is discovered then the mode, index, and e-value are not changed. In either case $h(x+1) = S_{p(n,e)}(x+1)$.

Case 2

The mode is g. This mode guarantees that $\mathbf{C}(f_j) \subset \mathbf{C}(h)$ for all j by having h follow g. If $p(k, e) \neq p(k, x+1)$ for some $k \leqslant n$ then the mode and index are not changed but $x+1$ is the new e-value and $h(x+1) = g(x+1)$. If $p(k, e) = p(k, x+1)$ for all $k \leqslant n$ then M marks off $g(x+1)$ tape squares and uses that space to look for a y such that $C_j(S_{p(k,e)})(y) < h(y)$ for all $j, k \leqslant n$. If such a y is discovered then the new mode is c and the index and e-value are not changed; if no such y is found then the mode, index, and e-value are not changed. In either case $h(x+1) = g(x+1)$.

Case 3

The mode is c. This is a coasting mode. If $S_{p(n+1,x+1)}(x+1) < h(x)+1$ then $h(x+1) = h(x)+1$ and the mode, index, and e-value are not changed. Otherwise $h(x+1) = S_{p(n+1,x+1)}(x+1)$, the new mode is f, the new index is $n+1$, and the new e-value is $x+1$.

This completes the description of the computation of h. Clearly h is strictly increasing and $h \leqslant g$. Note also that the index is always less than or equal to the argument. The rest of the proof is broken down into the demonstration of the following three facts.

Fact 1. As the argument x increases, the index n and the e-value e increase without bound.

Assume for the sake of a contradiction that the index attains a maximum. Then from the nature of the construction it follows that the index n and the mode remain fixed for all sufficiently large arguments. Assume the final mode is f. Since $\lim_k p(n, k) = p(n)$ it follows that $h = S_{p(n)}$ a.e. and so $\mathbf{C}(h) = \mathbf{C}(f_n) \subset \mathbf{C}(g)$. Therefore there is a y such that $C_n(h)(y) \neq g(y)$ and since $S_{p(n)}$ is strictly increasing, at some sufficiently large argument the computation in Case 1 will have enough space to discover such a y and the mode will be changed. Assume that the final mode is g; then $h = g$ a.e. From the first computation in Case 2 it follows that the e-value e will eventually be large enough so that $p(k, e) = p(k)$ for all $k \leqslant n$. Since $\mathbf{C}(S_{p(k)}) \subset \mathbf{C}(g) = \mathbf{C}(h)$ for all k, there will be a y such that $C_j(S_{p(k)})(y) < h(y)$ for all $j, k \leqslant n$. Since g is increasing this will eventually be discovered and the mode will be changed. Assume the final mode is c. Since $S_{p(n)}(x)+1 < S_{p(n)}(x+1)$ for all n and x it follows that the computation can stay in mode c for only finitely consecutive arguments.

Thus in any case we have a contradiction. That the e-value increases without bound is now clear.

Fact 2. The function h is honest.

The function h is not quite tape constructible, but the space in which it is computed is elementary in h. In fact there is an elementary function E such that for all x, $S_h(x) \leqslant E(x) \cdot h(x)$ where S_h is the space function for h. Let E be an increasing elementary function such that the computations of p in the computation of h can be done in space E and such that $p(x, x) < E(x)$ for all x; E exists by Lemma 2.7. That S_h satisfies the inequality follows by induction on the argument x. Assume that $S_h(y) \leqslant E(y) \cdot h(y)$ for all $y \leqslant x$. Since h and E are increasing, the recapitulation in the computation of $h(x+1)$ can be done in space $E(x+1) \cdot h(x+1)$. The computations of p can certainly be done in space $E(x+1)$. The remaining computation may require the computation of some $S_{p(n,e)}(x+1)$. Since M must be able to simulate infinitely many machines $M_{p(n,e)}$, this simulation could take as much as $p(n, e) \cdot S_{p(n,e)}$ space. Since $n, e \leqslant x+1$ and $S_{p(n,e)}(x+1) \leqslant h(x+1)$, the computation can be completed in space $E(x+1) \cdot h(x+1)$.

Fact 3. For all i, $\mathbf{C}(f_i) \subset \mathbf{C}(h) \subset \mathbf{C}(g)$.

Since h is honest and $h \leqslant g$ we have $\mathbf{C}(h) \subseteq \mathbf{C}(g)$. Once the index is greater than i, $C_i(h) \neq g$. Therefore from Fact 1, $\mathbf{C}(h) \subset \mathbf{C}(g)$. If x is large enough so that the index n is greater than i and the e-value e is large enough so that $p(i, e)$ has attained its limit $p(i)$, the inequalities in Lemma 3.5 and the construction give that $S_{p(i)}(x) \leqslant h(x)$. Therefore $S_{p(i)} \leqslant h$ a.e. Also, Case 2 of the construction yields that $C_j(S_{p(i)}) \neq h$ for all $j \leqslant n$. Therefore $\mathbf{C}(f_i) = \mathbf{C}(S_{p(i)}) \subset \mathbf{C}(h)$. This completes the proof of the theorem.

A completely symmetric version of the proof of Theorem 3.3 yields the following theorem.

THEOREM 3.6. *Let f be an honest function and let $g_0, g_1, g_2, \ldots,$ be a sequence of honest functions recursive in $\mathbf{O}'$ such that for all i, $\mathbf{C}(f) \subset \mathbf{C}(g_i)$ and $\mathbf{C}(g_{i+1}) \subseteq \mathbf{C}(g_i)$. Then there is an honest function h such that $\mathbf{C}(f) \subset \mathbf{C}(h) \subset \mathbf{C}(g_i)$ for all i.*

Meyer and Ritchie [9, p. 81, open problem 7] raise the question of the existence of least upper bounds for increasing sequences of elementary honest classes. Theorem 3.4 gives a partial solution to this problem. If $f_0, f_1, f_2, \ldots,$ is a sequence of honest functions recursive in $\mathbf{O}'$ such that for all i, $\mathbf{C}(f_i) \subset \mathbf{C}(f_{i+1})$, then the sequence $\mathbf{C}(f_0), \mathbf{C}(f_1), \mathbf{C}(f_2), \ldots,$ has no minimal upper bounds among the honest $\mathbf{C}$-classes. Symmetrically, Theorem 3.6 yields that a properly decreasing sequence of honest subrecursive classes recursive in $\mathbf{O}'$ has no maximal lower bounds among the honest subrecursive classes.

A simple combination of Theorems 3.2 and 3.4 (3.6) yields that any countable partial ordering can be embedded in the gap between an increasing (decreasing) sequence of honest subrecursive classes recursive in $\mathbf{O}'$ and an honest subrecursive class properly above (below) the sequence. Moreover, an examination of the proof of Theorem 3.4 shows that the construction gives a Gödel number of the function h as an elementary function of the Gödel numbers of the function k (recursive in $\mathbf{O}'$) which defines the sequence $f_0, f_1, f_2, \ldots$, and of the function g. A like observation holds for the proof of Theorem 3.2; the nonconstructive assumption that f and g are tape constructible can be eliminated by an application of Lemma 3.3.

4. Further Results

In this section we present some further results related to Theorem 3.4. The first theorem will show that the effectiveness restriction on the sequences in Theorem 3.4 cannot be appreciably relaxed, and it will also give additional information on the least upper bounds question raised by Meyer and Ritchie. In [8] it was shown that the honest subrecursive classes are a lattice, and that every honest subrecursive class is the meet (glb) of a pair of incomparable honest subrecursive classes. The second theorem of this section will show that for every pair of honest subrecursive classes, one containing the other, there is a pair of incomparable honest subrecursive classes with the larger as their join and the smaller as their meet.

A sequence of functions $f_0, f_1, f_2, \ldots$, is said to be *recursive* in $\mathbf{O}''$ if there is a function h recursive in the Turing degree $\mathbf{O}''$ such that for all i, $f_i = \varphi_{h(i)}$; a set of functions is said to be *recursively enumerable* (r.e.) if there is an r.e. set A such that $\{\varphi_i ; i \in A\}$ is the given set of functions. The zero (honest) $\mathbf{C}$-class, $\mathbf{C}(0)$, is the $\mathbf{C}$-class of zero function; $\mathbf{C}(0)$ is the minimum (honest) $\mathbf{C}$-class.

Theorem 4.1. *Every nonzero honest* $\mathbf{C}$*-class is the least upper bound of the set of honest* $\mathbf{C}$*-classes properly contained in it. Therefore, every nonzero* $\mathbf{C}$*-class is the least upper bound of an increasing sequence of honest* $\mathbf{C}$*-classes recursive in* $\mathbf{O}''$.

Proof. It is clear that every nonzero honest $\mathbf{C}$-class is a minimal upper bound of the set of honest $\mathbf{C}$-classes properly contained in it. Since the honest C-classes are a lattice, the meet of two upper bounds is also an upper bound, and therefore any minimal upper bound is actually a least upper bound.

If φ_i is total, $\mathbf{C}(\varphi_i)$ is uniformly r.e.; thus there is a $\sum_1$ predicate $C(i, j)$ such that if φ_i is total, $\{\varphi_j : C(i, j)\} = \mathbf{C}(\varphi_i)$. Let $T(i, x, y)$ be a $\sum_1$ predicate such that $T(i, x, y)$ iff $\varphi_i(x) = y$; let $S(i, x, y)$ be a $\sum_1$ predicate such that $S(i, x, y)$ iff $S_i(x) = y$; let $U(i)$ be a $\prod_2$ predicate such that φ_i is total iff $U(i)$. Define $D(i, j)$ to be the following $\sum_2$ predicate;

$$\exists k[C(i, k) \wedge \forall x, y, z[((T(j, x, y) \wedge T(k, x, z)) \Rightarrow y = z)]];$$

if φ_i and φ_j are total, then $D(i,j)$ iff $\varphi_j \in \mathbf{C}(\varphi_i)$. Define $E(j)$ to be the following $\prod_1$ predicate.

$$\forall x, y, z[(S(j, x, y) \wedge T(j, x, z)) \Rightarrow y = z];$$

if φ_j is total, then $E(j)$ iff $\varphi_j = S_j$. Finally define $F(i,j)$ to be the following predicate.

$$U(j) \wedge D(i,j) \wedge \sim D(j,i) \wedge E(j);$$

note that $F(i,j)$ is both a $\sum_3$ and $\prod_3$ predicate; if φ_i is total, then

$$\{\varphi_j: F(i,j)\} = \{\varphi_j: \varphi_j \in \mathbf{C}(\varphi_i), \varphi_i \notin \mathbf{C}(\varphi_j), \text{ and } \varphi_j = S_j\}.$$

Suppose that φ_i is honest and define

$$f_k(x) = \max_{j \leq k}\{\varphi_j(x): F(i,j)\}$$

($\max \varnothing = 0$); then the sequence of functions $f_0, f_1, f_2, \ldots$, is recursive in $\mathbf{O}''$. If g is an honest function such that $g \in \mathbf{C}(\varphi_i)$ and $\varphi_i \notin \mathbf{C}(g)$ then there is some k such that $g \in \mathbf{C}(f_k)$. Also, for all k, f_k is an honest function such that $f_k \in \mathbf{C}(\varphi_i)$ and $\varphi_i \notin \mathbf{C}(f_k)$. Therefore $\mathbf{C}(\varphi_i)$ is the least upper bound among the honest $\mathbf{C}$-classes of the increasing sequence $\mathbf{C}(f_0), \mathbf{C}(f_1), \mathbf{C}(f_2), \ldots$, of honest $\mathbf{C}$-classes, a sequence recursive in $\mathbf{O}''$.

The proof of Theorem 4.1 is complete. Note that a completely analogous proof yields the dual theorem about greatest lower bounds of decreasing sequences of honest subrecursive classes.

In [8] it was shown that the honest subrecursive classes are a distributive lattice, and that every honest subrecursive class is the greatest lower bound (intersection) of two incomparable honest subrecursive classes. We now give a stronger "splitting" property.

THEOREM 4.2. *Let f and g be honest functions such that $\mathbf{C}(f) \subset \mathbf{C}(g)$. There are honest functions h_0 and h_1 such that $\mathbf{C}(h_0)$ and $\mathbf{C}(h_1)$ are incomparable, and such that $\mathbf{C}(f)$ is the greatest lower bound of $\mathbf{C}(h_0)$ and $\mathbf{C}(h_1)$ and $\mathbf{C}(g)$ is the least upper bound of $\mathbf{C}(h_0)$ and $\mathbf{C}(h_1)$.*

Proof. In [8] it was shown that if h_0 and h_1 are strictly increasing tape constructible functions of one argument, then

$$\mathbf{C}(\min(h_0, h_1)) = \text{glb}(\mathbf{C}(h_0), \mathbf{C}(h_1))$$

and

$$\mathbf{C}(\max(h_0, h_1)) = \text{lub}(\mathbf{C}(h_0), \mathbf{C}(h_1)).$$

Therefore we could set our goal to be the construction of h_0 and h_1 such that $f = \min(h_0, h_1)$ and $g = \max(h_0, h_1)$. However, this might not be possible to do, keeping

h_0 and h_1 increasing. Thus we shall settle for constructing h_0 and h_1 such that $f = \min(h_0, h_1)$ and $g \in \mathrm{C}(\max(h_0, h_1))$. Since this proof uses the same techniques that were used in the proofs of Theorems 3.2 and 3.4, this proof will only be sketched.

Without loss of generality, we may assume that f and g are tape constructible functions of one argument such that for all x, $f(x) < g(x)$ and $f(x) + 1 < f(x + 1)$. For $i = 0, 1$, we describe a Turing machine M_i to compute h_i. On input $x + 1$, M_i first recapitulates its computations on inputs $0, \ldots, x$ to find $h_i(x)$ and the current index (n) and mode (f, g, c, d). Note that M_0 and M_1 will have the same mode and index at each argument. The rest of the computation is given in cases.

Case 1

The current mode is f. Then $h_0(x + 1) = f(x + 1)$ and $h_1(x + 1) = g(x + 1)$. In space $f(x + 1)$, each machine M_i looks for a $y \leqslant x + 1$ such that $h_1(y) \neq C_n(h_0)(y)$. If no such y is found then the mode and index are unchanged. If such a y is found then the new mode is c and the index is unchanged.

Case 2

The current mode is c. Then $h_0(x + 1) = f(x + 1)$ and $h_1(x + 1) = \max(f(x + 1), h_1(x) + 1)$. If $f(x + 1) < h_1(x + 1)$ then the mode and index are unchanged. If $f(x + 1) = h_1(x + 1)$ then the new mode is g and the index is unchanged.

Case 3

The current mode is g. Then $h_1(x + 1) = f(x + 1)$ and $h_0(x + 1) = g(x + 1)$. In space $f(x + 1)$, each machine M_i looks for a $y \leqslant x + 1$ such that $h_0(y) \neq C_n(h_1)(y)$. If no such y is found then the mode and index are unchanged. If such a y is found then the new mode in d and the index is unchanged.

Case 4

The current mode is d. Then $h_1(x + 1) = f(x + 1)$ and $h_0(x + 1) = \max(f(x + 1), h_0(x) + 1)$. If $f(x + 1) < h_0(x + 1)$ then the mode and index are unchanged. If $f(x + 1) = h_0(x + 1)$ then the new mode is f and the new index is $n + 1$.

From this construction we have that each h_i is a strictly increasing tape constructible function of one argument, and that $f = \min(h_0, h_1)$. The construction will stay in any given mode for only finitely many consecutive arguments, thus $\mathrm{C}(h_0)$ and $\mathrm{C}(h_1)$ are incomparable. Let $M(x) = \max(h_0(x), h_1(x))$. For infinitely many arguments x, $M(x) = g(x)$; and $M(x) \neq g(x)$ only during the coasting modes c and d. Thus for any x, let $y \leqslant x$ be largest such that $M(y) = g(y)$. Then there is a z, $x < z \leqslant g(y) \leqslant M(x)$ such that $g(x) \leqslant g(z) = M(z)$. Therefore, for all x, $g(x) \leqslant M(M(x))$. Thus $g \in \mathbf{C}(\max(h_0, h_1))$, and the proof of the theorem is complete.

References

1. Paul Axt, On a subrecursive hierarchy and primitive recursive degrees, *Trans. Amer. Math. Soc.* **92** (1959), 85–105.
2. Manuel Blum, Machine-independent theory of the complexity of recursive functions, *J. Assoc. Comput. Mach.* **14** (1967), 322–336.
3. Alan Cobham, The intrinsic computational difficulty of functions, *in* "Proc. 1964 Intern. Cong. Logic, Methodology, and Phil. Science" (Y. Bar-Hillel, Ed.) 1964, pp. 24–30.
4. Martin Davis, "Computability and Unsolvability," McGraw-Hill, New York, 1958.
5. A. Grzegorczyk, Some classes of recursive functions, *Rozprawy Matematcyzne* (1953), 1–45.
6. Hopcroft and Ullman, "Formal Languages and Their Relation to Automata," Addison-Wesley, Reading, MA, 1969.
7. Michael Machtey, Augmented loop languages and classes of computable functions, *J. Comput. System Sci.* **6** (1972), 603–624.
8. Michael Machtey, The honest subrecursive classes are a lattice, *Information and Control* **24** (1974), 247–263.
9. Albert Meyer and Dennis Ritchie, A classification of the recursive functions, *Z. Math. Logik Grundlagen Math.* **18** (1972), 71–82.
10. A. Mostowski, Über gewisse universelle Relationen, Polskiego Tow. *Matematycznego* **17** (1938), 117–118.
11. Rózsa Péter, "Recursive Functions," Academic Press, New York, 1967.
12. Robert W. Ritchie, Classes of predictably computable functions, *Trans. Amer. Math. Soc.* **106** (1963), 139–173.
13. Hartley Rogers, Jr., "Theory of Recursive Functions and Effective Computability," McGraw-Hill, New York, 1967.
14. Hartley Rogers Jr., Gödel numberings of partial recursive functions, *J. Symbolic Logic* **23** (1958), 331–341.