CLASSES OF COMPUTABLE FUNCTIONS DEFINED BY BOUNDS ON COMPUTATION: PRELIMINARY REPORT*

E. M. McCreight and A. R. Meyer
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa.

Abstract

The structure of the functions computable in time or space bounded by t is investigated for recursive functions t. The t-computable classes are shown to be closed under increasing recursively enumerable unions; as a corollary the primitive recursive functions are shown to equal the t-computable functions for a certain recursive t. Any countable partial order can be isomorphically embedded in the family of t-computable classes partially ordered by set inclusion. For any recursive t, there is a recursive t' which is (approximately) equal to an actual running time such that the t-computable functions equal the t'-computable functions.

1. Introduction

A rich structure is imposed on the computable functions by classifying functions according to the amount of time or space required to compute them. Following the axiomatic approach of Blum, it is possible to investigate this structure independently of particular notions of time or space arising from specific models of automata.

This paper contains three principal results. Theorem (5.2) establishes that any countable partial order can be isomorphically embedded in the ordering of the computable functions determined by bounds on computation. The Union Theorem (5.5) indicates that the classes of functions computable within time t, for recursive functions t, are closed under increasing recursively enumerable unions. As a consequence we conclude that the primitive recursive functions, the multiply recursive functions, the classes of the Grzegorczyk hierarchy, and several similar classes of recursive functions are in fact special cases of t-computable classes for appropriate recursive t and the familiar measure of Turing machine time or space. Finally, theorem (6.8) shows that the class of functions computable in time t can, for any recursive t, also be defined as the class of functions computable in time t', where t' is _almost_ the running time of a program. Other results in section 6 show that whether t' can equal a running time depends on the particular time measure chosen. The proof of theorem (6.8) is of independent interest as one of the first examples of a non-trivial priority construction applied to a theorem about computational complexity.

2. Preliminaries

$\mathcal{N}$ is the set of nonnegative integers. $\mathcal{P}_n$ is the set of partial recursive functions of n variables, and $\mathcal{R}_n$ is the set of (total) recursive functions of n variables. "Function" in this paper means function from $\mathcal{N} \times \mathcal{N} \times \cdots \times \mathcal{N}$ to $\mathcal{N}$.

The abbreviations "r.e.", "a.e.", and "i.o." are used for "recursively enumerable", "almost everywhere", and "infinitely often", respectively. If P(x) is a statement containing the variable x then $\lambda x[P(x)]$ is a predicate of one variable on $\mathcal{N}$. The λ-notation is also used for functions. The statement "$\lambda x[P(x)]$ (a.e.)" means that P(x) is true for all but finitely many $x \in \mathcal{N}$. Similarly, "$\lambda x[P(x)]$ (i.o.)" means that P(x) is true for infinitely many $x \in \mathcal{N}$. The phrase "for sufficiently large functions f..." means "there is a $b \in \mathcal{R}_1$ such that for all $f \geq b$ (a.e.)...". Similarly, "for arbitrarily large functions f..." means "for every $b \in \mathcal{R}_1$, there is an $f \geq b$ (a.e.) ...".

*The results reported here represent a portion of the first author's doctoral dissertation at Carnegie-Mellon University under the supervision of the second author. This research was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense under Contract No. F 44620-67-C-0058 and is monitored by the Air Force Office of Scientific Research, and in part by the Fannie and John Hertz Foundation.

The function $\varphi_i \in \mathcal{P}_1$ is the ith partial recursive function in a standard enumeration of $\mathcal{P}_1$. A Blum measure $\Phi = \{\Phi_0, \Phi_1, \ldots\}$ is a sequence of functions in $\mathcal{P}_1$ satisfying two axioms:

1. domain (φ_i) = domain (Φ_i) for all $i \in \mathbb{N}$, and
2. $\lambda i,x,y[\Phi_i(x) = y]$ is a recursive predicate.

Intuitively, $\Phi_i(x)$ represents the amount of time or space used by program number i when it finally halts after receiving input x. If $\Phi_i(x)$ is undefined, the statement "$\Phi_i(x) \geq y$" is true by convention for any $y \in \mathbb{N}$ or if y is undefined. We assume the reader is at least cursorily familiar with the basic paper of Blum , and with the papers of Hartmanis and Stearns on time and space bounded Turing machines. $T = \{T_0, T_1, \ldots\}$ is taken to be the tape squares measure on Turing machines which have separate input, output, and storage tapes; $T_i(x)$ = the number of tape squares visited on the storage tape by Turing machine i when it finally halts on input x. The input tape is taken to be a read-only tape, and the output tape is a write-only tape. This ensures that non-trivial computations must occur on the storage tape, and that T satisfies the Blum axioms.

As an aid in exposition, we sometimes informally assert that two functions t and t' are "approximately equal". This means that there is a fixed $g \in \mathcal{R}_2$ (determined by context and independent of t and t') such that $t \leq \lambda x[g(t'(x),x)]$ (a.e.) and $t' \leq \lambda x[g(t(x),x)]$ (a.e.).

3. Recursive Enumerability

The basic objects of study are the set of programs whose measures are bounded by functions $t: \mathbb{N} \to \mathbb{N}$, and the functions computed by these programs. Whether a program's measure is t-bounded depends on the measure.

(3.1) Definition. Let $t: \mathbb{N} \to \mathbb{N}$ be a function, and Φ a Blum measure. The set of programs (or more precisely, indices) t-bounded with respect to Φ is

$$F(\Phi,t) \overset{\text{def.}}{=} \{i \in \mathbb{N} \mid \varphi_i \in \mathcal{R}_1 \text{ and } \Phi_i \leq t \text{ (a.e.)}\}.$$

The set of functions t-computable with respect to Φ is

$$\mathfrak{F}(\Phi,t) \overset{\text{def.}}{=} \{\varphi_i \in \mathcal{R}_1 | \Phi_i \leq t \text{ (a.e.)}\} = \{\varphi_i | i \in F(\Phi,t)\}.$$

(3.2) Remark. For the Turing machine tape-measure T with separate input and output tapes, a function is t-computable iff there is a program which computes the function and which uses at most t(x) tape squares for all (rather than almost all) inputs x. On the other hand, the Turing machine number-of-steps measure has the property that a constant function, for example, much larger than t(0) cannot be computed in t steps at all arguments, because printing out the value of the function at argument zero requires more than t(0) steps. Hence, the "almost everywhere" condition is included in the definition of the F and $\mathfrak{F}$ classes.

(3.3) Remark. The requirement that $\varphi_i \in \mathcal{R}_1$ in Definition 3.1 is not redundant, since the condition $\Phi_i \leq t$ (a.e.) allows Φ_i (and hence φ_i) to be undefined finitely often. The results of this paper also apply with $\{i | \Phi_i \leq t \text{ (a.e.)}\}$ in place of $F(\Phi,t)$, but it is convenient to restrict attention to total functions.

Hartmanis and Stearns observe that it is undecidable whether a recursive function is t-computable for any given sufficiently large $t \in \mathcal{R}_1$. More strongly, Young has pointed out that a standard reduction to the halting problem implies

(3.4) Fact. For any Blum measure Φ and any function $t \in \mathcal{R}_1$, $\mathfrak{F}(\Phi,t) \neq \emptyset \Rightarrow \{i | \varphi_i \in \mathfrak{F}(\Phi,t)\}$ is not r.e.

Essentially the same reduction argument yields.

(3.5) Fact. For any Blum measure Φ, and all sufficiently large $t \in \mathcal{R}_1$, $F(\Phi,t)$ is not r.e.

An elegant generalization of fact (3.5) appears in Blum .

The observation that $\mathfrak{F}(\Phi,t)$ is an r.e. set of functions for sufficiently large $t \in \mathcal{R}_1$ is implicit in Hartmanis and Stearns . Moreover, when $t = \Phi_i$ for some $\Phi_i \in \mathcal{R}_1$, their proof yields an enumeration of $\mathcal{F}(\Phi,t)$ by programs which run in (approximately) bound t. Young asks whether this is true for arbitrary recursive t. The following theorem gives an affirmative partial answer.

(3.6) Definition. A sequence of (partial) functions $f_0, f_1, \ldots$ is recursively enumerable (r.e.) iff $\lambda i,x[f_i(x)] \in \mathcal{P}_2$. A set $\mathcal{A}$ of functions is r.e. iff $\mathcal{A} = \{f_i | i \in \mathcal{N}\}$ for some r.e. sequence of functions $f_0, f_1, \ldots$.

The proof of the following useful fact is left as an amusing exercise for the reader.

(3.7) Fact. A set $\mathcal{A} \subset \mathcal{P}_1$ is r.e. $\Leftrightarrow$ $\mathcal{A} = \{\varphi_i | i \in W\}$ for some r.e. set $W \subset \mathcal{N}$ $\Leftrightarrow$ $\mathcal{A} = \{\varphi_i | i \in R\}$ for some recursive set $R \subset \mathcal{N}$. If $\mathcal{A} \subset \mathcal{R}_1$ is r.e., then for any Blum measure Φ, and all sufficiently large $t \in \mathcal{R}_1$, $\mathcal{A} \subset \mathcal{F}(\Phi,t)$.

(3.8) Theorem. For any Blum measure Φ, there is a $g \in \mathcal{R}_2$ such that: for all sufficiently large $t \in \mathcal{R}_1$, there is an r.e. set $W \subset F(\Phi,\lambda x[g(t(x),x)])$ such that $\mathcal{F}(\Phi,t) = \{\varphi_i | i \in W\}$.

Sketch of the proof: The object is to enumerate a sequence of programs computing all and only the t-computable functions such that the programs enumerated actually have measures not much larger than t. We shall indicate how to do this for the tape measure T. The ith program in the enumeration is defined by the following instructions, "Given input x, try to compute t(0) within x tape squares. If this is possible, try to compute $\varphi_i(0)$ within any of the x tape squares remaining. If this is possible, try to compute t(1), then $\varphi_i(1)$, etc., continuing in this manner until the x tape squares are exhausted. If at any point this process turns up an argument y at which the computation of $\varphi_i(y)$ loops or requires more than t(y) tape squares, halt and give output zero. Otherwise try simultaneously computing $\varphi_i(x)$ and t(x) without at any point using more tape squares for $\varphi_i(x)$ than have been required for t(x) up to that point. If $\varphi_i(x)$ can be computed in the alloted space, halt and give output $\varphi_i(x)$. Otherwise (if $\varphi_i(x)$ loops or requires more space than did t(x)), halt and give output zero."

Clearly, if $T_i \leq t$, the preceding instructions compute φ_i. Also, if $T_i(y) > t(y)$ for some y, this will be discovered by the program on all large inputs, and the program will compute a function equal to zero (a.e.). Observing that the functions equal to zero (a.e.) can be computed in space zero, it follows from remark (3.2) that the programs above enumerate the t-computable functions. Moreover, each program uses at most $t(x) + x$ tape squares for almost all inputs x. $\square$

(3.9) Corollary. (Hartmanis-Stearns, Young) For any Blum measure Φ, and all sufficiently large $t \in \mathcal{R}_1$, $\mathcal{F}(\Phi,t)$ is an r.e. set of functions.

(3.10) Remark. Given any Φ, it is certainly not the case that all r.e. sets of recursive functions are equal to $\mathcal{F}(\Phi,t)$ for some $t \in \mathcal{R}_1$. For example, let $f,g \in \mathcal{R}_1$ be such that g is harder to compute than f, viz., $g \in \mathcal{F}(\Phi,t) \Rightarrow f \in \mathcal{F}(\Phi,t)$. Then for any $t \in \mathcal{R}_1$ larger than the measure of some program for g, the r.e. set $\mathcal{F}(\Phi,t) - \{f\}$ contains g but not f, and hence cannot be obtained. However, given any r.e. set of recursive functions, the Blum axioms are weak enough that we have

(3.11) Fact. Let $\mathcal{A} \subset \mathcal{R}_1$ be an r.e. set of functions. There is a Blum measure Φ such that $\mathcal{A} = \mathcal{F}(\Phi,\lambda x[0])$.

(3.12) Remark. Thus far we have restricted attention to $t \in \mathcal{R}_1$. From corollary (3.9) it follows immediately that $\mathcal{R}_1 \neq \mathcal{F}(\Phi,t)$ for any Blum measure Φ or $t \in \mathcal{R}_1$, since $\mathcal{R}_1$ is well-known not to be r.e. On the other hand, if we choose a (non-recursive) t which majorizes all recursive functions (e.g. Rado's function), then $\mathcal{R}_1 = \mathcal{F}(\Phi,t)$.

The preceding remark indicates that non-recursive functions t can lead to new classes. This remains true even if we require that t be bounded by a recursive function.

(3.13) Theorem. For every Blum measure Φ, there is a $b \in \mathcal{R}_1$ such that there are uncountably many sets $\mathcal{F}(\Phi,t)$ as t ranges over the (non-recursive) functions bounded above by b.

<u>Sketch of the proof</u>: Construct an infinite sequence $A_0, A_1, \ldots$ of disjoint, infinite recursive sets such that $\lambda i,x[x \in A_i]$ is a recursive predicate. Define $f_i(x) = \underline{huge}(x) \cdot C_{A_i}(x)$ where $\underline{huge} \in \mathcal{R}_1$ is very rapidly increasing, and $C_{A_i} \in \mathcal{R}_1$ is the characteristic function of A_i. The sequence $f_0, f_1, \ldots$ is r.e., and by Fact (3.7) is contained in the b-computable functions for some $b \in \mathcal{R}_1$. Let $d \in \mathcal{R}_1$ be a function which for each i exceeds the measure of some program for C_{A_i} (again d exists since $C_{A_0}, C_{A_i}, \ldots$ is an r.e. sequence of functions).

For $x \notin A_i$, we have $f_i(x) = 0$, and moreover $f_i(x)$ can be computed in roughly d(x) steps for almost all such x. On the other hand, for $x \in A_i$, $f_i(x) = \underline{huge}(x)$ and by choosing <u>huge</u> large enough, it can be guaranteed that $f_i(x)$ requires much more than d(x) steps to compute for almost all such x.

Now choose any $S \subset \mathbb{N}$, and let

$$t_S(x) = \begin{cases} b(x) & \text{if } x \in A_i \text{ for some } i \in S \\ d(x) & \text{otherwise.} \end{cases}$$

It follows that those functions f_i which are t_S-computable are precisely $\{f_i | i \in S\}$. Since there are uncountably many sets $S \subset \mathbb{N}$, there are uncountably many distinct classes of t_S-computable functions. □

(3.14) <u>Corollary</u>. For every Blum measure Φ, there is a (non-recursive) function t which is bounded above by a recursive function such that $\mathcal{F}(\Phi,t)$ is <u>not</u> an r.e. set of functions.

<u>Proof</u>: There are only countably many r.e. sets of functions. □

4. <u>Invariance</u>.

The Blum axioms are intentionally weak, the point being that any theorem following from the axioms applies to all notions of time or space arising from any conceivable machine model. Fact (3.11) indicates that whether an r.e. set of recursive functions is a class of t-computable functions varies with the measure. The following definition is useful in considering dependence on the measure.

(4.1) <u>Definition</u>. Let Φ and Φ' be Blum measures. Φ' is a <u>refinement</u> of Φ iff for all sufficiently large $t \in \mathcal{R}_1$, there is a $t' \in \mathcal{R}_1$ such that $\mathcal{F}(\Phi,t) = \mathcal{F}(\Phi',t')$.

In short, measures which are refinements of each other define the same classes of t-computable functions for large enough $t \in \mathcal{R}_1$. Refinement is obviously a transitive relation on measures. This definition allows one to ignore some of the pathologies illustrated by fact (3.11), and it is not obvious that there are measures which are not refinements of each other. This question was answered for us by Manuel Blum.

(4.2) <u>Theorem (Blum)</u>. Given any Blum measure Φ, one can construct a Blum measure Φ' such that neither Φ nor Φ' is a refinement of the other.

<u>Sketch of the proof</u>: By a minor modification of the proof of Blum's compression theorem , one can show that in any given measure Φ there are arbitrarily complicated pairs of functions $c,d \in \mathcal{R}_1$ such that range(c) = {0,1}, range(d) = {2,3}, and d is <u>slightly</u> harder to compute than c.

Define a new measure Φ' as follows:

$$\Phi'_i(x) = \begin{cases} \Phi_i(x) & \text{if } \Phi_i(x) \text{ is undefined, or if} \\ & \varphi_i(x) \notin \{0,1\}, \\ h(\Phi_i(x),x) & \text{otherwise.} \end{cases}$$

By choosing (independently of c and d) $h \in \mathcal{R}_2$ to grow sufficiently fast, one can guarantee that c is harder to compute than d in the Φ' measure. Hence neither Φ nor Φ' is a refinement of the other. □

Theorem (4.2) indicates that additional restrictions must be placed on the notion of measure before a measure-invariant notion of t-computable classes can be obtained. Additional axioms which restrict measures to more accurately reflect properties of time or space arising from familiar machine models have been considered by Borodin and Young . However, even without further restrictions, we can assert that the t-computable classes of different measures interlace in a regular manner.

(4.3) Theorem. For any Blum measures Φ and Φ', there is a $g \in \mathcal{R}_2$ such that for all functions $t: \mathbb{N} \to \mathbb{N}$,

$$F(\Phi,t) \subset F(\Phi',\lambda x[g(t(x),x)]) \text{ and}$$
$$F(\Phi',t) \subset F(\Phi,\lambda x[g(t(x),x)]).$$

Proof: By a theorem of Blum , for any Φ,Φ', there is a $g \in \mathcal{R}_2$ such that for every $i \in \mathbb{N}$, $\Phi_i \le \lambda x[g(\Phi_i'(x),x)]$ (a.e.) and $\Phi_i' \le \lambda x[g(\Phi_i(x),x)]$ (a.e.). The theorem follows if we assume (without loss of generality) that g is non-decreasing. □

5. Set Theoretic Structure

For $f,g \in \mathcal{R}_1$, the notion that f is as complicated (hard to compute) as g can be expressed by saying that for every program computing f, there is an almost everywhere faster program for g.

(5.0) Remark. An apparently more natural formulation might be that no program for f is faster than the "fastest" program for g. In view of Blum's Speed-up theorem , however, there may not be a "fastest" program for g.

The relation "as complicated as" on $\mathcal{R}_1$ was proposed by Rabin , who observed that it is transitive and leads to a partial order on the equivalence classes of equally complex computable functions. This partial order can also be considered in terms of set inclusion among the classes $\mathfrak{F}(t)$* since f is as complicated as g iff $(\forall t \in \mathcal{R}_1)[f \in \mathfrak{F}(t) \Rightarrow g \in \mathfrak{F}(t)]$. The fact that the $\mathfrak{F}$ classes are actually partially, rather than totally ordered follows from the fact that the $\mathfrak{F}$ classes are not closed under union (though they are trivially closed under intersection).

(5.1) Theorem. For all sufficiently large $t \in \mathcal{R}_1$, there is a $t' \in \mathcal{R}_1$ such that $\mathfrak{F}(t) \cup \mathfrak{F}(t') \ne \mathfrak{F}(f)$ for any function $f: \mathbb{N} \to \mathbb{N}$.

The proof, which we omit, resembles the proof by Hartmanis and Stearns of a similar theorem for Turing machine measures.

The partial order of the t-computability classes is extremely complicated, as is clear from:

(5.2) Theorem. Any countable partial order is isomorphic to some family $\mathcal{H}$ of t-computability $(t \in \mathcal{R}_1)$ classes partially ordered under set inclusion. Moreover, there is a $g \in \mathcal{R}_2$ (depending only on the measure) such that for arbitrarily large $h \in \mathcal{R}_1$, the family $\mathcal{H}$ can be chosen as a suborder of
$\{\mathfrak{F}(t) | t \in \mathcal{R}_1, h \le t \le \lambda x[g(h(x),x)](\text{a.e.})\}$.

The lengthy proof, which we omit, is based on a theorem of Mostowski asserting that there is a recursive partial order on $\mathbb{N}$ into which any countable partial order can be isomorphically embedded, and a generalization of recursion theorem construction which appears in Meyer and Fischer . Actually a stronger result is proved:

(5.3) Corollary. The family $\mathcal{H}$ of Theorem (5.2) can be chosen to satisfy the additional conditions that for all $\mathfrak{F}(t_1), \mathfrak{F}(t_2) \in \mathcal{H}$:

(5.3.1) $\mathfrak{F}(t_1) \supset \mathfrak{F}(t_2) \Rightarrow (\exists c \in \mathfrak{F}(t_1))[\varphi_i = c \Rightarrow \Phi_i > t_2 \text{ (a.e.) and range}(c) = \{0,1\}]$, and

(5.3.2) $\mathfrak{F}(t_1) \not\subset \mathfrak{F}(t_2) \Rightarrow (\exists c \in \mathfrak{F}(t_1) - \mathfrak{F}(t_2))[\text{range}(c) = \{0,1\}]$.

We now establish a powerful closure property of the t-computable classes which has some surprising corollaries.

(5.4) Definition. A set $\mathcal{S}$ of total functions from $\mathbb{N}$ to $\mathbb{N}$ is self-bounded iff for every finite subset $\mathcal{S}_0 \subset \mathcal{S}$ there is a $t \in \mathcal{S}$ such that $t \ge \lambda x[\underline{\max}\{f(x) | f \in \mathcal{S}_0\}]$ (a.e.).

(5.5) Union Theorem. Let $\mathcal{S} \subset \mathcal{R}_1$ be an r.e. self-bounded set of functions. There is a $t \in \mathcal{R}_1$ such that $\bigcup_{f \in \mathcal{S}} F(f) = F(t)$.

Proof: Say $\mathcal{S} = \{f_0, f_1, \ldots\}$ and $\lambda i,x[f_i(x)] \in \mathcal{R}_2$. Observe that $f_0, \underline{\max}\{f_0,f_1\}, \underline{\max}\{f_0,f_1,f_2\}, \ldots$ is a non-decreasing r.e. sequence which defines the same union as $\mathcal{S}$. Hence, we assume without loss of generality that $i \ge j \Rightarrow f_i \ge f_j$ (everywhere).

* The results in this section are measure independent, so we suppress mention of Φ.

The function t is computed in stages. Certain variables "guess(0)", "guess(1)",... are given values during the computation, and these variables are assumed to have the same values from stage to stage unless they are explicitly set to new values. The computation begins at stage zero. Stage x: "Set guess(x) = x. Find $\{i \leq x | \Phi_i(x) > f_{\text{guess}(i)}(x)\}$; call this set A(x). If $A(x) = \emptyset$, define $t(x) = f_x(x)$ and go to stage x+1. Otherwise, define
$t(x) = \underline{\min}\{f_{\text{guess}(i)}(x) | i \in A(x)\}$, set guess(i)=x for all $i \in A(x)$, and go to stage x+1."

We leave as a difficult exercise the proof of the claim that for all i, $\Phi_i \leq t$ (a.e.) $\Leftrightarrow$ $(\exists n,k \in \mathbb{N})$[guess(i)=k at every stage after stage n and $\Phi_i \leq f_k$ (a.e.)] $\Leftrightarrow$ $(\exists k \in \mathbb{N})[\Phi_i \leq f_k$ (a.e.)]. Hence $i \in F(t) \Leftrightarrow i \in F(f_k)$ for some $f_k \in \mathcal{J}$. □

(5.6) Remark. With the proof complete, it follows trivially that Theorem (5.5) remains true with "F" replaced by "$\mathcal{F}$".

(5.7) Corollary. There is a $t \in \mathcal{R}_1$ such that the set of primitive recursive functions (of one argument) is precisely the set of t-computable functions with respect to both the Turing machine space measure and the Turing machine time measure.

Proof: By theorems of R. W. Ritchie and Cobham , a function $p \in \mathcal{R}_1$ is primitive recursive iff some Turing machine computing p uses time or space bounded by a primitive recursive function. Since the primitive recursive functions of one argument are on r.e. self-bounded set of recursive functions, the corollary follows from remark (5.6). We let the reader convince himself that the same function t works for time and space. □

(5.8) Corollary. Same as corollary (5.7) with "primitive recursive" replaced by "elementary functions of Kalmar", "$\mathcal{E}^n$ functions of Grzegorczyk for each $n \geq 3$", "n-fold recursive functions of Péter for each $n \geq 1$", and "multiply recursive functions of Péter".

Proof: Each of these classes are well-known to be r.e. self-bounded sets of recursive functions. Meyer and D. Ritchie , and D. Ritchie observe that these classes also satisfy the R. W. Ritchie-Cobham property that a function is in the class iff it can be completed in time or space bounded by a function in the class. □

(5.9) Remark. Corollary (5.7) also applies to the Grzegorczyk class $\mathcal{E}^2$ and the R. W. Ritchie classes of predictably computable functions providing that only the Turing machine space measure is used.

(5.10) Remark. The function t of Corollary (5.7) must majorize (exceed almost everywhere) every primitive recursive function, and hence cannot itself be primitive recursive. Therefore, t cannot be t-computable in the Turing machine time or space measure. This means that t must grow considerably more slowly than Ackerman's function, for example, which also majorizes the primitive recursive functions. By remark (5.13) below, t can be made non-decreasing.

Borodin has observed that the minimal growth rate theorems for Turing machine time and space can be extended to a wide class of Blum measures, but not all Blum measures. He requires that a minimal growth rate be non-decreasing. By relaxing this condition we can obtain a somewhat weaker result which however applies to all measures.

(5.11) Definition. A weak minimal growth rate is a function $t: \mathbb{N} \to \mathbb{N}$ such that $\liminf_{x \to \infty} t(x) = \infty$ and for all $i \in \mathbb{N}$, $\Phi_i \leq t$ (a.e.) $\Leftrightarrow$ $(\exists k \in \mathbb{N})[\Phi_i \leq k$ (a.e.)].

(5.12) Corollary. There exists a recursive weak minimal growth rate.

Proof. The constant functions are an r.e. self-bounded set of recursive functions. By the union theorem, there is a $t \in \mathcal{R}_1$ such that the t-bounded programs are precisely the constant-bounded programs. That $\liminf t = \infty$ follows from the proof of the union theorem. □

(5.13) Remark. Borodin's construction of Blum measures in which non-decreasing recursive weak minimal growth rates do not exist suggests the following question: if $f_0, f_1, \ldots$ is an r.e. self-bounded sequence of non-decreasing recursive functions, is $\bigcup_k F(f_k) = F(t)$ for some non-decreasing $t \in \mathcal{R}_1$? A modification of the proof of the union theorem shows that the answer is yes providing that any one of $f_0, f_1, \ldots$ is unbounded. Thus Borodin's counter-example cannot be extended beyond the constant functions.

Another corollary is an amusing observation (made independently of theorem (5.5)) by Blum about non-deterministic space-bounded Turing machines. An important open problem in automata theory is whether the languages recognizable by nondeterministic linear space bounded Turing machines properly contain those languages recognized by deterministic linear space bounded by Turing machines. For certain peculiar space bounds the question can be settled trivially.

(5.14) Corollary (Blum). There are arbitrarily large $t \in \mathcal{R}_1$ such that a language is recognizable within space t on a nondeterministic Turing machine iff it is recognizable within space t on a deterministic Turing machine.

Sketch of the proof: There is a strictly increasing $g \in \mathcal{R}_1$ such that any language recognizable within space $f \in \mathcal{R}_1$ on a nondeterministic machine can be recognized within space $g \circ f$ on a deterministic machine. The function g essentially describes the extra space required to simulate the apparently more powerful nondeterministic machines with deterministic ones. Then f, $g \circ f$, $g \circ g \circ f, \ldots$ is an r.e. self-bounded sequence, and it is immediate that a language recognizable nondeterministically in space bounded by a function in the sequence is also recognizable deterministically in space bounded by the next function in the sequence. By reformulating language recognition in terms of the computation of characteristic functions and using a space measure for nondeterministic machines, one can appeal to the union theorem to obtain the desired $t \in \mathcal{R}_1$. □

The preceding corollary sheds no light on the original problem for linear space bounded Turing machines. However, since theorems about computational complexity are sometimes criticized on the grounds that they apply only to excessively time-consuming computations, it is worth mentioning that functions t satisfying corollary (5.14) can be found which are bounded above by $\lambda x[\log_2(x)]$.

6. Well-behaved Bounds

The classes of t-computable functions determined by non-recursive functions have properties significantly different from those classes determined by recursive t (cf. Theorem (3.13)). We now ask whether the recursive functions are themselves too broad a class of bounds, and in particular whether a more tractable structure arises by considering only bounds which actually are running times. The significance of this question becomes obvious from the contrast between the following thworems:

(6.1) Weak Compression Theorem. (Blum) For every Blum measure Φ, there is a $g \in \mathcal{R}_2$ such that for every $\Phi_i \in \mathcal{R}_1$,

$$\tilde{\mathcal{F}}(\Phi, \Phi_i) \subsetneqq \tilde{\mathcal{F}}(\Phi, \lambda x[g(\Phi_i(x), x)]).$$

(6.2) Gap Theorem. (Borodin) For every Blum measure Φ, and every $g \in \mathcal{R}_2$, there are arbitrarily large $t \in \mathcal{R}_1$ such that
$F(\Phi, t) = F(\Phi, \lambda x[g(t(x), x)])$ and hence
$\tilde{\mathcal{F}}(\Phi, t) = \tilde{\mathcal{F}}(\Phi, \lambda x[g(t(x), x)])$.

(6.2a) Remark. Blum's compression theorem is much more powerful than theorem (6.1) which follows as an immediate corollary. Theorem (6.2) is also a weaker assertion than the original gap theorem proved by Borodin.

The following corollary of (6.12) below, together with the two preceding theorems, form a surprising trilogy.

(6.3) Corollary. There is a Blum measure Φ such that for every $t \in \mathcal{R}_1$, there is a $\Phi_i \in \mathcal{R}_1$ such that
$F(\Phi, t) = F(\Phi, \Phi_i)$ and hence $\tilde{\mathcal{F}}(\Phi, t) = \tilde{\mathcal{F}}(\Phi, \Phi_i)$.

These results illustrate an important point; namely, that theorems like the gap theorem which appear to be making statements about the structure of t-computability classes may only be indicating the properties of badly chosen names for the classes. The same remark applies to Blum's observation (5.14) about nondeterministic automata.

One of the essential features of running times on which theorem (6.1) depends is that the running time of a running time approximately equals the running time. Such functions are "honest" (cf. Meyer and D. Ritchie) in that their values reflect their computational complexity.

(6.4) Definition. Let Φ be a Blum measure and $g \in \mathcal{R}_2$. A function $\psi \in \mathcal{P}_1$ is g-honest with respect to Φ iff $(\exists i \in \mathbb{N})[\varphi_i = \psi$ and $\Phi_i \leq \lambda x[g(\psi(x),x)]$ (a.e.)].

The notion of honesty can also be formulated using Blum's elegant definition of a measured set.

(6.5) Definition. (Blum) A sequence $\Psi = \{\psi_0, \psi_1 \ldots\}$ of (partial) functions forms a measured set iff $\lambda i,x,y[\psi_i(x) = y]$ is a recursive predicate.

(6.6) Fact. Any Blum measure is a measured set, as is any r.e. set of functions in $\mathcal{R}_1$. A function can be a member of a measured set iff its graph is recursive.

(6.7) Fact. The set of functions which are g-honest with respect to Φ form a measured set for any $g \in \mathcal{R}_2$ and Blum measure Φ. Conversely, given any measured set and Blum measure Φ, there is a $g \in \mathcal{R}_2$ such that the set of g-honest functions with respect to Φ contains the measured set.

(6.8) Theorem. For every Blum measure Φ, there is a measured set Ψ such that: for all $t \in \mathcal{R}_1$, there is a $t' \in \Psi \cap \mathcal{R}_1$ such that $F(\Phi,t) = F(\Phi,t')$ and hence $\mathfrak{F}(\Phi,t) = \mathfrak{F}(\Phi,t')$.

The proof of (6.8) is of independent interest as one of the first examples of a priority argument in the theory of computational complexity (see also, Young , and Borodin).

Detailed sketch of the proof: Given $t \in \mathcal{R}_1$, our aim is to construct $t' \in \mathcal{R}_1$ such that for all $i \in \mathbb{N}$, $\Phi_i > t$ (i.o) $\Leftrightarrow \Phi_i > t'$ (i.o.), which means $F(\Phi,t) = F(\Phi,t')$, and also such that t' is approximately equal to the number of steps required to compute t', which ensures that t' will be in a measured set.

In order to guarantee the second condition, t' will be defined by a dovetailing procedure so that small values of t' on large arguments will generally be defined before large values of t' on smaller arguments. The difficulty in satisfying the first condition is that there generally will not be time to compute $t(y)$ when defining $t'(y)$. This difficulty is met by defining $t'(y)$ to satisfy conditions which depend on the value of t of some easily computed argument other than y. The value of $t'(y)$ must be chosen to satisfy two conflicting conditions: first, it must be larger than the number of steps taken at argument y by those programs which appear to be t-bounded, and second, it must be smaller than those programs which have been discovered to exceed t at some arguments. The conflicts are resolved by a priority mechanism described below.

The computation of t' proceeds in stages beginning at stage zero. Programs $0,1,\ldots,x$ are under consideration at any stage x and have a priority ordering which remains the same as it was at the previous stage unless it is explicitly changed. Similarly, at each stage certain programs are requesting a "pop", which means their running times exceeded t somewhere and they want their running times to pop above t' somewhere. Pop requests also remain the same from stage to stage unless they are explicitly changed.

Let $\pi_1(x)$ be some recursive function which takes all integer values infinitely often and $\pi_1(x) \leq x$.

Stage x: Part I. Assign lowest priority to program x, and request that program x not be popped. Devote x steps to a dovetailed computation of t, and see if there is a z such that $t(z)$ can be computed at this stage, but not at Part I of any earlier stage. If such z can be found in the alloted time, go to Part II.

Otherwise, simulate programs $0,1,\ldots,z$ on input z for $t(z)$ steps, and request that those programs which did not halt be popped. Go to Part II.

Part II. Let $y = \pi_1(x)$. See if $t'(y)$ has been defined at an earlier stage; if so go to stage x+1. Otherwise, determine the priorities and requests of programs 0,1,...,y at the end of stage y.

Try to find the program P with highest priority (at stage y), such that P was requesting a pop at stage y and such that none of those programs which both

(1) at stage y were requesting not to be popped, and

(2) at stage y had higher priority than P,

actually takes as many steps on input y as P takes on input y.

If such a P can be found, define $t'(y)$ to be the largest number of steps taken on input y by any of the programs satisfying (1) and (2), assign lowest priority (at this stage x) to P, request that P not be popped, and go to stage x+1.

However, the preceding attempt to find P should be stopped if either

(3) it requires more than x steps, or

(4) it requires more steps than are required to compute $t(y)$.

If condition (3) occurs first, go to stage x+1. If condition (4) occurs first, define $t'(y)$ to be the larger of $t(y)$ and the number of steps required to compute $t(y)$; go to stage x+1. END.

Stopping condition (4) guarantees that $t'(y)$ will be defined for every y. Moreover, $t'(y)$ can fail to be defined at a stage x such that $y = \pi_1(x)$ only because $t'(y)$ is too large to compute in the time alloted to stage x. Since the time alloted to stage x is bounded by a fixed (independent of t) recursive function of x, it can now be proved that the number of steps required to compute t' is approximately equal to t'. (For example, in the Turing machine case the time to compute t' will be bounded by at worst an elementary function composed with t'.)

The proof that the t-bounded and t'-bounded programs are the same is based on the claim that $\Phi_i > t$ (i.o.) $\Leftrightarrow$ the pop request of program i is changed at infinitely many stages $\Leftrightarrow \Phi_i > t'$ (i.o.). Verification of the claim requires a delicate argument which we omit for lack of space. □

The following rather artificial definition allows us to state a corollary (6.3) in a stronger form.

(6.9) Definition. Two Blum measures, Φ and Φ', are similar iff $\Phi_i = \Phi'_i$ for all i such that $\text{range}(\varphi_i) \neq \{0\}$.

(6.10) Remark. Similar measures are trivially refinements of each other. In particular, if Φ and Φ' are similar, then for all $t: \mathbb{N} \to \mathbb{N}$, $\lambda x[0] \in \mathcal{F}(\Phi,t) \cap \mathcal{F}(\Phi',t) \Rightarrow \mathcal{F}(\Phi,t) = \mathcal{F}(\Phi',t)$.

(6.11) Fact. Given any measured set Ψ and Blum measure Φ, there is a Blum measure Φ' such that Φ is similar to Φ', and $\Psi \subset \Phi'$.

We have immediately from (6.8) and (6.11):

(6.12) Corollary. Every Blum measure is similar to a measure Φ' such that for every $t \in \mathcal{R}_1$ there is a $\Phi'_i \in \mathcal{R}_1$ such that

$$\mathcal{F}(\Phi',t) = \mathcal{F}(\Phi',\Phi'_i).$$

Our final result is that corollary (6.3) does not apply to the familiar time and space measures arising from Turing machines and other automaton models. Such measures have the important property that the measure functions are $\lambda x,y[x]$-honest.

(6.13) Definition. A Blum measure Φ is proper iff $\Phi_i \in \mathcal{F}(\Phi,\Phi_i)$ for all $\Phi_i \in \mathcal{R}_1$.

(6.14) Fact. The running time of a function cannot be much smaller than the function since many steps are required to print a large value. Formally, let Φ be any Blum measure. There is a $g \in \mathcal{R}_2$ such that $(\forall j \in \mathbb{N})[\varphi_j \leq \lambda x[g(\Phi_j(x),x)](\text{a.e.})]$.

(6.15) Theorem. Let Φ be a proper Blum measure. There exist arbitarily large $t \in \mathcal{R}_1$ such that for every $\Phi_i \in \mathcal{R}_1$,

$$\mathcal{F}(\Phi,t) \neq \mathcal{F}(\Phi,\Phi_i).$$

Proof: Let $h \in \mathcal{R}_2$ be the pointwise maximum of the functions $g \in \mathcal{R}_2$ of (6.1) and (6.14), and assume without loss of generality that h is increasing. For any $f \in \mathcal{R}_1$, define $f_0 = f$, and $f_{n+1} = \lambda x[h(f_n(x),x)]$.

The sequence $f_0, f_1, \ldots,$ is an r.e. increasing sequence of recursive functions. By the union theorem, there is a $t \in \mathcal{R}_1$ such that $F(\Phi,t) = \bigcup_n F(\Phi,f_n)$.

Suppose $\mathcal{F}(\Phi,t) = \mathcal{F}(\Phi,\Phi_i)$ for some $\Phi_i \in \mathcal{R}_1$. Then $\Phi_i \in \mathcal{F}(\Phi,t)$ since Φ is a proper measure, i.e., there is a $\varphi_j = \Phi_i$ such that $\Phi_j \leq t$ (a.e.). By choice of t, we have $\Phi_j \leq f_n$ (a.e.) for some n. By (6.14) and monotonicity of h, we have
$\Phi_i = \varphi_j \leq \lambda x[h(\Phi_j(x),x)] \leq \lambda x[h(f_n(x),x)] = f_{n+1}$ (a.e.). Hence,
$\lambda x[h(\Phi_i(x),x)] \leq \lambda x[h(f_{n+1}(x),x)] = f_{n+2}$ (a.e.).
Therefore, by (6.1)
$\mathcal{F}(\Phi,\Phi_i) \subsetneq \mathcal{F}(\Phi,\lambda x[h(\Phi_i(x),x)]) \subset \mathcal{F}(\Phi,f_{n+2}) \subset \mathcal{F}(\Phi,t)$, a contradiction. □

Acknowledgments

The authors would like to express their sincere appreciation for the interest shown by Manuel Blum, who provided the proof of Theorem (4.2), by Patrick C. Fischer who suggested a significant simplification in the proof of Theorem (6.8), and by Paul Young, who proposed the sharpened form of definition (4.1). We would especially like to thank Michael J. Fischer for his invaluable help over many months in revising and simplifying this material.

References

Blum, M. A machine independent theory of computational complexity. JACM, v.14(1967), 322-336.

Blum, M. Recursive function theory and speed of computation. Can.Bull.of Math., v.9(1966), 745-750.

Borodin, A. Complexity classes of recursive functions and the existence of complexity gaps, in this volume.

Cobham, A. The intrinsic computational difficulty of functions. Proc. of 1964 Congress for Logic, Meth., and Phil. of Science, North Holland(1964).

Grzegorczyk, A. Some classes of recursive functions. Rozprawy Matematyczne, Warsaw(1953), 1-46.

Hennie, F. One-tape, off-line Turing machine computations. Information and Control, v.8 (1965), 553-578.

Hartmannis, J. and R. E. Stearns. On the computational complexity of algorithms. TAMS, v.117(1965), 285-306.

Meyer, A. R. and D. M. Ritchie. The complexity of loop programs. Proc. 22 National ACM Conf., Thompson, Washington, D. C.(1967), 465-470.

Meyer, A. R. and D. M. Ritchie. A classification of functions by computational complexity. Proc. Hawaii International Conf. on System Sciences, Univ. of Hawaii(1968), 17-19.

Meyer, A. and P. C. Fischer. On computational speed-up. Conf. Record 9th Annual Symp. on Switching and Automata, IEEE Computer group(1968), 351-355.

Mostowski, A. Über gewisse universelle relationen. Ann. Soc. Polon. Math., v.17(1938), 117-118.

Péter, R. Rekursive funktionen. Académiai Kiadó, Budapest(1951).

Rabin, M. O. Degree of difficulty of computing a function and a partial ordering of recursive sets. Technical Report No. 2, Hebrew Univ., Israel(1960).

Ritchie, D. M. Program structure and computational complexity. Doctoral dissertation, Harvard Univ.(1969).

Ritchie, R. W. Classes of predictably computable functions. TAMS, v.106(1963), 139-173.

Stearns, R. E., J. Hartmannis, and P. M. Lewis. Hierarchies of memory-limited computations. Proc. 6th Annual Symp. on Switching Circuits and Logical Design, IEEE Computer group(1965), 179-190.

Young, P. R. Toward a theory of enumerations. Conf. Rec. of 9th Annual Symp. on Switching and Automata, IEEE Computer group(1968).

Young, P. R. Speed-ups by changing the order in which sets are enumerated, in this volume.