On the Size of Programs in Subrecursive Formalisms*

Robert L. Constable

Technical Report

No. 70-58

March 1970

Department of Computer Science
Cornell University
Ithaca, New York 14850

On the Size of Programs in Subrecursive Formalisms*

Robert L. Constable

Department of Computer Science
Cornell University
Ithaca, N. Y. 14850

Abstract

This paper gives an overview of subrecursive hierarchy theory as it relates to computational complexity and applies some of the concepts to questions about the size of programs in subrecursive programming languages. The purpose is three-fold, to reveal in simple terms the workings of subrecursive hierarchies, to indicate new results in the area, and to point out ways that the fundamental ideas in hierarchy theory can lead to interesting questions about programming languages. A specific application yields new information about Blum's results on the size of programs and about the relationship between size and efficiency.

§ 1 Introduction

Consider a programming language $\mathcal{L}$ such as reference Algol or LISP capable of expressing algorithms for all partial recursive functions $\phi : \mathbb{N}^m \to \mathbb{N}$ where $\mathbb{N} = \{0,1,2,\ldots\}$. It is well-known that such languages have the capacity to express algorithms which produce astronomically large computations. In other words, $\mathcal{L}$ contains algorithms for functions whose

---

* This research was supported in part by National Science Foundation Grant GJ-579.

computation at any input would exhause all imaginable computing resources. Letting $\mathcal{R}$ denote the class of all (total) recursive functions, this fact means that the functions "actually computed" belong to subrecursive classes, $\mathcal{S} \subset \mathcal{R}$. For instance, there is reason to believe that all functions actually used in computing belong to $\mathcal{R}^1$ the class of primitive recursive functions.†

Programming languages, $\mathcal{L}_{\mathcal{S}}$, can be designed which express algorithms only for the functions in a subrecursive class $\mathcal{S}$. These languages have been based on the logicians' formalisms for special classes like $\mathcal{R}^1$. In 1965 Cleave [5] designed a language for $\mathcal{R}^1$ based on ideas in Grzegorczyk [12] (1953) while Meyer & Ritchie [18] (1965) designed another such language based on the ideas of R. M. Robinson [22] (1947) and P. Axt [2] (1963). Constable designed languages for $\mathcal{R}^n$ (defined below) based on the notion of a stack [8] and of restricted program modification [7].

The subrecursive languages have several virtues. All programs terminate so there is no "halting problem". A bound for the running time of a program can be determined from the input and syntax. The conceptual structure of programs is simpler than the structure of general recursive programs.

†One can argue that only finite functions are "actually computed". However, for reasons of mathematical application a first approximation to actual computing should allow for the computability of infinite functions such as $x+y$, $x \cdot y$, x^y, etc. See Elgot & Robinson [10] and McCarthy [16] for a discussion of this point. It is in fact one of the tasks of computing theory to discover a class or classes of functions which adequately represents the functions actually computed. The class $\mathcal{E}$ of elementary functions may be a more reasonable candidate than $\mathcal{R}^1$.

Computational efficiency is not sacrificed for these advantages. In a forthcoming article, the author and Allan Borodin [9] show there is no significant loss of computational efficiency caused by computing with certain subrecursive languages. In Cleave's language the loss is at most a constant factor c of the total running time. In the case of various modifications of the "Loop" language of Meyer & Ritchie the loss is again at most a linear factor c, and for their original language the loss is at most a square factor.

What are the disadvantages of subrecursive languages? Blum [3] has shown that program compactness is sacrificed. He defines the notion of program size axiomatically. If i is a program, let $|i|$ be its size. One valid measure of size is the <u>length</u> of a program (number of cards in the deck). Blum shows that if f is any recursive function, there is a primitive recursive function f_i whose minimum length subrecursive program, say i_0, satisfies $f(|j|) < |i_0|$ for j some general recursive program for f_i. So for $f(x) = 100 \cdot x$, there is some primitive recursive function whose shortest subrecursive program is 100 times longer than one of its general recursive programs. Furthermore, Blum shows that the computational complexity, say run-time, of j is nearly the same as that for i except on a finite set.

Blum's result seems to indicate that general recursive programming languages have a decided advantage over subrecursive languages. He argues this by saying "in order for programs to be of economical size, the programming language must be powerful enough to compute arbitrary general recursive functions".

In this paper Blum's result is examined further, and it is simply shown that there is a language for $\mathcal{E}^1$ (or for $\mathcal{R}^1$) such that any program which can be significantly compressed without drastically degrading computational efficiency must be a computationally complex program. The same results apply all through the known subrecursive hierarchies, $\mathcal{E}^\alpha$, and they apply to the interesting languages such as Meyer & Ritchie [18]. The result also shows that there is a trade-off relationship between size and computational complexity (measured without an a.e. condition). Such facts can be construed to mean that for the purposes of working in the usable levels of the elementary functions, there is a subrecursive language capable of expressing all elementary functions and there is a size measure on that language such that the usable functions cannot be significantly compressed without degrading efficiency.

These results indicate some of the uses of hierarchies. The classes $\mathcal{R}^\alpha - \mathcal{R}^1$ are not of interest because their functions will be used in computing but because they serve to measure the capabilities of languages and computing systems. Moreover, the specific principles on which the hierarchies $\mathcal{G}^\alpha$ are constructed provide a fundamental description of recursive functions. Each function $f \in \mathcal{E}^\alpha$ can be represented in the normal form, $f(\ ) = E[f_\alpha(\ )]$ where $E[\ ]$ is an elementary operator and $f_\alpha(\ )$ is an element of a sequence of functions defining a subrecursive hierarchy.† The complexity of f( ) has two components: its height represented by α and its width represented by the elementary operator E[ ] (defined below).

In the sections that follow, two specific programming languages, G and P, will be defined and used to investigate the size results and to outline the development of subrecursive hierarchies. In addition, the subject will be treated from the viewpoint of abstract computational complexity. Hopefully such a treatment provides an easily intelligible overview of the subject, laying bare the methods and open problems.

§ 2 Programming Languages

General Recursive Language.

The main results will be developed first for specific computing systems (language and machine) and later for acceptable indexings and Blum measures. The particular programming languages used are based on Shepherdson & Sturgis [26] and Cleave [5]. The language G, for General recursion, is defined as follows:

---

†The notation "f( )" is used to indicate a function when the number of arguments is unimportant and when a single letter f might be construed as an integer or an algorithm. Operators from functions to functions are denoted by E[ ]. So E[f( )](x) is the value of the image of f( ) at x.

```
<constant> ::= 0|1|...|n|...
<letter> ::= a|b|c|...|x|y|z
<Letter> ::= A|B|C|...|X|Y|Z
<variable> ::= <Letter>|<Letter>_<constant>†
<binary operator> ::= +|x
<unary operator> ::= +1|∸1
<term> ::= <variable>|<constant>|<term>
     <binary op><term>|<term><unary op>|
     (<term>)
<assignment> ::= <variable> ← <term>
<conditional> ::= if<variable> = 0  then go to  <label> else
     go to next statement.
<label> ::= <constant>|<letter>|<letter><label>
<statement> ::= <assignment>;|<conditional>;
<program> ::= <statement>|<label><statement>|<program><program>
```

The conditional is usually abbreviated to "if __ then <label>" leaving the "go to" and "go to next statement" understood. For convenience, the conditional expression is used informally. The rule is

<conditional exp.> ::= if <logical> then <term> else <term>

where <logical> informally represents a true or false statement.

Examples: The following are G-programs.

```
     Z ← 0
     S ← X
1    if S = 0 then 2
     X ← X + 1
     S ← S ∸ 1
     if Z = 0 then 1
```

```
     Z ← 0
     N ← 1
     S ← X
     K ← 1
1    if S = 0 then 2
     K ← K + 1
     N ← N · K
     S ← S ∸ 1
     if Z = 0 then 1
```

†This notation takes liberties with BNF by allowing subscripts.

The first program doubles the contents of X, the second computes X! . Notice that programs halt when they attempt to branch to an unlabeled statement. Also recall $x \dot{-} y$ = if $x < y$ then 0 else $x - y$.

It is assumed that the reader knows the semantics of such a language from sources such as [26], [10] or [25]. It is interpreted on a <u>register machine</u> (named for the fact that the computer words which are the interpretations of the variables, can be used for arithmetic directly without the intervention of special registers).

<u>Subrecursive Language</u>

The subrecursive language relies heavily on its semantics. Consider the sequence of functions defined by

Def. 2.1 $f_0(x) = x + 1$ and

$$f_{n+1}(x) = f_n^{(x)}(x)$$

where for any function $f : \mathbb{N} \to \mathbb{N}$, the <u>iterate</u> of f is defined by $f^{(0)}(x) = x$, $f^{(n+1)}(x) = f(f^{(n)}(x))$. These functions have a particularly simple structure in terms of the standard high level iterative, such as the PL/1 DO, END pair. For instance, for an arbitrary G-program π let the code

```
DO N
π
END
exit ____
```

be interpreted as

$S \leftarrow N$
1 if $S = 0$ then exit
π
$S \leftarrow S \dot{-} 1$
go to 1

where S does not appear in the program π. Thus for instance

DO N
$N \leftarrow N + 1$
END

will double the contents of N

The functions $f_n(\ )$ are computed in a canonical manner by the programs:

f_0 is $X \leftarrow X + 1$ and f_{n+1} is

DO X
f_n
END.

The syntax for the subrecursive language P is obtained by adding a "clock" to programs in G. Specifically

<clock> ::= (<constant>;<constant>)
<P-program> ::= <clock>;<G-program>.

The language P is interpreted on a J-limited register machine as defined in Cleave [5]. Briefly the machine uses a special clock register J inaccessible to the program. When the program starts executing, J is given a positive value, n, and on every step J is decreased by one until either the program halts or J reaches 0. In the later case the program halts abnormally, the output being whatever is in the output register at termination.

A P-program with clock (n,p) will start on input x with $J = f_n^{(p)}(x)$ (or $J = f_n^{(p)}(\max\{x_1,\ldots,x_n\})$ on input $\bar{x} = \langle x_1,\ldots,x_n \rangle \quad \mathbb{N}^n$).

Let $\phi_0, \phi_1,\ldots,\phi_n,\ldots$ be a standard enumeration (more generally an <u>acceptable indexing</u> in the sense of Rogers [23]) of all G-programs and $\alpha_0, \alpha_1,\ldots,\alpha_n,\ldots$ an enumeration of all P-programs. Let $\sigma\phi_i(x)$ and $\sigma\alpha_i(x)$ be the number of steps taken by ϕ_i and α_i respectively on input x. Let $\phi_i(x)\downarrow$ abbreviate the fact that ϕ_i halts on input x, then $\phi_i(x)\uparrow$ means it does not halt. So $\sigma\phi_i(x)$ = if $\phi_i(x)\downarrow$ then (number of steps) else (undefined). By convention let $\alpha_i = \langle (n_i, p_i); \beta_i \rangle$ (so that $\beta_i = \phi_j$ for some j). Then notice $\sigma\alpha_i(x) = \min\{f_{n_i}^{(p_i)}(x)\ , \sigma\beta_i(x)\}$.

The symbol sigma, σ represents a measure of complexity on $\{\phi_i\}$ or $\{\alpha_i\}$.
The list $\Sigma_\phi = \{\sigma\phi_i\}$ is a complexity measure in the sense of Blum [4] (a Blum measure).

§ 3 Subrecursive Hierarchies

Algebraic Approach

One of the oldest and most influential subrecursive hierarchies is the Grzegorczyk hierarchy first presented in [12] in 1953. This hierarchy has since been defined in several different ways, see [1], [2], [7], [18], [21]. Crucial to the definition is the concept of the set of functions

elementary in f, $\mathcal{E}(f)$. The definition is sketched intuitively below so that the reader unfamiliar with one concept can see approximately what is involved. Let $\mathbb{Q}$ be the set of rational numbers, $\mathbb{Q} = \{0, \pm 1, \pm 2, \pm 1/2, \pm 3, \pm 1/3, \pm 2/3 \ldots\}$ Let $\mathcal{F}_{\mathbb{Q}}$ be the field of rational function under + and $\cdot$. Notice that the field is closed not only under + and $\cdot$ but also under the operation of substitution of functions for variables. Denote the operation of substitution by O_s. Now extend the field $\mathcal{F}_{\mathbb{Q}}$ by closing under the infinitary ring operations

$$s(x_1,\ldots,x_n,y) = \sum_{i=0}^{[y]} q(x_1,\ldots,x_n,i)$$

$$p(x_1,\ldots,x_n,y) = \prod_{i=0}^{[y]} q(x_1,\ldots,x_n,i)$$

for i an integer variable.

For brevity let B be any set of functions and $O_1,\ldots,O_p$ any operators $B^n \to B$, then $[B;O_1,\ldots,O_p]$ is the least class containing B and closed under O_i. The class of elementary functions over the rationals $\mathcal{E}_{\mathbb{Q}}$, is $[\mathcal{F}_{\mathbb{Q}};O_s,\Sigma,\Pi]$ (a superfield of $\mathcal{F}_{\mathbb{Q}}$). The class of elementary functions over $\mathbb{N}$, denoted simply $\mathcal{E}$, is obtained by relativizing $\mathcal{E}_{\mathbb{Q}}$ to $\mathbb{N}$. Succinctly defined, the class $\mathcal{E}$ is given by letting

$b_{-1}(x) = 0$, $b_0(x) = x + 1$.

$b_1(x,y) = x + y$, $b_2(x,y) = x \cdot y$,

$b_3(x,y) = x^y$, $B_1 = \{b_{-1},\ldots,b_i\}$.

Then $\mathcal{E} = [B_2, \text{Os}, \Sigma, \Pi] = [B_3, \text{Os}, \Sigma]$ and $\mathcal{E}(f) = [B_2, f; \text{Os}, \Sigma, \Pi]$

It is an open question of considerable interest whether $\mathcal{E}_{Q}$ can be obtained from $\mathcal{F}_{Q}$ using only Os and a sequence of new base functions, i.e. by a sequence of transcendental elements over the field $\mathcal{F}_{Q}$.

The algebraic way of extending the class $\mathcal{E}$ to the larger class $\mathcal{R}^1$ is to use a sequence of "transcendental extensions" of $\mathcal{E}$ by functions h_i. A sequence of such extending functions, $h_0, h_1, \ldots$ will be called a spine for a hierarchy up to $\mathcal{R}^1$ if

$\bigcup_{i=0}^{\infty} \mathcal{E}(h_i) = \mathcal{R}^1$. For the sequence f_i of Def. 21.†

Theorem 3.1: $\mathcal{E}(f_n) \subset \mathcal{E}(f_{n+1})$ for all $n > 2$.

Letting $\mathcal{E}^n = \mathcal{E}(f_n)$ for $n \geq 3$.

Theorem 3.2: $\bigcup_{n+3}^{\infty} \mathcal{E}^n = \mathcal{R}^1$.

These theorems are proved by routine rate of growth arguments. A function f is said to majorize a class $\mathcal{C}$ (written $f > \mathcal{C}$) iff for all $g \in \mathcal{C}$ there is a p such that $g(x_1, \ldots, x_n) < f_n^{(p)}(\max\{x_1, \ldots, x_n\})$. The notation $f < \mathcal{C}$ means there is a $g \in \mathcal{C}$ and $f(x) < g(x)$ for all x. It is shown by an inductive analysis of the definition of $\mathcal{E}(f_n)$ that f_n majorizes $\mathcal{E}(f_n)$. It follows that no one argument

†Grzegorczyk used a different sequence of functions, $g_0(x,y) = x + y$, $g_2(x,y) = (x + 1) \cdot (y + 1)$ and for $n > 2$ $g_{n+1}(0,y) = g_n(y + 1, y + 1)$ and $g_{n+1}(x+1,y) = g_{n+1}(x, g_{n+1}(x,y))$. His classes $\mathcal{E}^0 \subset \mathcal{E}^1 \subset \mathcal{E}^2 \subset \mathcal{E}^3 = \mathcal{E}$ are defined by Os and limited recursion from $g_i(\ )$. For $i < 3$ the $\mathcal{E}^i$ are not of interest here because they are not of the form $\mathcal{E}(f)$.

in $\mathcal{E}(f_n)$ can grow as fast as $f_{n+1}(x) = f_n^{(x)}(x)$. Thus $f_{n+1} \notin \mathcal{E}^n$.

Containment, $\mathcal{E}^n \subseteq \mathcal{E}^{n+1}$, is shown by giving an elementary scheme for defining f_n from f_{n+1}. A more intuitive approach to this step is discussed later.

To prove the equality of Theorem 3.2, Grzegorczyk used a formulation of $\mathcal{R}^1$ in terms of iteration, due to R. M. Robinson [22]. The functions f_n mirror the depth of nested iteration and this makes the argument straight forward. Once this theorem is available and the computational approach to $\mathcal{E}^n$ developed, as below, it becomes an easy matter to relate other hierarchies to $\mathcal{R}^1$.

The Grezegorczyk hierarchy $\mathcal{E}^n$ was first given a computational interpretation by Cleave [5] and later by Meyer & Ritchie [18]. It was related to the Kleene subrecursive hierarchy [15] and to the depth of nesting of primitive recursion by Axt [1] and [2].

The class $\mathcal{R}^1$ is but the first level in the oldest subrecursive hierarchy, Péter's [19] n-fold recursive functions $\mathcal{R}^n$. It was natural to ask whether Grzegorczyk's approach could be extended to $\mathcal{R}^n$. Robbin [21] answered the question for Grzegorczyk's hierarchy and Constable [7] answered the question for Cleave's computational version.

In terms of the algebraic approach, an extension procedure is quite naturally suggested. Simply consider a transfinite sequence of "transcendental elements". Thus for an ordinal α, let $\alpha_n \to \alpha$ be a fundamental sequence to α, e.g. $n \to \omega$, $\omega \cdot n \to \omega^2$, etc. Then new "longer" sequences are defined by the condition

$f_\alpha(x) = f_{\alpha_x}(x)$ for $\alpha_x \to \alpha$.

Given the standard fundamental sequences for ordinals $\alpha < \varepsilon_0$ it is shown in Constable [7] that for $\mathcal{E}^\alpha = \mathcal{E}(f_\alpha)$ $\alpha < \varepsilon_0$.

Theorem 3.3: $\mathcal{E}^\alpha \subset \mathcal{E}^\beta$ if $\alpha < \beta < \varepsilon_0$.

Robbin [21] showed that †

Theorem 3.4: $\bigcup_{\alpha<\omega^n} \mathcal{E}^\alpha = \mathcal{R}^n$.

In both of these results, the properness condition, $f_\alpha \notin \mathcal{E}^\beta$ if $\alpha < \beta$, is shown by a simple application of the growth rate arguments that Grzegorczyk used. The critical lemma is that f_α is strictly increasing for all α and all x.

To prove that $\mathcal{E}^\alpha \subseteq \mathcal{E}^\beta$, a computational analysis is used rather than a syntactic analysis of recursion schemes. (The syntactic method would result from a direct attempt to generalize Grzegorczyk's methods.) A computational approach appears necessary when the functions reach the complexity of $f_{\omega^{\omega}}$. The next section will consider the relationship between the Extended Grzegorczyk classes $\mathcal{E}^\alpha$ and measures of computational complexity.

Computational Approach

The computational method of analyzing hierarchies

---

† Robbin used a different set of functions, $w_0(x) = 2^x$ and $w_{\alpha+1}(x) = w_\alpha^{(x)}(1)$.

depends on two basic principles. They are stated first for the time measure, σ, and later generalized to restricted Blum measures

Given f( ), let ψ_f be a specific G-program for f( )

Principle 1: If $g \in \mathcal{E}(f)$, then there is a program ϕ_g for g and an elementary operator E[ ] such that $\sigma\phi_g(x) < E[\sigma\phi_f(\ )](x)$ for all x.†

This principle asserts that the computing system $\{\psi_i\}$ operates in an elementary manner. That is, if g( ) can be defined by a sequence of elementary operations, then the computing system can mimic those operations so that the cost is within an elementary operation of the cost of some specific algorithm for f( ). This principle is true for all existing models of computing systems, such as one-tape or multi-tape Turing machines or Register machines. Indeed it is a good criterion by which to judge the system, "does it do elementary arithmetic in <u>an elementary</u> manner?"

Principle II: If $\sigma\phi_i < \mathcal{E}(f)$, then $\phi_i \in \mathcal{E}(f)$.

This principle asserts that if there is a way to compute g( ) which is bounded by an elementary in f amount of time, then the function is elementary in f. This is less obvious that I. It was first noticed by Kleene for the notion of "primitive recursive in". Indeed

---

† E[ ] is an elementary operator if $E[f(\ )] \in \mathcal{E}(f(\ ))$ for all $f(\ ) \in \mathcal{R}$.

it is a direct consequence of the Kleene Normal Form Theorem (NFT) [14] or [24] which asserts that any $\phi_i \in \mathcal{R}$ satisfies

$\phi_i(x) = U(\mu y T(i,x,y)) \quad x \in \mathbb{N}^n$

where $U(\;) \in \mathcal{E}$ and the "T-predicate" is elementary. Principle II follows from the fact that although the operation of minimum, μ, is not elementary, the operation of limited minimum, $\mu \leq$, is. Therefore showing that $\sigma\phi_i < \mathcal{E}(f)$ implies that $\phi_i \in \mathcal{E}'(f)$, since all the operations involved are elementary in f.

The fundamental fact behind the NFT is that the operation of the computing system can be described in an elementary manner. It is difficult to imagine a real computing system which does not posses an elementary description. See Cobham [6] for a discussion of this point.

The two principles apply to the analysis of the Extended Grzegorczyk hierarchy, $\mathcal{E}'^{\alpha}$, in the following manner. Suppose each f_α has the property

Operator Honesty Property: There is an algorithm, f, for $f(\;)$ and an elementary operator $E_2[\;]$ such that $\sigma\phi_f(x) < E_2[f(\;)](x)$ for all x. Then observe that for such f

$g \in \mathcal{E}'(f)$ implies $\sigma\phi_{i(x)} < E_1[E_2[f(\;)]](x)$ for all x (by Honesty and I). So $g \in \mathcal{E}(f)$ implies $\sigma\phi_i < \mathcal{E}(f)$. Now by II $\sigma\phi_i < \mathcal{E}'(f)$ implies $\phi_i \in \mathcal{E}(f)$. Hence

Theorem 3.5: For f as above, $g \in \mathcal{E}(f)$ iff $\sigma\phi_g < \mathcal{E}(f)$.

As long as f_α has the (operator) Honesty Property and principle II holds, the question of membership in $\mathcal{E}^\alpha = \mathcal{E}(f_\alpha)$ is reduced to a question of estimating bounds. A crucial step in proving $\mathcal{E}^\alpha \subset \mathcal{E}^\beta$ is the verification of honesty. Once this is accomplished, then the hierarchy can be analyzed by the simple technique of comparing growth rates, in particular showing that f_α majorizes $\mathcal{E}^\alpha$.

It is easy to see that the run-time functions $\sigma\phi_i$ satisfy a stronger honesty condition. First define

Def. 3.1: f is h-honest iff $\exists \phi_i = f$ and $\sigma\phi_i(x) < h(\phi_i(x))$ for all x. f is elementary honest iff $h \in \mathcal{E}$. The $\sigma\phi_i$ are elementary honest. This allows

Theorem 3.6: If f has the (operator) honesty Property, then $g \in \mathcal{E}(f)$ iff $\sigma\phi_g \in \mathcal{E}(f)$.

The proof follows by noticing that $\sigma\phi_g < \mathcal{E}(f)$ and that honesty for $\sigma\phi_g$ implies that there is a $\phi_j = \sigma\phi_i$ and $\sigma\phi_j < \mathcal{E}(f)$.

Abstract Computational Approach

The abstract approach to computational complexity can be used to cast the previous observations in a more general setting. The abstraction begins with an acceptable indexing, $\{\phi_i\}$, as the generalization of a particular general recursive computing system. See Rogers [23] for a treatment of these indexings. Given $\{\phi_i\}$, a Blum measure of computational complexity is defined as a list of functions $\Phi = \{\Phi_i\}$ such that there is a 0,1-valued recursive function M, and the following axioms are satisfied:

Axiom 1: $\phi_i(x)\downarrow$ iff $\Phi_i(x)\downarrow$

Axiom 2: $M(i,x,y) = 1$ iff $\Phi_i(x) = y$

<u>Example</u>: The list $\Sigma = \{\sigma\phi_i\}$ is a Blum measure.

The kinds of classes of interest here are the complexity classes of Hartmanis & Stearns [13] and their "everywhere" counterparts.

Def. 3.2: $\mathcal{R}_t^{\Phi} = \{\text{total } \phi_i \mid \Phi_i(x) \leq t(x) \text{ a.e.x}\}$

$\overline{\mathcal{R}}_t^{\Phi} = \{\text{total } \phi_i \mid \Phi_i(x) \leq t(x) \text{ for all } x\}$

The a.e. (almost everywhere, i.e. except for a finite set) classes, $\mathcal{R}_t$, are most common in the literature of complexity theory, but the "everywhere" classes, $\mathcal{R}'_t$, will also be useful here.

Following Blum, call the pair $<\phi_i , \Phi_i>$ a <u>machine class</u>. Call the machine class <u>elementary</u> if the following additional axioms are satisfied for ϕ_i and ϕ_j total functions.

Axiom 3: If $\phi_i \in \mathcal{E}(\phi_j)$ then there is an elementary operator $E[\ ]$ and a $\phi_k = \phi_i$ such that $\Phi_k(x) \leq E[\Phi_j(\ )](x)$ for all x.

Axiom 4: $\Phi_i < \mathcal{E}(\phi_j)$ implies $\phi_i \in \mathcal{E}(\phi_j)$.

Axiom 5: ϕ_i are elementary honest, i.e. there is an $h \in \mathcal{E}$ such that for all i there is a $\phi_q(\ ) = \phi_i(\ )$, and if $\phi_i(x)\downarrow$ then $\Phi_q(x) < h(\phi_i(x))$.

Def. 3.3: Given an elementary machine class (emc), $<\phi_i, \Phi_i>$, call an ordinally indexed sequence of recursive functions, $\{h_\alpha\}$, an (elementary) spine iff

(i) each h_α is strictly increasing

(ii) each h_α is elementary operator honest

(iii) $h_\alpha(x) < h_\beta(x)$ for all $x > N_\alpha$ if $\alpha < \beta$.

Theorem 3.7: If $\{h_\alpha\}$ is an (elementary) spine, then

$$g \in \mathcal{E}(h_\alpha) \text{ iff } \exists\phi_g = g \text{ and } \Phi_g \in \mathcal{E}(h_\alpha).$$

Def. 3.4: Call an (elementary) spine normal iff

$h_{\alpha+1}(x) = h_\alpha^{(x)}(x)$ for all x. Note, supplying the initial value $h_0(x) = x + 1$ generates the normal Grzegorczyk spring up to ω.

Theorem 3.8: If $\{h_\alpha\}$ is a normal spine over an emc, then $\mathcal{E}(h_\alpha) \subset \mathcal{E}(h_\beta)$ for $3 < \alpha < \beta$.

The proof of this theorem requires properties (i) - (iii) of f_α, Axiom 4 (principle II) and the standard techniques of growth rate analysis.

Given this theorem the question of whether an Extended Grzegorczyk type hierarchy exists up to an ordinal α is reduced to the question of whether a normal spine exists up to α.

Theorem 3.9: If $h_0 \in \mathcal{R}^1$ and $\{h_\alpha\}$ is a normal spine, then

$$\bigcup_{\alpha<\omega} \mathcal{E}(h_\alpha) = \mathcal{R}^1.$$

Def 3.5: Call $\{h_\alpha\}$ $\alpha < \gamma$ an ε_0-standard spine if $h_\alpha(x) = h_{\alpha_x}(x)$ for $\alpha_n \to \alpha$ the standard fundamental sequence to $\alpha < \varepsilon_0$.

Theorem 3.10: If $h_0 \in \mathcal{R}^1$ and $\{h_\alpha\}$ is a normal ε_0-standard spine, then

$$\bigcup_{\alpha<\omega^n} \mathcal{E}(h_\alpha) = \mathcal{R}^n.$$

Interesting relationships exist between elementary classes, $\mathcal{E}(f)$, and Φ-complexity classes over an emc. For instance, if f is strictly increasing and $f > \mathcal{E}$, then f will majorize $\mathcal{E}(f)$. In fact a generalized Ritchie theorem holds.

Theorem 3.11: If f is strictly increasing, elementary operator honest over an emc and $f > \mathcal{E}$, then

$\mathcal{R}_f(p) \subset \mathcal{R}_f(p+1)$ and

$$\bigcup_{p=0}^{\infty} \mathcal{R}_{f}(p) = \mathcal{E}(f).$$

Cor. 3.12: If $\{h_\alpha\}$ is an elementary spine and $\mathcal{E}^\alpha = \mathcal{E}(h_\alpha)$, then if $h_\alpha > \mathcal{E}$, then $\bigcup_{p=0}^{s} \mathcal{R}_{h_\alpha}(p) = \mathcal{E}^\alpha$.

All of these results follow by applying the general principles I and II (axioms 3 and 4) as they have been applied in the literature for the special cases. The difficult matter of showing that relatively long spines exist is put aside.

Another relationship between complexity classes and Grzegorczyk type classes is given by the Union Theorem of McCreight & Meyer [17]. Putting $\mathcal{E}^n_\alpha = \mathcal{E}(h_{\alpha+n})$ and $\mathcal{R}^\alpha = \bigcup_{n=0}^{\infty} \mathcal{E}^n_\alpha$ the theorem asserts

Theorem 3.13: For $\{h_\alpha\}$ a normal spine over an emc, and $\mathcal{E}^\alpha = \mathcal{E}(h_\alpha)$ there are t_α and u_α such that $\mathcal{E}^\alpha = \mathcal{R}^\Phi_{t_\alpha}$

and $\mathcal{R}^\alpha = \mathcal{R}^\Phi_{u_\alpha}$.

From the recent work of McCreight & Meyer [17] a very interesting type of spine emerges. It could be considered a "minimal" spine. First it follows from Blum [4] that any complexity class $\mathcal{R}^\Phi_t$ named by an honest t can be extended by applying a "jump function" h( ) to t, i.e., $\mathcal{R}^\Phi_t \subset \mathcal{R}^\Phi_{h\circ t}$. The situation can be arranged so that

(i) $t_n(x) < t_{n+1}(x)$ a.e. x and $\mathcal{R}_{t_n} \subset \mathcal{R}_{t_{n+1}}$;

(ii) $t_{n+1}(x) < h_1(t_n(x))$ a.e.x $h_1(\ )$

$h_1(\ ) \in \mathcal{E}$, $h_1(\ )$ strictly increasing and

$h_2(t_n(x)) < t_{n+1}(x)$ a.e.x;

(iii) each t_n is h honest for $h \in \mathcal{E}$.

At the limit stage the union theorem guarantees that $\bigcup_{n=0}^{\infty} \mathcal{R}_{t_n} = \mathcal{R}_u$ for some increasing u.

The McCreight & Meyer [17] honest theorem guarantees that there is a measured set of functions Γ which can name every complexity class. In particular then there is an honest $\bar{u}$ such that $\mathcal{R}_u = \mathcal{R}_{\bar{u}}$. The h in (ii) is taken to be an h for which Γ is h-honest.

It appears possible that for u's constructed from an increasing sequence of the type t_n, $\bar{u}$ can be made strictly increasing. If this is the case, let t_ω = strictly increasing $\bar{u}$. Then for each ordinal γ a <u>minimal spine</u> up to γ can be selected simply by choosing fundamental sequences for ordinals $< \gamma$. In particular there exists an ε_0-standard minimal spine. Let $\mathcal{R}^{\Phi}_{\mu_\alpha}$ be the hierarchy produced by the minimal spine.

Unfortunately as the author has shown, even if minimal spines constructed via the McCreight-Meyer procedure do exist, they are so fine that

Theorem 3.14: If $t_0 \in \mathcal{E}$, then for every constructive ordinal γ

$$\bigcup_{\alpha < \gamma} \mathcal{R}^{\Phi}_{\mu_\alpha} \subset \mathcal{E}.$$

<u>§ 4 Size of Programs</u>

According to Blum [3] the notion of program size can be abstractly defined by specifying a <u>size function</u>

$|| : \mathbb{N} \to \mathbb{N}$ which satisfies

Condition 1: $|| : \mathbb{N} \to \mathbb{N}$ is recursive

Condition 2: $|y|^{-1}$ is finite for all y

condition 3: there is a recursive function b such that

b(y) is a bound on the cardinality of $|y|^{-1}$.

Given a programming language (formalism) $\{\phi_i\}$, the size of a program (index) i is simply $|i|$, the value of the size function.

As an example of a size function consider the following inductive definition of the <u>length</u> of a G-program. ln (<Letter>) = 1, ln(<Letter>$_n$) = ln(<variable>) = n + 1. If t is a term, say t = (a<operator>b) where a and b are terms, then ln(t) = ln(a) + ln(b) + 1. For assignment statements, ln(<variable>←<term>) = ln(<variable>) + ln(<term>) + L. If L is a label formed by concatenating the labels L_1 and L_2 then ln(L) = ln(L_1) + ln(L_2), and ln(<letter>) = 1, ln(n) = n. Finally ln(if<variable> = 0 then <label>) = ln(<variable>) + ln(<label>) + 2. The length function, ln( ), is a valid size function. From here on let $||$ be the ln( ) size function.

The results of the hierarchy section show that "clock-bounded" formalisms similar to P can be defined for all classes $\mathcal{E}^{\alpha}$, more generally for any class $\mathcal{R}_t^{\phi}$. In particular, the clocks $f_3^{(p)}(\ )$ can be used to define a c.b. (clock-bounded) language for $\mathcal{E}$. The programs have the form <clock,G-program> where the clock is (3,p). Let $\ell_0, \ell_1, \ldots$ be an enumeration of these programs (say E-programs), and let $e_i = \langle(3,p_i) , \beta_i\rangle$ where $\beta_i = \phi_j$ for some G-program ϕ_j.

A reasonable size function on E-program is

$|\ell_i|_3 = p_i + |\beta_i|$.

(notice there is a c, $c = |f_3|$, such $|\ell_i|_3 + c = |\bar{\ell}_i|$ where $\bar{\ell}_i$ is the G-program which first computes $f_3^{(p)}(x)$ and then behaves like ℓ_i .)

Because $\mathcal{E}$ is recursively enumerable and because all E-programs halt, it is possible to enumerate the shortest E-programs for $\mathcal{E}$. Let $m_0, m_1, \ldots$ be an enumeration of the shortest programs (thus $m_0(\ ), m_1(\ ), \ldots$ is an enumeration of $\mathcal{E}$). According to Blum [3], for each function $f(x) = x + s$ $s \in \mathbb{N}$ there is some m_i and ϕ_j such that $m_i(\ ) = \phi_j(\ )$ and $f(|\phi_j|) \leq |m_i|_3$. Without loss of generality assume $f(|\phi_j|) = |m_i|_3$, i.e. $|\phi_j| + s = |m_i|_3$. Say that $\underline{\phi_j \text{ shrinks } m_i \text{ by } s}$.

Call an E-program p-complex iff p is the least j such that $\sigma\ell_i(x) < f_3^{(j)}(x)$ for all x.

Theorem 4.1: Suppose that the program ϕ_j shrinks m_i by s without loss of efficiency, them m_i is at least s-complex.

The proof is simple. Since m_i is the shortest E-program and ϕ_j is efficient, the only way ϕ_j can shrink m_i is by removing the clock. Thus the clock must be of size s, so $s \leq p_i$. That is,

$|\phi_j| + s = |m_i|_3 = p_i + |\beta_i|$. If $|\phi_j| < |\beta_i|$ then

$<(3,p_i)\ ,\ \phi_j>$ would be a shorter E-program for $m_i(\ )$ because $\sigma\phi_j(x) < f_3^{(p_i)}(x)$ for all x. Thus $|\phi_j| \geq |\beta_i|$, so $d = (|\phi_j| - |\beta_i|) > 0$ and $s + d = p_i$.

Observe that for fixed p, say a p determining the limit of the usable levels of $\mathcal{E}$, the value q required to satisfy $f_3^{(p)}(x) < f_n^{(q)}(x)$ decreased monotonically as n increases. Therefore there is an n_p such that $f_3^{(p)}(x) < f_{n_p}(x)$ for all x. In the clock formalism for $\mathcal{C}(f_{n_p})$ with the size measure $||_{n_p}$, the programs of $\bar{\mathcal{R}}_{f_3}(p)$ cannot be shrunk by any G-program without loss of efficiency. The same results apply to any level of the hierarchy $\mathcal{E}_\alpha^n$ defined earlier.

These ideas can also be used to formulate a conservation or "trade-off" principle. Notice that if the level p of the Ritchie hierarchy, $\cup\,\mathcal{R}_{f_3}(p)$ can be enumerated, say $p_0, p_1, \ldots$. Then by Blum's result there is a G-program ϕ which can shrink the size of some p_i by s, even for s >> p. Moreover this can be done without loss of efficiency <u>except on a finite set S</u>. According to Thm 4.1, Blum's statement cannot be strengthened to hold everywhere. Stated in other terms

Theorem 4.2: Programs in a fixed level p of the (generalized) Ritchie hierarchy for $\mathcal{E}^\alpha$ cannot be shortened with respect to $||_\alpha$ more than p without loss of efficiency at least on a finite set S. For any s there are programs which can be shortened by s with a loss of efficiency equal to (s - p), i.e. the shorter program must be (s-p)-complex at least on a finite set S.

Thus when computational complexity is measured by an everywhere condition, a conservation principle holds between size and efficiency.

Acknowledgments

The author would like to thank Dr. Allan Borodin for his helpful discussions about this paper, especially for his help and independent conclusions on minimal spines. Special thanks go to Pat Hauk for her excellent typing work.

References

1]. Axt, Paul "Enumeration and the Grzegorczyk hierarchy," Z. Math Logik Grund.Math 9 (1963), 53-65.

2]. Axt, Paul "Iteration of primitive recursion," abstract 597-182, Noticea A.M.S., Jan. 1963.

3]. Blum, M. "On the Size of Machines," Information and Control, II, (1967), 257-265.

4]. Blum, M. "Machine-Independent Theory of the Complexity of Recursive Functions," JACM, 14, (1967), 322-36.

5]. Cleave, John P., "A Hierarchy of Primitive Recursive Functions," Zeitachr. F. Math Logik and Grund. D. Math., 9, (1963), 331-345.

6]. Cobham, Alan "The Intrinsic Computational Difficulty of Functions," Logic, Methodology and Philosophy of Science, Amsterdam, 1965.

7]. Constable, Robert L. Extending and Refining Hierarchies of Computable Functions. Comp. Sci. Tech. Report #25, Univ. of Wisc., 1968.

8]. Constable, Robert L. "Subrecursive programming languages for $\mathcal{R}^n$," Comp. Sci. Tech. Report 70-53, Cornell Univ., 1970.

9]. Conatable, Robert L. and Allan B. Borooin "On the efficiency of programs in subrecursive formalisms," Comp. Sci. Tech. Report 70-54, Cornell Univ., 1970.

10]. Elgot, C. C. and A. Robinson "Random-Access Stored Program Machines, An Approach to Programming Languages," J.A.C.M., 11, (1964), pp. 365-399.

11]. Fabian, Robert J. Hierarchies of general recursive functions and ordinal recursion, Ph.D. Diss. Case Inst of Tech., 1965.

12]. Grzegorczyk, A. "Some Classes of Recursive Functions," Rozprawy Matematcyzne, (1953), 1-45.

13]. Hartmanis, J. and R.E. Stearna "On the Computational Complexity of Algorithma," Trans. AMC, 117, 5, (1965), 285-306.

14]. Kleene, S. C. Introduction to Metamathematics, Princeton, 1952.

15]. Kleene, S. C. "Extension of an Effectively Generated Class of Functions by Enumeration," Collog, Math. 6, (1958), pp. 67-73.

16]. McCarthy, John "A Basis for a Mathematical Theory of Computation," Computer Programming and Formal Systems, Amsterdam, 1963, pp. 30-70.

17]. McCreight, E. M. and A. R. Meyer "Classes of computable functions defined by bounds on computation," ACM Symp. on Theory of Computing, 1969, 79-88.

18]. Meyer, A. R. and D. M. Ritchie "The complexity of Loop programs," Proc. 22 National ACM Conf. (1967), 465-470.

19]. Péter, Roze Recursive Functions, 3d ed., New York, 1967.

20]. Ritchie, Robert W. "Classes of Predictably Computable Functions, Trans." A.M.S., 106, 1963, pp. 139-173.

21]. Robbin, Joel Subrecursive hierarchies, Ph.D., Diss, Princeton, 1965.

22]. Robinson, R. M. "Primitive recursive functions," Bull AMS, 53, (1947), 915-942.

23]. Rogers, H. "Gödel numberings of partial recursive functions," J.SL, 23,3 (1958), 331-341.

24]. Rogers, Hartley Jr. Theory of Recursive Functions and Effective Computability, New York, 1967.

25]. Scott, Dana Some Definitional Suggestions for Automata Theory, J. Compts., & Syst. Sci., 1, (1967), pp. 187-212.

26]. Shepherdson, J. C. and H. E. Sturgis "Computability of Recursive Functions," J.A.C.M., 10, 1963, pp. 217-255.