

A NOTE ON "AXIOMS"
FOR COMPUTATIONAL COMPLEXITY
AND COMPUTATION OF FINITE FUNCTIONS¹

Paul R. Young
April 1970
CSD TR 39
(Revised version)

Computer Sciences Department
Purdue University
Lafayette, Indiana 47907

Short title: ON THE COMPUTATION OF FINITE FUNCTIONS

List of Symbols

Greek: ϕ, Φ, λ

Math: $<, \leq, \underline{f}, +$

$\Sigma, \{, \}, \langle, \rangle, |, /$

ABSTRACT

Recent studies of computational complexity have focused on "axioms" which characterize the "difficulty of a computation" (Blum, 1967a) or the measure of the "size of a program," (Blum 1967b and Pager 1969). In this paper we wish to carefully examine the consequences of hypothesizing a relation which connects measures of size and of difficulty of computation. The relation is motivated by the fact that computations are performed "a few instructions at a time" so that if one has a bound on the difficulty of a computation, one also has a bound on the "number of executed instructions". This relation enables one to easily show that algorithms exist for finding the most efficient programs for computing finite functions. This result, which has been obtained independently for certain Turing machine measures by David Pager, contrasts sharply with results for measures of size, where it is known that no algorithm can exist for going from a finite function to the shortest program for computing it, (Blum, 1969b) (Pager, 1969). In a concluding section, which can be read independently of the above-mentioned results, some remarks are made about the desirability of using a program for computing an infinite function when one is interested in the function only on some finite domain. There is nothing deep in this paper, and we hope that a reader familiar with the rudiments of recursion theory will find this paper a simple introduction to the "axiomatic" theory of computational complexity. Such a reader might do well to begin with the concluding remarks after reading the basic definitions.

We let $\lambda_i \phi_i$ be a standard enumeration of all partial recursive functions. For expository purposes it will be convenient to assume that with each i we have effectively associated some program P_i which computes exactly the function ϕ_i . Because we can pass back and forth effectively between programs P_i and indices i , we identify P_i with i and may, e.g., speak of "program i ". Following Blum (1967a), we call a sequence $\lambda_i \phi_i$ of partial recursive functions a measure if it satisfies

- Axiom 1. For all i , the domain of ϕ_i = the domain of ϕ_i .
 and Axiom 2. There is an algorithm for deciding, given i , x , and y whether $\phi_i(x) \leq y$.

For example, $\phi_i(x)$ might be the number of executed instructions if the i^{th} algol program or Turing machine is operated on input argument x , or it might be the amount of tape or storage space used if the program halts.

We follow Blum (1967b) in saying that a function $| \cdot |$ measures the size of programs if it satisfies

- Axiom 3. There is an effective procedure for listing, given n , the entire finite set of programs, P_j satisfying $|P_j| = n$, and for knowing when the listing is completed. We sometimes write $|j|$ for $|P_j|$.

(The reader should be warned that the finiteness condition rules out, e.g., measuring the size of a "FORTRAN-like" program by the number of its instructions. This follows by observing that there are infinitely many

simple instructions of the form: WRITE 0, WRITE 1, WRITE 2,.... . A suitable measure of size would be the total number of characters or even the total number of cards in a punched program.)

In this paper, we shall call a quadruple of the form $\langle P, \phi, \phi, | | \rangle$, where ϕ is a standard indexing of the partial recursive functions, ϕ is a measure of computational complexity satisfying Axioms 1 and 2, $| |$ is a measure of size satisfying Axiom 3, and P is a mapping from integers to programs such that P_i computes ϕ_i , a measured programming system. The programs P_i are included primarily as an aid to exposition. Since it is possible to always pass effectively back and forth between i and P_i , one can always dispense with the programs P_i in favor of working directly with the indexing $\lambda i \phi_i$.

From Blum (1967b) and Pager (1969), we know that for any programming system $\langle P, \phi, \phi, | | \rangle$, there is no algorithm which, given a finite function g , produces a program P_i for which the size $|i|$ is minimal while the program computes the function g for all arguments in the domain of g . (In fact, in Pager (1969), this is proven with no assumptions about the computability of the function $| |$.) We now ask whether we can find a program P_i such that $\phi_i(x) = g(x)$ for all $x \in \text{domain } g = D$ and for which $\sum_{x \in D} \phi_i(x)$ is a finite function minimal. Given g (e.g., by being given its table) we can certainly find some program P_i for which $\phi_i = g$, so suppose

$$\phi_i = g \text{ and } \sum_{x \in D} \phi_i(x) = M$$

Suppose for the moment that ϕ_i is a measure of how much time it takes for program i to operate. Now if there is some program P_j such that $\phi_j/D = g/D$ and $\sum_{x \in D} \phi_j(x) < M$ then there must be such a program $P_{j'}$ with $|P_{j'}| < M$. The intuitive reason for this is that programs execute one instruction at a time. Thus if we eliminate from P_j all instructions not actually executed in calculating ϕ_j/D , we obtain a program $P_{j'}$ such that

$$|P_{j'}| \leq \sum_{x \in D} \phi_j(x) < M$$

while $\phi_{j'}/D = \phi_j/D = g/D$.

The situation if ϕ_i measures the amount of storage used by program P_i is only slightly more complicated. Suppose $\sum_{x \in D} \phi_j(x) < M$. Consider any computation by P_j for argument $x \in D$. The requirement $\sum_{x \in D} \phi_j(x) < M$ bounds the amount of storage which program P_j may actually use. Therefore if an excessively large number of instructions are executed by P_j in computing $\phi_j(x)$, the contents of storage must be repeated. But the instructions which were executed between times when the storage contents were repeated could all have been bypassed. In short, if we have a bound on the storage which can be used, we can compute a bound on the number of instructions which need be executed, and hence can compute a bound on the size of the programs which need be considered in looking for programs which require less storage.

We would like to summarize the preceding discussion as a new principle which relates measures of size and complexity. We do this with Principle R below. Unfortunately, within the Blum theory we cannot talk about "program instructions", so we must formalize the preceding discussion by formalizing the conclusion rather than directly formalizing the reasons for the conclusion.

→ Although the principle may appear to have a complicated statement, it is a fairly direct translation of the conclusions in the final sentence of the preceding paragraph. We let $\lambda y D_y$ denote a canonical one-one enumeration of all finite sets: given y , we can list D_y and know when the listing is completed.

Principle R. A measured programming system $\langle P, \phi, \phi, || \rangle$ satisfies Principle R if there is a ^{total} computable function $c(z, y, i)$ such that, if ϕ_i is defined on D_y and if there exists some program P_j satisfying

$$(1) \quad \phi_i / D_y = \phi_j / D_y$$

with (2)
$$\sum_{x \in D_y} \phi_j(x) < \sum_{x \in D_y} \phi_i(x),$$

then there exists some j satisfying (1), (2) and

$$(3) \quad |P_j| \leq c(\sum_{x \in D_y} \phi_i(x), y, i)^2$$

Theorem 1. Let $\langle P, \phi, \Phi, || \rangle$ be any measured programming system, let F_y be an enumeration of all tables for defining functions mapping finite sets of integers into integers, and let D_y denote the domain of F_y . Then Principle R holds if and only if there is an effective procedure, given the table F_y , for finding a program P_j which computes F_y most efficiently (i.e., $\phi_j/D_{y'} = F_y$ and $\sum_{x \in D_y} \phi_j(x)$ is as small as possible).

Proof. We let $P_{f(y)}$ be some program which computes the function F_y , e.g., by encoding its table.

Suppose that Principle R holds. Then, since $\phi_{f(y)} = F_y$, we may, given y , compute

$$\sum_{x \in D_{y'}} \phi_{f(y)}(x) = \text{def. } M.$$

We may next list all programs P_j such that

$$(4) \quad |P_j| \leq c(M, y', f(y)).$$

(There are only finitely many such programs.) Of these, we can effectively find those programs P_j for which

$$(5) \quad \sum_{x \in D_{y'}} \phi_j(x) \leq M.$$

Finally for those programs P_j satisfying (4) and (5), $\phi_j(x)$ is defined for all $x \in D_{y'}$ so for such j we may actually decide whether

$$(6) \quad \phi_j/D_{y'} = F_y,$$

Thus to find a most efficient program for calculating F_y , we simply choose a program P_j satisfying (4), (5), and (6) for which the sum in (5) is minimal. If there is no program P_j satisfying (4), (5), and (6), program $P_{f(y)}$ must itself be a most efficient way of calculating F_y , so that iteration of this process must yield a most efficient program for calculating F_y .

Conversely, if we can, given F , find a most efficient way of calculating F , then Principle R holds because we may define $c(z,y,i)$ simply by

$$c(z,y,i) = \begin{cases} \text{the size of a most efficient program for} \\ \text{computing the table for } \phi_i/D_y \text{ if } \sum_{x \in D_y} \phi_i(x) \leq z, \\ \\ 0 \text{ if } \sum_{x \in D_y} \phi_i(x) \not\leq z. \end{cases}$$

We are indebted to John Berenberg for first pointing out to us the validity of the first part of the preceding proof for Turing machine models. A similar proof for Turing machine models may be found in Pager (1970). Pager also defines efficiency of programs over infinite sets and shows for his Turing machine models that an algorithm for finding the most efficient algorithm exists only if the domain set is finite. (Pager also uses an effective probability function which accounts for the probability that a given argument will be called, but for our purposes this is easily made part of the measure, ϕ .)

Theorem 2. Axioms 1-3 do not imply Principle R.

Proof. We first start with any measured programming system $\langle P, \phi, \phi, || \rangle$. Let K be any infinite set of integers which can be effectively generated, but which has no algorithm for deciding given n , whether or not $n \in K$. Let k be any 1-1 total recursive function which enumerates K , (so $K = \{k(0), k(1), k(2), \dots\}$). We now define a new measured programming system $\langle P', \phi', \phi', ||' \rangle$ as follows:

$$P'_{2i} = P_i, \phi'_{2i}(y) = \phi_i(y) + 1, |P'_{2i}|' = |P_i|, \text{ (so } \phi'_{2i} = \phi_i),$$

while P'_{2i+1} is any program P_x which writes $k(i)$ on input 0, and fails to halt on inputs $y \neq 0$,³

$$\phi'_{2i+1}(0) = 0 \text{ but } \phi'_{2i+1}(y) \text{ is undefined if } y \neq 0,$$

and $|P'_{2i+1}|' = 2i+1.$

Verification that Axioms (1), (2), and (3) hold is straightforward. But now the most efficient program P'_j for computing the finite function $\{ \langle 0, n \rangle \}$ has $\phi'_j(0) = 0$ if $n \in K$ while $\phi'_j(0) > 0$ if $n \notin K$. Thus if we could, given n , find a most efficient program for computing $\{ \langle 0, n \rangle \}$, we could decide whether $n \in K$ by finding a most efficient program $P'_{j(n)}$ for computing $\{ \langle 0, n \rangle \}$ and then testing whether $\phi'_{j(n)}(0) \leq 0$.

Theorem 1 says that if we are to be able, given a finite function, to find the most efficient algorithm for computing it, we can do so assuming Principle R. On the other hand Theorem 2 assures us that some such principle is really necessary. Although we feel that Principle R is really more basic than the ability to find the most efficient algorithm for computing finite functions, Theorem 1, suggests that these are perhaps really equivalent principles. That this is not in fact the case follows by showing that under a weakening of Axiom 3, Principle R no longer implies the existence of algorithms for finding the most efficient means for computing finite functions. Thus Principle R has (in our opinion) not only the advantage of being the more intuitively appealing of the two principles, but also the advantage of being the logically weaker principle. We show this next.

We say that a function $| \cdot |$ is a pseudo-measure of size if it satisfies

Axiom 3'. $| \cdot |$ is a finite-one total recursive function.

Clearly Axiom 3 implies Axiom 3', for if $| \cdot |$ satisfies 3 it is by definition finite-one and to compute $|i|$ one simply lists all j_0 such that $|j_0| = 0$, all j_1 such that $|j_1| = 1, \dots$ until eventually one lists i among those j_n for which $|j_n| = n$.

Theorem 3. A. In any measured programming system satisfying Axioms 1, 2, and 3', if there is an algorithm which enables one to pass effectively from a finite function to a most efficient program for computing the function, then Principle R holds.

B. There is a system satisfying Axioms 1, 2, and 3' in which Principle R holds but no such algorithm exists.

Proof of A. This is identical with the corresponding proof in Theorem 1.

We did not use the full force of Axiom 3 there.

Proof of B. We assume that $\langle P, \phi, \phi, || \rangle$ is any measured programming system satisfying Axioms 1, 2, and 3, and Principle R. We modify $\langle P, \phi, \phi, || \rangle$ to obtain a new measured programming system $\langle P', \phi', \phi', ||' \rangle$ much as in the proof of Theorem 2. Namely, we take $P'_{2i} = P_i$, $\phi'_{2i}(x) = \phi_i(x) + 1$, and $|P'_{2i}|' = |P_i|$. However we now take $P_{x(i)}$ to be the program which writes $k(i)$ (the i th member of a nonrecursive but enumerable set K) on input 0 and is obtained by the use of Theorem 1 so that $P_{x(i)}$ computes the function $\{ \langle 0, k(i) \rangle \}$ as efficiently as possible in the system $\langle P, \phi, \phi, || \rangle$. We obtain P'_{2i+1} by introducing new symbols not in the language of the system $\langle P, \phi, \phi, || \rangle$ and adding these to $P_{x(i)}$ to guarantee that P'_{2i+1} does not halt on inputs other than 0. Formally we have: $\phi'_{2i+1}(0) = k(i)$, and $\phi'_{2i+1}(x)$ is undefined if $x \neq 0$, but we now define $\phi'_{2i+1}(0) = 0$ and $|P'_{2i+1}|' = |P_{x(i)}|$.

The reader may easily verify that Axioms 1, 2, and 3' hold in $\langle P', \phi', \phi', ||' \rangle$. Furthermore, $n \in K$ iff the most efficient program P'_j for computing the function $\{ \langle 0, n \rangle \}$ has $\phi'_j(0) = 0$, so no algorithm for finding the most efficient program P'_j can exist.

To complete the proof we must verify the existence of a function c' which witnesses the fact that Principle R holds in the system $\langle P', \phi', \phi', ||' \rangle$. To calculate $c'(m, y, i)$ we proceed as follows: Given m, y, i , we first test whether

$$(7) \quad \sum_{x \in D_y} \phi_i(x) \leq m$$

If the answer is no, we do not care about the value of $c'(m, y, i)$ so we define

$$c'(m, y, i) = 0$$

If the answer is yes and if $D_y \neq \{0\}$, the most efficient program for computing ϕ_i/D_y in the two systems $\langle P', \phi', \phi', ||' \rangle$ and $\langle P, \phi, \phi, || \rangle$ are identical; because inequality (7) holds we may actually find ϕ_i/D_y , and by Theorem 1 we can effectively find the most efficient such program, call it P_q , in the system $\langle P, \phi, \phi, || \rangle$, so we may simply define

$$c'(m, y, i) = |P'_{2q}|' (= |P_q|).$$

If the answer is yes and $D_y = \{0\}$, since again $\phi_i(0) \leq m$, we may again actually find $\phi_i(0)$ and the most efficient program P_q in the system $\langle P, \phi, \phi, || \rangle$ for computing the function $\langle\langle 0, \phi_i(0) \rangle\rangle$. In this case, if $\phi_i(0) \in K$, P'_{2q} need not be the most efficient program for computing $\langle\langle 0, \phi_i(0) \rangle\rangle$ in the system $\langle P', \phi', \phi', ||' \rangle$, but it is clear from the construction that the size of the most efficient program for computing $\langle\langle 0, \phi_i(0) \rangle\rangle$ in the system $\langle P', \phi', \phi', ||' \rangle$ will be $|P'_{2q}|' (= |P_q|)$. In this case we may therefore again define

$$c'(m, y, i) = |P'_{2q}|'.$$

Concluding Remarks

Part of the purpose of Theorems 1 and 2 is to convince the reader that it may be worthwhile to consider the possibility that axioms 1-3 are still not an adequate basis for a fully developed theory of "abstract" computational complexity. (See also McCreight-Meyer (1968) and Young (1969).) Although we think that, upon reflection, the reader will find Principle R very reasonable and its consequences interesting, the results we have obtained are not deep. The justification for Axioms 1-3 is that they are not only intuitively appealing but that they have deep consequences, and any new axioms should also meet this test. .

We do believe that investigations of the computational complexity of finite functions should be further pursued because all functions in real computational problems are in fact finite. In any computational system $\langle P, \phi, \phi, | \rangle$, one can, given a finite function F_y , effectively find $f(y)$ such that $\phi_{f(y)} = F_y$. Since the most obvious method for doing this might be to encode the entire table for F_y into the program $P_{f(y)}$, one might say that program $P_{f(y)}$ computes $\phi_{f(y)}$ by table look up. In Young (1968) we proved that there exist 0-1 valued total recursive functions, ϕ_i , which are so difficult to compute that on almost all finite domains D , ϕ_i/D (the restriction of ϕ_i to D) is much more efficiently computed by table look-up than by any general program P_j for which $\phi_j = \phi_i$. Actually, as Albert Meyer pointed out to us, this holds whenever ϕ_i is a sufficiently difficult to compute 0-1 valued total recursive function. To see this, we now let $\lambda y F_y$ be an enumeration without repetitions of all finite 0-1 valued functions, and, as before we let f be a computable function for

which $F_y = \phi_{f(y)}$, and we denote the domain of F_y by D_y . We say that $P_{f(y)}$ computes F_y by table look-up.

Lemma. For any Blum measure ϕ , there exists a total recursive function g such that for all 0-1 valued finite functions F_y , g bounds the difficulty of computing F_y . Specifically, $\sum_{x \in D_y} \phi_{f(y)}(x) \leq \sum_{x \in D_y} g(x)$.

Proof. We define g by

$$g(n) = \max \left\{ \sum_{x \in D_y} \phi_{f(y)}(x) \mid n \in D_y, \subseteq \{0, 1, 2, \dots, n\} \right\}.$$

Clearly for any finite 0-1 valued function F_y , if m_y denotes the largest element of D_y , then

$$\sum_{x \in D_y} g(x) \geq g(m_y) \geq \sum_{x \in D_y} \phi_{f(y)}(x).$$

It should be pointed out that the preceding Lemma and the following theorem do not hold when $\lambda y F_y$ is allowed to range over all finite functions. This follows from the observation in McCreight-Meyer (1969) that for any Blum measure of complexity ϕ , there is a total recursive function $g(y, x)$ such that for all i , $\phi_i(x) \leq g(\phi_i(x), x)$ for all but finitely many x . Our next theorem is an immediate corollary of the preceding lemma.

Theorem 4. (Meyer) There exists a fixed total recursive function g such that whenever t is a 0-1 valued total recursive function for which $\phi_i = t$ implies $\phi_i \geq g$ a.e., then, on almost all finite domains D , t/D is more efficiently computed by table look-up than by any general program P_i for which $\phi_i = t$. Specifically, if $\phi_i = t$ then for all but finitely many finite domains D , if $F_y = t/D$, then $\phi_{f(y)} = t/D$ and $\sum_{x \in D} \phi_{f(y)}(x) < \sum_{x \in D} \phi_i(x)$.

Clearly, by requiring that t be more difficult to compute than some g' which is much greater than g , we may force table look-up to almost always be a much better method for computing t/D than is any general program for computing t .

Much recent work in complexity theory has considered only programs for infinite functions which are "sufficiently" difficult to compute. Theorem 4 suggests that, if one is interested in only finite segments of these functions, then these are just those programs which in practice should be used only for a few exceptional arguments in their domain. I.e., if an infinite function is sufficiently difficult to compute and one is interested in minimizing computational complexity/ ^{on finite domains}, then one should seldom use a program capable of computing the entire function because such a program will be unnecessarily inefficient. The situation is quite different if we are concerned with the size of programs:

Theorem 5. For any infinite function t , if $\phi_i = t$, then for all but finitely many finite domains D , if $F_y = t/D$ (so $\phi_{f(y)} = t/D$), then $|P_i| < |P_{f(y)}|$.

Proof. There are only finitely many programs P_j for which $|P_j| \leq |P_i|$.

Acknowledgements

This paper is a revision of an earlier paper, Young(1968), which included all results of this paper, except that Theorem 4 was given in a much weaker form. We are very grateful to Robert Ritchie and Albert Meyer, each of whom supplied an unusually large number of constructive criticisms and helpful suggestions for changes.

REFERENCES

(Extensive bibliographies of earlier work in computational complexity may be found in Blum(1967a) and of related work in pure recursion theory in Young(1969). For a more extensive bibliography of very recent work in "axiomatic" complexity theory, see [].)

Blum, Manuel, 1967a, A machine independent theory of the complexity of recursive functions, J. Assoc. Comp. Mach., 14, 322-336.

Blum, Manuel, 1967b, On the size of machines, Inf. and Control, 11, 257-265.

McCreight, E. M. & Meyer, A. R., 1969, Classes of computable functions defined by bounds on computation: Preliminary report, ACM Symposium on Theory of Computing, Assoc. Comp. Mach., New York, 79-88.

Pager, David, 1969, On finding programs of minimal length, Infor. and Control, 15, 550-554.

Pager, David, On the efficiency of algorithms, to appear

Rabin, M. O., 1963, Real time computation, Israel J. Math., 1, 203-211.

Young, P. R., 1969, Toward a theory of enumerations, J. Assoc. Comp. Mach., 16, 328-348.

Young, P. R., 1968, Computational speed-up by table look-ups, Purdue University Computer Sciences Dept. Tech. Report No. 30, 1-19.

FOOTNOTES

1. Supported by NSF Grant GP 6120

2. The preceding discussion in fact suggests that an even stronger principle should hold: namely c should be a function of the single variable $\sum_{x \in D_y} \phi_i(x)$. However, the weaker principle is adequate for our purposes and in any case in many models is not really weaker. In many models, $P, \phi, \phi, ||$, one must both read the input x and write the output $\phi_i(x)$. In such a situation, from a knowledge of $\phi_i(x)$ one can effectively bound both x and $\phi_i(x)$. But in this case, (assuming both the notation and results of Theorem 1), if we are given the function $\lambda z y i c(z, y, i)$ of Principle R, we may reduce it to a suitable function c' of a single variable as follows: Given z find d_z and r_z such that $w \geq d_z$ implies $\phi_i(w) > z$ and such that $w \geq r_z$ and $\phi_i(x) = w$ implies $\phi_i(x) > z$. Next set

$$c'(z) = \max\{|P_i| \mid P_i \text{ is a most efficient program for computing } F_y \text{ where } D_y = \{0, 1, \dots, d_z \text{ and } \max_{x \in P_y} F_y(x) \leq r_z.\}$$

3. It has been pointed out by the referee that this definition of P'_{2i+1} violates our initially stated requirement that the indexing of programs by one-to-one. This objection is easily overcome either by dropping the referencing to programs P and P' altogether or by enlarging the language of the programs P'_i to allow symbols not in the language of the programs P_i and then using these new symbols in defining P'_{2i+1} by adding to P_x a set of unexecutable instructions using these new symbols and letting the set depend on i .