Notre Dame Journal of Formal Logic
Volume 29, Number 2, Spring 1988

A Note on Arbitrarily Complex Recursive Functions

CARL H. SMITH*

1 Introduction The first result that one studies in abstract complexity theory is that there are arbitrarily difficult to compute recursive functions. In this note we prove a best possible generalization of that result. We show that there are arbitrarily sparse $\{0,1\}$ valued recursive funcions such that any finite variant of the constructed function is arbitrarily complex. Other potential strengthenings are shown to fail. The intricate nature of the main proof is necessitated only by the existence of pathological complexity measures. For the purpose of the main result, pathological complexity measures are shown to be ones where padding cannot be accomplished without significantly altering the complexity of the program being padded.

Rabin ([20]) first proved, in a machine dependent fashion, that there exist almost everywhere arbitrarily complex recursive functions with range $\{0,1\}$. His result was generalized to the machine independent case in [2]. Both proofs employ a nonconstructive element. A completely constructive proof of the existence of infinitely often arbitrarily complex recursive functions appears in [10]. The relative difficulty of proving functions almost everywhere arbitrarily complex, as opposed to infinitely often arbitrarily complex, is discussed in [8]. The nonconstructivity of the original Rabin–Blum proof (and other results from abstract complexity theory) motivated Lipton ([15]) to consider restricted models of arithmetic where only constructive proof techniques were allowed. This model was claimed adequate for research in theoretical computer science. However, it was shown in [11] that it was consistent with the model studied by Lipton to believe in some obviously false assertions about the complexity of computations,

*Supported, in part, by NSA OCREAE Grant MDA904-85-H-0002 and NSF Grant MCS 8301536. A preliminary version of this paper was presented at the Workshop on Logic and Theoretical Computer Science held in Lexington, Kentucky during June of 1985.

Received May 16, 1986; revised November 13, 1986

including the falsity of the result proved below. An excellent survey of results concerning models of arithmetic for computer science can be found in [12].

The proof below has a nonconstructive step. Indeed, Fulk, verifying a conjecture of Case (private communication), proved that any proof of the existence of $\{0,1\}$ valued almost everywhere arbitrarily complex recursive functions will have a nonconstructive component ([7]). In the main proof below, a program is written, effectively computing some function. The nonconstructive step enters the argument during the verification that the constructed function has the desired properties. To derive a contradiction to the existence of a fast finite variant of the function constructed, it is necessary to noneffectively pick certain points in the construction when crucial events must have occurred. Other proofs, such as the speed-up theorem ([2]) and some of the proofs in inductive inference ([4]), employ the construction of a sequence of programs where the program computing the desired function is selected necessarily noneffectively from the sequence. Perhaps it is the latter type of nonconstructive proof objected to in [15].

The importance of the range restriction is to emphasize that the constructed function is not complex because it takes a long time to print the results. The removal of the restriction to $\{0,1\}$ valued functions allows for a much simpler argument than the one given below ([3]). In [17] and [1] it is shown that arbitrarily complex functions can be made arbitrarily sparse as well. A program for the constant zero function will quickly compute such a complex function with a density of errors that decreases as the complex function becomes more sparse. Below it is shown that there are arbitrarily sparse $\{0,1\}$ valued functions such that any program computing a finite variant of the constructed function is arbitrarily complex. Consequently, there are sparse functions which are so complex that any program computing a close approximation to it must also be arbitrarily complex. It is shown that other generalizations of the arbitrarily difficult to compute functions result do not hold, making the result below best possible.

In most "reasonable" programming systems with "natural" complexity measures finite variants of an almost everywhere arbitrarily complex function are easily seen to be almost everywhere complex. Consider time on a *Random Access Machine* ([5]). If a finite variant of an arbitrarily complex function has a fast program, then that fast program can be patched with a finite table so as to compute the complex function. The patched program (in the RAM model) will have the same almost everywhere run time characteristics as the original program for the finite variant, modulo a constant factor required to search the finite table. Hence, the patched program computes the arbitrarily complex function quickly, a contradiction.

The only twist in the above argument is that considering *all* finite variants entails arbitrarily large finite patches. As the size of the patch grows, so does the search time, although not as fast as the size of the patch. The unbounded constant problem plagues similar proofs for complexity measures based on reusable resources, like Turing machine space. Extrapolating the result to other measures via recursive relatedness adds another factor to reckon with, one that typically increases complexity.

The "natural" aspect of RAM time exploited in the argument above was that patching in a finite table of exceptional values does not alter almost everywhere run time behavior very much. Some complexity measures are very sensitive to finite table patching. Since the canonical representations of all the finite functions can be recursively enumerated ([22]), it is possible to effectively enumerate all the finite variants of some (arbitrary) recursive function. With the latter enumeration in hand, an abstract complexity measure can be defined where all the programs enumerated have zero complexity. Hence, even if the original recursive function is very complex *all* its other finite variants are easy in the concocted measure. The result proved below holds for all acceptable programming systems and complexity measures. Furthermore, the argument entails none of the tweaking and twiddling necessary with a measure dependent proof.

One application of the finite table patching technique is to create "padded" versions of programs. If the table patched into a program does not alter the function computed by the original program then the patched program is a slightly larger version of the original. Furthermore, if the patching doesn't add any complexity, then the padded version will have approximately the same complexity as the original. In fact, padding can be accomplished without altering the complexity in several models of computation according to "natural" complexity measures. Examples of such models are given below, where it is shown that the additional complexity of proving the extended version of the Rabin–Blum result presented herein is due entirely to the existence of complexity measures where padding may also increase the complexity of the program.

Perhaps the main contribution of this paper is the isolation of one of the many pathologies admitted by an axiomatic approach to program complexity. The *only* reason the fully general version of the main result below is any harder than Blum's original argument is because of the existence of complexity measures where padding alters the complexity of the program being padded. Since program padding without altering the complexity is possible in all commonly agreed upon as natural complexity measures, the difficulty of the argument below is due entirely to the existence of pathological complexity measures. Consequently, this paper is of a technical nature relevant to the historical discussion of general axioms for program complexity. In this light, we have found an attribute of programming systems and their related complexity measures (the relationship between padding and program complexity) that makes the proof a fundamental result more difficult precisely when the particular incarnation of the attribute is deemed "unnatural".

2 *Notation*

Let $\phi_0, \phi_1, \ldots$ be an arbitrary acceptable programming system ([16]). Throughout, $\Phi_0, \Phi_1, \ldots$ denotes an arbitrary abstract complexity measure on $\phi_0, \phi_1 \ldots$ ([2]). The quantifiers "$\overset{\infty}{\forall}$" and "$\overset{\infty}{\exists}$" mean "for all but finitely many" and "for infinitely many", respectively. f, g, and h will denote recursive functions. f is a *finite variant* of g (written: $f =^* g$) iff $\{x \mid f(x) \neq g(x)\}$ is finite. The cardinality of the set of *support points* of a function f will be denoted by $S(f, n)$ and is defined to be the cardinality of $\{x < n \mid f(x) \neq 0\}$. When the particular f is clear from context, $S(f, n)$ will be written simply as $S(n)$. A function f is *sparse* iff $(\forall \epsilon > 0)(\exists n_0)(\forall n \geq n_0)[S(n)/n < \epsilon]$. Another notion of sparseness that is easier to use in recursion theoretic diagonalization arguments is to say that f is *g-sparse* iff $(\forall x)[f(x) \neq 0 \Rightarrow f(y) = 0$ for all y's such that $x < y \leq x + g(x)]$.

*3 **Results*** The first result relates the two notions of sparseness defined above. The following lemma is a slight modification of a result claimed without proof in [1] and [17].

Lemma 1 *If g is such that for all x, $g(x) \geq x$ and f is g-sparse, then f is sparse.*

Proof: Suppose f and g satisfy the hypothesis. If f is of finite support, then clearly f is sparse, so suppose that f has infinitely many support points. Suppose that $f(x) \neq 0$ and $S(x) > 1$ for some value x. Choose $y > x$ least such that $f(y) \neq 0$. It suffices to show $\lfloor x/S(x) \rfloor < \lfloor y/S(y) \rfloor$, as that would imply that for any n there is a z such that $1/(z/(S(z)) = S(z)/z < 1/n$. Since f is g-sparse we know that $y \geq x + g(x)$ and $S(x) + 1 = S(y) = S(x + g(x))$. Suppose by way of contradiction that

$$\left\lfloor \frac{x}{S(x)} \right\rfloor \geq \left\lfloor \frac{y}{S(y)} \right\rfloor;$$

then,

$$\left\lfloor \frac{x}{S(x)} \right\rfloor \geq \left\lfloor \frac{y}{S(x)+1} \right\rfloor.$$

Since

$$\frac{x}{S(x)} = \frac{\dfrac{xS(x)+x}{S(x)}}{S(x)+1}$$

either

$$\frac{xS(x)+x}{S(x)} > y \tag{1}$$

or

$$y - \frac{xS(x)+x}{S(x)} < S(x) + 1. \tag{2}$$

Equation (1) implies that $x + x/S(x) > y \geq x + g(x)$. Hence, $x/S(x) > g(x)$ so $x/S(x) > x$, a contradiction. Equation (2) and the choice of y imply $x + g(x) - x + x/S(x) < S(x) + 1$. Reducing, we find that $x \leq g(x) < S(x) - x/S(x) + 1$, so $x \leq S(x) - x/S(x)$, contradicting the bounding of $S(x)$ by x from the definition of S.

Now we come to the main result.

Theorem 2 *For any recursive functions g and h, with $g(x) \geq x$ for all x, there is a $\{0,1\}$ valued g-sparse recursive function f such that, for any program i, if ϕ_i is a finite variant of f then $\Phi_i(x) > h(x)$, for all but finitely many x.*

Proof: Suppose g and h are recursive functions with $g(x) \geq x$ for all x. By the Lemma, it suffices to construct a $\{0,1\}$ valued g-sparse recursive f such that any finite variant of f is h complex. The construction proceeds by finite extension. The essential idea is to count the cancellations performed against each program and use the count to give priority to some potential cancellation ensuring that all fast programs are diagonalized against infinitely often. The cancellations will be kept track of via counters. Counter C_i is initialized to i upon entry into stage i, thereby giving diagonalization "priority" to programs less than i. At stage s below, only counters C_i with $i \leq s$ will be in use. For $i \leq s$, C_i^s denotes the value of counter C_i on entry into stage s. At a given stage, the construction may have to choose one program for cancellation amongst several choices. Program i is *more wanting* (of diagonalization) than j at stage s if either $C_i^s < C_j^s$ or $C_i^s = C_j^s$ and $i < j$. Clearly, at any stage, amongst any finite set of programs, there is a single most wanting program. f^s denotes the finite initial segment of f constructed prior to stage s, $f^0 = \emptyset$, x^s denotes the least integer not in the domain of f^s.

Stage s. Initialize C_s by setting $C_s^s = s$. If $\Phi_i(x^s) > h(x^s)$ for all $i \leq s$ then set $f^{s+1} = f^s \cup \{(x^s,0)\}$ and go to stage $s + 1$. Otherwise, choose i the most wanting of $\{i | i \leq s \text{ and } \Phi_i(x^s) \leq h(x^s)\}$. Let $y = x^s$. While there is a z such that $y < z \leq y + g(y) + 1$ and a j more wanting than i such that $\Phi_j(z) \leq h(z)$ set $y = z$ and $i = j$. Set $f^{s+1} = f^s \cup \{(y, 1 \dot{-} \phi_i(y))\} \cup \{(z,0) | z \neq y \text{ and } x^s \leq z \leq y + g(y) + 1\}$. Set $C_i^{s+1} = C_i^s + 1$ and $C_j^{s+1} = C_j^s$ for all $j \leq s$ with $j \neq i$ and go to stage $s + 1$.
End stage s.

Once an initial value of i is chosen in stage s, it can only be replaced by a more wanting program. Since, for any value n, there are only finitely many counters that ever have value less than n, the initial choice of i can be replaced only finitely often. Hence, the construction is effective and f is recursive. Clearly, f has range $\{0,1\}$. By the construction, every time $f(z)$ is set equal to 1 for some z, $f(y)$ is set equal to 0 for all y's such that $z < y \leq g(z) + z + 1$. Hence, f is g-sparse. To complete the proof, suppose by way of contradiction that $\phi_k =^* f$ and $(\overset{\infty}{\exists} x)[\Phi_k(x) \leq h(x)]$. Since every diagonalization against program k is accompanied by an increment of C_k, the counter C_k is incremented only finitely often, as otherwise ϕ_k would not be a finite variant of f. Choose stage $s \geq k$ such that $(\forall t > s)[C_k^s = C_k^t]$. Choose stage $t > s$ such that $(\forall j \leq C_k^s)$ [either $C_k^s < C_j^t$ or $(\forall u > t)[C_j^t = C_j^u]]$. Stage t is the least point in the construction where all the counters that ever have value $\leq C_k^s$ have been incremented to be greater than C_k^s or are never incremented at or past stage t. Choose the least x such that $x \notin \text{domain} f^t$ and $\Phi_k(x) \leq h(x)$. x is guaranteed to exist since there are, by supposition, infinitely many such x's. Choose stage $u \geq t$ such that x is placed in domain f at stage u. By the choice of t, k is the most wanting of any program ever convergent on any value z not in the domain of f^u in less than or equal to $h(z)$ steps. The while loop of stage u will find k most wanting and set $i = k$ and increment C_k, contradicting the choice of s.

The above theorem can be seen to be best possible by examining the consequences of relaxing some of the conditions on the desired arbitrarily complex function f. There can be no recursive $\{0,1\}$ valued function f, sparse or otherwise, such that if $\phi_i = f$ on infinitely many arguments then $\Phi_i(x) > t(x)$ for even infinitely many x. To see this, note that for any such f, either $\lambda x[0]$ or $\lambda x[1]$ will equal f infinitely often and both those functions are computable in constant time in natural measures. However, for any time bound given by a recursive function h one can find a recursive function f such that any program computing f correctly on infinitely many arguments cannot have complexity bounded by h almost everywhere. The range of the function f can be restricted, but not to $\{0,1\}$.

Proposition 3 *For any recursive function h there is a recursive function f such that $f(x) \leq x$, for all x and for any program i if $\phi_i = f$ on infinitely many arguments then $\Phi_i > h$ on infinitely many arguments.*

Proof: Let h be given. Define $f(x)$ to be the least number not in $\{\phi_i(x) \mid i < x$ and $\Phi(x) \leq h(x)\}$. For any x, $f(x) \leq x$ by the pigeon hole principle. Suppose by way of contradiction that program i computes f on infinitely many arguments and $(\overset{\infty}{\forall} x)\,[\Phi_i(x) \leq h(x)]$. Choose $x > i$ largest such that $\Phi_i(x) > h(x)$. Then, $(\forall y > x)\,[f(y) \neq \phi_i(x)]$, a contradiction.

The function f of the previous proposition cannot be made sparse since any program for $\lambda x[0]$ computes any sparse function on infinitely many arguments and there are constant time programs for all constant functions. Similarly, the range of the f of the previous proposition cannot be restricted to any finite set since there would then be at least one value appearing infinitely often in the range of f.

4 Necessity One question that comes to mind is whether or not the construction of the previous section was actually necessary. Sometimes a more baroque construction is used to strengthen a result and then, later, the original construction is shown to be adequate to prove the strengthened result. For example, after Yates found a maximal T-complete set ([24]), Lachlan (see [19]) showed that Friedberg's ([6]) original maximal set was also T-complete. For many complexity measures, like Turing machine time, the diagonalization against fast programs also, implicitly, diagonalizes against the finite variants of the fast programs as well. That is because every program has infinitely many syntactically different, semantically identical padded versions that have same complexity properties. These variants are formed by adding useless quintuples to the original Turing machine. Suppose by way of contradiction that e is a fast program (with respect to Turing machine time) computing a finite variant of the function defined by the Blum construction. Let p_i be the same as program e with i useless quintuples added. The procedure to diagonalize against fast programs will define a function differing from ϕ_{p_i} on argument, say, x_i, where all the x_i's are distinct. Then the function produced will differ from ϕ_e on infinitely many arguments, a contradiction. The same argument will hold for any complexity measure that is not sensitive to superfluous padding.

As it turns out, the abstract version of Rabin's result given by Blum ([2]) does not always yield a function all of whose finite variants are arbitrarily complex with respect to an arbitrary complexity measure. Since the argument needed to see this depends on details of the proof, Blum's proof is reproduced below.

Theorem 4 (Rabin-Blum) *For all recursive functions h there is a $\{0,1\}$ valued recursive function f such that for all programs i, if $\phi_i = f$ then $\Phi_i(x) \geq h(x)$, for almost all x.*

Proof: The desired f is constructed via a cancellation argument in effective stages of finite extension below. Let h be as in the hypothesis.

Stage s. Let i be the least *uncanceled* program such that $i \leq s$ and $\Phi_i(s) \leq h(s)$. If there is no such i, set $f(s) = 0$. Otherwise, set $f(s) = 1 \dot{-} \phi_i(s)$ and cancel program i. Go to stage $s + 1$.
End stage s.

Suppose by way of contradiction that $\phi_i = f$ and $\Phi_i(x) \leq h(x)$ for infinitely many x's. Choose the least stage s such that $s \geq i$, $\Phi_i(s) \leq h(s)$ and all the programs $j < i$ that are ever canceled are canceled before stage s. Then, $f(s)$ will be made $\neq \phi_i(s)$ at stage s, a contradiction.

To complete the argument that the construction of the previous section was actually necessary, it suffices to exhibit a complexity measure and a fast finite variant of the function that results from applying the above construction to the concocted measure. The function f, constructed above, depends on the complexity measure Φ and the bounding function h. h is an arbitrary recursive function which is fixed for the remainder of this argument. In what follows we will confuse with Φ (the complexity measure) an index for a recursive function computing the predicate guaranteed to exist by the axioms for complexity measures. With this slight abuse of notation we have that $\phi_\Phi(i,x,y) = 1$ iff $\Phi_i(x) = y$. Let $\alpha(\Phi)$ be the program sketched above which computes the function f constructed above when Φ is used as the complexity measure. $\alpha(\Phi)$ can be effectively obtained from indices for Φ and h.

Next, a transformation on complexity measures is defined. Let $\mathcal{Z}(\Phi,e)$ denote the measure that is identical with the measure Φ except that program e has complexity zero everywhere:

$$\phi_{\mathcal{Z}(\Phi,e)}(i,x,y) = \begin{cases} 1, & \text{if } e = i \text{ and } y = 0; \\ 0, & \text{if } e = i \text{ and } y > 0; \\ \phi_\Phi(i,x,y), & \text{if } e \neq i. \end{cases}$$

If ϕ_e is a recursive function, then $\mathcal{Z}(\Phi,e)$ is also a complexity measure. In any event the index $\mathcal{Z}(\Phi,e)$ can be effectively calculated from indices for Φ and e. $\phi_{\mathcal{Z}(\Phi,e)}$ is always a recursive $\{0,1\}$ valued function. If $\mathcal{Z}(\Phi,e)$ is *not* a complexity measure, then program $\alpha(\mathcal{Z}(\Phi,e))$ may not compute a recursive function since, for some i and s, it may happen that $\mathcal{Z}(\Phi,e)_i(s) \leq h(s)$ when $\phi_i(s)$ is undefined. Next, the appropriate finite variant is defined. By the recursion theorem ([13]) there is a program e such that:

$$\phi_e(x) = \begin{cases} 0, & \text{if } e \leq x \leq 2e; \\ \phi_{\alpha(\mathcal{Z}(\Phi,e))}(x), & \text{otherwise.} \end{cases}$$

Referring to the proof of Theorem 4, consider the program $\alpha(\mathcal{Z}(\Phi,e))$. Program e is not tested before stage e, so $\phi_{\alpha(\mathcal{Z}(\Phi,e))}(x) = \phi_{\alpha(\Phi)}(x)$ for all $x < e$. Program e will be considered for cancellation starting with stage e. Clearly, $\mathcal{Z}(\Phi,e)_e(x) = 0 \leq h(x)$ for all x. Since there are only e programs with higher cancellation priority than e, program e will be canceled no later than stage $2e$. After stage $2e$, program e will never be considered for cancellation and the program $\mathcal{Z}(\Phi,e)$ will behave precisely like the program Φ on all argument values used by program $\alpha(\mathcal{Z}(\Phi,e))$. Consequently, $\phi_{\alpha(\mathcal{Z}(\Phi,e))}$ is a recursive function. Hence, ϕ_e is also recursive and $\mathcal{Z}(\Phi,e)$ is a complexity measure.

$\phi_{\alpha(\mathcal{Z}(\Phi,e))}$ is an almost everywhere h-complex recursive function with respect to the complexity measure $\mathcal{Z}(\Phi,e)$. ϕ_e is a recursive finite variant of $\phi_{\alpha(\mathcal{Z}(\Phi,e))}$ by construction. Furthermore, ϕ_e has complexity 0 everywhere according to the measure $\mathcal{Z}(\Phi,e)$. It is interesting to note that if padding can be accomplished without altering complexity according to the measure Φ, then according to the measure $\mathcal{Z}(\Phi,e)$, every program, except e, can be padded without changing its complexity.

5 ***Conclusions*** It was shown above that there are arbitrarily sparse $\{0,1\}$ valued recursive functions f such that any program computing a finite variant of f is arbitrarily complex. A most general notion of sparseness was used. There is no $\{0,1\}$ valued recursive function f such that any function g with $\{x \mid f(x) = g(x)\}$ infinite is arbitrarily complex. Consequently, the generalization of Rabin's result proved above is the strongest possible. Our result was shown, in a sense, not to be a consequence of the original Rabin–Blum construction. By way of contrast, Case (private communication) has shown how to obtain, from a recursive function h, for an arbitrary complexity measure, another recursive function H such that if f is a recursive function that is almost everywhere H complex, then f and all its finite variants will be almost everywhere h complex.

The main construction above was noted to be unnecessary for complexity measures like Turing machine time. In fact, a new construction to diagonalize against fast finite variants is unnecessary for any complexity measure where padding can be achieved without altering the complexity of the program. This seems to be the case for many "natural" measures like time or space for Turing machines or Random Access machines. Padding without increased complexity is also possible in the copy measure for data flow which seems to be different from both time and space ([18]). Riccardi evidenced the fundamental nature of padding with respect to computation when he showed that padding in conjunction with a suitable form of recursion (as used above) could be used instead of composition (or s_1^1) in the axiomatization of acceptable programming systems ([21]). Riccardi's result was strengthened by Royer ([23]). Perhaps padding without increased complexity is one of the properties needed to formalize the notion of "natural" complexity measures sought after by several authors ([9], [14], [25]).[1]

NOTE

1. The existence of functions so complex that all of their finite variants are also complex was first conjectured by J. Case. This document was enhanced as a consequence of continued conversations with J. Case. A. Nerode was the first to ask if Blum's construction was sufficient to obtain the main result above. J. Owings pointed out some errors in earlier versions of this paper and made other valuable comments. W. I. Gasarch pointed out the T-complete maximal set example. The referee made several comments that resulted in an improved exposition. Financial support came from NSF (MCS 8301536) and NSA (MDA904-85-H-002). Computer time was provided by the Department of Computer Science at the University of Maryland.

REFERENCES

[1] Adleman, L. and M. Blum, *Inductive Inference and Unsolvability*, Department of Electrical Engineering and Computer Science and the Electronics Research Laboratory, University of California, Berkeley, 1975.

[2] Blum, M., "A machine-independent theory of the complexity of recursive functions," *Journal of the Association for Computing Machinery*, vol. 14 (1967), pp. 322-336.

[3] van Emde Boas, P., *Abstract Resource-Bound Classes*, Ph.D. Dissertation, University of Amsterdam, 1974.

[4] Case, J. and C. Smith, "Comparison of identification criteria for machine inductive inference," *Theoretical Computer Science*, vol. 25 (1983), pp. 193-220.

[5] Elgot, C. C. and A. Robinson, "Random-access stored-program machines, an approach to programming languages," *Journal of the Association for Computing Machinery*, vol. 11 (1964), pp. 365-399.

[6] Friedberg, R., "Three theorems on recursive enumeration," *The Journal of Symbolic Logic*, vol. 23 (1958), pp. 309-316.

[7] Fulk, M., "A note on a.e. h-complex functions," *Journal of Computer and Systems Science*, to appear..

[8] Gill, J. and M. Blum, "On almost everywhere complex recursive functions," *Journal of the Association for Computing Machinery*, vol. 21 (1974), pp. 425-436.

[9] Hartmanis, J., "On the problem of finding natural complexity measures," pp. 95-103 in *Proceedings of the Symposium on Mathematical Foundations of Computer Science*, Bratislava, Czechoslovakia, 1973.

[10] Hartmanis, J. and R. E. Stearns, "On the computational complexity of algorithms," *Transactions of the American Mathematical Society*, vol. 117 (1965), pp. 285-306.

[11] Joseph, D. and P. Young, "Independence results in computer science?" *Journal of Computer and Systems Science*, vol. 23 (1981), pp. 205-222.

[12] Joseph, D. and P. Young, "A survey of some recent results on computational complexity in weak theories of arithmetic," *Annales Societatis Mathematicai Polonae*, vol. 8 (1985), pp. 103-121.

[13] Kleene, S., "On notation for ordinal numbers," *The Journal of Symbolic Logic*, vol. 3 (1938), pp. 150-155.

[14] Landweber, L. and E. Robertson, "Recursive properties of abstract complexity classes," *Journal of the Association for Computing Machinery*, vol. 19 (1972), pp. 293-308.

[15] Lipton, R., "Model theoretic aspects of computational complexity," pp. 193-200 in *Proceedings of the 19th Symposium on Foundations of Computer Science*, Ann Arbor, Michigan, 1978.

[16] Machtey, M. and P. Young, *An Introduction to the General Theory of Algorithms*, North-Holland, New York, 1978.

[17] Meyer, A. and E. McCreight, "Computational complex and pseudo-random zero-one valued functions," pp. 19-42 in *Theory of Machines and Computations*, ed., Z. Kohavi and A. Paz, Academic Press, New York, 1971.

[18] Motteler, H. and C. Smith, "Complexity measures for data flow models," *International Journal of Computer and Information Sciences*, vol. 14 (1985), pp. 107-122.

[19] Odifreddi, P., "Strong reducibilities," *Bulletin of the American Mathematical Society*, vol. 4 (1981), pp. 37-86.

[20] Rabin, M., "Degree of difficulty of computing a function," Hebrew University Technical Report 2, 1960.

[21] Riccardi, G., "The independence of control structures in abstract programming systems," *Journal of Computer and Systems Sciences*, vol. 22 (1981), pp. 107-143.

[22] Rogers, H., Jr., *Theory of Recursive Functions and Effective Computability*, McGraw Hill, New York, 1967.

[23] Royer, J., *A Connotational Theory of Program Structure*, Lecture Notes in Computer Science 273, Springer-Verlag, New York, 1987.

[24] Yates, C. E. M., "Three theorems on the degree of recursively enumerable sets," *Duke Mathematics Journal*, vol. 32 (1965), pp. 461-468.

[25] Young, P., "A note on 'axioms' for computational complexity and computation of finite," *Information and Control*, vol. 19 (1971), pp. 377-386.

Department of Computer Science

and

Institute for Advanced Computer Studies
The University of Maryland
College Park, Maryland 20742