A Note on Almost-Everywhere-Complex Sets and Separating Deterministic-Time-Complexity Classes

J.G. Geske, D.T. Huynh, and J.I. Seiferas

Technical Report 270
January 1989

UNIVERSITY OF
ROCHESTER
COMPUTER SCIENCE

A Note on Almost-Everywhere-Complex Sets and Separating Deterministic-Time-Complexity Classes

John G. Geske[†]
Department of Computer Science
Michigan State University
East Lansing, Michigan 48825
geske@cpswh.cps.msu.edu

Dung T. Huynh[‡]
Computer Science Department
University of Texas at Dallas
Richardson, Texas 75083-0688
huynh@utd.edu

Joel I. Seiferas
Computer Science Department
University of Rochester
Rochester, New York 14627
joel@cs.rochester.edu

January 3, 1989

Abstract. For each time bound T: {input strings} $\rightarrow$ {natural numbers} that is some machine's exact running time, there is a $\{0,1\}$-valued function f_T that can be computed within time proportional to T, but that cannot be computed within any time bound T' that is infinitely often significantly smaller than T ($T' \neq \Omega(T)$, typically). Equivalently, every algorithm to compute f_T requires time T' on almost every input if T' is almost everywhere significantly smaller than T ($T' = o(T)$, typically).

1. Introduction

The construction by diagonalization of $\{0,1\}$-valued functions (or, equivalently, of the sets or "languages" they characterize) of accurately known complexity was an early focus of complexity theory. Although the results are generally assumed to be complete and well understood, some misunderstandings and confusion still endure, especially in the case of time complexity. With this note, we aim to clear up the misconceptions and to present a definitive proof of the tightest such construction.

Part of the confusion stems from the (obvious) fact that, for functions, simplicity and complexity *are not complementary notions.* That a function cannot always

[†] This author was supported in part by the National Science Foundation under grant DCR 88-11996.

[‡] This author was supported in part by the National Science Foundation under grant DCR 85-17277.

be computed in time n^2, for example, does not imply that it *always* requires time n^2, but only that it *sometimes* does.

Since, in most natural models of computation, programs can be patched to run fast on any finite number of inputs (for *un*natural models, see [Smith, 1988]), the appropriate notions above are not quite "always" and "sometimes". Instead, they are "always, except in some finite number of cases" and "in infinitely many cases", respectively. The respective adverbial phrases we use when discussing functions and their complexities are *almost everywhere* and *infinitely often*. The "complexity classes" we usually define are sets of functions (or of the sets characterized by them) that are almost everywhere *easy*. (Stearns [1988] concedes that "simplicity classes" might have been a less misleading choice for the terminology.) *Non*membership in a complexity class implies only "infinitely-often complexity", and not the stronger "almost-everywhere complexity" [Gill and Blum, 1974].

An additional, more artificial impediment to clarity in the construction of almost-everywhere-complex functions has been the widespread acceptance of the convention to measure complexity in terms of input "length" (taking the worst case), rather than in terms of the particular input. Since, in a typically robust model, we expect to concede the possibility of constant-factor speedup, the tightest separation results easily conceivable in these terms are of the form, "There is a function computable within time $T(n)$, but not computable within time $T'(n)$ if $g(T'(n)) \neq \Omega(T(n))$," for some barely linear function g. Such results are reported by Hartmanis and Stearns [1965] (where $g(t) = t^2$), by Hennie and Stearns [1966] (where $g(t) = t \log t$), and by Cook and Reckhow [1973] (where $g(t) = t$), where the conditions on T' are stated in more classical terms: "$\liminf_{n\to\infty} \frac{g(T'(n))}{T(n)} = 0$." Although it follows from such a result that the witness function cannot be computed quickly, in the worst-case sense, on any infinite set *of lengths*, it does not follow that it cannot be computed quickly on any infinite set *of particular inputs*—unless the input alphabet is a singleton, so that there is just one input of each length. It follows that a key to the clean construction of almost-everywhere-complex $\{0,1\}$-valued functions is either to work with a single-letter input alphabet, or, equivalently, to measure complexity in terms of particular inputs, the approach in fact used successfully by Meyer and McCreight for *space* complexity [Meyer and McCreight, 1971], and by Rabin [1960] and Blum [1967] before them for coarser and more general results. When this insight has been overlooked, the result has been awkward constructions and unnecessarily weak results.

Even armed with the latter insight, one finds tight time diagonalization more difficult than tight space diagonalization, partly because space can be used over and over again, in a leisurely manner, whereas time apparently cannot. For the purposes of diagonalization, however, there is a trick that has essentially the effect of making time reusable (see the proof of Theorem 2′ in [Seiferas, Fischer, and Meyer, 1978]). This leads to tight diagonalization results such as those mentioned in Appendix I of [Seiferas, 1974] and those spelled out below.

2. The Generic Construction

The diagonalization strategy is the same "looking back" or "delayed diagonalization" strategy that is usually used (including in the works already cited) to construct almost-everywhere-complex functions. To avoid obscuring the essential simplicity and versatility of the construction, we will present it at a high level, leaving the machine model unspecified for now. For a particular machine model, at

least *some* concrete implementation will usually be obvious. In Section 3, we will point out some of the most efficient implementations.

Let T be the exact running time of any particular machine or algorithm that always halts. (Such a time bound is often called "honest" or "constructible".) Within time $\mathcal{O}(T(x))$ (shorthand for "at most some constant times" $T(x)$), we compute a $\{0,1\}$-valued function of x by downward diagonalization, as follows:

1. Calculate $T(x)$.
2. For $i = 1, 2, 3, \ldots$, successively, do the following:
   (a) Spend $\lfloor T(x)/2^i \rfloor$ steps recursively reviewing computations by this program on inputs preceding x in lexicographical order.
   (b) If i was not discovered in step 2(a) to have been "cancelled" on any earlier input, then "attack" i as follows: Spend $\lfloor T(x)/2^i \rfloor$ steps efficiently simulating the i-th program on input x. If a halt is discovered, then differ from the outcome, "cancel" i, and halt.
3. If this step is reached, then (arbitrarily) output 0, and halt without cancelling any integer.

Claim 1. *Under the right assumptions, this program can be made to run within time $\mathcal{O}(T(x))$.*

Claim 2. *Under the right assumptions, no program can compute the same function within any time bound $T'(x)$ that is infinitely often significantly smaller than $T(x)$.*

The proof of Claim 1 is straightforward for any reasonable programming language or model of computation. Note that sum of the successively halved time allocations does converge to only some constant times the first one.

The proof of Claim 2 is by contradiction. For the sake of argument, suppose the k-th program *does* compute the same function within a time bound $T'(x)$ that is infinitely often significantly smaller than $T(x)$. Then k must never get cancelled. For each $j < k$ that *does* get cancelled, let $h(j)$ be the number of steps it takes the recursive review to discover the "earliest" cancellation of j (i.e., the cancellation on the lexicographically earliest input). Consider an input x so long that every $j < k$ that *ever* gets cancelled does so on some input shorter than x, such that $\lfloor T(x)/2^k \rfloor \geq h(j)$ for each such j, and such that $\lfloor T(x)/2^k \rfloor$ is enough time for the efficient simulation of the k-th program on input x. Consider the computation by the program on this x. Note that the program does reach stage $i = k$, and that it does then successfully cancel k, a contradiction.

Note that, for reasonable models of computation, the main issue above is the time required for the "clocked universal simulation" in step 2(b). The efficiency of the recursive review does not matter.

3. Concrete Implementations

As a first application, consider the random-access machine of Cook and Reckhow [1973]. Cook and Reckhow show that, except at exceptionally low complexity levels,[1] clocked universal simulation requires only linear time, regardless of cost criterion. Therefore, we get the tightest possible result:

Theorem 1. *If $T(x)$ is the exact running time of some random-access machine,[1] then there is a $\{0,1\}$-valued function computable by a random-access machine within*

[1] Under the unit- and logarithmic-cost criteria, respectively, we must have $T(|x|) = \Omega(|x|)$ and $T(|x|) = \Omega(|x| \log |x|)$.

time $\mathcal{O}(T(x))$, but not computable by any random-access machine within time $T'(x)$ unless $T'(x) = \Omega(T(x))$.

This tightens the result of Cook and Reckhow, who measure complexity in terms of length, and who work with a binary alphabet, thus not getting an almost-everywhere-complex characteristic function. The following reformulation makes it clear that we do get an almost-everywhere-complex function.

Corollary 1. *If f_T is the witness function in Theorem 1, and if $T'(x) = o(T(x))$, then every random-access machine that computes f_T uses time exceeding $T'(x)$ almost everywhere.*

(The little-o notation is shorthand for "almost everywhere less than any positive fraction of", so that the condition above is just "$\lim_{|x|\to\infty} T'(x)/T(x) = 0$" in more classical terms.) Each result below has an analogous reformulation.

For Turing machines, the most obvious way to perform clocked universal simulation is to endow the simulator with one more tape than any simulated machine. This yields a mixed result:

Proposition 1. *If $T(x)$ is the exact running time of some $(k+1)$-tape Turing machine, then there is a $\{0,1\}$-valued function computable by a $(k+1)$-tape Turing machine within time $\mathcal{O}(T(x))$, but not computable by any k-tape Turing machine within time $T'(x)$ unless $T'(x) = \Omega(T(x))$.*

There is an obvious patch for Turing machines all with one fixed number of tapes: For a clocked universal simulation, maintain a counter in binary on one of the tapes, and shift the whole counter whenever that tape's head shifts. The extra time is logarithmic in the counter's contents, and hence certainly in $T(x)$, on each step. We state the result only for single-tape machines:

Theorem 2. *If $T(x)$ is the exact running time of some single-tape Turing machine, then there is a $\{0,1\}$-valued function computable by a single-tape Turing machine within time $\mathcal{O}(T(x))$, but not computable by any single-tape Turing machine within time $T'(x)$ unless $T'(x)\log T'(x) = \Omega(T(x))$.*

Like Theorem 1, this tightens an early infinitely-often complexity result [Hartmanis, 1968]. It also properly subsumes an unpublished almost-everywhere complexity result obtained by *upward* diagonalization [Fich and Goldwasser, 1981].

For Turing machines with a larger fixed number of tapes, we can do even better, thanks to a result of Fürer [1984]. By this result, for each $k \geq 2$, k tapes suffice for a linear-time simulation of k tapes *and a counter*, so that clocked universal simulation of k-tape machines *by* k-tape machines requires only linear time. This yields the best possible result again:

Theorem 3. *For $k \geq 2$, if $T(x)$ is the exact running time of some k-tape Turing machine, then there is a $\{0,1\}$-valued function computable by a k-tape Turing machine within time $\mathcal{O}(T(x))$, but not computable by any k-tape Turing machine within time $T'(x)$ unless $T'(x) = \Omega(T(x))$.*

Combining this with the well known simulation of any multitape machine by a two-tape machine [Hennie and Stearns, 1966; Hopcroft and Ullman, 1979 (Section 12.2)], we get a good result for general multitape Turing machines:

Theorem 4. *For $k \geq 2$, if $T(x)$ is the exact running time of some k-tape Turing machine, then there is a $\{0,1\}$-valued function computable by a k-tape Turing machine within time $\mathcal{O}(T(x))$, but not computable by any multitape Turing machine within time $T'(x)$ unless $T'(x)\log T'(x) = \Omega(T(x))$.*

This finally tightens the early infinitely-often complexity results of Hartmanis, Hennie, and Stearns [Hartmanis and Stearns, 1965; Hennie and Stearns, 1966], and

its reformulation (analogous to Corollary 1 above) properly subsumes the almost-everywhere versions reported in [Geske and Huynh, 1986] and [Geske, Huynh, and Selman, 1987].

For Turing machines with multidimensional tapes, the results analogously obtained are not as tight. The most efficient clock-maintenance technique that is known to adapt is Paul's [1979], and the most efficient simulations among the variants of the multidimensional Turing machine are Loui's [1982, 1984].

4. Further Discussion

Balcázar and Schöning [1985] relate almost-everywhere complexity to a notion of relative "bi-immunity". An infinite language L is *bi-immune for* the language class $\mathcal{C}$ if neither L nor its complement has an infinite subset that belongs to $\mathcal{C}$. If $\mathcal{C}$ is the *complexity class* of languages characterizable within an appropriate running time $T(x)$, then bi-immunity of L for $\mathcal{C}$ turns out to mean *precisely* that every machine characterizing L uses time exceeding $T(x)$ almost everywhere. Therefore, Corollary 1 and the corollaries of the other results in Section 3 yield a variety of bi-immune languages. For further developments along this line, see [Geske, 1987].

Sometimes separation results obtained by diagonalization can be tightened by "translational" methods [Ruby and Fischer, 1965; Hopcroft and Ullman, 1979 (Section 12.5)]. It can be shown, for example, that some multitape Turing machine can compute in time $\mathcal{O}(2^n\sqrt{n})$ a $\{0,1\}$-valued function that cannot be computed in time $\mathcal{O}(2^n)$ by any such machine, even though this does not follow from our Theorem 4 above. Unfortunately, however, the translational methods do not yield functions that are complex *almost everywhere*, for the following reason: From a contrary supposition of *non*separation, the techniques derive and chain together enough "translated" versions of the nonseparation supposition to contradict already known separation results. The translated versions chain together successfully because they are all of the form "almost-everywhere easy implies almost-everywhere a little easier". In our setting, however, the contrary supposition and its translated versions would be of the weaker form "almost-everywhere easy implies *infinitely-often* a little easier", so that they *would not* chain together.

Separating *nondeterministic*-time-complexity classes is more challenging. The only respectably tight results [Cook, 1973; Seiferas, Fischer, and Meyer, 1978; Žák, 1983] rely again on translational methods, and hence do not yield almost-everywhere-complex witnesses. There is a need for some new way to produce almost-everywhere-complex sets that bear witness to such tight separation results.

Acknowledgments. We thank Eric Allender, Jim Royer, John Case, Alan Selman, and Peter van Emde Boas for their comments and suggestions.

References

J. L. Balcázar and U. Schöning, *Bi-immune sets for complexity classes*, Mathematical Systems Theory **18**, 1 (June 1985), 1–10.

M. Blum, *A machine-independent theory of the complexity of recursive functions*, Journal of the Association for Computing Machinery **14**, 2 (April 1967), 322–336.

S. A. Cook, *A hierarchy for nondeterministic time complexity*, Journal of Computer and System Sciences **7**, 4 (August 1973), 343–353.

S. A. Cook and R. A. Reckhow, *Time bounded random access machines*, Journal of Computer and System Sciences **7**, 4 (August 1973), 354–375.

F. E. Fich and S. Goldwasser, *Compression and the deterministic time hierarchy*, unpublished manuscript, May 1981.

M. Fürer, *Data structures for distributed counting*, Journal of Computer and System Sciences **28**, 2 (April 1984), 231–243.

J. G. Geske, *On the structure of intractable sets*, Ph. D. Thesis, Iowa State University, Ames, Iowa, 1987.

J. G. Geske and D. T. Huynh, *Hierarchies of almost everywhere complex sets*, Technical Report 86-05, Department of Computer Science, Iowa State University, Ames, Iowa, April 1986.

J. G. Geske, D. T. Huynh, and A. L. Selman, *A hierarchy theorem for almost everywhere complex sets with application to polynomial complexity degrees*, 4th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science **247**, Springer-Verlag, Berlin, 1987, pp. 125–135.

J. Gill and M. Blum, *On almost everywhere complex recursive functions*, Journal of the Association for Computing Machinery **21**, 3 (July 1974), 425–435.

J. Hartmanis, *Computational complexity of one-tape Turing machine computations*, Journal of the Association for Computing Machinery **15**, 2 (April 1968), 325–339.

J. Hartmanis and R. E. Stearns, *On the computational complexity of algorithms*, Transactions of the American Mathematical Society **117**, 5 (May 1965), 285–306.

F. C. Hennie and R. E. Stearns, *Two-tape simulation of multitape Turing machines*, Journal of the Association for Computing Machinery **13**, 4 (October 1966), 533–546.

J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Massachusetts, 1979.

M. C. Loui, *Simulations among multidimensional Turing machines*, Theoretical Computer Science **21**, 2 (November 1982), 145–161.

M. C. Loui, *Minimizing access pointers into trees and arrays*, Journal of Computer and System Sciences **28**, 3 (June 1984), 359–378.

A. R. Meyer and E. M. McCreight, *Computationally complex and pseudo-random zero-one valued functions*, in: Theory of Machines and Computations, Z. Kohavi and A. Paz, editors, Academic Press, New York, 1971, pp. 19–42.

W. J. Paul, *On time hierarchies*, Journal of Computer and System Sciences **19**, 2 (October 1979), 197–202.

M. O. Rabin, *Degree of difficulty of computing a function and a partial ordering of recursive sets*, Technical Report No. 2, Hebrew University, Jerusalem, Israel, April 25, 1960.

S. Ruby and P. C. Fischer, *Translational methods and computational complexity*, IEEE Conference Record on Switching Circuit Theory and Logical Design, 1965, pp. 173–178.

J. I. Seiferas, *Nondeterministic time and space complexity classes*, Ph. D. Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 1974.

J. I. Seiferas, M. J. Fischer, and A. R. Meyer, *Separating nondeterministic time complexity classes*, Journal of the Association for Computing Machinery **25**, 1 (January 1978), 146–167.

C. H. Smith, *A note on arbitrarily complex recursive functions*, Notre Dame Journal of Formal Logic **29**, 2 (Spring 1988), 198–207.

R. E. Stearns, *Juris Hartmanis: The beginnings of computational complexity*, Proceedings Structure in Complexity Theory Third Annual Conference, IEEE Computer Society Press, 1988, pp. 128–134.

S. Žák, *A Turing machine time hierarchy*, Theoretical Computer Science **26**, 3 (October 1983), 327–333.