1978

Remarks on Recursion vs. Diagonalization

Michael Machtey

Report Number:
79-311

Remarks on Recursion vs. Diagonalization

Michael Machtey

Purdue University
Department of Computer Sciences
West Lafayette, Indiana 47907

November, 1977
(retyped July, 1979)

CSD TR 311

ABSTRACT

Results are presented which show precise ways in which recursion rests on very simple computational bases which do not support diagonalization. A method based on recursion and making no use of diagonalization is given for proving lower bounds on computational complexity. Thus the intractability of computational problems such as Presburger arithmetic does not depend on diagonalization.

This research was supported in part by the National Science Foundation under Grant MCS-76-09212.

This note is a preliminary report of continuing research. Its purpose is limited dissemination to interested experts, and it should be regarded and treated accordingly.

Remarks on Recursion vs. Diagonalization

Michael Machtey

Purdue University
Department of Computer Sciences
West Lafayette, Indiana 47907

November, 1977
(retyped July, 1979)

CSD TR 311

Introduction and Preliminaries

We wish to help clarify the distinction between recursion (i.e. self-reference) and diagonalization. Recursion is sometimes regarded as a simpler and more "natural" computational tool. We shall show precise ways in which recursion rests on very simple "computational bases" which do not support diagonalization. We shall also sketch a general and somewhat simplified method for proving i.o. lower bounds on computational complexity. This method is based on recursion through the use of limited halting problems, and it makes no use of diagonalization. Thus we show that the intractability of certain computational problems (e.g. Presburger arithmetic) does not depend on diagonalization - which is used in all previous proofs - but rests instead on the very narrow and natural computational base for recursion.

It is sometimes maintained, particularly in the realms of general computational complexity and recursive function theory,

that self-reference (i.e.. recursion) and diagonalization are very intimately related, if not identical. We hope to dispel any such illusions. Recursion often consists of a single, finite act of self-reference. It has a long history as a logical tool, with an early use generally attributed to Epimenedes [see Paul's "Epistle to Titus" I,12]. Diagonalization involves an ongoing or completed infinitary process. It seems to have originated as a mathematical tool about a century ago in Cantor's celebrated proof of the uncountability of the continuum.

In our current age of computational sophistication, recursion is a common feature basic to most high level programming languages. As a tool it has wide application throughout computer science (e.g. in the theory of program semantics). Diagonalization, although it is based on the common computational phenomenon of universal simulation (i.e. interpreters), seems to have its applications restricted primarily to general complexity theory and recursive function theory (and is regarded by some as being somewhat "contrived"). These are some intuitions and part of a cultural backdrop against which we wish to present our results.

Diagonalization constructions use a universal function (i.e. interpreter) for a programming system - that is, a program u such that $\phi_u(i,x)=\phi_i(x)$ for all programs i and inputs x - or some closely allied form of universal simulation. Moreover, in complexity theory diagonalizations often require that simulation have a small overhead. That is, they require that $\Phi_u(i,x)$ be not

much greater than $\Phi_i(x)$. The intuition that diagonalization and universal simulation are very closely related is substantiated by work of Kozen [4], Machtey, Meyer, and others (as yet unpublished). However, we shall not consider such results here. We are interested instead in the power of recursion, and in the small, natural bases which support it without necessarily supporting universal simulation.

Very general forms of recursion can be justified in programming systems by appealing to the Recursion Theorem. We shall consider a version of the Recursion Theorem as originally formulated by Kleene [3]:

> for every program i there is a program n (which can be found effectively from i) such that $\phi_n(x)=\phi_i(n,x)$ for all inputs x.

This version seems computationally more natural and "simpler" than the fixed point version stated in Rogers [8]:

> for every total recursive function f there is a program n such that $\phi_n=\phi_{f(n)}$.

Kleene's version is sufficient to justify recursive features used in programming languages, and it is the version actually used in nearly all applications in complexity theory and recursive function theory. In acceptable (i.e. general) programming systems - those satisfying the Enumeration and s-m-n Theorems - these two forms of the Recursion Theorem are easily shown to be equivalent. In what follows we shall show a precise sense in the

realm of computational complexity in which the fixed point version is properly stronger than Kleene's.

With most of the definitions and results we shall present there are a variety of precise formulations which are either equivalent or at least all sufficient for the purposes at hand. We shall not indicate here the full extent of this latitude, but restrict ourselves instead to single versions selected to be as simple as possible.

Recursion vs. Diagonalization

One small and natural base sufficient for a programming system to support recursion is that the system be able to handle prefixing (of strings) and simple subroutining, and in addition that the system be able to perform such simple program manipulations on itself. Specifically,

> Proposition: Let $\phi_0,\phi_1,\ldots$ be a programming system containing programs pre and sub such that for all inputs x and y and all programs i and j, $\phi_{pre(x)}(y)=(x,y)$ and $\phi_{sub(i,j)}(x,y)=\phi_i(\phi_j(x),y)$. Then for every program i there is a program n (which can be found easily from i) such that $\phi_n(x)=\phi_i(n,x)$ for all inputs x.

The proof first produces an s-1-1 function similarly to Machtey, Winklmann, and Young [6] and then proceeds with what is essentially Kleene's proof of the Recursion Theorem. It should

be noted that although the hypothesis holds in all acceptable programming systems, the proof does not require the programming system to be acceptable. That is, it makes no use of a universal function. Thus the proposition holds for an extremely wide range of programming systems, including subrecursive systems which do not have universal functions as well as nondeterministic systems. Also, it is extremely easy to verify the hypothesis of the proposition directly for almost any reasonable programming system.

We are interested not just in the base which supports recursion, but also in the base sufficient to have a low overhead for recursion as well. That is, we want $\Phi_n(x)$ to be not much greater than $\Phi_i(n,x)$. One way to accomplish this is to require the overhead for prefixing and subroutining to be low. Specifically,

> Definition: Let $\phi_0,\phi_1,\ldots$ be an acceptable programming system and Φ a Blum complexity measure on it. The measure is called linearly bounded if there are programs pre and sub as in the proposition above and a (positive integer) constant k such that for all x, y, i, and j,
>
> (a) $\Phi_{pre(x)}(y) \leq k|(x,y)|$, and
>
> (b) $\Phi_{sub(i,j)}(x,y) \leq k[\Phi_j(x) + \Phi_i(\phi_j(x),y)]$.

In any reasonable programming system with any reasonable complexity measure and definition of the functions pre and sub,

verification of conditions (a) and (b) is quite simple. Moreover, without malice aforethought it is extremely unlikely that someone would produce a complexity measure which is not at least "almost" linearly bounded (see the last paragraph of the previous section). By adding some fairly straightforward calculations to the proof of the previous proposition, we prove the following:

> Theorem: If Φ is a linearly bounded complexity measure, then for every program i there is a program n (which can be found easily from i) such that for all inputs x,
>
> (a) $\phi_n(x) = \phi_i(n,x)$, and
>
> (b) $\Phi_n(x) \leq k^2[\Phi_i(n,x) + |x|] + k'$,
>
> where k is from the definition above and k' is some other constant.

Note that as with the Recursion Theorem (i.e. the Proposition) above, the proof of this Theorem does not require the programming system to be acceptable. In addition, the proof does not require the measure to be a Blum measure. Specifically, the proof never uses the fact that $\Phi_i(x) \leq y$ is a decidable predicate of i, x, and y. Thus the conclusions hold in a wide variety of programming systems and "measures", including subrecursive and nondeterministic systems. (A somewhat different complexity theoretic subrecursive Recursion Theorem has been proved independently by Alton [1]).

The previous Theorem shows that linearly bounded complexity measures provide a small, natural base for performing recursion with low overhead. This base is not sufficient for diagonalization in the precise sense that linearly bounded complexity measures may require more than linear overhead for universal simulation (if they are capable of universal simulation at all). There are natural linearly bounded measures for which this overhead seems to be at least quadratic. However, here we shall content ourselves with "unnatural" linearly bounded measures with arbitrarily large overhead for universal simulation (as well as for the fixed point form of the Recursion Theorem).

Proposition: For any total recursive function t there is a linearly bounded complexity measure Φ such that

(a) if u is any universal program then there are infinitely many programs i such that $\Phi_u(i,x)>t(\Phi_i(x),x)$ a.e. x, and

(b) there are total recursive functions f such that for any program n with $\phi_n=\phi_{f(n)}$, $\Phi_n(x)>t(\Phi_{f(n)}(x),x)$ for all x.

The proof is by "measure manipulation", and we are indebted to Paul Chew for his assistance with it. Similar techniques also settle closely related questions concerning the complexity of simulation in linearly bounded measures. For example, the complexity of the predicate $\Phi_i(x)\leq y$ can be made either very small or very large, independently of the complexity of universal functions.

We close this section with a brief comment that linearly bounded measures are an important example of what we call "structured" complexity measures. That is, they are measures which are required to reflect program structure to at least some minimal extent. One goal of studying structured measures is to find small, natural restrictions can be placed on measures to guarantee that they exhibit various important complexity theoretic properties which occur in "natural" measures. Another example of structured measures is used in Machtey [5] to give general characterizations of complexity sequences which apply to subrecursive and nondeterministic programming systems as well as to acceptable systems. A speedup theorem for subrecursive systems follows as a special case. (Alton [1] has independently proved a somewhat different and weaker speedup theorem for subrecursive systems.)

Limited Halting Problems

It is well known that while in every Blum complexity measure there are arbitrarily complex recursive functions, no given total recursive function can be complex in every Blum measure. Measures are easily constructed in which the given function has zero complexity. Thus some restrictions must be placed on measures in order to establish lower bounds on the complexity of specific computational problems. This section and the next will sketch a method for using the results of the previous section to accomplish this goal.

The halting problem is the basic unsolvable computational problem, and at least intuitively, "limited" halting problems should be basic intrinsically difficult computational problems. Specifically, for any complexity measure Φ and total recursive function f we define the f-limited halting problem by

$$\text{Halt}_f^{\Phi} = \{ (i,x) \mid \Phi_i(x) \leq f(|x|) \}.$$

Intuitively, Halt_f^{Φ} should have complexity (at least) about f (i.o.). In fact, by combining the usual proof of the unsolvability of the halting problem (which uses a very simple self-referencing) with standard methods from general complexity theory we obtain the following:

> Proposition: For every Blum measure Φ there is some total recursive function h such that for every total recursive function f and any program d which decides membership in Halt_f^{Φ} there are infinitely many programs i such that $h(\Phi_d(i,x),x) \geq f(|x|)$ i.o. x.

Since Halt_f^{Φ} is defined in terms of the measure Φ, we might hope for more; however,

> Proposition: For any Blum measure Φ and any total recursive function f there is a "slightly altered" measure Ψ such that Halt_f^{Ψ} has zero complexity; the same holds with f replaced by any r.e. sequence $f_0, f_1, \ldots$ of total recursive functions.

Thus for specific functions f there is no nontrivial lower bound on the complexity of the f-limited halting problem which can be established for arbitrary measures. However, for linearly bounded complexity measures, f-limited halting problems must be at least about f-hard.

> Theorem: If Φ is a linearly bounded complexity measure and f is a total recursive function, then Halt_f^{Φ} must be at least f-hard in the following sense: if d is any program which decides membership in Halt_f^{Φ}, then there is a program n (depending effectively on d) such that for all x
>
> (a) (n,x) is not in Halt_f^{Φ}, and
>
> (b) $\Phi_d(n,x) > f(|x|)/k^3 - |x|/k^2 - k"$
>
> where k is from the definition of linearly bounded measures and k" is another constant.

The proof uses the complexity theoretic Recursion Theorem above to show that if the conclusion did not hold then a self-referencing program could be constructed which says, "If I am going to be cheap to run, then I shall not halt at all." Again, the proof does not require the programming system to be acceptable or the predicate $\Phi_i(x) \leq y$ to be decidable. Thus the conclusion holds for subrecursive and nondeterministic programming systems. In fact, the proof is actually somewhat simpler and more natural in the context of "partial" decision procedures, and nondeterministic decision procedures can quite easily be viewed as partial decision procedures.

Lower Bounds on Computational Complexity

The previous Theorem supplies appropriate i.o. lower bounds on limited halting problems for linearly bounded complexity measures, which include all "reasonable" complexity measures. Our constructions of linearly bounded measures in which simulation must be very costly which were mentioned earlier show that no appropriately tight upper bounds exist for limited halting problems in all linearly bounded complexity measures. Some additional restrictions on the measures are required. It is interesting to contrast this situation with those compression and hierarchy results in which upper bounds are relatively easy to verify while lower bounds are more difficult to obtain. In addition, we point out that these methods provide simple, direct, and diagonalizationless proofs of hierarchy theorems for such systems as nondeterministic Turing machines.

A common feature in the proofs of intractability of specific computational problems originated by Meyer and his colleagues is an "efficient translation" of some programming system - usually Turing machines - into the problem in question. This translation can be formalized as a complexity restricted (Many-one) reduction of one set to another.

Definition: For any complexity measure Φ, total recursive function f, and sets S and T, we write

$$S \in M_{\Phi,f}[T]$$

if there is a total recursive function r computed by a program R such that for all programs i and inputs x,

(a) $(i,x) \in S$ if and only if $r(i,x) \in T$,

(b) $\forall i \; \exists c \; (|r(i,x)| \leq c|x| \text{ a.e. } x)$, and

(c) $\forall c \geq 1 \; (\Phi_R(i,x) \leq f(|r(i,x)|)/c \text{ a.e. } x)$.

All existing proofs of intractability include, in some form, a demonstration that $\text{Halt}_f^\Phi \in M_{\Phi,f}[T]$ for some Φ, f, and T.

We now have the machinery for our general and somewhat simplified method for proving intractability. First, it is convenient to put some slight restrictions on the functions f. We say that a function f is more than linear if for all y and c

$$f(y) \leq f(y+1), \quad c*f(y) \leq f(c*y), \quad \text{and} \quad c*y \leq f(y) \text{ a.e. } y.$$

The previous Theorem together with some additional calculations yield the following:

> Theorem: Let Φ be a linearly bounded complexity measure, f be a total recursive function which is more than linear, and T be a set such that $\text{Halt}_f^\Phi \in M_{\Phi,f}[T]$. If d is any program which decides membership in T then there is a constant K which depends effectively on d such that $f(|z|/K) < \Phi_d(z)$ for infinitely many $z \notin T$.

The generality of the method provided by this Theorem lies in the wide variety of programming systems and complexity measures which can be employed. As we shall indicate below, one is free to choose a system and measure which most naturally reduces to a specific set T in question. In some cases the most

convenient measure, while easily seen to be linearly bounded and reasonably (e.g. polynomially) related to more standard measures, might be somewhat "unnatural" and not have sufficiently small overhead for simulation to allow intractability proofs based on diagonalization using that measure.

Since our method uses no universal simulation, it is now clear that such intractability results - including all of those in Stockmeyer [9], for example - do not depend in any way on diagonalization.

This method is somewhat simpler in that it makes no use of notions of honesty or of compression-hierarchy results, which are used in other methods (and involve diagonalization). Such a presentation appears in Machtey and Young [7].

One of the applications of our method which appears in Machtey and Young [7] is the result of Fischer and Rabin [2] for Presburger arithmetic. This application provides an extremely nice example of choosing a convenient programming system and complexity measure which are not entirely "natural" but which nevertheless work. Moreover, that presentation makes it absolutely clear to what extent the clever Fischer-Rabin construction of short predicates for limited multiplication is the key to proving their result.

We conclude with two final remarks. Various strengthened statements of intractability, such as on the density of "hard" inputs, can be derived in our method with the same amount of

additional work. Also, although no such application has yet been found for natural, interesting problems, our method is at least as likely as others to yield proofs of subexponential (e.g. polynomial) lower bounds. In fact, since our method does not use diagonalization, it may hold out hope of providing such proofs when other methods cannot.

References

[1] D.A. Alton, "Natural" complexity measures and a subrecursive speed-up theorem, Proc. IFIP '77.

[2] M. Fischer and M. Rabin, Super-exponential difficulty of Presburger arithmetic, SIAM-AMS Proc. VII (1974).

[3] S.C. Kleene, Introduction to Metamathematics Van Nostram, Princeton, 1952.

[4] D. Kozen, Indexings of Subrecursive Classes, TCS (to appear).

[5] M. Machtey, Characterizations of complexity sequences in a general context, (in preparation).

[6] Machtey, Winklmann, and Young, Simple Gödel numberings, isomorphisms, and programming properties, SIAM J. Comp. 7 (1978).

[7] Machtey and Young, An Introduction to the General Theory of Algorithms, North Holland, New York, 1978.

[8] H. Rogers, Theory of Recursive Functions and Effective Computability, McGraw-Hill, 1967.

[9] L. Stockmeyer, The complexity of decision problems in automata theory and logic, M.I.T. Project MAC Tech. Report TR-133 (1974).