COMPRESSION AND THE DETERMINISTIC TIME HIERARCHY

by

Faith E. Fich and Shafi Goldwasser

Memorandum No. UCB/ERL M82/4

1 February 1982

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Compression and the Deterministic Time Hierarchy

Faith E. Fich and Shafi Goldwasser
Computer Science Division
University of California
Berkeley, California 94720

1. Introduction

Throughout this paper, the model of computation being used is the one tape deterministic Turing machine. The symbol T_i will be used to denote the i + 1st Turing machine in the standard enumeration of all one tape Turing machines where i, viewed as a string, is actually the description of T_i. When we are interested in a Turing machine which computes a specific function f we will use the symbol T^f. If T is a Turing machine then $\#T[x]$ denotes the number of steps the Turing machine performs on input x. The notation $|x|$ is used to represent the length of the number x when written in binary with no leading zeros. In particular, $|0| = 0$.

In 1968, Hartmanis [3] proved a now well known theorem (Theorem 10) concerning complexity classes defined by time bounded one tape Turing machines. We prove a theorem (Theorem 9) similar to Hartmanis's theorem in which "infinitely often" is replaced to "almost everywhere". Suppose $f_2(n)$ is fully time constructible (Definition 8) and $f_2(n) \geq \varepsilon(n) f_1(n) |f_1(n)|$ infinitely often, where $\varepsilon(n)$ is any unbounded function of n. Then Hartmanis asserts there are languages that can be computed in $O(f_2(n))$ steps, but take more than $f_1(n)$ steps for infinitely many n. We assert there are languages that can be computed in $O(f_2(n))$ steps, but take more than $f_1(n)$ steps to compute for almost all n. Moreover, our f_1 and f_2 satisfy conditions surprisingly similar to Hartmanis's f_1 and f_2.

We obtain our result by developing a new version of the compression theorem for one tape Turing machines, and then applying it to fully time constructible functions.

In 1960, Rabin [9] proved the existence of arbitrarily complex 0-1 valued recursive functions. Later, Blum [2] proved the compression theorem which sets upper and lower bounds on the number of steps to compute certain of these functions. We give a new characterization of the functions that can be used as lower bounds in the compression theorem, in applications to the time complexity of one tape Turing machines. Our class of lower bound functions include not only honest functions (Definition 2) but also functions that are honest up to a polynomial (Definition 4). In addition we improve the upper bound given by the compression theorem for one tape Turing machines.

Similar results were obtained independently by Seiferas, Fischer and Meyer in [12]. Other related results pertaining to space complexity can be found in [8] and [10].

2. Expansion of the Compression Theorem

Let $\varepsilon(x)$ be any unbounded nondecreasing function of x such that $\varepsilon(x) < x$ for all $x \geq 0$. Furthermore suppose there is a Turing machine which computes $\varepsilon(x)$ that takes at most $O(x)$ steps on input x. Intuitively, think of $\varepsilon(x)$ as a slowly growing, easy to compute function of x, e.g. $\varepsilon(x) = |x| - 1$. Let $P(v,w) = w|w|\varepsilon(v) + w\varepsilon(v)|\varepsilon(w)| + \varepsilon(v)v$.

Theorem 1 . For every recursive function f there exists a 0-1 valued function g such that:

1) if T_i computes g then $\#T_i[x] > f(x)$ almost everywhere

and 2) there exists a Turing machine T^g which computes g such that

$$\#T^g[x] = O(\,P(|x|, f(x)) + \#T^f[x]\,).$$

Proof: The idea is to construct g in stages such that at stage x the value of $g(x)$ is determined. This value of $g(x)$ depends only on those previous stages j for which $0 \le j \le \varepsilon(|x|)$, $f(j) < \varepsilon(|x|)$, and $\#T^f[j] \le \varepsilon(|x|)$. During stage x, only the first $\varepsilon(|x|) + 1$ machines, $T_0, \ldots, T_{\varepsilon(|x|)}$, are examined.

Given a recursive function f consider the following procedure which successively defines the value of $g(x)$ for $x = 0, 1, 2, \cdots$

```
for $x \leftarrow 0, 1, 2, \cdots$ do
        begin
        compute $f(x)$
        for $i \leftarrow 0$ to $\varepsilon(|x|)$ do
                run $T_i[x]$ for $f(x)$ steps
        select the least $i$ such that
                1) $\#T_i[x] \le f(x)$
                2) $T_i$ was not cancelled during the computation of $g(j)$
                   for any $j$ such that $0 \le j \le \varepsilon(|x|)$,
                   $f(j) < \varepsilon(|x|)$, and $\#T^f[j] \le \varepsilon(|x|)$.
        if there is such an $i$
                then let $g(x) \leftarrow \begin{cases} 1 \text{ if } T_i[x] = 0 \\ 0 \text{ if } T_i[x] \neq 0 \end{cases}$     //thereby cancelling machine $T_i$ //
                else let $g(x) \leftarrow 0$
        end
```

Claim 1 . No Turing machine T_i gets cancelled infinitely often.

Proof: Suppose T_i is cancelled for the first time at stage z. Let y be the smallest value such that $\varepsilon(|y|) \ge \max\{z, f(z), \#T^f[z]\}$. Now, for all $x \ge y$, $\varepsilon(|x|) \ge z$, $\varepsilon(|x|) \ge f(z)$, and $\varepsilon(|x|) \ge \#T^f[z]$. Hence during the computation of $g(x)$ the programs knows that T_i has been cancelled, and thus does not cancel T_i again.

Claim 2 . If T_i computes g then $\#T_i[x] > f(x)$ almost everywhere.

Proof: Suppose that T_i computes g but $\#T_i[x] \le f(x)$ infinitely often. Since T_i never gets cancelled, for all x such that $i \le \varepsilon(|x|)$ and $\#T_i[x] \le f(x)$, there must exist a Turing machine T_k, with $k < i$, that gets cancelled. There are an infinite number of such x's. Therefore there exists a Turing machine T_k, with $k < i$, that gets cancelled infinitely often. This contradicts Claim 1.

The following program computes $g(x)$ quickly. Here *CANCELLED* is a list containing quadruples $(k, i, f(i), \#T^f[i])$ indicating that during *STAGE*(i) (i.e. *STAGE*$(i, f(i), \#T^f[i])$) Turing machine T_k is cancelled. For each nonnegative integer i, there will be at most one quadruple whose second component is i. The quadruples are kept sorted in increasing order of their second components.

```
function PSI($x$)
        begin
        CANCELLED ← $\phi$
        for $i$ ← 0 to $\varepsilon(|x|)$ do       //perform those stages $i$ for which $i$, $f(i)$,
                                                  and $\#T^f[i]$ are sufficiently small//
                begin
                run $T^f[i]$ for $\varepsilon(|x|)$ steps
                if $\#T^f[i] \le \varepsilon(|x|)$ and $f(i) \le \varepsilon(|x|)$
                        then perform STAGE($i, f(i), \#T^f[i]$)
                end
        run $T^f[x]$ until it halts                 // to compute $f(x)$ //
        PSI($x$) ← STAGE($x, f(x)$,0)                   // $\#T^f[x]$ is not needed in STAGE($x$)
                                                and computing it would take too much time //
        end

function STAGE($i,y,z$)        // when this function is invoked $y = f(i)$
                                and, except when $i = x$, $z = \#T^f[i]$.//
        begin
        for $k$ ← 0 to $\varepsilon(|i|)$ do
                begin
                if there is no quadruple $(k, j, f(j), \#T^f[j])$ in CANCELLED such that
                        [ $0 \le j \le \varepsilon(|i|)$, $f(j) \le \varepsilon(|i|)$, and $\#T^f[j] \le \varepsilon(|i|)$ ]
                then    begin
                        run $T_k[i]$ for $y$ steps
                        if $T_k[i]$ halts in this time
                                then    begin          // cancel Turing machine $T_k$ //
                                        if $i \ne x$ then append $(i,y,z,k)$ to CANCELLED
                                        if $T_k[i] = 0$
                                                then return(1)
                        else return(0)
                    end
            end
        end
    return(0)               // in this case no machine that
                            affects $g(x)$ will be cancelled
                            during the $i$th stage //
    end
```

Claim 3 . There is a one tape Turing machine T^g which implements the program G such that $\#T^g[x] = O(\ P(|x|, f(x)) + \#T^f[x]\ )$.

Proof: Consider the time taken to compute $g(x)$ by this algorithm.

STAGE(i) is computed for $i = x$ and for all $i \le \varepsilon(|x|)$ such that $f(i) \le \varepsilon(|x|)$ and $\#T^f[i] \le \varepsilon(|x|)$.

For all quadruples $(k, i, f(i), \#T^f[i])$ in *CANCELLED*, $i \le \varepsilon(|x|)$, $f(i) \le \varepsilon(|x|)$, $k \le \varepsilon(|i|) \le \varepsilon(|\varepsilon(|x|)|)$, and $\#T^f[i] \le \varepsilon(|x|)$. Thus the length of each such quadruple is $O(|\varepsilon(|x|)|)$. At most one quadruple is added to *CANCELLED* at each stage and at most $\varepsilon(|x|) + 2$ stages are performed. Hence the total length of *CANCELLED* is $O(\varepsilon(|x|)|\varepsilon(|x|)|)$.

Determining if there is a quadruple in *CANCELLED* whose first component is k and whose other components are at most $\varepsilon(|i|)$ can be done in O(length of *CANCELLED*) steps. Similarly, to append a new quadruple to *CANCELLED* takes O(length of *CANCELLED*) time.

The simulation of the one tape Turing machine $T_k[i]$ for $f(i)$ steps can be accomplished on one tape in time $f(i)(|k| + |f(i)|)$ by carrying both a timer and the description of T_k along with the tape head. The description of T_k, the initial value, $f(i)$, of the timer, and the input i must be copied to set up the initial configuration for the simulation. This takes time $O(|k| + |f(i)| + |i|)$.

By assumption the cost of computing $\varepsilon(|i|)$ is $O(|i|)$.

The cost of $STAGE(i)$ consists of the cost of computing $\varepsilon(|i|)$, the cost of controlling the for loop, the cost of appending a new quadruple to *CANCELLED* if a machine is cancelled during $STAGE(i)$, and for each iteration of the for loop, the cost of searching *CANCELLED* and the time taken to simulate machine $T_k[i]$ for $f(i)$ steps. Thus, in total, the number of steps performed at $STAGE(i)$ is

$$O(\ |i| + \varepsilon(|i|)|\varepsilon(|i|)| + \varepsilon(|x|)|\varepsilon(|x|)|$$
$$+ \sum_{k=0}^{\varepsilon(|i|)} [\,\varepsilon(|x|)|\varepsilon(|x|)| + f(i)(|k| + |f(i)|) + |k| + |f(i)| + |i|\ ]\,)$$
$$= O(\ \varepsilon(|i|)\ [\ \varepsilon(|x|)|\varepsilon(|x|)| + f(i)|\varepsilon(|i|)| + f(i)|f(i)| + |i|\ ]\,).$$

To test whether $f(i) \le \varepsilon(|x|)$ and $\#T^f[i] \le \varepsilon(|x|)$ it suffices to run $T^f[i]$ for $\varepsilon(|x|)$ steps. By carrying a timer along with the tape head, our one tape Turing machine can perform the computation of $T^f[i]$ for $\varepsilon(|x|)$ steps, in $\varepsilon(|x|)|\varepsilon(|x|)|$ steps. There is also an associated overhead of $|i| + |\varepsilon(|x|)|$ to initialize the input and the timer.

The cost of computing $G(x)$ consists of the cost of computing $f(x)$, the cost of computing $\varepsilon(|x|)$, the cost of performing $STAGE(x)$, and for each i, $0 \le i \le \varepsilon(|x|)$, the cost of running $T^f[i]$ for $\varepsilon(|x|)$ steps and possibly the cost of performing $STAGE(i)$. Computing $f(x)$ takes $\#T^f[x]$ steps. By assumption, the cost of computing $\varepsilon(|x|)$ is $O(|x|)$ steps.

Thus the total time taken by the Turing machine which implements $PSI(x)$ is
$\#T^f[x] + O(\,|x| + \varepsilon(|x|)\ [\ \varepsilon(|x|)|\varepsilon(|x|)| + f(x)(|\varepsilon(|x|)| + |f(x)|) + |x|\ ] +$

$$\sum_{i=0}^{\varepsilon(|x|)} [\, \varepsilon(|x|)|\varepsilon(|x|)| + |i| + |\varepsilon(|x|)| + \varepsilon(|i|)\,(\, \varepsilon(|x|)|\varepsilon(|x|)| + f(i)|\varepsilon(|i|)|$$

$$+ f(i)|f(i)| + |i|\,)\,])$$

$$= \#T^f[x] + O(\, f(x)\varepsilon(|x|)|f(x)| + f(x)\varepsilon(|x|)|\varepsilon(|x|)| + \varepsilon(|x|)|x|\,)$$

$$= O(\, \#T^f[x] + P(|x|, f(x))\,).$$

Although our description might seem to indicate the need for multiple tapes, careful inspection will reveal that only multiple tracks on one tape are required. []

Now we look at what happens when restrictions are placed on the function f.

Definition 2 . Let $Q(v,w)$ be a total function. A function $f : \mathbf{N} \to \mathbf{N}$ is said to be *Q–honest* if it is recursive and there exists a Turing machine T^f which computes f such that $\#T^f[x] = O(\, Q(|x|, f(x))\,)$.

This definition is closely related to a definition given in [8].

The following theorem is an immediate consequence of Theorem 1.

Theorem 3 . For every Q-honest function f there exists a 0-1 valued function g such that:

1) if T_i computes g then $\#T_i[x] > f(x)$ almost everywhere

and 2) there exists a Turing machine T^g which computes g such that

$$\#T^g[x] = O(\, P(|x|, f(x)) + Q(|x|, f(x))\,).$$

A similar result is also obtained for a larger class of functions.

Definition 4 . Let $Q(v,w)$ and $R(v,w)$ be total functions. A total function $f : \mathbf{N} \to \mathbf{N}$ is said to be *Q–honest mod R* if there exists a Q-honest function h such that $R(|x|, f(x)) \geq h(x) \geq f(x)$ almost everywhere.

Clearly, if $R(v,w) \geq w$ almost everywhere then every Q-honest function is Q-honest mod R. Also note that f is not Q-honest mod R if and only if for all Q-honest functions h either $h(x) < f(x)$ infinitely often or $h(x) > R(|x|,f(x))$ infinitely often.

Let $R(v,w)$ be a polynomial which is a nondecreasing function of its variables.

Theorem 5 . For all functions f which are Q-honest mod R there exists a 0-1 valued function g such that:

1) if T_i computes g then $\#T_i[x] > f(x)$ almost everywhere

and 2) there exists a Turing machine T^g which computes g such that

$$\#T^g[x] = O(\ P(|x|,\ R(|x|,f(x))) + Q(|x|,f(x))\ ).$$

Proof: Since f is Q-honest mod R, there exists a Q-honest function h such that $R(|x|,f(x)) \geq h(x) \geq f(x)$ almost everywhere. By Theorem 3 there exists a 0-1 valued function g such that:

1) if T_i computes g then $\#T_i[x] > h(x) \geq f(x)$ almost everywhere

and 2) there exists a Turing machine T that computes g such that $\#T^g[x] =$

$$O(\ P(|x|,h(x)) + Q(|x|,f(x))\ ) = O(\ P(|x|,\ R(|x|,f(x))) + Q(|x|,f(x))\ ). \square$$

3. A Converse for the Compression Theorem

Lemma 6 . Let T be a Turing machine which halts on all inputs. Then the function $h(x) = \#T[x]$ is a Q-honest function for all total functions $Q(v,w)$ such that $Q(v,w) \geq w|w|$ almost everywhere.

Proof: $h(x)$ can be computed by running $T[x]$ until it halts, keeping count of the number of steps which have been performed on a separate track. This counter is carried along with the tape head so that each step of $T[x]$ can be simulated in at most

$O(|h(x)|)$ steps. Thus $h(x)$ can be computed in $O(h(x)|h(x)|) = O(Q(|x|,h(x)))$ steps. []

Theorem 7 . If f is not Q-honest mod Q for all polynomial bounded functions $Q(v,w)$ then for all polynomials $P(v,w)$ and all Turing machines T which compute 0-1 valued functions

either 1) $\#T[x] < f(x)$ infinitely often

or 2) $\#T[x] > P(|x|,f(x))$ infinitely often.

Proof: Suppose f is not Q-honest mod Q for all polynomials $Q(v,w)$. Let $P(v,w)$ be any polynomial, let T be any Turing machine which computes a 0-1 valued function, and let $h(x) = \#T[x]$. Let $Q(v,w) = P(v,w) + w|w| \geq P(v,w)$ for all v,w. Since f is not Q-honest mod Q and h is Q-honest it follows from Definition 4 that either $h(x) < f(x)$ infinitely often or $h(x) > Q(|x|,f(x)) \geq P(|x|,f(x))$ infinitely often. []

4. Examples

The following example illustrates a recursive function f which is not Q-honest mod R for any polynomials $Q(v,w)$ and $R(v,w)$.

Consider any language $L \not\in \mathbf{P}$ (i.e. there is no deterministic Turing mcahine which recognizes L in polynomial time). Define

$$f(x) \leftarrow \begin{cases} x \text{ if } x \in L \\ |x| \text{ if } x \not\in L \end{cases}$$

Suppose f is Q-honest mod R for some polynomials $Q(v,w)$ and $R(v,w)$. Then by Theorem 5 there exist a polynomial $P_{Q,R}(v,w)$ and a 0-1 valued function g such that:

1) if T_i computes g then $\#T_i[x] > f(x)$ almost everywhere

and 2) there exists a Turing machine T^g which computes g such that

$$\#T^g[x] = O(P_{Q,R}(|x|, f(x))).$$

Let y be such that, for all $x > y$, $\#T^g[x] \le P_{Q,R}(|x|, f(x))$, $\#T^g[x] > f(x)$, and $s(x) > P_{Q,R}(|x|, |x|)$. Such a y must exist.

Now $\#T^g[x] < P_{Q,R}(|x|, |x|)$ if and only if $x \notin L$ for all $x > y$. If $x \in L$, then $\#T^g[x] > f(x) = x > P_{Q,R}(|x|, |x|)$. Conversely, suppose $x \notin L$. Then $f(x) = |x|$ and $\#T^g[x] < P_{Q,R}(|x|, f(x)) = P_{Q,R}(|x|, |x|)$.

Consider the following Turing machine T^L which recognizes L. For $x \le y$ use table look-up to determine whether $x \in L$. For $x > y$ run $T^g[x]$ for $P_{Q,R}(|x|, |x|)$ steps. If $T[x]$ halts in this time then $x \in L$ and T^L accepts; otherwise $x \notin L$ and T^L rejects. Clearly, $\#T^L[x] \le P(|x|, |x|)$ almost everywhere. Thus $L \in \mathbf{P}$ which contradicts our choice of L.

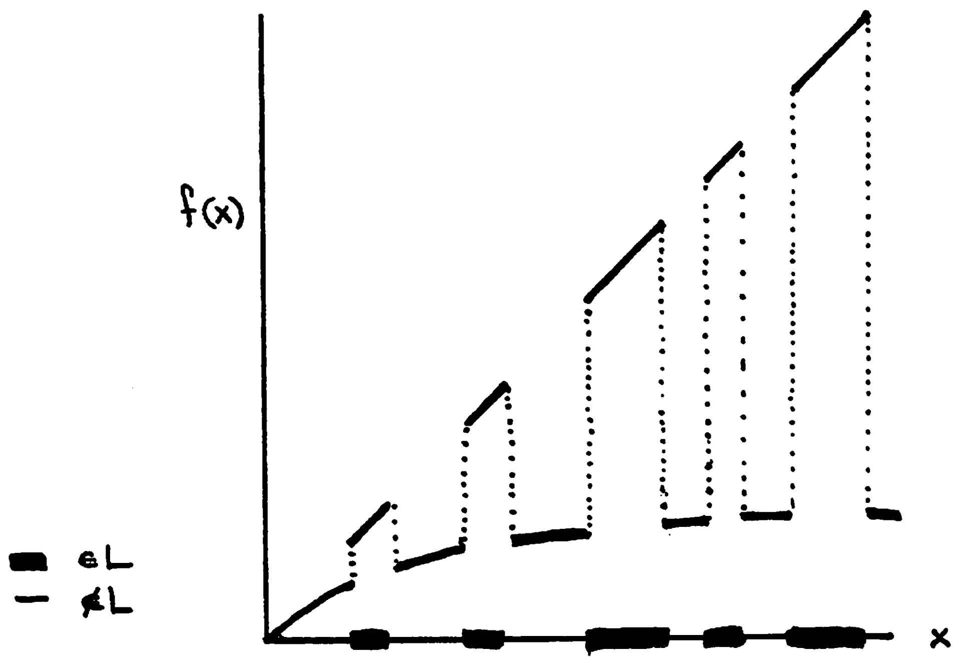

Figure 1. A function which is not Q-honest mod R.

Hence f is not Q-honest mod R for any polynomials $Q(v,w)$ and $R(v,w)$. The same result would be true if

$$f(x) \leftarrow \begin{cases} s(|x|) \text{ if } x \in L \\ r(|x|) \text{ if } x \notin L \end{cases}$$

Where $r(u)$ is a polynomial function of u and $s(u)$ is a recursive function which is not bounded by any polynomial function of u.

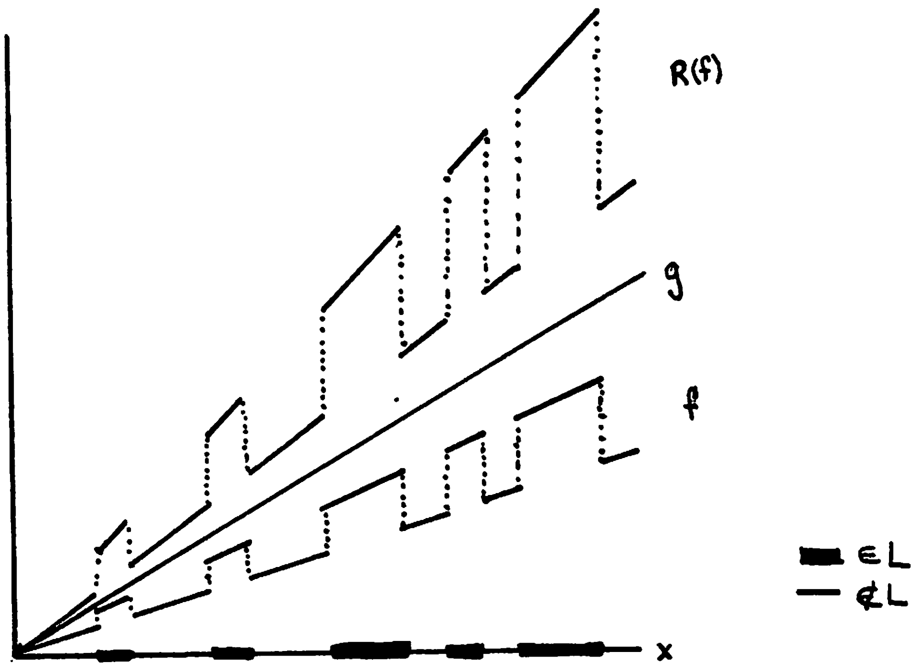

Figure 2. A function which is not Q-honest but is Q-honest mod R.

Let $c > 1$ be a constant. For any polynomial $Q(v,w) \geq |c|v$ and any polynomial $R(v,w) \geq w^2$ there is a recursive function which is Q-honest mod R but not Q-honest.

Consider any language $L \notin$ P. Define

$$f(x) \leftarrow \begin{cases} cx \text{ if } x \in L \\ x \text{ if } x \notin L \end{cases}$$

Let $h(x) = cx$. Then $R(|x|, f(x)) \geq f^2(x) \geq h(x) \geq f(x)$ almost everywhere and the number of steps to compute $h(x) = O(|x|) = O(Q(|x|, h(x)))$. Hence f is Q-honest mod R.

Since $Q(|x|, f(|x|))$ is polynomial function of $|x|$ and $L \not\in \mathbf{P}$ it follows that $f(x)$ cannot be Q-honest.

5. Implications for the Deterministic Time Hierarchy

The following definition is from [6] and [3].

Definition 8 . A function $f(n): \mathbf{N} \to \mathbf{N}$ is said to be ***fully time constructible*** if there exists a Turing machine T such that $\#T[x] = f(|x|)$ for all $x \geq 0$.

Most well known functions are fully time constructible. One consequence of a function f being fully time constructible is that $f(n) \geq n$ for all $n \geq 0$. Another is the following theorem, mentioned in the introduction.

Theorem 9 . Let f_1 be a fully time constructible function, and let $\varepsilon(x)$ be an unbounded nondecreasing function of x such that $\varepsilon(x) < x$ for $x \geq 0$ and which can be computed in $O(x)$ steps. For all total functions f_2 such that $f_2(|x|) \geq \varepsilon(|x|) f_1(|x|) |f_1(|x|)|$ almost everywhere, there exists a language L with characteristic function g such that

1) for all Turing machines T_i that compute g, $\#T_i[x] > f_1(|x|)$ almost everywhere and 2) there exists a Turing Machine T^g that computes g such that

$$\#T^g[x] = O(f_2(|x|)).$$

Proof: Let $h(x) = f_1(|x|)$. Then, by Lemma 6, h is Q-honest for $Q(v,w) = w|w|$.

From Theorem 3 it follows that there exists a 0-1 valued function g such that

1) for all Turing machines T_i that compute g, $\#T_i[x] > h(x) = f_1(|x|)$ almost everywhere

and 2) there exists a Turing Machine T^g that computes g such that $\#T^g[x] =$

$$O(\varepsilon(|x|)h(x)|h(x)| + \varepsilon(|x|)h(x)|\varepsilon(|x|)| + \varepsilon(|x|)|x| + h(x)|h(x)|).$$

But $h(x) = f_1(|x|) \geq |x| > \varepsilon(|x|)$. Therefore $\#T^g[x] = O(\varepsilon(|x|)f_1(|x|)|f_1(|x|)|) = O(f_2(|x|))$. □

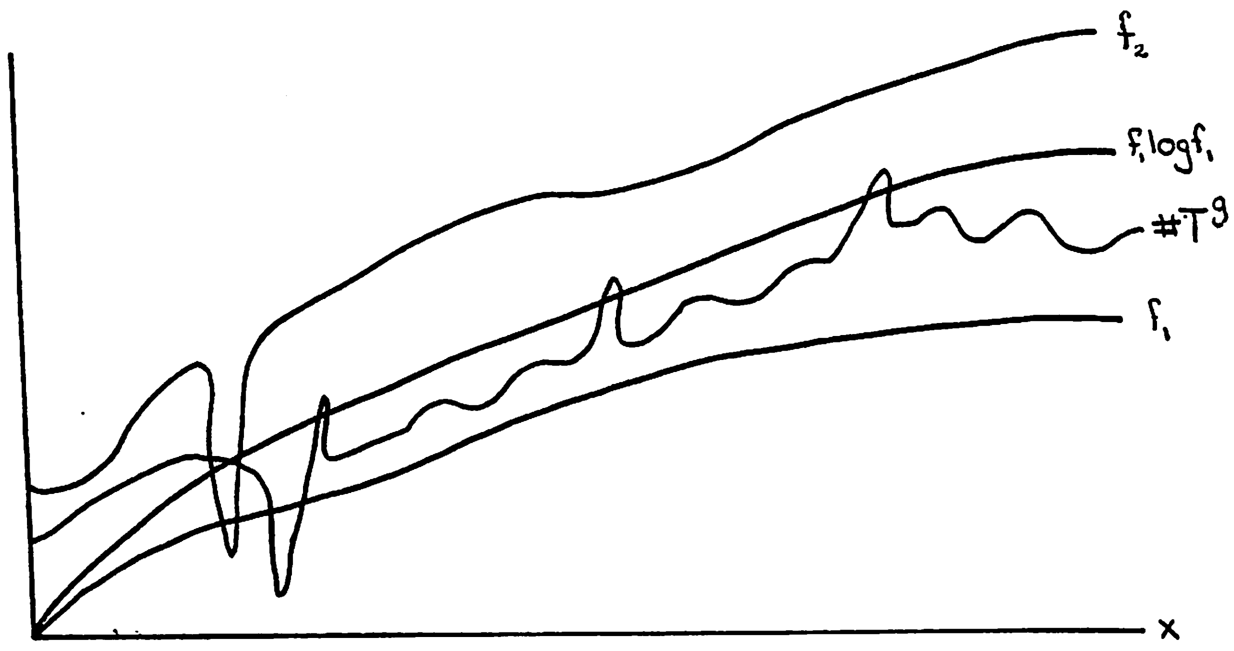

Figure 3. An illustration of the relationship between f_1, f_2 and $\#T^g$ in Theorem 9.

Now compare this with Hartmanis's theorem ([3], [6]).

Theorem 10 . Let $f_2(n)$ be a fully time constructible function, and let $\varepsilon(n)$ be any slow growing, unbounded function of n. Then for every total function $f_1(n)$ such that $f_2(n) \geq \varepsilon(n)f_1(n)|f_1(n)|$ infinitely often there exists a language L with characteristic function g such that

1) for all Turing machines T_i that compute g, $\#T_i[x] > f_1(|x|)$ infinitely often

i.e. $L \not\in DTIME(f_1(|x|))$

and 2) there exists a Turing machine T^g that computes g such that

$\#T^g[x] = O(f_2(|x|))$ i.e $L \in DTIME(f_2(|x|))$.

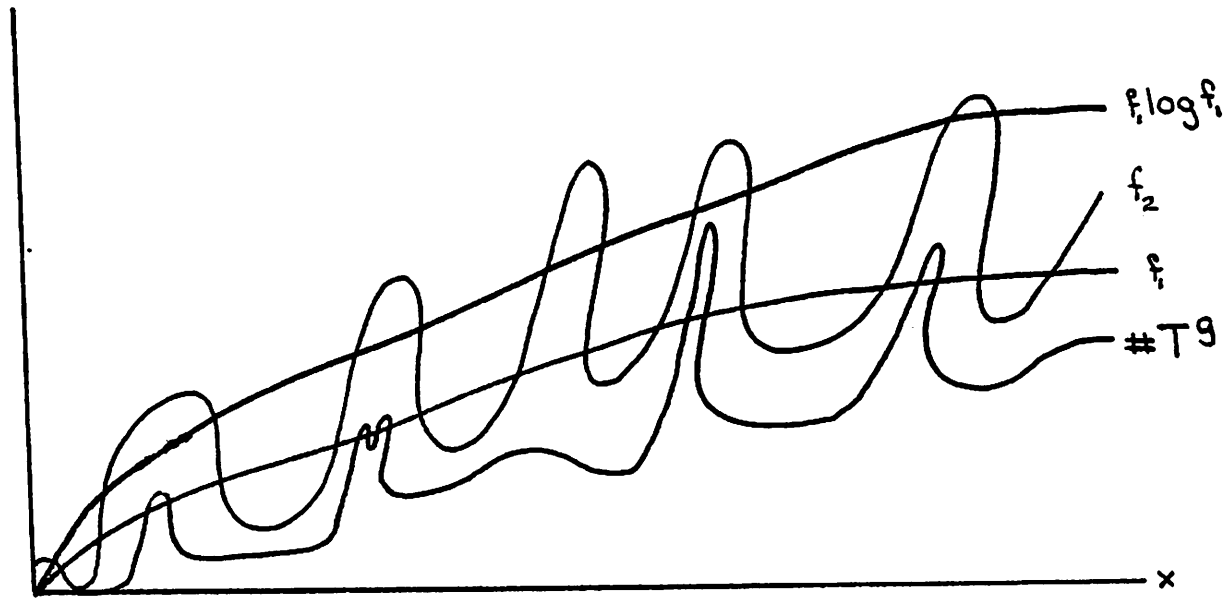

Figure 4. An illustration of the relationship between f_1, f_2, and $\#T^g$ in Theorem 10.

The major difference between this theorem and previous one is that this theorem only shows that all Turing machines which compute g must run slowly infinitely often whereas Theorem 9 shows that all Turing machines which compute g must run slowly almost everywhere. Notice that this corresponds to the difference in the relationship between f_1 and f_2 in the two theorems, i.e whether $f_2(n) \geq \varepsilon(n) f_1(n) | f_1(n) |$ infinitely often or almost everywhere. This is illutsrated in figures 3 and 4.

Our theorem requires f_1 to be fully time constructible and f_2 to merely be total, as compared to Hartmanis's theorem, which requires f_2 to be fully time constructible and f_1 to be total. This is a consequence of the way the two theorems are total, as compared to Hartmanis's theorem, which requires f_2 to be fully time constructible and f_1 to be total. This is a consequence of the way the two theorems are proved. Hartmanis does a diagonalization over all Turing machines which run "slower" than f_2. On the other hand, we perform a diagonalization over all Turing machines which run "faster" than f_1.

We do have to pay a small price for the improved result. The function ε in the statement of our theorem is restricted to be something which is "easy to compute" rather than being an arbitrary unbounded function.

Acknowledgements

We would like to thank Manuel Blum who suggested this problem to us and gave us vast amounts of encouragement and insight. This research was supported by a Natural Science and Engineering Research Council of Canada Postgraduate Scholarship, National Science Foundation Grant MCS79-15763, and National Science Foundation Grant MCS79-03767.

Bibiliography

[1] Borodin, A., *Computational Complexity and the Existence of Complexity Gaps*, JACM 19:1, 1972, 158-174.

[2] Blum, Manuel, *A Machine-Independent Theory of the Complexity of Recursive Functions*, JACM 14:2, 1967, 322-336.

[3] Hartmanis, J., *Computational Complexity of One-Tape Turing Machine Computations*, Functions, JACM 15:2, 1968, 325-339.

[4] Hartmanis, J., and Hopcroft, J. E., *An Overview of the Theory of Computational Complexity*, JACM 18:3, 1971, 444-475.

[5] Hartmanis, J., and Stearns, R. E., *On the Computational Complexity of Algorithms*, Trans. Amer. Math. Soc.117, 1965, 285-306.

[6] Hopcroft, J. E., and Ullman, J. D., *Introduction to Automata Theory, Languages, and*