RELATIONS BETWEEN DIAGONALIZATION, PROOF SYSTEMS, AND COMPLEXITY GAPS

by

Juris Hartmanis

TR 77-312

Computer Science Department
Cornell University
Ithaca, N.Y. 14853

Abstract

In this paper we study diagonal processes over time-bounded computations of one-tape Turing machines by diagonalizing only over those machines for which there exist formal proofs that they operate in the given time bound. This replaces the traditional "clock" in resource bounded diagonalization by formal proofs about running times and establishes close relations between properties of proof systems and existence of sharp time bounds for one-tape Turing machine complexity classes. These diagonalization methods also show that the Gap Theorem for resource bounded computations can hold only for those complexity classes which differ from the corresponding provable complexity classes. Furthermore, we show that there exist recursive time bounds $T(n)$ such that the class of languages for which we can formally prove the existance of Turing machines which accept them in time $T(n)$ differs from the class of languages accepted by Turing machines for which we can prove formally that they run in time $T(n)$.

1. INTRODUCTION

One of the central problems in computational complexity theory is to determine for a given computer model and computer resource by how much a given resource bound T(n) (satisfying some honesty conditions) has to be increased to be able to compute something new which cannot be computed in the old resource bound T(n) [5].

In this paper we show that these problems are very closely related to problems about what can and cannot be proven formally about resource bounded computations.

The standard way to obtain separation results for complexity classes is by efficiently diagonalizing over all the computations which can be performed in the given resource bound. The efficiency of the diagonal process over the resource bounded computations, or the additional amount of resources required to carry out the diagonalization over all the computations computable within the given resource bound, determines the sharpness of the results. The standard way to carry out such diagonal processes is to bound the given resource as a function of the length of the input and by simulation determine on successive inputs what different Turing machines do and do the opposite, provided their simulation did not try to exceed the given resource bound [1,4,5]. Such diagonal processes, in essence, require that we perform two separate computations: a simulation process and a process which shuts the computation off if it tries to use too much of the bounded resource.

This method works very well for reusable resource measures where we can first compute the resource bound and then perform the simulation within the bounded resources. For example, for the tape bounded Turing machine computations the following result holds [5].

1. Let $t(n)$, $t(n) \geq \log n$, be computable on $t(n)$ tape. Then there exists a language, A, acceptable on $t(n)$ tape but not acceptable of $t_1(n)$ tape, provided

$$\lim_{n \to \infty} \frac{t_1(n)}{t(n)} = 0.$$

For time bounded Turing machine computations the problems become more difficult and the results depend whether we consider the class of all many-tape Turing machines or the class of one-tape Turing machines (or more generally the class of Turing machines with a fixed number of tapes). In the first case, when we consider multi-tape Turing machines, it is easy to run the two computational processes of simulation and the shut-off clock independently on separate sets of tapes and not lose any time. On the other hand, since in this case we must simulate Turing machines with arbitrarily many tapes on a Turing machine with a fixed number of tapes (the diagonalizer), we lose time in this process. The use of the best known simulation result of many-tape machines on a two-tape machine yields the following hierarchy result for multi-tape Turing machines [7]; any improvement in the time loss during the simulation of many-tape Turing

machines on a machine with fixed number of tapes would lead to a corresponding improvement in this result.

2. Let $T(n)$, $T(n) \geq n$, be computable by a k-tape Turing machine in time $T(n)$. Then there exists a language which is accepted in time $T(n)$, but not in time $T_1(n)$, provided

$$\lim_{n \to \infty} \frac{T_1(n) \cdot \log T_1(n)}{T(n)} = 0.$$

For one-tape Turing machines (or Turing machines with a fixed number of tapes) the situation changes. In this case the simulation of one-tape machines on a fixed one-tape machine can be carried out without a substantial time loss, but the running of the shut-off "clock" must be performed (in parallel) with the simulation operations on the same tape (or the fixed number of tapes). The combining of two independent computations on one tape leads to a time loss and the best result about one-tape Turing machines known up to the present is stated below [4].

3. Let $T(n) \geq n \cdot \log n$ be computable on log $T(n)$ tape in $T(n)$ time by a one-tape Turing machine. Then there exists a set acceptable by a one-tape Turing machine in time $T(n) \cdot \log T(n)$ which is not acceptable by any one-tape Turing machine in time $T_1(n)$, provided

$$\lim_{n \to \infty} \frac{T_1(n)}{T(n)} = 0.$$

Next we consider Turing machines with k-tapes for a fixed k, $k \geq 2$. In this case one can use two tapes to cleverly move along parts of the "clock" and a recent result shows that one can get sharper results for these machines than for one-tape machines [8].

4. Let T(n) be time constructible on a k-tape Turing machine, $k \geq 2$. Then

$$\lim_{n \to \infty} \frac{T_1(n) \log^*[T_1(n)]}{T(n)} = 0$$

implies that there exists a set acceptable in time T(n) on a k-tape machine but not in time $T_1(n)$. Where

$$\log^* n = \min\{k \mid n \leq 2^{2^{\cdot^{\cdot^{2}}}} \}k \text{ times}\}.$$

It is seen from the last two results that the need to carry along a "clock" does not permit us (so far) to prove for time bounded computations of k-tape Turing machines as sharp results as we have for the tape bounded computations and which we conjecture also hold for time bounded computations.

In this paper we study a new class of diagonalization processes over resource bounded classes in which we do not use (explicitly) a "clock". In particular, we study the class of diagonal processes over time-bounded computations of one-tape (or k-tape) Turing machines in which we diagonalize over the computations of a Turing machine M_i only if there is a formal proof that M_i runs in the given time bound. Thus in these diagonal processes we place the "clock" which shuts off the simulation

before it takes too much time by a formal proof that the simulation will not take too much time. As we will show, this approach will establish close links between properties of proof systems and the existence of sharp time bounds for one-tape Turing machine computations. Thus these results raise some interesting questions about what can and cannot be proven formally about running times of computations and emphasizes the importance of these problems to computational complexity theory.

It should be observed that it is known that formal mathematical systems are not powerful enough for the analysis of algorithms, since we can exhibit algorithms which run in a specified time, say $T(n) = n^2$, but such that there is not proof in the formal system that they run in less time than 2^n [6]. The questions raised in this paper about provable properties of running times of computations are different and they have the following form:

> Is there for every set acceptable on a one-tape Turing machine in time n^2 a one-tape Turing machine M_{i_o} which accepts this set in time n^2 and for which it can be formally proven that M_{i_o} runs in time n^2?

Finally, we show that the diagonalization methods using formal proofs show that the well known Gap Theorem for resource bounded computations [1,5] does not hold for complexity classes consisting of languages accepted by Turing machines for which we can prove formally that they run in the given time bound. This result shows that the originally surprising gap phenomenon for computational complexity classes can appear only for the non-constructively defined computational complexity classes

$$\text{TIME}[T(n)] = \{A \mid A \text{ is accepted by a one-tape Tm in time } \leq T(n)\}.$$

If we formalize our reasoning and insist that the complexity classes consist only of the sets accepted by Turing machines for which we can prove formally that they run in the given time bound, then the gap phenomenon disappears.

We now summarize the basic concepts and notation used in this paper. It should be observed that though we formulate and prove our results in the first half of this paper only for one-tape Turing machines all results carry over directly for k-tape Turing machines for a fixed k. Let M_1, M_2, M_3, ... be a standard enumeration of one-tape Turing machines (Tm's). The <u>running time</u> T_i on M_i is given by

$$T_i(n) = \max\{\text{number of operations performed by } M_i \text{ on input } w \mid w \in \Sigma^n\}.$$

The set of tapes accepted by M_i is denoted by $L(M_i)$.

An axiomatizable theory is a triple $F = (\Sigma, W, T)$ where

1. Σ is a finite non-empty alphabet,
2. W, $W \subseteq \Sigma^*$, is a recursive set, referred to as the set of well-formed formulas,
3. T, $T \subseteq W$, is a recursively enumerable set, referred to as theorems provable in F.

If w is provable in F we will write

$$\vdash_F w \text{ or } \vdash w$$

when F is fixed.

If the set T is recursive the system F is said to be decidable.

We can think of W as the syntactically correctly formed formulas and T as the subset of these for which there exist proofs in the formal system F. (In practice we would prescribe a set of axioms and proof rules so that it can be recursively decided whether a given string following a well formed formula is a proof of this formula).

We furthermore assume that the Turing machines form a model (or a submodel) for the theory F. Thus we assume that we can express and prove elementary facts about Turing machines in F and that only true statements about Turing machines can be proven in F.

The complexity classes and provable complexity classes are defined for partial recursive functions T(n) as follows:

$$\text{TIME}[T(n)] = \{L(M_i) \mid T_i(n) \leq T(n)\}$$

and

$$\text{F-TIME}[T(n)] = \{L(M_i) \mid \text{"}T_i(n) \leq T(n)\text{" is provable in F}\} = \{L(M_i) \mid \vdash_F T_i(n) \leq T(n)\}.$$

Thus the class TIME[T(n)] is the class of all sets acceptable by one-tape Tm's whose running time is bounded by T(n), without specifying how to determine that this is so. On the other hand, for a fixed formal system F the class F-TIME[T(n)] is the complexity class consisting of the languages accepted by Tm's for which there is a proof in F that they run in the time bound T(n). We shall refer to these sets as provable complexity classes.

For related work on provable properties of computational complexity problems see [3,9] and for early work on provable recursive functions see [2].

2. AXIOMATIZABLE THEORIES AND DIAGONALIZATION

In this section we explore the use of formal proofs that machines run in a given time bound in diagonalization processes. As pointed out before, the standard way of diagonalizing is to simulate all Turing machines and shutting the simulation off by an independent "clock" mechanism if the simulation tries to take too much time. Instead, we assume that we have a fixed formalized mathematical system F, as described before, and only after (searching in an efficient way for proofs when) we find a proof

in F for a Tm M_i that $T_i(n) \leq T(n)$ do we diagonalize over M_i by simulating M_i and doing the opposite. Thus we are replacing the shut-off mechanism of the previously used diagonal processes by a formal proof that the machine under consideration halts in a given time. As we will see in the following this diagonalization method depends now on what can be proven formally about running times of Turing machines and will explore these relations.

THEOREM 1: Let $T(n)$, $T(n) \geq n \cdot \log n$, be a recursive function for which

$$F\text{-}TIME[T(n)] = TIME[T(n)].$$

Then for any non-decreasing, unbounded recursive function $g(n)$, $g(n) \geq 1$, we have that

$$TIME[T(n)] \neq TIME[T(n) \cdot g(n)].$$

Proof: Since $g(n) \geq 1$ we know that

$$TIME[T(n)] \subseteq TIME[T(n) \cdot g(n)].$$

To show that

$$TIME[T(n)] \neq TIME[T(n) \cdot g(n)]$$

we construct a Tm M_D such that

$$T(M_D) \in TIME[T(n) \cdot g(n)] - TIME[T(n)].$$

The machine M_D operates as follows:

1. For input x M_D lays off $\log |x|$ tape in $|x| \cdot \log |x|$ steps and checks if x has the format

$$x = x_1 \# x_2 \# x_3$$

with x_1, x_2, x_3 in $(\Sigma-\#)^*$ and such that

$$|x_1 \# x_2 \# x_3| \leq \log |x|.$$

If not the input is rejected, otherwise:

2. M_D chekcs whether x_1 is a description of a Tm, say M_i, and x_2 is a proof in F that $T_i(n) \leq T(n)$ for all n. If not the input is rejected, otherwise:

3. On the marked off tape M_D searches for m such that

$$1 \leq m \leq n \quad \text{and} \quad |x_1|^2 \leq g(m).$$

If no such m is found the input is rejected, otherwise:

4. M_D simulates $x_1 = M_i$ on input

$$x = x_1 \# x_2 \# x_3$$

and accepts x iff M_i rejects it.

Note that the steps 1, 2 and 3 can all be carried out in time $n \cdot \log n$ for $n = |x|$, since log n tape can be layed-off in $n \cdot \log n$ steps and since all the other processes halt and are carried out on log n tape, the total time is bounded by $n \cdot \log n$ for these steps. Since we can only prove true properties about Turing machines in F we know that successful completion of step 2 guarantees that M_i runs in time $T_i(n) \leq T(n)$. Furthermore, since we can simulate a step of M_i computation in $|x_1|^2$ steps on M_D, the condition $|x_1|^2 \leq g(m)$ for some $m \leq n$ implies that $|x_1|^2 \leq g(n)$ and therefore M_D operates in time

$$T_D(n) \leq T_i(n) \cdot g(n) \leq T(n) \cdot g(n).$$

Thus

$$L(M_D) \text{ is in } TIME[T(n)\cdot g(n)].$$

On the other hand, $L(M_D)$ cannot be in F-TIME[T(n)] since this would imply that there exists a Tm M_i such that $L(M_i) = L(M_D)$ and there is a proof in the formal system F that $T_i(n) \leq T(n)$ and therefore we must have that $L(M_i) \neq L(M_D)$. Thus $L(M_i)$ is not in F-TIME[T(n)] and because of our hypotheses we know that

$$L(M_D) \in TIME[T(n)\cdot g(n)] - TIME[T(n)],$$

as was to be shown. ■

From the proof of Theorem 1 it immediately follows that for any recursive time bound a "slight" increase in the bound permits a new computation which is not provably computable in the old complexity bound.

Corollory 2: Let T(n), $T(n) \leq n\cdot\log n$, be a recursive function and let $g(n) \geq 1$ be a non-decreasing, unbounded recursive function. Then

$$F\text{-}TIME[T(n)] \overset{c}{\neq} TIME[T(n)\cdot g(n)].$$

The above theorem shows that the condition

$$F\text{-}TIME[T(n)] = TIME[T(n)]$$

yields very sharp hierarchy results for time bounded one-tape Tm computations. Thus the very interesting open problem is about the validity of the assumption

$$F\text{-}TIME[T(n)] = TIME[T(n)].$$

We conjecture that this condition holds for time bounds with certain "honesty" conditions. For example, the previous diagonalization results [4] required that the time bound T(n) be computable in time T(n) and on log T(n) tape. Thus we would expect that

$$\text{F-TIME}[T(n)] = \text{TIME}[T(n)]$$

for such functions as

$$T(n) = n \cdot \log n,\ T(n) = n^2,\ T(n) = 2^n, \text{ etc.}$$

In the next section, we will show that the corresponding conjecture holds for tape-bounded computations for which we can, in essence, show that a complexity class is equal to the corresponding provable complexity class iff the complexity class can be defined by a tape constructible bound.

On the other hand, our next result shows there exist recursive time bounds for which the classic complexity classes differ from the corresponding provable complexity classes. That is, we will show that there exist recursive, nomotonically increasing functions T(n) such that for some set A acceptable in time T(n) there is no Tm M_i which accepts A and for which it can be proven in F that $T_i(n) \leq T(n)$.

Though this theorem is only stated for one-tape Tm's it is easily seen that it holds for all computational complexity measures [5].

THEOREM 3: There exist recursive, monotonically increasing time bounds T(n) such that

$$\text{F-TIME}[T(n)] \neq \text{TIME}[T(n)].$$

Proof: By the Gap Theorem [1,5] we can effectively construct a recursive, monotonically increasing function $T_o(n) \geq n \cdot \log n$ such that

$$\text{TIME}[T_o(n)] = \text{TIME}[T_o(n)^2].$$

But by Theorem 1 we know that, choosing $g(n) = T_o(n)$, we get

$$\text{F-TIME}[T_o(n)] \neq \text{TIME}[T_o(n)^2].$$

Thus

$$\text{F-TIME}[T_o(n)] \neq \text{TIME}[T_o(n)],$$

as was to be shown. ■

By similar reasoning we can get the next independence result about non-existance of formal proofs of resource bounds.

Corollary 4: Let G(n) be a non-decreasing, unbounded recursive function. Then ere exists an arbitrarily large, non-decreasing recursive function T(n) such that for any formal system F (satisfying our assumptions)

$$\text{F-TIME}[G(T(n))] \subsetneq \text{TIME}[T(n)].$$

It should be observed that for every formal mathematical system F we can effectively construct a recursive bound T(n) such that for no Tm M_j, with $T(n) \leq T_j(n)$, can it be proven in F that M_j is total.

Note though that the proof of Theorem 3 and Corollary 4 are not based on such a size argument. The time bound $T_o(n)$ of Theorem 3 is such that for any formal mathematical system F (satisfying our previous assumptions) we must have

$$F\text{-}TIME[T_o(n)] \neq TIME[T_o(n)].$$

Thus we see that for the time bound $T_o(n)$, yielded by the Gap Theorem, for every formal system F there will be Tm's accepting a set A in time $T_o(n)$ but for none of these machines can it be proven in F that they run in time $T_o(n)$. This effect is strengthened by Corollary 4.

From the above we see that though the time bound $T_o(n)$ is effectively constructed the complexity class defined by $T_o(n)$ is such that we cannot effectively list names of Tm's which accept these sets and run in time $T_o(n)$. Thus we see that the originally surprising Gap Theorem describes a fact about non-constructive complexity classes. Or, stated differently, it is a result about a class of languages whose defining properties cannot be verified formally.

Our next result shows that a gap result can hold for provable complexity classes for a formal system F only if they differ from the corresponding non-constructively defined complexity classes and, furthermore, that we can constructively exhibit a set on which they differ. As a matter of fact, this implies that there is no proof in F that, the simply constructed Tm, M_D of Theorem 1 runs in time $T(n)\cdot g(n)$ nor is there a proof in F, for any other M_i for which $L(M_i) = L(M_D)$, that

$$T_i(n) \leq T(n)\cdot g(n).$$

Corollary 5: For a recursive function $T(n)$, $T(n) \geq n \cdot \log n$, and a non-decreasing, unbounded recursive function $g(n)$, $g(n) \geq 1$, we can have, a gap in the hierarchy of the provable complexity classes,

$$F\text{-}TIME[T(n)] = F\text{-}TIME[T(n) \cdot g(n)]$$

if and only if

$$F\text{-}TIME[T(n) \cdot g(n)] \subsetneq TIME[T(n) \cdot g(n)]$$

and we can effectively construct a set A such that

$$A \in TIME[T(n) \cdot g(n)] - F\text{-}TIME[T(n) \cdot g(n)].$$

Proof: From our assumption about F we know that

$$F\text{-}TIME[T(n) \cdot g(n)] \subseteq TIME[T(n) \cdot g(n)],$$

from Theorem 1 we know that

$$F\text{-}TIME[T(n)] \neq TIME[T(n) \cdot g(n)],$$

and therefore

$$F\text{-}TIME[T(n)] \subsetneq TIME[T(n) \cdot g(n)].$$

Furthermore we know that, for the effectively constructed Tm M_D of Theorem 1 we have

$$L(M_D) \in TIME[T(n) \cdot g(n)]$$

and from the hypotheses of this theorem it follows that

$$L(M_D) \in TIME[T(n) \cdot g(n)] - F\text{-}TIME[T(n) \cdot g(n)],$$

as was to be shown. ■

We believe that for sufficiently powerful formal systems F it can be proven in F for M_D of Theorem 1 that

$$T_D(n) \leq T(n) \cdot g(n),$$

If this conjecture is true then, as the next result shows, there are no gaps between provable complexity classes.

Corollary 6: If it can be shown in F that the Tm M_D, constructed in the proof of Theorem 1, runs in time

$$T_D(n) \leq T(n) \cdot g(n)$$

(which we know is true), then

$$\text{F-TIME}[T(n)] \neq \text{F-TIME}[T(n) \cdot g(n)].$$

Proof: Obvious.

3. OTHER MEASURES AND A.E. CONDITIONS

To gain some further insight when complexity classes are equal to the corresponding provable complexity classes and what can and cannot be proven about complexity of computations, we consider tape bounded computations. As in many other cases, it turns out that we can prove sharper results for tape bounded computations than for time bounded computations.

Let $L_i(n)$ denote the maximum amount of tape used by Tm M_i on inputs of length n. Let

$$\text{TAPE}[T(n)] = \{L(M_i) \mid L_i(n) \leq T(n)\}.$$

and

$$\text{F-TAPE}[T(n)] = \{L(M_i) \mid \vdash L_i(n) \leq T(n)\}.$$

We recall that a function T(n) is tape constructible iff there exists a Tm M which for input of length n lays off exactly T(n) tape squares without using more than T(n) tape.

In the next result we will consider all Tm 's with a fixed tape alphabet $\Sigma = \Sigma_o$ and denote the corresponding complexity classes by

$$\text{TAPE}_{\Sigma_o}[T(n)] \text{ and } \text{F-TAPE}_{\Sigma_o}[T(n)].$$

Our next result shows that in essence a complexity class coincides with the corresponding provable complexity class if and only if the class can be defined by a tape constructible bound.

Note that the following result asserts that for tape bounded computations if a set is accepted by some Tm M_i which runs on $L_i(n)$ tape that then this fact can be proven formally in F. Surprisingly, the corresponding result for time-bounded one-tape Turing machines is not known to be true.

THEOREM 7:

1. If $t(n) \geq n$ is tape constructible then
   $\text{TAPE}[t(n)] = \text{F-TAPE}[t(n)]$.
2. If $\text{TAPE}_{\Sigma_o}[T(n)] = \text{F-TAPE}_{\Sigma_o}[T(n)]$ then there exists a tape constructible t(n) such that
   $\text{TAPE}_{\Sigma_o}[T(n)] = \text{TAPE}_{\Sigma_o}[t(n)]$.

Proof: Since t(n) is tape constructible and we can prove elementary facts about Turing machines in the formal system F, there exists a Tm M_{i_t} with the following properties:

1. For any input of length n the Tm M_{i_t} lays off exactly t(n) tape squares without using more than t(n) tape.
2. M_{i_t} halts for all inputs w such that t(|w|) is finite.
3. It is provable in F that M_{i_t} computes a (partial recursive) tape constructible function.

Properties 1) and 2) follow from the definition of a tape constructible function and elementary facts about detecting cycling for tape bounded computations. Property 3) follows from the fact that M_{i_t} can be so chosen that from inspection of its description it is clear that it can only halt after printing a "1" on each tape square it has visited except the left most and right most squares which are marked with "#". Thus it is provable in F that M_{i_t} computes a tape constructible function. It is not necessarily provable in F that t(n) is a total function, but that is not needed for our proof.

Using the machine M_{i_t} (as a subroutine to lay off the desired amount of tape) we can effectively construct for any i a Tm $M_{\sigma(i)}$ with the following properties:

1. $M_{\tau(i)}$, rejects input w if M_i on w uses more than $t(|w|)$ tape, otherwise, $M_{\tau(i)}$ accepts w iff M_i accepts.
2. It is provable in F that $M_{\tau(i)}$ uses no more tape than M_{i_t}.

Thus we conclude that for a tape constructible t(n) for every M_i which runs on tape t(n) there exists an equivalent Tm $M_{\tau(i)}$ which provably in F uses no more than t(n) tape. Thus

$$\text{TAPE}[t(n)] = \text{F-TAPE}[t(n)].$$

Conversely, if

$$\text{TAPE}_{\Sigma_o}[T(n)] = \text{F-TAPE}_{\Sigma_o}[T(n)]$$

then we can effectively enumerate the set of Tm's (with tape alphabet Σ_o)

$$\{M_{i_j} \mid \vdash T_{i_j}(n) \leq T(n)\} = \{M_{i_j} \mid j \in I\}.$$

But then it follows that the function

$t(n) = \max\{\{n\} \cup \{T_{i_j}(n) \mid M_{i_j}$ is enumerated on n tape$\}\}$ is tape constructible since we can simulate all enumerated M_{i_j} on $T_{i_j}(n)$ tape and find the maximum. Note that the machine computing t(n) may have a larger tape alphabet than Σ_o. Furhter, we can see that

$$t(n) \leq T(n)$$

and if M_{i_j} is in TAPE[T(n)], then there exists an equivalent M_s such that $L_s(n) \leq t(n)$. Thus

$$\text{TAPE}[t(n)] = \text{TAPE}[T(n)]$$

as was to be shown. ■

From the previous reuslt it follows immediately thet if some set A is accepted on a tape-bound achieved by a Tm then this fact can be proven in F.

Corollary 8: For every (total) M_i

$$\text{TAPE}[L_i(n)] = \text{F-TAPE}[L_i(n)].$$

Proof: Obvious. ■

It should be pointed out that it is not known whether the corresponding result holds for one-tape (k-tape) Turing machines. The best known result for one-tape Tm's follows form [4].

Corollary 9: For every (total) one-tape Tm M_i

$$\text{TIME}[T_i(n)] \subseteq \text{F-TIME}[T_i(n)\cdot\log T_i(n)].$$

Proof: Similar to the diagonalization proof in [4]. ■

For k-tape Tm's, $k\geq 2$, we can exploit [8] to get the following result.

Corollary 10: Fore very (total) k-tape M_i

$$\text{TIME}_k[T_i(n)] \subseteq \text{F-TIME}_k[T_i(n)\log^*[T_i(n)]].$$

It would be very interesting to see whether the last two results connot be improved . Unless the rusult for one-tape Tm 's can be improved, it leaves us with the possibility that there can exist sets which are accepted by a Tm M_i running in time $T_i(n)$, but that the best recognition time we can prove in F for these sets in $T_i(n)\cdot\log T_i(n)$. This would be a rather shocking situation.

Note that Theorem 3 and Corollary 4 yield time bounds for computations which we connot prove in F, but these are not the

actual running times of the Turing machines performing these computations.

The previous results suggest that in the study of different computational complexity measures we should investigate how sharp resource bounds can be proven formally for these measures. A natural measure for the "provability" is given by results analoguous to Corollaries 8,9 and 10.

Next we show that if a set A is accepted in time $T_i(n)$ by some one-tape Tm , M_i, then it can be proven in F that such a machine exists, but we do not know whether we can prove for any specific machine accepting A that it runs in time less or equal to $T_i(n)$ (since our best provability result is given by Corollary 9.)

We first prove a result due to A. Meyer.

Lemma 11: For a recursive function $T(n) \geq n \cdot \log n$

$$\{L(M_i) \mid T_i(n) \leq T(n)\} = \{L(M_j) \mid \vdash_F T_j(n) \leq T(n) \text{ a.e.}\}$$

Proof: We construct a recursive function τ such that for all i

1. $L(M_i) = L(M_{\tau(i)})$ if $T_i(n) \leq T(n)$ else $L(M_{\tau(i)})$ is finite
2. $\vdash [T_{\tau(i)}(n) \leq T(n) \text{ a.e.}]$

The machine $M_{\tau(i)}$ has a much larger tape alphabet than M_i so that it can simulate M_i in time $\frac{1}{4} T_i(n)$. $M_{\tau(i)}$ operates as follows:

> On input w, $|w| = n$, $M_{\tau(i)}$ lays off $\log^3 n$ tape and tries to find on this tape a z, $|z| \leq \log^3 n$, such that
>
> $$T_i(|z|) > T(|z|).$$
>
> If such a z is found then the input w is rejected. If

not then $M_{\tau(i)}$ simulates M_i on w and accepts iff M_i accepts w.

It is easily seen that $M_{\tau(i)}$ satisfies condition 1, furthermore $T_{\tau(i)}(n) \leq T(n)$ a.e. and since we can prove elementary facts about Tm's in F we can choose a simple construction for τ such that it is provable in F that

$$T_{\tau(i)}(n) \leq T(n) \quad \text{a.e.},$$

where T(n) is given by some Tm. Note again that the proof of $T_{\tau(i)}(n) \leq T(n)$ a.e. does not depend on knowing that T(n) is a total function. ■

Theorem 12: For one-tape Tm 's for any recursive T(n), $T(n) \geq n \cdot \log n$, we have

$$\text{TIME}[T(n)] = \{L(M_i) \mid \vdash (\exists M_s)[M_s \equiv M_i \text{ and } T_s(n) \leq T(n)]\}$$

Proof: The previous result showed that for every $M_{\tau(i)}$ we can prove in F that

$$T_{\tau(i)}(n) \leq T(n) \quad \text{a.e.}$$

Therefore we can prove in F that there exists some equivalent Tm to $M_{\tau(i)}$ which runs in time T(n) everywhere by means of a table look-up for the initial input values. ■

It follows from Theorem 12 that if A can be accepted in time T(n) then there is a formal proof that there exists a Tm which accepts A in time T(n). Though we know from Theorem 3 that for many time bounds T(n) and any formal system F there are sets which are acceptable in time T(n) but for no Tm accepting these sets

can it be shown that they run in time T(n). Thus showing an interesting difference in what can be formally proven about the existance of time bounds without being able to prove these bounds for any Tm.

Corollary 13: There exist monotomically increasing, arbitrarily large recursive functions T(n) such that

$$\{L(M_i) \mid \vdash T_i(n) \leq T(n)\} \subsetneq \{L(M_i) \mid \vdash (\exists M_s)[M_s \equiv M_i \text{ and } T_s(n) \leq T(n)$$

Proof: Follows directly from Theorems 3 and 12. ■

References

[1] Borodin, A.B., "Computational Complexity and Existence of Complexity Gaps", J.ACM, Vol. 19, (1972), 158-174.

[2] Fischer, P.C., "Theory of Provable Recursive Functions", Trans. American Math. Soc. Vol 117, (1965), 494-520.

[3] Gordon, D., "Complexity class of Provable Recursive Functions", Typed paper, February, 1977.

[4] Hartmanis, J., "Computational Complexity of One-Tape Turing Machine Computations", J.ACM, Vol. 15, (1968), 352-339.

[5] Hartmanis, J. and J.E. Hopcroft, "An overview of the Theory of Computational Complexity", J.ACM, Vol. 18, (1971), 444-475.

[6] Hartmanis, J. and J.E. Hopcroft, "Independence Results in Computer Science", ACM SIGACT NEWS, Vol., 8, No. 4, (October-December 1976), 13-24.

[7] Hennie, F.C. and R.E. Stearns, "Two-Tape Simulation of Multi-Tape Turing Machines", J.ACM, Vol. 13, (1966), 533-546.

[8] Paul, W.J., "On Time Hierarchies", Proc. Ninth Annual ACM Symposium on Theory of Computing, (May 2-4, 1977), 218-222.

[9] Young, P., "Optimization Among Provably Equivalent Programs". To appear in J.ACM.