Theoretical Computer Science 73 (1990) 313-317
North-Holland

NOTE

STRONG TIME BOUNDS: NON-COMPUTABLE BOUNDS AND A HIERARCHY THEOREM *

J.M. ROBSON

Computer Science Department, Australian National University, GPO Box 4, Canberra, ACT 2601, Australia

Communicated by M. Nivat
Received June 1989

Abstract. A RAM program is said to run within a "strong" time bound T if on every sequence of n inputs it terminates within $T(n)$ instruction executions. There are some programs whose execution time in this sense is a non-computable function of n. It is shown that such programs are essential in the sense that some functions can be computed within a non-computable time bound but not within any computable time bound. Nevertheless, strong time bounds are subject to a powerful hierarchy theorem. The condition such as being time constructable which normally applies to the "lower" function in such theorems is replaced by a condition of being the minimum strong time bound for some program.

Introduction

Recent papers [1, 4] have stressed the utility of considering complexity in terms of number of arithmetic operations rather than the classical process of counting bit operations or Turing machine steps. Meyer auf der Heide [4] suggests the study of "strong" time bounds (the paper uses the alternative term "genuine") where the time (i.e. the number of operations from some RAM instruction set) for a computation is bounded by a function of the number of inputs rather than the input length. Naturally the existence of such a strong time bound depends on there being for each n, a finite bound on the time taken to compute the function on n inputs.

Results

This paper answers two of the questions left open by [4]. Firstly it is proved that, for one instruction set, there are functions which have strong time bounds but for which every strong time bound is a non-computable function. This was known not

* The work described here was done while the author was visiting the Institut Blaise Pascal.

to be true for the simple instruction set $\{+, -\}$ but it becomes true when we add the multiplication operation to the set. The only properties of this instruction set used in the proof are (a) that any polynomial equality can be tested within a strong time bound and (b) that the maximum number computed after n steps is at most the square of the maximum after $n-1$ steps. Property (a) is crucial to the proof; property (b) could be relaxed, for example to allow an exponentiation instruction, necessitating only minor changes in the proof.

Secondly it will be shown that, even for non-computable strong time bounds, a very tight hierarchy theorem exists for any instruction set containing $\{+, -\}$.

Preliminaries

We consider some enumeration of Turing machines with a binary input alphabet and some convention that classifies computations as accepting or not. The ith machine in this enumeration will be referred to as T_i and its time to halt on a blank input tape as $T(i)$; $T(i)$ can be taken as zero for machines which do not halt on blank input.

We take f, a boolean function of a single variable x which is computable but such that any function computable in time 2^{3x} differs from $f(x)$ on at least one argument x of length l except for a finite number of values of l. Note that the existence of such functions f is shown by an argument very similar to the diagonalization argument used in [2]. If f on an input x simulates T_i on the same input, where i is the integer obtained by deleting the most significant bit in the binary representation of x, and returns a different result if T_i terminates in time 2^{3x}, then clearly f differs from the function, if any, computed in 2^{3x} by T_j on every input consisting of j plus a power of 2 greater than j.

Denote by $D_i(x_1, \ldots, x_{13})$ the Diophantine equations in thirteen variables guaranteed to exist by [3] which have a unique solution if T_i halts on a blank input tape and no solution otherwise. We note that if the Turing machine does halt in time t, D_i can be written within some computable time bound, say $DT(i, t)$.

Let CONS, HD and TL be three linear time computable functions satisfying

$$\mathrm{HD}(\mathrm{CONS}(x, y)) = x, \qquad \mathrm{TL}(\mathrm{CONS}(x, y)) = y,$$

$$\mathrm{CONS}(x, y) > x, \qquad \mathrm{CONS}(x, y) > y.$$

Then let LIST be the function which packs an arbitrary number of integers into one:

$$\mathrm{LIST}(x_1) = x_1,$$

$$\mathrm{LIST}(x_1, \ldots, x_n) = \mathrm{CONS}(x_1, \mathrm{LIST}(x_2, \ldots, x_n)).$$

The necessity for non-recursive strong time bounds

Now consider the language L defined as $\{(x_1, \ldots, x_n) \mid n \geq 14, x_1, \ldots, x_{13}$ are the solution of D_n, $2^{\mathrm{LIST}(n,x_1,\ldots,x_{13},T(n))} \leq x_{14} < 2^{\mathrm{LIST}(n,x_1,\ldots,x_{13},T(n))+1}$, $f(x_{14})\}$.

Theorem 1. *L is recognizable within a strong time bound but not within a computable strong time bound, for the instruction set* $\{+, -, \times\}$.

Proof. (1) L is recognizable within a strong time bound. For any given n, either the equations D_n have no solution (in which case the checking of membership in L terminates in the time required to write down D_n and evaluate these polynomials) or they have a unique solution. If $x_1, \ldots, x_{13}$ are not this unique solution, then again the checking terminates in the same time. Next, if they are the solution, we know that Turing machine n does terminate on a black input tape; simulation of this halting computation computes $T(n)$ in a finite time and the value $2^{\mathrm{LIST}(n,x_1,\ldots,x_{13},T(n))}$ is also computed in a finite time. Finally, if x_{14} is found to be within the bounds stated, the time to check $f(x_{14})$ is certainly bounded because there are only a finite number of values of x_{14} for which it will be done.

(2) L is not recognizable within a strong time bound ϕ for any computable function ϕ. Suppose that L is recognizable within a strong time bound ϕ by a machine M. We will describe a Turing machine algorithm A computing f by using M. Given an input x, A finds the value L such that $2^L \leq x < 2^{L+1}$; finds $\mathrm{HD}(L)$, $\mathrm{HD}(\mathrm{TL}(L)), \ldots, \mathrm{HD}(\mathrm{TL}^{13}(L))$, $\mathrm{TL}^{14}(L)$ and rejects if any of these are undefined (call the values obtained $N, X_1, \ldots, X_{13}, \tau$); simulates the computation of M on N inputs of which the first 13 are $X_1, \ldots, X_{13}$, the fourteenth is x and the remainder are zero; rejects if M does not halt within time τ and otherwise returns the result returned by M. A may often fail to find the correct value of f but it is certain to find the correct value for those x which happen to lie in a range $[2^L, 2^{L+1})$ for an L such that $L = \mathrm{LIST}(n, x_1, \ldots, x_{13}, T(n))$ where T_n halts on blank input and the x_i are the solution of D_n, provided M has halted within time $T(n)$, which must happen if $\phi(n) \leq T(n)$; note that there are an infinite number of such L unless $\phi(n) > T(n)$ for all except a finite number of halting machines T_n. The time to carry out this computation is dominated by the step of simulating M; the computation of M has time bounded by τ which is less than x and all inputs are also less than x; thus the numbers computed are bounded by x^{2^x} so that the normal logarithmic cost RAM time of the computation is $O(x2^x \log x)$; hence A can simulate the computation in time $O(x^2 2^{2x} \log^2 x) = o(2^{3x})$. Hence, if there are an infinite number of L for which A works, A runs in time $o(2^{3x})$ and computes f correctly in an infinite number of ranges $[2^L, 2^{L+1})$. This contradicts the choice of f as a function which differs from every 2^{3x} computable function on some x in the range $[2^l, 2^{l+1})$ except for finitely many l. This contradiction implies that $\phi(n) > T(n)$ except for finitely many n but it is well known that no computable function can exceed $T(n)$ for all

except finitely many n since that would imply the decidability of the blank tape halting problem. □

A hierarchy theorem

Throughout this section all the machines referred to will have the same instruction set of which we assume only that it includes addition and subtraction. We will need to use a method of coding two non-negative integers and a single bit into two non-negative integers such that both the coding and the decoding can be carried out efficiently with such an instruction set. Such a function is:

$$code(n_1, n_2, b) = \text{if } n_1 > n_2 \text{ then } (n_1, n_2 + (n_1 + 1)(\text{if } b \text{ then } 1 \text{ else } 2))$$
$$\text{else}(n_1 + (n_2 + 1)(\text{if } b \text{ then } 1 \text{ else } 2), n_2).$$

Theorem 2. *Let $T(n)$ be a function such that there exists a machine M with strong time bound T where, for every n, there is a sequence of n inputs for which M takes time exactly $T(n)$. Then there is a function computable with strong time bound $O(T(n))$ but not with strong time bound $T(n)$.*

Proof. We define the function f with strong time bound $O(T(n))$ by describing a machine M' which computes it. M' with inputs $x_1, \ldots, x_n$

(a) decodes the input into two sequences $y_1, \ldots, y_n$ of integers and $z_1, \ldots, z_{\lfloor n/2 \rfloor}$ of bits by decoding x_i and x_{i+1} to give y_i, y_{i+1}, $z_{(i+1)/2}$ for odd i,

(b) constructs a description of a machine M'' from the bit sequence z,

(c) simulates machine M on the inputs y and machine M'' on the inputs x, in parallel (one step of M and one step of M''), until M halts,

(d) if M'' has halted its computation, returns a result different from the result of M'', else returns 0.

It should be clear that the decoding in (a) is carried out within a strong time bound $O(n)$ using only addition and subtraction, that the construction of M'' in (b) can be done in the same time bound for a reasonable interpretation of how a bit sequence describes a machine and that the simulations in (c) can be done in linear time using the same instruction set as the simulated machines. Hence the function f computed by M' is computable within the strong time bound $O(T(n) + n) = O(T(n))$.

Finally consider any function f_0 computed within strong time bound T by some machine M_0. If the bit sequence describing M_0 is $z_1, \ldots, z_l$, the $2l$ inputs on which M takes time $T(2l)$ are $y_1, \ldots, y_{2l}$ and $x_1, \ldots, x_{2l}$ is the sequence obtained by coding the sequences y and z, then when M' is run on inputs $x_1, \ldots, x_{2l}$, it will:

(a) decode the sequence x to obtain the sequences y and z,

(b) reconstruct the machine M_0,

(c) simulate M on the sequence y (a computation taking time exactly $T(2l)$) and M_0 on the sequence x (which by assumption takes time at most $T(2l)$),

(d) return a value different from that returned by M_0.

Thus $f(x_1, \ldots, x_{2l}) \neq f_0(x_1, \ldots, x_{2l})$, proving, as required, that f is not the function computed within strong time bound T by any machine. □

References

[1] L. Blum, M. Shub and S. Smale, On a theory of computation over the real numbers; NP-completeness, recursive functions and universal machines, *Bull. Amer. Math. Soc.*, to appear.

[2] S.A. Cook and R.A. Reckhow, Time bounded random access machines, *J. Comput. System Sci.* 7 (1973) 354-375.

[3] Y.V. Matijasevic, Enumerable sets are Diophantine, *Dokl. Akad. Nauk. U.S.S.R.* **191** (1970) 279-282 (translation in *Soviet Math. Dokl.* **11** (1970) 354-357).

[4] F. Meyer auf der Heide, On genuinely time bounded computations, in: B. Monien and R. Cori, eds., *Proc. 6th Ann. Symp. on Theoretical Aspects of Computer Science* (Springer, Berlin, 1989) 1-16.