MACHINE DEPENDENCE OF DEGREES OF DIFFICULTY[1]

MICHAEL ARBIB AND MANUEL BLUM

One may define a degree-of-difficulty relation for total recursive functions by saying that $f \geqq g$ (f is at least as difficult as g) if, to each program for computing f, there corresponds a program for computing g such that $g(x)$ takes no more steps to compute than $f(x)$ for almost all x.[2] Unfortunately, such an ordering is highly dependent on our choice of a mathematical model for a computer (cf. Hartmanis and Stearns [1, §III]). It is the purpose of this note to so modify our degree-of-difficulty relation as to lessen this machine dependence.

1. **Monoid-induced machine equivalence.** Let S be a monoid of 2-variable functions, increasing for x and y positive. The associative multiplication is

$$f * g(x, y) = f(x, g(x, y)),$$

and S contains the identity $e(x, y) = y$.

Let F and G be two partial recursive functions of a single variable. We say that F S-bounds G iff $F(x)$ defined implies $G(x)$ defined, and there exists a function $p \in S$ such that

$$p(x, F(x)) \geqq G(x) \text{ for almost all } x \text{ for which } F(x) \text{ is defined.}$$

A machine M may be thought of as supplied with a collection of programs P_i ($i = 1, 2, 3, \cdots$); ϕ_i is then the function computed by M when supplied with program P_i, and $\Phi_i(x)$ is the "number of steps" it takes to compute $\phi_i(x)$.[3] (For a given partial recursive function f, there will, in general, be infinitely many i such that $\phi_i = f$; cf. a universal Turing machine.) For such a machine M, we then say that

Received by the editors December 30, 1963.

[1] This work was supported in part by the U. S. Army Signal Corps, the Air Force Office of Scientific Research, and the Office of Naval Research; in part by the National Science Foundation (Grant G-16526), the National Institutes of Health (Grants NB-01865-05 and MH-04737-03), and the National Aeronautics and Space Administration (Grant NsG-496); and in part by the U. S. Air Force (ASD Contract AF 33(616)-7783).

[2] Rabin [2] defines a strict partial ordering $f \succ g$ (f more difficult than g) as the existence of a program for computing g such that with any program for f the computation of $f(x)$ takes more 'steps' than the computation of $g(x)$ for almost all x.

[3] By way of contrast, Ritchie [3] bases his degree of difficulty on the amount of tape, rather than the time, used by a machine (in his case, a Turing machine) during a computation.

DEFINITION 1. $f \geqq_M g$ (f is at least as difficult as g, using M) if and only if, to each program P_i for computing f, there corresponds a program P_j for computing g such that Φ_i S-bounds Φ_j.

In §3, we shall particularize our choice of S. We observe that Hartmanis and Stearns [1] essentially employ the choice

$$S = \{f \mid f(x, y) = ky, k \text{ a positive constant}\}.$$

(Though they treat sequences, whereas we treat functions on the integers, much of their material is, of course, applicable.)

We chose S to be a monoid with a monotony condition simply so that we might deduce:

THEOREM 1. $\geqq_M$ *is reflexive and transitive.*

In describing the machine dependence of the ordering $\geqq_M$, we want to know, for each machine M, which are those machines N that give rise to the same ordering:

DEFINITION 2. Two machines M and N are S-equivalent, $M \equiv_S N$ iff, for all partial recursive functions f and g,

$$f \geqq_M g \Leftrightarrow f \geqq_N g.$$

This equivalence relation is obtained from a partial ordering $\geqq_S$ on machines. We say that M is at least as complex as N (modulo S), and write $N \geqq_S M$, iff each program ${}^N P_i$ for machine N can be replaced by a corresponding program ${}^M P_j$ for machine M so that ${}^N\Phi_i$ S-bounds ${}^M\Phi_j$ (${}^N P_i$ and ${}^M P_j$ are programs for the same function). In other words, M is at least as complex as N if, up to elements of S, M can compute any function as quickly as N can. It is now immediate that:

THEOREM 2. (i) $\geqq_S$ *is reflexive and transitive.*

(ii) $M \equiv_S N$ *if and only if* $M \geqq_S N$ *and* $N \geqq_S M$.

§III of Hartmanis and Stearns [1] may now be reread as a treatment of equivalence classes under the monoid $\{p \mid p(x, y) = ky^{2n}\}$.

2. **Turing machines as programmed computers.** To fit Turing machines into our present discussion we have to separate the machine from the program. We do this by adopting the following, somewhat nonstandard, model of a Turing machine:

It is a device equipped with a container for cards, a tape scanner-printer-mover, and a tape that is infinite in both directions. The tape is divided into squares along its length and the scanner can look at one square at a time. The device can print the blank, or one of a finite set Σ of symbols, on the square it is examining and shift the tape one

square to right or left. The container can hold an arbitrarily large but finite number of cards, together called the program. On each card is printed a single 5-tuple $q_iS_jS_kmq_l$. The q_i denote internal states of the device, the S_j are tape symbols, and m is a move L (left), R (right) or N (none). When the device is in state q_i and scans the symbol S_j, it prints the symbol S_k, moves the tape m, and changes its internal state to q_l. If there is no card starting with q_iS_j in the container when the machine is in state q_i and scanning the symbol S_j, then the machine stops. Any program is allowed, subject to the condition that any 2 cards must differ in the initial pair q_iS_j.

We can associate with each program a partial recursive function ϕ as follows:

The program is placed in the container, an input integer x is written in some suitably encoded form as a finite string of symbols on the tape, the scanner is placed over the rightmost digit of this string, and the device is put in states q_0. The device then operates in accordance with the instructions printed in the program. If it never stops, we say $\phi(x)$ diverges. If it does stop, we let $\phi(x)$ be the integer which is obtained from the string of symbols on the tape by some standard decoding.

In the following discussion we let ${}^kT$ be the above machine with the constraint that $\Sigma = \{0, 1, \cdots, k-1\}$ (with each of these k numbers considered as a single symbol) and that the coding of input and output is to be in radix k ($k \geqq 1$). In particular, we shall be interested in machines equivalent to ${}^{10}T$.

Now let ${}^{\mu}T$ differ from ${}^{10}T$ only in that Σ is augmented by some fixed finitude of further symbols (${}^{\mu}T$ still uses a radix 10 input-output code).

PROPOSITION 1. (i) *If P is a program which causes ${}^{10}T$ to compute the function f, then P will also cause ${}^{\mu}T$ to compute f, and in exactly the same number of steps.*

(ii) *If ${}^{\mu}P_i$ is a program which causes ${}^{\mu}T$ to compute $f(x)$ in ${}^{\mu}\Phi_i(x)$ steps, then there exists a program ${}^{10}P_j$ which causes ${}^{10}T$ to compute $f(x)$ in ${}^{10}\Phi_j(x)$ steps, where*

$$p(x, {}^{\mu}\Phi_i(x)) \geqq {}^{10}\Phi_j(x)$$

on setting $p(x, y) = \frac{1}{2}(l+1)[k(k+1)+(k+y)(k+y+1)]+(3l-1)y$ *with* $k = [\log_{10} x+1]$ = *number of digits of input to* ${}^{\mu}T$, K = *number of symbols of alphabet of* ${}^{\mu}T$, *and* $l = [\log_{10} K+1]$.

PROOF. (i) is obvious.

(ii) follows on letting ${}^{10}P_j$ be the following simulation of ${}^{\mu}P_i$ on ${}^{10}T$:

a. The input string x to ${}^{\mu}T$ is replaced by an input string for ${}^{10}T$, each digit, d, being replaced by an l-tuple, $d0 \cdots 0$. This can be done in at most

$$\tfrac{1}{2}(l+1)k(k+1) \text{ steps,}$$

b. After ${}^{10}P_j$ has encoded x, it acts on each l-tuple of digits just as ${}^{\mu}P_i$ acts on each of ${}^{\mu}T$'s symbols. The number of steps required for this portion of the simulation is at most

$$(3l-1) \cdot {}^{\mu}\Phi_i(x).$$

c. When ${}^{\mu}P_i$ stops, its output is an integer y. Since ${}^{10}P_j$ simulates ${}^{\mu}P_i$, its tape at this point must contain an encoded y. ${}^{10}P_j$ then decodes this to obtain y, an operation taking not more than

$$\tfrac{1}{2}(l+1)k'(k'+1) \text{ steps,}$$

where k' is the number of digits in y. Since ${}^{\mu}P_i$ may, in an extreme case, merely print ${}^{\mu}\Phi_i(x)$ digits to the left of x, the best bound we can give for k' is $k+{}^{\mu}\Phi_i(x)$.

Our estimate

$$p(x, y) = \tfrac{1}{2}(l+1)[k(k+1) + (k+y)(k+y+1)] + (3l-1)y$$

is thus verified. Q.E.D.

3. **An equivalence class for** ${}^{10}T$. Since the purpose of this paper is to define computational difficulty to be invariant over a large subclass of machines, a measure of our success must be the size of the machine class containing ${}^{10}T$. But this class must depend on our choice of S. If we choose S very small, say the set consisting of the single function $e(x, y)=y$, then ${}^{\mu}T \not\equiv {}^{10}T$. On the other hand, if S is too large, e.g., the totality of total recursive functions $f(x, y)$ satisfying the monotony condition, then all machines are S-equivalent to ${}^{10}T$ and, therefore, all total recursive functions are equidifficult. Motivated by Proposition 1, we choose an $\tilde{S}$ large enough to have

$$(*) \qquad {}^{10}T \equiv_{\tilde{S}} {}^{\mu}T,$$

but small enough to make the $\tilde{S}$-equivalence class of ${}^{10}T$ interesting. In fact, we let

$$\tilde{S} = \{q(\log x, y) \mid q(z, y) \text{ is a polynomial, monotone increasing for positive } z \text{ and } y\}.$$

Quite obviously, the proof of Proposition 1 can be generalized to show

PROPOSITION 1a. *${}^{k}T$ is $\tilde{S}$-equivalent to a machine with a base k code, capable of printing l symbols, $l>k>1$.*

We next state and prove:

PROPOSITION 2. ${}^{10}T \equiv_{\tilde{S}} {}^{k}T$ *for* $k > 1$.

PROOF. We shall prove this proposition for the special case $k = 2$; a generalization of the proof to any integer k greater than 1 is quite easy. To simplify the proof, we make use of a machine (similar to ${}^{u}T$) which we call ${}^{\eta}T$: It has a single tape, is capable of printing the 10 symbols 0 through 9, and operates with a radix 2 code. The proof involves showing that ${}^{2}T \geqq_{\tilde{S}} {}^{10}T$ and that ${}^{10}T \geqq_{\tilde{S}} {}^{\eta}T$. From Proposition 1a we know that ${}^{\eta}T \equiv_{\tilde{S}} {}^{2}T$, and hence we may conclude that ${}^{2}T \equiv_{\tilde{S}} {}^{10}T$.

${}^{2}T \geqq_{\tilde{S}} {}^{10}T$: We simulate the program ${}^{2}P_i$ with ${}^{10}P_j$ which operates as follows:

a. ${}^{10}P_j$ takes the input integer x base 10 and converts it to x base 2. This can be done in less than $2([\log_2 x]+1)^2$ steps.

b. ${}^{10}P_j$ operates on x base 2 in precisely the same way as does ${}^{2}P_i$. This takes ${}^{2}\Phi_i(x)$ steps.

c. ${}^{10}P_j$ converts the result base 2 to base 10 in at most

$$2([\log_2 x] + 1 + {}^{2}\Phi_i(x))^2 \text{ steps.}$$

So ${}^{2}\Phi_i$ $\tilde{S}$-bounds ${}^{10}\Phi_j$ with

$$q(z, y) = 2(z + 1)^2 + y + 2(z + 1 + y)^2.$$

By essentially the same proof we have that ${}^{10}T \geqq_{\tilde{S}} {}^{\eta}T$. Q.E.D.

We shall finally state a very general theorem without proof: The proof, though tedious, does not involve any methods other than those contained in the proofs of Propositions 1 and 2.

THEOREM 3. ${}^{10}T$ *is* $\tilde{S}$*-equivalent to any machine with a radix* k *input and radix* l *output* ($k > 1$, $l > 1$), *with a finite number of tapes, each of finite dimension, and with a finite number of scanners on each tape.*

In the following example, two degrees of difficulty are compared in the radix 2 machine ${}^{2}T$. By Theorem 3, these comparisons remain valid on a wide variety of machines.

EXAMPLE. $2^x > {}^{2}_{T}x$.

Computation of 2^x involves printing x zeroes after a 1. Computation of the identity function x takes only 1 step. Clearly, x exceeds any given polynomial in $\log x$ for almost all x.

We also see why Proposition 2 demands that the radix k be greater than 1. Certainly ${}^{1}T \geqq_{\tilde{S}} {}^{10}T$, but it is not true that ${}^{1}T$ is $\tilde{S}$-equivalent to ${}^{10}T$. We may see this by considering 2^x. Actually, 2^x requires about 2^x steps just for writing time on ${}^{1}T$, but only needs about x steps on

^{10}T, and x cannot $\tilde{S}$-bound 2^x. Nevertheless, a proof like that for Theorem 3 shows that ^1T is $\tilde{S}$-equivalent to any machine with a radix 1 input-output code, with a finite number of tapes each of finite dimemsion, and with a finite number of scanners on each tape.

References

1. J. Hartmanis and R. E. Stearns, *On the computational complexity of algorithms*, Trans. Amer. Math. Soc. **117** (1965), 285–306.

2. M. O. Rabin, *Degree of difficulty of computing a function, and a partial ordering of recursive sets*, Hebrew University, Jerusalem, 1960.

3. R. W. Ritchie, *Classes of predictably computable functions*, Trans. Amer. Math. Soc. **106** (1963), 139–173.

Research Laboratory of Electronics,
Massachusetts Institute of Technology