The Journal of Symbolic Logic
Volume 40, Number 4, Dec. 1975

"HELPING": SEVERAL FORMALIZATIONS

NANCY LYNCH

§1. Introduction. Much recent work in the theory of computational complexity ([Me], [FR]. [S1]) is concerned with establishing "the complexity" of various recursive functions, as measured by the time or space requirements of Turing machines which compute them. In the above work, we also observe another phenomenon: knowing the values of certain functions makes certain other functions easier to compute than they would be without this knowledge. We could say that the auxiliary functions "help" the computation of the other functions.

For example, we may conjecture that the "polynomial-complete" problems of Cook [C] and Karp [K] and Stockmeyer [S2], such as satisfiability of propositional formulas or 3-colorability of planar graphs, in fact *require* time proportional to $n^{\log_2 n}$ to be computed on a deterministic Turing machine. Then since the time required to decide if a planar graph with n nodes is 3-colorable can be lowered to a polynomial in n if we have a precomputed table of the satisfiable formulas in the propositional calculus, it is natural to say that the satisfiability problem "helps" the computation of the answers to the 3-coloring problem. Similar remarks may be made for any pair of polynomial-complete problems.

As a further illustration, Meyer and Stockmeyer [MS] have shown that, for a certain alphabet Σ, recognition of the set of regular expressions with squaring which are equivalent to Σ^* requires Turing machine space c^n for some constant c, on an infinite set of arguments. We also know that this set of regular expressions, which we call RSQ, may actually be recognized in space d^n for some other constant d. Theorem 6.2 in [LMF] implies that there is some problem (not necessarily an interesting one) of complexity approximately equal to that of RSQ, which does not reduce the complexity of RSQ below c^n. It does not "help" the computation of RSQ.

Thus, we have many examples of a phenomenon we can think of as "helping", but we have no formal definition. We would like to have a definition which applies not only to problems with well-defined complexity, but to *all* recursive functions, even those with speed-up properties [B].

In this paper, we first define "complexity sequences" and present two results which motivate their use in one possible definition of "helping". We then present several other definitions which naturally suggest themselves. Finally, we show that these definitions are all essentially equivalent.

§2. Notation. We assume familiarity with the notation in [R]. In addition, we use the following:

Received March 26, 1974.

"a.e." (almost everywhere) ($\forall^\infty$) will mean "for all but a finite number of arguments". Similarly, "i.o." (infinitely often) will mean "for finitely many arguments".

We write "$a \dot{-} b$" to mean

$$\begin{array}{cl} a - b & \text{if } a \geq b, \\ 0 & \text{if } a < b. \end{array}$$

The composition "$g \circ t$", where t is a function of one variable and g is a function of two variables, will indicate $\lambda x[g(x, t(x))]$. Similarly, if g and h are both functions of two variables, "$g \circ h$" denotes $\lambda x, y[g(x, h(x, y))]$. An extended composition of two-variable functions "$g_1 \circ g_2 \circ g_3 \circ \cdots \circ g_k$" denotes $(\cdots((g_1 \circ g_2) \circ g_3) \circ \cdots \circ g_k)$. We use "$g^{(2)}$" as an abbreviation for $g \circ g$. "$g^{(3)}$" for $g \circ g \circ g$, etc.

"R_n" represents the set of total recursive functions of n integer variables.

To represent computation using a set for help, we will use "relatively computable functions" [LMF] which are partial recursive functions of one set variable and one integer variable. We assume a Gödel numbering of such functions [LMF], writing "$\varphi_i^{(\ )}$" for the ith function in this enumeration. We write "$\varphi_i^{(X)}$" for the partial X-recursive function computed by $\varphi_i^{(\ )}$ using set X. We write $\varphi_i^{(\phi)}$ as simply "φ_i", and thus obtain an acceptable Gödel numbering [R] for the partial recursive functions.

The standard way to derive the collection of relatively computable functions and a Gödel numbering thereof is by Davis' oracle Turing machine model [R]. The oracle Turing machine is exactly like an ordinary Turing machine, except that it has the ability to pause during its computation to ask the oracle about membership in the oracle set of a number written on a specified portion of the Turing machine's worktape. The oracle returns the answer, and, based on the answer, the Turing machine's computation proceeds. However, we avoid reference to a specific oracle Turing machine model, following instead the abstract approach in [LMF].

We must provide a way to measure the complexity of relatively computable functions. We use the following:

Definition 1. A relative complexity measure $\{\Phi_i^{(\ )}\}$ is a sequence of relatively computable functions satisfying:

(1) $(\forall i, x, X)\Phi_i^{(X)}(x)$ converges if and only if $\varphi_i^{(X)}(x)$ converges, and

(2) there exists $\Psi'^{(\ )}$, a relatively computable function, such that

$$\begin{aligned} (\forall i, x, y, X)\Psi'^{(X)}(\langle i, x, y\rangle) &= 1 \quad \text{if } \Phi_i^{(X)}(x) = y, \\ &= 0 \quad \text{otherwise.} \end{aligned}$$

These two properties are similar to the axioms of Blum for the complexity of partial recursive functions [B], and are used in the study of relative complexity in [LMF]. The most natural examples of relative complexity measures are time and space measures on oracle Turing machines.

We abbreviate $\Phi_i^{(\phi)}$ by Φ_i, obtaining a complexity measure on the set of partial recursive functions, in the sense of Blum.

Finally, if A is a set, $f \in R_1$, and b is a total function of one variable, then we say:

"$\text{Comp}^{(A)} f \leq b$ i.o." to mean $(\exists i)[(\varphi_i^{(A)} = f)$ and $(\Phi_i^{(A)} \leq b$ i.o.$)]$.

Similarly, we say:

"$\text{Comp}^{(A)} f > b$ i.o." to mean $(\forall i)[(\varphi_i^{(A)} = f) \Rightarrow (\Phi_i^{(A)} > b$ i.o.$)]$.

We use analogous definitions for "a.e." in place of i.o. We write "Comp f" in place of $\mathrm{Comp}^{(\Phi)} f$.

§3. Complexity sequences. Much work in "concrete" complexity theory involves attempts to find "the complexity" of problems (i.e. a single function describing the optimal running time for a solution to the problem). The speed-up theorem says that this is impossible in general, for recursive functions. However, we discover in [MF] that every recursive function has a certain sequence of functions describing its complexity. We show in this section that every recursive function has a sequence of functions of a type which we call "monotone-honest" describing its complexity, and also that a converse holds: any monotone-honest sequence describes the complexity of some recursive function. Thus, the complexities of recursive functions may be identified with monotone-honest sequences.

These results are similar to those of Schnorr [SS].

We define the two concepts central for this section:

DEFINITION 2. If h is a function of two variables, $f \in R_1$ and $\{p_i\}$ is a sequence of total functions of one variable, we say that $\{p_i\}$ is an *h-complexity sequence* for f provided:

$$(\forall i)(\exists j)[(\varphi_i = f) \Rightarrow (p_j \leq h \circ \Phi_i \text{ a.e.})],$$

and

$$(\forall i)(\exists j)[(\varphi_j = f) \text{ and } (\Phi_j \leq h \circ p_i \text{ a.e.})].$$

That is, to within amount h, the running-times for programs computing f are interlaced with the functions in the sequence $\{p_i\}$.

DEFINITION 3. If h is a function of two variables and $\{p_i\}$ is a sequence of total functions, we say that $\{p_i\}$ is *h-monotone-honest* if it satisfies the following three properties:

(1) $(\exists q \in R_1)(\forall i)[p_i = \varphi_{q(i)}]$,
(2) $(\forall i)[p_{i+1} \leq h \circ p_i \text{ a.e.}]$, and
(3) $(\forall i)(\exists j)[(p_i = \varphi_j) \text{ and } (\Phi_j \leq h \circ p_i \text{ a.e.})]$.

That is, the sequence is a recursively enumerable sequence of total recursive functions, approximately successively decreasing (hence "monotone"), and with every function in the sequence "honest". Honesty, the property which requires that a function be computable within running time which is not much greater than the size of the function itself, is explored in [MM].

We first state a result which shows that every recursive function has a complexity sequence which is monotone-honest. The ideas are essentially similar to those in [MF]; we include the construction here because it does not explicitly appear in the preceding paper. We omit the verification as it is analogous to the verification of Lemma 4 of [MF].

PROPOSITION 4. *There exists $h \in R_2$ such that $(\forall f \in R_1)(\exists \{p_i\})[\{p_i\}$ is an h-complexity sequence for f and $\{p_i\}$ is h-monotone-honest$]$.*

PROOF. We first use the *s-m-n* theorem to define a collection of partial recursive functions as follows:

Instructions for computing $\varphi_{\alpha(e,i)}(x)$.

1. If $i < e$, compute and output $\varphi_e(x)$ if it converges. (If it does not converge, the function will be undefined.)

2. If $i \geq e$, let $C = \{j \leq i \mid (\forall y \leq x)[(\Phi_j(y) \leq x \text{ and } \Phi_e(y) \leq x) \Rightarrow (\varphi_j(y) = \varphi_e(y))]\}$.

Find $j \in C \cup \{e\}$ such that $\Phi_j(x)$ is minimal. (If no $\varphi_j(x)$ converges, the function will be undefined.)

If several give the minimal value, consider the least such j. Compute and output $\varphi_j(x)$.

END OF CONSTRUCTION

For fixed e, i and x, these instructions tell us to use x steps to see which indices $\leq i$ could possibly represent the function φ_e. We run all of those possibilities "in parallel" on argument x, and output the first answer that arises.

For $\varphi_e = f$, we claim that $p_i = \Phi_{\alpha(e,i)}$ has the required properties. As in [MF], h is constructed using an elementary convergence argument, similar to many of those in [LMF]. Q.E.D.

The next result is a converse of the preceding proposition—that every monotone-honest sequence is a complexity sequence for some recursive function. My original proof was a complex direct diagonalization; the following simplified version is due to Schnorr [S].

THEOREM 5. *For any $g \in R_2$, there is an $h \in R_2$ with the following property: Whenever $\{p_i\}$ is a g-monotone-honest sequence, then $\{p_i\}$ is an h-complexity sequence for some recursive function.*

PROOF. We use the following lemma to obtain from $\{p_i\}$ a sequence of honest indices for the respective functions:

LEMMA 6. *For any $g \in R_2$, there exists $h \in R_2$ satisfying the following: If $\{\varphi_{q(i)}\}$ is a recursively enumerable sequence of total functions, and if $(\forall i)(\exists j)[(\varphi_{q(i)} = \varphi_j)$ and $(\Phi_j \leq g \circ \varphi_{q(i)}$ a.e.$)]$, then there exists $r \in R_1$ such that $\{\varphi_{r(i)}\}$ is also a recursively enumerable sequence of total functions, and*

$$(\forall i)[(\varphi_{q(i)} = \varphi_{r(i)} \text{ a.e.}) \text{ and } (\Phi_{r(i)} \leq h \circ \varphi_{r(i)} \text{ a.e.})].$$

PROOF. The result and construction are similar to Machtey's Lemma 3.5 in [Ma]. We assume without loss of generality that g is monotone increasing in both variables. We use the *s-m-n* theorem to define the following collection of functions:

Instructions for computing $\varphi_{\alpha(i,a)}(x)$.

1. See if $\Phi_a(i) \leq x$. If not, let $\varphi_{\alpha(i,a)}(x) = 0$.
2. Otherwise, consider

$$(\mu\langle y_1, y_2\rangle \leq x)(\forall z, y_2 \leq z \leq x)$$
$$[(\Phi_{\varphi_a(i)}(z) \leq x) \Rightarrow ((\Phi_{y_1}(z) \leq g \circ \varphi_{\varphi_a(i)}(z)) \text{ and } (\varphi_{y_1}(z) = \varphi_{\varphi_a(i)}(z)))].$$

Let

$$s = y_1 \quad \text{if } \langle y_1, y_2\rangle \text{ exists,}$$
$$= x \quad \text{otherwise.}$$

a. If $\varphi_s(x)$ converges and $\Phi_s(x) \leq g \circ \Phi_{\varphi_a(i)}(x)$, then let $\varphi_{\alpha(i,a)}(x) = \varphi_s(x)$.

b. Otherwise, if $\varphi_{\varphi_a(i)}(x)$ converges, let $\varphi_{\alpha(i,a)}(x) = 0$. If $\varphi_{\varphi_a(i)}(x)$ diverges, let $\varphi_{\alpha(i,a)}(x)$ be undefined.

END OF CONSTRUCTION

In the above construction, we assume that φ_a will be equal to q, where $p_i = \varphi_{q(i)}$, and that $\alpha(i, a)$ will be the required $r(i)$. We are trying to guess an honest index for the function p_i, or at least for a function which is equal to p_i almost everywhere. We let s be the first index that, within measure x, cannot be disproved to be such an honest index, and use s to determine the value of $\varphi_{\alpha(i,a)}(x)$.

If $\{\varphi_{q(i)}\}$ is a recursively enumerable sequence of total functions, then $\{\varphi_{r(i)}\}$ is also a recursively enumerable sequence of total functions. Also, $\varphi_{r(i)} = \varphi_{q(i)}$ a.e., since the guessed values for s will change as x increases until we eventually settle on one such that

$$((\varphi_s(z) = \varphi_{q(i)}(z)) \text{ and } (\Phi_s(z) \le g \circ \varphi_{q(i)}(z)))$$

for sufficiently large z.

To show that $\Phi_{r(i)} \le h \circ \varphi_{r(i)}$ a.e., we use a simple domain-of-convergence argument [LMF]. We define $h(x, y) = \max_{i \le x;\, a \le x} h'(x, y, i, a)$, where

$$\begin{aligned} h'(x, y, i, a) &= \Phi_{\alpha(i,a)}(x) \quad \text{if, for } s \text{ in the definition of } \varphi_{\alpha(i,a)}(x),\ \Phi_s(x) \le g(x, y), \\ &= 0 \qquad\qquad\quad \text{otherwise.} \end{aligned}$$

$h \in R_2$, since the condition in the definition of h' insures the convergence of $\varphi_{\alpha(i,a)}(x)$. The inequality $\Phi_{r(i)} \le h \circ \varphi_{r(i)}$ follows from the hypotheses on q.

Lemma 6 applied to $\{p_i\} = \{\varphi_{q(i)}\}$ leaves us to show that $\{\varphi_{r(i)}\}$ is a complexity sequence for some recursive function, in order to complete the proof of Theorem 5. Specifically, we need the following:

LEMMA 7.[1] *For any $g \in R_2$, there exists $h \in R_2$ satisfying the following: If $\{\varphi_{r(i)}\}$ is a recursively enumerable sequence of total functions, and for all i, $\Phi_{r(i)} \le g \circ \varphi_{r(i)}$ a.e. and $\varphi_{r(i+1)} \le g \circ \varphi_{r(i)}$ a.e., then $\{\varphi_{r(i)}\}$ is an h-complexity sequence for some $f \in R_1$.*

PROOF. The basic idea for this lemma is due to Schnorr, with a slight improvement due to Meyer. It is based originally on the speed-up construction, to be found in [B] and [MF].

We may surely assume g is monotone increasing in both variables. We first use $\{\varphi_{r(i)}\}$ to construct f. After this construction is completed, we will define the function h making $\{\varphi_{r(i)}\}$ an h-complexity sequence for f.

The function f is constructed depending uniformly on $\{\varphi_{r(i)}\}$ and since we will need this uniformity in the definition of h, we define f (using the *s-m-n* theorem and the recursion theorem) in the form $\varphi_{\alpha(a)}$. a is to be thought of as a partial recursive index for r.

Instructions for computing $\varphi_{\alpha(a)}(x)$.

1. For all $y < x$, test if $\Phi_{\alpha(a)}(y) \le x$. For any y such that this inequality holds,

[1] A careful implementation on a Turing machine of the construction given here for abstract measures essentially shows that for Turing machine time, h need only grow linearly in its second argument [SS].

simulate the computation $\varphi_{\alpha(a)}(y)$. Put a *mark* next to any index that becomes *cancelled* during any of these simulated computations.

2. See if $\varphi_{\varphi_a(0)}(x)$ converges. If not, $\varphi_{\alpha(a)}(x)$ will diverge. If it does, then, for each *unmarked* i, $i \le x$, see if $[(\forall j \le i)(\Phi_a(j) \le x)$ and $(\forall j, 1 \le j \le i)(\Phi\varphi_{a(j)}(x) \le g^{(2)} \circ \varphi_{\varphi_a(j-1)}(x))$ and $(\Phi_i(x) < \varphi_{\varphi_a(i)}(x))]$.

If there is no i for which this is true, let $\varphi_{\alpha(a)}(x) = 0$.

If there is, consider the least such i.

Let $\varphi_{\alpha(a)}(x) = 1 \dot{-} \varphi_i(x)$, and *cancel i.*

END OF CONSTRUCTION

Part 1 of the above construction is an attempt to avoid redundancy by not forcing us to cancel the same index more times than necessary. Part 2 tries to define f to be different from any partial recursive function which runs too quickly, assuming the successive functions in the given are decreasing, as they should be.

To show that this construction works, it remains to show:

$$(\forall i)(\exists j)[(\varphi_i = f) \Rightarrow (\varphi_{r(j)} \le h \circ \Phi_i \text{ a.e.})], \quad \text{and}$$
$$(\forall i)(\exists j)[(\varphi_j = f) \text{ and } (\Phi_j \le h \circ \varphi_{r(i)} \text{ a.e.})],$$

provided that $f = \varphi_{\alpha(a)}$, where a is a partial recursive index for r. To do this, we must, of course, define h.

For the first condition, we claim that

$$(\forall i)[(\varphi_i = f) \Rightarrow (\varphi_{r(i)} \le \Phi_i \text{ a.e.})].$$

To see this, assume that $\varphi_{r(i)} > \Phi_i$ i.o. For sufficiently large x, any index $<i$ whichever gets *cancelled* during the construction of f also gets *marked* in part 1 of the definition of $f(x)$ and so will not be considered in part 2 of the definition of $f(x)$. For sufficiently large x, $(\forall j \le i)(\Phi_a(j) \le x)$. Also, for sufficiently large x,

$$(\forall j, 1 \le j \le i)(\Phi_{r(j)}(x) \le g^{(2)} \circ \varphi_{r(j-1)}(x)),$$

by the hypotheses on $\{\varphi_{r(i)}\}$. Thus, for some x, the condition for index i in part 2 of the definition of $f(x)$ will be satisfied, and so $f(x)$ is defined to be different from $\varphi_i(x)$.

This implies that, for the first condition, any $h \ge \lambda x, y[y]$ will suffice.

For the second condition, we must show how to obtain j from i. For this purpose, we use the *s-m-n* theorem to define a collection of functions $\varphi_{\beta(a,u,v)}$, $\beta \in R_3$, such that if a is a partial recursive index for r, $u = i$ and F_v is a certain finite function, then $\varphi_j = \varphi_{\beta(a,u,v)}$ will have the desired properties.

We assume that $\{F_v\}$ is an effective enumeration of all finite functions. The construction of $\varphi_{\beta(a,u,v)}$ is similar to the construction of $\varphi_{\alpha(a)}$ with two exceptions: we only consider tests involving indices $\ge u$ in part 2 instead of all indices, and F_v provides values of the function on a certain finite set of arguments for which we do not use the general procedure.

Instructions for computing $\varphi_{\beta(a,u,v)}(x)$.

0. If $x \in$ domain F_v, let $\varphi_{\beta(a,u,v)}(x) = F_v(x)$. Otherwise, proceed through parts 1 and 2.

1. Exactly the same as part 1 of the definition of $\varphi_{\alpha(a)}(x)$.
2. See if $\varphi_{\varphi_a(u)}(x)$ converges. If not, $\varphi_{\beta(a,u,v)}(x)$ will diverge.

If it does, then for each *unmarked i*, $u \leq i \leq x$, see if $[(\forall j \leq i)(\Phi_a(j) \leq x)$ and $(\forall j, u+1 \leq j \leq i)(\Phi_{\varphi_a(j)}(x) \leq g^{(2)} \circ \varphi_{\varphi_a(j-1)}(x))$ and $(\Phi_i(x) < \varphi_{\varphi_a(i)}(x))]$.

If there is no i for which this is true, let $\varphi_{\beta(a,u,v)}(x) = 0$.

If there is, consider the least such i.

Let $\varphi_{\beta(a,u,v)}(x) = 1 \dot{-} \varphi_i(x)$.

END OF CONSTRUCTION

We first claim that for any partial recursive index for r, and for any u, there is some v such that $\varphi_{\beta(a,u,v)} = \varphi_{\alpha(a)}$. We simply let F_v provide the values of $\varphi_{\alpha(a)}(x)$ on small values of x. For sufficiently large values of x, any value of $i < u$ which-*ever* gets cancelled in the definition of $f = \varphi_{\alpha(a)}$ will be marked in part 1 of the definition of $f(x)$. Also, for sufficiently large x, $(\forall j, 1 \leq j \leq u)(\Phi_{r(j)}(x) \leq g^{(2)} \circ \varphi_{r(j-1)}(x))$, by the hypotheses on the sequence $\{\varphi_{r(i)}\}$, so that $\varphi_{\alpha(a)}$ and $\varphi_{\beta(a,u,v)}$ will yield exactly the same answers on these larger values of x.

We next define $h_1 \in R_2$. Letting $h = \max(h_1, \lambda x, y[y])$ will provide both the required inequalities.

Let $h_1(x, y) = \max_{a \leq x; u \leq x; v \leq x} h'(x, y, a, u, v)$, where

$$h'(x, y, a, u, v) = \begin{cases} \Phi_{\beta(a,u,v)}(x) & \text{if } \Phi_a(u) \leq x \text{ and } \Phi\varphi_{a(u)}(x) \leq g(x, y), \\ 0 & \text{otherwise.} \end{cases}$$

h', and hence h_1, is total recursive, since if the conditions in the definition of h' are satisfied, then $\varphi_{\beta(a,u,v)}(x)$ is easily shown to converge.

Now if $\varphi_\alpha = r$, $u = i$ and F_v is the finite function described above, then $h_1(x, \varphi_{r(i)}(x)) \geq h'(x, \varphi_{r(i)}(x), a, i, v)$ a.e. For sufficiently large x, $\Phi_a(i) \leq x$ and $\Phi_{r(i)}(x) \leq g(x, \varphi_{r(i)}(x))$. Thus, $h_1(x, \varphi_{r(i)}(x)) \geq \Phi_{\beta(a,i,v)}(x)$ a.e., as required.

Taking $h = \max(h_1, \lambda x, y[y])$ gives the required conditions. Q.E.D.

Proposition 4 and Theorem 5 show that it is reasonable to identify the complexities of recursive functions with monotone-honest sequences. For example, we may obtain the following:

COROLLARY 8. $(\exists h \in R_2)(\forall f \in R_1)(\exists$ 0-1 *valued* $f_1 \in R_1)$ [f *and* f_1 *have the same h-complexity sequence*].

PROOF. Immediate from Proposition 4 and Theorem 5, since the function constructed in Theorem 5's proof is 0-1 valued. It also follows from related work of Schnorr.

If $\{\Phi_i\}$ represents Turing machine worktape space rather than an arbitrary measure, the results have slightly cleaner statements. First, we sharpen the definitions:

DEFINITION 2S. If $f \in R_1$ and $\{p_i\}$ is a sequence of total functions of one variable, we say that $\{p_i\}$ is a *strict complexity sequence* for f if

$$(\forall i)(\exists i)[(\varphi_i = f) \Rightarrow (p_j \leq \Phi_i \text{ a.e.})], \quad \text{and}$$
$$(\forall i)(\exists j)[(\varphi_j = f) \text{ and } (\Phi_j \leq p_i \text{ a.e.})].$$

DEFINITION 3S. If $\{p_i\}$ is a sequence of total functions, we say that $\{p_i\}$ is strictly monotone-honest if it satisfies the following three conditions:

(1) $(\forall i)[p_i = \varphi_{q(i)}]$ for some total recursive q,
(2) $(\forall i)[p_{i+1} \leq p_i$ a.e.], and
(3) $(\forall i)(\exists j)[(p_i = \varphi_j)$ and $(\Phi_j \leq p_i$ a.e.)].

We may now obtain the following:

PROPOSITION 4S. *Assume $\{\Phi_i\}$ represents Turing machine space. If $f \in R_1$ and* Comp $f > \lambda x, y[x]$ *a.e., then there exists a sequence $\{p_i\}$ such that $\{p_i\}$ is a strict complexity sequence for f and $\{p_i\}$ is strictly monotone-honest.*

THEOREM 5S. *Again assume $\{\Phi_i\}$ represents Turing machine space. Assume $\{p_i\}$ is a strictly monotone-honest sequence. Finally, assume $(\forall i)[p_i > \lambda x, y[x]$ a.e.]. Then $\{p_i\}$ is a strict complexity sequence for some recursive function.*

Verification is left to the reader.

§4. Definitions of "helping".

We are now ready to present several alternative ways of formalizing the idea that a set helps the computation of a function. The first definition specifies that there is a way of computing the function using the set as an oracle which is much more efficient than any way of computing the function without using the set.

DEFINITION A. For any set A, any function g of two variables and any function f of one variable, we say that A *g-improves f* provided

$$(\exists i)[(\varphi_i^{\{A\}} = f) \text{ and } (\text{Comp} f > g \circ \Phi_i^{\{A\}} \text{ i.o.})].$$

The previous section motivates the next definition of helping. Since we here work with relative computability, it will be convenient to generalize the notation used in that section.

DEFINITION 9. If h is a function of two variables, f a function of one variable, A a set and $\{p_i\}$ a sequence of total functions of one variable, then we say that $\{p_i\}$ is an *h, A-complexity sequence* for f provided

$$(\forall i)(\exists j)[(\varphi_i^{\{A\}} = f) \Rightarrow (p_j \leq h \circ \Phi_i^{\{A\}} \text{ a.e.})], \text{ and}$$
$$(\forall i)(\exists j)[(\varphi_j^{\{A\}} = f) \text{ and } (\Phi_j^{\{A\}} \leq h \circ p_i \text{ a.e.})].$$

That is, to within amount h, the running-times for A-oracle programs computing f are interlaced with the functions in the sequence $\{p_i\}$. Proposition 4 asserts the existence of a function $h_1 \in R_2$, $h_1 \geq \lambda x[x]$, such that all $f \in R_1$ have h_1, ϕ-complexity sequences $\{p_i\}$ with several additional desirable properties. Henceforth, we use h_1 to refer to this specific function.

Our next formalization of "helping" asserts that the introduction of the set modifies a complexity sequence for the function.

DEFINITION B. If A is a set, g a function of two variables and f a function of one variable, we say *A g-destroys a complexity sequence for f* if there exists $\{p_i\}$, an h_1, ϕ-complexity sequence for f, such that $\{p_i\}$ is not a $g \circ h_1$, A-complexity sequence for f.

A closely related definition is the following, which specifies that using the set as an oracle modifies *all* complexity sequences for the function.

DEFINITION C. If A is a set, g a function of two variables and f a function of one variable, we say *A g-destroys all complexity sequences for f* if, for all $\{p_i\}$, h_1, ϕ-complexity sequences for f, $\{p_i\}$ is not a $g \circ h_1$, A-complexity sequence for f.

For the next definition, we consider that a set helps the computation of a function if its use as an oracle reduces the complexity of the function below some previous i.o. lower bound.

DEFINITION D. If A is a set, g is a function of two variables and f a function of one variable, we say *A destroys a g-i.o. lower bound for f* provided ($\exists b$, a total function) [(Comp $f > g \circ b$ i.o.) and (Comp$^{(A)} f \leq b$ a.e.)].

A closely related definition arises if we add on the hypothesis that b is recursive:

DEFINITION E. If A is a set, g a function of two variables and f a function of one variable, we say *A destroys a recursive g-i.o. lower bound for f* provided [$(\exists b \in R_1)$[(Comp $f > g \circ b$ i.o.) and (Comp$^{(A)} f \leq b$ a.e.)].

Finally, our last definitions are similar to the immediately preceding ones, but for a different sort of lower bound on the function's complexity:

DEFINITION F. If A is a set, g a function of two variables and f a function of one variable, we say *A destroys a g-a.e. lower bound for f* provided ($\exists b$, a total function) [(Comp $f > g \circ b$ a.e.) and (Comp$^{(A)} f \leq b$ i.o.)].

DEFINITION G. If A is a set, g a function of two variables and f a function of one variable, we say *A destroys a recursive g-a.e. lower bound for f* provided $(\exists b \in R_1)$[(Comp $f > g \circ b$ a.e.) and (Comp$^{(A)} f \leq b$ i.o.)].

Any of these provides a reasonable formalization of the idea that a set helps the computation of a function. We can express this concisely by saying "*A g-helps* f according to Definition A", etc.

§5. The equivalence theorem.

We now prove that all the definitions in the preceding section are equivalent in the following sense: if a set A helps a function f according to any of these definitions, it also helps f according to all of the others. The function g involved in the various definitions may differ, however.

We require two lemmas:

LEMMA 10. ($\exists h_2 \in R_2$, *monotone increasing in both variables*) $(\forall i, A)$($\forall f$, *a total function of one variable*)[$(\varphi_i = f) \Rightarrow$ (Comp$^{(A)} f \leq h_2 \circ \Phi_i$ *a.e.*)].

This lemma simply states the fact (obvious for time and space) that having an unnecessary oracle set does not increase the difficulty of computing a function. This lemma may be proved by an easy application of recursive relatedness [LMF].

The second lemma is of interest in itself; it states that if a function is complex i.o., then there is an infinite set of arguments on which it is complex a.e. The proof is a simplification of one due to Meyer.

LEMMA 11. *There exists $h_3 \in R_2$ with the following property: If $b, f \in R_1$ and* Comp $f > h_3 \circ b$ *i.o., then there exists an infinite recursive set X such that*

$$(\forall i)(\forall^\infty x)[(\varphi_i = f \text{ and } x \in X) \Rightarrow (\Phi_i(x) > b(x))].$$

PROOF. The result will follow by recursive relatedness if we give a proof for the Turing machine space measure. For space, we will prove the lemma with $h_3 = \lambda x, y[y]$.

Given b and f, we construct our set X by executing an effective sequence of stages numbered 0, 1, 2, . . . ; a new element x_n will be added into X at stage n.

During the construction, we *cancel* an index i when we have discovered that $\varphi_i \neq f$.

Stage n. Let $y = 0$ if $n = 0$. Otherwise, let $y = x_{n-1} + 1$.

Substage 1. For each *uncancelled* i, $0 \le i \le n$, see if $\Phi_i(y) \le b(y)$ and $\varphi_i(y) \ne f(y)$. If, for any i, both are true, *cancel* these i.

Substage 2. See if $(\forall i, 0 \le i \le n$ and i not cancelled$)\ [\Phi_i(y) > b(y)]$. If so, let $x_n = y$ and go on to stage $n + 1$. If not, let $y = y + 1$ and go back to substage 1.

END OF CONSTRUCTION

We must show that for each n, stage n must terminate. Assume not, and let n be the first stage that does not terminate. Then for each sufficiently large y, there exists an index i with $i \le n$, such that i is *never* cancelled, and $\Phi_i(y) \le b(y)$. By substage 1, for each sufficiently large y, we know that if $i \le n$ and i is never cancelled, then $[(\Phi_i(y) \le b(y)) \Rightarrow (\varphi_i(y) = f(y))]$. We may therefore run "in parallel" all Turing machines of index $i \le n$ such that i is never cancelled, with a finite modification for small arguments, and obtain a new Turing machine computing f and requiring only space b. But this contradicts the assumption $\text{Comp} f > b$ i.o.

Since the construction is effective and $(\forall i)(x_{i+1} > x_i)$, it follows that X is infinite and recursive.

We claim $(\forall i)(\forall^\infty x)[(\varphi_i = f \text{ and } x \in X) \Rightarrow (\Phi_i(x) > b(x))]$. This is because if $\phi_i = f$, then i is never cancelled during the construction of X. Then if $n \ge i$, substage 2 guarantees that $\Phi_i(x_n) > b(x_n)$. Q.E.D.

We now state and prove the equivalence. We say that a statement is true "for sufficiently complex $f \in R_1$" if $(\exists t \in R_1)(\forall f \in R_1)[(\text{Comp} f > t \text{ a.e.}) \Rightarrow$ (the statement is true for f)].

THEOREM 12. *For any $g \in R_2$, there exists $h \in R_2$ satisfying the following: For any sufficiently complex $f \in R_1$ and any recursive set A, if A h-helps f according to any of the definitions* A, B, C, D, E, F *or* G, *then A g-helps f according to all of these definitions.*

PROOF. We will show:

$$\begin{tikzcd}
G \arrow[r] & F \arrow[r] & A \arrow[rr, bend left] \arrow[ddll, bend left] & B \arrow[l] & C \arrow[l] \\
 & D \arrow[ur] & & & \\
E \arrow[uu] \arrow[ur] & & & &
\end{tikzcd}$$

G $\Rightarrow$ F and E $\Rightarrow$ D are obvious.

C $\Rightarrow$ B follows by Proposition 4.

A $\Rightarrow$ E. For g, f, A as above, if A g-improves f, then A destroys a recursive g-i.o. lower bound.

For, if A g-improves f, Definition A yields an index i such that $[(\varphi_i^{(A)} = f)$ and $\text{Comp} f > g \circ \Phi_i^{(A)}$ i.o.$)]$. Let $b = \Phi_i^{(A)}$. Since A is recursive, $b \in R_1$, and b satisfies Definition E.

D $\Rightarrow$ A. For g, f, A as above, with g monotone increasing in its second variable, if A destroys a g-i.o. lower bound for f, then A g-improves f.

For, Definition D yields a function b and an index i such that $\varphi_i^{(A)} = f$ and

$\Phi_i^{(A)} \leq b$ a.e. This index i satisfies Definition A, with g's monotonicity giving the desired result.

F $\Rightarrow$ A. Similar to the preceding case.

B $\Rightarrow$ A. For g, f, A as above, with g monotone increasing in its second variable and $g \geq \max(h_1, h_2)$, if A $g^{(2)}$-destroys a complexity sequence for f, then A g-improves f. (Here, h_1 and h_2 are previously specified functions.)

For, if A $g^{(2)}$-destroys a complexity sequence $\{p_i\}$ for f, then since $g \geq h_2$ and by Lemma 10, we obtain

$$(\exists i)(\forall j)[(\varphi_i^{(A)} = f) \text{ and } (p_j > g^{(2)} \circ h_1 \circ \Phi_i^{(A)} \text{ i.o.})].$$

Thus, since $\{p_i\}$ is an h_1, ϕ-complexity sequence for f, we have

$$(\forall k)[(\varphi_k = f) \Rightarrow (h_1 \circ \Phi_k > g^{(2)} \circ h_1 \circ \Phi_i^{(A)} \text{ i.o.})].$$

Since $g \geq h_1$ and g is monotone increasing in its second variable we obtain

$$(\forall k)[(\varphi_k = f) \Rightarrow (\Phi_k > g \circ \Phi_i^{(A)} \text{ i.o.})].$$

But this satisfies Definition A.

A $\Rightarrow$ C. For g, f, A as above, with g monotone increasing in its second variable and $g \geq h_1$, if A $g^{(3)}$-improves f, then A g-destroys all complexity sequences for f.

By Definition A,

$$(\exists i)(\forall j)[(\varphi_i^{(A)} = f) \text{ and } ((\varphi_j = f) \Rightarrow (\Phi_j > g^{(3)} \circ \Phi_i^{(A)} \text{ i.o.}))].$$

Let $\{p_i\}$ be any h_1, ϕ-complexity sequence for f. For any p_j, there exists $\varphi_k = f$ with $h_1 \circ p_j > \Phi_k$ a.e. Thus, $h_1 \circ p_j > g^{(3)} \circ \Phi_i^{(A)}$ i.o., for i as above.

Since $g \geq h_1$, this implies $p_j > g \circ h_1 \circ \Phi_i^{(A)}$ i.o., thus satisfying Definition C.

E $\Rightarrow$ G. For g, f, A as above, with g monotone increasing in its second variable, with $g \geq h_3$, and with Comp $f > \lambda x[g(x, 0)]$ a.e., if A destroys a recursive $g^{(2)}$-i.o. lower bound for f then A destroys a recursive g-a.e. lower bound for f.

For if A destroys a recursive $g^{(2)}$-i.o. lower bound, then

$$(\exists b \in R_1)[(\text{Comp} f > g^{(2)} \circ b \text{ i.o.}) \text{ and } (\text{Comp}^{(A)} f \leq b \text{ a.e.})].$$

It follows that Comp $f > h_3 \circ g \circ b$ i.o., so we may apply Lemma 11 and obtain set X. We define $c \in R_1$ as follows:

$$\begin{aligned} c(x) &= b(x) \quad \text{if } x \in X, \\ &= 0 \quad\quad\; \text{if not.} \end{aligned}$$

Then Comp $f > g \circ c$ a.e., by the lower bound on the complexity of f, and $\text{Comp}^{(A)} f \leq c$ i.o., thus satisfying Definition G.

It is now straightforward to combine the pieces and show that the theorem holds. Q.E.D.

Once again, if $\{\Phi_i^{(\ )}\}$ represents Turing machine work-tape space rather than an arbitrary measure, the result has a slightly cleaner statement. Specifically, we may obtain:

THEOREM 12S. *Assume $\{\Phi_i^{(\ )}\}$ represents Turing machine space. Assume $g \in R_2$ and g monotone increasing in both variables. Then for any sufficiently complex*

$f \in R_1$ and any recursive set A, if A g-helps f according to any of the definitions A, B, C, D, E, F *or* G, *then A g-helps f according to all of these definitions.*

Finally, the equivalence we have proved enables us to formulate without ambiguity the following conjecture. The statement is a generalization of Theorems 6.2 and 6.3 of [LMF], which essentially give the desired result for functions with well-determined complexities. We would like to have the same result for all recursive functions, including those with speed-up properties. We say that a statement is true for "arbitrarily complex B" if

$$(\forall t \in R_1)(\exists B, \text{ recursive})[(\text{Comp } C_B > t \text{ a.e.}) \text{ and (the statement is true for } B)].$$

CONJECTURE 13. *There exists $g \in R_2$ with the following property:*

$$(\forall f \in R_1)(\exists B, \textit{ arbitrarily complex})[B \textit{ does not g-help } f].$$

In attempting to prove this conjecture, we have at our disposal all of the equivalent formalizations of "helping".

REFERENCES

[B] M. BLUM, *A machine-independent theory of the complexity of recursive functions*, ***Journal of the Association for Computing Machinery***, vol. 14 (1967), pp. 322–336.

[C] S. A. COOK, *The complexity of theorem-proving procedures*, ***Conference Record of the Third Annual ACM Symposium on Theory of Computing***, 1971, pp. 151–158.

[FR] M. FISCHER and M. RABIN, *Super-exponential complexity of Presburger arithmetic*, Project MAC TM 43 (1974).

[K] R. KARP, *Reducibility among combinatorial problems*, ***Complexity of computer computations***, Plenum, 1972.

[LMF] N. LYNCH, A. MEYER and M. FISCHER, *Relativization of the theory of computational complexity*, ***Transactions of the American Mathematical Society*** (to appear).

[Ma] M. MACHTEY, *The honest subrecursive classes are a lattice*, ***Purdue University Technical Report 82.***

[ME] A. MEYER, *Weak monadic second order theory of successor is not elementary-recursive*, MIT manuscript, 1972.

[MF] A. MEYER and P. FISCHER, *Computational speed-up by effective operators*, this JOURNAL, vol. 37 (1972), pp. 55–68.

[MM] A. MEYER and R. MOLL, *Honest bounds for complexity classes of recursive functions*, 13th Annual Symposium on Switching and Automata Theory, IEEE, 1972.

[MS] A. MEYER and L. STOCKMEYER, *The equivalence problem for regular expressions with squaring requires exponential space*, 13th Annual Symposium on Switching and Automata Theory, IEEE, 1972.

[R] H. ROGERS, ***Theory of recursive functions and effective computability***, McGraw-Hill, New York, 1967.

[S] C. P. SCHNORR, private communication.

[S1] L. STOCKMEYER, Ph.D. Thesis, MIT, Department of Electrical Engineering, 1974.

[S2] ———, *Planar three-colorability is polynomial-complete.* ***SIGACT News***, vol. 5 (1973), pp. 19–25.

[SS] C. P. SCHNORR and G. STUMPF, *A characterization of complexity sequences.* ***Zeitschrift für Logik und mathematische Grundlagenforschung***, Spring/1975.

UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CALIFORNIA 90007