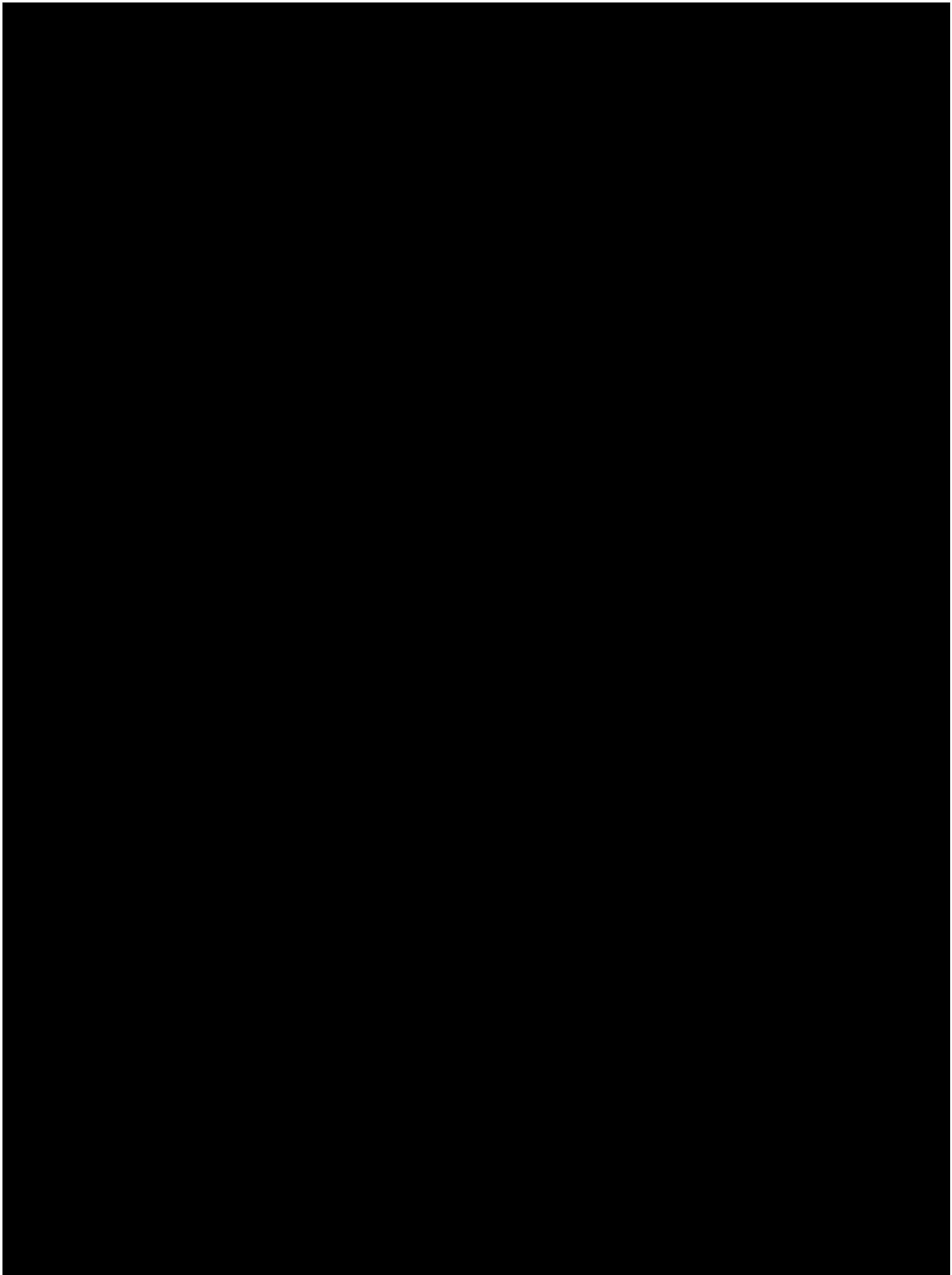


This blank page was inserted to preserve pagination.

MAC TR-99

RELATIVIZATION OF THE THEORY OF COMPUTATIONAL COMPLEXITY

NANCY A. LYNCH

JUNE, 1972

This research was supported by the
Advanced Research Projects Agency of
the Department of Defense under ARPA
Order No. 433, and was monitored by
ONR under Contract No. N00014-70-A-0362-0001.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

CAMBRIDGE MASSACHUSETTS 02139

This empty page was substituted for a blank page in the original document.

ABSTRACT

Blum's machine-independent treatment of the complexity of partial recursive functions is extended to relative algorithms (as represented by Turing machines with oracles). We prove relativizations of several results of Blum complexity theory, such as the compression theorem. A recursive relatedness theorem is proved, showing that any two relative complexity measures are related by a fixed recursive function. This theorem allows us to obtain proofs of results for all measures from proofs for a particular measure.

We study complexity-determined reducibilities, the parallel notion to complexity classes for the relativized case. Truth-table and primitive recursive reducibilities are reducibilities of this type, while other commonly-studied reducibilities are not.

We formalize the concept of a set helping the computation of function (by causing a saving in resource when used as an oracle in the computation of the function). Basic properties of the "helping" relation are proved, including non-transitivity and bounds on the amount of help certain sets can provide.

Several independence results (results about sets that don't help each other's computation) are proved; they are subrecursive analogs to degrees-of-unsolvability theorems, with similar proofs using diagonalization and priority arguments. In particular, we discuss the existence of a "universally-helped set," obtaining partial results in both directions. The deepest result is a finite-injury priority argument (without an apparent recursive bound on the number of injuries) which produces sets preserving an arbitrary lower bound on the complexity of a set.

Our methods of proof include proof for a simple measure (e.g. space) and appeal to recursive relatedness, diagonalization and priority techniques, and heavy use of arguments about the domain of convergence of partial recursive functions in order to define total recursive functions.

This empty page was substituted for a blank page in the original document.

ACKNOWLEDGEMENTS

I would like to thank Albert Meyer for his investment of time, enthusiasm and ideas.

I am also grateful to Amitava Bagchi and Michael Fischer for their interest and suggestions.

Finally, I am indebted to my husband Dennis for his patience and encouragement, as well as help with the cooking during the final stages of preparation of the thesis.

Work reported herein was supported in part by the National Science Foundation and in part by Project Mac, an MIT research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number N00014-70-A-0362-0001. Reproduction in whole or in part is permitted for any purpose of the United States Government.

This empty page was substituted for a blank page in the original document.

CONTENTS

This empty page was substituted for a blank page in the original document.

1. Introduction

Blum [B1] introduces an axiomatic framework for discussing the complexity of partial recursive functions of integer variables. In this thesis, we use a parallel approach for the case of relative algorithms (i.e. partial recursive functions of one integer and one set variable, as represented by Turing machines with oracles [D] [Ro1]). Our extension of Blum's ideas allows us to discuss axiomatically several problems impossible to formulate from the original set of axioms.

For example, we can formalize the idea that one function helps the computation of a second function; we can also give some meaning to "the complexity of a nonrecursive function."

The axioms we give include as special cases the "natural" measures on relative computations, namely the time and space measures on oracle Turing machines. Thus, the axiomatic statements of our various theorems are also true when interpreted for the specific measures. If we were to state and prove our theorems directly for the time and space measure, the results would be more precise and the proofs more intuitive. However, the axiomatic proofs are much shorter and cleaner; therefore, our general policy in this thesis is to state and prove results axiomatically, giving intuitive remarks about time and space wherever possible.

In Chapter 2, we present our axioms for relative complexity and prove some basic results suggested by theorems of non-relativized complexity theory.

The first important result is that any two measures satisfying the axioms are recursively related; the proof is by König's lemma. This theorem is important primarily because it provides an alternative method of proof of general-measure theorems. Certain types of theorems may be proved easily for a particular measure (usually space) and then the recursive relatedness between space and other measures can give the general result. We employ this method occasionally.

We note that the standard results of complexity theory, such as speed-up, compression and gap theorems [HH] all have full relativizations with proofs exactly like the usual proofs. Several partial relativizations are also true; we prove some which are of interest or will be of later use. In particular, we prove a relativization of the combining lemma [HH], which states that the complexity of a computation is closely related to the complexity of its subcomputations. This will imply some later results; it is the first example of our use of a method of proof which we call the "domain-of-convergence" method, and which is used in axiomatic proofs throughout the thesis.

In Chapters 3-6, we study questions natural to treatment within relative complexity theory.

A notion which parallels that of a complexity class [McC] [McCMe] in the relativized theory is that of a "complexity-determined reducibility," which we study in Chapter 3.

To any class $\mathcal{C}$ of functions corresponds:
$\{(A,B) \mid A$ is computable from B within measure equal to some function in $\mathcal{C}\}$.
For certain classes $\mathcal{C}$, this provides a reasonable reducibility.

Truth-table reducibility [Rol] and the relation "primitive recursive in" [K] are examples of reducibilities of this type, while other commonly-studied reducibilities such as many-one and one-one reducibilities [Rol] are not.

We show that neither truth-table reducibility nor primitive recursive reducibility can be completely specified by a single bound function (i.e. a singleton class $\mathcal{C}$). However, each may be so specified on any countable class of oracle sets, as we show by a relativization of the Meyer-McCreight union theorem [McC] [McCMe]. For example, there is a function t such that:

$(A \leq_{tt} B \quad \Leftrightarrow \quad$ A is computable from B within measure t)

for all arithmetical sets A and B.

By selecting special classes of functions $\mathcal{C}$, we may define new complexity-determined reducibilities; for example, by letting:

$$\mathcal{C} = \{\text{A-recursive functions}\}$$

for some set A, we define a reducibility somewhere between truth-table and Turing reducibility, which we call "A-reducibility." By considering all sets A, we arrive at a hierarchy of reducibilities.

A relativization of the compression theorem shows that:

$(\forall A,B)$[(A-reducibility = B-reducibility) $\Leftrightarrow$ (A-recursive functions and B-recursive functions are the "same size")].

This fact reduces questions about the reducibility hierarchy to purely recursion-theoretic questions; we prove several results about this hierarchy, for example, that there exist Turing-incomparable sets determining the same reducibility.

In Chapter 4, we try to establish a formalism within which to discuss questions such as the following:

(1) Which sets make the computation of a function easier than it would be without the help of these sets?

(2) How much help (increase in speed) does an oracle for some set provide in a computation?

We propose several possible definitions of "helping," each of which provides a reasonable way of discussing the concept. Briefly, we define helping of the computation of a function on either an infinite set of arguments or on "almost all" arguments. We also discuss helping in the sense of lowering the complexity of a function below a given lower bound function.

We then present a series of basic results which can be formalized using the definitions. First, we show that any set whose complexity is small cannot give much help to the computation of any function. We then show that any recursive set has arbitrarily complex recursive sets (with their complexity closely determined) that do help its computation.

As done by Trachtenbrot ("autoreducible sets") [T1], we formalize the idea of a set helping its own computation, by having values of its characteristic function at different arguments strongly interdependent. We then present a result of Paterson [P] proving the existence of such sets, of complexity approximately equal to any given monotone running time.

Independence results (theorems that state that certain sets do not help the computation of certain recursive functions) are analogous to theorems about functions having a certain lower bound on their complexity; diagonalization is the only tool we have for proving them. We prove a version of the following statement:

> "There exist pairs of complex recursive sets that don't help each other's computation."

We use a diagonalization method in the proof, based on work by Meyer, M.J. Fischer and Trachtenbrot; priorities are used. We first construct a set with no interdependence between the values of its characteristic function at different arguments. We then split this set into two pieces and argue that neither piece can help the other's computation.

This result illustrates proof techniques which will be used in a more complicated fashion in Chapter 6. It has several interesting corollaries, including the fact that "helping" is not a transitive relation.

Since the independent sets are constructed by a diagonalization, it is difficult to understand much about them. A more interesting result would arise if we could arbitrarily fix one of the sets. Thus, in Chapters 5 and 6, we ask the following question:

Which is true?

(1) There is a recursive set A whose computation is helped by all sufficiently complex recursive sets B (a "universally-helped set"), or

(2) For all recursive sets A, there exist arbitrarily complex recursive sets B that don't help the computation of A.

We obtain some partial results in both directions, using different interpretations of "helping."

In Chapter 5, we produce the strongest results we can to obtain the first answer. We note that the complexity axioms are sufficiently general to be satisfied by various "pathological" measures; specifically, that any recursive set will be a "universally-helped set" in some relative complexity measure. From here on, we use a mechanism for eliminating such trivial cases.

We go on, in theorem 5.2, to construct sets which are not "universally-helped," but which are "almost universally-helped," in the sense that they are helped by all recursive sets whose complexity is "nicely determined." More specifically, for any recursive function h, we obtain a recursive set A_h such that the computation of A_h's characteristic function is helped on infinitely many arguments by any recursive set whose complexity is (to within accuracy h) equal to a running time. This is the strongest result we have obtained in the direction of answer (1).

In Chapter 6, we work in the opposite direction, beginning with a recursive set A and constructing sets B not helping A's computation. As before, we use diagonalization and priority techniques in obtaining our results. There are two major results in the chapter.

The first theorem, theorem 6.2, states the following:

If we have a recursive set A and a recursive function t_A with the property that every Turing machine computing A's characteristic function requires more than t_A space on an infinite set of arguments, then there are arbitrarily complex recursive sets B such that every B-oracle Turing machine computing A's characteristic function still requires more than t_A space on an infinite set of arguments.

The proof idea is due to Machtey [Ma1] and involves a diagonalization with simple priorities.

The second theorem, theorem 6.3, states that, provided we restrict our attention to functions t_A which are running times, we have a similar result to theorem 6.2 for a different type of lower bound t_A. Namely, if we have a recursive set A and a total running time t_A with the property that every Turing machine computing A's characteristic function requires more than t_A space on almost all arguments, then there are arbitrarily complex recursive sets B such that every B-oracle Turing machine computing A's characteristic function still requires more than t_A space on almost all arguments.

This theorem is the deepest result in the thesis. The diagonalization required is considerably more complicated than that required for theorem 6.2, and involves a finite-injury priority argument in which there is no apparent recursive bound on the number of times a requirement may be injured.

The independence results serve to demonstrate that there exist arbitrarily complex pairs of recursive sets which are recursive for "different reasons."

There is, of course, no conflict between the results of Chapters 5 and 6, as we show.

Open problems are interspersed throughout the thesis as they arise, and are collected in Chapter 7. Also, in Chapter 7, we present additional open problems and directions for further research. One particular direction mentioned is that studied by Symes in [Sy], where he considers helping not only by oracle sets, but also by partial functions. In general, we would like to formalize other notions of "helping," specifically those which represent the way in which a subroutine "helps" the computation of a function computed by a computer program.

2. Notation, Axioms and Basic Results

We assume familiarity with the notation used by Rogers [Ro1].

We use "$(\overset{\infty}{\forall}x)$" and "a.e. (x)" to mean "for all but a finite number of x." When no confusion is likely, we simply write "a.e." ("almost everywhere").

Similarly, "$(\overset{\infty}{\exists}x)$" or "i.o. (x)" means "for infinitely many x," and we write "i.o." to mean "infinitely often."

We write "$a \dot{-} b$" to mean $\begin{cases} a - b & \text{if } a \geq b \\ 0 & \text{if } a < b. \end{cases}$

The composition "$g \circ t$" where t is a function of one variable and g is a function of two variables, will indicate $\lambda x[g(x,t(x))]$.

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

"$R_n^{(A)}$" represents the set of total A-recursive functions of n integer variables.

"P_n" represents the set of partial recursive functions of n integer variables.

"$P_n^{(A)}$" represents the set of partial A-recursive functions of n integer variables.

We write "$\uparrow$" for divergence and "$\downarrow$" for convergence of computations.

"$\lfloor k \rfloor$" represents the integer part of k.

For any i, A, if $\varphi_i^{(A)}(x)\uparrow$, we use the convention that $\varphi_i^{(A)}(x) = \infty$. By convention, $\infty \leq \infty$, and $n < \infty$ for any $n \in N$.

The notions of "relative algorithm" and of an enumeration of relative algorithms $\{\varphi_i^{( )}\}$ are amply described in [Rol, §9.2]. Specifically, we use the following:

Definition 2.1: A sequence $\{\varphi_i^{( )}\}$ of relative algorithms is called "acceptable" if:

(1) $\{\varphi_i^{( )}\}$ includes all relative algorithms

(2) Universal Property:

$$(\exists \psi \in \{\varphi_i^{( )}\})(\forall i,x,A)\ [\psi^{(A)}(\langle i,x\rangle) = \varphi_i^{(A)}(x)]$$

(3) s-m-n Property:

$$(\forall \psi \in \{\varphi_i^{( )}\})(\exists s \in R_1)(\forall i,x,A)\ [\varphi_{s(i)}^{(A)}(x) = \psi^{(A)}(\langle i,x\rangle)].$$

We discover by methods analogous to those used in [Ro2] that:

Lemma 2.2: Let $\{\varphi_i^{( )}\}$ and $\{\hat{\varphi}_i^{( )}\}$ be any two acceptable orderings of relative algorithms. Then there exists a recursive isomorphism r such that:

$$(\forall A,i)\ [\varphi_{r(i)}^{(A)} = \hat{\varphi}_i^{(A)}].$$

Lemma 2.2 will make our theory independent of the particular formalism chosen. We will generally refer to the development in [Rol] or to the notion of an oracle Turing machine when precision is required.

We now define a "relative complexity measure."

Definition 2.3: A relative complexity measure $\Phi^{(\ )}$ is a collection of partial functions from N to N, $\{\Phi_i^{(A)}\}$, one for each (i,A), satisfying the following two conditions:

(1) $(\forall i,A)\ [\text{domain } \varphi_i^{(A)} = \text{domain } \Phi_i^{(A)}]$

(2) There exists $\psi^{(\ )}$, a relative algorithm, such that:

$$(\forall i,x,y,A)\quad \psi^{(A)}(\langle i,x,y\rangle) = \begin{cases} 1 & \text{if } \Phi_i^{(A)}(x) = y \\ 0 & \text{otherwise} \end{cases}$$

We abbreviate $\varphi_i^{(\phi)}$ as φ_i, and $\Phi_i^{(\phi)}$ as Φ_i. The functions Φ_i are often referred to informally as "running times." There is no confusion here with the usual Gödel numbering notation, as $\{\varphi_i^{(\phi)}\}$ is an acceptable Gödel numbering for the partial recursive functions [Ro2].

A note on our choice of axioms: axiom (1) is surely reasonable, but it may be thought that axiom (2) is stronger than we ought to assume. However, both axioms are satisfied by all natural measures on relative computations (i.e. time and space on oracle Turing machines). Also, axiom (2) is plausible in that it merely requires the existence of a single "unified description" of any measure.

Thus, for the time measure, axiom (1) says that a computation takes a finite amount of time if and only if it converges, and axiom (2) says that we can effectively tell if a computation halts in a given number of steps. For the space measure, axiom (1) says that a computation uses a finite number of tape squares if and only if it converges, while axiom (2) says that we can effectively tell if a computation halts without exceeding a given amount of workspace.

Later in the chapter, we verify that these axioms hold for time and space measures.

These axioms are extremely simple, and similar to Blum's axioms for partial recursive functions [Bl]. We will see in the following chapters that they are quite powerful.

We refer to [Sy, Chapter 3] for some interesting results using these axioms. In particular, we shall use the fact that $\Phi_i^{(A)}(x)$ is a partial recursive function of x, uniformly in A and i. That is:

$$(\exists a)(\forall A,i,x)\ [\varphi_a^{(A)}(\langle i,x\rangle) \simeq \Phi_i^{(A)}(x)].$$

In spite of the theory's independence of the particular formalization of relative algorithm, enumeration and measure, it is desirable to keep in mind the natural measures (time and space on oracle Turing machines). The particular oracle Turing machine model we will use is as follows:

Each Turing machine has four semi-infinite tapes: an input tape, an output tape, an oracle tape and a worktape. The first three are marked in binary, with the exception that the input tape has a marker to indicate the end of the input. The worktape has k possible symbols, for some number k which depends on the machine. We assume for definiteness that the input and output heads cannot move left. Also, the machine cannot write on its input tape or read from its output tape. There are otherwise no restrictions on the operation of the machine, other than the usual Turing machine constraints [Rol].

This Turing machine is designed to be used in conjunction with an

"oracle" for any set. (An X-oracle is an unspecified agent having information about set X.) This is done as follows:

In addition to its other states, the Turing machine may have a state called "INTERROGATE." When the machine enters this state, it asks the oracle whether the number currently written on the oracle tape is a member of the oracle set. The oracle gives its answer by causing the machine to enter either of two different states. The oracle tape is then automatically erased, the oracle tape head reset to the first tape square, and the computation is allowed to continue.

Each oracle Turing machine may be described by a flowchart or some other finite description. The machine's description is independent of the particular oracle set used, so the same oracle machine may be used with any oracle. The finite descriptions may be enumerated in a natural way. We identify $\varphi_n^{(\ )}$ with the n^{th} machine description in this enumeration; our enumeration is "acceptable," and so there is no notational inconsistency with usage in [Rol].

We now define two measures on this machine model:

$T^{(\ )}$, time measure

For any i, x, A, we define $T_i^{(A)}(x)$ to be the total number of steps executed in the computation $\varphi_i^{(A)}(x)$. Here, each oracle interrogation counts as a single step.

It is clear that the axioms for relative complexity are satisfied; for instance, to discover if $T_i^{(A)}(x) = y$, $\Psi^{(A)}(\langle i,x,y\rangle)$ must construct the machine $\varphi_i^{(\ )}$, then simulate $\varphi_i^{(A)}(x)$ for y steps to see if it

converges.

$S^{(\ )}$, space measure

For any i, x, A, we define $S_i^{(A)}(x)$ to be the maximum of the number of worktape squares visited and the number of oracle tape squares visited during the computation $\varphi_i^{(A)}(x)$, provided that $\varphi_i^{(A)}(x)\downarrow$. Otherwise, we let $S_i^{(A)}(x) = \infty$.

Axiom (1) is satisfied by definition. To see that axiom (2) is also satisfied, we note that for any i, x, y and A, if $\varphi_i^{(A)}(x)$ operates for $(i)(i^y)(y)(2^y)(y)(\log x)$ steps without exceeding space y, it must be in an infinite loop and hence will not converge. This bound arises since if the machine is ever twice in the same state, with the same worktape contents, the same worktape head position, the same oracle tape contents, the same oracle tape head position and the same input tape head position, it must be in an infinite loop. The six factors in the above expression represent bounds on the number of different possibilities for each of the six items.

Thus, to see if $S_i^{(A)}(x) = y$, we need only simulate $\varphi_i^{(A)}(x)$ for $(i)(i^y)(y)(2^y)(y)(\log x)$ steps to see if it converges.

We note that our machine model has linear speed-up [HLS] for machines that don't use their oracle tapes. That is, given any $\epsilon > 0$ and any such machine $\varphi_i^{(\ )}$, we can effectively find $\varphi_j^{(\ )} = \varphi_i^{(\ )}$ such that for all sets A, $\epsilon \cdot S_i^{(A)} \geq S_j^{(A)}$ a.e.

We also note that the space measure has the following property, sometimes called the "parallel computation property": [LR]

There exists a recursive function η such that for all i and j,

$$\varphi_{\eta(i,j)}(x) = \begin{cases} \varphi_i(x) & \text{if } S_i(x) \leq S_j(x) \\ \varphi_j(x) & \text{otherwise} \end{cases}$$

and $S_{\eta(i,j)}(x) = \min(S_i(x), S_j(x))$.

This property, which essentially allows us to re-use the same tape squares for different portions of a computation, often makes it very easy to prove theorems for the space measure. It also causes some results for space measure to be sharper than those for other measures. We will point out such cases where they occur.

Theorems concerning the complexity of partial recursive functions [B1] [HH] [McC] all have straightforward full relativizations with proofs parallel to the original proofs. For example, from Blum's speed-up theorem [B1] we obtain:

Proposition 2.4: (relativized speed-up theorem)

$$(\forall A)(\forall s \in R_2^{(A)})(\exists f \in R_1^{(A)},\ f \text{ 0-1 valued})(\forall i)$$
$$[(\varphi_i^{(A)} = f) \Rightarrow (\exists j)(\varphi_j^{(A)} = f \ \wedge\ s(x, \Phi_j^{(A)}(x)) \leq \Phi_i^{(A)}(x) \text{ a.e.})].$$

(That is, for every program for f using an A-oracle, there is an a.e. much faster program also using an A-oracle.)

The proof is exactly like the usual proof of the speed-up theorem, using a relativization of the recursion theorem in place of the recursion theorem itself.

More interesting and useful are partial relativizations of the results on complexity of partial recursive functions. Following are

several examples.

Our first theorem asserts that any two relative complexity measures are related by a fixed recursive function. Its usefulness lies in enabling us to draw conclusions about one relative measure from hypotheses about another relative measure, as we do in some of the results following the theorem.

Theorem 2.5: (recursive relatedness)

If $\Phi^{(\ )}$ and $\hat{\Phi}^{(\ )}$ are two relative complexity measures on the same acceptable Gödel numbering $\{\varphi_i^{(\ )}\}$, then there exists $r \in R_2$ such that:

$$(\forall A,i)\ [\Phi_i^{(A)} \leq r \circ \hat{\Phi}_i^{(A)} \text{ a.e.}]$$

$$\text{and } (\forall A,i)\ [\hat{\Phi}_i^{(A)} \leq r \circ \Phi_i^{(A)} \text{ a.e.}].$$

Proof: We require a lemma which is a direct consequence of König's lemma ("Endlichkeitslemma," [Rol,Ex. 9-40]) and which will be used in several later theorems as well.

Lemma 2.5.1: Suppose we have a recursive function f of k integer variables and one set variable. Suppose that f is total. Suppose finally that $(\forall x_1,\ldots,x_k)\ [f'(x_1,\ldots,x_k) = \max_{A \subseteq N} f(x_1,\ldots,x_k,A)]$. Then $f' \in R_k$.

Proof of lemma 2.5.1: The computation of $f'(x_1,\ldots,x_k)$ may be carried out as follows:

Generate a "computation tree" for the function $f(x_1,\ldots,x_k,A)$ as A ranges over all subsets of N. Each branch of the tree must terminate,

since $f(x_1,\ldots,x_k,A)$ converges for all sets A. Therefore, the entire tree is finite and we will eventually finish generating it. We can then take the maximum of the outputs on all branches as the value of $f'(x_1,\ldots,x_k)$.

Proof of theorem 2.5, continued: By symmetry, it suffices to obtain $r \in R_2$ satisfying the first inequality.

We define $r(x,y) = \max_{i\leq x} p(i,x,y)$, where

$$p(i,x,y) = \max_{A\subseteq N} p'(i,x,y,A), \text{ and}$$

$$p'(i,x,y,A) = \begin{cases} \Phi_i^{(A)}(x) & \text{if } \hat{\Phi}_i^{(A)}(x) = y \\ 0 & \text{otherwise.} \end{cases}$$

p' is a total recursive function of three integer variables and one set variable. Therefore, by lemma 2.5.1, $p \in R_3$. Thus, $r \in R_2$.

To see that r has the required properties, we consider a particular A and i.

If $\hat{\Phi}_i^{(A)}(x)$ diverges, the inequality holds by convention.

If $\hat{\Phi}_i^{(A)}(x)$ converges and $x \geq i$, then:

$$r(x,\hat{\Phi}_i^{(A)}(x)) \geq p'(i,x,\hat{\Phi}_i^{(A)}(x),A)$$
$$\geq \Phi_i^{(A)}(x), \qquad \text{as required.}$$

QED

Remark 2.5.2: The recursive isomorphism between any two acceptable enumerations of relative algorithms (lemma 2.2) allows us to conclude the recursive relatedness of relative complexity measures on two different enumerations. Specifically, we obtain:

"If $\{\varphi_i^{(\ )}\}$ and $\{\hat{\varphi}_i^{(\ )}\}$ are any two acceptable enumerations of relative algorithms, with relative complexity measures $\Phi^{(\ )}$ and $\hat{\Phi}^{(\ )}$ respectively, then there exists a recursive isomorphism f and a function $r \in R_2$ such that:

$$(\forall A,i)\ [\Phi_i^{(A)} \leq r \circ \hat{\Phi}_{f(i)}^{(A)} \text{ a.e.}]$$

and $$(\forall A,i)\ [\hat{\Phi}_{f(i)}^{(A)} \leq r \circ \Phi_i^{(A)} \text{ a.e.}]."$$

The proof is a simple modification of the proof of theorem 2.5, using the recursive isomorphism whose existence is given by lemma 2.2.

Theorem 2.5 and remark 2.5.2 provide an alternate method to general axiomatic proof for certain types of theorems about relative complexity measures. The method is to prove the theorem for one specific measure, and then apply theorem 2.5 (or remark 2.5.2) to obtain the result for all measures. We will use this new proof method in some cases; as an example of its use, we give the following corollary to theorem 2.5 and remark 2.5.2.

The result has two parts; in part (1) we see that (just as in the non-relativized case) there exist arbitrarily complex functions. However, in contrast to the non-relativized case, part (2) shows that inherently complex functions cannot be 0-1 valued. In fact, their complexity must result from the size of the function values.

First, a definition:

Definition 2.6: Assume B is a set, $f \in R_1$ and g is a total function of one variable.

"$\mathrm{Comp}^{(B)}$ $f > g$ i.o. (a.e.)" means

$$(\forall i)\,[\varphi_i^{(B)} = f \Rightarrow \Phi_i^{(B)} > g \text{ i.o. (a.e.)}].$$

"$\mathrm{Comp}^{(B)}$ $f \leq g$" means

$$(\exists i)\,[\varphi_i^{(B)} = f \wedge \Phi_i^{(B)} \leq g].$$

"$\mathrm{Comp}^{(B)}$ $f \leq g$ i.o. (a.e.)" means

$$(\exists i)\,[\varphi_i^{(B)} = f \wedge \Phi_i^{(B)} \leq g \text{ i.o. (a.e.)}].$$

"Comp $f > g$ i.o." means

$\mathrm{Comp}^{(\phi)}$ $f > g$ i.o., and similarly for the other abbreviations.

If $f = C_A$ for some set A, we may write "Comp A" in place of "Comp f."

We are now ready to state and prove the corollary:

Corollary 2.5.3: Let $\Phi^{(\ )}$ be any relative complexity measure. Then:

(1) $(\forall f,\ f \text{ total})(\exists g,\ g \text{ total})(\forall A)$

$[\mathrm{Comp}^{(A)}\ g > f \text{ a.e.}]$.

(2) $(\forall h,\ h \text{ total})(\exists f,\ f \text{ total})(\forall g,\ g \text{ total})$

$[(g \leq h \text{ a.e.}) \Rightarrow (\exists A)(\mathrm{Comp}^{(A)} g \leq f \text{ a.e.})]$.

Proof: (1) Let r be the function obtained by applying theorem 2.5 to $\Phi^{(\ )}$ and $T^{(\ )}$. We may assume without loss of generality that r is monotone nondecreasing in its second variable.

Given f, let $g(x) = 2^{r(x,f(x))} + 1$.

If $\varphi_j^{(A)} = g$, then clearly $(\forall x)[T_j^{(A)}(x) > r(x,f(x))]$, since it requires $r(x,f(x)) + 1$ steps merely to output the result in binary.

But $r(x,\Phi_j^{(A)}(x)) \geq T_j^{(A)}(x)$ a.e., by theorem 2.5.

Thus, $r(x,\Phi_j^{(A)}(x)) > r(x,f(x))$ a.e.

$\Phi_j^{(A)}(x) > f(x)$ a.e., as required.

If the relative complexity measure $\Phi^{(\ )}$ is on an enumeration of relative algorithms other than oracle Turing machines, we apply remark 2.5.2 in place of theorem 2.5 and obtain the same result.

(2) Let r be the function obtained by applying theorem 2.5 to $\Phi^{(\ )}$ and $S^{(\ )}$, again chosen to be monotone nondecreasing in its second variable.

Assume h is given.

Define $f(x) = r(x,x^2 + h^2(x))$.

Now consider any g with $g \leq h$ a.e.

Let $A = \{\langle x,g(x)\rangle \mid x \in N\}$.

It is straightforward to design a machine $\varphi_j^{(\ )}$ such that $\varphi_j^{(A)} = g$ and for which $S_j^{(A)}(x) \leq x^2 + h^2(x)$ a.e. For instance, the machine $\varphi_j^{(A)}$ on argument x can operate by successively computing $\langle x,0\rangle$, $\langle x,1\rangle$, $\langle x,2\rangle$,..., and asking if each is in A. If so, the machine terminates with the appropriate output.

The bound $x^2 + h^2(x)$ results from the particular form of the pairing function used [Rol, § 5.3].

But then $\Phi_j^{(A)}(x) \leq r(x,S_j^{(A)}(x))$ a.e., by theorem 2.5.

$\leq r(x,x^2 + h^2(x))$ a.e.

$= f(x)$.

So $\Phi_j^{(A)}(x) \leq f(x)$ a.e., as required.

As in (1), if the relative complexity measure $\Phi^{(\ )}$ is on an enumeration of relative algorithms other than oracle Turing machines, we apply remark 2.5.2 in place of theorem 2.5 and obtain the same result.

QED

Informally, corollary 2.5.2 shows that (1) functions must be as complex as their size, and (2) given the proper oracle, a function need be no more complex than its size.

Henceforth, whenever we use this new method of proof, we will appeal to "recursive relatedness"; it will be understood that we intend this to mean we are applying theorem 2.5 or remark 2.5.2, whichever is appropriate, in a fashion similar to that used in the proof of corollary 2.5.3.

The non-relativized compression theorem [B1] asserts the existence of a recursive "compression function" h such that whenever we are given any total running time Φ_i, we can obtain a 0-1 valued function not computable in measure $\leq \Phi_i$, but computable in measure $\leq h \circ \Phi_i$.

Lemma 2.7 is a relativization of this result; it asserts the existence of a recursive "compression function" h such that whenever we are given <u>any</u> total function g, we can obtain an oracle set B and a 0-1 valued function not computable from a B-oracle in measure g, but computable in measure $h \circ g$.

This lemma will later be used to prove theorem 3.6.

<u>Lemma 2.7</u>: Assume we are given a relative complexity measure $\Phi^{(\ )}$. Then $(\exists h \in R_2)(\forall g,\ g \text{ total})(\exists B,A)$

(1) $\text{Comp}^{(B)}A > g$ i.o.

and (2) $\text{Comp}^{(B)}A \leq h \circ g$.

<u>Proof</u>: Given g, we define $B = \{\langle x,g(x)\rangle \mid x \in N\}$.

Define:

$$C_A(x) = \begin{cases} 1 \dot{-} \varphi_{\pi_1(x)}^{(B)}(x) & \text{if } \Phi_{\pi_1(x)}^{(B)}(x) \leq g(x), \\ 0 & \text{otherwise} \end{cases}$$

A is thus defined from B by a diagonalization which insures that the first condition is satisfied. To verify the second condition, we must define h.

First, define a relative algorithm $\varphi_j^{(\ )}$ as follows:

$$(\forall x, X)\ \varphi_j^{(X)}(x) = \begin{cases} 1 \dot{-} \varphi_{\pi_1(x)}^{(X)}(x) & \text{if } (\exists z)[\langle x,z\rangle \in X] \text{ and} \\ & \Phi_{\pi_1(x)}^{(X)}(x) \leq \mu z[\langle x,z\rangle \in X], \\ 0 & \text{if } (\exists z)[\langle x,z\rangle \in X] \text{ and} \\ & \Phi_{\pi_1(x)}^{(X)}(x) > \mu z[\langle x,z\rangle \in X], \\ \infty & \text{otherwise.} \end{cases}$$

(A note on this definition: the existence of the relative algorithm ψ given by axiom (2) of definition 2.3 immediately implies that the tests for inequality may be made effectively in x and X.)

Now define $h(x,y) = \max_{X \subseteq N} h'(x,y,X)$, where

$$h'(x,y,X) = \begin{cases} \Phi_j^{(X)}(x) & \text{if } \langle x,y\rangle \in X, \\ 0 & \text{otherwise.} \end{cases}$$

h' is total recursive in x, y and X, since:

$$\langle x,y\rangle \in X \Rightarrow \varphi_j^{(X)}(x)\downarrow$$
$$\Rightarrow \Phi_j^{(X)}(x)\downarrow, \text{ by axiom (1).}$$

Therefore, $h \in R_2$, by lemma 2.5.1.

For the particular g, A and B under consideration, we compare

the definition of C_A with the definition of $\varphi_j^{(\ )}$ and conclude that $\varphi_j^{(B)} = C_A$.

Also, since $\langle x,g(x)\rangle \in B$ for all x, it follows that:

$$h(x,g(x)) \geq h'(x,g(x),B)$$
$$= \Phi_j^{(B)}(x) \qquad \text{for all x, as required.}$$

QED

Remark: We note that the proof of lemma 2.7 actually provides a result considerably stronger than that stated. Namely,

$$\text{"}(\forall h_1 \in R_2)(\exists h_2 \in R_2)(\forall g,\ g \text{ total})(\forall B)(\exists A)$$
$$[\text{Comp}^{(B)}g \leq h_1 \circ g \text{ a.e.} \Rightarrow (1)\ \text{Comp}^{(B)}A > g \text{ i.o., and}$$
$$(2)\ \text{Comp}^{(B)}A \leq h_2 \circ g \text{ a.e.}]\text{."}$$

To prove this result, we can either modify the given proof of lemma 2.7, or note that the result holds for space measure and use recursive relatedness.

The condition "$\text{Comp}^{(B)}g \leq h_1 \circ g$ a.e." is an example of an "honesty condition" - one which specifies that a function has a running time which is approximately equal to its size. Honest functions (a generalization of running times, as we will later show) are extensively studied in [MeMo].

Honesty conditions will turn out to be necessary hypotheses for many of our later theorems, particularly in chapters 5 and 6. There, for simplicity, we will usually require that a function be a running time, whereas a less restrictive honesty hypothesis would have been sufficient.

In lemma 2.7, the first conclusion may be sharpened to assert that $\text{Comp}^{(B)}A > g$ a.e., rather than merely i.o. This is done by introducing two additional tricks into the construction.

The first is a sharper form of diagonalization in the construction of A which makes g an a.e. lower bound on A's complexity. The basic construction is due to Rabin and may be found in [HH].

Rabin's method defines C_A at successively larger values of x, in order. Thus, computing $C_A(x)$ for any x requires first computing $g(0), g(1), \ldots, g(x)$. In order to keep the complexity of A as small as possible, we introduce the second modification, due to Blum: we compute C_A on arguments not in order of size of the arguments, but in order of size of the values of g.

Since both of these ideas will be used in the succeeding chapters, we give the detailed construction:

Theorem 2.8: Assume we are given a relative complexity measure $\Phi^{(\ )}$. Then:

$(\exists h \in R_2)(\forall g,\ g \text{ total and } g \geq \lambda x\lceil x\rceil)(\exists B,A)$

(1) $\text{Comp}^{(B)}A > g$ a.e.,

and (2) $\text{Comp}^{(B)}A \leq h \circ g$

Proof: Given g, we define B as before.

We define a relative algorithm $\gamma^{(\ )}$ as follows:

For any X, $\gamma^{(X)}$ will be defined in stages; thus, to compute $\gamma^{(X)}(x)$, we begin executing stages in the definition of $\gamma^{(X)}$ until

the value of $\gamma^{(X)}(x)$ becomes defined. At each stage, at most one additional integer is added to the domain of $\gamma^{(X)}$.

During the construction, an index i will become "cancelled" when we have insured that $\gamma^{(X)} \neq \varphi_i^{(X)}$.

Stage n in the definition of $\gamma^{(X)}$:

Find the smallest integer y for which there exists an integer $x \leq y$ such that $\gamma^{(X)}(x)$ was not defined at an earlier stage and $\langle x,y \rangle \in X$, and for this y the smallest x.

(It is possible that this search may not terminate, in which case $\gamma^{(X)}$ will diverge at all arguments for which it has not already been defined.)

When $\langle x,y \rangle$ has been found, we find the smallest uncancelled $i \leq x$ such that $\Phi_i^{(X)}(x) \leq y$.

If no such i exists, define $\gamma^{(X)}(x) = 0$.

If i does exist, define $\gamma^{(X)}(x) = 1 \dot{-} \varphi_i^{(X)}(x)$ and cancel i.

In either case, go on to stage n + 1.

END OF CONSTRUCTION

Verification: We let A be a set such that $\gamma^{(B)} = C_A$. (This is possible since $\gamma^{(B)}$ is 0-1 valued and total.)

We claim $\text{Comp}^{(B)}A > g$ a.e.

For if not, then for some i, $\varphi_i^{(B)} = C_A$ and $\Phi_i^{(B)} \leq g$ i.o.

But after some stage n in the construction of $\gamma^{(B)} = C_A$, all the indices smaller than i which ever get cancelled have already been cancelled. But for some x such that $C_A(x)$ is defined after stage n,

$$\Phi_i^{(B)}(x) \leq g(x),$$

so that we will define:

$$C_A(x) = 1 \dot{-} \varphi_i^{(B)}(x), \text{ a contradiction.}$$

To verify the second conclusion, we choose j such that $\varphi_j^{(\ )} = \gamma^{(\ )}$, define h and proceed exactly as in the proof of lemma 2.7.

QED

Remark: As for lemma 2.7, the property of B that we actually require in this theorem is that B makes the function g honest (i.e. g can be computed from a B-oracle within measure approximately equal to g). We can thus obtain the more general result:

$$\text{"}(\forall h_1 \in R_2)(\exists h_2 \in R_2)(\forall g,\ g \text{ total and } \geq \lambda x\lceil x\rceil)(\forall B)(\exists A)$$
$$\lceil \text{Comp}^{(B)} g \leq h_1 \circ g \text{ a.e.} \Rightarrow (1)\ \text{Comp}^{(B)} A > g \text{ a.e., and}$$
$$(2)\ \text{Comp}^{(B)} A \leq h_2 \circ g \text{ a.e.}]\text{"}$$

A formal proof of this remark uses techniques we have not yet developed, namely a method of proof we will call the "domain-of-convergence method." In Chapters 4 through 6 we will discover ourselves repeatedly using this type of method to prove theorems. A restricted form of the idea of domain-of-convergence arguments may be stated in the form of a lemma, a relativization of the combining lemma [HH]. The statement of a lemma sufficiently general to imply all the later results is necessarily cumbersome; we will therefore present it in something less than its full generality.

In the form of lemma 2.9 below, the relativized combining lemma implies some of the later results (corollary 2.9.1, lemma 4.7, lemma 6.2.1). Several others (theorems 4.8, 6.3) will use essentially similar methods.

The lemma relates the complexity of a computation to the complexity of its subcomputations. As in [Ro1, §5.6], we let D_k represent the finite set with canonical index k.

Lemma 2.9: (combining lemma)

Assume we are given a relative complexity measure $\Phi^{(\ )}$ and a function $c \in R_{m+2}$ such that:

$$(\forall i_1,\ldots,i_m,j_1,j_2,x,A)$$
$$[(\varphi_{i_1}{}^{(A)}(x)\downarrow \wedge \ldots \wedge \varphi_{i_m}{}^{(A)}(x)\downarrow \wedge \varphi_{j_1}{}^{(D_{j_2})}(x)\downarrow)$$
$$\Rightarrow \quad (\varphi_{c(i_1,\ldots,i_m,j_1,j_2)}{}^{(A)}(x)\downarrow)].$$

Then there exists $g \in R_2$ such that for all A,

$$g(x,\max\{\Phi_{i_1}{}^{(A)}(x),\ldots,\Phi_{i_m}{}^{(A)}(x),\Phi_{j_1}{}^{(D_{j_2})}(x)\})$$
$$\geq \Phi_{c(i_1,\ldots,i_m,j_1,j_2)}{}^{(A)}(x) \quad \text{a.e. } (x).$$

Proof:

Note: This proof is still valid if j and k are eliminated, or it c is also a function of additional parameters which don't affect the convergence implication.

Define $g(x,y) = \max_{i_k \leq x \text{ for } k \leq m,\ j_1 \leq x,\ j_2 \leq x} g'(x,y,i_1,\ldots,i_m,j_1,j_2)$,

where $g'(x,y,i_1,\ldots,i_m,j_1,j_2) = \max_{A \subseteq N} g''(x,y,i_1,\ldots,i_m,j_1,j_2,A)$, and

$$g''(x,y,i_1,\ldots,i_m,j_1,j_2,A) = \begin{cases} \Phi_{c(i_1,\ldots,i_m,j_1,j_2)}^{(A)}(x) & \text{if } (\forall k \leq m)\ [\Phi_{i_k}^{(A)} \leq y], \text{ and } \Phi_{j_1}^{(D_{j_2})}(x) \leq y \\ 0 & \text{otherwise.} \end{cases}$$

It is easily seen that g'' is total recursive in m + 4 integer variables and 1 set variable. Therefore, by lemma 2.5.1, $g' \in R_{m+4}$, so that $g \in R_2$.

To see that g has the desired properties, we note that if $x \geq \max(i_1,\ldots,i_m,j_1,j_2)$, then:

$$g(x,\max\{\Phi_{i_1}^{(A)}(x),\ldots,\Phi_{i_m}^{(A)}(x),\Phi_{j_1}^{(D_{j_2})}(x)\}) \geq$$

$$g''(x,\max\{\Phi_{i_1}^{(A)}(x),\ldots,\Phi_{i_m}^{(A)}(x),\Phi_{j_1}^{(D_{j_2})}(x)\},i_1,\ldots,i_m,j_1,j_2,A)$$

$$= \Phi_{c(i_1,\ldots,i_m,j_1,j_2)}^{(A)}(x), \quad \text{as required.}$$

QED

As a simple example of the use of lemma 2.9, we give the following corollary. The result, a relativization of the compression theorem, is closely related to lemma 2.7. Here, however, we fix the oracle set B in advance and work with B-recursive functions $\varphi_i^{(B)}$, whereas in lemma 2.7 we work with <u>any</u> total function g and find a set which makes g honest.

Honesty is relevant for this corollary as well. We begin with

any B-recursive function $\varphi_i^{(B)}$, but we only obtain a set A with complexity approximately equal to $\Phi_i^{(B)}$ (a B-honest function) rather than $\varphi_i^{(B)}$.

Corollary 2.9.1: Assume we are given a relative complexity measure $\Phi^{(\ )}$. Then $(\exists h \in R_2)(\forall B)(\forall i, \varphi_i^{(B)} \text{ total})(\exists A)$ such that:

$$\text{Comp}^{(B)}A > \varphi_i^{(B)} \text{ i.o.}$$

$$\text{and} \quad \text{Comp}^{(B)}A \leq h \circ \Phi_i^{(B)} \text{ a.e.}$$

Proof: We define a relative algorithm $\gamma^{(\ )}$ as follows:

For all i, x, B,

$$\gamma^{(B)}(\langle i,x\rangle) = \begin{cases} 1 \dot{-} \varphi_{\pi_1(x)}^{(B)}(x) & \text{if } \varphi_i^{(B)}(x)\downarrow \text{ and} \\ & \Phi_{\pi_1(x)}^{(B)}(x) \leq \varphi_i^{(B)}(x), \\ 0 & \text{if } \varphi_i^{(B)}(x)\downarrow \text{ and} \\ & \Phi_{\pi_1(x)}^{(B)}(x) > \varphi_i^{(B)}(x), \\ \infty & \text{if } \varphi_i^{(B)}(x)\uparrow. \end{cases}$$

By the relativized s-m-n theorem, there exists $c \in R_1$ such that

$$\gamma^{(B)}(\langle i,x\rangle) = \varphi_{c(i)}^{(B)}(x).$$

Now it is clear that $(\forall i,x,B)[\varphi_i^{(B)}(x)\downarrow \Rightarrow \varphi_{c(i)}^{(B)}(x)\downarrow]$.

We may now apply lemma 2.9 and assert that:

(*) $(\exists h \in R_2)(\forall i,B)[h(x,\Phi_i^{(B)}(x)) \geq \Phi_{c(i)}^{(B)}(x) \text{ a.e.}]$.

We now fix i and B as in the hypotheses, and let $C_A = \varphi_{c(i)}^{(B)}$. This is possible since the hypotheses imply that $\varphi_{c(i)}^{(B)}$ is 0-1 valued and total.

By the diagonal construction defining it, C_A satisfies the first conclusion; the second conclusion follows from (*).

QED

Many of the interesting partial relativizations of the speed-up theorem [B1] may be expressed in terms of "helping"; we discuss in these terms the amount by which possession of an oracle speeds up the computation of a function. This type of question forms the subject matter of Chapters 4, 5 and 6.

A relativization of the union theorem [McC] will be given in Chapter 3, together with some interesting consequences.

3. Complexity-Determined Reducibilities

Just as we study complexity classes within non-relativized complexity theory [McC] [McCMe], we may consider resource-bounded relative computation. A fixed resource bound defines a kind of "reducibility" as follows:

Definition 3.1: For any relative complexity measure $\Phi^{(\ )}$, any sets A and B, and any total function f of one variable,

"$A \leq_f B\ (\Phi^{(\ )})$" means $\mathrm{Comp}^{(B)}A \leq f$ a.e., where complexity is measured in $\Phi^{(\ )}$.

More generally, if $\mathcal{C}$ is any class of total functions of one variable,

"$A \leq_{\mathcal{C}} B\ (\Phi^{(\ )})$" means $(\exists f \in \mathcal{C})[A \leq_f B\ (\Phi^{(\ )})]$

We read this notation as "A is f-reducible to B" and "A is $\mathcal{C}$-reducible to B," respectively. When no confusion is likely, we omit mention of the measure we are using, and write simply "$A \leq_f B$" and "$A \leq_{\mathcal{C}} B$."

Several commonly-studied reducibilities usually defined via "natural" (i.e. non-complexity-theoretic) restrictions on the method of computation may be expressed as $\mathcal{C}$-reducibilities for appropriate choices of the class $\mathcal{C}$, and thus may be regarded as complexity-determined. In particular, truth-table reducibility [Rol] and the relation "primitive recursive in" are complexity-determined reducibilities, while many-one and one-one reducibilities are not.

We first consider primitive recursive reducibility. We write "$A \leq_p B$" to indicate that A is primitive recursive in B, and "$f \leq_p B$"

to indicate that f is primitive recursive in B [K].

Theorem 3.2: Let $\mathcal{C}$ = {primitive recursive functions of one variable}. Then $(\forall A,B)[A \leq_p B \Leftrightarrow A \leq_{\mathcal{C}} B\ (T^{(\ )})]$.

Proof: We use the type of T-predicate used by Davis [D], modified slightly for our Turing machine model.

As in [D], we see that:

$$(\forall B)[\lambda z,x,y[T^B(z,x,y)] \text{ is primitive recursive in } B].$$

An examination of the encoding used in the T-predicate shows that there exists f, a primitive recursive function of three variables, such that:

$$(\forall B,w,x,y,z)[(T_z^{(B)}(x) \leq x) \Rightarrow (\exists y \leq f(z,x,w))[T^B(z,x,y)]].$$

(That is, some code number for the computation is effectively bounded by a function of the number of steps in the computation, the input and the index of the machine.)

We now define, for every set B, a function g_B of three variables as follows: (Notation is from [K].)

$$g_B(z,x,w) = \begin{cases} U(\mu y \leq f(z,x,w)[T^B(z,x,y)]) & \text{if y exists} \\ 0 & \text{otherwise} \end{cases}$$

(Intuitively, $g_B(z,x,w)$ represents the output of the computation $\varphi_z^{(B)}(x)$, provided $T_z^{(B)}(x) \leq w$.)

g_B is obviously primitive recursive in B, for any set B.

Now assume we have sets A, B with $A \leq_{\mathcal{C}} B$. This implies:

$(\exists i, \exists h$ a primitive recursive function of one variable)

$$\lceil (C_A = \varphi_i^{(B)}) \wedge (\forall x)[T_i^{(B)}(x) \leq h(x)]\rceil.$$

Then the definition of g_B shows that $C_A = \lambda x \lceil g_B(i,x,h(x))\rceil$, and the function on the right-hand side of this equation is primitive recursive in B.

Thus, $A \leq_C B \Rightarrow A \leq_p B$.

The converse is proved using the following lemma:

Lemma 3.2.1: $(\forall B)(\forall f)\lceil (f \leq_p B) \Rightarrow (\exists p$, a primitive recursive function) $\lceil p \geq f\rceil\rceil$.

(That is, any function primitive recursive in any set is no larger than some primitive recursive function.)

Proof of lemma 3.2.1: We carry out a straightforward proof by induction on the definition of the class of functions primitive recursive in B.

In particular, $C_B \leq \lambda x \lceil 1\rceil$, which is primitive recursive. The other base functions for the induction are themselves primitive recursive.

The two induction steps (composition and recursion) follow without difficulty if we note that:

$(\forall f$, primitive recursive$)(\exists f'$, primitive recursive)

$\lceil (f' \geq f) \wedge (f'$ is monotone increasing in each of its variables)$\rceil$

For example, we verify the recursion step:

Assume that h is a function of k + 1 variables with $h \leq_p B$.

Assume that g is a function of k - 1 variables with $g \leq_p B$.

Assume $h \leq p_h$ and $g \leq p_g$, where p_h and p_g are each primitive recursive and monotone increasing in each variable.

Assume a function f is defined by primitive recursion from g and h: (We write "$\bar{x}$" to indicate "$x_1,\ldots,x_k$.")

$$f(0,\bar{x}) = g(\bar{x})$$
$$(\forall y) \quad f(y+1,\bar{x}) = h(y,f(y,\bar{x}),\bar{x})$$

Now define p_f as follows:

$$p_f(0,\bar{x}) = p_g(\bar{x})$$
$$(\forall y) \quad p_f(y+1,\bar{x}) = p_h(y,p_f(y,\bar{x}),\bar{x})$$

It is easy to verify that p_f is primitive recursive and $f \leq p_f$, as required.

Proof of theorem 3.2, continued:

We again use induction on the class of functions primitive recursive in B.

An oracle Turing machine with a B-oracle can obviously compute C_B rapidly. In particular,
$(\exists i)(\exists p$, a primitive recursive function$)[(\varphi_i^{(B)} = C_B) \wedge (T_i^{(B)} \leq p)]$.

The other base functions are primitive recursive, and so are computable in primitive recursive time. [C]

The two induction steps are straightforward; we verify the primitive recursion step, leaving the composition step to the reader:

Assume we have f, g and h as in the proof of lemma 3.2.1. As inductive hypothesis, we assume:

$$(\exists i)[(\varphi_i^{(B)} = g) \wedge (T_i^{(B)} \leq p_g)]$$

and

$$(\exists j)[(\varphi_j^{(B)} = h) \wedge (T_j^{(B)} \leq p_h)]$$

where p_g and p_h are primitive recursive and monotone increasing in each variable.

By lemma 3.2.1, there exists f', a primitive recursive function such that $f \leq f'$.

We define a primitive recursive function p_f as follows:

$$p_f(0,\bar{x}) = p_g(\bar{x})$$

$$(\forall y) \quad p_f(y+1,\bar{x}) = p_f(y,\bar{x}) + p_h(y,f'(y,\bar{x}),\bar{x})$$

We claim that the primitive recursive function $p_f + x_1 + \ldots + x_k + y$ is an upper bound for the time required to compute f. If further details on this induction step are desired, see [C], [A1], [RD], [RRW] or [MeRD].

Corollary 3.2.2: Theorem 3.2 is true for $S^{(\ )}$ in place of $T^{(\ )}$.

Proof: $S^{(\ )}$ and $T^{(\ )}$ are related, in the sense of theorem 2.5, by a primitive recursive function, as we can show by an argument similar to the looping argument in the discussion of $S^{(\ )}$ in Chapter 2.

Remark: Theorem 3.2 is false for some pathological measures.

We now consider truth-table reducibility [Ro1]. A result of McLaughlin [McL] combined with theorem 2.5, gives the following complexity-determination result for truth-table reducibility:

Proposition 3.3: Fix any relative complexity measure.

Let $\mathcal{C} = R_1$. Then:

$$(\forall A,B)\ [A \leq_{tt} B \ \Leftrightarrow\ A \leq_{\mathcal{C}} B].$$

On the other hand, many-one and one-one reducibilities are not determined by a complexity restriction, in any relative complexity measure. The reason is that there are pairs of sets computable from each other in a very small amount of resource but which are not many-one reducible to each other (for example, any nonrecursive recursively enumerable set and its complement). Thus, for natural measures, it is obvious that many-one and one-one reducibilities are not complexity-determined. For general measures, however, a little work is required:

Proposition 3.4: Fix any relative complexity measure. Let $\mathcal{C}$ be any class of total functions of one variable. Then it cannot be the case that:

$$(\forall A,B \in R_1)\ [A \leq_m B \ \Leftrightarrow\ A \leq_{\mathcal{C}} B].$$

Proof: Assume the contrary: let $\mathcal{C}$ determine many-one reducibility for measure $\Phi^{(\ )}$.

By consideration of $T^{(\ )}$ and remark 2.5.2, we see that:

$$(\exists s \in R_1)(\exists i)\ [(\varphi_i^{(\overline{K})} = C_K) \ \wedge\ (\Phi_i^{(\overline{K})} \leq s)].$$

But $K \not\leq_m \overline{K}$, so that $(\forall c \in \mathcal{C})\ [c(x) < s(x)\ \text{i.o.}]$.

To obtain a contradiction, it suffices to show that:

$$(\exists A,B)\ [(A \leq_m B) \ \wedge\ (\text{Comp}^{(B)}A > s\ \text{a.e.})]$$

But this follows from the following lemma:

Lemma 3.4.1: Fix any relative complexity measure. Then:

$(\forall s \in R_1)(\forall B \text{ recursive})(\exists A \text{ recursive})$

$$[\text{Comp}^{(B)}A > s \text{ a.e.}].$$

Proof of lemma 3.4.1: We note that $\begin{cases} \hat{\varphi}_i \stackrel{\text{def}}{=} \varphi_i^{(B)} \\ \hat{\Phi}_i \stackrel{\text{def}}{=} \Phi_i^{(B)} \end{cases}$ satisfy the requirements for an acceptable Gödel numbering and a Blum complexity measure.

Then the existence of arbitrarily complex (a.e.) recursive sets in any Blum measure [B1] gives us a set A such that:

$$\hat{\varphi}_i = C_A \Rightarrow \hat{\Phi}_i > s \text{ a.e.}$$

which translates into the desired result by the definitions of $\hat{\varphi}_i$ and $\hat{\Phi}_i$.

Proof of proposition 3.4, continued:

Now select any infinite, coinfinite recursive set B, and use lemma 3.4.1 to obtain an appropriate set A. We can easily obtain A infinite and coinfinite, and so $A \equiv_1 B$. But $\text{Comp}^{(B)}A > s$ a.e., giving the desired result.

QED

Corollary 3.4.2: Proposition 3.4 is true for one-one reducibility in place of many-one reducibility.

Proof: Implicit in the proof above.

Open Question: Is it true that:

$(\forall s \in R_1)(\forall B \text{ infinite and coinfinite})(\exists A \blacksquare_1 B)$

$$[\text{Comp}^{(B)}A > s \text{ a.e.}] \ ?$$

Having shown that primitive recursive and truth-table reducibilities are complexity-determined, we ask if it is possible to express them even more succinctly; for instance, is it possible to characterize each by a single resource bound function rather than a class of functions?

This question immediately suggests that we would like an analog to the union theorem [McC], and so we prove the following:

Theorem 3.5: (relativized union theorem)

Assume we have a sequence of total functions $\{t_i\}$, with:

$$(\forall i,n)[t_{i+1}(n) \geq t_i(n)].$$

Let T be a set such that $\lambda i,n[t_i(n)]$ is recursive in T.

Also assume that we have a sequence $\{B_i\}$ of sets, and a set B such that: $\lambda i,n[C_{B_i}(n)]$ is recursive in B.

Then there exists a function $f \in R_1^{(B \text{ join } T)}$ such that:

$$(\forall i,j)[(\Phi_i^{(B_j)} \leq f \text{ a.e.}) \Leftrightarrow (\exists k)(\Phi_i^{(B_j)} \leq t_k \text{ a.e.})].$$

(This means that for any B_j, the class of functions computable with oracle B_j within measure f is exactly the union of the classes of functions computable with oracle B_j within measure t_k, the union being taken over all t_k.)

Proof: The construction of f is carried out in stages, with f(n) being defined at stage n.

We define an auxiliary function g(i,j), whose values may be changed at successive stages. The significance of g(i,j) is as follows:

We "guess" that $\Phi_i^{(B_j)}(x) \leq t_{g(i,j)}(x)$ a.e.

Stage n: (Define f(n).)

For all (i,j) such that $i + j = n$, define $g(i,j) = n$.

Let $E = \{(i,j) \mid i + j \leq n \text{ and } \Phi_i^{(B_j)}(n) > t_{g(i,j)}(n)\}$.

Define:
$$f(n) = \begin{cases} t_n(n) & \text{if } E = \emptyset \\ \min_{(i,j)\in E} t_{g(i,j)}(n) & \text{otherwise.} \end{cases}$$

For all $(i,j) \in E$, redefine $g(i,j) = n$.

Go on to stage $n + 1$.

END OF CONSTRUCTION

Verification:

Assume we have i, j, k and we know that $\Phi_i^{(B_j)} \leq t_k$ a.e. We would like to conclude that $\Phi_i^{(B_j)} \leq f$ a.e.; it therefore suffices to show that $(\forall k)[t_k \leq f \text{ a.e.}]$.

If not, then for some k we have $f(n) < t_k(n)$ on infinitely many arguments $n > k$.

At stage k, there can only be finitely many pairs (i,j) with $g(i,j) < k$. We let F be this finite set of pairs. After stage k, no pair (i,j) ever has g(i,j) become defined to be less than k. Therefore, if $g(i,j) < k$ at some stage after stage k, we know that $(i,j) \in F$.

Now if $f(n) < t_k(n)$ on infinitely many arguments $n > k$, then for these n, f(n) is defined to equal $t_{g(i,j)}(n)$ for some $(i,j) \in F$ with $g(i,j) < k$. But then at stage n, g(i,j) is redefined to equal n.

Since F is a finite set, this can only occur finitely often before no pairs (i,j) remain with $g(i,j) < k$.

Therefore, we have $(\forall i,j)$

$$[(\exists k)(\Phi_i^{(B_j)} \leq t_k \text{ a.e.}) \Rightarrow (\Phi_i^{(B_j)} \leq f \text{ a.e.})]$$

Conversely, assume we have (i,j) with $(\forall k)[\Phi_i^{(B_j)} > t_k \text{ i.o.}]$. Then each time we define g(i,j), we will subsequently reach a stage n where:

$$\Phi_i^{(B_j)}(n) > t_{g(i,j)}(n).$$

At this stage n, (i,j) will be in set E, so the definition of f will insure that $\Phi_i^{(B_j)}(n) > f(n)$. We will also redefine g(i,j). But it is easy to see that this must happen for infinitely many arguments n, so that:

$$\Phi_i^{(B_j)} > f \text{ i.o.}$$

Thus, we have:

$$(\forall i,j)[(\forall k)(\Phi_i^{(B_j)} > t_k \text{ i.o.}) \Rightarrow (\Phi_i^{(B_j)} > f \text{ i.o.}].$$

It is clear that f is recursive in B join T.

QED

We now apply theorem 3.5 to the cases of truth-table reducibility and primitive recursive reducibility.

Corollary 3.5.1: Consider any countable collection of sets $\{B_i\}$ with B as in theorem 3.5. There exists $f \in R_1^{(B \text{ join } K)}$ such that:

$$(\forall i,A)[A \leq_{tt} B_i \Leftrightarrow A \leq_f B_i].$$

Proof: We define a sequence $\{t_i\}$ as follows:

Let $t_i(x) = \begin{cases} \max_{j \le i} \{\varphi_j(x) \mid \varphi_j(y)\downarrow \text{ for all } y \le x\} & \text{if this set is nonempty} \\ 0 & \text{otherwise} \end{cases}$

These t_i have the properties required for theorem 3.5, with $T = K$.

Also, $(\forall r \in R_1)(\exists j)[r \le t_j \text{ a.e.}]$

$(\forall j)(\exists r \in R_1)[t_j \le r \text{ a.e.}]$

Thus, by proposition 3.3, if $\mathcal{C} = \{t_i\}$, then

$$(\forall A,B)[A \le_{tt} B \Leftrightarrow A \le_{\mathcal{C}} B].$$

Application of theorem 3.5 now gives the desired result.

QED

Corollary 3.5.2: Assume we are working with $S^{(\ )}$ or $T^{(\ )}$. Consider any countable collection of sets $\{B_i\}$, with B as in theorem 3.5. There exists $f \in R_1^{(B)}$ such that:

$$(\forall i,A)[A \le_p B_i \Leftrightarrow A \le_f B_i].$$

Proof: Let $\{p_i\}$ be an enumeration of the primitive recursive functions such that $\lambda i,x[p_i(x)]$ is recursive. Then define:

$$t_i(x) = \max_{j \le i} p_j(x).$$

$\{t_i\}$ satisfies the required properties for theorem 3.5, with $T = \emptyset$.

Clearly, $(\forall i)[p_i \le t_i \text{ a.e.}]$, and

$(\forall i)(\exists j)[t_i \le p_j \text{ a.e.}]$.

Applying theorems 3.2 and 3.5 gives the desired result.

QED

Thus, we see that for any countable collection of oracle sets (e.g. recursive sets, arithmetical sets), truth-table reducibility is determined by a single resource bound function on any measure, and primitive recursive reducibility is determined by a single resource bound function on measures $T^{(\ )}$ and $S^{(\ )}$.

The next question we consider is whether any single function can determine either of these two reducibilities on all pairs of sets. This we show to be impossible; thus, the countability hypothesis in corollaries 3.5.1 and 3.5.2 cannot be eliminated.

Theorem 3.6: There is no function f of one variable such that: $(\forall A,B \text{ recursive in } f)[\, A \leq_{tt} B \Leftrightarrow A \leq_{f} B]$.

Proof: Assume such a function f exists.

We claim that $(\forall r \in R_1)[f > r \text{ a.e.}]$.

For if not, then $(\exists r \in R_1)[r \geq f \text{ i.o.}]$.

But then, by Rabin's diagonal method, there exists a recursive set A such that Comp A > r a.e. We have $A \leq_{tt} \emptyset$, since A is recursive, but clearly $\neg(A \leq_{f} \emptyset)$, a contradiction.

Now consider the function h whose existence is asserted in lemma 2.7. We may assume without loss of generality that h is monotone increasing in both variables.

Define a function g as follows:

$$g(x) = \begin{cases} \max\{y \mid h(x,y) \leq f(x)\} & \text{if the set is nonempty} \\ 0 & \text{otherwise} \end{cases}$$

We claim that $(\forall r \in R_1)[g > r \text{ a.e.}]$. This is easily concluded from the facts that $(\forall r \in R_1)[f > r \text{ a.e.}]$ and that h is recursive.

We now apply lemma 2.7 to obtain A and B such that:

$$[(A \leq_{h \circ g} B) \wedge \neg(A \leq_g B)].$$

But $(A \leq_{h \circ g} B)$ implies $(A \leq_f B)$ since $h \circ g \leq f$ a.e.

$\neg(A \leq_g B)$ implies $\neg(A \leq_{tt} B)$, since g is almost everywhere greater than each recursive function.

Thus, f does not determine truth-table reducibility on all pairs of sets.

QED

Theorem 3.7: Assume that we are working with space measure on oracle Turing machines. There is no function f of one variable such that:

$$(\forall A,B)[A \leq_p B \Leftrightarrow A \leq_f B].$$

Proof: The proof is analogous to that of theorem 3.6:

We claim, if such an f exists, that:

(∀r, primitive recursive in one variable)[f > r a.e.],

For if not, then:

(∃r, primitive recursive in one variable)[f ≤ r i.o.]. We may assume without loss of generality that r is monotone nondecreasing. But then, by a Rabin diagonalization argument, there exists a recursive set A such that Comp A > r a.e.

However, by a result of Cobham [C] and an examination of the diagonalization, we see that:

$(\exists i)(\exists s$, primitive recursive of one variable$)[\varphi_i = C_A \wedge S_i \leq s$ a.e.$]$.

We thus have $A \leq_p \emptyset$, but $\neg(A \leq_f \emptyset)$.

If we let $\Phi^{(\ )} = S^{(\ )}$ in lemma 2.7, it is possible to obtain a function h satisfying the conditions of the lemma which is primitive recursive. We construct B and A as in lemma 2.7, and define g as in the proof of theorem 3.6.

As before, we obtain:

$(\forall r$, primitive recursive in one variable$)[g > r$ a.e.$]$.

Thus, as before, $A \leq_f B \ \wedge \ \neg(A \leq_p B)$.

Therefore, f does not determine primitive recursive reducibility on all pairs of sets.

QED

Remark: An analogous proof also holds for $T^{(\ )}$ in place of $S^{(\ )}$.

Open Question: Is theorem 3.7 true for all Blum measures?

Open Question: Examine other natural reducibilities, such as bounded truth-table reducibility, or any of the others mentioned in [J1], to see if any are complexity-determined.

We have seen that some reducibilities with "natural" definitions may be alternatively described by a complexity restriction. Conversely, it is possible to define new reducibilities by a complexity restriction. In the remainder of this chapter, we give an example of such a definition,

and examine some properties of the resulting reducibilities.

Definition 3.8: For any sets A, B, C, we say "A is C-reducible to B" ($A \leq_C B$) provided:

$$A \leq_{\mathcal{C}} B \quad \text{for} \quad \mathcal{C} = R_1^{(C)}.$$

We write "C-reducibility" to indicate $\{(A,B) \mid A \leq_C B\}$.

Thus, any set C determines a new reducibility, namely, the collection of pairs of sets computable from each other in C-recursive measure. The reducibilities are clearly measure-invariant, by theorem 2.5 and remark 2.5.2.

Strictly speaking, anything we call a "reducibility" ought to be reflexive and transitive, properties which do not hold for general classes $\mathcal{C}$. However, our C-reducibilities are reflexive and transitive: reflexivity is clear, for any C. We demonstrate transitivity:

Lemma 3.9: For any sets A, B, C and D,

$$[(A \leq_C B \ \wedge \ B \leq_C D) \ \Rightarrow \ A \leq_C D].$$

Proof: By measure invariance of C-reducibility and closure of $R_1^{(C)}$ under finite modification, we obtain:

$$(\exists i)(\exists c_1 \in R_1^{(C)})[C_A = \varphi_i^{(B)} \ \wedge \ S_i^{(B)} \leq c_1], \text{ and}$$

$$(\exists j)(\exists c_2 \in R_1^{(C)})[C_B = \varphi_j^{(D)} \ \wedge \ S_j^{(D)} \leq c_2].$$

We describe an oracle Turing machine which computes C_A using a D-oracle:

The Turing machine computes C_A according to procedure $\varphi_i^{(B)}$, but the values about which we query the B-oracle get written on a second

track of the worktape instead of the oracle tape. Then, to decide their membership in B, we use $\varphi_j^{(D)}$, with our D-oracle.

How much space is required by this new machine?

For input x, the machine uses $S_i^{(B)}(x)$ to carry out the computation $\varphi_i^{(B)}(x)$. In addition, the largest argument for which we might need to compute C_B is $2^{S_i^{(B)}(x)}$, so we might also require:

$$S_j^{(D)}(0),\ S_j^{(D)}(1),\ldots,S_j^{(D)}(2^{S_i^{(B)}(x)}).$$

Thus, the space needed is bounded above by the maximum of:

$$S_i^{(B)}(x),S_j^{(D)}(0),\ldots,S_j^{(D)}(2^{S_i^{(B)}(x)}),$$

which is bounded above by the maximum of:

$$c_1(x),c_2(0),\ldots,c_2(2^{c_1(x)}).$$

But this maximum is a function in $R_1^{(C)}$.

Thus, $A \leq_C D$.

QED

We remark that results similar to theorems 3.6 and 3.7 may be obtained for these reducibilities as well.

There are simple relationships between these reducibilities and others:

Proposition 3.10: (a) For any set C, $A \leq_{tt} B \Rightarrow A \leq_C B \Rightarrow A \leq_T B$. (b) If C is a recursive set, $A \leq_C B \Leftrightarrow A \leq_{tt} B$.

Proof: Immediate from proposition 3.3.

We would like to have a structural description of C-reducibility as an alternative to the complexity definition. We obtain the following partial result in this direction:

Remark 3.11: Assume $C \leq_C B$. Then $(\forall A)$
$A \leq_C B \Leftrightarrow (\exists f \in R_1^{(C)})[x \in A \Leftrightarrow$ tt-condition f(x) is satisfied by B]
(For notation, see [Ro1]).

(This says that, provided the oracle set has a sufficiently high degree of unsolvability, any reducibility of our type may be described by the ability to construct a truth table for the computation with the help of the appropriate oracle.)

Proof: ($\Rightarrow$) Similar to McLaughlin's proof [McL]. The condition $C \leq_C B$ is not needed for the proof in this direction.

($\Leftarrow$) We assume that $C \leq_C B$, specifically, that :
$C_C = \varphi_i^{(B)}$, $\Phi_i^{(B)} \leq g$ where $g \in R_1^{(C)}$.

We assume also that:

$[x \in A \Leftrightarrow$ tt-condition f(x) is satisfied by B].

We show that C_A is computable from B in C-recursive time. The procedure we will use for computing C_A using a B-oracle is as follows:

"Given input x, we compute f(x). f is recursive in C, so we simulate a machine computing f from C; we use $\varphi_i^{(B)}$ to obtain answers to questions about membership in C.

Once we have the truth-table f(x), we then ask the B-oracle about membership of each argument in the truth table and use the

answers to find the value of $C_A(x)$ from the truth table."

How much time is required by this procedure?

If we assume that $f = \varphi_j^{(C)}$, then we can obtain f(x) in time approximately bounded by:

$$T_j^{(C)}(x) + \sum_{y=0}^{2^{T_i^{(C)}(x)}} g(y),$$

since the largest value y for which we might need to compute $\varphi_i^{(B)}(y)$ is $2^{T_i^{(C)}(x)}$. Clearly, this sum is bounded by a total C-recursive function.

Once we have f(x), it is not difficult to show that the remaining time required to obtain $C_A(x)$ by asking the appropriate questions about membership in B is bounded by a C-recursive function.

Thus, the total time is bounded by a C-recursive function, so $A \leq_C B$.

QED

Each set determines a reducibility. We may obtain a "hierarchy of reducibilities" between truth-table and Turing reducibilities, ordered by a comparability relation. Using a relativization of the compression theorem, we conclude that comparability is exactly determined by size of functions:

Theorem 3.11: Assume we are given two sets, C and D. Then $\ulcorner(\forall A,B)\ulcorner A \leq_C B \Rightarrow A \leq_D B\urcorner\urcorner \Leftrightarrow \ulcorner(\forall f \in R_1^{(C)})(\exists g \in R_1^{(D)})\ulcorner g \geq f \text{ a.e.}\urcorner\urcorner$

Proof: ($\leftarrow$) Obvious.

($\rightarrow$) Assume $(\exists f \in R_1^{(C)})(\forall g \in R_1^{(D)})[g < f \text{ i.o.}]$.

Then by a direct relativization of the compression theorem,

$(\exists A \text{ recursive in } C)(\forall i)[\varphi_i^{(C)} = C_A \Rightarrow \Phi_i^{(C)} > f \text{ a.e.}]$.

It is easy to show that $A \leq_C C$ but $\neg(A \leq_D C)$, a contradiction.

QED

Corollary 3.11.1: For any sets C, D,

$$(C \equiv_T D) \Rightarrow (\text{C-reducibility} = \text{D-reducibility}).$$

In the remainder of this chapter, we ask about the converse of Corollary 3.11.1. That is, if sets determine the same reducibility, need they be Turing equivalent? In certain cases, the answer is yes:

Definition 3.12: A set A is weakly majoreducible if there exists $f \in R_1^{(A)}$ such that:

$$(\forall g)[g \geq f \Rightarrow A \text{ is recursive in } g].$$

This definition is weaker than, although similar to, the definition of "majoreducible" used and studied extensively by Jockusch in [J2].

Theorem 3.13: If sets C and D are weakly majoreducible, then:

$$(\text{C-reducibility} = \text{D-reducibility}) \Rightarrow (C \equiv_T D).$$

Proof: If C-reducibility = D-reducibility, then by theorem 3.11 and closure of $R_1^{(D)}$ under finite modification, we have:

$$(\forall f \in R_1^{(C)})(\exists g \in R_1^{(D)})[g \geq f].$$

By weak majoreducibility of C, C is recursive in g for the appropriate choice of f.

Therefore, $C \leq_T D$.

Symmetrically, we have $D \leq_T C$.

QED

Corollary 3.13.1: If sets C and D are recursively enumerable, then (C-reducibility = D-reducibility) $\Leftrightarrow$ $(C \equiv_T D)$.

Proof: It follows immediately from work of Jockusch in [J2] that all recursively enumerable sets are majoreducible (his definition) and hence weakly majoreducible. The reason is as follows:

Suppose C is a recursively enumerable set. If C is finite, $f \equiv 0$ satisfies definition 3.12.

Otherwise, let $\{c_i\}$ be an effective enumeration of C without repetitions.

Define $f(n) = \mu z[(\forall y)(y > z \Rightarrow c_y > n)]$.

It is easy to show that $f \in R_1^{(C)}$.

If $g \geq f$, we may compute $C_C(n)$ by listing C for g(n) steps to see if n turns up. Thus, C is recursive in g, so is majoreducible.

QED

Note: Examination of the proofs above, combined with the Friedberg-Muchnik theorem [Rol, §10.2] shows that there exist pairs of recursively enumerable sets C and D determining incomparable reducibilities; that is,

$$(\exists A,B)[(A \leq_C B) \wedge \neg(A \leq_D B)]$$

and

$$(\exists A,B)[(A \leq_D B) \wedge \neg(A \leq_C B)].$$

We have thus shown that for a large collection of sets, if any pair

determines the same reducibility, the two sets must be Turing equivalent. However, this is not true in general. In fact:

Proposition 3.14: Given any set C, there exist two sets A and B such that $A \mid_T B$ and A-reducibility = B-reducibility = C-reducibility.

We omit a detailed proof because it is quite long and not very different from other proofs in the literature. There is a modified version [Ma1] of Spector's splitting-tree construction of minimal sets [Ro1, §13.5] which produces a nonrecursive set A which is "small" rather than minimal: that is,

$$(\forall f \in R_1^{(A)})(\exists g \in R_1)[g \geq f].$$

In outline, in the proof of proposition 3.14 we simultaneously construct two "small" sets, A and B, by modified splitting-tree constructions, with two added changes:

(1) We encode C into both sets at the beginning of the construction.

(2) We alternate the splitting-tree construction with a straightforward diagonalization making A and B Turing incomparable.

The resulting sets A and B are such that:

$(\forall f \in R_1^{(C)})(\exists g \in R_1^{(A)})$ $[g \geq f$ a.e.$]$ since $C \leq_T A$,

and

$(\forall f \in R_1^{(A)})(\exists g \in R_1^{(C)})$ $[g \geq f$ a.e.$]$ by the construction,

and similarly for B.

Thus, by theorem 3.11, A-reducibility = B-reducibility = C-reducibility.

QED

Open Question: Given any nonrecursive set A, is it always possible to find a set B such that $A \mid_T B$ but A-reducibility = B-reducibility?

Open Question: What are necessary and sufficient conditions on sets A and B for A-reducibility to equal B-reducibility?

Although pairs of sets can have the same reducibility and still be Turing-incomparable, there do exist limits on what Turing-reducibility relationships sets can have and still determine the same reducibility. For example:

Proposition 3.15: If C-reducibility = D-reducibility, then we cannot have $C' \leq_T D$.

Proof: Assume $C' \leq_T D$.

Let $g(n) = \max \{\varphi_i^{(C)}(n) \mid 0 \leq i \leq n \wedge \varphi_i^{(C)}(n)\downarrow\}$.

$g \in R_1^{(C')}$, so $g \in R_1^{(D)}$.

But clearly $(\forall f \in R_1^{(C)})[g \succ f \text{ i.o.}]$.

So by theorem 3.11, C-reducibility $\neq$ D-reducibility.

QED

Open Question: In [J1], Jockusch develops the properties of various types of truth-table reducibilities, e.g. containment properties of degrees. Explore the answers to these questions for C-reducibilities for various sets C. For example, does C-reducibility have any properties significantly different from truth-table reducibility?

4. Helping

Intuitively, we have the idea that some sets B help to compute some functions f. That is, when we use B as an oracle, we can reduce the complexity of f below what it could have been without the oracle B.

In this chapter, we try to formalize this idea. We use the word "helping" in informal discussions only, and give precise meanings to several interpretations. We also give several basic results about the existence of sets which help or don't help the computation of certain functions.

Definition 4.1: Assume B is a set, $f \in R_1$ and h is a total function of two variables.

"B h-improves f i.o." means:

$$(\exists i)(\forall j)[\varphi_i^{(B)} = f \ \wedge \ \varphi_j = f \ \Rightarrow \ h(x,\Phi_i^{(B)}(x)) < \Phi_j(x) \text{ i.o.}]].$$

"B h-improves f a.e." means:

$$(\exists i)(\forall j)[\varphi_i^{(B)} = f \ \wedge \ \varphi_j = f \ \Rightarrow \ h(x,\Phi_i^{(B)}(x)) < \Phi_j(x) \text{ a.e.}]].$$

We remark that these definitions do not provide us with notions of helping that are transitive or symmetric. Appropriate counterexamples will be found as corollaries near the end of this chapter.

An alternative way of measuring the amount of help given by a set B to a function f is to ask which lower bounds on the complexity of f are maintained after introduction of the B-oracle. To speak about this kind of "helping" we use the definitions of $\text{Comp}^{(B)}f$ and Comp f, etc., introduced as definition 2.6.

To place these definitions in some perspective, it is helpful to note a relationship between "$A \leq_p B$" and "B h-improves A":

Assume A is not primitive recursive. Then for any primitive recursive function p of one variable, we know that Comp A > p i.o. [C]

But if $A \leq_p B$, then for some primitive recursive function q of one variable, $\text{Comp}^{(B)}A \leq q$ a.e. Because the primitive recursive functions are closed under composition, $h \circ q$ is primitive recursive and therefore Comp A > $h \circ q$ i.o.

But since $\text{Comp}^{(B)}A \leq q$ a.e., we know that B h-improves A.

We note that the amount of help a recursive oracle B is able to give the computation of a function is restricted by the complexity of B. This is because any program using B as an oracle may be converted to one not using the B-oracle, by directly computing the answers to the oracle queries. The complexity of the new program is bounded as follows:

Theorem 4.2: There exist $g_1, g_2 \in R_3$ with the following property:

For all B, i, j, if $C_B = \varphi_i$, then there exists k such that:

$$\varphi_k = \varphi_j^{(B)}$$

and $$\Phi_k(x) \leq g_1(x, \Phi_j^{(B)}(x), \max_{0 \leq y \leq g_2(j,x,\Phi_j^{(B)}(x))} \Phi_i(y)) \quad \text{a.e. } (x).$$

Proof: Although this proof does not exactly fit the statement of the combining lemma, we note the essential similarity of the proofs; we call this type of argument a "domain-of-convergence" argument.

We use the following general lemma:

Lemma 4.2.1: Fix any acceptable enumeration of relative algorithms $\{\varphi_i^{(\ )}\}$ and any relative complexity measure $\Phi^{(\ )}$. Then:

$$(\exists g \in R_3)(\forall i,x,y)(\forall A,B)$$
$$[(A \cap \{0,\ldots,g(i,x,y)\} = B \cap \{0,\ldots,g(i,x,y)\}) \Rightarrow$$
$$((\Phi_i^{(A)}(x) \leq y \Leftrightarrow \Phi_i^{(B)}(x) \leq y) \wedge (\Phi_i^{(A)}(x) \leq y \Rightarrow \varphi_i^{(A)}(x) = \varphi_i^{(B)}(x)))].$$

Proof of lemma 4.2.1: Let $\{T_i^{(\ )}\}$ be the standard enumeration of oracle Turing machines.

Axiom (2) for relative complexity measures will allow us to conclude the existence of a relative algorithm $\alpha^{(\ )}$ such that:

$$(\forall i,x,y,X) \quad \alpha^{(X)}(\langle i,x,y\rangle) = \begin{cases} 1 & \text{if } \Phi_i^{(X)}(x) \leq y, \\ 0 & \text{if not.} \end{cases}$$

Therefore, by lemma 2.2, there exists an oracle Turing machine $T_j^{(\ )}$ such that:

$$T_j^{(X)}(\langle i,x,y\rangle) = \begin{cases} 1 & \text{if } \Phi_i^{(X)}(x) \leq y, \\ 0 & \text{if not.} \end{cases}$$

Now fix i, x and y. Let f be the recursive isomorphism (lemma 2.2) between the two Gödel numberings.

Define $g(i,x,y) = \max_{X \subseteq N} g'(i,x,y,X)$, where $g'(i,x,y,X)$ is defined as follows:

$g'(i,x,y,X) =$

$$\begin{cases} \text{the largest number whose membership in X is questioned in oracle Turing machine computation } T_j^{(X)}(\langle i,x,y\rangle) & \text{if } T_j^{(X)}(\langle i,x,y\rangle) = 0, \\ \text{the largest number whose membership in X is questioned in either computation } T_j^{(X)}(\langle i,x,y\rangle) \text{ or computation } T_{f(i)}^{(X)}(x) & \text{if } T_j^{(X)}(\langle i,x,y\rangle) = 1. \end{cases}$$

Lemma 2.5.1 shows that $g \in R_3$.

Now assume that $A \cap \{0,\ldots,g(i,x,y)\} = B \cap \{0,\ldots,g(i,x,y)\}$.

Then by definition of g', we have that:

$$T_j^{(A)}(\langle i,x,y\rangle) = T_j^{(B)}(\langle i,x,y\rangle)$$

and $$(T_j^{(A)}(\langle i,x,y\rangle) = 1) \Rightarrow (T_{f(i)}^{(A)}(x) = T_{f(i)}^{(B)}(x)).$$

But by definition of T_j and f, this implies the lemma.

<u>Proof of theorem 4.2, continued</u>: We let g_2 = the function g from lemma 4.2.1.

The s-m-n theorem allows us to define a partial recursive function $\varphi_{\alpha(a,b)}$ as follows:

$$\varphi_{\alpha(a,b)}(x) = \begin{cases} y & \text{if } (\exists A)\ \varphi_a^{(A)}(x) = y, \text{ and } (\forall w \le g_2(a,x,\Phi_a^{(A)}(x))) \\ & (w \in A \Leftrightarrow \varphi_b(w) = 1) \text{ and} \\ & (w \notin A \Leftrightarrow \varphi_b(w) = 0) \\ \infty & \text{otherwise} \end{cases}$$

By the definition of g in lemma 4.2.1, the function $\varphi_{\alpha(a,b)}$ must be well-defined. It is easy to see that it is partial recursive.

Intuitively, for oracle Turing machines, $\varphi_{\alpha(a,b)}$ is simply the function computed by $\varphi_a^{(\ )}$, where we use the partial recursive function φ_b in place of an oracle.

We now define $g_1(x,y,z) = \max_{a,b\leq x, A\subseteq N} g'(x,y,z,a,b,A)$, where:

$$g'(x,y,z,a,b,A) = \begin{cases} \Phi_{\alpha(a,b)}(x) & \text{if } \Phi_a^{(A)}(x) = y, \text{ and} \\ & \max_{0\leq w\leq g_2(a,x,y)} \{\Phi_b(w)\} = z, \text{ and} \\ & (\forall w \leq g_2(a,x,y)) \\ & (w \in A \Leftrightarrow \varphi_b(w) = 1) \text{ and} \\ & (w \notin A \Leftrightarrow \varphi_b(w) = 0), \\ 0 & \text{otherwise.} \end{cases}$$

By the definition of $\varphi_{\alpha(a,b)}$, we see that the listed conditions are sufficient to insure the convergence of $\Phi_{\alpha(a,b)}(x)$, and so by lemma 2.5.1, $g_1 \in R_3$.

We now fix a = j, b = i.

We claim $\alpha(j,i) \stackrel{\text{def}}{=} k$ has the required properties:

If $x \geq \max(i,j)$, then:

$$g_1(x,\Phi_j^{(B)}(x),\max_{0\leq y\leq g_2(j,x,\Phi_j^{(B)}(x))} \Phi_i(y)) \geq$$

$$g'(x,\Phi_j^{(B)}(x),\max_{0\leq y\leq g_2(j,x,\Phi_j^{(B)}(x))} \Phi_i(y),j,i,B)$$

$\geq \Phi_{\alpha(a,b)}(x)$, since all the listed conditions in the definition of g' are satisfied.

QED

Open Question: Can we obtain a version of theorem 4.2 for oracle sets B which are nonrecursive? That is, can we find any way to bound the amount of help a nonrecursive set B can give to the computation of a function (for example, relative to B's Turing-reducibility properties, or to B's complexity relative to some set)?

We next show that for any sufficiently complex recursive set A, there exist arbitrarily complex recursive sets B that do help the computation of C_A; in fact, which reduce it to triviality.

We may further specify that the set B be "compressed" (i.e. B's complexity is very closely determined, to within a fixed amount h depending on the measure only).

Theorem 4.3: Let $\Phi^{(\ )}$ be any relative complexity measure. There is a function $h \in R_2$ with the following property:

Let t be any total, monotone nondecreasing running time.

Let A be any recursive set such that Comp $A \leq t$ a.e.

Then there exists a recursive set B with:

$$\text{Comp } B > t \text{ a.e.}$$

$$\text{Comp } B \leq h \circ t \text{ a.e.}$$

and

$$A \leq_p B.$$

(Note: As mentioned in the remark following lemma 2.7, this is an example of a theorem which uses an honesty hypothesis.)

Proof: The proof is a domain-of-convergence argument.

We carry out the construction in stages, using a Rabin diagonal construction with one modification: we introduce new programs into the

construction slowly, so most arguments are not needed for the diagonalization. We use the remaining arguments to encode A in a simple way. The idea is similar to that used by Paterson in theorem 4.6.

We define a function f as follows:

$f(0) = 0, \qquad f(n) = \lfloor \sqrt{n} \rfloor - 1$ for all $n \geq 1$.

By the s-m-n theorem, we can define a partial recursive function $\varphi_{\alpha(a,b)}$ (where $\alpha \in R_2$) according to the following construction in stages:

Stage n: (Define $\varphi_{\alpha(a,b)}(n)$)

Find the smallest uncancelled $i \leq f(n)$ such that $\Phi_i(n) \leq \Phi_b(n)$. (We diverge if $\Phi_b(n)\uparrow$.)

If no such i exists, define $\varphi_{\alpha(a,b)}(n) = \varphi_a(f(n))$.

If i exists, define $\varphi_{\alpha(a,b)}(n) = 1 \dot{-} \varphi_i(n)$ and cancel i.

Go on to stage n + 1.

END OF CONSTRUCTION

Now assume we have A, t as in the hypotheses. If we choose a*, b* with $\Phi_{b*} = t$ and $\varphi_{a*} = C_A$ and $\Phi_{a*} \leq t$, then we claim that $C_B = \varphi_{\alpha(a*,b*)}$ has the desired properties:

B is clearly a recursive set.

As in the proof of theorem 2.8, we can show that Comp B > t a.e.

To show $A \leq_p B$, we note the following:

For any n, consider all x with $(n+1)^2 \leq x < (n+2)^2$. For all such x, $f(x) = n$. There are $2n+3$ such values of x. However, before stage $(n+2)^2$, we only cancel indices $\leq n$. Thus, only $n+1$ of the values of x may have $C_B(x)$ defined by a cancellation. For the remaining $n+2$ values of x, we have $C_B(x) = C_A(n)$.

Then for any n,

$$C_A(n) = \begin{cases} 1 & \text{if } \sum_{x=(n+1)^2}^{(n+2)^2-1} C_B(x) \geq n+2, \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, $A \leq_p B$.

It remains to show Comp $B \leq h \circ t$ a.e.; to do this, we must first define h:

Let $h(x,y) = \max_{a,b \leq x} h'(x,y,a,b)$, where:

$$h'(x,y,a,b) = \begin{cases} \Phi_{\alpha(a,b)}(x) & \text{if } (\forall w \leq x)[\Phi_b(w) \leq y \ \wedge \ \Phi_a(w) \leq y], \\ 0 & \text{otherwise.} \end{cases}$$

The conditions on the right in the definition of h' are sufficient to insure that $\varphi_{\alpha(a,b)}(x)\downarrow$, so that $h' \in R_4$ and thus $h \in R_2$.

Now if we fix $a = a^*$ and $b = b^*$, we see that for $x \geq \max(a^*,b^*)$,

$$h(x,t(x)) \geq h'(x,\Phi_{b^*}(x),a^*,b^*)$$
$$= \Phi_{\alpha(a^*,b^*)}(x) \text{ a.e., as required.}$$

QED

Remark: If we do not require the compression of set B, a much simpler construction suffices:

Definition 4.4: For any X, Y, we define $X \oplus Y$ as follows:

$$(\forall x)[(x \in X \oplus Y) \Leftrightarrow ((x \in X \wedge x \notin Y) \vee (x \in Y \wedge x \notin X))].$$

Then if we take sufficiently large $t' \in R_1$ relative to t, we can obtain C recursive with Comp C > t' a.e., and let B = C join $(A \oplus C)$. This set B has two properties:

$$A \leq_p B$$

and

$$\text{Comp } B > t \text{ a.e.}$$

This second property is easily shown for space measure, using the parallel computation property, and recursive relatedness gives the result for general measures.

Results in this chapter have so far been rather intuitive and natural; less so are results stating "independence" of sets (for example, demonstrating the existence of pairs of recursive sets which do not help each other's computation).

Solutions to problems of this latter type turn out to be analogous to work on degrees of unsolvability [Sa] [Ro1, §10.2, Chapter 13] in the following sense:

Independence proofs proceed by a diagonalization (the only general tool we have thus far for proving such results). The diagonalizations require a countable sequence of conditions, or perhaps two different countable sequences of conditions, to be satisfied. Satisfaction of these various conditions may cause conflict. To insure that each condition

gets satisfied, we establish before the construction a "priority ordering" of conditions; in our theorems, this is a simple numerical ordering.

We allow the satisfaction of a condition to be interrupted only by switching to an attempt to satisfy a higher-priority condition. It follows that once we begin trying to satisfy some condition, we must thereafter succeed in satisfying either that condition or one of higher priority; thus, all conditions will eventually become satisfied.

Our arguments use priority more complex that the "initial segment" priority constructions in [Ro1, Chapter 13]; we do construct our sets by determining values first on initial segments, but we also carry with us "tentative commitments" to definition of the set at arguments a finite distance beyond the defined initial segment. It is only a finite distance beyond, so we are not using the full power of splitting-tree arguments, for example.

Our constructions differ from those in [Sa] and [Ro1], however, since we are constructing recursive sets. Our constructions are always effective, and we insure definition of the functions we construct at all arguments.

After a degree-of-unsolvability priority construction, arguments are usually presented showing what oracles are used in the construction, and thereby placing the constructed set in its proper Turing degree. We are working with a subrecursive analog of these constructions; we are generally interested in the complexity of the resulting set. Thus, we generally follow our constructions with arguments showing what

subcomputations were used in the computation constructing our set, thereby placing the constructed set in its proper complexity class.

We now aim to prove an independence theorem. In order to make the proof as compact as possible, we first introduce definitions designed to allow us to discuss the independence of the values of a 0-1 valued function at its different arguments. In theorem 4.6, we give an example of a simple theorem using this definition. Lemma 4.7 shows the existence of a 0-1 valued recursive function whose values at its different arguments are independent, while theorem 4.8 shows how to split this type of set into two sets which don't help each other's computation, thus giving a complexity-theoretic analog to the Friedberg-Muchnik theorem [Ro1, §10.2].

Definition 4.5: Assume A is a recursive set and g is a total function of one variable. Then:

"$\mathrm{Comp}^{[A]}A > g$ a.e." means:

$$(\forall i)[[(\forall x)(\varphi_i^{(A-\{x\})}(x) = C_A(x))] \Rightarrow (\Phi_i^{(A-\{x\})}(x) > g(x) \text{ a.e.})].$$

"$\mathrm{Comp}^{[A]}A \leq g$ a.e." means:

$$(\exists i)[[(\forall x)(\varphi_i^{(A-\{x\})}(x) = C_A(x))] \wedge (\Phi_i^{(A-\{x\})}(x) \leq g(x) \text{ a.e.})].$$

The following theorem, due to Paterson [P], shows the abundance of 0-1 valued functions whose values at different arguments are strongly dependent. This settles a question raised by Trachtenbrot [T1].

Theorem 4.6: There exist $r \in R_1$, $h \in R_2$ with the following property:

Whenever t is a monotone increasing running time, there exists a recursive set A such that:

$$\text{Comp } A > t \text{ a.e.}$$

$$\text{Comp } A \leq h \circ t \text{ a.e.}$$

and $$\text{Comp}^{\lceil A \rceil} A \leq r \text{ a.e.}$$

Proof: We define the set A, depending on t; we indicate how to construct r and h afterwards.

We define A by a construction in stages. As we do so, we cancel indices i such that we know $\varphi_i \neq C_A$.

Let $f(y) = \lfloor \frac{y}{4} \rfloor$.

Stage x: (Define $C_A(x)$)

See if there exists uncancelled $i \leq f(x)$ such that $\Phi_i(x) \leq t(x)$.

1. If so, define $C_A(x) = 1 \dot{-} \varphi_i(x)$.

   Cancel i.

   Go on to stage x + 1.

2. If no such i exists,

   2.1. If $|\{y | y < x \text{ and } y \in A\}|$ is even, we define $C_A(x) = 0$.

   Go on to stage x + 1.

   2.2. Otherwise, define $C_A(x) = 1$.

   Go on to stage x + 1.

END OF CONSTRUCTION

A is clearly recursive. We leave the reader to verify that substage 1. insures Comp A > t a.e.

Verification of the second claim depends on the construction of the proper h, which may be done by a domain-of-convergence argument.

To verify the third claim, we use the following procedure for obtaining $C_A(x)$ from C_A on other arguments:

For all x, B, define:

$$\varphi_i^{(B)}(x) = \begin{cases} 0 \text{ if there are more arguments } y,\ x+1 \leq y \leq 2x \text{ for which} \\ \qquad |(\{0,\ldots,y\} - \{x\}) \cap B| \text{ is even than for which} \\ \qquad |(\{0,\ldots,y\} - \{x\}) \cap B| \text{ is odd,} \\ 1 \text{ otherwise.} \end{cases}$$

That is, we use the fact that most arguments y with $x + 1 \leq y \leq 2x$ were used not to cancel indices, but to maintain parity.

From this procedure, it is easy to construct the function r for the time or space measure. Recursive relatedness then gives the result for general measures.

QED

Open Question: Is theorem 4.6 true without the monotonicity restriction?

Having some familiarity with the way a function's values at different arguments may interrelate, we now go on to produce a set A whose values at different arguments are independent. This result in announced by Trachtenbrot in [T2]. He gives no proof, however; the proof here is due to Meyer.

Lemma 4.7: (Trachtenbrot): There exists $g \in R_2$ with the following property:

For any sufficiently large total running time t, there exists a recursive set A with:

$$\text{Comp } A \leq g \circ t \text{ a.e.},$$

and

$$\text{Comp}^{\lceil A \rceil} A > t \text{ a.e.}$$

Proof: We would like to insure:

$$(\forall i)[(\Phi_i^{(A-\{x\})}(x) \leq t(x) \text{ i.o.}) \Rightarrow (\exists x)(\varphi_i^{(A-\{x\})}(x) \neq C_A(x))].$$

As before, we use cancellation; we cancel an index i when we've insured that:

$$(\exists x)(\varphi_i^{(A-\{x\})}(x) \neq C_A(x)).$$

In addition, at any time during the construction, a single index may be "tentatively cancelled." If an index i is tentatively cancelled, it means that we are in the process of attempting to cancel i by defining A according to an appropriate "tentative commitment." If we succeed in defining A in this way, we will then cancel i; otherwise, the tentative cancellation of i will be removed.

We will use the s-m-n theorem to define a partial recursive function $\varphi_{\alpha(a)}$, according to a construction in stages. For $\Phi_a = t$, the function $\varphi_{\alpha(a)}$ will turn out to be the C_A of our theorem. We use the parameter a to allow us later to obtain the desired recursive function g by a domain-of-convergence argument.

We will have stages numbered 0, 1, 2,..., where at stage n, we will define $\varphi_{\alpha(a)}(n)$. The stages are not executed in consecutive order, however. The order in which the stages are executed is determined as follows:

At any time when we are ready to decide which stage is next to be executed, the next stage will be stage n if and only if:

1. Stage n has not yet been executed,

2. $\Phi_a(n)\downarrow$,

3. For any m for which stage m is not yet executed,

$\Phi_a(n) \leq \Phi_a(m)$,

and 4. If $\Phi_a(n) = \Phi_a(m)$ for any m for which stage m is not yet executed, then $n \leq m$.

(That is, stages are executed in order of size of values of Φ_a.)

This trick is similar to that used in theorem 2.8.

We now describe stage n of the construction:

Stage n: (Define $\varphi_{\alpha(a)}(n)$)

Find the smallest $i \leq n$ that is not yet cancelled and such that:

(a) if some index j is tentatively cancelled, then $i < j$, and

(b) there exists E such that:

(b1) $(\forall x \mid \text{stage } x \text{ has already been executed})[x \in E \Leftrightarrow \varphi_{\alpha(a)}(x) = 1]$,

(b2) $E \subseteq \{x \mid x \leq h(i,n,\Phi_a(n))\}$, where h is the function whose existence is asserted by lemma 4.2.1,

(b3) $n \notin E$,

and (b4) $\Phi_i^{(E)}(n) \leq \Phi_a(n)$.

1. If such an i, E exist, remove any previous tentative cancellation and tentative commitment.

Define $\varphi_{\alpha(a)}(n) = 1 \dot{-} \varphi_i^{(E)}(n)$.

1.1. If $(\forall x \leq h(i,n,\Phi_a(n)))[\text{stage } x \text{ has already been executed}]$, then cancel i and go on to the next stage.

1.2. Otherwise, tentatively cancel i and make a "tentative commitment to define $\varphi_{\alpha(a)}$ so that:

$(\forall x | x \leq h(i,n,\Phi_a(n)) \wedge x \neq n) \lceil \varphi_{\alpha(a)}(x) = C_E(x) \rceil$.

Go on to the next stage.

2. If no such i exists,

2.1. If some index j is tentatively cancelled, consider E from its tentative commitment.

Define $\varphi_{\alpha(a)}(n) = C_E(n)$.

2.1.1. If for a, n arising from j's tentative commitment, we have $(\forall x \leq h(j,n,\Phi_a(n)))\lceil$stage x has already been executed$\rceil$, then remove j's tentative commitment. Change j's tentative cancellation to a cancellation.

Go on to the next stage.

2.1.2. Otherwise, just go on to the next stage.

2.2. If no index j is tentatively cancelled, just define $\varphi_{\alpha(a)}(n) = 0$.

Go on to the next stage.

END OF CONSTRUCTION

Now assume we have t as in the hypotheses. If we choose a* with $\Phi_{a*} = t$, then we claim that $C_A = \varphi_{\alpha(a*)}$ has the desired properties:

A is a recursive set:

If we assume that $t \geq \lambda x \lceil x \rceil$, then we may easily show that after each stage, the next stage may be chosen effectively. Also, the search for E in substage (b) of each stage will terminate. With these facts,

the execution of successive stages is an effective process.

Comp$^{[A]}_A > t$ a.e.:

We make an important observation about cancellations:

Fact: If an integer k is tentatively cancelled at some stage, then at that stage or later, some integer $\leq k$ will become cancelled.

Proof of fact: By induction on k.

Now for any index i, suppose that $(\forall x)[\varphi_i^{(A-\{x\})}(x) = C_A(x)]$.

Then it is not difficult to show that i is never cancelled. For if i were cancelled, it means that at some stage x we set up a tentative commitment for i which was eventually fulfilled, so that for some x, E as in substage (b), we defined:

$$C_A(x) \neq \varphi_i^{(E)}(x)$$

But by lemma 4.2.1 and the definition of E, we can conclude that:

$$\varphi_i^{(E)}(x) = \varphi_i^{(A-\{x\})}(x),$$

so that $C_A(x) \neq \varphi_i^{(A-\{x\})}(x)$.

Thus, i can never be cancelled.

There is some stage in the construction such that all cancellations of indices smaller than i that will ever occur have already occurred by that stage. By the fact above, it follows that at subsequent stages, condition (b) must fail to be satisfied for index i. But then lemma 4.2.1 implies that $\Phi_i^{(A-\{x\})}(x) > t(x)$ a.e.

Comp A $\leq g \circ t$ a.e.:

We see that $(\forall a,x)\ [\varphi_a(x)\downarrow \ \Rightarrow\ \varphi_{\alpha(a)}(x)\downarrow]$.

Thus, by the combining lemma, an appropriate function g exists.

QED

Remark: The following related result has been obtained by Meyer using similar methods (It has also been announced by Trachtenbrot in [T2], but without proof.):

"For any $g \in R_2$, there exists a recursive set A such that:

$$(\forall i)[((\forall x)(\varphi_i^{(A-\{x\})}(x) = C_A(x))) \ \Rightarrow\ ((\exists j)(\varphi_j = C_A \ \wedge \ g(x,\Phi_j(x)) \leq \Phi_i^{(A-\{x\})}(x) \text{ a.e.}))]."$$

In other words, the values of C_A at its different arguments are independent in the sense that, for any procedure for C_A which uses information about C_A on other arguments, there is a faster procedure for C_A which does not use any such information.

As a result of lemma 4.7, we now obtain the desired independence result:

Theorem 4.8: There exists $h \in R_2$ with the following property:

For any sufficiently large total running times t_B and t_C, there exist recursive sets B and C with:

$$\text{Comp } B \leq h \circ t_B \text{ a.e.},$$

$$\text{Comp } C \leq h \circ t_C \text{ a.e.},$$

$$\text{Comp}^{(C)}B > t_B \text{ a.e.},$$

and

$$\text{Comp}^{(B)}C > t_C \text{ a.e.}$$

Proof:

We use the following lemma:

Lemma 4.8.1: There exists $k \in R_2$ with the following property:
$(\forall \Phi_i)(\exists S_j)$ [$S_j \geq \Phi_i$ a.e. and $S_j \leq k \circ \Phi_i$ a.e.].

(That is, for any measure function, there exists a larger "space function" which is not much larger.)

Proof of lemma 4.8.1: By recursive relatedness, there exists a function r relating $S^{(\ )}$ and $\Phi^{(\ )}$. Without loss of generality, we may assume that $\lambda\langle x,y\rangle[r(x,y)]$ is a space function and r is monotone nondecreasing in both variables.

Assume f is the recursive isomorphism between the Gödel numberings of the two measures.

Then $r \circ S_{f(i)}$ is easily shown to be a space function, and if we let $k = \lambda x,y[r(x,r(x,y))]$, then recursive relatedness gives the required properties.

Proof of theorem 4.8, continued: Lemma 4.8.1 shows that if we prove the theorem for space measure on oracle Turing machines, recursive relatedness will give the general result.

It remains to prove the theorem for $S^{(\ )}$.

Definition 4.8.2: If f and g are any functions, we define "f join g" by:

$$(\forall x) \quad f \text{ join } g\ (2x) = f(x)$$
$$f \text{ join } g\ (2x + 1) = g(x).$$

If t_B and t_C are space functions, then t_B join t_C may also easily

be shown to be a space function. We may apply lemma 4.7 to t_B join t_C and obtain a recursive set A with:

$$\text{Comp } A \leq g \circ (t_B \text{ join } t_C) \text{ a.e.,}$$

and

$$\text{Comp}^{[A]} A > t_B \text{ join } t_C \text{ a.e.}$$

We write A = B join C, and claim that B and C have the required properties:

Comp$^{(C)}$B > t_B a.e.:

Otherwise, $(\exists i)[\varphi_i^{(C)} = C_B$ and $S_i^{(C)} \leq t_B$ i.o.].

But then we can convert $\varphi_i^{(\ )}$ to a program which computes C_A fast on infinitely many arguments, namely:

$$(\exists j)[(\forall x)(\varphi_j^{(A-\{x\})}(x) = C_A(x)) \text{ and } (\exists^{\infty} x)(S_j^{(A-\{x\})}(x) \leq t_B \text{ join } t_C(x))].$$

But this contradicts Comp$^{[A]}$A > t_B join t_C a.e.

Comp$^{(B)}$C > t_C a.e.:

By symmetry.

Comp B ≤ h ∘ t_B a.e., and Comp C ≤ h ∘ t_C a.e.:

Define h(x,y) = g(2x,y) + g(2x+1,y), where g is the function arising from lemma 4.7. We claim this function h has the required properties:

Comp A ≤ g ∘ (t_B join t_C) a.e. implies that:

$(\exists i)[\varphi_i = C_A$ and $S_i \leq g \circ (t_B$ join $t_C)$ a.e.].

But then φ_i may be easily converted to a program for C_B which on argument x, requires space g ∘ (t_B join t_C) (2x)

$= g(2x, t_B \text{ join } t_C(2x))$

$= g(2x, t_B(x))$

$= h(x, t_B(x))$, as required.

The other case is symmetric.

QED

The earliest independence result in the literature which can be stated in terms of existence of two recursive sets not helping each other is due to Axt [A2]. Axt states the existence of recursive sets A and B such that neither is primitive recursive in the other. His proof does not use the complexity formulation of "primitive recursive in"; it is an initial segment diagonal construction similar to theorem IV in [Ro1, Chapter 13]. With the complexity formulation (theorem 3.2) we may obtain Axt's result as a corollary to our theorem 4.8.

We now use theorem 4.8 to obtain counterexamples to transitivity and symmetry of helping.

Corollary 4.8.3: For sufficiently large functions k, the relations "k-improvement a.e." and "k-improvement i.o." are neither transitive nor symmetric.

Proof: We will describe three sets which provide a counterexample to all four properties.

We choose running times t_B and t_C with t_B much larger than t_C.

By theorem 4.8, we may obtain B, C and h. We then consider the three sets B, B join $(B \oplus C)$, and C. We note the following relationships between the sets:

1. $\neg$(B k-improves C i.o.)

2. $\neg$(C k-improves B i.o.)

3. B k-improves B join (B $\oplus$ C) a.e.

4. B join (B $\oplus$ C) k-improves B a.e.

5. B join (B $\oplus$ C) k-improves C a.e.

6. $\neg$(C k-improves B join (B $\oplus$ C) i.o.)

1. and 2. are clear by theorem 4.8, if $k > h$.

3. is true because a B-oracle reduces the complexity of B join (B $\oplus$ C) on even arguments to triviality and on odd arguments to the complexity of C. Since t_B is much larger than t_C, this is a large reduction in the complexity of B join (B $\oplus$ C) a.e. To formalize this argument, we can use a proof for the space measure and recursive relatedness.

4. is trivial.

5. is true since:

$$(\forall x)\lceil C_C(x) = C_{B \text{ join } (B \oplus C)}(2x) \oplus C_{B \text{ join } (B \oplus C)}(2x+1)\rceil.$$

6. If C k-improves B join (B $\oplus$ C) i.o., then either C k-improves B join (B $\oplus$ C) on infinitely many even arguments or infinitely many odd arguments. However, the independence of B and C obtained from theorem 4.8 shows that improvement cannot occur on infinitely many even arguments. Thus, C k-improves B join (B $\oplus$ C) on infinitely many odd arguments.

It then follows that C k'-improves (B $\oplus$ C) i.o., for some k' which is only slightly smaller than k. We show that this is impossible:

Consider the space measure. By definition of k'-improvement,

$$(\exists i)(\forall j)[\varphi_i^{(C)} = C_{B \oplus C} \quad \text{and} \quad [\varphi_j = C_{B \oplus C} \Rightarrow k'(x, S_i^{(C)}(x)) \leq S_j(x) \text{ i.o.}]].$$

By the parallel-computation property of $S^{( )}$,

$$(\exists j)[\varphi_j = C_{B \oplus C} \quad \text{and} \quad S_j \leq h \circ t_B \text{ a.e.}].$$

Combining the two facts,

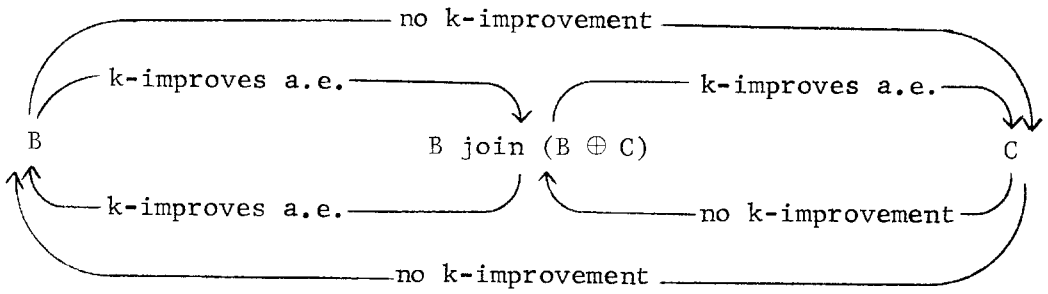
$$(\exists i)[\varphi_i^{(C)} = C_{B \oplus C} \quad \text{and} \quad k'(x, S_i^{(C)}(x)) \leq h \circ t_B(x) \text{ i.o.}].$$

We can use $\varphi_i^{(C)}$ and a program φ_1 for C_C with $S_1 \leq h \circ t_C$ a.e., and thereby obtain a program $\varphi_m^{(C)} = C_B$ with $k' \circ S_m^{(C)} \leq h \circ t_B$ i.o.

But for k' sufficiently large, this contradicts the hypothesized independence of B and C.

For general measures $\Phi^{( )}$, we obtain the same result if we require k to be much larger than the measure-invariance function r obtained by applying theorem 2.5 or remark 2.5.2 to $\Phi^{( )}$ and $S^{( )}$.

The following diagram summarizes our results:



Checking the diagram, we see that we have a counterexample to all four properties.

QED

5. Universally-Helped Sets

In this and the next chapter, we ask if it is possible to improve on theorem 4.8. Specifically, theorem 4.8 proves the existence of two independent recursive sets by a diagonalization. This leaves open the possibility that independent sets are pathological; we would like to know if we can obtain a stronger result which allows us to fix one of the two sets arbitrarily.

We are thus led to ask the following (informal) question:

Which is true?

(1) There is a recursive set A whose computation is helped by all sufficiently complex recursive sets B (a "universally-helped set"), or,

(2) For any recursive set A, there exist arbitrarily complex recursive sets B that don't help the computation of A.

Remark 5.1: We first note that any recursive set will be universally-helped in an appropriate measure: Fix any $k \in R_2$, monotone increasing in both variables. Let our model for computation be oracle Turing machines. Define a measure $\Phi^{(\ )}$ as follows:

$$\Phi_i^{(A)}(x) = \begin{cases} S_i^{(A)}(x) & \text{if } (\exists y \leq x)[y \in A], \\ k \circ k \circ S_i^{(A)}(x) & \text{otherwise.} \end{cases}$$

Now consider a "strongly k-compressed" set A; in other words, assume that there exists a total function t with:

$$(\forall i)[\varphi_i = C_A \Rightarrow S_i > t \text{ a.e.}],$$

and $$(\exists i)[\varphi_i = C_A \wedge S_i \leq k \circ t \text{ a.e.}].$$

We claim that in measure $\Phi^{(\ )}$, A is k-improved by any set $B \neq \phi$. The reason is as follows:

We have $(\exists i)[\varphi_i = C_A \wedge S_i \leq k \circ t \text{ a.e.}]$.

Let $\varphi_j^{(\ )}$ be the Turing machine that acts exactly like φ_i but never asks any questions of its oracle. Then:

$$(\forall X)[\varphi_j^{(X)} = C_A \text{ and } S_j^{(X)} = S_i \leq k \circ t \text{ a.e.}].$$

Consider any set $B \neq \phi$. Then $\varphi_j^{(B)} = C_A$.

Also, $\Phi_j^{(B)} = S_j^{(B)} = S_i$ a.e.

Thus, $\Phi_j^{(B)} \leq k \circ t$ a.e.

However, $(\forall 1)[\varphi_1 = C_A \Rightarrow \Phi_1 = k \circ k \circ S_1$

$\Rightarrow \Phi_1 = k \circ k \circ t \text{ a.e.}]$.

Therefore, B k-improves A a.e.

QED

Pathological measures such as those above show why we will require a "simulation overhead function" g in the various theorems of Chapter 6.

In the remainder of Chapter 5, we work within an arbitrary complexity measure and produce a recursive set A whose computation is helped by oracles for all sets whose complexity may be compressed between "honest" bounds (bound functions whose running times are closely related to their sizes.) We refer back to the remark following lemma 2.7.

Further discussion of honesty may be found in [McC] [McCMe] and [MeMo].

We note that theorem 4.3 has already given us that every recursive set A is helped by arbitrarily complex recursive sets B. These sets B, however, were constructed in a very special way, to encode A. In the next result, the sets B are not encodings of A, but may be described by a restriction (more-or-less) independent of A.

The function g in theorem 5.2, as well as in the theorems in Chapter 6, will depend only on the relative complexity measure $\Phi^{(\ )}$ we are considering, and will represent a fixed amount of extra resource needed to carry out certain simulations. For measures like space, we may often think of g as any function which majorizes (is greater than) linear functions; for other measures, we may still regard it as small relative to the other functions we are considering.

First, we require a definition:

Definition 5.2.1: For any $h \in R_2$ define a recursive set A_h as follows: Write $x = \langle x_1, x_2, x_3 \rangle$.

$$C_{A_h}(x) = \begin{cases} 1 \dot{-} (1 \dot{-} \varphi_{x_1}(x_2)) & \text{if } \Phi_{x_1}(x_2) \leq h(x_2, x_3), \\ 0 & \text{otherwise.} \end{cases}$$

(The use of the "$\dot{-}$" is only to keep C_{A_h} 0-1 valued.)

Theorem 5.2: (universally-helped set)

There exists $g \in R_2$ with the following property:

For all k, $h \in R_2$, $k(x,y) \geq y$, and all total running times t, any set B with:

$$\text{Comp } B > g \circ k \circ g \circ t \text{ i.o.}$$

and $\quad \text{Comp } B \leq h \circ t \text{ a.e.}$

k-improves A_h i.o.

Proof: We give three lemmas defining functions g_1, g_2 and g_3 which represent extra resource required to combine certain processes in the proof. The g in the statement of the theorem will be a combination of g_1, g_2 and g_3. We use domain-of-convergence arguments.

Lemma 5.2.2: There exists $g_1 \in R_2$ such that whenever Φ_i is total, there exists $\varphi_j = \Phi_i$ with:

$$g_1(x,\varphi_j(x)) \geq \Phi_j(x) \text{ a.e.}$$

(Note: This says that running times are honest. See the remark following lemma 2.7.)

Proof of lemma 5.2.2: By the axioms for relative complexity measures and the relativized s-m-n theorem, it is easy to see that there exists a recursive function α such that:

$$(\forall i,B,x)[\Phi_i^{(B)}(x) = \varphi_{\alpha(i)}^{(B)}(x)].$$

We define $g_1(x,y) = \max_{a \leq x} g'(x,y,a)$, where:

$$g'(x,y,a) = \begin{cases} \Phi_{\alpha(a)}(x) & \text{if } y = \Phi_a(x), \\ 0 & \text{otherwise.} \end{cases}$$

The second lemma gives an upper bound on the amount of resource needed, with a B-oracle, to compute C_{A_h}:

Lemma 5.2.3: There exists $g_2 \in R_2$ with the following property:

For any $h \in R_2$, any set B and any total running time t for which:

$$(\exists i)[(\varphi_i = C_B) \wedge (\Phi_i \leq h \circ t \text{ a.e.})],$$

it must be true that:

$$(\exists j)[(\varphi_j^{(B)} = C_{A_h}) \wedge (\Phi_j^{(B)}(\langle i,x,t(x)\rangle) \leq g_2 \circ t(x) \text{ a.e.})].$$

Proof of lemma 5.2.3: We let $\{F_i\}$ be a canonical enumeration of all functions defined on a finite domain. That is, we assume that $\lambda i,x[F_i(x)]$ is partial recursive and $\lambda i,x[C_{\text{domain } F_i}(x)]$ is total recursive.

Using the relativized s-m-n theorem, we define:

$$\varphi_{\alpha(a,b,c,d)}^{(X)}(x) = \begin{cases} F_a(x) & \text{if } x \in \text{domain } F_a, \\ C_X(x_2) & \text{if } x \notin \text{domain } F_a \text{ and } x_1 = b \text{ and} \\ & \quad \varphi_d(x_2) = x_3, \\ \varphi_c(x) & \text{otherwise.} \end{cases}$$

Now define $g_2(x,y) = \max_{a,b,c,d \leq x} g_2'(x,y,a,b,c,d)$, where:

$$g_2'(x,y,a,b,c,d) = \max_{X \subseteq N} g_2''(x,y,a,b,c,d,X), \text{ and:}$$

$$g_2''(x,y,a,b,c,d,X) = \begin{cases} \Phi_{\alpha(a,b,c,d)}^{(X)}(\langle b,x,\varphi_d(x)\rangle) & \text{if } \Phi_d(x) \leq g_1(x,y), \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 2.5.1 is used to show that g_2 is recursive.

We now fix a,b,c,d,X as follows:

Let F_a be a finite function giving values of C_{A_h} on all arguments of the form $\langle i,x,t(x)\rangle$ for which $\Phi_i(x) > h \circ t(x)$.

Let b = i, the given index for C_B.

Let c = an index for a program for C_{A_h}.

Let d be an index for t with the property that $\Phi_d(x) \leq g_1(x,t(x))$ a.e. (by lemma 5.2.2).

Let X = B.

It is now easy to verify that $\varphi_{\alpha(a,b,c,d)}^{(X)} = C_{A_h}$.

Also, if $x \geq \max(a,b,c,d)$, then:

$$g_2(x,t(x)) \geq g_2''(x,t(x),a,b,c,d,B)$$
$$= \Phi_{\alpha(a,b,c,d)}^{(B)}(\langle b,x,\varphi_d(x)\rangle) \text{ a.e., as required.}$$

The third lemma provides us with an upper bound for the amount of resource needed to convert a program for C_{A_h} into a program for C_B:

Lemma 5.2.4: There exists $g_3 \in R_2$ having the following property:

For any $h \in R_2$, any set B, any total running time t, and any i, j, 1 for which:

$$[(\varphi_i = C_B) \wedge (\Phi_i \leq h \circ t \text{ a.e.})],$$

and

$$\varphi_j = C_{A_h},$$

it must be true that:

$$(\exists 1)[(\varphi_1 = C_B) \wedge (\Phi_1(x) \leq g_3(x,\max(g_1(x,t(x)),\Phi_j(\langle i,x,t(x)\rangle))) \text{ a.e.})].$$

Proof of lemma 5.2.4: We use the s-m-n theorem to define:

$$\varphi_{\alpha(a,b,c,d)}(x) = \begin{cases} F_a(x) & \text{if } x \in \text{domain } F_a, \\ \varphi_c(\langle b,x,\varphi_d(x)\rangle) & \text{otherwise.} \end{cases}$$

Now define $g_3(x,y) = \max_{a,b,c,d \leq x} g'(x,y,a,b,c,d)$, where

$$g'(x,y,a,b,c,d) = \begin{cases} \Phi_{\alpha(a,b,c,d)}(x) & \text{if } \Phi_d(x) \leq y \text{ and } \Phi_c(\langle b,x,\varphi_d(x)\rangle) \leq y, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, $g_3 \in R_2$.

Now fix a, b, c and d as follows:

Let F_a be a finite function giving values of C_B on all arguments x such that $\Phi_i(x) > h \circ t(x)$.

Let b = i.

Let c = j.

Let d = an index for t with the property that $\Phi_d \leq g \circ t$ a.e. (by lemma 5.2.2).

It is easy to see that $\varphi_{\alpha(a,b,c,d)} = C_B$.

Now if $x \geq \max(a,b,c,d)$, then:

$$g_3(x,\max(g_1(x,t(x)),\Phi_j(\langle i,x,t(x)\rangle))) \geq$$

$$g'(x,\max(g_1(x,t(x)),\Phi_j(\langle i,x,t(x)\rangle)),a,b,c,d)$$

$$= \Phi_{\alpha(a,b,c,d)}(x) \quad \text{a.e.}(x), \text{ as required.}$$

<u>Proof of theorem 5.2, continued</u>:

We now define $g = \max(g_1,g_2,g_3)$.

Let $\varphi_i = C_B$ with $\Phi_i \leq h \circ t$ a.e.

Then by lemma 5.2.3,

$$(\exists j)[(\varphi_j^{(B)} = C_{A_h}) \wedge (\Phi_j^{(B)}(\langle i,x,t(x)\rangle) \leq g \circ t(x) \text{ a.e.})].$$

But lemma 5.2.4 implies:

$$(\forall j)[(\varphi_j = C_{A_h}) \Rightarrow (\Phi_j(\langle i,x,t(x)\rangle) > k \circ g \circ t(x) \text{ i.o.})],$$

since otherwise, we obtain a contradiction to the lower bound on B's

complexity.

But this clearly shows that B k-improves A_h i.o.

QED

We have thus described an interesting situation: we have sets which are helped by all sets whose complexities are compressed between honest bounds, and the extent to which they help depends on how tightly compressed their complexities are.

Nevertheless, there are many sets whose complexities are not so compressed; sets with speed-up are one example.

Open Question: Are there recursive sets which are h-compressed, but not between honest bounds? Specifically, is it true that:

For all g, h $\in R_2$, there exist sets A and functions t $\in R_1$ such that Comp A > t a.e. and Comp A $\leq$ g $\circ$ t a.e., but such that for no total running time t' is it true that [Comp A > t' a.e. and Comp A $\leq$ h $\circ$ t' a.e.]?

We can ask similar questions for Comp A > t i.o. and Comp A > t' i.o.

6. Sets That Don't Help

In this chapter, we present two theorems which have the opposite intuitive interpretation to the main result of Chapter 5. Both theorems begin with a recursive set A and a lower bound on the complexity of A, and conclude the existence of arbitrarily complex sets B that "preserve" the lower bound on A's complexity.

In both theorems in this chapter, a function g is used, representing a minimal amount of helping which the set B is allowed to give to A's computation. The functions g are necessary because of the existence of pathological measures such as those given in remark 5.1. As before, each function g will depend on the measure only and may therefore be considered to be very small compared to the other functions we are considering.

We begin with a definition:

Definition 6.1: We say that a property holds "for arbitrarily complex recursive sets" if:
$(\forall r \in R_1)(\exists B \text{ recursive})[\text{Comp } B > r \text{ a.e. and } B \text{ has the desired property}]$.

In the first theorem, we consider an i.o. lower bound. Intuitively, theorem 6.2 makes the following statement:

"For any recursive set A whose complexity exceeds a known lower bound i.o., there exist arbitrarily complex sets B such that the complexity of A with a B-oracle still exceeds the lower bound i.o."

The method of proof is similar to that used by Machtey [Ma2, theorem

4.9] in his proof of the result:

"If f and g are recursive functions with f not primitive recursive, then there exists a recursive set C such that f is not primitive recursive in C."

We remark that for the sets B of our construction, it is still possible that B may greatly help A i.o.

Theorem 6.2: There exists $g \in R_2$ with the following property:

For any $t_A \in R_1$, and any recursive set A with:

$$\text{Comp } A > g \circ t_A \text{ i.o.},$$

there exist arbitrarily complex recursive sets B with:

$$\text{Comp}^{(B)} A > t_A \text{ i.o.}$$

Proof: We obtain g from the following lemma:

Lemma 6.2.1: There exists $g \in R_2$ with the following property:

Whenever A is a finite set, $r \in R_1$, and $\varphi_i^{(A)} = r$ a.e.,

there exists j such that:

$$\varphi_j = r \qquad \text{(on all arguments)},$$

and $$\Phi_j \leq g \circ \Phi_i^{(A)} \quad \text{a.e.}$$

Proof of lemma 6.2.1: Follows from the combining lemma.

Without loss of generality, we may assume that g is monotone increasing in both variables.

Proof of theorem 6.2, continued:

We choose $t_B \in R_1$ arbitrarily. t_B will be a lower bound on B's complexity.

We will define B in stages, with $C_B(x)$ being defined at stage x. During the construction, we cancel indices of programs we know to differ from C_B. In the course of the construction, integers a, b and c will be defined and changed from stage to stage, where:

a keeps count of how many conditions of a certain type we have so far succeeded in satisfying,

b ($= \pi_1(a)$) indicates which B-oracle program we are currently examining, and

c keeps track of a tentative commitment to an extension of the already-defined initial segment of B.

We let $B_x = \{y \leq x | y \in B\}$.

We start with a = b = 0, c undefined. (That is, we have not yet satisfied any of the conditions we would like to satisfy, we are examining $\varphi_0^{(B)}$, and we have no tentative commitment to an extension of the already-defined initial segment of B.)

Stage x:

See if there exists $i < a$ such that i is not yet cancelled and $\Phi_i(x) \leq t_B(x)$.

1. If so, consider the smallest such i.

   Let $C_B(x) = 1 \dot{-} \varphi_i(x)$, and cancel i.

   Let c become undefined.

   Go on to stage x + 1.

2. If no such i exists, define $C_B(x) = 0$.

See if c is defined.

2.1. If so, see if c = x.

2.1.1. If c = x, redefine $a = a + 1$,

$b = \pi_1(a)$,

c = undefined,

and go on to stage x + 1.

2.1.2. If $c \neq x$, just go on to stage x + 1.

2.2. If c is not defined, see if there exists an argument y such that $a \leq y \leq x$ and <u>either</u> $\Phi_b^{(B_x)}(y) > t_A(y)$,

<u>or</u> $(\Phi_b^{(B_x)}(y) \leq t_A(y)$ and $\varphi_b^{(B_x)}(y) \neq C_A(y))$.

2.2.1. If so, let h be the function whose existence is asserted in lemma 4.2.1 and consider $h(b,y,t_A(y))$.

2.2.1.1. If $h(b,y,t_A(y))$ is $\leq x$, then redefine:

$a = a + 1$,

$b = \pi_1(a)$,

c = undefined.

Go on to stage x + 1.

2.2.1.2. If $h(b,y,t_A(y)) > x$, then define $c = h(b,y,t_A(y))$.

Go on to stage x + 1.

2.2.2. If no such argument y exists, just retain the values of a and b and go on to stage x + 1.

END OF CONSTRUCTION

Verification of the construction:

The key claim is that the variable a in the construction must increase without bound. Suppose it does not.

This would mean that eventually a would reach a stable value, say a_0. Thereafter, neither 2.1.1 nor 2.2.1.1 will be executed.

Eventually, we will reach a stage high enough so that 1. can no longer be executed (since there are only finitely many $i < a$). Thereafter, 2. must always be executed.

Subsequently, if c is ever defined, then there is no way for c to become undefined. Thus, we would be forced to execute 2.1 at every stage until we are compelled to execute 2.1.1, a contradiction. Thus, c is subsequently never defined.

But this implies that 2.2. must be executed at every stage from some point on. However, 2.2.1.1. cannot be executed, and 2.2.1.2. cannot be executed since c cannot become defined. Therefore, from some stage on, no argument y satisfying the conditions in 2.2. will ever be found.

But this means that for $b_0 = \pi_1(a_0)$:

$$(\exists x)(\forall y \geq a_0)[(\varphi_{b_0}^{(B_x)}(y) = C_A(y)) \wedge (\Phi_{b_0}^{(B_x)}(y) \leq t_A(y))].$$

But then lemma 6.2.1 gives a program φ_j such that:

$$\varphi_j = C_A \quad \text{and} \quad \Phi_j \leq g \circ t_A \text{ a.e.},$$

contradicting the hypotheses of the theorem.

Thus, we see that a must increase without bound.

Comp B > t_B a.e.:

For any index i, if $\Phi_i \leq t_B$ i.o., then clause 1. will eventually become executed for i, insuring that $\varphi_i \neq C_B$.

Comp$^{(B)}$A > t_A i.o.:

Assume the contrary: $(\exists i)[(\varphi_i^{(B)} = C_A) \wedge (\Phi_i^{(B)} \leq t_A \text{ a.e.})]$.

Then there exists some least integer a_0 such that $\pi_1(a_0) = i$ and $(\forall y \geq a_0)[\Phi_i^{(B)}(y) \leq t_A(y)]$. When a is first set equal to a_0 at some stage, c is undefined, by 2.1.1. or 2.2.1.1.

Since a grows without bound, it follows that eventually 2.2.1. must get executed at some stage x when $a = a_0$.

But this implies that:

$$(\exists y, a_0 \leq y \leq x)[(\Phi_i^{(B_x)}(y) > t_A(y)) \vee [(\Phi_i^{(B_x)}(y) \leq t_A(y)) \wedge (\varphi_i^{(B_x)}(y) \neq C_A(y))]].$$

Moreover, eventually thereafter, either 2.2.1.1. or 2.1.1. must get executed, unless prior to their execution, clause 1. is executed. But if this happens, then c again becomes undefined so 2.2.1. must again get executed. 1. can only intervene finitely many times, since there are only finitely many indices less than a_0. Thus, we can assume without loss of generality that 1. does not intervene.

But in this case, we insure that:

$(\exists y \geq a_0)[(\Phi_i^{(B)}(y) > t_A(y)) \vee [(\Phi_i^{(B)}(y) \leq t_A(y)) \wedge (\varphi_i^{(B)}(y) \neq C_A(y))]]$, by lemma 4.2.1., which is a contradiction.

QED

Remark 6.2.2: We note that, for the space measure $S^{(\ )}$, the function $\lambda x,y\lceil y\rceil$ will suffice to satisfy lemma 6.2.1 and hence theorem 6.2, provided that t_A is nontrivial (i.e. $t_A \geq \lambda x\lceil x\rceil$). The method for showing this is not the method of the given proof of the lemma, but rather a direct proof by analysis of oracle Turing machine space measure; information about A and about the finitely-many exceptions to "$\varphi_i^{(A)} = r$" may be stored in the Turing machine's finite control.

In fact, if we are interested only in the space measure, it is not only possible to sharpen our result, but to simplify its proof as well. This is because of the following fact about $S^{(\ )}$, not true for general measures:

Fact 6.2.3: $(\forall A,\ i,\ \forall t \in R_1$ with $t \geq \lambda x\lceil x\rceil)$

If $S_i^{(A)} \leq t$ a.e. and $\varphi_i^{(A)}$ is total, then:

$$(\exists j)\lceil(\varphi_j^{(A)} = \varphi_i^{(A)}) \wedge (S_j^{(A)} \leq t \text{ everywhere})\rceil.$$

Fact 6.2.3 implies that for $S^{(\ )}$, we need only insure:

$$(\forall i)(\exists y)\lceil\varphi_i^{(B)} = C_A \Rightarrow S_i^{(B)}(y) > t_A(y)\rceil,$$

rather than:

$$(\forall i)(\exists^{\infty} y)\lceil\varphi_i^{(B)} = C_A \Rightarrow S_i^{(B)}(y) > t_A(y)\rceil.$$

This eliminates the need to consider each B-oracle program infinitely often during the construction; we need only consider it once. Thus, the need for variable b is eliminated.

Theorem 6.2 provides the following corollary about primitive recursive reducibility:

Corollary 6.2.4: If A is any recursive set which is not primitive recursive, then there exist (in any measure) arbitrarily complex sets B such that $A \not\leq_p B$ and $B \not\leq_p A$.

(Recall that $\leq_p$ means "primitive recursive in.")

Proof: We show the result for the space measure; clearly, recursive relatedness gives the general-measure result.

Since $(\forall h)[h \text{ is primitive recursive} \Leftrightarrow h \leq_p \phi]$, the proof of corollary 3.5.2, with each $B_i = \phi$, shows how to obtain a recursive function f such that:

$$(\forall h)[h \text{ is primitive recursive} \Leftrightarrow \text{Comp } h \leq f \text{ a.e.}].$$

Thus, Comp A > f i.o.

By theorem 6.2 for $t_A = f$ and space measure (where $g = \lambda x,y[y]$, as in remark 6.2.2), we obtain arbitrarily complex sets B such that $\text{Comp}^{(B)}A > f$ i.o.

We claim that f is greater than or equal to each primitive recursive function of one variable, a.e. For if not, then:

$(\exists h,$ a primitive recursive function of one variable$)[h > f \text{ i.o.}]$.

The time and space measures on oracle Turing machines may be shown to be recursively related (in the sense of theorem 2.5) by a primitive recursive function r. Then the function $2^{r \circ h}$ is clearly primitive recursive. However, it requires <u>time</u> $\geq r \circ h$ to compute the function on all arguments, since it requires that much time just to output the answer. Therefore, it requires <u>space</u> $\geq h$ to compute this function on almost all arguments, by recursive relatedness. Therefore,

$\text{Comp}\ 2^{r \circ h} > f$ i.o., a contradiction. Thus, f must be greater than or equal to each primitive recursive function of one variable a.e.

But then $\text{Comp}^{(B)}A > f$ i.o. implies that A cannot be computed using a B-oracle in primitive recursive space, so $A \not\leq_p B$ by theorem 3.2.

By making B sufficiently complex, theorem 4.2 shows that we can obtain $\text{Comp}^{(A)}B > f$ i.o. Thus, by theorem 3.2, $B \not\leq_p A$.

QED

We would like to compare theorem 5.2 with theorem 6.2 to demonstrate that there is no conflict between them. We note that theorem 5.2 produces sets A_h which are helped infinitely often by all sets B whose complexities are "compressed" around running times. Theorem 6.2 does not produce sets B with such a restriction on their complexities. For proper comparison, we would therefore like a stronger, "compressed" version of theorem 6.2.

We may obtain such a strengthened version of the theorem if we are willing to allow some additional assumptions: namely, we assume that t_B is a running time, and that t_B is monotone and much larger than the complexity of t_A and the complexity of A.

New Assumptions: $(\exists i,j,k)[(t_A = \varphi_i) \wedge (t_B = \Phi_j) \wedge (C_A = \varphi_k)$
$\wedge (\forall x)[t_B(x) \geq \max(t_B(x-1), \Phi_i(x), \Phi_k(x))]]$

The new statement of the theorem is as follows:

Proposition 6.2.5:

There exists $g \in R_2$ with the following property:

Whenever we have t_A, $t_B \in R_1$, A a recursive set, Comp $A > g \circ t_A$ i.o., and the New Assumptions satisfied, then there exists a recursive set B such that:

$$\text{Comp } B > t_B \text{ a.e.,}$$

$$\text{Comp } B \leq g \circ t_B \text{ a.e.,}$$

and

$$\text{Comp}^{(B)} A > t_A \text{ i.o.}$$

Proof: Uses the construction in theorem 6.2, and a domain-of-convergence argument to estimate the complexity of B. We omit the details.

We require one further lemma before making our comparison:

Lemma 6.2.6: In any measure $\Phi^{(\ )}$, there exist arbitrarily large monotone increasing running times.

Proof: Let us fix any $t \in R_1$.

We use the recursion theorem [Rol] to define:

$$\varphi_i(x) = \begin{cases} 0 & \text{if } x = 0, \\ & \text{or if } [(\varphi_i(x - 1)\downarrow) \text{ and } (\Phi_i(x) > \max(t(x), \Phi_i(x - 1)))] \\ \infty & \text{otherwise.} \end{cases}$$

It is easy to show that φ_i must be total and Φ_i has the required properties.

We now note the following:

Let us use the function g found in proposition 6.2.5.

Define $g^2 \in R_2$ as $\lambda x,y[g(x,g(x,y))]$. Then we may obtain, by theorem 5.2, a set A_g, which is i.o. g^2-improved by all recursive sets B whose

complexity is weakly g-compressed around a running time. (That is, there exists a total running time t_B such that Comp $B > t_B$ i.o. and Comp $B \leq g \circ t_B$ a.e.)

Now we claim that there exists a recursive function t_A which is a "good" i.o. lower bound for $A_{g'}$'s complexity, in the sense that $C_{A_{g'}}$ can be computed a.e. in measure not much greater than t_A. (This is true provided g' is honest, an assumption we may make without loss of generality. An examination of the proof of theorem 5.2 shows that:

$$\lambda\langle i,x,y\rangle[g'(x,y)]$$

approximates an i.o. lower bound for the complexity of $A_{g'}$, and that $C_{A_{g'}}$ can actually be computed a.e. in measure not much greater than this function.)

Using lemma 6.2.6. to obtain the appropriate function t_B, we may apply proposition 6.2.5 to the function t_A and obtain a set B. What is B's relationship to $A_{g'}$?

B must g^2-improve $A_{g'}$ i.o., by theorem 5.2. On the other hand, since B preserves the i.o. lower bound t_A (at least to within amount g), it is impossible that B g^2-improve $A_{g'}$ a.e. Intuitively, B g^2-improves $A_{g'}$ i.o. and B fails to g^2-improve $A_{g'}$ i.o. There is, of course, no conflict here.

We note that theorem 6.2 has a real relationship to "improvement" only in the case where t_A is actually a "good" lower bound for A's complexity (i.e. C_A <u>can</u> be computed a.e. in measure not much more than t_A) In the case of sets A having such "good" lower bounds, theorem

6.2 allows us to conclude (for sufficiently large k) the existence of sets B for which it is false that B k-improves A a.e.

However, not all recursive sets have such "good" i.o. lower bounds. For example, sets whose characteristic functions have sufficient speed-up cannot have good i.o. lower bounds. For this type of set, theorem 6.2 gives us no information about improvement.

Open Question: Can we obtain a more symmetrical version of theorem 6.2, in which A also preserves a lower bound on B's complexity?

This question may be precisely formulated in several different ways. One example is as follows: Is it true that:

There exists $g \in R_2$ with the following property:

Whenever we have t_A, $t_B \in R_1$, A a recursive set, Comp $A > g \circ t_A$ i.o. and the New Assumptions satisfied, then there exists a recursive set B such that:

$$\text{Comp}^{(A)}B > t_B \text{ i.o.},$$

$$\text{Comp } B \leq g \circ t_B \text{ a.e.},$$

and

$$\text{Comp}^{(B)}A > t_A \text{ i.o.}$$

Open Question: For any recursive set A, we have managed to find sets B which preserve any single i.o. lower bound t_A on A's complexity. Can we find, for each A, a single set B which preserves all i.o. lower bounds which happen to be total running times)? Further discussion of this question will appear in Chapter 7.

The next theorem, theorem 6.3, is similar to theorem 6.2, but the kind of lower bound we are considering is an a.e. lower bound instead

of an i.o. lower bound. Theorem 6.3 is almost a companion theorem to theorem 6.2; it does require the additional assumptions that t_A is a running time, and that t_A is sufficiently large, however.

A note on the type of priority construction used for this theorem: the proof is a finite-injury priority argument with no apparent recursive bound on the number of injuries of each condition.

Theorem 6.3: There exists $g \in R_2$ with the following property:

For any total running time t_A such that $t_A \geq \lambda x\lceil x \rceil$, and any recursive set A with:

$$\text{Comp } A > g \circ t_A \text{ a.e.},$$

there exist arbitrarily complex recursive sets B with:

$$\text{Comp}^{(B)} A > t_A \text{ a.e.}$$

We choose a function t_B to be an a.e. lower bound on B's complexity. Without loss of generality (as we see from lemma 6.2.6 above) we may assume that t_B is a monotone increasing running time.

We describe a construction which will give us the required set B, working from t_A, t_B and A. We use the s-m-n theorem. The parameters a, b and c in the construction are to be thought of as follows:

Φ_a will be t_A,

Φ_b will be t_B,

φ_c will be C_A.

Definition of $\varphi_{\beta(a,b,c)}$ (which will turn out to be C_B for a, b, c as above)

$\varphi_{\beta(a,b,c)}$ will be defined in stages, with $\varphi_{\beta(a,b,c)}(n)$ being

defined at stage n.

During the construction, we keep track of two types of cancellation, which we call 1-cancellation and 2-cancellation. We 1-cancel an index i when we have succeeded in defining $\varphi_{\beta(a,b,c)}$ in such a way that: $(\exists x,y)(\forall C)[(C_C|\{0,\ldots,y\} = \varphi_{\beta(a,b,c)}|\{0,\ldots,y\}) \Rightarrow (\varphi_i^{(C)}(x) \neq C_A(x))]$. These 1-cancellations will be used to insure $\text{Comp}^{(B)}A > t_A$ a.e.

We 2-cancel an index i when we have insured that $\varphi_i \neq \varphi_{\beta(a,b,c)}$. Indices i get 2-cancelled when Φ_i is less than t_B sufficiently many times. This will insure Comp $B > t_B$ a.e.

Once an index is 1-cancelled or 2-cancelled, it remains so at all later stages.

Also, at any particular time during the construction, we may have some "tentatively 1-cancelled" indices. If an index i is tentatively 1-cancelled, a pair of integers (x_i,y_i) will be defined such that if we ever discover that $C_A(x_i) \neq y_i$, then i will become 1-cancelled. If we ever discover that $C_A(x_i) = y_i$, then the tentative 1-cancellation will be removed.

The same index may become tentatively 1-cancelled and lose its tentative 1-cancellation repeatedly, the values of (x_i,y_i) changing with each tentative 1-cancellation, but we will see that (in the cases in which we are interested) may index can only become tentatively 1-cancelled finitely often.

Finally, at any time during the construction we may have a "tentative commitment for (an index) i". A tentative commitment for

i is a quadruple (i,x_i,y_i,z_i), where z_i is the canonical index of a 0-1 valued function F_{z_i} with finite domain such that:

$$(\forall C)[(C_C \mid \text{domain } F_{z_i} = F_{z_i}) \Rightarrow (\varphi_i^{(C)}(x_i) = y_i)],$$

and F_{z_i} is an extension of the finite portion of $\varphi_{\beta(a,b,c)}$ defined at the time of the tentative commitment to i. The tentative commitment is designed to allow us to subsequently tentatively 1-cancel i, if possible.

We will eventually fulfill the tentative commitment for i, at which time i becomes tentatively 1-cancelled, unless we are interrupted by the 2-cancellation of an index smaller than (i.e. of higher priority than) i, or by a new tentative commitment for an index smaller than i.

In both the following constructions, we will speak of the "first" members of certain collections of finite sets; it is to be understood that the ordering we are using is lexicographic.

At the beginning of Stage 0, there are no 1-cancellations, tentative 1-cancellations, or 2-cancellations, or tentative commitments.

Stage n: (Define $\varphi_{\beta(a,b,c)}(n)$)

1. Compute $\Phi_b(n)$ and $\Phi_b(n-1)$.

   (If either diverges, then $\varphi_{\beta(a,b,c)}$ will diverge.)

   Let $X = \{x \leq \Phi_b(n) \mid \Phi_b(n-1) < \Phi_a(x) \leq \Phi_b(n)\}$.

   See if either of the following, (a) or (b), holds:

   (a) There exist i, x, E such that:

      (a1) $i \leq n$, i is neither 1-cancelled nor tentatively 1-cancelled, and if there is a tentative commitment for some j, then $i < j$,

(a2) $x \in X$,

(a3) $E \subseteq \{y | y \leq h(i,x,\Phi_a(x))\}$ (where h is the function whose existence is asserted in lemma 4.2.1), and:

$$(\forall y \leq n - 1)[y \in E \Leftrightarrow \varphi_{\beta(a,b,c)}(y) = 1],$$

and (a4) $\Phi_i^{(E)}(x) \leq \Phi_a(x)$.

(b) There exists $i \leq n$, where i is not 2-cancelled, and if there is a current tentative commitment for some j, then $i < j$, and:

$$\Phi_i(n) \leq \Phi_b(n).$$

1.1. If neither (a) nor (b) holds,

1.1.1. If there is no current tentative commitment, define $\varphi_{\beta(a,b,c)}(n) = 0$ and go on to substage 2.

1 1.2. If there <u>is</u> a tentative commitment (j,x_j,y_j,z_j), let $\varphi_{\beta(a,b,c)}(n) = F_{z_j}(n)$. Go on to substage 2.

1.2. If either (a) or (b) does hold, fix i to be the smallest index for which either (a) or (b) is true.

1.2.1. If i arises from (a), choose the x such that $\Phi_a(x)$ is smallest (if two are equal, choose the smaller x), and for this x choose the first set E such that (i,x,E) satisfy (a). Remove any previous tentative commitment, and make a new <u>tentative commitment for i</u>,

$$(i,x,\varphi_i^{(E)}(x),z_i),$$

where z_i is the canonical index of the function:

$$F_{z_i} = C_E|\{y|y \leq \max(n,h(i,x,\Phi_a(x)))\}.$$

Define $\varphi_{\beta(a,b,c)}(n) = F_{z_i}(n)$, and go on to substage 2.

1.2.2. If i arises from (b) but not from (a), define:

$$\varphi_{\beta(a,b,c)}(n) = 1 \dot{-} \varphi_i(n),$$

and 2-cancel i. Remove any previous tentative commitment and go on to substage 2.

2. See if there is a current tentative commitment (i,x_i,y_i,z_i) such that $n \geq \max$ (domain F_{z_i}).

2.1. If so, tentatively 1-cancel i, associating (x_i,y_i) with the tentative 1-cancellation.

Remove the tentative commitment and go on to substage 3.

2.2. If not, then just go on to substage 3.

3. For each tentatively 1-cancelled index i with an associated pair of integers (x_i,y_i), see if $\Phi_c(x_i) \leq \Phi_b(n)$.

3.1. If not, then make no change.

3.2. If so, then:

3.2.1. If $\varphi_c(x_i) = y_i$, remove i's tentative 1-cancellation.

3.2.2. If $\varphi_c(x_i) \neq y_i$, remove i's tentative 1-cancellation and 1-cancel i.

Go on to stage n + 1.

END OF CONSTRUCTION

It is easy to verify that if Φ_b is total, then for any a, c $\in$ N,

$\varphi_{\beta(a,b,c)} \in R_1$ and $\varphi_{\beta(a,b,c)}$ is 0-1 valued.

Now choose a*, b* and c* such that $\Phi_{a*} = t_A$, $\Phi_{b*} = t_B$ and $\varphi_{c*} = C_A$.

Let $C_B \stackrel{def}{=} \varphi_{\beta(a*,b*,c*)}$.

We claim that this set B has the required properties. The key step in the proof is the claim that no index gets 1-cancelled i.o. If we assume this for the moment, the rest of the proof is straightforward:

It is easy to see that Comp B > t_B a.e., as in earlier proofs: if $\Phi_i \leq t_B$ i.o., then i will be 2-cancelled once all the finitely-many higher priority indices which are ever going to be 2-cancelled are so cancelled, and once all the (finitely many) tentative 1-cancellations of higher priority indices have been made. When i is 2-cancelled, clause 1.2.2. guarantees that $\varphi_i \neq C_B$.

We now claim that $\text{Comp}^{(B)}A > t_A$ a.e.

For if not, then there is an index i such that $\varphi_i^{(B)} = C_A$ and $\Phi_i^{(B)} \leq t_A$ i.o.

Such an i could never be 1-cancelled during the construction of B, for this would mean that for some finite set E and some argument x,

$C_A(x) \neq \varphi_i^{(E)}(x)$ by the 1-cancellation,

$= \varphi_i^{(B)}(x)$ by lemma 4.2.1, and clauses 1.2.1. and 2.1.

Therefore, each tentative 1-cancellation of i will eventually be removed by clause 3.2.1.

We will eventually reach some stage e in the construction of B such

that after stage e, no $j < i$ becomes tentatively 1-cancelled or 2-cancelled. Beyond stage e, clauses (a), (b) and 1.2 insure that no index smaller than i can prevent a tentative commitment for i from being made, nor can an index smaller than i interrupt such a tentative commitment for i.

Thus, whenever i satisfies clause (a) at some stage $n > e$, and i is not already tentatively 1-cancelled at stage n, i will become tentatively 1-cancelled at stage n.

But by lemma 4.2.1, $\Phi_i^{(B)}(x) \leq t_A(x)$ implies the existence of a set E such that (i,x,E) satisfies clause (a). Since $\Phi_i^{(B)}(x) \leq t_A(x)$ i.o., i will satisfy clause (a) i.o., and so must become tentatively 1-cancelled i.o.

But we have assumed that no index i is tentatively 1-cancelled i.o.

Thus, $\text{Comp}^{(B)}A > t_A$ a.e.

It remains to show that no index can become tentatively 1-cancelled infinitely often. In order to do this, we construct (by the s-m-n theorem) a function with five parameters, $\varphi_{\gamma(a,b,c,d,e)}$.

The parameters are to be thought of as follows:

$\Phi_a = t_A$,

$\Phi_b = t_B$,

$\varphi_c = C_A$,

d = the first index which becomes tentatively 1-cancelled infinitely many times, and

e = the number of a stage beyond which no index smaller than d ever becomes tentatively 1-cancelled or 2-cancelled. We assume $e > d$.

Then $\varphi_{\gamma(a,b,c,d,e)}$ will represent a program for C_A requiring measure $\leq g \circ t_A$ (for an appropriate function g). We will let this be the g in the hypothesis of the theorem, so that we here obtain a contradiction to "Comp $A > g \circ t_A$ a.e."

Definition of $\varphi_{\gamma(a,b,c,d,e)}$:

To compute $\varphi_{\gamma(a,b,c,d,e)}(x)$, we proceed as follows:

If $\Phi_a(x)\uparrow$ or $(\forall n)[\Phi_a(x) > \Phi_b(n)]$, then $\varphi_{\gamma(a,b,c,d,e)}(x)\uparrow$.

Otherwise, let $n = \mu m[\Phi_a(x) \leq \Phi_b(m)]$.

1. If $n \leq e$, let $\varphi_{\gamma(a,b,c,d,e)}(x) = \varphi_c(x)$.

2. If $n > e$, then perform stages 0 through n - 1 in the construction of $\varphi_{\beta(a,b,c)}$. At the point immediately after completing stage n - 1, see if <u>either</u> there is a tentative commitment (d,x_d,y_d,z_d) <u>or</u> d is tentatively 1-cancelled.

   2.1. If either condition is true, let $\varphi_{\gamma(a,b,c,d,e)}(x) = \varphi_c(x)$.

   2.2. Otherwise, see if some tentative commitment (d,x',y,z), for $x \neq x'$ would be made at clause 1.2.1. of stage n in the construction of $\varphi_{\beta(a,b,c)}$. That is, see if:

$$(\exists x' \leq \Phi_a(x))[[(\Phi_b(n-1) < \Phi_a(x') < \Phi_a(x)) \vee (\Phi_a(x) = \Phi_a(x') \text{ and } x' < x)] \wedge [(\exists E \subseteq \{y \mid y \leq h(d,x',\Phi_a(x'))\})[(\forall y \leq n-1)(y \in E \Leftrightarrow$$

$\varphi_{\beta(a,b,c)}(y) = 1) \wedge (\Phi_d^{(E)}(x') \leq \Phi_a(x'))]]$.

2.2.1. If so, let $\varphi_{\gamma(a,b,c,d,e)}(x) = \varphi_c(x)$.

2.2.2. If not, then see if $(\exists E \subseteq \{y | y \leq h(d,x,\Phi_a(x))\})$
$[(\forall y \leq n - 1)(y \in E \Leftrightarrow \varphi_{\beta(a,b,c)}(y) = 1) \wedge$
$(\Phi_d^{(E)}(x) \leq \Phi_a(x))]$.

(We are checking to see if at stage n in the construction of B, a tentative commitment $(d,x,\varphi_d^{(E)}(x),z)$ was made.)

2.2.2.1. If not, let $\varphi_{\gamma(a,b,c,d,e)}(x) = \varphi_c(x)$.

2.2.2.2. If so, consider the first such E and let:

$$\varphi_{\gamma(a,b,c,d,e)}(x) = \varphi_d^{(E)}(x).$$

END OF CONSTRUCTION

We now assume (as indicated briefly before the construction of $\varphi_{\gamma(a,b,c,d,e)}$ that a*, b*, c*, d* and e* are fixed as follows:

$\Phi_{a*} = t_A$,

$\Phi_{b*} = t_B$,

$\varphi_{c*} = C_A$,

d* = the first index which becomes tentatively 1-cancelled infinitely many times during the construction of $\varphi_{\beta(a*,b*,c*)}$, and

e* = the number of a stage in the construction of $\varphi_{\beta(a*,b*,c*)}$ after which no index smaller than d* ever becomes tentatively 1-cancelled or 2-cancelled. We assume e* > d*.

We claim $\varphi_{\gamma(a*,b*,c*,d*,e*)} = C_A$.

For all clauses except 2.2.2.2., $\varphi_{\gamma(a^*,b^*,c^*,d^*,e^*)} = \varphi_{c^*} = C_A$. We must check what happens if clause 2.2.2.2. defines $\varphi_{\gamma(a^*,b^*,c^*,d^*,e^*)}(x)$ to be $\varphi_{d^*}^{(E)}(x)$.

If 2.2.2.2. is executed in defining some $\varphi_{\gamma(a^*,b^*,c^*,d^*,e^*)}(x)$, then it means that there is a stage $n > e^*$ in the construction of $\varphi_{\beta(a^*,b^*,c^*)}$ at which d^*, x and some set E satisfy the conditions in 1. (a) of that construction.

d^* must be the smallest index for which either (a) or (b) is satisfied because we are already past stage e^*.

Thus, in stage n of the construction of $\varphi_{\beta(a^*,b^*,c^*)}$, clause 1.2.1. must be executed for $i = d^*$.

But since clause 2.2.1. of the construction of $\varphi_{\gamma(a^*,b^*,c^*,d^*,e^*)}(x)$ was not executed, it must be the case that no other argument x' could interfere with a tentative commitment (d^*,x,y_{d^*},z_{d^*}) being made at stage n in the definition of $\varphi_{\beta(a^*,b^*,c^*)}$, and so some tentative commitment $(d^*,x,\varphi_{d^*}^{(E)}(x),z_{d^*})$ will be made.

Eventually, this tentative commitment for d^* will cause d^* to become tentatively 1-cancelled, since $n > e^*$. When d^* becomes tentatively 1-cancelled, it will be associated with the pair of integers $(x,\varphi_{d^*}^{(E)}(x))$.

Since d^* becomes tentatively 1-cancelled infinitely often during the construction of $\varphi_{\beta(a^*,b^*,c^*)}$, this tentative cancellation must eventually be removed. This can only happen because of clause 3.2.1. at some stage $m > n$ in the construction of $\varphi_{\beta(a^*,b^*,c^*)}$, but 3.2.1. gets

executed only if:

$$C_A(x) = \varphi_{d^*}^{(E)}(x) = \varphi_{\gamma(a^*,b^*,c^*,d^*,e^*)}(x).$$

This establishes the claim that $C_A = \varphi_{\gamma(a^*,b^*,c^*,d^*,e^*)}$.

Next, we would like to show that:

$$\Phi_{\gamma(a^*,b^*,c^*,d^*,e^*)} \leq g \circ \Phi_{a^*} \ (= g \circ t_A) \text{ i.o.}$$

To do this, we must first define g.

Let $g(x,y) = \max_{a,b,c,d,e \leq x} g'(x,y,a,b,c,d,e)$, where we will define g' below. The idea behind the definition of g' is the following: we list enough conditions, each recursive assuming the preceding ones are satisfied, to insure that $\varphi_{\gamma(a,b,c,d,e)}(x)$, and hence $\Phi_{\gamma(a,b,c,d,e)}(x)$, converges. On the other hand, we don't put in so many restrictions that we exclude any of the cases we're interested in (i.e. we only list properties actually satisfied by a*, b*, c*, d* and e*).

Here, we basically follow the construction of $\varphi_{\gamma(a,b,c,d,e)}(x)$ and select which of the conditions on a*, b*, c*, d* and e* were needed for the convergence of $\varphi_{\gamma(a^*,b^*,c^*,d^*,e^*)}(x)$.

We define $g'(x,y,a,b,c,d,e) = \Phi_{\gamma(a,b,c,d,e)}(x)$ provided all the following conditions are satisfied:

1. $e > d$,

2. $y = \Phi_a(x)$,

3. $\Phi_b(y) \geq \Phi_a(x)$,

4. Let $n = \mu m[\Phi_a(x) \leq \Phi_b(m)]$. Then:

4.1. $n > e$,

4.2. Immediately after performing stage $n - 1$ in the construction of $\varphi_{\beta(a,b,c)}$, there is no tentative commitment (d,x_d,y_d,z_d), and d is not tentatively 1-cancelled,

(Note: This is an effective test since we know that for any $m < n$, $\Phi_b(m) < \Phi_a(x)$, so that $\varphi_b(m)\downarrow$. This suffices to insure that $\varphi_{\beta(a,b,c)}(m)\downarrow$.)

4.3. There is no x' satisfying clause 2.2. of the construction of $\varphi_{\gamma(a,b,c,d,e)}(x)$. Precisely,

$$\neg(\exists x' \leq \Phi_a(x))[[(\Phi_b(n-1) < \Phi_a(x') < \Phi_a(x)) \vee (\Phi_a(x) = \Phi_a(x') \text{ and } x' < x)] \wedge [(\exists E \subseteq \{z \mid z \leq h(d,x',\Phi_a(x'))\})[(\forall z \leq n-1)(z \in E \Leftrightarrow \varphi_{\beta(a,b,c)}(z) = 1) \wedge (\Phi_d^{(E)}(x') \leq \Phi_a(x'))]]].$$

4.4. $(\exists E \subseteq \{z \mid z \leq h(d,x,\Phi_a(x))\})$
$[(\forall z \leq n-1)(z \in E \Leftrightarrow \varphi_{\beta(a,b,c)}(z) = 1) \wedge (\Phi_d^{(E)}(x) \leq \Phi_a(x))]$.

If one of the conditions fails to be satisfied, we define:

$g'(x,y,a,b,c,d,e) = 0$.

Now by definition,

$$g(x,\Phi_{a^*}(x)) \geq g'(x,\Phi_{a^*}(x),a^*,b^*,c^*,d^*,e^*) \quad \text{a.e.}$$
$$= \Phi_{\gamma(a^*,b^*,c^*,d^*,e^*)}(x) \quad \text{for all } x \text{ such that}$$

a tentative commitment (d^*,x,y_{d^*},z_{d^*}) is made at some stage after stage e^* in the construction of B.

But since we have assumed that d^* gets tentatively 1-cancelled

infinitely often, this latter equality must occur for infinitely many x.

Thus, we have $\Phi_{\gamma(a^*,b^*,c^*,d^*,e^*)} \leq g \circ \Phi_{a^*}$ i.o.

But since $\varphi_{\gamma(a^*,b^*,c^*,d^*,e^*)} = C_A$, this contradicts the hypothesis: Comp $A > g \circ t_A$ a.e. Therefore, our assumption that d^* was tentatively 1-cancelled infinitely often was wrong, and so we conclude that no index gets tentatively 1-cancelled infinitely often in the construction of $\varphi_{\beta(a^*,b^*,c^*)}$.

QED

Remark: Part of the difficulty of the preceding proof arises from the fact that there is no evident effective way to estimate how many tentative commitments for an index may be removed. The resemblance to finite injury priority arguments without recursive bounds on the number of injuries is readily apparent.

We now wish to discuss the relationship between theorems 6.3 and 5.2. As in the parallel discussion for theorem 6.2, we require a "compressed" version of the theorem:

We obtain:

Proposition 6.3.3: There exists $g \in R_2$ with the following property:

Whenever we have total running times t_A and t_B with t_B monotone increasing, and a recursive set A with Comp $A > g \circ t_A$ a.e., and if $t_A \geq \lambda x\lceil x \rceil$, then there exists a recursive set B such that:

$$\text{Comp } B > t_B \text{ a.e.}$$

$$\text{Comp } B \leq g \circ t_B \text{ a.e.,}$$

and

$$\text{Comp}^{(B)} A > t_A \text{ a.e.}$$

Proof: It is easy to verify that $(\forall x,a,b,c)\lceil x \in \text{domain } \varphi_{\beta(a,b,c)}$ whenever $\{0,1,\ldots,x\} \subseteq \text{domain } \Phi_b\rceil$.

Thus, by a simple domain-of-convergence argument, we may find a function $g' \in R_2$ such that:

$$\Phi_{\beta(a,b,c)} \leq g' \circ \Phi_b \text{ a.e.}$$

whenever Φ_b is monotone increasing, and the proposition follows immediately from theorem 6.3.

QED

Having this stronger version of theorem 6.3, we may now note the following:

Let us use the function g found in proposition 6.3.3.

As before, define $g^2 \in R_2$ to be $\lambda x,y\lceil g(x,g(x,y))\rceil$. Then we obtain, by theorem 5.2, a set $A_{g'}$ which is i.o. g^2-improved by all recursive sets B whose complexity is weakly g-compressed around a running time. (That is, there exists a total running time t_B such that Comp $B > t_B$ i.o. and Comp $B \leq g \circ t_B$ a.e.)

Now assume that there exists a recursive function t_A which is a "good" a.e. lower bound for $A_{g'}$, is complexity, in the sense that $C_{A_{g'}}$ can actually be computed a.e. in measure not much greater than t_A.

If such a function t_A exists, we may apply proposition 6.3.3. to t_A and an appropriate t_B and obtain a set B. What is B's relationship to $A_{g'}$?

B must g^2-improve $A_{g'}$ i.o., by theorem 5.2. On the other hand, since B preserves the a.e. lower bound t_A (at least to within amount g),

it is impossible that B g^2-improve A_g, i.o. But this is a contradiction.

Hence, we see that the assumed existence of the "good" a.e. lower bound was false, so A_g, can have no "good" a.e. lower bound on its complexity.

Open Question: Can we obtain a more symmetrical version of theorem 6.3, in which A also preserves a lower bound on B's complexity? (This is stronger than theorem 4.8 because the set A may be fixed arbitrarily.) One way of formulating this is the following:

Is this true:

"There exists $g \in R_2$ with the following property:

Whenever we have total running times t_A and t_B with t_B monotone increasing, and a recursive set A with Comp $A > g \circ t_A$ a.e., then there exists a recursive set B with:

$$\text{Comp}^{(A)}B > t_B \text{ a.e.}$$

$$\text{Comp } B \leq g \circ t_B \text{ a.e.}$$

and $$\text{Comp}^{(B)}A > t_A \text{ a.e."} \quad ?$$

Open Question: For any recursive set A, we have found sets B which preserve any single a.e. lower bound t_A on A's complexity, provided t_A is a running time. Can we find, for each A, a single set B which preserves all running time a.e. lower bounds? Further discussion of this question will appear in Chapter 7.

Open Question: In theorem 6.3, can the hypothesis that t_A is a running time be eliminated?

7. Suggestions for Further Study:

In this chapter, we collect open problems from Chapters 3-6 and make further suggestions for additional work.

7.1. In theorem 6.3, can the "running time" hypothesis on t_A be eliminated? That is, can we prove the following theorem:

"There exists $g \in R_2$ with the following property:

For any $t_A \in R_1$ (with $t_A \geq \lambda x\lceil x \rceil$), and any recursive set A with Comp $A > g \circ t_A$ a.e.,

there exist arbitrarily complex recursive sets B such that:

$$\text{Comp}^{(B)}A > t_A \text{ a.e."} \quad ?$$

7.2. Can we strengthen theorem 5.2 to omit or weaken the complexity restrictions on B?

For example, can we prove:

"For all $k \in R_2$, there exist sets A and functions $t \in R_1$ such that:

$$(\forall B)\lceil \text{Comp } B > r \text{ a.e.} \Rightarrow B \text{ k-improves } A \text{ i.o.}\rceil" \quad ?$$

There are other possible formulations of the same (intuitive) question.

7.3. Part of the intention of introducing the notion of "helping" was to give a formal interpretation of the way in which a subroutine helps the computation of a function. (The oracle set plays the role of a subroutine). Intuitively, it appears that a definition of "helping" which accurately reflects this situation should be transitive, contrary to corollary 4.8.5. We see that the difficulty

arises because the "helping sets" we consider in the counterexample are in fact unlike subroutines because they encode the entire function whose computation they are supposed to help (i.e., we say B join (B $\oplus$ C) helps the computation of C).

We would like some way of eliminating this difficulty. Can we make some suitable restriction on the kinds of sets allowed as oracles (e.g. restrict their complexity relative to that of the function being helped) to make our notion of helping a transitive one?

If we intend, as above, to give a formal interpretation of the way in which a subroutine helps the computation of a function, then we should not restrict our attention to set oracles, but rather we should have a model for computation which allows help from arbitrary partial recursive functions.

Some work in this direction has already been done by Symes in his thesis [Sy]. Symes has defined an acceptable ordering of "subroutine operators" which work not in conjunction with an arbitrary set oracle (as do relative algorithms), but rather in conjunction with a partial recursive function. Complexity axioms in the style of Blum are then developed for these operators.

7.4. Consideration of the kind of complexity restrictions needed on B in theorem 5.2, and elsewhere in the thesis, leads us to inquire about the relationships between the different types of complexity restrictions.

For example, we ask which of the following statements are true:

7.4.1. $(\exists g \in R_2)(\forall h \in R_2)(\exists A, \exists t \in R_1)$

Comp $A > t$ a.e. and Comp $A \leq g \circ t$ a.e.,

but such that for no total running time t' is it true that:

Comp $A > t'$ a.e. and Comp $A \leq h \circ t'$ a.e.

("A is strongly compressed but not between honest bounds.")

7.4.2. $(\exists g \in R_2)(\forall h \in R_2)(\exists A, \exists t \in R_1)$

Comp $A > t$ i.o. and Comp $A \leq g \circ t$ a.e.,

but such that for no total running time t' is it true that:

Comp $A > t'$ i.o. and Comp $A \leq h \circ t'$ a.e.

("A is weakly compressed but not between honest bounds.")

7.5. Can we obtain a version of theorem 4.2 for oracle sets B which are nonrecursive? That is, can we find any way to bound the amount of help a nonrecursive set B can give the computation of a function (for example, relative to B's Turing-reducibility properties, or to B's complexity relative to some oracle set)?

7.6. For any recursive set A, theorem 6.2 gives sets B which preserve any single i.o. lower bound t_A on A's complexity. Can we find, for each A, a single recursive set B which preserves all i.o. lower bounds on A's complexity (or, more restrictively, all i.o. lower bounds which happen to be running times)? This would imply that B failed to help A, in a somewhat more natural sense than theorem 6.2.

For example, it might be possible to somehow take into account <u>all</u> i.o. lower bounds on A's complexity, and hence construct B by

working from all the i.o. lower bounds rather than a single one.

7.7. For any recursive set A, we have found sets B which preserve any single running time a.e. lower bound on A's complexity. Can we find, for each A, a single set B which preserves all a.e. lower bounds, (or all running time a.e. lower bounds)?

A serious drawback to the idea of taking into account all a.e. lower bounds on A's complexity is the following claim of Meyer [Me]:

"There exists a recursive set A such that no sequence of total functions $\{p_i\}$ satisfies the following properties:

(a) $\lambda i,x[p_i(x)]$ is recursive,

(b) $(\forall i)[\text{Comp } A > p_i \text{ a.e.}]$

(c) $(\forall r \in R_1)[(\text{Comp } A > r \text{ a.e.}) \Rightarrow (\exists j)(p_j > r \text{ a.e.})]$"

We do not know whether a similar result holds for i.o. lower bounds.

7.8. In Chapter 6, the sets B which don't help A, fail to help it in that they preserve a given lower bound on A's complexity. It would be nice to be able to sharpen theorems 6.2 and 6.3 to involve k-improvement, by applying them to a single "best possible" lower bound on A's complexity.

Unfortunately, Blum's speed-up theorem [B1] tells us that for some recursive functions, no lower bound is close enough to the actual running time of a program for that function to insure that the resulting set B does not k-improve A. That is, some recursive functions do not have their complexities well-described by a single lower bound function.

However, it can be shown [MeF] that the complexity of any recursive function may be described by a sequence of recursive functions $\{p_i\}$ called a "complexity sequence," having some nice properties:

(a) $\lambda i,x[p_i(x)]$ is recursive,

(b) Each p_i is a total running time,

(c) $(\forall i)[h \circ p_i \geq p_{i+1}$ a.e.] (for some h depending on the measure only).

Using the concept of a "complexity sequence," Meyer has obtained a version of theorem 4.8 in which neither of the sets B or C k-improves the other for any nontrivial k.

In the hope that this method will extend to other problems (notably the questions in Chapter 6), it would be nice to better understand how the complexity of recursive functions can be characterized in terms of complexity sequences.

For example, two specific questions:

7.8.1. Does every recursive function f have a 0-1 valued recursive function g with "approximately" the same complexity sequence? This means that there exists a function $h \in R_2$ depending on the measure only, for which:

$(\exists \{f_i\}$, a complexity sequence for f), and

$(\exists \{g_i\}$, a complexity sequence for g)

$[(\forall f_i)(\exists g_j)(h \circ f_i \geq g_j$ a.e.) $\wedge$ $(\forall g_i)(\exists f_j)(h \circ g_i \geq f_j$ a.e.)].

7.8.2. Give necessary and sufficient conditions on a sequence of functions that it be a complexity sequence.

For example, in [L], we have two sets of sufficient conditions, where each set of conditions includes a synchronization condition (i.e. infinitely many of the functions in the sequence are large and small at the same arguments). However, we can show that synchronization conditions are not necessary in the following strong sense: we can obtain functions f with effective i.o. speed-up [B2] such that no complexity sequence for f can be synchronized.

7.9. Many other questions about helping besides those in Chapters 4-6 may be asked, some of which are probably answerable by methods similar to those used in Chapters 4-6. There are different formalizations of the same intuitive questions, but some examples are:

7.9.1. Does there exist a "universally-helping set"?

Specifically, is it true that:

$(\forall h \in R_2)(\exists A$, a recursive set$)(\forall t$, total running times$)(\forall B)$

$[(\text{Comp } B > t \text{ a.e. and Comp } B \leq h \circ t \text{ a.e.}) \Rightarrow (A \text{ h-improves } B)]$?

7.9.2. Can a set always be helped in a "controlled" way?

Specifically, is it true that:

$(\forall h \in R_2)(\forall$ total running times t, t' with $t > t')(\exists$ recursive sets $A,B)$

$[(\text{Comp } A > t \text{ a.e.}) \wedge (\text{Comp } A \leq h \circ t \text{ a.e.}) \wedge (\text{Comp}^{(B)} A > t' \text{ a.e.})$

$\wedge (\text{Comp}^{(B)} A \leq h \circ t' \text{ a.e.})]$?

7.10. Is theorem 4.6 true without the monotonicity restriction? Namely, is it true that:

"There exist $r \in R_1$, $h \in R_2$ with the following property:

Whenever t is a running time, there exists a recursive set A with:

$$\text{Comp } A > t \text{ a.e.},$$

$$\text{Comp } A \leq h \circ t \text{ a.e.},$$

and $$\text{Comp}^{\lceil A \rceil} A \leq r \text{ a.e."} \quad ?$$

7.11. Can we obtain more symmetrical versions of theorems 6.2 and 6.3, in which A also preserves a lower bound on B's complexity?

One example of a precise formulation of this question for theorem 6.2 is the following: Is it true that:

"There exists $g \in R_2$ with the following property:

Whenever we have t_A, $t_B \in R_1$, A a recursive set, Comp A $> g \circ t_A$ i.o., there exists a recursive set B such that:

$$\text{Comp}^{(A)} B > t_B \text{ i.o.}$$

$$\text{Comp } B \leq g \circ t_B \text{ a.e.}$$

and $$\text{Comp}^{(B)} A > t_A \text{ i.o."} \quad ?$$

7.12. This thesis deals almost exclusively with results about functions in R_n and $R_n^{(A)}$. It would be interesting to consider similar results in P_n and $P_n^{(A)}$. For example, questions 7.4 and 7.8 may be rephrased for partial functions.

We may use Symes' definitions [Sy] for programs with partial recursive functions in place of oracles, and reformulate our questions about helping in these terms.

7.13. Is is possible to strengthen proposition 3.14 in the following way:

"Given any nonrecursive set A, it is always possible to find a

set B such that:

$A \mid_{T} B$ and A-reducibility = B-reducibility." ?

7.14. Is bounded truth-table reducibility, or any of the other reducibilities mentioned by Jockusch in his thesis [J1] complexity-determined?

7.15. In his thesis, Jockusch develops the properties of various types of truth-table reducibilities, such as containment properties of degrees. Explore the answers to these questions for C-reducibilities for various sets C. For example, does C-reducibility have any properties significantly different from truth-table reducibility?

BIBLIOGRAPHY

[A1] Axt, P. Enumeration and the Grzegorczyk Hierarchy, Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik, 9, 1963, pp. 53-65.

[A2] Axt, P., On a Subrecursive Hierarchy and Primitive Recursive Degrees, Transactions of the A. M. S., 92, 1959, pp. 85-105.

[B1] Blum, Manuel, A Machine-Independent Theory of the Complexity of Recursive Functions, JACM, Vol. 14, No. 2, April 1967, pp. 322-336.

[B2] Blum, Manuel, On Effective Procedures for Speeding Up Algorithms, JACM, Vol. 18, No. 2, April, 1971, pp. 290-305.

[C] Cobham, Alan, The Intrinsic Computational Difficulty of Functions, Proceedings of the 1964 International Conference for Logic, Methodology and Philosophy of Science, Jerusalem, Aug. 26-Sept. 2, 1964, pp. 24-30.

[D] Davis, Martin, Computability and Unsolvability, McGraw-Hill, 1958.

[HH] Hartmanis, J. and Hopcroft, J. E., An Overview of the Theory of Computational Complexity, JACM, Vol. 18, No. 3, July, 1971, pp. 444-473.

[HLS] Stearns, R. E., Hartmanis, J., Lewis II, P. M., Hierarchies of Memory Limited Computations, IEEE Conf. Record on Switching Circuit Theory and Logical Design, 6, 1965, pp. 179-190.

[J1] Jockusch, Carl G, Jr., Reducibilities in Recursive Function Theory, PhD thesis, Department of Mathematics, M. I. T., June, 1966.

[J2] Jockusch, Carl G., Jr., Uniformly Introreducible Sets, Journal of Symbolic Logic, Vol. 33, No. 4, Dec. 1968.

[K] Kleene, Stephen Cole, Introduction to Metamathematics, Van Nostrand, 1959.

[LR] Landweber, L.H., and Robertson, E.L., Recursive Properties of Abstract Complexity Classes, accepted for publication in JACM.

[Ly] Lynch, Nancy, desk drawer.

[Ma1] Machtey, Michael, private communication.

[Ma2] Machtey, Michael, Augmented Loop Languages and Classes of Computable Functions, to appear.

[McC] McCreight, Edward M., Classes of Computable Functions Defined by Bounds on Computation, PhD thesis, Department of Computer Science, Carnegie-Mellon University, July, 1969.

[McCMe] McCreight, E. M. and Meyer, A. R., Classes of Computable Functions Defined by Bounds on Computation, Symposium on Theory of Computing, Marina del Rey, May, 1969.

[McL] McLaughlin, T. G. Private communication.

[Me] Meyer, Albert R., private communication.

[MeF] Meyer, Albert R. and Fischer, Patrick C., Computational Speed-up by Effective Operators, Journal of Symbolic Logic, Vol. 36, No. 4 Dec. 1971.

[MeRD] Meyer, A. R. and Ritchie, Dennis M., A Classification of Functions by Computational Complexity: Extended Abstract, Hawaii Int'l Conference on System Science, Jan. 1968.

[MeMo] Meyer, A. R. and Moll, Robert, Honest Bounds for Complexity Classes of Recursive Functions, MIT Project Mac publication, April, 1972.

[P] Paterson, Michael, private communication.

[RD] Ritchie, Dennis M., Program Structure and Computational Complexity, PhD thesis, Division of Engineering and Applied Physics, Harvard University, 1967.

[RRW] Ritchie, R. W., Classes of Predictably Computable Functions, Trans. A. M. S., 106, 1963, 139-173.

[Ro1] Rogers, Hartley Jr., Theory of Recursive Functions and Effective Computability, McGraw-Hill, 1967.

[Ro2] Rogers, Hartley Jr., Gödel Numberings of Partial Recursive Functions, JSL, Vol. 23, No. 3, September, 1958, pp. 331-341.

[Sa] Sacks, Gerald E., Degrees of Unsolvability, Annals of Mathematical Studies, No. 55, 1963, Princeton, N. J.

[Sy] Symes, David M., The Extension of Machine-Independent Computational Complexity Theory to Oracle Machine Computation and to the Computation of Finite Functions, PhD thesis, Department of Applied Analysis and Computer Science, University of Waterloo, Oct., 1971.

[T1] Trachtenbrot, B. A., On Autoreducibility, Dokl. Akad. Nauk.SSSR, Vol. 11 (1970), No. 3.

[T2] Trachtenbrot, B. A., private communication.

BIOGRAPHICAL NOTE

The author was born Nancy Evraets on January 19, 1948, in Brooklyn, New York. She attended P.S. 192, where she won the annual spelling bee in 1959.

She attended Hunter College High School from 1961-1964, and Brooklyn College from 1964-1968. At Brooklyn College, she held a New York State Regents Scholarship and a National Science Foundation grant for summer study. She was elected to Pi Mu Epsilon, Phi Beta Kappa and Sigma Xi. She served in numerous offices, including those of president of the Brooklyn College chapter of Pi Mu Epsilon and editor of the school's mathematics journal.

She received her B. S. in June, 1968, summa cum laude.

She began graduate work at M. I. T. in September, 1968, as an N. S. F. fellow. Her final year of graduate study was completed as a Project Mac research assistant.

During her attendance at M. I. T., she became Nancy Lynch, worked for a short time on the LOGO Project (for computers in education) and learned how to ski.

She has accepted an appointment as Assistant Professor of mathematics at Tufts University, Medford, Mass., beginning in September, 1972.

This empty page was substituted for a blank page in the original document.

CS-TR Scanning Project
Document Control Form

Date : 1/23/96

Report # LCS-TR-99

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 130 (136 -IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☒ Unknown ☐ Other:_____

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number): FOLLOW TITLEPAGE, 2, 3, 4 AND 125

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:

(A) IMAGE MAP: (1-130) UNH'ED TITLE, BLANK, 2, BLANK, 3, UNH BLANK, 4, UNH'ED BLANK, 5-125, UNH BLANK, (131-136) SCANCONTROL, COVER, TRGT'S (3)

(B) SIZE: 8"X 11"

Scanning Agent Signoff:
Date Received: 1/23/96 Date Scanned: 1/26/96 Date Returned: 2/1/96

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**