INDEXINGS OF SUBRECURSIVE CLASSES

Dexter KOZEN
IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598, U.S.A.

Communicated by R. Book
Received October 1978
Revised August 1979

Abstract. A subrecursive indexing is a programming language or Gödel numbering for a class of total recursive functions. Several properties of subrecursive indexings, such as effective composition and generation of constant functions, are investigated from an axiomatic point of view. The result is a theory akin to the axiomatic treatment of recursive function theory of Strong and Wagner. Using this formalism, we prove results relating the complexity of uniform simulation, diagonalization, deciding membership, and deciding halting; we give a subrecursive analog of Rice's theorem; we give a characterization of the combinatorial power of subrecursive indexings analogous to the combinatorial completeness of the lambda calculus; finally, we give a characterization the power of diagonalization over subrecursive classes and show that if \( P \neq NP \) is provable at all, then it is provable by diagonalization.

1. Introduction

An indexing is like a Gödel numbering or programming language for a class of computable functions. It provides a syntactic representation (indices or programs) for semantic objects (functions or sets). A class of functions may have many different indexings; for example, both ALGOL programs and Turing machines provide an indexing for the partial recursive functions. Indexings allow functions to behave as functionals (functions of higher type) by treating some of their inputs as indices of functions. This admits important notions like uniform effective composition, uniform simulation, translation, and diagonalization.

The purpose of this work is to initiate a study of the properties of commonly used indexings of classes of total functions, such as complexity classes, in order to understand the relationship between these properties and the complexity of diagonalization and uniform simulation. For example, the class of polynomial time computable functions is often indexed by Turing machines with polynomial time counters. Might it be that insisting on this particular indexing increases the difficulty of proving \( P \neq NP \) by diagonalization? We give an affirmative answer to this question in Section 7.

Recursive function theory is largely the study of well-behaved indexings of the partial recursive functions, variously called Gödel numberings, programming systems,
or acceptable indexings. The power of the partial recursive functions is captured in the two axioms for Gödel numberings, namely the s-m-n property and the universal function property. It is interesting to note that these axioms only provide for the existence of certain computable functionals. Although the combination of the s-m-n and universal function properties is too strong for classes of total functions, there are nevertheless certain agreeable properties enjoyed by most standard subrecursive indexings. For example, given programs $x$ and $y$ for functions $\phi_x$ and $\phi_y$, it is usually easy to construct a program for their composition $\phi_x \circ \phi_y$. For another example, it is usually easy to construct from $x$ a program computing the constant function $\lambda y \cdot x$.

By these criteria, ALGOL is a very agreeable indexing indeed: to compose two ALGOL programs, simply concatenate them with a semicolon in between, perhaps changing some variable names.

Why do most standard indexings satisfy these agreeable properties? The reason is that such properties allow easy construction of programs from specifications. Unfortunately, it seems that the more agreeable such properties are, the more difficult it is to simulate functions in the class uniformly or to show the class is properly contained in another by diagonalization. As a motivating example, consider the polynomial time counter indexing of PTIME mentioned above. Let us be a little more careful in the description of this indexing: each Turing machine is equipped with a preprocessor and an extra tape initially inscribed with some $k > 0$ in binary. On inputs of length $n$, the preprocessor writes $n^k$ in unary on the extra tape, then enters the main processor. The machine then computes as usual, erasing one symbol of the extra tape per move, and shutting off when all symbols on the extra tape have been erased. The collection of all (encodings over $\{0, 1\}^*$ of) such machines provides an indexing of PTIME. It is easy to see that this indexing is agreeable in the above sense. Yet, we have

**Theorem.** No universal simulator for this indexing can run in polynomial space.

This is proved in Section 7 as a special case of Theorem 7.4. Thus, given that diagonalization and uniform simulation are related (their relationship is made precise in Sections 4 and 6), the above theorem says that it will be difficult to prove PTIME $\neq$ PSPACE by diagonalization, using this indexing. A complimentary result of Section 6 says that if PTIME $\neq$ PSPACE, then there is a diagonal function in PSPACE $-\text{PTIME}$, no matter what the indexing.

Our approach to this topic is axiomatic, in the spirit of [14, 18, 20]. The reasons for this approach are twofold: first, we can avoid the subtle quirks peculiar to each machine model. The resulting theory, besides being more general, is easier to understand, since we know exactly what assumptions are in effect. For example, by considering functions over an arbitrary domain $\mathcal{D}$ instead of $\Sigma^*$, say, we can be sure that none of the properties of $\Sigma^*$ have crept into the arguments in some subtle way. Second, by observing the discrepancies between the standard and nonstandard models of the theory, we learn what extra properties are satisfied by the standard
models, which are the ones we are really interested in. For example, studying abstract complexity measures as axiomatized by Blum [4] has greatly aided our understanding of the more natural measures of time and space.

Several authors have studied subrecursive indexings from an abstract point of view. Alton [1, 2] was interested in speed-up phenomena in subrecursive classes. Machtey and Young [13] developed applications for limited halting problems in complexity theory. Both these axiomatizations use some notion of absolute complexity in the form of a complexity measure. Mehlhorn [14] formulated axioms for subrecursive reducibilities with the intent of generalizing Ladner's [10] and Machtey's [12] structure results for polynomial time and other concrete reducibilities. His axiomatization did not deal with absolute complexity, but rather with relative complexity in the form of a reducibility relation.

In this paper we have attempted to extract the essential elements common to all these attempts. In Section 2 we give an axiomatization of subrecursive indexings and give several examples. In Section 3 we discuss the relationship between subrecursive indexings and subrecursive reducibility relations as viewed by Mehlhorn [14]. In Section 4 we focus on the relationship between uniform simulation, diagonalization, deciding membership, and the halting problem, and prove a subrecursive analog of Rice's Theorem [16, 17]. We also prove a theorem relating to Baker, Gill, Solovay relativization results [3]. In Section 5, we show that all subrecursive indexings satisfying the axioms of Section 2 have the s-m-n property and therefore satisfy the Kleene Recursion Theorem [9]. We also prove a limited combinatorial completeness theorem. In Section 6 we explore the power of diagonalization. We show that separation by some form of diagonalization is always possible, but in many instances is as difficult as uniform simulation. In Section 7 we apply the results of the previous sections to a particular class of indexings, herein called the counter indexings of PTIME.

2. An axiomatization

Let $D$ be a set containing at least two elements, and let $\langle \cdot, \cdot \rangle$ be a pairing function on $D$ with projections $\pi_1, \pi_2$; that is, $\langle \cdot, \cdot \rangle$ is a one to one and onto function $D^2 \to D$ satisfying the identity $\langle \pi_1(x), \pi_2(x) \rangle = x$. For all practical purposes, we may take $D$ to be $\{0, 1\}^*$ or $\mathbb{N}$ and the pairing function to be any of the standard pairing functions in common use (see e.g. [13, 17]).

Elements of $D$ will be denoted $a, b, c, w, x, y, z$ and functions $D \to D$ will be denoted $e, f, g, h$. The pairing function forces $D$ to be infinite; we let $0, 1, 2, \ldots$ refer to a countable sequence of distinct elements of $D$.

The purpose of the pairing function is mainly to avoid unwieldy notation for functions of multiple arity. Accordingly, we will write $\langle x, y, z \rangle$ for $\langle x, \langle y, z \rangle \rangle$, $f(x, y)$ for $f(\langle x, y \rangle)$, $\lambda x y. [ \cdots x \cdots y \cdots ]$ for $\lambda z. [ \cdots \pi_1(z) \cdots \pi_2(z) \cdots ]$, etc. The pairing $\langle \cdot, \cdot \rangle$ on individuals of $D$ induces a pairing on functions $D \to D$, also denoted $\langle \cdot, \cdot \rangle$, by taking $\langle f, g \rangle = \lambda x. \langle f(x), g(x) \rangle$. 
**Definition 2.1.** Let $C$ be a class of functions $D \to D$. $C$ is closed provided

1. $\pi_1, \pi_2 \in C,$
2. each constant function $\kappa_x = \lambda y . x \in C,$
3. the conditional $\text{cond} \in C,$ where

$$\text{cond} = \lambda xyzw . \begin{cases} z, & \text{if } x = y, \\ w, & \text{otherwise,} \end{cases}$$

4. $C$ is closed under composition and pairing, i.e. if $f, g \in C$, then $f \circ g, \langle f, g \rangle \in C.$

In addition, $C$ is indexed provided there exists a universal function $U$ for $C$ (i.e. $C = \{ \lambda y . U(x, y) : x \in D \}$), such that (2.1.5)–(2.1.7) below hold. The function $\lambda y . U(x, y)$ is denoted $\phi_\ast$, and $x$ is said to be an index for $\phi_\ast$. Note that $U = \lambda xy . \phi_\ast(y)$.

5. there is a function $\text{const} \in C$ such that $\text{const}(x)$ is some index for constant function $\kappa_x$,
6. there is a function $\text{comp} \in C$ such that $\text{comp}(x, y)$ is some index for composition $\phi_x \circ \phi_y$,
7. there is a function $\text{pair} \in C$ such that $\text{pair}(x, y)$ is some index for the pair $\langle \phi_x, \phi_y \rangle$.

The indexing of $C$ is said to be computable if $D = \{0, 1\}^*$ or $\mathbb{N}$ and if $U$ is computable.

Many other authors (e.g. [1, 13, 14]) have considered these primitives in various contexts.

The following are examples of closed classes: PTIME, the class of functions computable on one-tape Turing machines in polynomial time; PSPACE, the class of functions computable on one-tape Turing machines in polynomial space, whose outputs are restricted in length to a polynomial in the length of the input; elementary functions; primitive recursive functions; total recursive functions.

All of the above closed classes have indexings. For example, PTIME can be indexed by Turing machines with polynomial time counters as described in Section 1 (strings that are not encodings of such machines index the zero function). The primitive recursive functions can be indexed by the LOOP programs of Meyer and Ritchie [15]. Elementary functions can be indexed by SR$_1$ programs of Constable and Borodin [5]. All of the above classes have computable indexings except the total recursive functions, for which no indexing is computable. We will not make use of the assumption of computability until Section 6.

Let $C$ be an indexed class of functions. We denote by $\Omega$ the smallest closed class of functions containing the index manipulating functions $\text{comp}, \text{const}, \text{pair}$. Note $\Omega \subseteq C$. Usually $\Omega$ is a very small class; for most indexings in common use, all functions of $\Omega$ are computable in logarithmic space.
3. Reducibility relations and indexings

Let \( C \) be a closed class of functions. In order to discuss the relative complexity of functions, we introduce two useful reducibility relations \( \leq_m^C \) and \( \leq^C \).

**Definition 3.1.** \( f \leq^C_m g \) if there exists an \( h \in C \) such that \( f = g \circ h \).

When restricted to recognition problems (0, 1-valued functions), \( \leq^C \) is just \( C \)-limited many-one reducibility (see [11, 17]).

**Definition 3.2.** \( f \leq^C g \) if \( f \) is contained in the smallest closed class containing \( g \) and \( C \). In other words, \( f \leq^C g \) if it follows inductively from the rules

\[
\begin{align*}
(3.2.1) & \quad g \leq^C g \text{ and } h \leq^C g \text{ for any } h \in C, \\
(3.2.2) & \quad \text{if } f \leq^C g \text{ and } h \leq^C g, \text{ then } \langle f, h \rangle \leq^C g \text{ and } f \circ h \leq^C g.
\end{align*}
\]

The reducibility \( \leq^C \) is best described as a bounded Turing reducibility. When restricted to decision problems (0, 1-valued functions), \( \leq^C \) coincides with the bounded truth-table reducibility of [11, 17]. This is proved in the next section (Lemma 4.9).

We write \( f \equiv^C g \) if \( f \equiv^C g \) and \( g \equiv^C f \). \( \equiv^C \) is defined similarly. The following properties of \( \leq^C_m \) and \( \leq^C_m \) are immediate from the definitions: \( \leq^C \) and \( \leq^C_m \) are transitive; \( \equiv^C_m \) and \( \equiv^C \) are equivalence relations; \( \leq^C_m \) refines \( \leq^C \) and \( \equiv^C_m \) refines \( \equiv^C \); the set of \( \equiv^C \)-classes forms an upper semilattice with bottom \( C \) and with the join operation given by \( \langle , \rangle \); if \( C \subseteq D \), then \( \leq^C_m \) refines \( \leq^D \) and \( \equiv^C_m \) refines \( \equiv^D \).

**Definition 3.3.** If \( C \) is closed and indexed, we write \( \leq \) for \( \leq^\Omega \), where \( \Omega \) is the smallest closed class containing the index manipulating functions \( \text{comp, const, pair} \). \( \leq_m, \equiv_m \), and \( \equiv_m \) are defined analogously.

Since \( \Omega \subseteq C \), \( \leq \) refines \( \leq^C \) and \( \leq_m \) refines \( \leq^C_m \). The notation \( \leq \) for \( \leq^\Omega \) is perhaps a bit misleading, since \( \Omega \) is not really independent of \( C \). However, it is justified on the grounds of conservation of notation, since virtually all \( \leq^C \) reductions we will exhibit will be \( \leq^\Omega \) reductions.

Mehlhorn [14] defined an abstract subrecursive reducibility to be an indexed set of operators

\[
\{\text{op}_i[\ ]|i \in \mathbb{N}\}
\]

satisfying a set of axioms analogous to those of the previous section. Most of the common subrecursive reducibilities give indexings by taking

\[
C = \{\text{op}_i[\lambda x . 0]|i \in \mathbb{N}\} \quad \text{and} \quad \phi_i = \text{op}_i[\lambda x . 0].
\]

Neither \( \leq^C \) nor \( \leq^C_m \) need be an abstract subrecursive reducibility in the strict sense of Mehlhorn, since he postulated a "rudimentary simulation property" in order to obtain
density results, which by our definition \( \leq^C \) and \( \leq_m^C \) need not satisfy. However, we have seen how some useful reducibilities are defined from an indexing of a subrecursive class, and how an indexing is obtained from a reducibility. Thus, in an intuitive sense at least, the notions of subrecursive reducibility and subrecursive indexing are complementary in the sense that either may be taken as primitive and the other defined.

4. Elementary results

In this section we prove some elementary consequences of the axioms of Section 2. In most cases the results have direct analogs in recursion theory and the proofs are conceptually similar (see e.g. [13, 17]).

Throughout this section, \( C \) will be an indexed class with universal function \( U = \lambda xy . \phi_x(y) \), and \( \Omega \subseteq C \) will be the smallest closed class containing the index manipulating functions \( \text{comp, const, pair} \). \( \Omega \) contains the identity \( \iota = (\pi_1, \pi_2) \). Also, the projection \( \pi^n_i \) which gives the \( i \)th element of an \( n \)-tuple \( \langle x_1, \ldots, x_n \rangle = \langle x_1, \langle x_2, \ldots, \langle x_{n-1}, x_n \rangle \cdots \rangle \rangle \) is in \( \Omega \), since it can be formed by composing \( \pi_1 \) and \( \pi_2 \) in the appropriate order. If \( f \in C \), we let \( \cdot f \) denote any index for \( x \).

4.1. Equivalence of uniform simulation, computing the main diagonal, and the halting problem

Definition 4.1. The graph of the universal function \( U \), denoted \( \text{gr } U \), is defined by

\[
\text{gr } U = \lambda xyz . \begin{cases} 
1, & \text{if } U(x, y) = z, \\
0, & \text{otherwise.}
\end{cases}
\]

The function \( \text{gr } U \) decides for \( x, y, z \) whether \( \phi_x(y) = z \), and therefore represents the minimum power necessary to simulate \( C \) uniformly.

Definition 4.2. The main diagonal is the function

\[
\text{diag} = \lambda x . \begin{cases} 
1, & \text{if } \phi_x(x) = 0, \\
0, & \text{otherwise.}
\end{cases}
\]

\( \text{diag} \notin C \) since it differs from each \( \phi_x \in C \) on at least one input, name \( y \) \( x \).

Definition 4.3. The membership function for \( x \) is

\[
\text{member}_x = \lambda y . \begin{cases} 
1, & \text{if } \phi_y(x) = 1, \\
0, & \text{otherwise.}
\end{cases}
\]

The function \( \text{member}_x \) asks uniformly whether \( x \) is a member of sets in \( C \). It may also be viewed as the uniform halting problem for \( C \) (see [13]).
It is not surprising that $\text{diag} \leq_m \text{gr} U$, since the ability to decide whether $\phi_x(y) = z$ for any $x, y, z$ surely gives enough power to decide whether $\phi_x(x) = 0$. Similarly, it is not surprising that $\text{member}_x \leq_m \text{gr} U$. However, it is a little surprising that the inequalities hold in the other direction.

**Theorem 4.4.** $\text{gr} U =_m \text{diag} =_m \text{member}_x$.

**Proof.** It is easy to show that $\text{diag} \leq_m \text{gr} U$ and $\text{member}_x \leq_m \text{gr} U$:

\[
\text{diag}(x) =\begin{cases} 1, & \text{if } \phi_x(x) = 0, \\ 0, & \text{otherwise} \end{cases}
\]

\[
= \text{gr} U(x, x, 0) = \text{gr} U \circ \langle \iota, \iota, \kappa_0 \rangle(x)
\]

so $\text{diag} = \text{gr} U \circ \langle \iota, \iota, \kappa_0 \rangle$. Similarly, $\text{member}_x = \text{gr} U \circ \langle \iota, \kappa_x, \kappa_1 \rangle$.

To show $\text{gr} U \leq_m \text{diag}$, construct $g \in \Omega$ so that

\[
\phi_{g(x,y,z)}(w) =\begin{cases} 0, & \text{if } \phi_x(y) = z, \\ 1, & \text{otherwise}. \end{cases}
\]

The function $g$ is given by

\[
\text{comp} \circ \langle \kappa_{-\text{cond}}, \text{pair} \circ \text{comp} \circ \langle \pi^3, \text{const} \circ \pi^3 \rangle, \\
\text{pair} \circ \text{const} \circ \pi^3, \text{pair} \circ \langle \kappa_{\text{const}(0)}, \kappa_{\text{const}(1)} \rangle \rangle).
\]

Then

\[
\text{diag}(g(x, y, z)) =\begin{cases} 1, & \text{if } \phi_{g(x,y,z)}(g(x, y, z)) = 0, \\ 0, & \text{otherwise} \end{cases}
\]

\[
=\begin{cases} 1, & \text{if } \phi_x(y) = z \\ 0, & \text{otherwise} \end{cases}
\]

\[
= \text{gr} U(x, y, z),
\]

so $\text{gr} U = \text{diag} \circ g$. The proof that $\text{gr} U \leq_m \text{member}_x$ is similar.

Since $f \in C$ and $g \leq^C f$ imply $g \in C$, and since $\text{diag} \not\in C$, it follows that $\text{member}_x \not\in C$ and $\text{gr} U \not\in C$ as well.

4.2. **Rice's theorem for indexed subrecursive classes**

**Definition 4.5.** A nontrivial property $P$ on $C$ is a function $P : D \to \{0, 1\}$ such that

1. there exist $a, b$ such that $P(a) \neq P(b)$ ($P$ is nontrivial),
2. $\phi_x = \phi_y \to P(x) = P(y)$ ($P$ is a property of functions, not just of indices).

Rice's theorem [16] (see also [13, 17]) states that every nontrivial property of the r.e. sets is undecidable. This can be proved by giving a many-one reduction of the halting
problem to the nontrivial property. The direct subrecursive analog of this theorem holds as well: every nontrivial property of $C$ is at least as hard to compute as the graph of the universal function. For example, halting (membership) is a nontrivial property of $C$; together, Theorems 4.4 and 4.6 say that there is no nontrivial property on $C$ that is any easier to compute than halting. Theorem 4.6 says that $gr\ U$ represents an important threshold, namely the minimum power needed to answer nontrivial questions about $C$ in a uniform way.

**Theorem 4.6.** If $P$ is a nontrivial property on indexed class $C$, then $gr\ U \leq_m P$.

**Proof.** Suppose $a, b$ are such that $P(a) = 1$ and $P(b) = 0$. There is a $g \in \Omega$ such that

$$
\phi_{g(x,y,z)} = \begin{cases} 
\phi_a, & \text{if } \phi_x(y) = z, \\
\phi_b, & \text{otherwise.}
\end{cases}
$$

The function $g$ is given by

$$
\text{comp} \circ (\kappa_{\text{cond}}, \text{pair} \circ (\text{comp} \circ (\pi_1^3, \text{const} \circ \pi_2^3)), \text{pair} \circ (\text{const} \circ \pi_3^3, \text{pair} \circ (\kappa_a, \kappa_b))).
$$

Then

$$
P(g(x,y,z)) = \begin{cases} 
P(a), & \text{if } \phi_x(y) = z, \\
P(b), & \text{otherwise}
\end{cases}
$$

$$
= gr\ U(x, y, z),
$$

so $gr\ U = P \circ g$.

### 4.3. Bounded truth table reducibility, functions vs. decision problems, and relativization

The universal function $U$ on the surface appears more powerful than its graph $gr\ U$, since it can compute the output of any function in $C$ on any input, whereas $gr\ U$ can only answer 'yes' or 'no' to a proposed output. This raises the question: in general, is the ability to compute functions strictly more powerful than the ability to answer corresponding decision problems? In this context, the answer is affirmative, as Theorem 4.11 below shows. The key to the argument is that, on decision problems (0, 1-valued functions), the $\leq_C$ reducibility is no more powerful than bounded truth table reducibility [11, 17].

**Definition 4.7.** $C$-limited bounded truth table reducibility: We write $f \leq_{\text{bt}}^C h$ if there is a nonnegative integer $k$, a function $g \in C$ called the $k$-tt condition generator, and a function $e \in C$ called the $k$-tt condition evaluator, such that

$$
f = e \circ (\epsilon, \langle h \circ \pi_1^k, \ldots, h \circ \pi_n^k \rangle \circ g).
$$

In other words, to compute $f(x)$, $g$ is applied to $x$ to produce a $k$-tt condition or
$k$-tuple of queries $\langle x_1, \ldots, x_k \rangle$, then $h$ is applied to each of these queries to get $\langle h(x_1), \ldots, h(x_k) \rangle$, and then $e$ is applied to $x$ and $\langle h(x_1), \ldots, h(x_k) \rangle$.

In case $f$ and $h$ are 0, 1-valued, this definition coincides with the definition of bounded truth table reducibility of [11, 17].

**Definition 4.8.** A function is **bounded** if it takes on only finitely many values.

**Lemma 4.9.** If $f \leq^C h$ and $h$ is bounded, then $f \leq^\text{bit}_\text{bit}^C h$.

**Proof.** The proof is by induction on the length of a derivation via the rules (3.2.1) and (3.2.2) that $f \leq^C h$. Trivially, $h \leq^\text{bit}_\text{bit}^C h$ and $g \leq^\text{bit}_\text{bit}^C h$ for all $g \in C$, so it remains to show that if $f \leq^\text{bit}_\text{bit}^C h$ and $f' \leq^\text{bit}_\text{bit}^C h$, then $\langle f', f \rangle \leq^\text{bit}_\text{bit}^C h$ and $f' \circ f \leq^\text{bit}_\text{bit}^C h$. We prove that $f' \circ f \leq^\text{bit}_\text{bit}^C h$; the proof that $\langle f', f \rangle \leq^\text{bit}_\text{bit}^C h$ is similar and is left to the reader.

If $f, f' \leq^\text{bit}_\text{bit}^C h$, then there exist $k$ and $k'$, generators $g$ and $g'$, and evaluators $e$ and $e'$ such that

$$f = e \circ \langle \iota, \langle h \circ \pi^k_1, \ldots, h \circ \pi^k_k \rangle \circ g \rangle$$

and

$$f' = e' \circ \langle h \circ \pi^{k'}_1, \ldots, h \circ \pi^{k'}_k \rangle \circ g' \rangle.$$  

Thus

$$f' \circ f = e' \circ \langle \iota, \langle h \circ \pi^k_1, \ldots, h \circ \pi^k_k \rangle \circ g' \rangle \circ e \circ \langle \iota, \langle h \circ \pi^{k'}_1, \ldots, h \circ \pi^{k'}_k \rangle \circ g \rangle.$$  

Suppose $h$ is bounded. Let $M$ be the finite set of possible values of $h$, and let $m$ be the cardinality of $M$. Let $g''$ be the $(k'm^k + k)$-condition generator which on input $x$ generates all $m^k$ possible $k'$-ut conditions $g' \circ e(x, x_1, \ldots, x_k)$, one for each $\langle x_1, \ldots, x_k \rangle \in M^k$, and the $k$-vector $g(x)$, and places them end to end to get a $(k'm^k + k)$-condition. After $h$ is applied to all elements of this vector, the $(k'm^k + k)$-condition evaluator $e''$ will check the resulting vector to determine which $\langle x_1, \ldots, x_k \rangle \in M^k$ satisfies

$$\langle x_1, \ldots, x_k \rangle = \langle h \circ \pi^k_1, \ldots, h \circ \pi^k_k \rangle \circ g(x),$$

find the appropriate

$$\langle h \circ \pi^{k'}_1, \ldots, h \circ \pi^{k'}_k \rangle \circ g' \circ e(x, x_1, \ldots, x_k)$$

in the $(k'm^k + k)$-vector, and apply $e'$ to it. $e''$ can be computed using a finite number of nested conditionals.

**Lemma 4.10.** If $f \leq^C h$ and $h$ is bounded, then there is a bounded $p \in C$ such that for all $x$, $p(x) \neq f(x)$.

**Proof.** By the above lemma, $f \leq^\text{bit}_\text{bit}^C h$, so there exist $g$, $e$ such that

$$f = e \circ \langle \iota, \langle h \circ \pi^k_1, \ldots, h \circ \pi^k_k \rangle \circ g \rangle.$$
If $M$ is the finite set of possible values of $h$, and if $m$ is its cardinality, then for all $x$,  
$$f(x) \in \{e(x, x_1, \ldots, x_k) | x_i \in M, 1 \leq i \leq k\}.$$  

On input $x, p$ finds the least of $0, 1, \ldots, m^k$ not among $\{e(x, x_1, \ldots, x_k) | x_i \in M, 1 \leq i \leq k\}$, using a finite number of nested conditionals, and takes that as its value.

**Theorem 4.11.** $U \not\leq^C h$ for any bounded $h$. In particular, $U \not\leq^C \text{ gr } U$.

**Proof.** If $U \leq^C h$, then the function $\lambda x . \phi_x(x) = U \circ \langle \iota, \iota \rangle \leq^C h$. But $\lambda x . \phi_x(x)$ agrees with every function in $C$ on at least one input, contradicting the previous lemma.

The bounded nature of the $\leq^C$ reducibility relation, as captured in Lemmas 4.9 and 4.10, also impacts Baker, Gill, Solovay relativization results [3].

**Definition 4.12.** Let $C^f$ denote the class $C$ relativized to $f$, i.e.

$$C^f = \{g | g \leq^C f\}.$$  

In light of Lemma 4.9, if $f$ and $g$ are 0, 1-valued, then $g \in C^f$ means that $g$ can be computed by a function in $C$ with the help of oracle $f$, with the restriction that the oracle may be consulted at most $k$ times on any input, for some $k$. This restriction makes our notion of relativization somewhat weaker than that of [3]; the difference is analogous to the difference between bounded truth table and Turing reducibility.

The following theorem says that if $C$ and $D$ are closed classes of functions, $C \subseteq D$, and if $C$ and $D$ can be made equal by relativizing with respect to a bounded oracle $f$, then $D$ must be very close to $C$, in the sense that $D$ is too small to contain the universal function of any indexing of $C$.

Let $C, D$ be closed, $C \subseteq D$. Clearly $C^f \subseteq D^f$ for any $f$.

**Theorem 4.13.** If $C^f = D^f$ for some bounded $f$, then $D$ does not contain the universal function of any indexing of $C$.

**Proof.** Let $U$ be the universal function of some indexing of $C$. Since $D \subseteq D^f \subseteq C^f$, if $U \in D$, then $U \in C^f$, contradicting Theorem 4.11.

Recall $\text{PTIME} = \{\text{functions computable by Turing machines in polynomial time}\}$, $\text{PSPACE} = \{\text{functions computable in polynomial space with outputs restricted in length to a polynomial in the length of the input}\}$. The above theorem says that if $\text{PTIME}^f = \text{PSPACE}^f$ for some bounded $f$, then $\text{PSPACE}$ does not contain the universal function of any indexing of $\text{PTIME}$. We have not been able to find such an $f$, but it is worth noting that

1. there is an unbounded $f$ with $\text{PTIME}^f = \text{PSPACE}^f$; and
2. there is a 0, 1-valued function $f$ such that the classes $\text{PTIME}$ and $\text{PSPACE}$ relativized to $f$ in the sense of [3] are equal.
5. The combinatorial power of subrecursive indexings

In this section we attempt to characterize the combinatorial power of subrecursive indexings (by combinatorial power we mean the power to manipulate and combine programs). Although this power is much weaker than that of the partial recursive functions, many interesting and useful combinatorial functions which appear in recursion theory still exist in a subrecursive environment.

5.1. The s-m-n theorem and the recursion theorem

Gödel numberings of the partial recursive functions are often axiomatized by two properties: the s-m-n property and the universal function property. In this section we show that any subrecursive indexing has the s-m-n property. The universal function property does not hold in subrecursive indexings, since \( U \notin C \), as previously noted. Likewise, the fixed-point version of the Recursion Theorem (see [13, 17]) does not hold, but we show that \( C \) satisfies the Kleene Recursion Theorem [9]. Both these observations were implicit in previous work: for example, although they were working with Gödel numberings of the partial recursive functions, Machtey and Young's [13] construction of s-m-n functions depended on effective composition but did not make use of the universal function property; and Alton [1] observed that effective composition and the s-m-n property imply the Kleene Recursion Theorem.

Let \( C \) be an indexed class.

**Theorem 5.1 (s-m-n Theorem).** For all natural \( m, n \) there is an \( s = s'' \in \Omega \) such that

\[
\phi_s(x_1, \ldots, x_n, y_1, \ldots, y_m) = \phi_{s'(x_1, \ldots, x_n)}(y_1, \ldots, y_m).
\]

**Proof.** Take

\[
s = \text{comp} \circ (\pi_1^{n+1}, \text{pair} \circ (\text{const} \circ \pi_2^{n+1},
\text{pair} \circ (\text{const} \circ \pi_3^{n+1}, \ldots, (\text{const} \circ \pi_{n+1}^{n+1}, k, \ldots)))).
\]

Then \( s \in \Omega \) and a straightforward calculation shows that \( s \) has the desired properties.

The following theorem is the Kleene Recursion Theorem [9], the original form of the theorem. The more familiar fixed-point form (see [13, 17]) does not hold in any indexed subrecursive class. For example, there is a function \( g \in \Omega \) satisfying

\[
\phi_{g(x)}(y) = \begin{cases} 1, & \text{if } \phi_x(y) = 0, \\ 0, & \text{otherwise}, \end{cases}
\]

but \( g \) has no fixed point. Nevertheless, the original form is strong enough for many applications (see [2, 9]).

**Theorem 5.2 (Kleene Recursion Theorem).** There is an \( f \in \Omega \) such that for any \( x, y \),

\[
\phi_{f(x)}(y) = \phi_x(f(x), y).
\]
Proof. The proof given here is essentially an abstraction of Kleene's original proof [9] (see also [2, p. 34] and [13, p. 176]).

Let

\[ g = \text{comp} \circ (\text{pair} \circ (\text{comp} \circ (\kappa \circ s_1, \text{pair} \circ (\kappa \circ \pi_1, \kappa \circ \pi_2)), \kappa \circ \pi_2)). \]

Then \( g \in \Omega \) and

\[ \phi_{g(x)}(y, z) = \phi_x(s_1(y, y), z). \]

Let \( f = s_1 \circ (g, g) \). Then \( f \in \Omega \) and a straightforward calculation shows that \( f \) is the desired function.

5.2. Combinatorial completeness

In the following discussion, a term will be something of the form

\[ \phi_{\phi_x(x_0)}(\phi_{\phi_y(y)(x_1)})(\phi_{x_0}(b)) \]

for example, where \( x_0, x_1 \) are variables ranging over \( D \) and \( a, b \) are constants in \( D \). Terms give an easy way of specifying functions: the term \( t[x_1, \ldots, x_n] \) with variables \( x_1, \ldots, x_n \) represents the function \( \lambda x_1 \ldots x_n \cdot t[x_1, \ldots, x_n] \).

In recursive function theory, all functions specified by terms have indices; i.e. for any term \( t[x_1, \ldots, x_n] \) (where the \( x_i \) range over Gödel numbers of partial recursive functions), the function \( \lambda x_1, \ldots, x_n \cdot t[x_1, \ldots, x_n] \) is itself partial recursive. This phenomenon is common to several formal systems computationally equivalent to the partial recursive functions, such as the URS's of Wagner [20], BRFT's of Strong [18, 19], the \( \lambda \)-calculus, and combinator logic[6]. In the latter, the property is known as 'combinatorial completeness'. This property may be viewed as good evidence for Church's Thesis, which states that if a function is intuitively computable, i.e. if it is specified by a set of intuitively executable instructions, then it is a partial recursive function.

In an indexed subrecursive class \( C \), we do not have full combinatorial completeness, since functions specified by terms are not necessarily in \( C \). For example, the term \( \phi_x(y) \) specifies the universal function \( U \). It is not hard to show, however, that a function is specified by a term if and only if it is \( \leq^C \)-reducible to \( U \).

In the following theorem, we show that any function represented by a term is represented by a term of a special form, called an iterative term. This result has both a theoretical and practical significance. Its practical significance is that several earlier constructions, such as those of Theorems 4.4 and 4.6, become trivial applications; its theoretical significance is that it characterizes the power of subrecursive indexings in the same way that combinatorial completeness characterizes the power of the partial recursive functions.

Let \( C \) be an indexed class with \( U, \Omega \) defined as above.

Definition 5.5. Let \( L \) be a language with a constant \( a \) for each \( a \in D \), a constant \( f \) for
each \( f \in \Omega \), and individual variables \( x_0, x_1, \ldots \) ranging over \( D \). Terms are defined inductively:

(i) all constants and variables are terms,
(ii) if \( s, t \) are terms, then so are \( \langle s, t \rangle, \phi_s(t) \) and \( f(t) \).

For example,

\[
\phi_{\text{compl}(x_1, x_2)}(\phi_{\text{def}(x_1)}((x_1, x_2)))
\]

is a term. We write \( t[x_1, \ldots, x_n] \) to indicate that all variables of \( t \) are among \( x_1, \ldots, x_n \).

Let \( L^p \) be \( L \) with the addition of a single function variable \( p \) ranging over \( \Omega \). Iterative terms are terms in \( L^p \) of a special form. Let \( d_1, d_2, \ldots \) denote either constants \( a \in D \) or variables \( x_1, x_2, \ldots \).

Definition 5.4. Iterative terms, denoted \( \sigma, \tau \), are defined inductively:

(i) \( p(d_1, \ldots, d_m) \) is an iterative term,
(ii) if \( \sigma \) is an iterative term, then \( \phi_\sigma(d_1, \ldots, d_k) \) is.

Iterative terms represent functions which can be computed in a special way: a function in \( \Omega \) is applied to part of the input; the result is interpreted as an index of a function in \( C \), and this function is applied to part of the input; etc., some finite number of times. Theorem 5.5 below says that all functions represented by terms can be computed in this way. Moreover, there is a great deal of leeway in the choice of the sequence of \( d_i \)s appearing at each level.

If symbol \( d \) occurs in term \( t \), we define the depth of \( d \) in \( t \), denoted \( \delta(d, t) \), to be the depth of the most deeply nested subscript of \( t \) in which \( d \) occurs. For example, \( \delta(x, \phi_a(x)) = 0 \) and \( \delta(x, \phi_{\phi(a)}((x, y))) = 2 \). If \( d \) does not occur in \( t \), we define \( \delta(d, t) = -1 \).

If \( t \) is a term and \( \sigma \) is an iterative term, we write \( \sigma \geq t \) if all variables occurring in \( t \) also occur in \( \sigma \), and occur at least as deep, i.e.

\[
\delta(x_n, \sigma) \geq \delta(x_n, t) \quad \text{for all } x_n,
\]

and \( p \) occurs in \( \sigma \) as deep as any symbol of \( t \), i.e.

\[
\delta(p, \sigma) \geq \delta(d, t) \quad \text{and} \quad \delta(p, \sigma) \geq \delta(f, t)
\]

for any function symbol \( f \) or constant or variable \( d \).

Theorem 5.5. Let \( s[x_1, \ldots, x_n] \) be any (non-iterative) term and \( \sigma[p, x_1, \ldots, x_n] \) any iterative term with \( \sigma \geq s \). Then there exists a \( g \in \Omega \) such that

\[
\lambda x_1, \ldots, x_n . \sigma[g, x_1, \ldots, x_n] = \lambda x_1, \ldots, x_n . s[x_1, \ldots, x_n].
\]
Remark. In the proof of Theorem 4.6 we had to construct a \( g \in \Omega \) such that for all \( x, y, z, w \),

\[
\phi_{g(x,y,z)}(w) = \begin{cases} 
\phi_a(w), & \text{if } \phi_x(y) = z, \\
\phi_b(w), & \text{otherwise}
\end{cases}
\]

\[
= \text{cond}(\phi_x(y), z, \phi_a(w), \phi_b(w)).
\]

Since \( \phi_{p(x,y,z)}(w) \) is an iterative term and

\[
\phi_{p(x,y,z)} \equiv \text{cond}(\phi_x(y), z, \phi_a(w), \phi_b(w)),
\]

Theorem 5.5 immediately guarantees such a \( g \).

Proof of Theorem 5.5. The proof is by induction on the structure of \( s \) and \( \sigma \), as given in Definitions 5.3 and 5.4.

For the basis, assume \( \sigma[p, x_1, \ldots, x_n] \) is of the form \( p(d_1, \ldots, d_m) \). We want \( g \in \Omega \) such that for all \( x_1, \ldots, x_n \),

\[
g(d_1, \ldots, d_m) = s[x_1, \ldots, x_n].
\]

If \( s[x_1, \ldots, x_n] = a \), take \( g = \kappa_a \). If \( s[x_1, \ldots, x_n] = x_i \), then by depth requirements \( x_i \) must occur among \( d_1, \ldots, d_m \), so take \( g \) to be the appropriate projection. If \( s[x_1, \ldots, x_n] = (t[x_1, \ldots, x_n], u[x_1, \ldots, x_n]) \), then \( \sigma \supseteq t \) and \( \sigma \supseteq u \), so by the induction hypothesis there exist \( f, h \in \Omega \) such that for all \( x_1, \ldots, x_n \),

\[
f(d_1, \ldots, d_m) = t[x_1, \ldots, x_n] \quad \text{and} \quad h(d_1, \ldots, d_m) = u[x_1, \ldots, x_n],
\]

so take \( g = (f, h) \).

If \( s[x_1, \ldots, x_n] = f(t[x_1, \ldots, x_n]) \), then \( \sigma \supseteq t \), so by the induction hypothesis there is an \( h \in \Omega \) such that for all \( x_1, \ldots, x_n \),

\[
h(d_1, \ldots, d_m) = t[x_1, \ldots, x_n],
\]

and we may take \( g = f \circ h \).

The term \( s \) may not be of the form \( \phi_t(u) \), or else depth requirements are violated. This establishes the basis.

Now suppose \( \sigma \) is of the form

\[
\phi_{\tau[p_1, x_1, \ldots, x_n]}(d_1, \ldots, d_m).
\]

We want a \( g \) such that for all \( x_1, \ldots, x_n \),

\[
\phi_{\tau[g \circ x_1, \ldots, x_n]}(d_1, \ldots, d_m) = s[x_1, \ldots, x_n].
\]

If \( s[x_1, \ldots, x_n] = a \), then by the induction hypothesis there exists a \( g \in \Omega \) such that for all \( x_1, \ldots, x_n \),

\[
\tau[g, x_1, \ldots, x_n] = \text{const}(a).
\]
Then
\[ \phi_{r[\xi_1, \ldots, \xi_n]}(d_1, \ldots, d_m) = \phi_{\text{const}(\alpha)}(d_1, \ldots, d_m) = a. \]

If \( s[x_1, \ldots, x_n] = x_i \), and if \( x_i \) occurs among \( d_1, \ldots, d_m \), then by the induction hypothesis there exists a \( g \in \Omega \) such that \( \tau[g, x_1, \ldots, x_n] \) is an index for the appropriate projection. If \( x_i \) does not occur among \( d_1, \ldots, d_m \), then it must occur in \( \tau \), by depth requirements. But
\[ \tau[p, x_1, \ldots, x_n] \supseteq \text{const}(x_i), \]
so by the induction hypothesis there is a \( g \) such that for all \( x_1, \ldots, x_n \),
\[ \tau[g, x_1, \ldots, x_n] = \text{const}(x_i), \]
therefore
\[ \phi_{r[\xi_1, \ldots, \xi_n]}(d_1, \ldots, d_m) = \phi_{\text{const}(\xi)}(d_1, \ldots, d_m) = x_i. \]

If \( s[x_1, \ldots, x_n] = \langle u[x_1, \ldots, x_n], v[x_1, \ldots, x_n] \rangle \), then \( \sigma \supseteq u \) and \( \sigma \supseteq v \), so by the induction hypothesis there exist \( f, h \in \Omega \) such that for all \( x_1, \ldots, x_n \),
\[ \phi_{r[f, x_1, \ldots, x_n]}(d_1, \ldots, d_m) = u[x_1, \ldots, x_n] \]
and
\[ \phi_{r[h, x_1, \ldots, x_n]}(d_1, \ldots, d_m) = v[x_1, \ldots, x_n], \]
thus
\[ \langle \phi_{r[f, x_1, \ldots, x_n]}, \phi_{r[h, x_1, \ldots, x_n]} \rangle(d_1, \ldots, d_m) = s[x_1, \ldots, x_n] \]
and so
\[ \phi_{\text{pair}(r[f, x_1, \ldots, x_n], r[h, x_1, \ldots, x_n])}(d_1, \ldots, d_m) = s[x_1, \ldots, x_n]. \]

But
\[ \tau[p, x_1, \ldots, x_n] \supseteq \text{pair}(\tau[f, x_1, \ldots, x_n], \tau[h, x_1, \ldots, x_n]), \]
so by the induction hypothesis there exists a \( g \in \Omega \) such that for all \( x_1, \ldots, x_n \),
\[ \tau[g, x_1, \ldots, x_n] = \text{pair}(\tau[f, x_1, \ldots, x_n], \tau[h, x_1, \ldots, x_n]). \]

If
\[ s[x_1, \ldots, x_n] = \phi_{u[x_1, \ldots, x_n]}(v[x_1, \ldots, x_n]), \]
then
\[ \tau[p, x_1, \ldots, x_n] \supseteq u[x_1, \ldots, x_n] \quad \text{and} \quad \phi_{r[p, x_1, \ldots, x_n]}(d_1, \ldots, d_m) \supseteq v[x_1, \ldots, x_n], \]
so by the induction hypothesis there exist \( f, h \in \Omega \) such that for all \( x_1, \ldots, x_n \),
\[ \phi_{r[f, x_1, \ldots, x_n]}(d_1, \ldots, d_m) = v[x_1, \ldots, x_n] \quad \text{and} \quad \tau[h, x_1, \ldots, x_n] = u[x_1, \ldots, x_n], \]
or in other words
\[ \phi_{\tau[h, x_1, \ldots, x_n]}(\phi_{\tau[f, x_1, \ldots, x_n]}(d_1, \ldots, d_m)) = s[x_1, \ldots, x_n]. \]
Then
\[ \phi_{\text{comp}(\tau[h, x_1, \ldots, x_n], \tau[f, x_1, \ldots, x_n])}(d_1, \ldots, d_m) = s[x_1, \ldots, x_n], \]
By the induction hypothesis there exists \( g \in \Omega \) such that for all \( x_1, \ldots, x_n \),
\[ \tau[g, x_1, \ldots, x_n] = \text{comp}(\tau[h, x_1, \ldots, x_n], \tau[f, x_1, \ldots, x_n]) \]
so
\[ \phi_{\tau[g, x_1, \ldots, x_n]}(d_1, \ldots, d_m) = s[x_1, \ldots, x_n]. \]
Finally, the case \( s[x_1, \ldots, x_n] = f(t[x_1, \ldots, x_n]) \) is a special case of the previous case, since \( f(t[x_1, \ldots, x_n]) = \phi_f(t[x_1, \ldots, x_n]) \), where \( \phi_f \) denotes some index for \( f \).

The theory of subrecursive indexings can be developed within the framework of combinator logic and \( \lambda \)-calculus [6]. In this framework, the theory would take the form of a restricted combinator logic built on the primitives \( B \) and \( K \), where \( B \) is the composition combinator \( B_{xyz} = x(yz) \) and \( K \) is the constant combinator \( K_{xy} = x \). It is not hard to show that the equational theory with these primitives is decidable, thus much weaker than full combinator logic. Another analogous formulation, modeled after the URS's of Wagner [20], would use primitives similar to \( B \) and \( K \) over a domain without \( * \). Unfortunately, formal systems like combinator logic and URS’s customarily avoid functions of multiple arity by using iterated application. This convenient if somewhat artificial device entails no loss of generality in the presence of a universal function, but in the subrecursive case it is unacceptable; for example, the identity function \( \epsilon \) would become the universal function. Thus in order to develop a subrecursive combinator logic, a pairing function or some other means of handling multiple arity would need to be introduced.

6. Diagonalization

For the remainder of the paper, \( C \) is an indexed class over \( \{0, 1\}^* \) or \( \mathbb{N} \) with computable universal function \( U \). It should be noted that this is the first we have used the assumption of computability.

In this section we explore the power of diagonalization, a well-known technique for proving separation of complexity classes. The simplest example of a diagonal function is the main diagonal of Section 4,
\[ \text{diag} = \lambda x. \begin{cases} 1, & \text{if } \phi_x(x) = 0, \\ 0, & \text{otherwise.} \end{cases} \]
As shown in Theorem 4.4, diag embodies all the power necessary to simulate functions in \( C \) uniformly.
In general, the process of diagonalization over an indexed class \( C \) consists of constructing a single function \( g \) which differs from each function in \( C \) on at least one input. If \( g \) can be shown to belong to some class \( D \), then this suffices to show that \( C \) and \( D \) are different classes. Usually, \( g \) is constructed so that on input \( x \), it computes the description \( h(x) \) of some function \( \phi_{h(x)} \in C \), computes \( \phi_{h(x)}(x) \) by direct simulation, then does something different. This motivates the following definition:

**Definition 6.1.** A diagonal is a function \( \text{diag}_h \) of the form

\[
\text{diag}_h = \lambda x . \begin{cases} 1, & \text{if } \phi_{h(x)}(x) = 0, \\ 0, & \text{otherwise}, \end{cases}
\]

where \( h \) is a computable function satisfying either of conditions (6.1.1), (6.1.2) below. The purpose of conditions (6.1.1) or (6.1.2) is to guarantee that each function in \( C \) is simulated at some point, thus insuring that \( \text{diag}_h \notin C \).

(6.1.1) \( h \) is functionally onto i.e. for all \( f \in C \) there exists an \( x \) such that \( h(x) \) is an index for \( f \).

Alternatively we might require

(6.1.2) \( h \) is a.e. functionally onto i.e. for all \( f \in C \) there is an \( f' \in C \) such that \( f' = f \) a.e. and for infinitely many \( x \), \( h(x) \) is an index for \( f' \).

Clearly either (6.1.1) or (6.1.2) insures \( \text{diag}_h \notin C \). Note that diag = diag\(_\varepsilon\), where \( \varepsilon \) is the identity.

Virtually all separation results in the literature are proved by diagonalization in one form or another. However, the technique has failed in the most intriguing cases. For example, functions constructed to diagonalize over \( P \) always seem to require more than polynomial space to compute; consequently, the \( P \neq NP \) question remains open. It has been argued [7, 11] that \( P \) and \( NP \), although they may be different, are nevertheless too close together to admit separation by diagonalization. The justification for this standpoint is that known diagonalization arguments relativize; if every diagonalization proof were to relativize, then \( P \neq NP \) would not be provable by diagonalization, by results of [3]. Nevertheless, the following result shows that if \( P \neq NP \) is provable at all, then it is provable by diagonalization; therefore, there must exist diagonalizations which do not relativize.

**Theorem 6.2.** Any 0, 1-valued computable function not in \( C \) is a diagonal. I.e., if \( g \notin C \), and if \( g \) is 0, 1-valued and computable, then there exists a computable \( h \) such that \( g = \text{diag}_h \). Moreover, \( h \) can be chosen to satisfy both (6.1.1) and (6.1.2).

**Remark.** The proof of Theorem 6.2 is constructive in the sense that \( h \) is constructed explicitly from \( g \). This says that there is an effective method for going from proofs of \( 'P \neq NP' \) to diagonalization proofs of \( 'P \neq NP' \) (i.e. those in which a specific diagonal function in \( NP - P \) is exhibited). It does not say that if \( P \neq NP \), then the statement \( 'P \neq NP' \) is provable by diagonalization; it may still be the case that \( P \neq NP \), yet \( 'P \neq NP' \) is not provable at all.
Proof of Theorem 6.2. Let \( g \not\in C \) be computable and 0, 1-valued. We first construct a computable witness function for \( g \not\in C \), i.e. a function \( f \) such that for every \( x \), \( f(x) \) gives an input on which \( g \) and \( \phi_x \) differ. We will actually construct \( f \) so that

\[
\text{for all } x, \ g(f(x)) = 0 \text{ iff } \phi_x(f(x)) \neq 0,
\]

\[
f \text{ is strictly increasing.}
\]

To do this, first note that for all \( \phi_x \) there are infinitely many \( y \) such that

\[
g(y) = 0 \text{ iff } \phi_x(y) \neq 0,
\]

for if not, then for some \( \phi_x \) and all but finitely many \( y \),

\[
g(y) = \begin{cases} 0, & \text{if } \phi_x(y) = 0, \\ 1, & \text{otherwise}, \end{cases}
\]

thus \( g \not\in C \) using nested conditionals, contradicting the assumption that \( g \not\in C \). Then \( f(x) \) is computed by looking for the first \( y \) greater than \( f(x) - 1 \) satisfying (6.5). \( f \) is computable since \( g \) and \( U \) are.

Now that we have \( f \), construct \( h \) to be a functional inverse of \( f \). For input \( y \), if there exists an \( x \) with \( f(x) = y \), take \( h(y) = x \). Whether such an \( x \) exists is decidable, since \( f \) is computable and strictly increasing. Otherwise, take

\[
h(y) = \begin{cases} \text{const}(0), & \text{if } g(y) = 1, \\ \text{const}(1), & \text{otherwise}. \end{cases}
\]

Then \( h \) is computable. We claim that \( g = \text{diag}_h \). If there is an \( x \) such that \( f(x) = y \), then \( h(y) = x \), and

\[
g(y) = g(f(x))
\]

\[
= \begin{cases} 1, & \text{if } \phi_x(f(x)) = 0, \\ 0, & \text{otherwise} \end{cases} = \begin{cases} 1, & \text{if } \phi_{h(y)}(y) = 0, \\ 0, & \text{otherwise}. \end{cases}
\]

If \( y \) is not in the range of \( f \), then

\[
g(y) = \begin{cases} 1, & \text{if } h(y) = \text{const}(0), \\ 0, & \text{if } h(y) = \text{const}(1) \end{cases}
\]

It remains to show that \( h \) satisfies (6.1.1) and (6.1.2). If clearly satisfies (6.1.1), since it is onto. To show that it satisfies (6.1.2) as well, it suffices to show that every function in \( C \) has infinitely many indices. But if \( f \in C \) had only finitely many indices, then there would be a function in \( C \) to decide whether \( \phi_x = f \), a nontrivial property of \( C \), contradicting Theorem 4.6.

Thus we can say nothing more about whether diagonalization proofs exist – they always do, if any proof does. However, we can say something about their complexity. Recall from the foregoing proof that a witness function for \( g \not\in C \) is a function \( f \) such that \( g(f(x)) \neq \phi_x(f(x)) \) for any \( x \). Witness functions are intimately associated with
diagonals; in fact, if \( f \) is any right inverse of \( h \) (i.e. if \( h \circ f = \iota \)), then \( f \) is a witness for \( \text{diag}_x \notin C \). The following theorem may be interpreted as a tradeoff between the complexity of a function \( g \notin C \) and any witness \( f \) of \( g \notin C \); that is, the closer \( g \) is to \( C \) in complexity, the more complex \( f \) must be.

**Theorem 6.6.** If \( g \) is \( 0, 1 \)-valued and not is \( C \), and if \( f \) is any witness function for \( g \notin C \), then \( \text{gr } U \leq (g, f) \).

**Proof.** Let \( e \in \Omega \) such that

\[
\phi_{e(x, y, z)}(w) = \begin{cases} 
0, & \text{if } \phi_x(y) = z, \\
1, & \text{otherwise}.
\end{cases}
\]

The function \( e \) exists by Theorem 5.5. A straightforward calculation shows that

\[ g(f(e(x, y, z))) = \text{gr } U(x, y, z), \]

so \( \text{gr } U = g \circ f \circ e \).

There are many restricted forms of diagonalization we might consider. The most straightforward diagonalization arguments use what we call *simple* diagonals.

**Definition 6.7.** A diagonal \( \text{diag}_h \) is *simple* provided \( h \) is an operator, i.e.

\[
\text{if } \phi_x = \phi_y, \text{ then } \phi_{h(x)} = \phi_{h(y)}. \quad (6.7.1)
\]

Thus if two indices \( x, y \) represent the same function, the two programs that \( \text{diag}_h \) simulates on inputs \( x \) and \( y \) compute the same function. A simple diagonal is just that – advanced diagonalization techniques like priority arguments are excluded. Nevertheless, many elementary separation results (e.g. \( P \neq \text{DTIME}(2^n) \), \( \text{DSpace}(n) \neq \text{DSpace}(n^2) \)) can be proved using simple diagonalizations.

**Theorem 6.8.** Any simple diagonal is as hard to compute as the graph of the universal function: i.e. if \( \text{diag}_h \) is simple, then \( \text{gr } U \leq_m \text{diag}_h \).

**Proof.** Both (6.1.1), (6.1.2) imply

\[
\text{for all } f \in C \text{ there exists an } f' = f \text{ a.e. and an } x \text{ such that } \phi_{h(x)} = f', \quad (6.9)
\]

so assume that \( h \) is such that (6.7.1) and (6.9) hold. Let \( a, b \) be such that \( \phi_{h(a)} = \kappa_0 \) a.e. and \( \phi_{h(b)} = \kappa_1 \) a.e. (recall \( \kappa_\iota \) is the constant function \( \lambda y . x \)). The indices \( a \) and \( b \) exist by (6.9). Let \( c \in D \) be such that for all \( x \geq c \), \( \phi_{h(a)}(x) = 0 \) and \( \phi_{h(b)}(x) = 1 \). Since both \( \phi_a \) and \( \phi_b \) have an infinite number of indices (see the proof of Theorem 6.2) and since (6.7.1) holds, we may assume that \( a, b \geq c \).

Let \( g \in \Omega \) be such that

\[
\phi_{g(x, y, z)}(w) = \begin{cases} 
\phi_a(w), & \text{if } \phi_x(y) = z, \\
\phi_b(w), & \text{otherwise}.
\end{cases}
\]
The function $g$ exists by Theorem 5.5. Then

$$
\phi_{g(x,y,z)} = \begin{cases} 
\phi_a, & \text{if } \phi_x(y) = z, \\
\phi_b, & \text{otherwise},
\end{cases}
$$

so that by (6.7.1),

$$
\phi_{h(g(x,y,z))} = \begin{cases} 
\phi_{h(a)}, & \text{if } \phi_x(y) = z, \\
\phi_{h(b)}, & \text{otherwise}.
\end{cases}
$$

Thus for any $(x, y, z)$ such that $g(x, y, z) \geq c$,

$$
\phi_{h(g(x,y,z))}(g(x, y, z)) = \begin{cases} 
0, & \text{if } \phi_x(y) = z, \\
1, & \text{otherwise},
\end{cases}
$$

i.e. for any $(x, y, z)$ with $g(x, y, z) \geq c$,

$$
diag_h(g(x, y, z)) = \begin{cases} 
1, & \text{if } \phi_x(y) = z, \\
0, & \text{otherwise}
\end{cases}
= \text{gr } U(x, y, z).
$$

Now let $f \in \Omega$ be such that

$$
f(x, y, z) = \begin{cases} 
g(x, y, z), & \text{if } g(x, y, z) \geq c, \\
a, & \text{if } g(x, y, z) < c \text{ and } \phi_{g(x,y,z)} = \phi_a, \\
b, & \text{if } g(x, y, z) < c \text{ and } \phi_{g(x,y,z)} = \phi_b.
\end{cases}
$$

Then $f$ can be computed from $g$ using a finite number of nested conditionals. Thus for all $x, y, z, f(x, y, z) \geq c$, and if $g(x, y, z) \geq c$, then

$$
diag_h(f(x, y, z)) = diag_h(g(x, y, z)) = \text{gr } U(x, y, z);
$$

otherwise, if $g(x, y, z) < c$, then

$$
\phi_{h(f(x,y,z))}(f(x, y, z)) = \begin{cases} 
\phi_{h(a)}(a), & \text{if } f(x, y, z) = a, \\
\phi_{h(b)}(b), & \text{if } f(x, y, z) = b
\end{cases}
= \begin{cases} 
0, & \text{if } \phi_x(y) = z, \\
1, & \text{otherwise}
\end{cases}
= \text{gr } U(x, y, z).
$$

Thus $\text{gr } U = diag_h \circ f$.

We might relax (6.7.1) to get

$$
\text{if } \phi_x = \phi_y, \text{ then } \phi_{h(x)} = \phi_{h(y)} \text{ a.e.} \tag{6.10}
$$

A standard separation result says that if $L_2$ is tape-constructible, then $\text{DSPACE}(L_2)$ contains a function not in $\text{DSPACE}(L_1)$ for any $L_1 = o(L_2)$. It is not known whether this result can be proved via a simple diagonalization. It can, however, be proved via a diagonalization in which the diagonal satisfies condition (6.10).
Open problem. Are diagonals satisfying the weaker condition (6.10) always as hard to compute as \( \text{gr } U \)?

Condition (6.10) diagonals cannot be used to prove all separation results; i.e. if diagonals are required to satisfy (6.10), then there exists counterexamples to Theorem 6.2. We leave the construction of such counterexamples to the reader.

7. An application

We now apply the results of the previous section to counter indexings of PTIME. Recall from Section 1 that in this indexing, each machine has some \( k \) inscribed in binary on an extra tape. On inputs \( x \) length \( n \), it constructs \( n^k \) on the extra tape, then runs as usual, erasing a symbol on the extra tape at each step and shutting off after \( n^k \) steps if the computation has not yet completed. This constitutes a common indexing of \( \text{PTIME} \); using this indexing, it can be proved that \( \text{PTIME} \subseteq \text{PSPACE} \) and \( \text{PTIME} \subseteq \text{DTIME}(2^n) \). Theorem 7.4 below implies that the graph of the universal function for this indexing cannot be computed in \( \text{PSPACE} \). Thus by Theorem 6.8, no simple diagonalization over this indexing can be used to show \( \text{PTIME} \neq \text{PSPACE} \). We mention this as motivation for the following generalization, which emphasizes the dependence of the complexity of diagonalization of the choice of indexing.

Definition 7.1. A counter indexing of PTIME is given by Turing machines with \( n^{f(k)} \) time counters, where \( f \) is any unbounded, nondecreasing, computable function. A subroutine computing \( f \) is encoded in the finite control of the machine and \( k \) is initially inscribed in binary on the extra tape.

Thus the indexing described above is a counter indexing with \( f = \iota \).

Definition 7.2. A counter indexing of PTIME satisfies the succinct composition property if the size of the machine produced by \( \text{comp} \) on input machines \( M, N \) is at most a constant multiple of the sizes of \( M \) and \( N \); i.e. if there is a constant \( c \) such that

\[
|\text{comp}(M, N)| \leq c(|M| + |N|)
\]

for any two \( M, N \).

All commonly used indexings satisfy this property, and in most cases \( c = 1 \); e.g. to compose two ALGOL programs, just concatenate them.

Lemma 7.3 below says that the succinct composition property forces \( f \) to grow at a certain rate. Intuitively, if a machine \( M \) outputs a string of length twice the length of its input, say, and if \( M^m \) is \( M \) composed with itself \( m \) times, then \( M^m \) must output a string of length \( 2^m \) times the length of its input. But if \( f \) grows too slowly and if the succinct composition property is in effect, then the counter of \( M^m \) will not be able to count high enough to accommodate this.
Lemma 7.3. If the counter indexing with $n^{f(x)}$ time counters satisfies the succinct composition property, then $f(x) \geq (\log x)^d$ a.e. for all $d \in \mathbb{N}$.

Proof. Suppose for a contradiction that $d$ is such that $f(x) \leq (\log x)^d$ i.o. and $c \geq 3$ is such that

$$|\text{comp}(M, N)| \leq c(|M| + |N|)$$

for any $M, N$. Let $M$ be any Turing machine and let $M^x$ represent $M$ with $x$ in binary and a subroutine computing $f$ attached. Let $a > 2d(1 + \log c)$ and $k > 2^{a(1+\log c)}$, and let $M$ be a polynomial time bounded Turing machine which on inputs of length $n$ outputs $n^k$ zeros. Since $f$ is unbounded and nondecreasing, for all sufficiently large $x$, $M^x$ computes $M$. Let $M^{m^x}$ be $M^x$ composed with itself $m$ times, using comp. By the restriction on comp's output length, solution of the appropriate recurrence gives

$$|M^{m^x}| \leq \left(2^{m-1} + \sum_{i=1}^{m^2} c^i\right)|M^x| \leq c^m|M^x|,$$

and $M^{m^x}$ on inputs on length $n$ outputs $n^{km}$ zeros. In particular, for $m = \log|M^x|$, $|M^{m^x}| \leq c^m|M^x| = |M^x|^{1+\log c} \leq (\log x)^{a/d}$

for sufficiently large $x$, by choice of $a$; and $M^{m^x}$ must output $n^{k \log |M^x|} = n^{M^{m^x} \log k}$ zeros.

Call $x$ principal if for all $y$, $x < y \leq 2^{(\log x)^{a/d}}$ implies $f(y) \leq (\log y)^a$.

We claim that there are infinitely many principal $x$. If $f(x) \leq (\log x)^a$ a.e., then the claim certainly holds. If $f(x) > (\log x)^a$ i.o., then there must be infinitely many $x$ and $z > x$ for which $f(x) \geq (\log x)^a$, $f(z) \leq (\log z)^d$, and for all $y$ in the interval $(x, z]$, $f(y) \leq (\log y)^a$. Then any such $x$ is principal, since

$$(\log x)^a \leq f(x) \leq f(z) \leq (\log z)^d,$$

therefore

$$z \geq 2^{(\log x)^{a/d}}.$$  

We claim now that for sufficiently large principal $x$, if $N$ is a Turing machine such that $M^{m^x} = N^x$, then $f(y) < |M^x|^{|\log k|}$. This claim establishes the theorem, since $M^{m^x}$ runs for at most $n^{f(x)} < n^{M^{m^x} \log k}$ steps on inputs of length $n$, hence cannot possibly output $n^{M^{m^x} \log k}$ zeros.

If $y \leq x$, then

$$f(y) \leq f(x + 1) \leq (\log(x + 1))^a \leq |M^x|^a < |M|^{|\log k|},$$

by choice of $k$. Otherwise $y > x$. Then

$$y \leq 2^{|N^x|} = 2^{|M^{m^x}|} \leq 2^{(\log x)^{a/d}},$$

so
so since \( x \) is principal,

\[
f(y) \leq (\log y)^a \leq (\log 2)^a M^{x_m} < |M^x|^{(1+\log c)} < |M^x|^{\log k},
\]

by choice of \( k \).

Thus if machines with \( n^{f(k)} \) time counters are to satisfy the succinct composition property, \( f \) must majorize \( \{ (\log x)^d \mid d \in \mathbb{N} \} \). Surprisingly, this is exactly the assumption we need to show that \( \text{gr } U \notin \text{PSPACE} \).

**Theorem 7.4.** If \( f(x) \geq (\log x)^d \) a.e. for all \( d \in \mathbb{N} \), then the graph of the universal function for this indexing (and hence any simple diagonal) cannot be computed in \( \text{PSPACE} \).

**Proof.** Let \( f \) majorize \( \{ (\log x)^d \mid d \in \mathbb{N} \} \). Suppose \( \text{gr } U \) runs in \( \text{DSpace}(n^c) \) for some \( c \). Let \( M \) be a Turing machine accepting a set in \( \text{DSpace}(n^{c+1}) - \text{DSpace}(n^c) \). Note that \( \text{DSpace}((2n + k)^c) = \text{DSpace}(n^c) \), by elementary space reduction results for Turing machines [8]. We will show, for a contradiction, how \( \text{gr } U \) can be used to simulate \( M \) in deterministic space \( (2n + k)^c \).

Let \( M^x \) denote \( M \) with \( x \) in binary and an \( n^{f(x)} \) counter attached, as in Definition 7.1. Let \( g \) be the function

\[
g = \lambda x. M^x(x).
\]

Then \( g \in \text{DSpace}(n^c) \), since there is a Turing machine which computes \( g \) by

(i) computing a description of \( M^x \),

(ii) running \( \text{gr } U \) on the pair of inputs \( M^x, x \).

The machine requires no work space at all to do (i), since it only involves appending a constant string (the description of \( M \) and a subroutine computing \( f \)) onto the binary representation of \( x \). The result is of length \( |x| + k \) for some constant \( k \), and since \( \text{gr } U \) runs in space \( n^c \), \( g \) requires space \( (|M^x| + |x|)^c = (2|x| + k)^c \), thus

\[
g \in \text{DSpace}((2n + k)^c) \subseteq \text{DSpace}(n^c).
\]

Now \( M \) runs in time \( b^{n^{c+1}} \) on inputs of length \( n \), for \( b \) large enough to encode states, tape symbols, etc. of \( M \), and \( M^x \) may run for \( n^{f(x)} \) steps on inputs of length \( n \) before its counter runs out. Thus on input \( x \), \( M^x \) may run for \( |x|^{f(x)} \) steps. But since \( f(x) \) majorizes \( \{ (\log x)^d \mid d \in \mathbb{N} \} \), we have \( f(x) \geq |x|^c \) a.e., thus

\[
|x|^{f(x)} > b|x|^{c+1} \quad \text{a.e.}
\]

This says that for almost all \( x \), \( M^x \) on input \( x \) has enough time to complete the simulation of \( M \) on \( x \) before its clock runs out, so that the function \( g = \lambda x. M^x(x) \) is the same as \( M \) a.e., hence has the same space complexity as \( M \). This is a contradiction.

**Corollary 7.5.** If a counter indexing of \( \text{PTIME} \) satisfies the succinct composition
property, then no simple diagonalization over this indexing can be used to show \( \text{PTIME} \neq \text{PSPACE} \).

**Proof.** Lemma 7.3 and Theorems 6.8 and 7.4.

8. Concluding remarks

We have given an axiomatization capturing the idea of a well-behaved subrecursive indexing, and explored its elementary consequences. Relationships between diagonalization, uniform simulation, deciding membership, and deciding other nontrivial properties have been established.

Several directions for further investigation present themselves. It is possible that much of the extant work in abstract subrecursive complexity [1, 2, 13] can be reformulated in this framework. This would require the introduction of an absolute complexity measure in the form of axioms governing \text{comp}, \text{const}, and \text{pair}, similar in spirit to those of Blum [4]; one version of this has already been done by Machtney and Young [13].

There are several interesting questions concerning the structure of the semilattice of \( \leq^C \)-degrees. Is it dense? Is there an indexing for \( C \) whose universal function has minimal \( \leq^C \)-degree among all \( \leq^C \)-degrees of universal functions of indexings of \( C \)? It is conjectured that, in the presence of the axioms of Section 2, a 'rudimentary simulation' property similar to that of Mehlhorn [14] is necessary and sufficient for density.

Finally, can it be proved that gr \( U \) for any indexing of \text{PTIME} \ requires more than polynomial space to compute? We have proved this for a wide class of indexings, namely counter indexings satisfying the succinct composition property. The general result would have profound implications regarding the \( \text{P} = \text{PSPACE} \) question.

References


