Reprinted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES
Vol. 12, No. 2, April 1976
Printed in Belgium

Polynomial and Abstract Subrecursive Classes*

KURT MEHLHORN

Fachbereich 10, Universität des Saarlandes, 66 Saarbrücken, West Germany

Received January 5, 1975

The reducibility "polynomial time computable in" for arbitrary functions is introduced. It generalizes Cook's definition from sets to arbitrary functions. A complexity-theoretic as well as a syntactic characterization is given and their equivalence is shown. This equivalence and the naturalness of both definitions give evidence that our notion is "correct." The computable functions are classified into polynomial classes according to this reducibility. The ordering of these classes under set inclusion is studied. Honest classes are introduced and using them, the classification is related to the computational complexity of the functions classified. The algebraic structure of honest classes is also investigated.

In Section II abstract subrecursive reducibilities are introduced. An axiomatic definition in purely recursion-theoretic terms is given; in particular, no reference to a particular machine model is needed. The definition is in the spirit of Strong's and Wagner's characterizations of basic recursive function theories. All known reducibilities are abstract reducibilities. The algebraic structure of abstract classes is explored.

INTRODUCTION

We investigate the relative difficulty (complexity) of computable functions in the framework of subrecursive reducibilities. We treat two problems in this area.

First, all subrecursive reducibilities mentioned in the literature are very coarse. In particular, they do not yield any information about the relative difficulty of functions of subexponential complexity. However, this is the class of functions in which computer scientists are most interested. We define the reducibility "polynomial time computable in," which is fine enough to classify the functions of subexponential complexity, and study the rich structure which is obtained. Constable [7] also obtained results in this direction. Second, many theorems about subrecursive classes hold true for "all" reducibilities. This suggests that there is an underlying common characteristic which makes these theorems true. Such a characteristic should be suceptible to an axiomatic treatment in the sense of Strong [36] and Wagner [37].

* The research presented in this paper was supported in part by the NSF under Grant GJ-579. It was done while the author was a graduate student at Cornell University.

We attempt such an axiomatic treatment and thus answer challenges of Constable and Borodin [8] and Machtey [SIGACT, '71, p. 256]. Ladner [18] also obtained results in this direction.

The reducibility "polynomial time computable in" is defined in terms of the class of polynomial time computable operators. We give a complexity-theoretic as well as a syntactic definition of this class. Both definitions are natural extensions of the corresponding characterizations of the set of polynomial time computable functions. The equivalence of the two definitions is established, thus making the "correctness" of our definition plausible.

The algebraic properties of polynomial time reducibility are studied. It is compared with Cook's and Constable's definition of relative feasibility and with elementary recursive reducibility. Our strongest algebraic result is a density theorem: Every countable partial ordering can be embedded between any two polynomial degrees.

Then we study the relationship between relative and absolute (computational) complexity. We pick as a measure of computational complexity Turing machine time. In this context, it is established that the relative complexity of functions is mirrored in their absolute complexity.

A polynomial class is honest if it is generated by the running time of some Turing machine. Such classes have the Ritchie–Cobham property, i.e., they are computation time full and closed. They are complexity classes. We also establish a strong density theorem for honest classes.

In Section II we attempt an axiomatic treatment of subrecursive reducibilities. The following observation is crucial to our approach: Subrecursive reducibilities are always induced by a recursively enumerable class of general recursive operators, properties of reducibilities are elegantly formulated as properties of the corresponding class of operators, and conditions on classes of operators are easily stated axiomatically. An abstract subrecursive reducibility is given by a recursively enumerable list of general recursive operators, which contains the operator APPLY[ ], where $\mathrm{APPLY}[f](x) = f(x)$, is closed under composition, definition by cases, and finite variants and permits rudimentary simulations. We draw one interesting conclusion from the set of axioms listed so far: There are no minimal degrees.

Then we further restrict the class of reducibilities under consideration by requiring the existence of pairing functions. The additional axiom implies a strong density theorem and the existence of minimal pairs.

I. Preliminaries

$\mathbb{N}$ is the set of natural numbers including 0, and $\Sigma = \{a_1, \ldots, a_r\}$ is a fixed finite alphabet with more than one symbol ($r > 1$). Σ^* is the set of words over Σ, and ϵ is the empty word in Σ^*.

For y in Σ^*, $|y|$ is the length of y, and for $x \in (\Sigma^*)^n$, where $x = (x_1, \ldots, x_n)$, $|x|$ is the n-tuple $(|x_1|, \ldots, |x_n|)$. For a function $g: (\Sigma^*)^n \to \Sigma^*$, $|g|: (\Sigma^*)^n \to \mathbb{N}$ is defined as $|g|(x) = |g(x)|$.

There is a natural correspondence between strings and natural numbers. Interpret strings over Σ as integers in r-ary notation, i.e.,

$$a_{i_n} \cdots a_{i_0} \mapsto \sum_{k=0}^{n} (i_k - 1)\, r^k.$$

We also map ϵ on O. This correspondence implies an ordering on Σ^+ which we denote by $\leqslant$. We extend this ordering to Σ^*, by defining $\epsilon < y$ for all $y \in \Sigma^+$.

An operator maps functions into functions. The operations of substitution *Os* [9] are composition and explicit transformation. $f(x) = h_0(h_1(x), \ldots, h_m(x))$ is defined from $h_0, h_1, \ldots, h_m$ by composition and $f(x_1, \ldots, x_n) = h(y_1, \ldots, y_m)$, where each y_i is either a constant in Σ^* or one of the x_j's, is defined from h by explicite transformation. If $f_1, \ldots, f_n$ are functions and $O_1, \ldots, O_m$ are operators, then $[f_1, \ldots, f_n ; O_1, \ldots, O_m]$ denotes the smallest class of functions containing $f_1, \ldots, f_n$ and closed under $O_1, \ldots, O_m$.

Functions $\langle\ \rangle: (\Sigma^*)^n \to \Sigma^*$ and $(\ )_i: \Sigma^* \to \Sigma^*$ for $1 \leqslant i \leqslant n$ are called encoding and decoding functions of n-tuples if $(\langle x_1, \ldots, x_n \rangle)_i = x_i$ for $1 \leqslant i \leqslant n$ and all $x_1, \ldots, x_n \in \Sigma^*$. A well-known example is

$$\langle x_1, \ldots, x_n \rangle = c(x_1) \cdots c(x_n),$$

where

$$c(a_{i_1} \cdots a_{i_m}) = a_1 a_2^{i_1} a_1 a_2^{i_2} a_1 \cdots a_1 a_2^{i_m} a_1 .$$

II. Polynomial Classes

1. *Polynomial Operators*

Our primary goal is to define relative polynomial time computability (relative feasibility). Of course there can be no mathematical proof of the correctness of our formalization; we can only compile evidence for the correctness. To do so, we give two definitions, a complexity-theoretic as well as a syntactic one, and show their equivalence. Both definitions are natural extensions of the corresponding characterizations of the set of polynomial time (feasibly) computable functions. This naturalness adds further evidence that our definition is "correct."

Our machine models are multitape Turing machines (cf. [14]). Such a machine has some number $n_i \geqslant 0$ read-only input tapes, some number $n_w \geqslant 0$ work tapes and one write-only output tape. The tape alphabet of input, work, and output tapes is Σ. Thus, Turing machines compute partial functions from $(\Sigma^*)^n$ to Σ^*; we often

denote the function computed by a Turing machine M also by M. We pick as a measure of computational complexity the number of steps taken (time).

DEFINITION 1.1. A function $f: (\Sigma^*)^n \to \Sigma^*$ is *polynomial time computable* if there is a Turing machine M and a polynomial p such that for all inputs $x = (x_1, x_2, \ldots, x_n)$, M halts within $p(|x|)$ steps with $f(x)$ on its output tape. We denote the class of such functions by $\mathcal{P}ol$.

Several investigators [3, 5, 38] have given a syntactic definition for $\mathcal{P}ol$.

DEFINITION 1.2 (Cobham, Weihrauch). $\mathcal{L} = [S_1, S_2, \ldots, S_r, A_2; Os, \text{LRN}]$, where the S_i are the generalized successor functions $S_i(y) = ya_i$, A_2 is the length-bounded exponentiation function $A_2(x, y) = a_1^{|x||y|}$, Os are the operations of substitution [8], and LRN is limited recursion on notation: f is defined from $g, h_1, \ldots, h_r$ and b by limited recursion on notation if for some $n \geqslant 0$,

$$
\begin{aligned}
&g: (\Sigma^*)^n \to \Sigma^*,\\
&h_i: (\Sigma^*)^{n+2} \to \Sigma^*, \qquad (1 \leqslant i \leqslant r),\\
&b: (\Sigma^*)^{n+1} \to \Sigma^*,\\
&f: (\Sigma^*)^{n+1} \to \Sigma^*,\\
&f(x, \epsilon) = g(x),\\
&f(x, ya_i) = h_i(x, y, f(x, y)),\\
&|f(x, y)| \leqslant |b(x, y)|,
\end{aligned}
$$

for all $x \in (\Sigma^*)^n$ and $y \in \Sigma^*$.

Fact 1.3 [5, 38]. $\mathcal{P}ol = \mathcal{L}$.

In the main theorem of this section (Theorem
operator level, i.e., to relativized computations. First we need to generalize the definitions.

DEFINITION 1.4. $\mathcal{L}(\ ) = [S_1, S_2, \ldots, S_r, A_2, f; Os, \text{LRN}]$. We consider f as an uninterpreted function symbol, $\mathcal{L}(\ )$ as a class of operators.

The complexity-theoretic definition is not as easily generalized. We first make an observation about the growth rates of functions in $\mathcal{L}$.

Observation 1.5. Let $g \in \mathcal{L}$, $g: (\Sigma^*)^n \to \Sigma^*$. Then there is a polynomial p such that for all $x \in (\Sigma^*)^n$,

$$|g(x)| \leqslant p(|x|).$$

Conversely, let $p: \mathbb{N}^n \to \mathbb{N}$ be a polynomial. Then there is a function $g \in \mathscr{L}$, $g: (\Sigma^*)^n \to \Sigma^*$ such that for all $x \in (\Sigma^*)^n$

$$p(|x|) \leqslant |g(x)|.$$

The proof of this observation is a simple induction. It is left to the reader. Using Observation 1.5, we can give an alternate definition of $\mathscr{P}ol$.

$$\mathscr{P}ol = \{f; \text{ there is a Turing machine } M \text{ computing } f \text{ such that the running time of } M \text{ is bounded by } |p| \text{ for some } p \in \mathscr{L}\}.$$

An oracle Turing machine is a multitape Turing machine which has, in addition, some number $n_0 + 1 \geqslant 1$ oracle tapes and a query state q_f. n_0 of the oracle tapes are write-only oracle input tapes and one oracle tape is a read-only oracle output tape. The oracle tape alphabet is also Σ. An oracle machine takes two inputs: a function $f: (\Sigma^*)^{n_0} \to \Sigma^*$ and an n-tuple $x = (x_1, ..., x_n)$ of strings. It operates exactly as an ordinary Turing machine does, except when it enters its query state q_f. Suppose it enters its query state q_f. Let $y = (y_1, ..., y_{n_0})$ be the content of the n_0 oracle input tapes at that moment. In *one* machine step the content of the oracle output tape is replaced by the string $f(y_1, ..., y_{n_0})$ and the oracle input tapes are erased. Also, the read head on the oracle output tape is placed on the first symbol of $f(y_1, ..., y_{n_0})$ and some other state is entered.

Oracle machines compute operators. A machine with n_0 oracle input tapes and n_i input tapes takes n_0-ary functions into n_i-ary functions.

As a measure of complexity we again pick the number of steps executed before halting. Oracle calls count as just *one* step. One can consider the oracle as a subroutine. We count only the number of steps taken by the main program; the calls of the subroutine are free. Thus, the number of steps taken by main program is a measure of the complexity of the computed function relative to the given subroutine (oracle).

Two remarks on our definition might be useful. We distinguish between oracle input and output tapes even in the case $n_0 = 1$. This idea is implicit in [7] and stated in [24a]. Suppose $n_0 = 1$. Compare the complexity of $Op[f] = i$-fold composition of f with itself under the assumption of having an oracle input tape as well as an output tape and under the assumption of just one oracle tape. In the latter case the i iterations can be done in i steps independent of the values of f. It is our strong belief that this is unreasonable. In our model, though a machine is not charged for calling the oracle, it is charged in a reasonable way for reading the result of the call.

We required the erasure of the oracle input tapes, whenever an oracle call occurred. This convention is purely technical; it does not change the complexity of computation by a great amount; however, it facilitates the proof of Theorem 1.7 considerably.

DEFINITION 1.6 (*Polynomial time computable operator*). $\mathscr{P}ol(\ ) = \{\Phi[\ ];$ there is an oracle Turing machine M and a bounding expression $P[\ ] \in \mathscr{L}(\ )$ such that

(1) M computes the operator $\Phi[\ ]$,

(2) M on f and x halts within $|P[f](x)|$ steps for all oracles f and n-tuples $x\}$.

So far we have given two definitions for the class of polynomial operators. The next theorem relates them.

THEOREM 1.7. $\mathscr{P}ol(\ ) = \mathscr{L}(\ )$.

Proof. (a) $\mathscr{L}(\ ) \subseteq \mathscr{P}ol(\ )$: For every $H[\ ] \in \mathscr{L}(\ )$ we have to construct an oracle machine M and a bounding expression $P[\ ] \in \mathscr{L}(\ )$ such that

(1) M computes $H[\ ]$,

(2) M on f and x halts within $|P[f](x)|$ steps for all oracles f and n-tuples x.

Just for this proof we denote by $T_M[f](x)$ the length of the computation of M with oracle f and argument x. We construct M and $P[\ ]$ by induction on the structure of $H[\ ]$. The base step follows from Fact 1.3. For the induction step we consider only the case of limited recursion on notation. The case of the operations of substitution is similar and is left to the reader.

Let $H_0[\ ], H_1[\ ], \ldots, H_r[\ ]$ and $B[\ ]$ be in $\mathscr{L}(\ )$ and let $H[\ ]$ be defined from them by limited recursion on notation, i.e.,

$$H[f](x, \epsilon) = H_0[f](x),$$
$$H[f](x, ya_i) = H_i[f](x, y, H[f](x, y)),$$
$$|H[f](x, y)| \leqslant |B[f](x, y)|.$$

By induction hypothesis, all $H_i[\ ]$ are in $\mathscr{P}ol(\ )$; say oracle machine M_i with bounding expression $P_i[\ ] \in \mathscr{L}(\ )$ computes $H_i[\ ]$. The M_i are put together to a single oracle machine M.

M operates as follows. Suppose the argument is (x, y), where $x = (x_1, \ldots, x_n)$ and $y = y_1 y_2 \cdots y_m$. Then M first computes $H_0[f](x)$ using M_0 and then in sequence $H[f](x, y_1), H[f](x, y_1 y_2), \ldots, H[f](x, y_1 y_2 \cdots y_m)$ using the machines $M_1, M_2, \ldots, M_r$.

The length $T_M[f](x, y)$ of the computation of M on f and (x, y) is bounded above by

$$T_M[f](x, y) \leqslant \text{trivial terms} + \sum_{k=0}^{|y|-1} T_{M_j}[f](x, y_1 \cdots y_k, H[f](x, y_1 \cdots y_k)),$$

where $y_{k+1} = a_j$. Since $\mathscr{L}(\ )$ is closed under concatenation we only have to consider the nontrivial term. This term is bounded above by

$$|y| \max\{|P_j[f](x, y_1 \cdots y_k, H[f](x, y_1 \cdots y_k))|;\ 0 \leqslant k \leqslant |y| - 1,\ 1 \leqslant j \leqslant r\}.$$

The standard recursion bounding technique (e.g., [20, 34, 38]) appeals now to the monotonicity of the P_j's and of B. We cannot use this kind of reasoning here; running times are not monotone functions in general and the "smoothing" operation $f \to \lambda x \max_{y \leqslant x} f(y)$ is not polynomial time computable. Note, however, that we do not need to maximize over all predecessors of y but only over the prefixes of y. This can be done within $\mathscr{L}(\ )$.

LEMMA 1.8. *If* $G[\ ](x, y) \in \mathscr{L}(\ )$ *then* $H[\ ](x, y) = G[\ ](x, y')$ *for some prefix* y' *of* y *with* $|H[\ ](x, y)| = \max\{|G[\ ](x, y'')|; y''$ *prefix of* $y\}$ *is in* $\mathscr{L}(\ )$.

Proof. We determine y' by limited recursion on notation

$$\begin{aligned} L[f](x, \epsilon) &= \epsilon, \\ L[f](x, ya_i) &= \textit{if } |G[f](x, ya_i)| \geqslant |G[f](x, L[f](x, y))| \\ &\qquad \textit{then } ya_i \\ &\qquad \textit{else } L[f](x, y), \\ |L[f](x, y)| &\leqslant |y|, \end{aligned}$$

and set $H[f](x, y) = G[f](x, L[f](x, y))$. ∎

Since $\mathscr{L}(\ )$ is closed under condatenation, (1) can be bounded above by

$$|y| \cdot \max\{|G[f](x, y')|; y' \text{ prefix of } y\}$$

for some $G[\ ] \in \mathscr{L}(\ )$. Now appeal to Lemma 1.8 and take $A_2(y, H[f](x, y))$ as the bounding expression for M.

(b) The converse inclusion $\mathscr{P}ol(\ ) \subseteq \mathscr{L}(\ )$ is proved by low-level arithmetization of oracle machines. Low-level arithmetization of nonoracle Turing machines is well known. (e.g., [5, 10, 38]).

Such an arithmetization is usually done in three steps:

(1) Choose an encoding of instantaneous descriptions into numbers [10] or strings [5, 38].

(2) Define a function Yield which simulates one step of the computation.

(3) Iterate the function Yield by means of limited recursion [10] or limited recursion on notation [5, 38].

Two problems arise: For step (1), one needs a "good" encoding function. This is easily solved in this context. The coding function described in Section I is in $\mathscr{L}(\ )$ [38]. For step (3) one has to derive a relation between the size of the instantaneous description, i.e., the length of the nonblank portions of the tapes, and the length of the computation. This is easy in the case of nonoracle Turing machines. To use

a new tape square, a Turing machine has to put a head on it and therefore has to perform a move.

The same reasoning applies to all tapes of an oracle machine except its oracle output tape. An oracle call can cause the length of the oracle output tape to grow by an arbitrary amount. Thus, we cannot store the oracle output tape explicitly. Now inspect the definition of oracle machine more closely. Though an oracle machine is charged only a nominal amount for calling the oracle, it is charged in a reasonable way for reading the output of an oracle call. Thus, the position (distance from the left end of the tape) of the head on the oracle output head is bounded by the number of steps taken so far and can therefore be included in the instantaneous description. Furthermore, we include not only the actual content of the oracle input tapes but also their content just prior to the most recent oracle call.

The character scanned by the head on the oracle output tape is computed by applying the oracle to the arguments of the most recent call (which we have saved) and measuring the result against the variable which represents the head position. The details are left for the reader. ∎

DEFINITION 1.9. (a) A function f is *polynomial time computable in* a function g if there is a polynomial operator $Op[\ ] \in \mathscr{P}ol(\ )$ such that $f = Op[g]$. We write $f \leqslant_{\text{pol}} g$.

(b) $\mathscr{P}ol(g) = \{f; f \leqslant_{\text{pol}} g\}$ is the *polynomial class* of (generated by) g.

(c) $\deg(g) = \{f; f \leqslant_{\text{pol}} g \text{ and } g \leqslant_{\text{pol}} f\}$ is the *polynomial degree* of (generated by) g.

2. *Elementary Properties of Relative Feasibility*

We state some simple properties of relative feasibility, relate our definition to previous ones, and finally investigate the relation to elementary recursive reducibility.

THEOREM 2.1. (a) $\leqslant_{\text{pol}}$ *is a preorder, i.e.,* $\leqslant_{\text{pol}}$ *is reflexive and transitive.*

(b) $\mathscr{P}ol = \mathscr{P}ol(\lambda x \,.\, \epsilon)$ *is the smallest polynomial class.*

(c) $\leqslant_{\text{pol}}$ *is an upper semilattice. We denote the least upper bound of* f *and* g *by* f join g.

(d) *There is an effective enumeration of* $\mathscr{P}ol(\ )$ *(and hence of* $\mathscr{P}ol(f)$ *for* f*).*

Proof. (a) APPLY$[\ ] \in \mathscr{P}ol(\ )$ implies the reflexitivity of $\leqslant_{\text{pol}}$. To show the transitivity of $\leqslant_{\text{pol}}$ we use the syntactic rather than the semantic definition. It makes the proof almost trivial.

Assume $f \leqslant_{\text{pol}} g$ and $g \leqslant_{\text{pol}} h$. Then there are operators $Op_1[\ ]$ and $Op_2[\ ]$ in $\mathscr{L}(\ )$ such that $f = Op_1[g]$ and $g = Op_2[h]$. $Op_1[\ ]$ and $Op_2[\ ]$ are defined by systems S_1 and S_2 of equations, respectively. Replacing the uninterpreted function symbol

in S_1 by the "terminal" function symbol of S_2 gives the system S. S defines an operator $Op[\ ] \in \mathscr{L}(\ )$ such that $f = Op[h]$. Hence, $f \leqslant_{\text{pol}} h$.

(b) The operator $E[f](x) = \epsilon$ is obviously polynomial time computable. Thus, $\lambda x\,.\,\epsilon \leqslant_{\text{pol}} f$ for every f.

(c) Let $\mathscr{P}ol(f)$ and $\mathscr{P}ol(g)$ be two polynomial classes. Define

$$h(z) = \textit{if } (z)_1 = \epsilon \textit{ then } f((z)_2) \textit{ else } g((z)_2).$$

Obviously, $f, g \leqslant_{\text{pol}} h$.

Suppose $f, g \leqslant_{\text{pol}} k$. Then $f = Op_f[k]$ and $g = Op_g[k]$ for some polynomial operators. Hence,

$$h(z) = \textit{if } (z)_1 = \epsilon \textit{ then } Op_f[k]((z)_2) \textit{ else } Op_g[k]((z)_2) = Op[k]$$

for some polynomial operator $Op[\ ]$. A bounding expression for $Op[\ ]$ is easily derived from the bounding expressions for $Op_f[\ ]$ and $Op_g[\ ]$. Thus $h \leqslant_{\text{pol}} k$, and hence h is the least upper bound of f and g.

(d) $\mathscr{P}ol(\ ) = \mathscr{L}(\ )$ and $\mathscr{L}(\ )$ is effectively enumerable by definition. ∎

For the remainder of the section we assume a fixed enumeration $\{Op_i[\ ]\}$ of $\mathscr{P}ol(\ )$.

As we mentioned in the Introduction, Cook [9] gave a definition of polynomial reducibility in the case of decision problems (sets), and Constable [7] gave a definition in the general case. Constable defined relative feasibility in a complexity-theoretic as well as in a syntactic way. The two characterizations are equivalent in the case of nondecreasing (interpret strings as integers in r-ary notation) functions. The next theorem relates our definition to theirs. They are equivalent in their respective realms.

THEOREM 2.2. (a) *If g is a characteristic function then $f \leqslant_{\text{pol}} g$ if and only if f is polynomial reducible to g in the sense of Cook.*

(b) *If g is nondecreasing, then $f \leqslant_{\text{pol}} g$ if and only if f is polynomial reducible to g in the sense of Constable.*

Proof. (a) f is polynomial reducible to g in the sense of Cook if there is an oracle machine M and a polynomial p such that

(1) M with oracle g computes f,

(2) M on h and x halts within $p(|\ x\ |)$ steps for every oracle h.

Since $\mathscr{L} \subseteq \mathscr{L}(g)$ and the growth rates of functions in $\mathscr{L}$ are essentially the polynomials (Observation 1.5), Cook's definition is more restrictive than ours. The converse follows immediately from the following lemma.

LEMMA 2.3. *For every operator* $H[\ ] \in \mathscr{L}(\ )$ *there is a polynomial* p *such that for all characteristic functions* g, $|H[g](x)| \leqslant p(|x|)$.

Proof. The proof is by induction on the structure of $H[\ ]$. Since similar arguments will be used extensively later on, we give a detailed proof.

Induction Base

Case 1. $H[g](x) = S_i(x)$ for some i. Then $|H[g](x)| = |x| + 1$.

Case 2. $H[g](x, y) = A_2(x, y)$. Then $|H[g](x, y)| = |x| \cdot |y|$.

Case 3. $H[g](x) = g(x)$. Then $|H[g](x)| \leqslant 1$.

Note that $|x| + 1$, $|x| \cdot |y|$, 1 are monotone functions.

Induction Step

Case 1. H is defined from $H_0, H_1, \ldots, H_m \in \mathscr{L}(\ )$ by composition. By induction hypothesis there are monotonically increasing polynomials $p_0, p_1, \ldots, p_m$ such that

$$|H_i[g](x)| \leqslant p_i(|x|) \qquad \text{for} \quad 0 \leqslant i \leqslant m.$$

Hence,

$$\begin{aligned} |H[g](x)| &= H_0[g](H_1[g](x), \ldots, H_m[g](x)) \\ &\leqslant p_0(|H_1[g](x)|, \ldots, |H_m[g](x)|) \\ &\leqslant p_0(p_1(|x|), \ldots, p_m(|x|)) \\ &\leqslant p(|x|) \end{aligned}$$

for some monotonically increasing polynomial p.

Case 2. H is defined from H' by explicit transformation, i.e.,

$$H[g](x_1, \ldots, x_n) = H'[g](y_1, \ldots, y_m),$$

where each y_i is either a constant in Σ^* or one of the x_j. By induction hypothesis there exists some monotonically increasing polynomial p' such that

$$|H'[g](y)| \leqslant p'(|y|).$$

Hence,

$$\begin{aligned} |H[g](x)| &\leqslant |H'[g](y_1, \ldots, y_m)| \\ &\leqslant p'(|y_1|, \ldots, |y_m|) \\ &\leqslant p(|x|) \end{aligned}$$

for some monotonically increasing polynomial p.

Case 3. H is defined from $H_0, H_1, \ldots, H_r$ and B by limited recursion on notation. By induction hypothesis there exists a monotonically increasing polynomial p such that

$$|B[g](x)| \leqslant p(|x|).$$

Hence, by definition of LRN,

$$|H[g](x)| \leqslant |B[g](x)| \leqslant p(|x|).$$

(b) f is polynomial reducible to g in the sense of Constable if there is an oracle machine M and a bounding expression $G[\,] \in [+, \cdot, g; Os, \sum_{\|}, \prod_{\|}]$ such that

(1) M with oracle g computes f,

(2) M on h and x halts within $|G[h](x)|$ steps for every oracle h.

Here $\sum_{\|}$ denotes length-bounded summation and $\prod_{\|}$ denotes length-bounded product.

Again it is clear that this definition is more restrictive than ours: Summation and multiplication are in $\mathscr{L}$ and $\mathscr{L}(\,)$ is closed under length-bounded summation and multiplication; i.e., they are definable by limited recursion on notation. This is easily proved by an argument similar to the one used in the proof of Lemma 1.8.

The converse follows from the following lemma.

LEMMA 2.4. *For every operator* $H[\,] \in \mathscr{L}(\,)$ *there is an operator* $G[\,] \in [+, \cdot, g; Os, \sum_{\|}, \prod_{\|}]$ *such that for all nondecreasing functions* g,

$$|H[g](x)| \leqslant |G[g](x)|.$$

Proof. The proof is similar to the proof of Lemma 2.3 and therefore is left to the reader. Because of the monotonicity of g we can assume the bounds $G[\,]$ to be monotone: Take g itself as a monotone bound for g in Case 3 of the induction base. Monotonicity of the bounds is essential for Case 1 (composition) of the induction step. ∎

As we remarked in the proof of Theorem 1.7, the smoothing operation $g \mapsto \lambda y \,.\, \max\{g(x); x \leqslant y\}$ is not in $\mathscr{L}(\,)$. Lemma 2.4 therefore probably fails in the case of nonmonotone g.

Elementary recursive reducibility is used extensively in the literature [18, 20, 27, 29].

DEFINITION 2.5. $\mathscr{E}(f) = [+, \cdot, \lambda x \cdot r^x, f; Os, R_{\leqslant}]$, where $R_{\leqslant}$ is limited recursion, is the *elementary recursive class* of f. k is defined from g, h, and b by limited recursion if

$$k(x, 0) = g(x),$$
$$k(x, y+1) = h(x, y, k(x, y)),$$
$$k(x, y) \leqslant b(x, y).$$

The next theorem relates polynomial classes to elementary recursive classes.

THEOREM 2.6. (a) $\mathscr{P}ol(f) \subseteq \mathscr{E}(f)$ *for all* f.

(b) *If* $|f(x)| \geqslant 2^{|x|}$ *for all* x, *then* $\mathscr{P}ol(f) = \mathscr{E}(f)$.

Proof. Part (a) is an immediate consequence of the equality $\mathscr{P}ol(f) = \mathscr{L}(f)$, which was proved above (Theorem 1.7).

(b) We first note that $\mathscr{L}(f)$ contains multiplication and summation. Then we define exponentiation by limited recursion on notation.

$$r^{\epsilon} = a_2 \qquad (a_2 \text{ is the } r\text{-ary notation of } 1),$$
$$r^{xa_i} = (r^x)^r \cdot r^{(i-1)},$$
$$|r^x| \leqslant r\,|f(x)|.$$

It remains to show that $\mathscr{L}(f)$ is closed under limited recursion. Assume k is defined from g, h and j by limited recursion. We mimic limited recursion by limited recursion on notation. If

$$k'(x, \epsilon) = g(x),$$
$$k'(x, ya_i) = h(x, |y|, k'(x, y)),$$
$$|k'(x, y)| \leqslant |j(x, |y|)|,$$

then

$$k(x, y) = k'(x, r^y)$$

and hence, $k \in \mathscr{L}(f)$. ∎

A polynomial class $\mathscr{P}ol(f)$ is thus always contained in the corresponding elementary recursive class $\mathscr{E}(f)$: polynomial reducibility refines elementary reducibility. However, $\mathscr{P}ol(f)$ is a proper subset of $\mathscr{E}(f)$ only if f is not larger than exponentiation. This fact might be surprising, but it is not counterintuitive. Intuitively speaking, polynomial (time-bounded) operations are length-bounded, e.g., length-bounded search, length-bounded summation $\sum_{|\ |}$, length-bounded product $\prod_{|\ |}$. Elementary operations are value-bounded, e.g., value-bounded search, value-bounded summation $\sum$, value-bounded product $\prod$. Since $|2^x| = x$, the difference disappears in the presence of an exponentiation function.

3. *Algebraic Properties of Polynomial Reducibility*

This section concerns itself with the algebraic properties of polynomial reducibility. We ask the following questions. Is it a dense order? Are there minimal degrees? Are there minimal pairs of elements? Do greatest lower bounds always exist?

Machtey and Ladner answered these questions in the case of elementary recursive classes. Their proofs generalize quite nicely, though some care is needed to execute the constructions within $\mathscr{P}ol(\ )$. To demonstrate the refined "looking-back" technique and to prepare the reader for the abstract treatment of Section III, we give one sample proof in some detail. It refines constructions in [18, 21].

THEOREM 3.1. (*Minimal degrees do not exist.*) *Let a be a polynomial degree such that $\mathscr{P}ol <_{\text{pol}} a$. Then there exists a polynomial degree b such that $\mathscr{P}ol <_{\text{pol}} b <_{\text{pol}} a$.*

Proof. Let $\mathscr{P}ol = \deg(0)$ and $a = \deg(g)$. The hypothesis of the theorem is $g \notin \mathscr{P}ol$, i.e., $g \neq Op_i[0]$ for all i.

We have to construct a function h such that:

(1) $h \leqslant_{\text{pol}} g$;

(2) $g \not\leqslant_{\text{pol}} h$, i.e., $g \neq Op_i[h]$ for all i;

(3) $h \not\leqslant_{\text{pol}} 0$, i.e., $h \neq Op_i[0]$ for all i.

If h looks like 0, then requirement (2) is ensured; if h looks like g, then requirement (3) is ensured. Therefore, our policy is to let h look like 0 and g alternately; i.e., we define a sequence $\epsilon = z_1 \leqslant z_2 \leqslant \cdots$ of strings such that

(1) $h(x) = g(x)$ for $z_{2i+1} \leqslant x \leqslant z_{2i+2}$,

(2) $h(x) = 0(x)$ for $z_{2i+2} \leqslant x < z_{2i+3}$.

This sequence has the additional properties that

(1′) $h(y) \neq Op_i[0](y)$ for some $y < z_{2i+2}$,

(2′) $g(y) \neq Op_i[h](y)$ for some $y < z_{2i+3}$.

The z_i are defined inductively. Assume that z_{2i+1} is already defined. By the hypothesis of the theorem there are infinitely many y such that $g(y) \neq Op_i[0](y)$. (Note that polynomial degrees are closed under forming finite variants. This is an immediate consequence of the semantic definition of $\mathscr{P}ol(\ )$.) If we tentatively set $h(x) = g(x)$ for $x \geqslant z_{2i+1}$, then there are infinitely many y such that $h(y) \neq Op_i[0](y)$. Therefore, we only have to choose z_{2i+2} large enough such that there is a $y < z_{2i+2}$ with $h(y) \neq Op_i[0](y)$. Since we also have to make sure that $h \leqslant_{\text{pol}} g$, the selection process must not be too costly. It is easy to provide for that. We choose z_{2i+2} large enough such that a y with $h(y) \neq Op_i[0](y)$ can be found in time $| z_{2i+2} |$. These ideas are formalized in the following program. $\{x_i\}$ is the enumeration of Σ^* in lexicographical order. M is a Turing machine which computes g.

To compute $h(x)$:

(A) Let $n = |x|$; execute n steps of each of the following programs.

"Compute $g(x_1), g(x_2),\ldots$ using M and build up a table G of the values."
"Compute $h(x_1), h(x_2),\ldots$ and build up a table H of the values."

Let G_0 and H_0 be the tables computed by the programs above. Using these tables, compute n steps of the following program.

```
"mode ← 2; index ← 1;
 while index < ∞ do
 begin i ← 1;
       while G(x_i) = Op_index[H](x_i) do i ← i + 1;
       mode ← 3;
       i ← 1;
       while H(x_i) = Op_index[0](x_i) do i ← i + 1;
       mode ← 2;
       index ← index + 1;
 end;"
```

Let mode_0 be the value of mode upon termination of this program.

(B) *if* $\text{mode}_0 = 2$ *then* $h(x) \leftarrow \epsilon$
else $h(x) \leftarrow g(x)$.

The execution time of this operator is bounded by $c_1 + c_2|x| + |g(x)|$ for some constants c_1 and c_2. Hence, $h \leqslant_{\text{pol}} g$. It remains to show that $0 <_{\text{pol}} h <_{\text{pol}} g$. To show this, it is sufficient to show that index in part (A) of the program for h does not reach a stable value. Suppose it does. Then mode also reaches a stable value, say 3. Then we set $h(x) = g(x)$ for almost all x. Also, since mode reaches the stable value 3, $h(x) = Op_{\text{index}}[0](x)$ for all x. Since h is a finite variant of g we infer $g \leqslant_{\text{pol}} 0$. This contradicts the hypothesis $0 <_{\text{pol}} g$. The case where mode reaches the stable value 2 is treated similarly. ∎

The use of recursion in part (A) of the program can be justified by an application of the recursion theorem. This is done in Section III.

The looking-back technique of the proof of Theorem 3.1 is a refinement of "construction in stages," which is frequently used in recursive function theory (cf. [35]). Note, however, that h is not constructed in stages in the preceding proof. To compute h on input x we do not repeat the computation on all shorter inputs but only on some of them. This constitutes Ladner's refinement [18] of Machtey's density

construction [21]. The actual number of arguments, for which h is recomputed, is not important; we only require that this number becomes arbitrarily large.

Theorem 3.1 is easily extended to show the density of polynomial reducibility. We prove an even stronger result: Every countable partial ordering can be embedded between any two degrees. To the author's knowledge, this result has so far not even been stated for elementary degrees, though it is implicit in the work of Ladner and Machtey.

THEOREM 3.2. *Let a and b be polynomial degrees with $a <_{\text{pol}} b$. Then every countable partial ordering can be embedded into the set $\{c;\ a <_{\text{pol}} c <_{\text{pol}} b\}$ of polynomial degrees between a and b.*

Proof. In this proof we combine ideas developed by Ladner [18], Machtey [21], and Sacks [39]. In [20], Machtey proved the existence of a primitive recursive universal partial ordering, in [21] the existence of an elementary one. The proofs carry over.

Fact 3.3. $\mathscr{P}ol$ contains a universal countable partial ordering, i.e., there is a universal countable partial ordering $\leqslant_R$ whose characteristic function is in $\mathscr{P}ol$.

The next lemma asserts the existence of an antichain between a and b.

DEFINITION 3.4. Let $k\colon \Sigma^* \times \Sigma^* \to \Sigma^*$. k_x denotes the xth cross section of k, $k^{(x)}$ denotes k minus the xth cross section.

$$k_x\colon \Sigma^* \to \Sigma^*, \qquad k_x(y) = k(x, y),$$

$$k^{(x)}\colon \Sigma^* \times \Sigma^* \to \Sigma^*, \qquad k^{(x)}(z, y) = \begin{cases} k(z, y) & \text{if } x \neq z, \\ \epsilon & \text{otherwise.} \end{cases}$$

LEMMA 3.5. *Let f, g be functions such that $f <_{\text{pol}} g$. There exists a function $h\colon \Sigma^* \times \Sigma^* \to \Sigma^*$ such that*

(1) $h <_{\text{pol}} g$,

(2) $f \leqslant_{\text{pol}} h_x$ *for all* $x \in \Sigma^*$,

(3) $h_x \not\leqslant_{\text{pol}} h^{(x)}$ *for all* $x \in \Sigma^*$.

Proof. We construct a function k such that

(1) $k \leqslant_{\text{pol}} g$;

(2) $k_x \not\leqslant_{\text{pol}} f$ for all x, i.e., $k_x \neq Op_i[f]$ for all i;

(3) $g \not\leqslant_{\text{pol}} f$ *join* k, i.e., $g \neq Op_i[f \text{ join } k]$ for all i;

(4) f *join* $k_x \not\leqslant_{\text{pol}} f$ *join* $k^{(x)}$ for all x, i.e., f *join* $k_x \neq Op_i[f \text{ join } k^{(x)}]$ for all i.

Then h with $h_x = f \; join \; k_x$ satisfies the theorem.

$\{x_i\}$ is the enumeration of Σ^* in increasing order, M_f is a Turing machine computing f, and M_g is a Turing machine computing g. The following program computes k. To compute $k(\langle z, y\rangle)$:

(A) Let $n = |\langle z, y\rangle|$. Execute n steps of each of the following programs.

"Compute $M_f(x_1)$, $M_f(x_2)$,... and build up a table F of the values."

"Compute $M_g(x_1)$, $M_g(x_2)$,... and build up a table G of the values."

"Compute $k(x_1)$, $k(x_2)$,... and build up a table K of the values."

"mode $\leftarrow 1$; $l \leftarrow 1$;
while $l < \infty$ *do*
begin $m \leftarrow (l)_1$; $x \leftarrow (l)_2$;

$j \leftarrow 1$;
while $Op_m[F](x_j) = K_x(x_j)$ *do* $j \leftarrow j + 1$;
mode $\leftarrow 2$;
$j \leftarrow 1$;
while $Op_m[F \; join \; K](x_j) = G(x_j)$ *do* $j \leftarrow j + 1$;
mode $\leftarrow 3$;
$j \leftarrow 1$;
while $Op_m[F \; join \; K^{(x)}](x_j) = F \; join \; K_x(x_j)$
do $j \leftarrow j + 1$;
mode $\leftarrow 1$;
$l \leftarrow l + 1$;

end"

Let mode_0 and x_0 be the values of mode and x upon termination of part (A).

(B) *if* $\text{mode}_0 = 1$ *then* output $\leftarrow g(y)$;
if $\text{mode}_0 = 2$ *then* output $\leftarrow \epsilon$;
if $\text{mode}_0 = 3$ *then if* $z = x_0$ *then* output $\leftarrow g(y)$;
else output $\leftarrow \epsilon$;

The running time of this operator is bounded by $c_1 |\langle z, y\rangle| + c_2 + |g(y)|$ for some constants c_1 and c_2. Hence $k \leqslant_{\text{pol}} g$. It remains to show that k satisfies requirements (2), (3), and (4). To show this, it suffices to show that l becomes arbitrarily large. For the sake of contradiction, assume otherwise, say l reaches the stable value l_0. Then mode also reaches a stable value; we consider the case where mode reaches the stable value 3. Let $m = (l_0)_1$ and $x_0 = (l_0)_2$. Since f, g, and k are total functions,

larger and larger tables F, G, and K are computed for longer and longer inputs. Since l reaches the stable value l_0 and mode reaches the stable value 3,

$$Op_m[F \; join \; K^{(x_0)}](x_j) = F \; join \; K_{x_0}(x_j) \qquad \text{for all } j,$$

and hence

$$(*) \qquad Op_m[f \; join \; k^{(x_0)}] = f \; join \; k_{x_0} .$$

Also,

$$k(z, y) = g(y) \qquad \text{if} \quad z = x_0 ,$$
$$= \epsilon \qquad \text{otherwise,}$$

for almost all $\langle z, y\rangle$. Therefore, $k^{(x_0)}(\langle z, y\rangle) = \epsilon$ for almost all $\langle z, y\rangle$ and $k_{x_0}(y) = g(y)$ for almost all y. Combined with (*) this implies $g \leqslant_{\text{pol}} f$.

In the other cases a contradiction is derived similarly. ∎

We now use both lemmas to derive the theorem. Let $a = \deg(f)$ and $b = \deg(g)$ and let h satisfy the preceding lemma. Further, let $M \in \mathscr{P}ol$ be the characteristic function of a universal countable partial ordering $\leqslant_M$ on Σ^*. Define functions c_z for all $z \in \Sigma^*$.

$$c_z(x, y) = \textit{if } x \leqslant_M z \textit{ then } h(x, y) \textit{ else } \epsilon.$$

Then $f \leqslant_{\text{pol}} c_z \leqslant_{\text{pol}} g$ for all z. Furthermore, $c_{z_1} \leqslant_{\text{pol}} c_{z_2}$ if and only if $z_1 \leqslant_M z_2$.

If: If $z_1 \leqslant_M z_2$, then

$$\begin{aligned} c_{z_1}(x, y) &= \textit{if } x \leqslant_M z_1 \textit{ then } h(x, y) \textit{ else } \epsilon \\ &= \textit{if } x \leqslant_M z_1 \textit{ then if } x \leqslant_M z_2 \textit{ then } h(x, y) \\ &\qquad\qquad \textit{else } \epsilon \\ &\qquad \textit{else } \epsilon \\ &= \textit{if } x \leqslant_M z_1 \textit{ then } c_{z_2}(x, y) \textit{ else } \epsilon \end{aligned}$$

hence $c_{z_1} \leqslant_{\text{pol}} c_{z_2}$.

Only if: If $z_1 \not\leqslant_M z_2$, then $c_{z_2}(z_1, y) = \epsilon$ for all y. Thus, $c_{z_2} \leqslant_{\text{pol}} h^{(z_1)}$. Since also $h_{z_1} \leqslant_{\text{pol}} c_{z_1}$, we infer $c_{z_1} \not\leqslant_{\text{pol}} c_{z_2}$. ∎

In [18] Ladner proves some interesting theorems about Cook's notion of polynomial reducibility. Since our notion of polynomial reducibility agrees with Cook's notion in the case of characteristic functions, Ladner's proofs can be adapted to imply the corresponding theorems for our notion. We state the theorems without proof. Theorem 3.6 was obtained independently by Machtey [24].

THEOREM 3.6. *There are polynomial degrees a, b of characteristic functions such that $\mathcal{P}ol <_{\text{pol}} a$, $\mathcal{P}ol <_{\text{pol}} b$ and $c <_{\text{pol}} a$, $c <_{\text{pol}} b$ implies $c = \mathcal{P}ol$.*

THEOREM 3.7. *There is a recursively enumerable sequence $c_1 <_{\text{pol}} c_2 <_{\text{pol}} \cdots$ and a pair a, b of polynomial degrees of characteristic functions such that $c_i \leqslant_{\text{pol}} a, b$ for all i and $d \leqslant_{\text{pol}} a, b$ implies $d \leqslant_{\text{pol}} c_i$ for some i.*

Both theorems require an intricate diagonalization argument.

In [27] we take a different approach to subrecursive degree theory. Many statements about subrecursive degrees can be formulated in the following language. 0 stands for the least degree, the functions $\cup$, $\cap$ denote the join and meet of two degrees, respectively, $\leqslant$ stands for the reducibility, $\vee$, &, $\neg$ denote the standard logical connectives and $(\forall a)$ stands for the quantifier "for almost all a." Using constructive measure theory, we define:

P is true for almost all degrees iff $\{f; P(\deg(f))\}$ has measure 1.

We show that the "almost all" theory of elementary recursive degrees is decidable.

As a by-product, we obtain an interesting result about minimal pairs.

THEOREM 3.8. *Let a be a nonzero elementary recursive degree with $0 < a$. Then there is a nonzero degree b such that $c < a, b$ implies $c = 0$.*

Our proof of Theorem 3.8 relies on the fact that elementary operators are able to search through an interval whose size is exponential in the length of the input. Therefore, our proof cannot be used in the case of polynomial degrees. It is currently open whether the theorem holds for polynomial degrees.

4. *Relative and Absolute Complexity*

So far we have considered only the relative complexity of computable functions. Polynomial reducibility affords us a classification into polynomial degrees, which we studied in the previous section. Computational (absolute) complexity theory provides us with another classification scheme: complexity classes. We will now relate both approaches.

First we define step counting functions. Here we have a choice of definition; we can let them count in either unary or binary (ternary,...). In the second case there would be an exponential gap between the length of the result of a step counting function and the time needed to compute it. However, step counting functions are usually considered to be the prototype of honest functions. Therefore, we decide to let step counting functions count in unary. We want to emphasize that Sections 4 and 5 depend upon this agreement. In [24] Machtey discusses the implications of the other choice of definition.

DEFINITION 4.1. Let M be a Turing machine; say M has $\boldsymbol{n}$ input tapes. A function $T_M: (\Sigma^*)^n \to \{a_1\}^\infty$ (= the set of finite and infinite strings of a_1's) is the running time of M if, for all $x \in (\Sigma^*)^n$, M with input x halts in exactly $|T_M(x)|$ steps. We denote the running time of M by T_M.

Next we state some facts without proofs. Running times are very honest, i.e., they are computable in time equal to their size, time bounded simulation can be done efficiently, and functions are contained in the polynomial classes of all their running times. For the proofs we refer the reader to [20, 22].

Fact 4.2. Let M be any Turing machine. Then there exists a Turing machine P, which computes the running time T_M of M such that $T_P = T_M$.

Fact 4.3. For every Turing machine M there exists a polynomial time bounded Turing machine S_M which on input x, t simulates M on x for exactly $|t|$ steps.

Fact 4.4. Let f be a computable function and let the Turing machine M compute f. Then $f \in \mathscr{P}ol(T_M)$.

We are now able to state the main theorem of this section. Intuitively speaking, it says that the relative complexity of functions is mirrored in their absolute complexity.

THEOREM 4.5. *Let $f \in \mathscr{P}ol(g)$ and let the Turing machine M compute g. Then there is a Turing machine P computing f such that $T_P \in \mathscr{P}ol(T_M)$.*

Proof. This proof differs considerably from the proof of the corresponding theorem in [20, 22].

Let $R[\ ]$ be a polynomial operator which takes g into f. Let the oracle machine R compute the operator $R[\ ]$ and let $G[\ ] \in \mathscr{L}(\ )$ be a bounding expression for R. For every oracle g, the running time of R with inputs g and x is bounded by $|G[g](x)|$.

P is obtained from R by replacing the oracle with the Turing machine M. Whenever R enters its query state q_f, M is used to compute the result of the oracle call. Thus, the running time of P is bounded by the sum of the running time of R and the time spent in actually computing the results of the oracle calls. Since $G[\ ]$ is a bounding expression for R, the first factor of the sum is bounded by $|G[g](x)|$. From Fact 4.4 we know that $g \in \mathscr{P}ol(T_M)$ and hence $G[g] \in \mathscr{P}ol(T_M)$. The second factor is bounded by the sum over all $|T_M(y)|$, where the oracle g is called with argument y during the computation of $R[g](x)$. Since $\mathscr{P}ol(\ )$ is closed under concatenation it therefore suffices to show that:

$$\underset{\substack{\text{oracle } g \text{ is called}\\ \text{with argument } y\\ \text{during the evaluation}\\ \text{of } R[g](x).}}{\text{Concatenate}} \quad T_M(y) \in \mathscr{P}ol(T_M).$$

We proceed in two stages. First we construct an operator $R'[\ ] \in \mathscr{P}ol(\ )$, which computes a list of all y, such that $g(y)$ is called during the computation of $R[g](x)$. $R'[\ ]$ simulates $R[\ ]$. Whenever $R[\ ]$ calls the oracle, $R'[\ ]$ prints the encoded input of the oracle call on its output tape. Since we defined oracle machines such that the oracle input tapes are erased in the event of an oracle call, the following inequality holds for the running times of $R'[\ ]$ and $R[\ ]$.

$$| T_{R'}[g](x)| \leqslant 2 | T_R[g](x)|.$$

$R'[g](x)$ computes a list of all y such that $g(y)$ is called during the computation of $R[g](x)$. This list is fed into an operator $S[\ ]$, which splits the list up into its components and applies the oracle to them. It outputs a list of the results. $S[\ ]$ is obviously a polynomial operator.

$S[\ ]$ and $R'[\ ]$ together allow us to compute

$$\operatorname*{Conc}_{\substack{g(y) \text{ is} \\ \text{called}}} T_M(y),$$

namely,

$$\operatorname*{Conc}_{\substack{g(y) \text{ is} \\ \text{called}}} T_M(y) = S[T_M](R'[g](x)).$$

Since $g \in \mathscr{P}ol(T_M)$ and $\mathscr{P}ol(\ )$ is closed under composition, this establishes

$$\operatorname*{Conc}_{\substack{g(y) \text{ is} \\ \text{called}}} T_M(y) \in \mathscr{P}ol(T_M). \quad \blacksquare$$

5. *Honest Classes*

We now restrict our attention to a subset of the set of all polynomial classes. Honest polynomial classes are those polynomial classes which are generated by running times.

DEFINITION 5.1. (a) $\mathscr{P}ol(f)$ is an honest polynomial class if $\mathscr{P}ol(f) = \mathscr{P}ol(T_M)$ for some running time T_M.

(b) f is honest if $\mathscr{P}ol(f)$ is honest.

Machtey [20, 22] proved many interesting results about honest primitive recursive elementary, etc., classes. The proofs of most theorems rely upon Theorem 4.5. With Theorem 4.5 established for polynomial classes, the proofs carry over. We state the theorems without proofs.

THEOREM 5.2. *f is honest if and only if there is a Turing machine M computing f such that $T_M \in \mathscr{P}ol(f)$.*

For the remainder of this section we also need the concept of a complexity class (cf. [4, 12]).

DEFINITION 5.3. Let $t: \Sigma^* \to \mathbb{N}$ be any function.

$$C_t = \{f; \text{ there is a Turing machine } M \text{ computing } f \text{ such that } |T_M(x)| \leqslant t(x) \text{ for all except finitely many } x\}$$

is the complexity class determined by t.

THEOREM 5.4. *g is honest iff $\mathscr{P}ol(g) = \bigcup_{h \in \mathscr{P}ol(g)} C_{|h|}$.*

The following corollary is just a reformulation of Theorem 5.4.

COROLLARY 5.5. *Let g be honest. Then $f \in \mathscr{P}ol(g)$ if and only if there is a Turing machine M which computes f such that $T_M \in \mathscr{P}ol(g)$.*

Corollary 5.5 is usually called the Ritchie–Cobham property. The next theorem establishes the connection between honest classes and complexity classes.

THEOREM 5.6. *g is honest iff $\mathscr{P}ol(g)$ is a complexity class.*

Next we consider the ordering of honest polynomial classes under set inclusion. $\subset$ denotes proper containment.

THEOREM 5.7. *Let g be honest with $\mathscr{P}ol \subset \mathscr{P}ol(g)$. Then there exists an honest h such that $\mathscr{P}ol \subset \mathscr{P}ol(h) \subset \mathscr{P}ol(g)$.*

Proof. The key idea in this proof is the observation that the density construction given in Section II.3 preserves honesty.

Since g is honest there is a running time T_M such that $\mathscr{P}ol(g) = \mathscr{P}ol(T_M)$. Therefore, we may assume without loss of generality that g is a running time. Then Lemma 4.2 implies the existence of a Turing machine P computing g with $T_P = g$.

Reconsider now the proof of Theorem 3.1. There, a function h is constructed such that $0 <_{\text{pol}} h <_{\text{pol}} g$. In that proof we derived the bound $c_1 + c_2 |x| + |h(x)|$ for the running time of the operator $Op[\ ]$, which takes g into h. If we substitute the program P for the oracle g we obtain the bound $c_1 + c_2 |x| + |h(x)|$ for the running time of a Turing machine which computes h. Hence h is honest. ∎

As in the case of general polynomial classes this theorem is easily generalized to the following.

THEOREM 5.8. *Let f, g be honest functions with $\mathcal{P}ol(f) \subset \mathcal{P}ol(g)$. Every countable partial ordering can be embedded into the set*

$$\{\mathcal{P}ol(h); \mathcal{P}ol(f) \subset \mathcal{P}ol(h) \subset \mathcal{P}ol(g) \text{ and } h \text{ honest}\}$$

of honest classes between $\mathcal{P}ol(f)$ and $\mathcal{P}ol(g)$.

In [22] Machtey showed that the elementary honest classes form a lattice under set inclusion. We consider the corresponding problem for polynomial honest classes.

THEOREM 5.9. *The join of two honest polynomial classes is an honest polynomial class.*

Proof. The construction given in the proof of Theorem 2.1 preserves honesty. ∎

Next we consider the intersection of two honest polynomial classes. We are not able to show that it is always an honest class. Machtey's construction carries over only for a special class of honest functions, namely, level functions.

DEFINITION 5.10. $f: (\Sigma^*)^n \to \Sigma^*$ is a level function if for all $x, y \in (\Sigma^*)^n \mid x \mid = \mid y \mid$ implies $f(x) = f(y)$.

THEOREM 5.11. *Let f and g be honest level functions. Then there exists an honest level function h such that $\mathcal{P}ol(f) \cap \mathcal{P}ol(g) = \mathcal{P}ol(h)$.*

Finally, we quote a theorem of Machtey [24].

THEOREM 5.12 (Machtey). *There are honest polynomial functions f and g such that $\mathcal{P}ol \subset \mathcal{P}ol(f)$, $\mathcal{P}ol \subset \mathcal{P}ol(g)$, and $\mathcal{P}ol(f) \cap \mathcal{P}ol(g) = \mathcal{P}ol$.*

6. *Conclusion*

In this section we introduced polynomial classes and investigated their properties. Two definitions of polynomial time reducibility were given and their equivalence was shown. The definition was related to previous definitions given by Cook [9] and Constable [7], and equivalence in the respective realms was demonstrated. This is evidence that our definition catches the intuitive concept of relative feasibility.

Then many properties of polynomial classes were established. Except for the intersection closure of the class of honest polynomial classes, all results obtained by Ladner [18] and Machtey [20–24] for primitive recursive and elementary classes were proved. Some of the proofs are more complicated because we could not assume monotonicity of the generators.

Finally, we mention that we unified Ladner's and Machtey's density constructions in a way which preserves honesty.

III. Abstract Subrecursive Classes

1. *Introduction*

In this section we attempt an axiomatic treatment of subrecursive classes.

Machtey [20] notes that his results are valid for the relations "elementary recursive in," "primitive recursive in," "doubly recursive in," "triply recursive in," . . ., and "multiple recursive in." Ladner considers complexity determined reducibilities and shows that the results of [18] hold true for all such reducibilities. Two restrictions are inherent in Ladner's approach. The definitions are machine-dependent and only applicable to reducibilities between decision problems. We overcome both limitations.

"A more abstract theory of subrecursive computing systems would clarify the extent of this generality, it would render the whole approach to subrecursive phenomena more palatable. It would also help isolate the critical features of the proofs and constructions" (quoted from [8]).

We develop an abstract notion of subrecursive reducibilities, which includes all the interesting cases that are known, and which is strong enough to imply the density results of Section II.3. We also indicate that the axioms can be applied to situations where one usually would not speak of a reducibility relation. The density of certain natural complexity measures is so inferred.

Subrecursive reducibilities are always induced by a class of general recursive operators. A general recursive operator is a computable operator which is defined on all total functions and takes total functions into total functions [35, p. 149]. For example, in Section II we defined polynomial reducibility in terms of the class of polynomial operators.

Definition 1.1. Let $D = \{x_i\}$ be a recursively enumerable domain, let C be a set of computable functions over D, let S be a class of general recursive operators such that $Op[C] \subseteq C$ for every $Op[\ ] \in S$, and let $f, g \in C$. f is S-reducible to g (we write $f \leqslant_s g$) if there is an operator $Op[\ ] \in S$ such that $f = Op[g]$.

2. *The Axiom System*

We now set up the axiom system. Again, let D be a recursively enumerable domain, let S be a set of general recursive operators, and let C be a set of total computable functions. Further, let $\{\varnothing_i[\ ]\}$ be an acceptable indexing [35a] of the computable operators. In the following, x, y range over D, f, g range over C, and $Op[\ ]$ ranges over S.

Subrecursive classes are always recursively enumerable. Our first axiom ensures that abstract classes have the same property.

Axiom 1. There is a computable function $s: \mathbb{N} \to \mathbb{N}$ such that $S = \{\varnothing_{s(i)}[\ ]; i \in \mathbb{N}\}$.

Thus, s enumerates S. From now on we write $Op_i[\ ]$ instead of $\varnothing_{s(i)}[\ ]$.

The next three axioms are suggested by the axioms for basic recursive function theories (cf. [36, 37]). We require S to contain the identity (the operator APPLY) and to be closed under composition and the conditional.

Axiom 2. For every $f \in C$ there is an operator $\mathrm{APPLY}_f[\ ]$ such that

$$\mathrm{APPLY}_f[f](x) = f(x).$$

Axiom 3. There is a computable function *comp*: $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that

$$Op_{\mathrm{comp}(i,j)}[f](x) = Op_i[\lambda y Op_j[f](y)](x).$$

Axiom 4. There is a computable function *cond* such that

$$Op_{\mathrm{cond}(i,j,k,l)}[f](x) = \textit{if } Op_i[f](x) = Op_j[f](x) \textit{ then } Op_k[f](x) \textit{ else } Op_l[f](x).$$

It is easy to see that Axioms 2 and 3 make $\leqslant_S$ into a reflexive and transitive relation. The S-degree of a function g is $\deg(g) = \{f; f \leqslant_S g \text{ and } g \leqslant_S f \text{ and } f \in C\}$. We extend $\leqslant_S$ by $\deg(f) \leqslant_S \deg(g)$ iff $f \leqslant_S g$ to a partial order on the set of S-degrees.

Assume now that $f \leqslant_S g$. Intuitively speaking, this means that f is simpler than g. This should still be true, if we change either f or g slightly. Redefining a function at finitely many places is certainly a small change. This leads to Axiom 5.

Axiom 5. (a) For every i, g and finite function f there is a j such that

$$Op_j[g](x) = f(x) \quad \text{if} \quad x \in \mathrm{dom}\, f,$$
$$= Op_i[g](x) \quad \text{otherwise.}$$

(b) For every i, g, f such that $g(x) = f(x)$ for all but finitely many x, there is a j such that $Op_j[g] = Op_i[f]$.

Degrees can still be very thin.

EXAMPLE 2.1. Let C be the set of all computable functions, let $\{f_i\}_{i=1}^{\infty}$ be some recursive enumeration of all finite functions (including the empty one), and let $Op_i[g](x) = \textit{if } x \in \text{domain } f_i \textit{ then } f_i(x) \textit{ else } g(x)$. Set $S = \{Op_j[\ ]\}$. This system satisfies Axioms 1 to 5. Apparently, $h \in \deg(g)$ iff $h = g$ almost everywhere.

In Example 2.1 the operators in S are all trivial. They essentially reproduce the argument. Therefore, we need an axiom which forces S to have some minimal computational power. One of the smallest classes of functions considered so far is the

class $\mathscr{E}^0$ [10]. Yet it permits arithmetization of Turing machines; i.e., every recursively enumerable set can be enumerated by a function in $\mathscr{E}^0$. This suggests interpreting minimal computational power as a rudimentary simulation property.

Axiom 6. There is a computable function $sim: \mathbb{N} \to \mathbb{N}$ such that: If $\varnothing_i[f]$ is total, then there is a nondecreasing function (not necessarily computable) l, whose range is D, such that

$$Op_{\mathrm{sim}(i)}[f](x) = \varnothing_i[f](l(x)).$$

We assume the ordering $x_i \leqslant x_j$ if $i \leqslant j$ on D.

PROPOSITION 2.2. *There is a least S-degree, namely,* $\deg(\lambda x[d])$ *for any* $d \in D$. *We denote it by* 0.

Proof. Let d be any element of D. There is a computable operator, say $\varnothing_i[\ ]$, which takes every function $f \in C$ into the constant function $\lambda x[d]$.

$$\varnothing_i[f](x) = d.$$

Axiom 6 implies for every f the existence of an l such that

$$Op_{\mathrm{sim}(i)}[f](x) = \varnothing_i[f](l(x)) = d.$$

Hence $\lambda x[d] \leqslant_s f$ for every $f \in C$. Thus $\deg(\lambda x[d])$ is the least S-degree. ∎

If a reducibility as defined in Definition 1.1 satisfies axioms 1 to 6, then we call it an abstract subrecursive reducibility. All subrecursive reducibilities which are mentioned in the literature are abstract reducibilities in this sense. We state this fact as a theorem; the proof is known and is therefore left to the reader.

DEFINITION 2.3. $\mathscr{R}$ is the set of all computable functions, $\mathscr{R}^*$ is the set of computable sets, $\mathbb{N}$ is the set of natural numbers, and Σ^* is the set of strings over some finite alphabet Σ.

THEOREM 2.4. *Each of the following is an abstract subrecursive reducibility*:

(a) *primitive recursive, doubly recursive,..., reducibility* (*for* $\mathscr{R}$ *and* $\mathbb{N}$),
(b) $\mathscr{E}_n$*-reducibility for* $n \geqslant 1$ (*for* $\mathscr{R}$ *and* $\mathbb{N}$) [10],
(c) *polynomial time reducibility* (*for* $\mathscr{R}$ *and* Σ^*),
(d) $\mathscr{T}$*-time reducibility for time class* $\mathscr{T}$ [18] (*for* $\mathscr{R}^*$ *and* Σ^*),
(e) $\mathscr{S}$*-space reducibility for space class* $\mathscr{S}$ [18] (*for* $\mathscr{R}^*$ *and* Σ^*),
(f) *linear space, polynomial time reducibility* [16] (*for* $\mathscr{R}^*$ *and* Σ^*),
(g) *log-space reducibility* [16] (*for* $\mathscr{R}^*$ *and* Σ^*).

3. *Minimal Degrees*

We begin to study the density properties of abstract subrecursive reproducibilities. By Proposition 2.5 there is a least S-degree 0. We now ask if there is a minimal degree above 0.

THEOREM 3.1. *Let a be an S-degree with $0 <_s a$. Then there is an S-degree b such that $0 <_s b <_s a$.*

Proof. Let $0 = \deg(f)$ and $a = \deg(g)$ for $f, g \in C$. We have to construct an $h \in C$ such that

(1) $h \leqslant_s g$;

(2) $h \not\leqslant_s f$, i.e., $h \neq Op_i[f]$ for all i;

(3) $g \not\leqslant_s h$, i.e., $g \neq Op_i[h]$ for all i.

In Section II.3 we proved the analogous results for polynomial degrees. Keeping the axioms in mind, the reader should now read over that section. He will notice that the same proof can be used again. Only the use of recursion in the program for h has to be justified by an application of the recursion theorem.

We define a computable function e such that range $e \subseteq$ range s. By the recursion theorem (cf. [35]), e has a fixed point; i.e., there is an i_0 such that

$$\varnothing_{e(i_0)}[\ ] = \varnothing_{i_0}[\ ].$$

Since range $e \subseteq$ range s, the fixed-point operator $\varnothing_{i_0}[\ ]$ is an operator in S. $h = \varnothing_{i_0}[g]$ will be the desired function.

Let $\{x_i\}$ be an enumeration of the domain D. We first define a general recursive operator $\varnothing_{e_1(i)}[\ ]$, where i is a free variable. The numbers 1, 2, 3 are distinct elements of the domain D.

$\varnothing_{e_1(i)}[g](x_n)$:

Execute each of the following programs for n steps:

"compute $g(x_1), g(x_2),\ldots$ and build up a table G of the values"

"compute $\varnothing_i[g](x_1), \varnothing_i[g](x_2),\ldots$ and build up a table H of the values"

"compute $f(x_1), f(x_2),\ldots$ and build up a table F of the values"

"use the tables G, H, F computed above: initialize mode $\leftarrow 2$; $j \leftarrow 1$;

while $j < \infty$ *do*

begin $k \leftarrow 1$;

while $H(x_k) = Op_j[F](x_k)$ *do* $k \leftarrow k + 1$;

mode $\leftarrow 3$; $k \leftarrow 1$;

while $G(x_k) = Op_j[H](x_k)$ *do* $k \leftarrow k + 1$;

mode $\leftarrow 2$;

$j \leftarrow j + 1$;

end"

$\varnothing_{e_1(i)}[g](x_n)$ outputs the value of mode upon termination of the last program. We now define

$$\varnothing_{e(i)}[g](x_n) = \textit{if } Op_{\mathrm{sim}(e_1(i))}[g](x_n) = 2 \textit{ then } g(x_n) \textit{ else } f(x_n).$$

f generates the least degree 0; hence, there is an operator $Op_f[\ ] \in S$ such that $f = Op_f[g]$. Furthermore, $g = Op_c[g]$ for some c by Axiom 2, and $\lambda x[2] = Op_2[g]$ by Axiom 4 (cf. the proof of Proposition 2.5). Hence,

$$\varnothing_{e(i)}[g](x_n) = \textit{if } Op_{\mathrm{sim}(e_1(i))}[g](x_n) = Op_2[g](x_n) \textit{ then } Op_c[g](x_n) \textit{ else } Op_f[g](x_n),$$

which by Axiom 5 is

$$= Op_{\mathrm{cond}(\mathrm{sim}(e_1(i)),2,c,f)}[g](x_n) = \varnothing_{s(\mathrm{cond}(\mathrm{sim}(e_1(i)),2,c,f))}[g](x_n).$$

Hence, we can set

$$e(i) = s(\mathrm{cond}(\mathrm{sim}(e_1(i)), 2, c, f)).$$

Thus, range $e \subseteq$ range s. Let i_0 be the fixed point of e, i.e.,

$$\varnothing_{e(i_0)}[\ ] = \varnothing_{i_0}[\ ].$$

Since range $e \subseteq$ range s, there is i_0' such that

$$Op_{i_0'}[\ ] = \varnothing_{e(i_0)}[\ ] = \varnothing_{i_0}[\ ].$$

We set

$$h = Op_{i_0'}[g].$$

It remains to show that h has the desired properties. It satisfies requirement (1) by definition. Let us suppose that either requirement (2) or (3) is not satisfied. Then there is a least i such that either $h = Op_i[f]$ or $g = Op_i[h]$.

Case 1. $h = Op_i[f]$. Since $\varnothing_{i_0}[\,]$ is a general recursive operator, $\varnothing_{i_0}[g](x_n)$ is defined for all $x_n \in D$. Thus, larger and larger tables G, H, and F are computed for larger and larger arguments x of $\varnothing_{e_1(i_0)}[g]$. Hence, for sufficiently large x the value of $\varnothing_{e_1(i_0)}[g](x)$ is equal to 2. Axiom 4 implies the existence of a nondecreasing function l, whose range is equal to D, such that

$$\varnothing_{e_1(i_0)}[g](l(x)) = 2.$$

Therefore $Op_{\mathrm{sim}(e_1(i_0))}[g](x) = 2$ for sufficiently large values of x. So $h(x) = g(x)$ except for finitely many x. Axiom 5(a) together with the assumption $h \leqslant_s f$ implies $g \leqslant_s f$. This contradicts the hypothesis of the theorem.

Case 2. $g = Op_i[h]$. Analogously to Case 1, we argue this time that $\varnothing_{e_1(i_0)}(x) = 3$ for almost all x, and hence, $h(x) = f(x)$ for almost all x. Axiom 5(b) combined with $g \leqslant_s h$ implies $g \leqslant_s f$. Again we have derived a contradiction. ∎

4. *Density*

In this section we extend the result of the previous one. All known reducibilities are upper semilattices. From now on we require abstract reducibilities to be so, too.

Axiom 7. There are constants encode, decode 1, and decode 2 such that

$$Op_{\mathrm{decode1}}[\lambda y Op_{\mathrm{encode}}[f, g](y)](x) = f(x),$$
$$Op_{\mathrm{decode2}}[\lambda y Op_{\mathrm{encode}}[f, g](y)](x) = g(x).$$

PROPOSITION 4.1. *$\leqslant_s$ is an upper semilattice.*

Proof. Let $f, g \in C$ and let $h = Op_{\mathrm{encode}}[f, g]$. h is an upper bound of f and g by Axiom 7.

Let k be any upper bound of f and g, i.e., $f \leqslant_s k$ and $g \leqslant_s k$. Then there are operators $Op_1[\,]$ and $Op_2[\,]$ in S such that $f = Op_1[k]$ and $g = Op_2[k]$, and hence

$$h = Op_{\mathrm{encode}}[Op_1[k], Op_2[k]].$$

Closure under composition (Axiom 3) implies $h \leqslant_s k$. ∎

DEFINITION 4.2. We abbreviate $Op_{\mathrm{encode}}[f, g]$ by *f join g*.

We now prove the density of abstract reducibilities.

THEOREM 4.3. *Let a, b be S-degrees such that $a <_s b$. Then there is an S-degree c such that $a <_s c <_s b$.*

Proof. Let $f, g \in C$ be such that $\deg(f) = a$ and $\deg(g) = b$. We have to construct a function $h \in C$ such that

(1) $h \leqslant_s g$,
(2) $g \not\leqslant_s h$,
(3) $f \leqslant_s h$,
(4) $h \not\leqslant_s f$.

Instead of constructing h directly, we construct a function k with the properties

(1′) $k \leqslant_s g$,
(2′) $g \not\leqslant_s k \; join \; f$,
(4′) $k \not\leqslant_s f$,

and set $h = k \; join \; f$. If k satisfies (1′), (2′), and (4′), then h satisfies (1) to (4). Since we gave a detailed proof of Theorem 3.1 and the technique used there carries over, we leave the remainder of the proof to the reader. ∎

5. *Conclusion*

In Section III we have attempted an axiomatic treatment of subrecursive reducibilities. We inferred two density theorems from the axioms. In fact, even the strong density theorem and (if $\mathscr{R}^* \subseteq C$) Ladner's theorem (3.6) follow from the axioms. Due to space considerations we do not include proofs.

THEOREM 5.1. *Let a, b be S-degrees with $a <_s b$. Every countable partial ordering can be embedded into the set $\{c;\ a <_s c <_s b\}$ of S-degrees between a and b.*

THEOREM 5.2. *If $\mathscr{R}^* \subseteq C$, then there are S-degrees a, b with $0 <_s a$ and $0 <_s b$ such that $c <_s a$ and $c <_s b$ implies $c = 0$.*

The scope of the axioms is not limited to subrecursive reducibilities. In [11] Hartmanis poses the problem whether certain natural complexity measures are dense. In particular, he asks if the measure defined by the number of steps taken by a multitape Turing machine is a dense measure. Assume that two running times t_1 and t_2 are given such that $C_{t_1} \subset C_{t_2}$. We want to construct a running time t such that $C_{t_1} \subset C_t \subset C_{t_2}$. Since the particular measure under consideration is proper

in the technical sense of [11] and the result of a computation is never greater than the number of steps taken, t_1 and t_2 are suitable as witnesses for the propriety of the inclusion. Reconsider now the proof of Theorem 4.3. If the simulation described there can be done in C_{t_1}, then we construct t exactly as k was constructed there. In Theorem 3.2 of Section II we showed how to do the simulation in linear time. Therefore, for sufficiently large t_1 the simulation can be done in C_{t_1}.

THEOREM 5.3. *Let t_1 and t_2 be sufficiently large running times. If $C_{t_1} \subset C_{t_2}$, then there is a running time t such that $C_{t_1} \subset C_t \subset C_{t_2}$.*

It would be interesting to find some set of general properties of measures which imply density. Consider the following class S of operators.

$$\begin{aligned} Op_{\langle i,j,k\rangle}[f](x) = \textit{if } \forall y \leqslant x[y < j \Rightarrow t\phi_i(y) \leqslant k] \\ \& \, [y \geqslant j \Rightarrow t\phi_i(y) \leqslant f(x)] \\ \textit{then } \phi_i(x) \\ \textit{else } 0. \end{aligned}$$

If this set of operators "satisfies" the axiom system of abstract subrecursive reducibilities then the measure is dense. Therefore, one can find a set of properties which imply density by looking for a set which implies the simpler requirements of the axioms. For example, if a measure is proper then for every running time $t\phi_i$, there is an operator $Op_{\langle j,0,0\rangle}[\ ]$ such that $Op_{\langle j,0,0\rangle}[t\phi_i] = t\phi_i$ (Axiom 2).

We were not able to find a subset of the set {finite invariance, proper, parallel computation property,...} of standard properties of measures which implies the axioms. However, slight variants of these properties are strong enough. Unfortunately, these variants are not as natural and not as easily stated. Therefore, we do not list them here.

Some of the most interesting results in subrecursive degree theory are about honest classes. It would be interesting to define abstract honest classes. This would require adding Blum's complexity axioms in a way which is compatible with the simulation axiom (Axiom 4). Because of the close connection between honest classes and complexity classes, this would probably shed some light on the problem of defining "natural" complexity measures.

ACKNOWLEDGMENT

I thank my thesis advisor Professor R. L. Constable for his continuous advise and encouragement.

REFERENCES

1. A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMANN, "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, Mass., 1974.
2. P. AXT, On a subrecursive hierarchy and primitive recursive degrees, *Trans. Amer. Math. Soc.* **92** (1959).
3. J. H. BENNET, On spectra, Doctoral Dissertation, Princeton University, 1962.
4. M. BLUM, A machine-independent theory of the complexity of recursive functions, *J. Assoc. Comput. Mach.* **14** (1967).
5. A. COBHAM, The intrinsic computational difficulty of functions, *in* "Logic, Methodology and Philosophy of Science," Proceedings of 1964 Conference, North–Holland, Amsterdam, 1965.
6. R. L. CONSTABLE, "Mathematical Computing Theory," Prentice–Hall, Englewood Cliffs, N.J., to appear.
7. R. L. CONSTABLE, Type two computational complexity, *in* "Proceedings of the Fifth ACM Symposium on the Theory of Computing" (1973), pp. 108–122.
8. R. L. CONSTABLE AND A. B. BORODIN, Subrecursive programming languages. I. Efficiency and program structure, *J. Assoc. Comput. Mach.* **19** (1972).
9. S. A. COOK, The complexity of theorem proving procedures, *in* "Proceedings of the Third ACM Symposium on the Theory of Computing" (1971).
10. A. GRZEGORCZYK, Some classes of recursive functions, *Rozprawy Mat. Warsaw* **4** (1953).
11. J. HARTMANIS, On the problem of finding natural complexity measures, Cornell University, Technical Report TR73-175.
12. J. HARTMANIS AND J. E. HOPCROFT, An overview of the theory of computational complexity, *J. Assoc. Comput. Mach.* **18** (1971).
13. J. HARTMANIS AND R. E. STEARNS, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* **117** (May 1965).
14. J. E. HOPCROFT AND J. D. ULLMAN, "Formal Languages and Their Relation to Automata," Addison–Wesley, Reading,(Mass., 1969.
15. H. B. HUNT, On the time and tape complexity of languages, I, *in* "Proceedings of the Fifth ACM Symposium on the Theory of Computing" (1973).
16. N. D. JONES, Reducibility among combinatorial problems, Princeton Conference, 1973.
17. S. C. KLEENE, Extension of an effectively enumerated class of functions by enumeration, *Colloq. Math.* **6** (1958).
18. R. E. LADNER, On the structure of polynomial time reducibility, *J. Assoc. Comput. Mach.* **22** (1975), 155–171.
19. N. LYNCH, Relativisation of the theory of computational complexity, Project MAC, TR-99, Ph.D. Thesis; revised version to appear in *Trans. Amer. Math. Soc.* (jointly authored with A. MEYER AND M. FISHER).
20. M. MACHTEY, Augmented loop languages and classes of computable functions, *J. Comput. System Sci.* **6** (1972).
21. M. MACHTEY, On the density of honest subrecursive classes, *J. Comput. System Sci.* **10** (1975), 183–199.
22. M. MACHTEY, The honest subrecursive classes are a lattice, *Inform. Contr.* **24** (1974), 247–263.
23. M. MACHTEY, On a notion of helping, *in* "Proceedings of the 14th IEEE SWAT Conference" (1973).

23a. M. MACHTEY, Helping and the meet of pairs of honest subrecursive classes, *Inform. Contr.* **28** (1975), 76–89.

24. M. MACHTEY, Honesty techniques and polynomial degrees, Project MAC Conference on Concrete Complexity, 1973. (No printed proceedings.)
24a. M. MACHTEY, Minimal pairs of polynomial degrees with subexponential complexity, *Theoretical Computer Science,* to appear.
25. E. M. MCCREIGHT AND A. R. MEYER, Classes of computable functions defined by bounds on computation, *in* "Proceedings of the First ACM Symposium on the Theory of Computing" (1969).
26. K. MEHLHORN, On the size of sets of computable functions, *in* "Proceedings of the 14th IEEE SWAT Conference" (1973).
27. K. MEHLHORN, The "almost all" theory of subrecursive degrees is decidable, *in* "Proceedings of the Second Colloquium on Automata, Languages and Programming," Springer Lecture Notes, Springer–Verlag, New York/Berlin, 1974.
28. A. R. MEYER, Weak monadic second order theory of succesor is not elementary recursive, Project MAC, Technical Memorandum 38, 1973.
29. A. R. MEYER AND D. M. RITCHIE, A classification of the recursive functions, *Z. Math. Logik Grundlagen Math.* **18** (1972).
30. A. R. MEYER AND L. I. STOCKMEYER, Word problems requiring exponential time, *in* "Proceedings of the Fifth ACM Symposium on Theory of Computing" (1973).
31. R. MOLL, Complexity classes of recursive functions, Project MAC TP-110, 1973.
32. A. MOSTOWSKI, Ueber gewisse universelle Relationen, *Ann. Soc. Polon. Math.* **17** (1938).
33. E. POST, Recursively enumerable sets and their decision problems, *Bull. Amer. Math. Soc.* **50** (1944).
34. R. W. RITCHIE, Classes of predictably computable functions, *Trans. Amer. Math. Soc.* **106** (1963), 139–173.
35. H. ROGERS, JR., "Theory of Recursive Functions and Effective Computability," McGraw–Hill, New York, 1967.
35a. H. ROGERS, JR., Gödel numberings of partial recursive functions, *J. Symbolic Logic* **23** (1958), 331–341.
36. H. R. STRONG, Algebraically generalized recursive function theory, *IBM J. Res. Develop.* (Nov. 1968).
37. E. G. WAGNER, Uniformly reflexive structures: An axiomatic approach to computability, *in* "Logic, Computability, and Automata," Proceedings of the Joint RADC–HAC Symposium, Trinkaus Manor, Oriskany, N.Y. (1965).
38. K. WEIHRAUCH, Doctoral Dissertation, Bonn University, 1972.
39. G. E. SACKS, "Degrees of Unsolvability," Princeton University Press, Princeton, N.J., 1963.

Printed by the St Catherine Press Ltd., Tempelhof 37, Bruges, Belgium.