STRAIGHT-LINE PROGRAM LENGTH AS A PARAMETER FOR COMPLEXITY MEASURES*

NANCY A. LYNCH

GEORGIA INSTITUTE OF TECHNOLOGY
Atlanta, Ga. 30332

I. INTRODUCTION

This paper represents a continuation of work in [LB1] and [LB2] directed toward the development of a unified, relative model for complexity theory. The earlier papers establish a simple, natural and fairly general model, and demonstrated its attractiveness by using it to state and prove a variety of technical results. The present paper uses the same model but deals more specifically with the problems involved in stating complexity bounds in a usable closed form for arbitrary operations on arbitrary data types. Work currently in progress is directed toward similar unified treatment of complexity of data structures.

When one thinks of oneself as computing with bit strings or natural numbers, using some simple set of basic operations (such as Turing machine or RAM operation), it is generally easy to express and to understand complexity bounds. The number of basic operations performed is considered to be an order-of-magnitude approximation to the "time" taken by the computation. For convenience, this number is usually presented by a closed-form function t of the length n (or logarithm n) of the input. Since lengths of strings and logarithms of numbers are considered to be natural size measures on their respective domains, they provide natural parameters for measurement of low-level complexity. The principal caution that seems necessary in this circumstance is that the set of basic operations being counted should be clearly specified.

However, if one attempts to understand programs by imposing a modular structure on them where possible, then one does not always want to think of oneself as computing with bit strings or natural numbers, but often with higher-level objects. Similarly, one does not always want to count only basic operations, but often more "complex" operations. Such a method of understanding programs is strongly suggested by the extensive recent research in programming logics, formal semantics, formal specification techniques for data structures and program verification. In this case it is not quite so obvious how to define and how to express complexity bounds.

*This work was partially supported by the National Science Foundation through Grant DCR75-02373.

One first requires a suitable general model for complexity, capable of realistically measuring the complexity of computations performed using either low or high-level operations. Second, one needs a general, natural way of expressing these measurements in a usable closed form. The uniformity of the model and of the complexity statements are important, since modular understanding of complexity-theoretic ideas requires that complexity analysis of components of an algorithm be combinable into complexity analysis of the entire algorithm.

We are thus naturally led to a general model for complexity measurement which is inherently relative [LB1]. Some of the features of the models are the following: Two kinds of modularity are easily expressible -- the definition of a new operation on a previously defined data type, and the representation of an entirely new data type with its associated operations. The model is algebraic in character. In particular, new data types are assumed to be defined up to algebraic isomorphism (although we have not been concerned with particular methods of specifying this definition.) Encodings are not constrained. The point of view taken is that there is an inherent coding-independent relative complexity for data types and their operations, which can be thought of as a trade-off between the complexity of the various operations. The framework used is much closer to models of programming used in other branches of computability theory than is a RAM - Turning machine-style framework.

Ideally, complexity analysis of (function or predicate) modules of a program should be "composable" to yield a complexity analysis of the entire program. Furthermore, the analysis of the program in terms of high-level operations on high-level objects should not require knowledge of the lower level representation of those objects or of the lower level implementation of those operations. In the extreme, an analysis which included, for each input to a program, the exact total number of each type of operation performed on each individual element of the representing domain would satisfy these requirements. But recording all of this information is, of course, not generally feasible. For convenience, one would prefer to express as much information as possible about the analysis by a closed-form function of some numerical parameter on the represented domain.

Hence, the following concern arises. For arbitrary domains, a parameter as natural as length for bit strings and logarithm for natural numbers is required, upon which to base complexity analysis. Moreover, in order to be able to compose analyses, it is preferable that such a parameter be chosen in a uniform way for all domains. So that results about bit strings and natural numbers might be expressible in our framework, the parameter should generalize the length and logarithm measures. The parameter should reside in the represented rather than the representing system. The contention of the present paper is that a simple information-theoretic size measure equivalent to the length of a straight-line program to generate an element is an appropriate parameter.

The insistence that the size parameter reside in the represented rather than the representing system is at odds with previous general work on relative complexity of algebraic systems [M] [C] [CG]. Definitions of the style used in those papers (i.e. parameters in the representing system) are somewhat easier to state than ours, but our feeling that they are less natural is borne out by the difficulties encountered in those papers. Intuitively, we wish to measure the complexity of the accomplishment of a certain task -- the implementation of a new data type. If measure were based on "size" of representing elements, then we would have the odd phenomenon that the task would be deemed "more efficiently accomplished" if only the representing elements were chosen to be of greater size! The actual numerical (time or space) complexity might be unchanged, yet because it is expressed as a function of a larger parameter, a smaller closed-form function might be used. It seems that the only way to make valid comparisons of efficiency of various ways of accomplishing the task is to base the compared measurements on a common parameter, one in the represented system.

A version of this size parameter is used to state the complexity results in [Si]. In [LB1] and [LB2], several results are presented about optimal codings of some basic algebraic systems; these results are also stated in terms of the straight-line program length parameter. Also, a slightly disguised use of this type of parameter appears (for example) in the UNION-FIND and INSERT-MEMBER-DELETE algorithms and lower bounds in [AHU] and [T]. If "dictionaries" [AHU] and other data structures are thought of in the many-sorted algebraic framework of [LZ] [GHM] [ADJ], then the "number of operations" parameter used in the above results can be formally expressed as the size of an element in an appropriate many-sorted algebra. All of these results may be regarded as evidence for the naturalness of the measure, and the present paper is intended to provide further evidence.

Section II contains notation, definitions and basic results for the size measure. Section III classifies several systems by relative "accessibility" complexity, based on the size parameter. Section IV contains several results measuring other types of complexity in terms of the size parameter. Section V re-examines the computable group theory of [Ra], developing a very natural theory of complexity-bounded groups, with complexity once again based on the size parameter. Results in this section seem to be neater and sharper than those obtained in previous attempts at developing such a theory. Much work, however, remains to be done.

II. NOTATION, DEFINITIONS AND BASIC RESULTS

An algebraic system $S = \langle D_S; F_S; P_S \rangle$ is a set D_S (the domain of S) a collection F_S of partial functions (i.e. operations) and a finite collection P_S of partial predicates on that set. Constants are 0-ary functions. Consider the set A of all well-formed terms over the symbols in F_S (including constants). For x in A, let val(x) be the value of x when evaluated in S. (val(x) may be undefined.) Let *Free (S)* be the algebraic system having as its domain the set of x in A for which val(x) is defined, its functions defined as the restrictions to $D_{Free(S)}$ of the usual functions on free algebras, and its predicates defined as follows. Let each basic predicate p on $D_{Free(S)}$ be defined by $p(x_1,\dots,x_n) = p(val(x_1),\dots,val(x_n))$. Note that $p(x_1,\dots,x_n)$ is defined if and only if $p(val(x_1),\dots,val(x_n))$ is defined, and similarly for basic functions.

We are now able to define the size parameter. If A, $B \subseteq D_S$, then $size_S$ (A : B) (the size, in system S, of A relative to B) is defined by:

(a) If $A \subseteq B$, $size_S$ (A : B) = 0.

(b) If $size_S$ (A : B) = k, $f \in F_S$, $x_1,\dots,x_n \in A$ and $size_S (A \cup \{ f (x_1,\dots,x_n)\} : B) \nleq k$, then $size_S (A \cup \{ f (x_1,\dots,x_n)\} : B) = k + 1$.

(c) If $size_S$ (A : B) = k, $C \subseteq A$ and $size_S$ (C : B) $\nleq k - 1$, then $size_S$ (C : B) = k.

(d) $Size_S$ (A : B) is otherwise undefined (and is said to be equal to ∞).

By convention, $n < \infty$ for all $n \in N$, and $\infty \leq \infty$.

In other words, $size_S$ (A : B) describes the number of steps required by a straight-line program, given the values in B, to generate the values in A. We write $size_S$(x : B) for $size_S$ ({x} : B), $size_S$(A) for $size_S(A : \phi)$, and $size_S$(x) for $size_S(\{x\} : \phi)$. This definition is more general than that used, for example, in [Si]. This level of generality was chosen for the naturalness of its use in proofs. Many basic properties can be shown, among which some of those we most frequently use are:

Theorem 1: (a) (Triangle Inequality)

$size_S$ (A : B) + $size_S$ (B : C) $\geq$ $size_S$ (A : C).

(b) (A connection between size in a system and in its associated free system)

$\mathrm{size}_S\,(\mathrm{val}(A)\colon \mathrm{val}\,(B)) \le \mathrm{size}_{Free(S)}\,(A : B)$.

(c) (Another such connection)

If val $(D) = B$, then there exists C with val $(C) = A$ and $\mathrm{size}_S\,(A : B) = \mathrm{size}_{Free(S)}(C{:}D)$.

(d) Let $C \subseteq D_S$, C finite. Then $\mathrm{size}_S(A : B \cup C) \le \mathrm{size}_{\langle D_S;\, F_S \cup C;\, P_S\rangle}(A : B) \le \mathrm{size}_S\,(A : B \cup C) + |C|$.

Proof: Straightforward.

$\boxed{x}$

As in [LB1], let $\tau : A' \to A$ be a partial, onto function, f and f' partial functions on A and A' respectively. Then f' is a simulator of f (with respect to τ) if whenever $f(\tau(x_1),\ldots,\tau(x_n))$ is defined, then so is $\tau(f'(x_1,\ldots,x_n))$ and their values are equal. Similarly, if p and p' are partial predicates on A and A' respectively, then p' is a simulator of p if whenever $p(\tau(x_1),\ldots,\tau(x_n))$ is defined, then so is $p'(x_1,\ldots,x_n)$ and their values are equal.

To measure several different kinds of complexity in terms of the size parameter, we define several different "programming languages." The most general simply measures accessibility of values. A partial function f on D_S is closed if $f(x_1,\ldots,x_n) \in [x_1,\ldots,x_n]$ (the algebraic closure of $x_1,\ldots,x_n$ under the functions in F_S) for every $x_1,\ldots,x_n$ in the domain of f.

We say $S \underset{\tau}{\le}^{\text{closed}} S'$ if every basic function symbol of S has closed τ-simulator over S'. (Predicates are unrestricted, so that only accessibility of values is considered.) Implicit in this reducibility is a set E of expressions which witness the closure property. That is, E assigns an expression to every $(f, x_1,\ldots,x_n)$, where f is one of the basic functions of S, and $(x_1,\ldots,x_n)$ is in the domain of the assumed closed simulator of f. In particular, E assigns an expression to each constant of S. The expressions provided by E may be regarded as rudimentary "programs" to compute the simulators.

Given $S \underset{\tau}{\le}^{\text{closed}} S'$ with E a witnessing set of expressions, it is natural to define a total mapping $\rho\colon D_{Free\,(S)} \to D_{Free\,(S')}$, inductively on the structure of expressions in $D_{Free(S)}$. Briefly, if x is a constant symbol, then $\rho(x)$ is the expression assigned by E to $\mathrm{val}(x)$. If x is an expression of the form $f(x_1,\ldots,x_n)$, where $x_1,\ldots,x_n$ are in $D_{Free(S)}$, then $\rho(x)$ will be the expression assigned by E to f and $(\mathrm{val}(\rho(x_1)),\ldots,\mathrm{val}\,(\rho(x_n)))$, with each $\rho(x_i)$ replacing the corresponding formal variable. Thus, ρ maps sequences of operations over S into the simulating sequences of operations over S', and we have the following diagram commuting:

$$\begin{tikzcd}
Free(S) \arrow[r, "\rho"] \arrow[d, "\mathrm{val}"'] & Free(S') \arrow[d, "\mathrm{val}"] \\
S & S' \arrow[l, "\tau"]
\end{tikzcd}$$

We say $S \underset{\tau}{\le}^{\text{closed}} S'$ with complexity t if $\tau\colon N \to N$ is monotone nondecreasing and unbounded, if $S \underset{\tau}{\le}^{\text{closed}} S'$, with E a witness, ρ as above, and if for every $f \in F_S$, and every $x_1,\ldots,x_n \in D_{Free(S)}$, $\mathrm{size}_{S'}\,(\mathrm{val}(\rho(f(x_1,\ldots,x_n)))\colon \{\mathrm{val}(\rho(x_1)),\ldots,\mathrm{val}(\rho(x_n))\}) \le t\,(\mathrm{size}_{Free(S)}\,(\{x_1,\ldots,x_n\}))$. (In particular, if f is a constant, this definition implies that $\mathrm{size}_{S'}(\mathrm{val}(\rho(f))) \le t(0)$.)

In other words, the "accessibility" of the representing output value from the representing input values (surely a lower bound on "computation time" in any programming language) is bounded as shown. As all of our definitions, this one has the bound expressed in terms of the size of the input values in the represented system, and that size is measured in the associated free system. This allows use of infinitely many representations in S' for each element of S, by permitting longer computation times for those representations which are only reached as a result of the simulation of longer sequences of operations in S. This convention is consistent with the data structure results in [AHU] and [T]. Another point to be noted is that definition of the size measure on sets permits very simple uniform treatment of operations of all arities (including constants) in statements and proofs of theorems.

Since we wish to obtain results by composing bounds, we obtain the following composition result. Although the bound is messy (for instance, not in closed form), it suffices in some cases (in Section III) to yield reasonably good closed form upper and lower bounds.

Theorem 2: Assume $S \underset{\tau}{\le}^{\text{closed}} S'$ with complexity t and $S' \underset{\tau'}{\le}^{\text{closed}} S''$ with complexity t'. Then $S \underset{\tau\circ\tau'}{\le}^{\text{closed}} S''$ with complexity

$$\lambda n \left[\sum_{i=0}^{t(n)-1} t'\left(\sum_{j=0}^{n-1} t(j) + i\right)\right].$$

Proof: An apparently stronger version of the complexity definition is required:

Lemma 3: If $S \overset{\text{closed}}{\underset{\tau}{\leq}} S'$ with complexity t, then there exists E with $S \overset{\text{closed}}{\underset{\tau}{\leq}} S'$ using E as the witnessing set, ρ derived from E as before, and with the following property:

(*) For every $f \in F_S$, and every $x_1,\dots,x_n \in D_{Free(S)}$, $\text{size}_{Free(S')}\ (\rho(f(x_1,\dots,x_n)): \{\rho(x_1),\dots,\rho(x_n)\}) \leq t(\text{size}_{Free(S)}\ (\{x_1,\dots,x_n\}))$.

Proof of Lemma 3: Intuitively, it suffices to choose "shortest possible" expressions. $\boxtimes$

Lemma 4: If $S \overset{\text{closed}}{\underset{\tau}{\leq}} S'$ with complexity t, with E, ρ satisfying (*), then for all $A \subseteq D_{Free(S)}$, $\text{size}_{Free(S')}\ (\rho(A)) \leq \sum_{i=0}^{\text{size}_{Free(S)}\ (A)-1} t(i)$.

Proof: If $\text{size}_{Free(S)}\ (A) = \infty$, the inequality holds by convention. If it is finite, an inductive argument is applied. The Triangle Inequality is used here. $\boxtimes$

Proof of Theorem 2, continued: Use Lemma 3 to obtain E and E' (and corresponding ρ and ρ') for the two hypothesized reducibilities. Then obtain E" (and ρ") for the reducibility $S \overset{\text{closed}}{\underset{\tau o \tau'}{\leq}} S'$ by substitution from E and E'. Note ρ" = ρ'oρ.

Consider f, $x_1,\dots,x_n$. If $f(x_1,\dots,x_n)$ is not defined, then our conventions cause the result to be true, so assume $f(x_1,\dots,x_n)$ is defined. $\text{size}_{Free(S')}\ (\rho(f(x_1,\dots,x_n)): \{\rho(x_1),\dots,\rho(x_n)\}) \leq t(\text{size}_{Free(S)}\ (\{x_1,\dots,x_n\})$ by hypothesis. Thus, there exists a sequence $A_0,\dots,A_\ell$ of sets, $0 \leq \ell \leq t(\text{size}_{Free(S)}\ (\{x_1,\dots,x_n\})$, with $A_0 = \{\rho(x_1),\dots,\rho(x_n)\}$, $\rho(f(x_1,\dots,x_n)) \in A_\ell$ and $\text{size}_{Free(S')}\ (A_{i+1} : A_i) = 1$ for all i, $0 \leq i \leq \ell - 1$. By Lemma 4, $\text{size}_{Free(S')}\ (A_0) \leq \sum_{j=0}^{\text{size}_{Free(S)}\ (\{x_1,\dots,x_n\})-1} t(j)$. By the Triangle Inequality, $\text{size}_{Free(S')}\ (A_i) \leq \sum_{j=0}^{\text{size}_{Free(S)}\ (\{x_1,\dots,x_n\})-1} t(j) + i$, for each i, $0 \leq i \leq \ell$.

Now $\text{size}_{S''}\ (\text{val}(\rho''(f(x_1,\dots,x_n))): \{\text{val}(\rho''(x_1)),\dots,\text{val}(\rho''(x_n))\}) \leq \text{size}_{Free(S'')}\ (\rho''(f(x_1,\dots,x_n)): \{\rho''(x_1),\dots,\rho''(x_n)\})$, by Theorem 1(b), $= \text{size}_{Free(S'')}\ (\rho'(\rho(f(x_1,\dots,x_n))) : \{\rho'(\rho(x_1)),\dots,\rho'(\rho(x_n))\}) \leq \text{size}_{Free(S'')}\ (\rho'(A_\ell) : \rho'(A_0))$.

For each i, $0 \leq i \leq \ell - 1$, obtain f_i, an n_i - ary function in $F_{S'}$, with $A_{i+1} = A_i \cup \{f_i(y_{i_1},\dots,y_{i_{n_i}})\}$, where the y's are in A_i. Then the last expression is $\leq \sum_{i=0}^{\ell-1} \text{size}_{Free(S'')}\ (\rho'(f_i(y_{i_1},\dots,y_{i_{n_i}})) : \rho'(A_i)) \leq \sum_{i=0}^{\ell-1} \text{size}_{Free(S'')}\ (\rho'(f_i(y_{i_1},\dots,y_{i_{n_i}})) : \{\rho'(y_{i_1}),\dots,\rho'(y_{i_{n_i}})\})$

$\leq \sum_{i=0}^{\ell-1} t'(\text{size}_{Free(S')}(\{y_{i_1},\dots,y_{i_{n_i}}\}))$ by hypothesis, $\leq \sum_{i=0}^{\ell-1} t'\ (\text{size}_{Free(S')}\ (A_i))$

$\leq \sum_{i=0}^{\ell-1} t'(\sum_{j=0}^{\text{size}_{Free(S)}(\{x_1,\dots,x_n\})} t(j) + i)$

$\leq \sum_{i=0}^{t(\text{size}_{Free(S)}\ (\{x_1,\dots,x_n\})) - 1} \cdot t'(\sum_{j=0}^{\text{size}_{Free(S)}(\{x_1,\dots,x_n\})} t(j) + i)$, as needed. $\boxtimes$

Note that the above definition for complexity-bounded reducibility uses the size measure in a dual role--as parameter and also as the complexity being measured. Although all work in this paper uses the size measure in the free system as the parameter for complexity measurement, many different kinds of complexity besides accessibility can be expressed in terms of that parameter. For instance, we define $S \overset{\text{tree}}{\underset{\tau}{\leq}} S'$ with complexity t if each operation of S has an infinite tree scheme over S' computing a simulator, if E is a corresponding collection of expressions (constructed from the tree programs in a straightforward way), if ρ is constructed from E as before, and if each (function or predicate) simulator program F satisfies

$L_F(\text{val}(\rho(x_1)),\dots, \text{val}(\rho(x_n))) \leq t(\text{size}_{Free(S)}\ (\{x_1,\dots x_n\}))$.

(L_F is the number of steps executed by flowchart F). Entirely analogously, we define $S \underset{\tau}{\leq} S'$ with complexity t for finite flowcharts. Versions of Theorem 2 are true for these reducibilities. (The style of these definitions is more closely akin to property (*) of Lemma 3 than to the original definition of complexity-bounded reducibility.) In addition, we define $S \overset{\text{closed}}{\underset{\tau}{\leq}} S'$ with time-size complexity t, s if t, s: N→N are monotone nondecreasing and unbounded, if $S \overset{\text{closed}}{\underset{\tau}{\leq}} S'$ with complexity t, E and ρ witnesses to the reducibility definition, and if for all n, x, $|\{\text{val}(\rho(y)): \text{size}_{Free(S)}\ (y) \leq n \text{ and val } (y)=x\}| \leq s(n)$. In other words, the time and number of representations are both bounded in terms of our parameter. (It may be seen that val(ρ(y)) is really dependent only on y, τ and the given simulators, and independent of the particular E and ρ used.) Similarly, many other kinds of complexity can be expressed.

III. ACCESSIBILITY

In this section, our most general reducibility definition is used to classify simple arithmetic and string processing systems. We attempt to conclude all that we can about the relative complexity of these systems using "accessibility" considerations only. Of course, relative complexity of systems will usually depend on more than the accessibility of values.

The first remark states, roughly, that whether or not upper bounds on accessibility are also upper bounds on flowchart complexity can depend entirely on the choice of predicates in a system.

Theorem 5: Assume S and S' are systems with no predicates, and with only 0-ary and unary functions. Assume $S \underset{\tau}{\le}^{\text{closed}} S'$ with complexity t. Assume further that $\langle N;\ 0,\ \text{succ}\rangle \underset{\tau'}{\le} S'$ with constant complexity. (This last assumption just means that it is possible systematically to generate new elements of $D_{S'}$.) Then there is a set P of predicates over $D_{S'}$, one for each unary function in F_S, such that $S \underset{\tau}{\le} \langle D_S;\ F_S;\ P\rangle$ with complexity ct for some constant c.

For, the predicates in P can be chosen to give answers which correctly guide the computation of the corresponding functions by flowcharts.

On the other hand, the accessibility lower bounds are a priori lower bounds on flowchart running time. We examine seven systems representing a variety of rates of neighborhood size growth, and differing in directedness and in numbers and arities of operations. All systems in the remainder of this section have functions only, and no predicates.

(a) $\langle N;0,\ \text{succ}\rangle$ (where $\text{succ}(x) = x+1$),

(b) $\langle Z;0,\ \text{succ},\ \text{pred}\rangle$ (where $\text{pred}(x) = x-1$),

(c) $\langle N+Ni;\ 0,\ \text{succ},\ \text{isucc}\rangle$ (where $\text{isucc}(x) = x+i$),

(d) $\langle Z+Zi;\ 0,\ \text{succ},\ \text{pred},$
$\quad \text{isucc},\ \text{ipred}\rangle$ (where $\text{ipred}(x) = x-i$),

(e) $\langle \{0,1\}^*;\ \lambda,\ 0\text{succ},\ 1\text{succ}\rangle$,

(f) $\langle \{0,1\}^*;\ \lambda,\ 0\text{succ},\ 1\text{succ},\ \text{pred}\rangle$,

(g) $\langle N;0,1,+\rangle$.

For each pair of systems, bounds on relative accessibility complexity are calculated. Reasonably close upper and lower bounds are obtained in all cases. Results are summarized in the following table.

	(a)	(b)	(c)	(d)	(e)	(f)	(g)
(a)	1	1	1	1	1	1	1
(b)	cn	1	1	1	1	1	3
(c)	cn	cn	1	1	1	1	3
(d)	cn^2	cn	cn	1	2	2	6
(e)	$c.2^n$	$c.2^n$	$c.2^{\frac{n}{2}}$	$c.2^{\frac{n}{2}}$	1	1	3
(f)	$c.2^n$	$c.2^n$	$c.2^{\frac{n}{2}}$	$c.2^{\frac{n}{2}}$	2	1	5
(g)	$c.2^n$	$c.2^n$	$c.2^{\frac{n}{2}}$	$c.2^{\frac{n}{2}}$	cn	cn	1

The function t in the box in the row corresponding to S and the column corresponding to S' is such that $S \le^{\text{closed}} S'$ with complexity t. c indicates a constant, which may be different for different boxes. Some upper bounds are calculated directly, but others use Theorem 2. For instance,

(b) in (c): Use $\tau(x + iy) = x - y$ if $x \ge y$, undefined otherwise. (Other bounds in the upper triangle are similarly easy.)

(c) in (a): The pairing function in [Y] suffices. (Then note that (b) in (a) and (c) in (b) follow by Theorem 2.)

(d) in (a): By a "spiral numbering".

(d) in (b): Again by a "spiral numbering", but this time values can be accessed quickly from preceding values rather than always from 0. ((d) in (c) now follows.)

(e) in (c): Correspond strings in a natural order with elements of N+Ni in order of increasing sum of real and imaginary coefficients, and where equal, in order of increasing imaginary part. Measure accessibility from 0. (Now not only (e) in (d), but also (e) in (a) and (e) in (b) follow by Theorem 2. Also, (f) is handled similarly.)

(g) in (c): Again use the pairing function in [y], this time to represent elements of N by elements of N+Ni. For the simulation of +, note that if $\text{size}_{Free(g)}(\{x,y\}) = n$, then $\text{size}_{Free(g)}(\{x+y\}) = n+1$, and $\text{val}(x+y) \le 2^n$. Then $\text{size}_{(c)}(\text{val}(\rho(x+y))) \le c.\ 2^{\frac{n}{2}}$ for some c, which suffices. ((g) in (a), (b) and (d) follow.)

(g) in (e): Similar to the preceding case, using a natural coding. ((g) in (f) then follows.)

Next, an i.o.(infinitely-often) style lower bound definition will be formulated. With this definition, we can show that all of the non-constant functions above, with one exception, are also lower bounds for the designated coding. The one exception is (d) in (c), where the best lower bound we have is of the form $cn^{1/2}$ rather than cn. Techniques are combinatorial, using size of neighborhood arguments similar to those used in [Ro], with several complications introduced by the fact that a system's constants are always available in one step, by different arities and by "directionality" considerations. Again, some results are proved directly and others via Theorem 2.

More specifically, we use:

Definition: t is a lower bound for $S \leq^{\text{closed}} S'$ if $t{:}N \rightarrow N$ is total, monotone nondecreasing and unbounded, and for any $t^1{:}N \rightarrow N$, with $t^1 = t$ a.e. (on all but possibly finitely many arguments) it is false that $S \leq^{\text{closed}} S'$ with complexity t^1.

Theorem 6: t is a lower bound for $S \leq^{\text{closed}} S'$ iff t is total, monotone nondecreasing and unbounded and the following condition holds: For any τ, E, ρ having $S \leq_{\tau}^{\text{closed}} S'$ with E, ρ as witnesses, $(\overset{\infty}{\exists} n)$ $(\exists f \in F_S)$ $(\exists x_1,\ldots,x_m \in D_{Free(S)})$ $[\text{size}_{Free(S)}$ $(\{x_1,\ldots,x_m\}) \leq n$ and $f(x_1,\ldots,x_m)$ is defined and size $(\text{val}(\rho(f(x_1,\ldots,x_m))) : \{\text{val}(\rho(x_1)),\ldots,$ $\text{val}(\rho(x_m))\}) > t(n)$. (In words, any coding of S in S' has infinitely many arguments on which the given bound is exceeded.)

Proof: By finite patching techniques. $\boxed{x}$

The function t in the box in the row corresponding to S and the column corresponding to S' in the above diagram, with the exception of (d) in (c), is such that t is a lower bound for $S \leq^{\text{closed}} S'$. c indicates a nonzero constant, which may be different for different boxes. For (d) in (c), the appropriate box should contain $cn^{1/2}$. The form of the lower bound definition was chosen so that Theorem 2 could be applied for the proof of some of the lower bounds.

For instance,

(b) in (a): We prove this bound as a separate theorem, since it is fairly complicated and since its proof ideas are also used in the proofs of other lower bounds. It is a typical example of a coding - independent trade-off lower bound for embedding a two-directional system in a similar one-direction system:

Theorem 7: Let $c = \frac{3}{4} - \frac{\sqrt{2}}{2}$ $(\approx .043)$. Let $S = \langle Z;0,$ $\text{succ},\text{pred}\rangle$, $S' = \langle N;0,\text{succ}\rangle$. Then $\lambda n\ [cn]$ is a lower bound for $S \leq^{\text{closed}} S'$. Furthermore, in the notation of Theorem 6, n and x $(\in D_{Free(S)})$ can be chosen with $\text{val}(x) \geq \frac{n}{2}$. (In other words not only infinitely many different elements of $D_{Free(S)}$) but in fact infinitely many different elements of D_S are involved in the inequality.)

Proof sketch:

Use is made of the fact that, if the given function were not a lower bound, then for sufficiently many $x \in D_{Free(S)}$, succ (pred(x)) and x would have different val $\circ$ ρ - images in S'. Then neighborhood size considerations suffice to give the needed result. Although the basic ideas are fairly simple, some detailed care must be taken because of the easy accessibility of 0 in S'. Thus, the details become fairly complicated, as might be guessed from the appearance of the value of c.

In more detail, consider any $a \in Z$, $a > \frac{1}{2c} - 1$, $b > a$, $b > \text{size}_{S'}$ $(\text{val}(\rho(\text{succ}^{(z)}(0))))$ for all z, $0 \leq z \leq a+1$, where $\text{succ}^{(z)}$ (0) is shorthand for $\underbrace{\text{succ}(\ldots(\text{succ}(0))\ldots)}_{z}$. Write $A = \{z \in D_{Free(S)}:$ $\text{size}_{Free(S)}$ $(z) \leq 2b$ and $\text{val}(z) \geq b$ and $\text{succ}^{(b)}(0)$ is a subexpression of z$\}$. We will prove that at least one of (*), (**) must hold:

(*) For some z in A, $f \in \{\text{succ, pred}\}$, $\text{size}_{S'}$ $(\text{val}(\rho(f(z))) : \ \text{val}(\rho(z))) > 2cb$,

(**) For some w, $a < w < b$, $\text{size}_{S'}$ $(\text{val}(\rho(\text{succ}^{(w+1)}$ $(0))) : \ \text{val}(\rho(\text{succ}^{(w)}(0)))) > 2cb$.

Once we have done this, the theorem follows. Assume (**) fails. Then by the Triangle Inequality, we can show $\text{size}_{S'}$ $(\text{val}(\rho(\text{succ}^{(b)}(0)))) < b +$ $(b-(a+1))$. $2cb < 2cb^2$.

There are two possible cases:

(***) For some z in A, $f \in \{\text{succ, pred}\}$, $\text{size}_{S'}$ $(\text{val}(\rho(f(z)))) > 2cb$ and $\text{val}(\rho(f(z))) < \text{val}(\rho(z))$.

In this case, since in S', if $r < s$, then $\text{size}_{S'}$ $(r{:}s) = \text{size}_{S'}$ (r), (*) holds.

(****) For all z in A, f in {succ, pred}, if $\text{size}_{S'}$ $(\text{val}(\rho(f(z)))) > 2cb$, then $\text{val}(\rho(f(z))) >$ $\text{val}(\rho(z))$. (Note that equality cannot hold.)

Now $|\{z \in N : \text{size}_{S'}$ $(z) \leq 2cb\}| \leq 2cb$, so $|\tau(\{z \in N : \text{size}_{S'}$ $(z) \leq 2cb\})| \leq 2cb$. $|\text{val}\ (A)| =$ b. Thus if $B = \text{val}(A) - \tau(\{z \in N : \text{size}_{S'}$ $(z) \leq$ $2cb\})$, then $|B| \geq b - 2cb$. B has the properties that $[(z \in A$ and $\text{val}(z) + 1 \in B) \Rightarrow \text{val}(\rho(\text{succ}(z))) >$ $\text{val}(\rho(z))]$, and $[(z \in A$ and $\text{val}(z) - 1 \in B) \Rightarrow \text{val}(\rho(\text{pred}$ $(z))) > \text{val}(\rho(z)))]$. (For instance, assume $z \in A$, $\text{val}(z) + 1 \in B$. Then $\tau(\text{val}(\rho(\text{succ}(z)))) = \text{val}(\text{succ}$ $(z)) = \text{val}(z) + 1 \notin \tau(\{z : \text{size}_{S'}$ $(z) \leq 2cb\})$, so $\text{size}_{S'}$ $(\text{val}(\rho(\text{succ}(z)))) > 2cb$, so by the defining condition for this case, $\text{val}(\rho(\text{succ}(z))) > \text{val}(\rho$ $(z))$.) But then there exists $C \subseteq B$, $|C| \geq (1-4c)b$ and $[z \in C \Rightarrow z+1$ or $z - 1 \in C]$. (Simply take $C =$ $\{z \in B : z+1$ or $z-1 \in B\}$).

Now it can be shown, because of the distinctness of so many val $\circ\rho$ - values as shown above, that $|\text{val}(\rho(A \cap \text{val}^{-1}(C)))| \geq (\frac{1}{2} - 2c)^2\ b^2$. But then clearly there must exist (by neighborhood size) some z in $\text{val}(\rho(A \cap \text{val}^{-1}(C)))$ with $\text{size}_{S'}$ $(z) \geq$ $(\frac{1}{2} - 2c)^2\ b^2$.

Consider $w \in A$ with $\text{size}_{S'}$ $(\text{val}(\rho(w))) \geq (\frac{1}{2} -$ $2c)^2 b^2$. Then $\text{size}_{S'}$ $(\text{val}(\rho(w))) \leq \text{size}_{S'}$

$(\text{val}(\rho(w)) : \text{val}(\rho(\text{succ}^{(b)}(0)))) + \text{size}_{S}$, $(\text{val}(\rho(\text{succ}^{(b)}(0))))$. Thus, $(\frac{1}{2} - 2c)^2 b^2 \leq \text{size}_{S}$, $(\text{val}(\rho(w)) : \text{val}(\rho(\text{succ}^{(b)}(0)))) + 2cb^2$. The Triangle Inequality is applied several times to expand the right-hand side by successive substrings of w, all in A. There are at most b-1 terms, so one is greater than $(\frac{1}{4} - 4c + 4c^2)b = 2cb$ by choice of value of c. That is, for some u, $v \in A$ with v = pred(u) or v = succ(u), size_{S}, $(\text{val}(\rho(v)) : \text{val}(\rho(u))) > 2cb$. $\boxed{x}$

<u>(c) in (b)</u>: Let $c = \frac{1}{2}$, and consider any n. There are $\frac{(n + 1)\ (n + 2)}{2}$ elements of $D_{(c)}$ with $\text{size}_{(c)} \leq n + 1$. One of these x, must be such that $\text{size}_{(b)}\ (y : \text{val}(\rho(0)) \geq \frac{(n + 1)\ (n + 2)}{2} - 1$, for all y with $\tau(y) = x$. In particular, these exists $z \in D_{Free(c)}$ with $\text{val}(z) = x$, $\text{size}_{Free(c)}(z) \leq n+1$ and $\text{size}_{(b)}\ (\text{val}(\rho(z)) : \text{val}(\rho(0))) \geq \frac{(n + 1)\ (n + 2)}{2} - 1$. By the Triangle Inequality, there exists $w \in D_{Free(c)}$, $\text{size}_{Free(c)}(w) \leq n$, and either $\text{size}_{(b)}$ $(\text{val}(\rho(\text{succ}(w))) : \text{val}(\rho(w)))$ or $\text{size}_{(b)}$ $(\text{val}(\rho(\text{succ}(w))) : \text{val}(\rho(w))) > \frac{n}{2}$. (Note (c) in (a) and (d) in (b) follow from this case, the given <u>upper</u> bounds and Theorem 2.

<u>(d) in (c)</u>: A proof similar to that for (b) in (a) shows that $cn^{\frac{1}{2}}$ is a lower bound for $(d) \leq^{\text{closed}}$ (c) for some nonzero constant c. As before, infinitely many different elements of D_{S} are involved in the inequality. The counting arguments for this case are more complicated than those in the preceding case.

<u>(d) in (a)</u>: Similar to (b) in (a) and (d) in (c).

<u>(e) in (b)</u>: By neighborhood size. Say $(e) \leq^{\text{closed}}_{\tau}$ (b) with complexity t, where $t(n) = c.2^n$ for some $c < 1$, and almost all n.

There are $2^{n+1} - 1$ elements $x \in \{0,1\}^*$ with $\text{size}_{(e)}(x) \leq n+1$. Therefore, there is some $x \in D_{Free(e)}$, $\text{size}_{Free(e)}(x) \leq n+1$, and $\text{size}_{(b)}\ (\text{val}(\rho(x)): \text{val}(\rho(\lambda))) \geq 2^{n+1} - 2$. By the given upper bound and the Triangle Inequality, $\text{size}_{(b)}\ (\text{val}(\rho(x)): \text{val}(\rho(\lambda))) \leq \sum_{i=1}^{n} t(i)$, which is strictly less than $2^{n+1} - 2$ for sufficiently large n, a contradiction.

(Now (f) in (b), (e) in (a) and (f) in (a) follow. Moreover, with a careful use of Theorem 2, giving proper attention to the finitely many allowed exceptions to lower bounds, (e) in (d), (f) in (d), (e) in (c) and (f) in (c) can be shown.)

<u>(g) in (b)</u>: Similar to (e) in (b), using $\text{val}(\rho(1))$ in place of $\text{val}(\rho(\lambda))$. (Then (g) in (a), (c) and (d) follow.)

<u>(g) in (e)</u>: This case gives some suggestions for proving relative coding bounds with operations of arities ≥ 2. It suffices to show that for arbitrarily large n, there exist x, y with $\text{size}_{Free(g)}\ (\{x,y\}) \leq n$ and $\text{size}_{(e)}\ (\text{val}(\rho(x+y)): \{\text{val}(\rho(x)), \text{val}(\rho(y))\}) > \frac{n}{8}$. Assume not, and fix n, a sufficiently large multiple of 8. For each u, $\text{size}_{(g)}\ (u) \leq \frac{n}{2}$, select $x_u \in D_{Free(g)}$, $\text{size}_{Free(g)}(x_u) \leq \frac{n}{2}$, $\text{val}(x_u) = u$. There are at least $2^{\frac{n}{4}}$ such $u \in N$.

For any of the above u, there are at most $2^{\frac{n}{8}}$ values $v \neq u$, $\text{size}_{(g)}\ (v) \leq \frac{n}{2}$ and $\text{size}_{(e)}$ $(\text{val}(\rho(x_u + x_v)) : \text{val}(\rho(x_u)) \leq \frac{n}{8}$. Since all such v have $\text{size}_{(e)}\ (\text{val}(\rho(x_u+x_v)) : \{\text{val}(\rho(x_u)), \text{val}(\rho(x_v))\} \leq \frac{n}{8}$ (by hypothesis), it follows that at least $2^{\frac{n}{4}} - 2^{\frac{n}{8}} - 1$ values $v \neq u$ with $\text{size}_{(g)}\ (v) \leq \frac{n}{2}$, and $\text{size}_{(e)}\ (\text{val}(\rho(x_u + x_v)) : \text{val}(\rho(x_v))) \leq \frac{n}{8}$. (There are only unary operations in $F_{(e)}$.)

Define partial $\alpha : (D_{Free(g)})^2 \rightarrow D_{Free(g)} \times \{0,1\}^*$. α assigns to (x_u, x_v) (with $u \neq v$) a pair (a,b) such that $a = x_u$ or x_v, such that if $a=x_u$, then $\text{size}_{(e)}\ (\text{val}(\rho(x_u + x_v)): \text{val}(x_u)) \leq \frac{n}{8}$, and if $a = x_v$, then $\text{size}_{(e)}\ (\text{val}(\rho(x_u + x_v)): \text{val}(x_v)) \leq \frac{n}{8}$. Also, b gives the sequence of operations applied to $\text{val}(\rho(x_u))$ or $\text{val}(\rho(x_v))$ (depending on whether $a = x_u$ or $a = x_v$) to obtain $\text{val}\ (\rho(x_u + x_v))$. The length of b is at most $\frac{n}{8}$.

For each x_u, at least $2^{\frac{n}{4}} - 2^{\frac{n}{8}} - 1$ of the x_v have the property that $\alpha(x_u,x_v) = (x_v,b)$ for some b. Moreover, if $\alpha(x_u,x_v) = \alpha(x_{u'},x_{v'})$, then either x_v is one of the $\leq 2^{\frac{n}{8}}$ exceptions for x_u, or $x_{v'}$ is one of the $\leq 2^{\frac{n}{8}}$ exceptions for $x_{u'}$, or else $x_v = x_{v'}$, and the same sequence of operations can be applied to $\text{val}(\rho(x_v))$ to obtain both $\text{val}(\rho(x_u + x_v))$ and $\text{val}(\rho(x_{u'} + x_v))$. But then $\text{val}(\rho(x_u + x_v)) = \text{val}(\rho(x_{u'} + x_v))$, so $\tau(\text{val}(\rho(x_u + x_v))) = \tau(\text{val}(\rho(x_{u'} + x_v)))$, so $\text{val}(x_u + x_v) = \text{val}(x_{u'} + x_v)$, so $u + v = u' + v$, so $u = u'$. In other words, α is "almost" 1-1.

A counting argument now shows $|\ \text{range}\ \alpha\ | \geq 2^{\frac{n}{4}}(2^{\frac{n}{4}} - 2^{\frac{n}{8}} - 1)$. But by definition of α ,

$|\text{range}\,\alpha| \leq 2^{\frac{n}{4}}(2^{\frac{n}{8}}+1)$, a contradiction. (Now (g) in (f) follows.)

IV. OTHER COMPLEXITY MEASURES

In this section, we treat types of complexity other than accessibility, in terms of the size parameter.

There is clearly a close relationship between the questions treated in Section III and those considered in various studies of "graph embedding" problems ([Ro] [LED1] [LED2] and others). The framework used here is somewhat more general than the graph-theoretic framework. Also, our style of question is slightly different from the style of question in those studies, as we are interested in the inherent complexity involved in mapping paths (computation sequences) into paths, bounding "path dilation" in terms of path length rather than uniformly.

Noting that several of the efficient codings in III ((d) in (f), (b) in (c) and (f) in (e)) seem to involve multiple representations, we here use the definition given at the end of Section II to attempt to quantify this tradeoff. A similar tradeoff behavior is the subject of study of [LED1] [LED2], in a setting of finite graphs and uniform bounds. By re-casting their definition in our notation, we can show a connection between the two style definitions. Namely,

Lemma 8: Assume $\mathcal{S} \underset{\tau}{\leq}^{\text{closed}} \mathcal{S}'$ with time-size complexity f, g, with E, ρ as witnesses. Assume $n_0 \varepsilon N$ satisfies $n_0 > |\text{val}(\rho(\{y \varepsilon D_{Free(\mathcal{S})} : \text{size}_{Free(\mathcal{S})}(y) \leq n_0\}))|$.
Assume s, tεN with $f(mn_0) \leq t$, where m is the maximum arity of a function in $F_{\mathcal{S}}$, and $g(n_0) \leq s$.

Then $\mathcal{S} \underset{\tau}{\leq}^{\text{closed}} \mathcal{S}'$ with time-size complexity $\lambda n(t)$, $\lambda n(s)$.

(In other words, if a (finite) system $\mathcal{S}$ has a bound on the dilation and size increase for paths of sufficiently great length n_0, it is sometimes possible to conclude a uniform bound for simulation of arbitrary paths through $\mathcal{S}$.)

For the embedding of (d) in (f), this relationship and the main ancestor tree result of [LED2] imply the following rather weak result.

Theorem 9: Let $\mathcal{S}$ = (d), $\mathcal{S}'$ = (f). Assume $\mathcal{S} \underset{\tau}{\leq}^{\text{closed}} \mathcal{S}'$ with time-size complexity f, g. Assume N, n are such that $N \geq (2n-1)^2 .\ 2^n$.

Then loglog g(N) + f(N) $\geq$ log n − c, c a fixed constant.

In other words, for sufficiently long paths, a tradeoff similar to that in [LED2] holds. It does seem, however, that this corollary should not be the strongest result obtainable. We conjecture, but have thus far been unable to prove,
Conjecture: For $\mathcal{S}$, $\mathcal{S}'$, f, g as in Theorem 9, log log g(n) + f(n) $\geq$ log n−c, c a fixed constant.

That is, just embedding the paths of length n through a rectangular grid into the paths of a binary tree seems to require the above tradeoff. We have not been able to apply the "boundary techniques" of [LED2] directly to obtain this sharper result, however. The best we have obtained is the proof of the degenerate case involving single representations only:

Theorem 10: For $\mathcal{S}$, $\mathcal{S}'$, f, g as in Theorem 9, with g=λn[1], it follows that f(n) $\geq$ log n−c, c a fixed constant.

Proof: A version of a boundary argument in [LED1] is carefully redone in terms of the size parameter. $\boxtimes$

For the other two cases, (b) in (c) and (f) in (e), the questions have been more completely answered; tradeoffs may be obtained using ideas such as those used in the proof of Theorem 7. We obtain:

Theorem 11: Let $\mathcal{S}$ = (b), $\mathcal{S}'$ = (c). Assume $\mathcal{S} \underset{\tau}{\leq}^{\text{closed}} \mathcal{S}'$ with time-size complexity f, g.
Then for some nonzero constant c, and all n, we have f(n) $\geq$ c $\sqrt{n}$ or g(n) $\geq$ cn.

Theorem 12: Let $\mathcal{S}$ = (f), $\mathcal{S}'$ = (e). Assume $\mathcal{S} \underset{\tau}{\leq}^{\text{closed}} \mathcal{S}'$ with time-size complexity f, g.
Then for some nonzero constant c, and all n, we have f(n) $\geq$ cn or g(n) $\geq 2^{cn}$.

These tradeoff lower bounds are both close to upper bounds (at least for the extreme values of f and g), so that these cases could be regarded as settled.

Finally, we consider complexity of tree programs. One of the results in [LB1] is a lower bound on the flowchart complexity of $\leq$ over $\langle N;0,1,+;=\rangle$. That proof is apparently dependent on the (identity) coding, but further consideration shows that the reason for this lower bound is a basic coding-independent incompatibility between the "shapes" of the systems $\langle N;0,1,+;\leq\rangle$ and $\langle N;0,1,+;=\rangle$. This incompatibility can be expressed in terms of the size parameter, using a natural i.o. - style lower bound definition:

Theorem 13: Let $\mathcal{S} = \langle N;0,1,+;\leq\rangle$, $\mathcal{S}' = \langle N;0,1,+;=\rangle$. Then $n[c.2^{n/4}]$ is a lower bound for $\mathcal{S} \leq^{\text{tree}} \mathcal{S}'$.

Proof: An abstraction of the one in [LB1], redone in terms of the size parameter. $\boxtimes$

Some of the other results in [LB1] can also be stated in a coding-independent way, using tree complexity. Noting that tree complexity provides a lower bound on flowchart complexity and complexity in all other reasonable languages, we may regard these results as fairly general.

V. COMPLEXITY OF GROUPS

In this section, flowchart time complexity is used, measured in terms of the size parameter. Work in [Ra] [M] [C] [CG] on computable and complexity-bounded algebraic systems is reconsidered, to determine whether our flowchart and size definitions might not yield sharper and more natural complexity results. We restrict attention to computable groups in this paper.

In [Ra] and [M], the primary emphasis was on computability of the systems; where the latter paper does deal with complexity issues, it is only at a very high level (primitive recursive). In [C] and [CG], the computability definitions of [Ra] are restricted in the most straightforward way to yield definitions for groups of different complexity. However, application of very basic group-theoretic constructions seems to cause complexity in their definition to rise one level in the Grzegorczyk hierarchy, suggesting that their definition is not useful for low-order complexity.

We restate definitions from the earlier papers in our notation, for comparison with our definitions. We call a group $G = \langle D_G;\ \circ,\ ^{-1}\rangle$ Rabin computable if there is an injection $i : D_G \to \{0,1\}^*$ such that $i\ (D_G)$ is a recursive set, and such that $\lambda(i(x), i(y))\ [i(x\circ y)]$ (and therefore $\lambda(i(x))\ [i(x^{-1})]$ is recursive. Thus, Rabin's definition differs from our style by specifying a recursiveness condition on the representing set, and by not permitting multiple representations of elements.

Mal'cev's work contains many different definitions, of a style quite similar to ours. As typical examples, we specialize two of his definitions to groups and restate them in our notation. A group $G = \langle D_G;\ \circ,\ ^{-1};\ =\rangle$ is Mal'cev computable if $G \underset{\tau}{\leq} B$ for some τ, where B is the basic Turing-Machine-like system of [LB1]; G is Mal'cev primitive recursive if $G \underset{\tau}{\leq} B$ with the simulators all primitive recursive. Note that the simulators are specified to be primitive recursive in the usual sense. Since primitive recursive functions are exactly those computable in primitive recursive time on a Turing maching, or by [LB1], by flowcharts over B with primitive recursive runtime, it follows that the primitive recursiveness restriction amounts to a complexity restriction on G with parameter residing in B rather than in G.

Cannonito and Gatterdam study complexity definitions based on a straightforward restriction of Rabin's definition. In our notation, a group $G = \langle D_G;\circ,^{-1}\rangle$ may be said to be of CG complexity t if $t:N\to N$ is total and G is Rabin computable with $i(D_G)$'s characteristic function, $\lambda(i(x), i(y))\ [i(x\circ y)]$ and $\lambda(i(x))\ [i(x^{-1})]$ all of (Turing) complexity t. Thus, only single representations are used, and in addition, parameters are based in the representing rather than the represented system.

By contrast, the style of definition which we consider to be most natural specializes as follows. $G = \langle D_G;\ \circ,\ ^{-1},\ =\rangle$ is computable if $G \underset{\tau}{\leq} B$ for some τ. (This is the same as Mal'cev's computability definition.) A finitely generated (f.g.) group $G = \langle D_G; g_1,\ldots,g_k,\circ,^{-1};=\rangle$, where $g_1,\ldots,g_k$ is a set of generators, is of complexity t if $G \underset{\tau}{\leq} B$ with complexity t from some τ.

From now on, all groups considered in this paper will be f.g.

At the level of computability, all of the definitions are equivalent.

Theorem 14: Let $G = \langle D_G;\ \circ,\ ^{-1}\rangle$ be a f.g. group, $G_1 = \langle D_G;\circ,^{-1};=\rangle$. Then G_1 is computable iff G is Rabin computable iff G_1 is Mal'cev computable.

Proof: Rabin computability easily implies computability.

A direct proof that computability implies Rabin computability takes a little work, but an indirect proof may be obtained by noting a simple equivalence between computability of a group and solvability of its word problem, coupled with a similar equivalence in [Ra] for Rabin computability. Namely, first consider a direct proof. Assume $G_1 \underset{\tau}{\leq} B$ and let $\{g_1,\ldots,g_k\}$ be any finite set of generators for G. Let $H = \langle D_G;\ g_1,\ldots,g_k,\circ,^{-1};=\rangle$. Then $H \underset{\tau}{\leq} B$. Define $\sigma : D_{Free(H)} \to \{0,1\}^*$ by $\sigma(x) =$ val$\rho(x)$, where ρ is a witnessing map for the reducibility $H \underset{\tau}{\leq} B$. (Note that σ is really independent of ρ, and is determined by the simulators.)

Now elements of $D_{Free(H)}$ can be regarded as expressions over $\{g_1,\ldots,g_k,\circ,-1\}$, and so can be coded into $\{0,1\}^*$ in a straightforward way. Let $\lambda : D_{Free(H)} \to \{0,1\}^*$ be this coding. λ may be defined in such a way as to be total, injective, and with its range a recursive set. Furthermore, a partial recursive function $\alpha : \{0,1\}^* \to \{0,1\}^*$ can be defined, with $\sigma(x) = \alpha(\lambda(x))$ for all $x\epsilon D_{Free(H)}$. (To compute $\sigma(y)$, we first check that $y\epsilon\lambda(D_{Free(H)})$, and then recursively evaluate the expression represented by y, following the given simulators of $g_1,\ldots,g_k,\circ,-1$.)

Fix some effective enumeration of elements of $\lambda(D_{Free(H)})$. Define $i(x)$, for $x\epsilon D_G$, as λ(the $y\epsilon D_{Free(H)}$ for which $\lambda(y)$ is first in the enumeration and for which val(y)=x). Then i may be shown to have the necessary properties for Rabin computability. For instance, we show $i(D_G)$ is recursive. Given $x\epsilon\ \{0,1\}^*$, check that

$x \varepsilon \lambda(D_{Free(H)})$. Then enumerate all elements of $\lambda(D_{Free(H)})$ up to x. For each such coded expression v in turn, apply the given simulator of = to $\alpha(v)$ and $\alpha(x)$ to determine whether $\tau(\alpha(v)) = \tau(\alpha(x))$. This is equivalent to determining if $val(\lambda^{-1}(v)) = val(\lambda^{-1}(x))$. If we ever obtain a "yes" answer, we know $x \not\in i(D_G)$, but if not, $x \varepsilon i(D_G)$. Similar arguments show the other needed properties.

As for the indirect proof, if $G_1 \underset{\tau}{\leq} B$, $\{g_1,\ldots,g_k\}$ any finite set of generators, then H as above satisfies $H \underset{\tau}{\leq} B$. Given two words over $\{g_1,\ldots,g_k,\circ,-1\}$, their equivalence is determined simply by simulating their evaluation and then applying the simulator of =. Conversely, assume the word problem with respect to $\{g_1,\ldots,g_k\}$ is solvable. Let H be as above, define λ as before, and τ by $\tau(\lambda(x)) = val(x)$. Simulators of $\circ$ and -1 are trivial to construct, and a simulator of = uses the solvability of the word problem. The equivalence of Rabin computability with solvability of the word problem, proved in [Ra], completes the proof. $\boxed{x}$

Next, consider complexity definitions. There does not seem to be any nice relationship between our definition and the CG definition, but our definition may be seen to be equivalent to the Mal'cev definition at the primitive recursive level. More specifically, we may first show that the complexity of a group in our definition is an algebraic invariant:

Theorem 15: Let $G = \langle D_G;\circ,^{-1}\rangle$ be a f.g. group, $\{g_1,\ldots,g_k\}$ and $\{h_1,\ldots,h_\ell\}$ two sets of generators. If $\langle D_G;g_1,\ldots,g_k,\circ,^{-1};=\rangle$ is of complexity t, then $\langle D_G:h_1,\ldots,h_\ell,\circ,^{-1};=\rangle$ is of complexity t', where $t'(n) = t(n+c)$, c some constant.

Proof: Composition lemmas in the style of Theorem 2 are used. $\boxed{x}$

Then we may show

Theorem 16: A f.g. group G is of primitive recursive complexity (with respect to any finite set of generators) iff G is Mal'cev primitive recursive.

Proof: If the group is Mal'cev primitive recursive, then the closure of the primitive recursive functions under unlimited iteration may be used to show that it is of primitive recursive complexity. Conversely, coding sequences of group operations by long strings keeps the simulators primitive recursive in the usual sense.

In somewhat more detail, assume G is Mal-cev primitive recursive. Let $\{g_1,\ldots,g_k\}$ be any finite set of generators, H as above, so $H \underset{\tau}{\leq} B$ with the simulators of $\circ$, -1 and = all primitive recursive, hence computable in primitive recursive time. Then by the closure of the primitive recursive functions under iteration, we can show that there exists primitive recursive p with $p(size_{Free(H)}(A)) \geq size_{Free(B)}(\rho(A))$ for all $A \subseteq D_{Free(H)}$, which in turn is $\geq size_B(val(\rho(A)))$. The implication then follows easily. Conversely, assume G is of primitive recursive complexity. Then if $H = \langle D_G;g_1,\ldots,g_k,\circ,^{-1};=\rangle$ for any finite set of generators, it follows that $H \underset{\tau}{\leq} B$ with primitive recursive complexity. The only possible difficulty would arise if $val \circ \rho(x)$ were considerably smaller than $size_{Free(H)}(x)$ for some values of x. We therefore modify τ to insure sufficiently long coding strings. Define σ, λ as in Theorem 14, and define $\tau^1 : \{0,1\}^* \to D_G$ by $\tau^1(\lambda(x)) = val(x)$. Now $\circ$ and -1 clearly have primitive recursive simulators, while simulation of = on $\lambda(x)$ and $\lambda(y)$ involves computing $\sigma(x)$ and $\sigma(y)$ and using the given τ- simulator of =. The time required is primitive recursive in $size_{Free(H)}(\{x,y\})$, and therefore primitive recursive in the lengths of $\lambda(x)$ and $\lambda(y)$. $\boxed{x}$

The need for iteration closure in Theorem 16 suggests that it is unlikely that the Mal'cev style definition could extend to lower levels of complexity with any sharp results. He did not attempt such a definition. The CG definition [C] [CG] leads to many iteration difficulties. By contrast, we obtain several simple, fairly sharp complexity results. For example, our definition of group complexity is very closely related to the complexity of the word problem, as one would expect from the simple equivalence proof. [C] [CG] obtain a relationship between the two questions' complexity, but because of the flavor of their definitions, a f.g. group of CG-complexity in $\mathcal{E}_\alpha$ can only be shown to have a word problem of complexity in $\mathcal{E}_{\alpha+1}$. This gap arises because a word of length n might be represented in $\{0,1\}^*$ by a string of length $f^n(0)$, $f \varepsilon \mathcal{E}_\alpha$. Closure under iteration requiers the jump of one level. Our definition, becuase of its choice of parameter, avoids the iteration difficulty:

Theorem 17: Let G be a group, $\{g_1,\ldots,g_k\}$ a finite set of generators, $H = \langle D_G;g_1,\ldots,g_k,\circ,^{-1};=\rangle$.

(a) if H is of complexity t, then the word problem for G with respect to $\{g_1,\ldots,g_k\}$ is solvable with (Turing) complexity $\lambda n[cn^2\ t^2\ (cn)]$ for some constant c.

(b) If $t:N \to N$ is total, nondecreasing, and $t(n) \geq n$ for all n, and the word problem for G with respect to $\{g_1,\ldots,g_k\}$ has complexity t (of the length of the word), then H is of complexity $\lambda n\ [t(c.2^n)]$ for some c mstant c.

While an exponential difference between the complexities is certainly not negligible, it appears to be unavoidable because of the incompatibility of the parameters involved. Namely, an expression $x \varepsilon D_{Free(H)}$, with $size_{Free(H)} = n$, can have its free-group-reduced representation of length about 2^n. But on the other hand, the best we can say about a free-group-reduced string of length n is that it has a generating expression in $D_{Free(H)}$ of $size_{Free(H)}$ roughly n (not necessarily log n).

In the same vein, we obtain a simple classification result:

Theorem 18: The free group over a finite set of generators is of complexity $\lambda n[c.2^n]$ for some constant c.

Proof: Code elements of $D_{Free(G)}$ as string representations of their free-group-reduced forms. The length of the representations stay bounded as above, and that length dominates the complexity. $\boxed{x}$

Question: Can this bound be improved?

Further evidence for the naturalness of our definitions is provided by the fact that simple algebraic constructions preserve complexity. We prove results which show that complexity is preserved under taking subgroups (up to a linear factor) under taking arbitrary quotients (with no increase at all), under direct product (up to a linear factor) and under "amalgamated free product" [MKS] (up to a triple exponential). The first and last of these four constructions, by contrast, cause rises of a level in the Grzegorczyk hierarchy when the CG definition is used to measure complexity.

More specifically, we obtain:

Theorem 19: Let $G = \langle D_G; g_1,\ldots,g_k, o, ^{-1}; = \rangle$ be a f.g. group, $\{g_1,\ldots,g_k\}$ a set of generators. Assume G is of complexty t. Let $H = \langle D_H; h_1,\ldots, h_\ell, o, ^{-1}; = \rangle$ be a f.g. subgroup of G with generating set $\{h_1,\ldots,h_\ell\}$. Then H is of complexity $\lambda n[t(n+c)]$ for some constant c.

Theorem 20: Let G be a f.g. group, H a normal subgroup of G, $\{g_1,\ldots,g_k\}$ a set of generators of G, and $\langle D_G; g_1,\ldots,g_k, o, ^{-1}; =, \equiv_H \rangle \underset{\tau}{\leq} B$ with complexity t. Then the quotient group $\langle D_{G/H}; [g_1],\ldots, [g_k], o, ^{-1}; = \rangle$ is of complexity t.

Theorem 21:: Assume $G = \langle D_G; g_1,\ldots,g_k, o, ^{-1}; = \rangle$ and $H = \langle D_H; h_1,\ldots,h_\ell, o, ^{-1}; = \rangle$ (with $\{g_1,\ldots,g_k\}$ and $\{h_1,\ldots,h_\ell\}$ respective sets of generators) are each of complexity t. Then the direct product of G and H, $GxH = \langle D_{GxH}; (g_i, e_H)$, $1 \leq i \leq k$, (e_G, h_i), $1 \leq i \leq \ell$, o, $^{-1}; = \rangle$, is of complexity $\lambda n[cnt(n)]$ for some constant c.

Proofs of the preceding three results are fairly straightforward. As an example of a less trivial construction, the final theorem bounds the complexity of the amalgamated free product of two f.g. groups [MKS] in terms of the complexity of the component groups and certain basic mappings and relations involving the component groups. Some notation is required. Let G_1 and G_2 be two disjoint groups, H a subgroup of G_1 and α an isomorphism of H into G_2. The amalgamated free product, $G_1 * G_{2(\alpha)}$, is then the quotient group of the free product of G_1 and G_2 which results from identification of H and its image. Using this notation, we state:

Theorem 22: Let $K = \langle D_{G_1} \cup D_{G_2}; h_1,\ldots,h_k, g_1,\ldots, g_\ell, g_1^1,\ldots,g_m^1, o_{G_1}, o_{G_2}, {}^{-1}{}_{G_1}, {}^{-1}{}_{G_2}, \alpha, \alpha^{-1}; =, \varepsilon D_{G_1}, \varepsilon D_{G_2}, \varepsilon D_H, \varepsilon\alpha(D_H), right_1, right_2, \rangle$ where $\{h_1,\ldots,h_k\}$ is a set of generators for H, $\{h_1,\ldots,h_k, g_1,\ldots,g_\ell\}$ a set of generators for G_1, $\{\alpha(h_1),\ldots,\alpha(h_k), g_1^1,\ldots,g_m^1\}$ a set of generators for G_2, where

$$right_1(x,y) = \begin{cases} \text{true if } x,y \varepsilon D_{G_1} \text{ and belong to the same right coset of } D_H, \\ \text{false if } x,y \varepsilon D_{G_1} \text{ and belong to different right cosets of } D_H, \\ \text{undefined otherwise.} \end{cases}$$

and $right_2$ is similar but with G_2 and $\alpha(D_H)$ replacing G_1 and D_H respectively. Assume $K \underset{\tau}{\leq} B$ with complexity t.

Then if $L = \langle D_{G_1 * G_{2(\alpha)}}; [h_1],\ldots,[g_m^1], o, ^{-1}; = \rangle$, it follows that $L \underset{\tau'}{\leq} B$ with complexity $\lambda n[c^{c^{c^n}} t(c^{c^n})]$ for some τ^1,c.

Proof: By means of careful bounds on a representation derived in Theorem 4.4 of [MKS]. $\boxed{x}$

We see that the computational complexity of a group measured according to our definition seems to relate to the complexity of the group measured in terms of algebraic decomposition. Perhaps then, our definition might have some use as a classification tool for group theory. We have not gone very deeply into the algebra, but it seems that complexity classification of groups and other mathematically interesting structures might be an enterprise of some value.

[ADJ] Goguen, J., Thatcher, J., and Wagner, E. An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types., IBM Technical Report.

[AHU] Aho, A., Hopcroft, J. and Ullman, J. The Design and Analysis of Computer Algorithms, Addison-Wesley, 1976.

[C] Cannonito, F. B. The Algebraic Invariance of the Word Problem in Groups, In Word Problems, Boone, W. W. Cannonito, F. B., Lyndon, R. C. Studies in Logic and the Foundations of Mathematics, Vol. 71. North-Holland, 1973.

[CG] Cannonito, F. B. and Gatterdam, R. W. The Computability of Group Constructions, Part I. Same volume as [C].

[GHM] Guttag, J., Horowitz, E. and Musser, D. Abstract Data Types and Software Validation. Research Report 76-48, Information Sciences Institute, Aug. 1976.

[LED1] Lipton, R., Eisenstat, S. and DeMillo, R. Space and Time Hierarchies for Classes of Control Structures and Data Structures. JACM Vol. 23 No. 4, October, 1976.

[LED2] Lipton, R., Eisenstat, S. and DeMillo, R. Space - Time Tradeoffs in Structured Programming: An Improved Combinatorial Embedding Theorem. To appear.

[LZ] Liskov, B. and Zilles, S. Specification Techniques for Data Abstractions, Software Engineering Vol. SE-1, No.1, March 1975, pp. 7-19.

[LB1] Lynch, N. and Blum, E.K. Efficient Reducibility Between Programming Systems: Preliminary Report. 9th Annual ACM STOC, 1977, pp. 228-238.

[LB2] Lynch, N. and Blum, E. K. Relative Complexity of Programming Systems, manuscript in preparation.

[M] Mal'cev, A. The Metamathematics of Algebraic Systems. Studies in Logic and the Foundations of Mathematics, Vol. 66. North-Holland, 1971.

[MKS] Magnus, W., Karrass, A. and Solitar, D. Combinational Group Theory, Presentations of groups in terms of generators and relations, New York Interscience Publishers, Vol.13, 1966

[Ra] Rabin, M. O. Computable Algebra, General Theory and Theory of Computable Fields, Trans. AMS 95 (1960), pp. 341-360.

[Ro] Rosenberg, A. Preserving Proximity in Arrays. IBM T. J. Watson Research Center Report RC-4875, 1974.

[Si] Simon, J. On Feasible Numbers. 9th Annual ACM STOC, 1977.

[T] Tarjan, R. Reference Machines Require Non-linear Time to Maintain Disjoint Sets. 9th Annual ACM STOC, 1977.

[Y] Yasuhara, A. Recursive Function Theory and Logic. Academic Press, 1971.