JOURNAL OF COMPUTER AND SYSTEM SCIENCES 21, 251–280 (1980)

Straight-Line Program Length as a Parameter for Complexity Analysis*

NANCY A. LYNCH

Georgia Institute of Technology, Atlanta, Georgia 30332

Received September 11, 1978; revised June 4, 1980

A definition is proposed for a size measure to be used as a parameter for algorithm analysis in any algebra. The parameter is simply the straight-line program length in the associated free algebra. This parameter generalizes the usual measures in basic arithmetic and string algebras, as well as some apparently different measures used for data structure algorithms. Another use is illustrated with an introduction to complexity-bounded group theory.

I. INTRODUCTION

This paper continues the work in [9–12] directed toward the development of a unified, relative framework for complexity theory. Those papers establish a natural model for situations in which each data element is regarded as "atomic"; for example, it can be copied in one computation step. The attractiveness of the model is demonstrated by its use in stating and proving a variety of technical results, principally involving data types whose elements are bit strings or natural numbers. It would be useful to extend those ideas to data types whose elements are not usually regarded as atomic (such as matrices, graphs, or storage-retrieval structures of various kinds). This paper treats a problem that permeates the other work: How should complexity bounds be *stated* for arbitrary operations on arbitrary data types? More specifically, if algorithms for operations in data types such as groups or matrices are to be classified as $O(n)$, $O(n^2)$, or $O(2^n)$, then what are appropriate choices for the parameter n? Some choices in the literature seem to be ad hoc; this paper is an attempt to unify them.

When one computes with bit strings or natural numbers, using some simple set of basic operations, 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 are appropriate parameters for

* This work was partially supported by the National Science Foundation through Grants DCR 75-02373 and MCS 77-15628 and U. S. Army Research Office Contract DAAG29-79-C-0155.

0022-0000/80/060251-30$02.00/0

complexity measurement. (Even in this simple situation, there might be algorithms for which other parameters are more appropriate; the complexity of some numerical algorithms might depend more naturally on the number of prime factors than on the logarithm of a number.)

However, if one attempts to understand programs by imposing a multi-level, hierarchical structure on them where possible, then one does not always want to think of oneself as computing with low-level objects such as 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 as method of understanding programs is suggested by the extensive recent research in formal semantics, programming logics, and formal specification techniques for data structures. In this case, it is not quite so obvious how to define and express complexity bounds.

One first requires a suitable general model, capable of measuring realistically the complexity of computations performed using either low-level or high-level objects and operations. Second, one needs a general way of expressing these measurements in a usable form. Uniformity of the model and of the complexity statements are important, since complexity analyses of the separate levels of an algorithm should be combinable in a straightforward way into a complexity analysis of the entire algorithm.

We are thus led to a general model for complexity measurement which is inherently relative [9–12]. Some of the features of our model are the following. Two kinds of modularity are expressible—the definition of a new operation on a previously defined data type, and the representation of an entirely new "higher-level" data type (with its associated operations) relative to a previously defined "lower-level" data type. The new, higher-level data type is designated as *represented* and the given lower-level data type as *representing*. (Thus, if one is given bit strings with a set of operations and wishes to "implement" a particular group, then he seeks a representation of the group's elements and operations relative to the bit string data type. The group is the represented data type, while the bit string data type is the representing data type. Similarly, if one is given the natural numbers with zero, successor and certain other operations and wishes to compute using integers with appropriate operations, then he seeks a representation of the integers as natural numbers and suitable representation of the integer operations in terms of the natural number operations. The integers comprise the represented data type, while the natural number data type is the representing data type.) The model is algebraic. In particular, data types are assumed to be defined up to isomorphism. (Unlike [5, 6, 13], however, we are not concerned with particular techniques for specifying this definition.) Encodings are not constrained a priori. Our point of view is that there is an inherent coding-independent relative complexity for data types, which can be thought of as a trade-off between the complexity of their various operations. The framework used is closer to models of programming used in other branches of computation theory than is the RAM- or Turing machine-style framework generally used in complexity theory.

As mentioned above, complexity analyses of the separate levels of an algorithm should be combinable in a reasonably simple way to yield a complexity analysis for

the entire algorithm. Furthermore, an analysis of the algorithm in terms of high-level operations on high-level objects should not require knowledge of the lower-level representation of those objects nor of the lower-level implementation of those operations. In the extreme, an analysis which includes, for each input to the 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 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 goal arises. For arbitrary data types, a parameter as natural as length for bit strings and logarithm for natural numbers is required, upon which to base complexity analysis. This parameter should be chosen in a uniform way for all data types, in order to facilitate combination of analyses. It should generalize the length and logarithm measures, so that results about bit strings and natural numbers will be expressible. It should reside in the represented rather than in the representing data type. This paper contends that a simple size measure, the length of a straight-line program to generate an element, is an appropriate parameter.

Insistence that the size parameter reside in the represented rather than the representing data type distinguishes this work from previous work on relative complexity of algebras [2–4, 14]. Definitions of the style used in those papers (i.e., parameters in the representing system) are somewhat easier to state than ours, but evidence that they are less natural is provided by the difficulties encountered in those papers. Intuitively, one wishes to measure the complexity of the accomplishment of a certain task—the implementation of a new data type. If measures are based on size of representing elements, then one observes the odd phenomenon that the task is deemed "more efficiently accomplished" when the representing elements are 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 function might be used. The only way to make valid comparisons of complexity of various ways of accomplishing the task is to base the compared measurements on a common parameter, one derived from the task itself.

Our use of the straight-line program length parameter discussed in this paper originates in [9, 11]. There, the optimality of a standard coding of N into $\{0, 1\}^*$ is proved in several different formulations, one of which is generalizable to arbitrary algebras. Another complexity study using a version of this size parameter appears in [17]. Also, a disguised use of this parameter appears (for example) in the UNION-FIND and INSERT-MEMBER-DELETE algorithms and lower bounds in [1, 18]. If "dictionaries" [1] and other such data structures are described in the many-sorted algebraic framework of [5, 6, 13], then the "number of operations simulated" parameter used in those results can be expressed formally as the size of an element in an appropriate many-sorted algebra. All of this work way be regarded as evidence for the naturalness of the measure, and this paper is intended to provide further evidence.

Following this section, the organization of the rest of the paper is as follows. Section II contains notation, definitions with their technical motivation, and

elementary results. Section III contains theorems about combination of analyses. Section IV contains, as an extended example, an introduction to a theory of complexity for finitely-generated groups. This theory is a refinement of the computable group theory of [16]. Basic results are shown; they appear to be neater and sharper than those obtained in previous attempts at developing such a theory. Much work, however, remains to be done.

Another application of the present ideas appear in a companion paper [7]; there, several numeric and bit-string algebras are classified by relative "accessibility" complexity. Those results are useful primarily as coding-independent lower bounds on computation time in ordinary programming languages. The apparent tradeoff between accessibility complexity and number of representations is also examined in that paper. Arguments about the sizes of neighborhood are used. The group theory results and the results of [7] involve measurement of several different types of complexity; however, all are expressed in terms of the same size parameter.

Reference [8] includes earlier versions of the present results, as well as the results of [7].

II. Notation, Definitions, and Basic Results

Size Parameter

Let N denote the set of natural numbers, including 0, $\{0,1\}^*$ the set of binary strings of finite length, Z the integers, and R the real numbers.

For $x, y \in N$, let $\langle x, y\rangle$ denote $\frac{1}{2}((x+y)^2+3x+y)$. This function maps N^2 bijectively onto N. Let $\pi_1, \pi_2: N \to N$ be total functions such that $\langle \pi_1(x), \pi_2(x)\rangle = x$. For $x, y \in \{0,1\}^*$, let $\langle x, y\rangle$ denote the string $x_1 0 x_2 0 \cdots 0 x_k 1 1 y_1 0 y_2 0 \cdots 0 y_l$, where $x = x_1 \cdots x_k$ and $y = y_1 \cdots y_l$. This function maps $(\{0,1\}^*)^2$ injectively into $\{0,1\}^*$. Let $\pi_1, \pi_2: \{0,1\}^* \to \{0,1\}^*$ be partial functions such that $\langle \pi_1(x), \pi_2(x)\rangle = x$. Both pairing functions are extended to more than two arguments by repeated application.

An *algebra* $\mathscr{A} = \langle \mathrm{Dom}_{\mathscr{A}}; \mathrm{Fun}_{\mathscr{A}}; \mathrm{Rel}_{\mathscr{A}}\rangle$ is a set $\mathrm{Dom}_{\mathscr{A}}$ (the *domain* of $\mathscr{A}$) together with a set $\mathrm{Fun}_{\mathscr{A}}$ of partial functions (i.e., operations) and a set $\mathrm{Rel}_{\mathscr{A}}$ of partial relations on $\mathrm{Dom}_{\mathscr{A}}$. Constants are 0-ary functions. The members of $\mathrm{Fun}_{\mathscr{A}}$ and $\mathrm{Rel}_{\mathscr{A}}$ are called *basic* functions and relations of $\mathscr{A}$. We assume that every element of $\mathrm{Dom}_{\mathscr{A}}$ can be generated by a finite number of applications of functions in $\mathrm{Fun}_{\mathscr{A}}$ (to constants in $\mathrm{Fun}_{\mathscr{A}}$).

Although $\mathrm{Fun}_{\mathscr{A}}$ and $\mathrm{Rel}_{\mathscr{A}}$ have been defined to be sets of functions and relations, the same notation is sometimes used to represent corresponding sets of symbols for those functions and relations. We rely on the reader to make such distinctions when necessary.

Let $\{v_i\}_{i=1}^{\infty}$ be a set of formal variables, Fun a set of function symbols. Then $V\,\mathrm{Exp}_n(\mathrm{Fun})$, $n \in N$, denotes the set of all well-formed expressions over the symbols in Fun and (any subset of) $\{v_i\}_{i=1}^{n}$. $V\,\mathrm{Exp}(\mathrm{Fun})$ denotes $\bigcup_{n\in N} V\,\mathrm{Exp}_n(\mathrm{Fun})$.

If $\mathscr{A}$ is an algebra, $e \in V\,\mathrm{Exp}_0(\mathrm{Fun}_{\mathscr{A}})$, then $\mathrm{val}(e)$ denotes the value of e when evaluated in $\mathscr{A}$. ($\mathrm{val}(e)$ may be undefined.) Let $\mathscr{F}ree(\mathscr{A})$ be the algebra having as its domain the set of e in $V\,\mathrm{Exp}_0(\mathrm{Fun}_{\mathscr{A}})$ for which $\mathrm{val}(e)$ is defined, as its functions

the set of free applications of function symbols in $\mathrm{Fun}_{\mathscr{A}}$ to elements of $V\mathrm{Exp}_0(\mathrm{Fun}_{\mathscr{A}})$, and its relations defined as follows. Let each basic relation r on $\mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{A})}$ be defined by $r(e_1,\ldots,e_n) = r(\mathrm{val}(e_1),\ldots,\mathrm{val}(e_n))$. Note that $r(e_1,\ldots,e_n)$ is defined if and only if $r(\mathrm{val}(e_1),\ldots,\mathrm{val}(e_n))$ is defined, and similarly for basic functions.

Whenever any function or relation is undefined, its "value" is written as ∞. Thus, for $x_1,\ldots,x_n \in \mathrm{Dom}_{\mathscr{A}}$, $f(x_1,\ldots,x_n) = \infty$ indicates that f is undefined for the arguments $x_1,\ldots,x_n$. For $e \in V\mathrm{Exp}_0(\mathrm{Fun}_{\mathscr{A}})$, $\mathrm{val}(e) = \infty$ indicates that $\mathrm{val}(e)$ is undefined (i.e., $e \notin \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{A})}$). In formulas involving compositions, the "value" ∞ will always propagate outward. By convention, $n < \infty$ for all $n \in N$, and $\infty \leqslant \infty$.

The size parameter is now defined. If $A, B \subseteq \mathrm{Dom}_{\mathscr{A}}$, then $\mathrm{size}_{\mathscr{A}}(A:B)$, (the size, in $\mathscr{A}$, of A relative to B) is given as follows:

(a) $\mathrm{size}_{\mathscr{A}}(A:B) = 0$ iff $A \subseteq B$.

(b) $\mathrm{size}_{\mathscr{A}}(A:B) = k+1$ iff both of the following hold:

(b1) $\mathrm{size}_{\mathscr{A}}(A:B) \not\leqslant k$.

(b2) For some $C \subseteq \mathrm{Dom}_{\mathscr{A}}$, $f \in \mathrm{Fun}_{\mathscr{A}}$, $x_1,\ldots,x_n \in C$, it is the case that $A \subseteq C \cup \{f(x_1,\ldots,x_n)\}$ and $\mathrm{size}_{\mathscr{A}}(C:B) = k$.

(c) $\mathrm{size}_{\mathscr{A}}(A:B)$ is otherwise undefined (and is said to be equal to ∞).

Thus, $\mathrm{size}_{\mathscr{A}}(A:B)$ describes the number of steps required by a straight-line program, given the values in B, to generate the values in A. Let $\mathrm{size}_{\mathscr{A}}(x:B)$ denote $\mathrm{size}_{\mathscr{A}}(\{x\}:B)$, $\mathrm{size}_{\mathscr{A}}(A)$ denote $\mathrm{size}_{\mathscr{A}}(A:\varnothing)$ and $\mathrm{size}_{\mathscr{A}}(x)$ denote $\mathrm{size}_{\mathscr{A}}(\{x\}:\varnothing)$. Clearly, only the functions in $\mathrm{Fun}_{\mathscr{A}}$ (and not the relations in $\mathrm{Rel}_{\mathscr{A}}$) are needed to determine the size measure. This definition is more general than that used, for example, in [17]; this level of generality was chosen for the naturalness of its use in proofs.

If Fun is a set of function symbols and $e \in V\mathrm{Exp}(\mathrm{Fun})$ then $\mathrm{size}(e) = \mathrm{size}_{\langle V\,\mathrm{Exp}(\mathrm{Fun});\ \mathrm{Fun}\cup\{v_i\}_{i=1}^{\infty};\rangle}(e:\{v_i\}_{i=1}^{\infty})$. That is, the size of e is the number of steps required by a straight-line program to generate e from its formal variables.

THEOREM 2.1 (Basic properties of the size measure).

(a) *If* $A, B, C \subseteq \mathrm{Dom}_{\mathscr{A}}$ *and* $A \subseteq C$, *then* $\mathrm{size}_{\mathscr{A}}(A:B) \leqslant \mathrm{size}_{\mathscr{A}}(C:B)$.

(b) *If* $A, B, C \subseteq \mathrm{Dom}_{\mathscr{A}}$ *and* $B \subseteq C$, *then* $\mathrm{size}_{\mathscr{A}}(A:B) \geqslant \mathrm{size}_{\mathscr{A}}(A:C)$.

(c) (*Triangle Inequality*)
If $A, B, C \subseteq \mathrm{Dom}_{\mathscr{A}}$, *then* $\mathrm{size}_{\mathscr{A}}(A:B) + \mathrm{size}_{\mathscr{A}}(B:C) \geqslant \mathrm{size}_{\mathscr{A}}(A:C)$.

(d) *If* $A, B \subseteq \mathrm{Dom}_{\mathscr{A}}$, *then* $\mathrm{size}_{\mathscr{A}}(A:B) = \mathrm{size}_{\mathscr{A}}(A \cap \bar{B}:B)$. ($\bar{B}$ *is the complement of* B, $\mathrm{Dom}_{\mathscr{A}} - B$.)

(e) *If* $A, B, C \subseteq \mathrm{Dom}_{\mathscr{A}}$, *then* $\mathrm{size}_{\mathscr{A}}(A \cup B:C) \leqslant \mathrm{size}_{\mathscr{A}}(A:C) + \mathrm{size}_{\mathscr{A}}(B:C)$.

(f) (*A connection between size in an algebra and in its free version*)
If $A, B \subseteq \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{A})}$, *then* $\mathrm{size}_{\mathscr{A}}(\mathrm{val}(A):\mathrm{val}(B)) \leqslant \mathrm{size}_{\mathscr{F}_{free}(\mathscr{A})}(A:B)$.

(g) (*Another such connection*)
If $A, B \subseteq \mathrm{Dom}_{\mathscr{A}}$, $D \subseteq \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{A})}$ *with* $\mathrm{val}(D) = B$, *then there exists* $C \subseteq \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{A})}$ *with* $\mathrm{val}(C) = A$ *and* $\mathrm{size}_{\mathscr{A}}(A:B) = \mathrm{size}_{\mathscr{F}_{free}(\mathscr{A})}(C:D)$.

(h) *Let* $A, B, C \subseteq \mathrm{Dom}_{\mathscr{A}}$. *Then* $\mathrm{size}_{\mathscr{A}}(A : B \cup C) \leqslant \mathrm{size}_{(\mathrm{Dom}_{\mathscr{A}}; \mathrm{Fun}_{\mathscr{A}} \cup C; \mathrm{Rel}_{\mathscr{A}})}(A : B) \leqslant \mathrm{size}_{\mathscr{A}}(A : B \cup C) + |C|$.

(i) *If* $x_1, \ldots, x_n \in \mathrm{Dom}_{\mathscr{F}ree(\mathscr{A})}$ *and* $f(x_1, \ldots, x_n)$ *is defined, then* $\mathrm{size}_{\mathscr{F}ree(\mathscr{A})}(f(x_1, \ldots, x_n)) = \mathrm{size}_{\mathscr{F}ree(\mathscr{A})}(\{x_1, \ldots, x_n\}) + 1$.

(j) *If* $\mathrm{Fun}_{\mathscr{A}} \subseteq \mathrm{Fun}_{\mathscr{A}'}$ *and if* $A \subseteq \mathrm{Dom}_{\mathscr{F}ree(\mathscr{A})}$, *then* $\mathrm{size}_{\mathscr{F}ree(\mathscr{A})}(A) = \mathrm{size}_{\mathscr{F}ree(\mathscr{A}')}(A)$.

Proof. Straightforward. Properties (i) and (j) indicate that there is no faster way to generate terms in a free algebra than by first generating their subterms. ∎

Simulators

As in [9, 11], let $\tau : A' \to A$ be a partial, surjective function, f and f' partial functions on A and A', respectively. Then f' is a *τ-simulator* of f 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 r and r' are partial relations on A and A', respectively, then r' is a *τ-simulator* of r if whenever $r(\tau(x_1), \ldots, \tau(x_n))$ is defined, $r'(x_1, \ldots, x_n)$ is as well, and their values are equal. Note that τ is not required to be injective; multiple representations are allowed for single objects.

EXAMPLE 2.1. Let $\tau_1 : N \to Z$ translate the natural numbers into integers by letting even numbers represent nonnegative integers and odd numbers represent negative integers: $\tau_1(2y) = y$ and $\tau_1(2y - 1) = -y$, $y \in N$. Let $\tau_2 : N \to Z$ translate the same domains by decoding its argument into two natural numbers and taking the difference: $\tau_2(\langle x, y \rangle) = x - y$. τ_2 is surjective but not injective.

Then the following functions f_1 and f_2 are a τ_1-simulator and a τ_2-simulator of integer addition, respectively.

$$
\begin{aligned}
f_1(x, y) &= x + y && \text{if } x \text{ and } y \text{ are both even,} \\
&= x + y + 1 && \text{if } x \text{ and } y \text{ are both odd,} \\
&= x - y - 1 && \text{if } x \text{ is even, } y \text{ is odd and } x > y, \\
&= y - x && \text{if } x \text{ is even, } y \text{ is odd and } x < y, \\
&= y - x - 1 && \text{if } y \text{ is even, } x \text{ is odd and } y > x, \\
&= x - y && \text{if } y \text{ is even, } x \text{ is odd and } y < x.
\end{aligned}
$$

$$f_2(x, y) = \langle \pi_1(x) + \pi_1(y), \pi_2(x) + \pi_2(y) \rangle.$$

Also, the following functions g_1 and g_2 are a τ_1-simulator and a τ_2-simulator of unary minus, respectively.

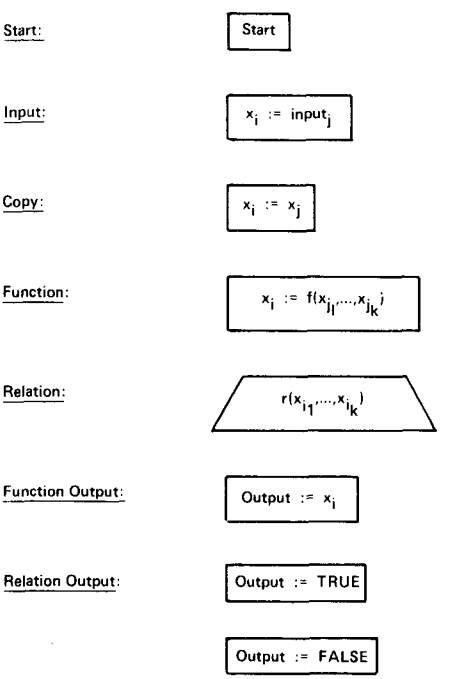
$$
\begin{aligned}
g_1(x) &= 0 && \text{if } x = 0, \\
&= x - 1 && \text{if } x > 0 \text{ is even,} \\
&= x + 1 && \text{if } x > 0 \text{ is odd.}
\end{aligned}
$$

$$g_2(x) = \langle \pi_2(x), \pi_1(x) \rangle.$$

FLOWCHARTS AND EXPRESSION ASSIGNMENTS

In order to understand and express the relative complexity of two arbitrary algebras, $\mathscr{A}$ and $\mathscr{A}'$, we define a *translation map between* $\mathscr{A}$ *and* $\mathscr{A}'$ to be a partial surjective function $\tau: \mathrm{Dom}_{\mathscr{A}'} \to \mathrm{Dom}_{\mathscr{A}}$. We wish to examine the "complexity" of τ-simulators of the basic functions and relations of $\mathscr{A}$. To establish a reasonable meaning for this complexity, we must relate these simulators to the basic functions and relations of $\mathscr{A}'$. This we do by specifying that the simulators be computable by programs in some programming language using the basic functions and relations of $\mathscr{A}'$. There are many possible choices of such languages; a simple flowchart programming language is used in [9–12] to prove complexity upper and lower bounds. In this paper, two languages are considered—flowcharts (again) and expression assignments. The latter generalizes most programming languages and is therefore useful for proving widely applicable lower bounds.

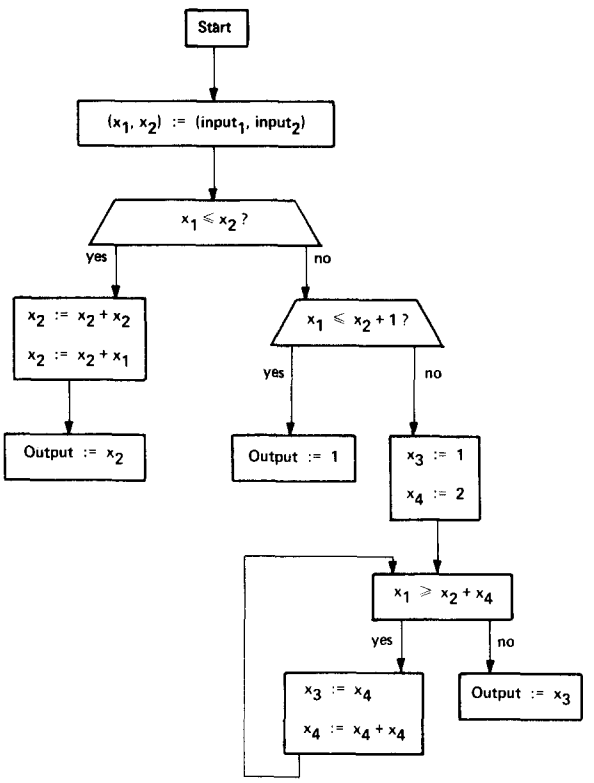
For an algebra $\mathscr{A}$, a flowchart program P over $\mathscr{A}$ is composed in the usual way from a finite number of boxes of the types



where $f \in \mathrm{Fun}_{\mathscr{A}}$ and $r \in \mathrm{Rel}_{\mathscr{A}}$ and the x's are variables. Output boxes have no successors, relation boxes have two successors, and all others have one successor. A flowchart is either a *function flowchart*, in which case all output boxes are function

output boxes, or a *relation flowchart*, in which case all output boxes are relation output boxes. There is exactly one start box. A flowchart P defines a partial function fn_P or a partial relation rel_P.

EXAMPLE 2.2. A straightforward flowchart follows for the function $f(x, y) =$ the largest power of two not greater than $x - y$ if $x > y$; $x + 2y$ otherwise, over the algebra $\mathcal{N} = \langle N; 0, 1, +; \leqslant \rangle$. Some abbreviations are used, but it is easy to convert the given flowchart to the format specified in the definition.



Clearly, for every flowchart over an algebra $\mathcal{A}$ (as well as any program over $\mathcal{A}$ in most other programming languages) there is an input–output–equivalent "tree program" over $\mathcal{A}$, constructed simply by unwinding loops. It may seem that tree programs are as basic a language as one might wish to consider, but we find it convenient to use even a further abstraction. The new rudimentary programming language we define is based on "expression assignment" programs.

If $\mathcal{A}$ is an algebra, then $e \in V\,\text{Exp}_n(\text{Fun}_{\mathcal{A}})$ defines a partial function

fn_e: $(\mathrm{Dom}_{\mathscr{A}})^n \to \mathrm{Dom}_{\mathscr{A}}$; $fn_e(x_1,..., x_n)$ is the value in $\mathrm{Dom}_{\mathscr{A}}$, if any, resulting when e is evaluated with $x_1,...,x_n$ replacing $v_1,..., v_n$, respectively. If $\mathscr{A}$ is an algebra, then an *n-ary expression assignment* over $\mathscr{A}$ is a partial mapping E: $(\mathrm{Dom}_{\mathscr{A}})^n \to V\,\mathrm{Exp}_n(\mathrm{Fun}_{\mathscr{A}})$. E can be regarded as a program as follows, if $(x_1,..., x_n)$ is in domain(E), then $E(x_1,..., x_n)$ is a formal expression over $\mathrm{Fun}_{\mathscr{A}} \cup \{v_i\}_{i=1}^n$ which describes a computation on n arbitrary inputs. This computation can be applied to $(x_1,..., x_n)$, where each x_i replaces the corresponding v_i. A value in $\mathrm{Dom}_{\mathscr{A}}$ may result. E defines a partial function fn_E: $(\mathrm{Dom}_{\mathscr{A}})^n \to \mathrm{Dom}_{\mathscr{A}}$, given by $\mathrm{fn}_E(x_1,..., x_n) = \mathrm{fn}_{E(x_1,...,x_n)}(x_1,..., x_n)$. All expression assignments compute functions rather than relations.

Every function flowchart (as well as practically every other function program) yields a natural input–output-equivalent expression assignment. Intuitively, any input vector $(x_1,..., x_n)$ determines a path through the flowchart. If that path terminates, one can extract from the path the expression which was applied to $(x_1,..., x_n)$ during the computation, to build the output value. This expression, with the formal variables $(v_1,..., v_n)$ as placeholders for the inputs, is the expression assigned to $(x_1,..., x_n)$. The expression assignment thereby defined is called the *derived expression assignment* for the original flowchart.

EXAMPLE 2.3. The derived expression assignment for the flowchart of Example 2.2 is described. If $x \leqslant y$, then $E(x, y) = ((v_2 + v_2) + v_1)$. (That is, the value assigned to inputs x, $y \in N$ is the formal expression given in the right-hand side of the equality.) If $x = y + 1$, then $E(x, y) =$ (the formal expression) 1. Finally, if $x > y + 1$, then $E(x, y)$ is a formal expression giving a fully expanded unary representation of the largest power of two not greater than $x - y$; the particular expression is symmetric, so that $E(5, 0) = (1 + 1) + (1 + 1)$ and $E(8, 0) = ((1 + 1) + (1 + 1)) + ((1 + 1) + (1 + 1))$. Note that an assigned expression might or might not contain formal variables v_i. Note also that tests performed during the computation are not explicitly represented in the expression assignment. Finally, note that expressions with repeated subexpressions have those subexpressions written out repeatedly; thus, the length of an assigned expression e might be much greater than size(e).

Relative Computability of Algebras

Before defining the relative complexity of two algebras, we define their relative computability.

A mapping from one set of programs or functions to another is *arity-preserving* provided the image of each program or function has the same number of arguments as the program or function.

Write $\mathscr{A} \leqslant^{\exp}_{\tau,\mathscr{F}} \mathscr{A}'$ if τ is a translation map between $\mathscr{A}$ and $\mathscr{A}'$, $\mathscr{F}$ is a total, arity-preserving mapping from $\mathrm{Fun}_{\mathscr{A}}$ to the set of expression assignments over $\mathscr{A}'$, and $\mathrm{fn}_{\mathscr{F}(f)}$ is a τ-simulator of f for each $f \in \mathrm{Fun}_{\mathscr{A}}$. (Relations of $\mathscr{A}$ are unconstrained in this weak definition.)

Let $\mathscr{P}$ be an arity-preserving total mapping from $\mathrm{Fun}_{\mathscr{A}} \cup \mathrm{Rel}_{\mathscr{A}}$ to the set of

flowchart programs over $\mathscr{A}'$. Then $\mathscr{A} \leqslant^{\text{flow}}_{\tau,\mathscr{E},\mathscr{P}} \mathscr{A}'$ provided $\mathscr{A} \leqslant^{\text{exp}}_{\tau,\mathscr{E}} \mathscr{A}'$ and (a) and (b) hold.

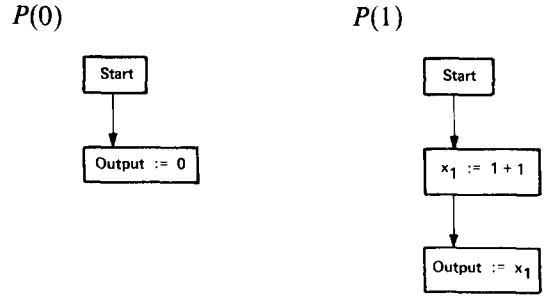
(a) Each $\mathscr{E}(f)$ is the derived expression assignment of flowchart $\mathscr{P}(f)$. (Thus, $\text{fn}_{\mathscr{P}(f)} = \text{fn}_{\mathscr{E}(f)}$.)

(b) If $r \in \text{Rel}_{\mathscr{A}}$, then $\mathscr{P}(r)$ computes a τ-simulator of r. When $\mathscr{P}$ is irrelevant, write $\mathscr{A} \leqslant^{\text{flow}}_{\tau,\mathscr{E}} \mathscr{A}'$ for $(\exists \mathscr{P})[\mathscr{A} \leqslant^{\text{flow}}_{\tau,\mathscr{E},\mathscr{P}} \mathscr{A}']$. When $\mathscr{E}$ is irrelevant, write $\mathscr{A} \leqslant^{\text{exp}}_{\tau} \mathscr{A}'$ (resp. $\mathscr{A} \leqslant^{\text{flow}}_{\tau} \mathscr{A}'$) for $(\exists \mathscr{E})[\mathscr{A} \leqslant^{\text{exp}}_{\tau,\mathscr{E}} \mathscr{A}']$(resp. $(\exists \mathscr{E})[\mathscr{A} \leqslant^{\text{flow}}_{\tau,\mathscr{E}} \mathscr{A}']$). The definition of $\mathscr{A} \leqslant^{\text{exp}}_{\tau} \mathscr{A}'$ requires simply that each basic function of $\mathscr{A}$ have a τ-simulator whose value is always in the algebraic closure of its arguments. The reader is referred to [9, 11] for further discussion of some of these definitions, as well as further motivational remarks.

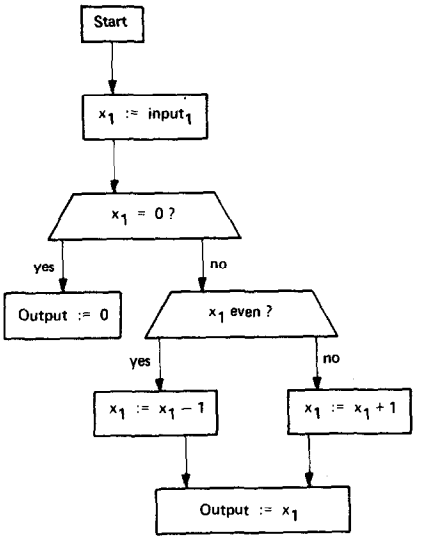
Conventions for the heavily embellished "$\leqslant$" are the following. Explicit mappings pertaining to reducibilities appear below the symbol, while modifiers appear as superscripts. Complexity bounds, to be introduced shortly, will be written after the reducibility statement.

EXAMPLE 2.4. Let $\mathscr{Z} = \langle Z; 0, 1, \text{unary}-, +; \rangle$ and $\mathscr{N} = \langle N; 0, 1, +, \dot{-}; =, <, \text{even} \rangle$. (Here, $x \dot{-} y = x - y$ if $x \geqslant y$; 0 otherwise.) Let τ be τ_1 of Example 2.1. Define $\mathscr{E}$ as follows. $\mathscr{E}(0)() = 0$, $\mathscr{E}(1)() = 1 + 1$, $\mathscr{E}(\text{unary}-)(\times) = 0$ if $x = 0$, $v_1 \dot{-} 1$ if $x > 0$ is even, $v_1 + 1$ if $x > 0$ is odd, $\mathscr{E}(+)(x, y) = v_1 + v_2$ if x and y are both even, $(v_1 + v_2) + 1$ if x and y are both odd, $(v_1 \dot{-} v_2) \dot{-} 1$ if x is even, y is odd and $x > y$, $v_2 \dot{-} v_1$ if x is even, y is odd and $x < y$, $(v_2 \dot{-} v_1) \dot{-} 1$ if y is even, x is odd and $y > x$, and $v_1 \dot{-} v_2$ if y is even, x is odd and $y < x$. Clearly, $\mathscr{Z} \leqslant^{\text{exp}}_{\tau,\mathscr{E}} \mathscr{N}$.

Now define $\mathscr{P}$ by four flowcharts



P(unary−)



$P(+)$

Left to the reader.

Then $\mathscr{Z} \leqslant^{\exp}_{\tau,\mathscr{F},\rho} \mathscr{N}$.

Translation, Representation, and Simulation

The translation map τ is from $\mathscr{A}'$, the representing algebra, to $\mathscr{A}$, the represented algebra. For computability considerations this appears to be sufficient. However, for complexity considerations it is helpful also to consider a "representation" map in the other direction. Since τ is permitted to be many-to-one (i.e., multiple representations are allowed for single objects), the new mapping is not defined from $\mathrm{Dom}_{\mathscr{A}}$ to $\mathrm{Dom}_{\mathscr{A}'}$, but rather from $\mathrm{Dom}_{\mathscr{F}ree(\mathscr{A})}$ to $\mathrm{Dom}_{\mathscr{A}'}$. Intuitively, for every generation of an element of $\mathrm{Dom}_{\mathscr{A}}$, a representing element of $\mathrm{Dom}_{\mathscr{A}'}$ is selected.

If $\mathscr{A}$ is an algebra, Fun a set of function symbols, and $\mathscr{F}$ an arity-preserving total mapping from Fun to the set of partial functions on $\mathrm{Dom}_{\mathscr{A}}$, then a partial mapping $\rho_{\mathscr{F}}: V\,\mathrm{Exp}_0(\mathrm{Fun}) \to \mathrm{Dom}_{\mathscr{A}}$ is defined inductively as follows. If c is a constant symbol, then $\rho_{\mathscr{F}}(c) = \mathscr{F}(c)(\,)$. If $e_1,\ldots,e_n \in V\,\mathrm{Exp}_0(\mathrm{Fun})$, then $\rho_{\mathscr{F}}(f(e_1,\ldots,e_n)) = \mathscr{F}(f)(\rho_{\mathscr{F}}(e_1),\ldots,\rho_{\mathscr{F}}(e_n))$.

EXAMPLE 2.5. Consider $\mathscr{Z} = \langle Z; 0, 1, \mathrm{unary}-, +;\rangle$ and $\mathscr{N} = \langle N; 0, 1, +, \pi_1, \pi_2, \langle\,\rangle;\rangle$. Let τ be the translation map τ_2 defined in Example 2.1. Let $\mathscr{F}$ assign to each $f \in \mathrm{Fun}_{\mathscr{Z}}$ a partial function as follows: $\mathscr{F}(0)(\,) = 0$, $\mathscr{F}(1)(\,) = 2$, $\mathscr{F}(\mathrm{unary}-)(x) = \langle \pi_2(x), \pi_1(x)\rangle$, $\mathscr{F}(+)(x, y) = \langle \pi_1(x) + \pi_1(y), \pi_2(x) + \pi_2(y)\rangle$. Thus, $\mathscr{Z} \leqslant^{\exp}_{\tau_2} \mathscr{N}$.

Note that $\rho_{\mathscr{F}}(1) = 2$, $\rho_{\mathscr{F}}(1 + 1) = 5$, $\rho_{\mathscr{F}}(-1) = 1$, and $\rho_{\mathscr{F}}((1 + 1) + (-1)) = 8$. Thus, $\rho_{\mathscr{F}}(1) \neq \rho_{\mathscr{F}}((1 + 1) + (-1))$, so that different ways of generating an element of the represented algebra can produce different representations in the representing algebra.

In order to prove composition results in Section III, it is helpful also to define a "simulation" map from $\mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{A})}$ to $\mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{A}')}$. Intuitively, for every generation of an element of $\mathrm{Dom}_{\mathscr{A}}$, a simulating generation of a representing element in $\mathrm{Dom}_{\mathscr{A}'}$ is selected.

Let Fun be a set of function symbols. If $\{e\} \cup \{e_i : i \in I\} \subseteq V\,\mathrm{Exp}(\mathrm{Fun})$, then $e(e_i \mid v_i)_{i \in I}$ denotes the expression obtained by replacing in e, each occurrence of the variable v_i with expression e_i, for all $i \in I$. $e(e' \mid v_j)$ is used to abbreviate $e(e' \mid v_i)_{i \in \{j\}}$.

If $\mathscr{A}$ is an algebra, Fun a set of function symbols and $\mathscr{E}$ an arity-preserving total mapping from Fun to the set of expression assignments over $\mathscr{A}$, $\mathscr{F}(f) = fn_{\mathscr{E}(f)}$ for all f, then a partial mapping $\sigma_{\mathscr{E}} : V\,\mathrm{Exp}_0(\mathrm{Fun}) \to V\,\mathrm{Exp}_0(\mathrm{Fun}_{\mathscr{A}})$ is defined inductively as follows. If c is a constant symbol, then $\sigma_{\mathscr{E}}(c) = \mathscr{E}(c)(\,)$. If $e_1, \ldots, e_n \in V\,\mathrm{Exp}_0(\mathrm{Fun})$, then $\sigma_{\mathscr{E}}(f(e_1, \ldots, e_n)) = \mathscr{E}(f)(\rho_{\mathscr{F}}(e_1), \ldots, \rho_{\mathscr{F}}(e_n))(\sigma_{\mathscr{E}}(e_i) \mid v_i)_{i \in \{1, \ldots, n\}}$.

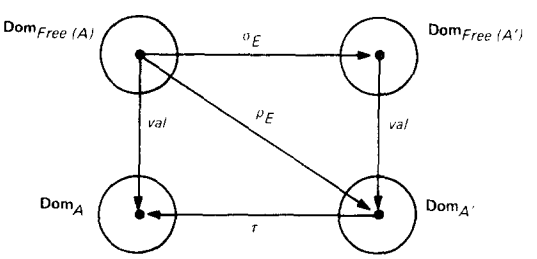
If $\mathscr{E}$, $\mathscr{F}$ are as in the preceding paragraph, $\rho_{\mathscr{E}}$ will sometimes be written in place of $\rho_{\mathscr{F}}$.

Remark 2.1. If $\mathscr{A} \leqslant^{\exp}_{\tau, \mathscr{E}} \mathscr{A}'$, then it is immediate from the definition of $\sigma_{\mathscr{E}}$ that $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{A}')}(\sigma_{\mathscr{E}}(f(e_1, \ldots, e_n)) : \sigma_{\mathscr{E}}(\{e_1, \ldots, e_n\})) \leqslant \mathrm{size}(\mathscr{E}(f)(\rho_{\mathscr{F}}(e_1), \ldots, \rho_{\mathscr{F}}(e_n)))$ for every $e_1, \ldots, e_n \in \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{A})}$ and $f \in \mathrm{Fun}_{\mathscr{F}_{free}(\mathscr{A})}$ with $f(e_1, \ldots, e_n)$ defined.

EXAMPLE 2.6. In Example 2.5, let $\mathscr{E}$ be the straightforward mapping which assigns to each $f \in \mathrm{Fun}_{\mathscr{I}}$ a (constant-valued) expression assignment as follows: $\mathscr{E}(0)(\,) = 0$, $\mathscr{E}(1)(\,) = 1 + 1$, $\mathscr{E}(\mathrm{unary}-)(x) = \langle \pi_2(v_1), \pi_1(v_1) \rangle$, and $\mathscr{E}(+)(x, y) = \langle \pi_1(v_1) + \pi_1(v_2), \pi_2(v_1) + \pi_2(v_2) \rangle$. (The values represented on the right-hand sides of these equations are all formal expressions.) Clearly, $\mathscr{Z} \leqslant^{\exp}_{\tau, \mathscr{E}} \mathscr{N}$ and $fn_{\mathscr{E}(f)} = \mathscr{F}(f)$ for all f.

Note that $\sigma_{\mathscr{E}}(1) = 1 + 1$, $\sigma_{\mathscr{E}}(1 + 1) = \langle \pi_1(1 + 1) + \pi_1(1 + 1), \pi_2(1 + 1) + \pi_2(1 + 1) \rangle$, $\sigma_{\mathscr{E}}(-1) = \langle \pi_2(1 + 1), \pi_1(1 + 1) \rangle$, and $\sigma_{\mathscr{E}}((1 + 1) + (-1)) = \langle \pi_1(\langle \pi_1(1 + 1) + \pi_1(1 + 1), \pi_2(1 + 1) + \pi_2(1 + 1) \rangle) + \pi_1(\langle \pi_2(1 + 1), \pi_1(1 + 1) \rangle), \pi_2(\langle \pi_1(1 + 1) + \pi_1(1 + 1), \pi_2(1 + 1) + \pi_2(1 + 1) \rangle) + \pi_2(\langle \pi_2(1 + 1), \pi_1(1 + 1) \rangle) \rangle$. Clearly, $\sigma_{\mathscr{E}}(1) \neq \sigma_{\mathscr{E}}((1 + 1) + (-1))$, since they are different formal expressions. Thus, different ways of generating an element of the represented algebra can produce different ways of generating a representing element in the representing algebra.

THEOREM 2.2. *If $\mathscr{A} \leqslant^{\exp}_{\tau, \mathscr{E}} \mathscr{A}'$, then the following diagram commutes:*



Proof. Left to the reader. ∎

Relative Complexity of Algebras

The complexity of an algebra, $\mathscr{A}$, relative to another, $\mathscr{A}'$, is now defined. Several types of complexity can be measured, but always expressed in terms of the same parameter. The parameter to be used will be the size measure defined above, applied to $\mathscr{F}_{ree}(\mathscr{A})$. In [7], both running time and number of representations per element are measured; here, running time only is considered.

For flowchart programs, there is a natural "path length" running time measure. Namely, for flowchart program P with n inputs over algebra $\mathscr{A}$, write $L_P\colon (\mathrm{Dom}_{\mathscr{A}})^n \to N$ for the partial function giving the number of steps executed by the flowchart on each possible n-tuple of inputs. This definition applies to both function and relation flowcharts.

Let $\mathscr{A}$ and $\mathscr{A}'$ be arbitrary algebras, $t\colon N \to R^+$ (the non-negative real numbers), τ a translation map between $\mathscr{A}$ and $\mathscr{A}'$, $\mathscr{E}$ a total, arity-preserving mapping from Fun to the set of expression assignments over $\mathscr{A}'$, and $\mathscr{P}$ a total, arity-preserving mapping from $\mathrm{Fun}_{\mathscr{A}} \cup \mathrm{Rel}_{\mathscr{A}}$ to the set of flowcharts over $\mathscr{A}'$. Write $\mathscr{A} \leqslant^{\mathrm{flow}}_{\tau,\mathscr{E},\mathscr{P}} \mathscr{A}'$ *with complexity* t provided $\mathscr{A} \leqslant^{\mathrm{flow}}_{\tau,\mathscr{E},\mathscr{P}} \mathscr{A}'$ and the following holds. For each $p \in \mathrm{Fun}_{\mathscr{A}} \cup \mathrm{Rel}_{\mathscr{A}}$ of n arguments, and for all $e_1,\ldots,e_n \in \mathrm{Dom}_{\mathscr{F}_{ree}(\mathscr{A})}$ such that $p(e_1,\ldots,e_n)$ is defined, it is the case that $L_{\mathscr{P}(p)}(\rho_{\mathscr{E}}(e_1),\ldots,\rho_{\mathscr{E}}(e_n)) \leqslant t(\mathrm{size}_{\mathscr{F}_{ree}(\mathscr{A})}(\{e_1,\ldots,e_n\}))$. In particular, if p is a constant symbol, then $L_{\rho(p)}(\,) \leqslant t(0)$.

Recall that $\rho_{\mathscr{E}}$ selects that representation in $\mathrm{Dom}_{\mathscr{A}'}$ which is produced for a particular way of generating an object in $\mathrm{Dom}_{\mathscr{A}}$. Thus, the running time of the given flowcharts on these selected representations is bounded in terms of the original way of generating the object. In this case, as in all other complexity bounds in this paper and in [7], a resource of interest is bounded in terms of the size of the input values in the represented algebra, and that size is measured in the associated free algebra. This latter convention accomodates use of infinitely many representations in $\mathscr{A}'$ for each element of $\mathscr{A}$, by permitting longer computation times for those representations which are only reached as a result of the simulation of longer sequences of operations in $\mathscr{A}$. This convention is consistent with the data structure results in [13, 18]. (Actually, in [1], the relations are counted in addition to the functions, a minor technical difference for the situations considered here). t has its values in the reals rather than in N because in [7], we consider functions such as $t(n) = \sqrt{n}$.

Write $\mathscr{A} \leqslant^{\mathrm{flow}}_{\tau,\mathscr{E}} \mathscr{A}'$ with complexity t provided $(\exists \mathscr{P})$ $[\mathscr{A} \leqslant^{\mathrm{flow}}_{\tau,\mathscr{E},\mathscr{P}} \mathscr{A}'$ with complexity $t]$, and $\mathscr{A} \leqslant^{\mathrm{flow}}_{\tau} \mathscr{A}'$ *with complexity* t provided $(\exists \mathscr{E})$ $[\mathscr{A} \leqslant^{\mathrm{flow}}_{\tau,\mathscr{E}} \mathscr{A}'$ with complexity $t]$.

An equivalent way of stating the definition of "$\mathscr{A} \leqslant^{\mathrm{flow}}_{\tau,\mathscr{E},\mathscr{P}} \mathscr{A}'$ with complexity t" is to use $L_{\mathscr{P}(p)}(\sigma_{\mathscr{E}}(e_1),\ldots,\sigma_{\mathscr{E}}(e_n))$ in place of $L_{\mathscr{P}(p)}(\rho_{\mathscr{E}}(e_1),\ldots,\rho_{\mathscr{E}}(e_n))$ in the above definition. This definition is equivalent because a flowchart computes in closely corresponding ways over an algebra and over its free version.

It remains to give a definition for relative complexity using expression assignments. For expression assignment E with n arguments over algebra $\mathscr{A}$, write $L_E\colon (\mathrm{Dom}_{\mathscr{A}})^n \to N$ for the partial function mapping $(x_1,\ldots,x_n)$ to $\mathrm{size}(E(x_1,\ldots,x_n))$. Now let $\mathscr{A}$, $\mathscr{A}'$, t, τ, $\mathscr{E}$ be as in the definition of relative flowchart complexity. Write

$\mathscr{A} \leqslant^{\exp}_{\tau,\mathscr{E}} \mathscr{A}'$ *with strong complexity* t provided $\mathscr{A} \leqslant^{\exp}_{\tau,\mathscr{E}} \mathscr{A}'$, and the following condition holds. For each $f \in \mathrm{Fun}_{\mathscr{A}}$ of n arguments and for all $e_1,..., e_n \in \mathrm{Dom}_{\mathscr{F}_{tree}(\mathscr{A})}$ for which $f(e_1,..., e_n)$ is defined, it is the case that $L_{\mathscr{E}(f)}(\rho_{\mathscr{E}}(e_1),..., \rho_{\mathscr{E}}(e_n)) \leqslant t(\mathrm{size}_{\mathscr{F}_{tree}(\mathscr{A})}(\{e_1,..., e_n\}))$. In particular, if f is a constant symbol, this condition says that size $(\mathscr{E}(f)(\,)) \leqslant t(0)$. No conditions are specified for relations $r \in \mathrm{Rel}_{\mathscr{A}}$.

Another possible definition for relative complexity using expression assignments uses the size measure in $\mathscr{A}'$ for the resource being measured. Let $\mathscr{A}$, $\mathscr{A}'$, t, τ, $\mathscr{E}$ be as above. Write $\mathscr{A} \leqslant^{\exp}_{\tau,\mathscr{E}} \mathscr{A}'$ *with weak complexity* t provided $\mathscr{A} \leqslant^{\exp}_{\tau,\mathscr{E}} \mathscr{A}'$ and the following condition holds. For each $f \in \mathrm{Fun}_{\mathscr{A}}$ of n arguments and all $e_1,..., e_n \in \mathrm{Dom}_{\mathscr{F}_{tree}(\mathscr{A})}$ for which $f(e_1,..., e_n)$ is defined, it is the case that

$$\mathrm{size}_{\mathscr{A}'}(\rho_{\mathscr{E}}(f(e_1,..., e_n)): \rho_{\mathscr{E}}(\{e_1,..., e_n\})) \leqslant t(\mathrm{size}_{\mathscr{F}_{tree}(\mathscr{A})}(\{e_1,..., e_n\})).$$

Write $\mathscr{A} \leqslant^{\exp}_{\tau} \mathscr{A}'$ *with strong* (resp. *weak*) *complexity* t provided $(\exists \mathscr{E})[\mathscr{A} \leqslant^{\exp}_{\tau,\mathscr{E}} \mathscr{A}'$ with strong (resp. weak) complexity $t]$.

Note that if $\mathscr{A} \leqslant^{\exp}_{\tau,\mathscr{E}} \mathscr{A}'$, then $\mathrm{size}_{\mathscr{A}'}(\rho_{\mathscr{E}}(f(e_1,..., e_n)): \rho_{\mathscr{E}}(\{e_1,..., e_n\}))$, $\leqslant \mathrm{size}_{\mathscr{F}_{tree}(\mathscr{A}')}(\sigma_{\mathscr{E}}(f(e_1,..., e_n)): \sigma_{\mathscr{E}}(\{e_1,..., e_n\}))$ by Theorems 2.1(f) and 2.2, $\leqslant L_{\mathscr{E}(f)}(\rho_{\mathscr{E}}(e_1),..., \rho_{\mathscr{E}}(e_n))$ by the definition of $L_{\mathscr{E}(f)}$ and Remark 2.1.Thus, $\mathscr{A} \leqslant^{\exp}_{\tau,\mathscr{E}} \mathscr{A}'$ with strong complexity t implies that $\mathscr{A} \leqslant^{\exp}_{\tau,\mathscr{E}} \mathscr{A}'$ with weak complexity t.

THEOREM 2.3. *If* $\mathscr{A} \leqslant^{\exp}_{\tau} \mathscr{A}'$ *with weak complexity* t, *then* $\mathscr{A} \leqslant^{\exp}_{\tau} \mathscr{A}'$ *with strong complexity* t.

Proof. Intuitively, it suffices to choose "shortest possible" expressions. Specifically, $\mathscr{E}'$ is chosen so that $\mathscr{A} \leqslant^{\exp}_{\tau,\mathscr{E}'} \mathscr{A}'$ with weak complexity t. Write $\mathscr{F}'(f) = fn_{\mathscr{E}'(f)}$ for all f. $\mathscr{E}$ is constructed as follows.

Consider $f \in \mathrm{Fun}_{\mathscr{A}}$ with n arguments, and $x_1,..., x_n \in \mathrm{Dom}_{\mathscr{A}'}$. If $\mathscr{F}'(f)(x_1,..., x_n)$ is undefined, then $\mathscr{E}(f)(x_1,..., x_n)$ is undefined. Otherwise, let $k = \mathrm{size}_{\mathscr{A}'}(\mathscr{F}'(f)(x_1,..., x_n): \{x_1,..., x_n\})$. k is finite. Choose a k-step derivation of $\mathscr{F}'(f)(x_1,..., x_n)$ from $\{x_1,..., x_n\}$. Then $\mathscr{E}(f)(x_1,..., x_n)$ is the expression in $V\,\mathrm{Exp}_n(\mathrm{Fun}_{\mathscr{A}'})$ describing this derivation.

Clearly, $fn_{\mathscr{E}(f)} = \mathscr{F}'(f)$, so that $\mathscr{A} \leqslant^{\exp}_{\tau,\mathscr{E}} \mathscr{A}'$. In order to verify the complexity bound, fix $f \in \mathrm{Fun}_{\mathscr{A}}$ of n arguments and $e_1,..., e_n \in \mathrm{Dom}_{\mathscr{F}_{tree}(\mathscr{A})}$ such that $f(e_1,..., e_n)$ is defined. Then $L_{\mathscr{E}(f)}(\rho_{\mathscr{E}}(e_1),..., \rho_{\mathscr{E}}(e_n)) = L_{\mathscr{E}(f)}(\rho_{\mathscr{F}'}(e_1),..., \rho_{\mathscr{F}'}(e_n))$, $= \mathrm{size}_{\mathscr{A}'}(\mathscr{F}'(f)(\rho_{\mathscr{F}'}(e_1),..., \rho_{\mathscr{F}'}(e_1),..., \rho_{\mathscr{F}'}(e_n)): \rho_{\mathscr{F}'}(\{e_1,..., e_n\}))$ by the construction of $\mathscr{E}$, $= \mathrm{size}_{\mathscr{A}'}(\rho_{\mathscr{F}'}(f(e_1,..., e_n)): \rho_{\mathscr{F}'}(\{e_1,..., e_n\}))$ by definition of $\rho_{\mathscr{F}'}$, $= \mathrm{size}_{\mathscr{A}'}(\rho_{\mathscr{E}'}(f(e_1,..., e_n)): \rho_{\mathscr{E}'}(\{e_1,..., e_n\}))$, $\leqslant t(\mathrm{size}_{\mathscr{F}_{tree}(\mathscr{A})}(\{e_1,..., e_n\}))$ by hypothesis. ∎

Thus, $\mathscr{A} \leqslant^{\exp}_{\tau} \mathscr{A}'$ *with complexity* t is sometimes written, without the adjective "weak" or "strong."

Note that $\mathscr{A} \leqslant^{\mathrm{flow}}_{\tau,\mathscr{E}} \mathscr{A}'$ with complexity t implies that $\mathscr{A} \leqslant^{\exp}_{\tau,\mathscr{E}} \mathscr{A}'$ with strong complexity t, because a flowchart must execute at least enough steps to apply the expressions in its derived expression assignment to its inputs. A similar remark holds for most other programming languages in place of flowcharts. Several lower bounds

on expression assignment complexity are proved in [7]; thus, they imply similar lower bounds for other programming languages.

Note also that the definitions of this section treat operations with different numbers of arguments uniformly.

III. Composition Theorems

One goal of this work is the ability to combine complexity analyses of different levels of an algorithm; thus, theorems about composition of reducibilities are important. In this section, several composition results are proved for use in Section IV.

Substitution of expression assignments is first described. Let $\mathscr{A}$ be an algebra, Fun a set of function symbols, and $\mathscr{E}$ a total, arity-preserving mapping from Fun to the set of expression assignments over $\mathscr{A}$. Define inductively two partial mappings $\mu_{\mathscr{E},n}: V\,\mathrm{Exp}_n(\mathrm{Fun}) \times (\mathrm{Dom}_{\mathscr{A}})^n \to V\,\mathrm{Exp}_n(\mathrm{Fun})$ and $\nu_{\mathscr{E},n}: V\,\mathrm{Exp}_n(\mathrm{Fun}) \times (\mathrm{Dom}_{\mathscr{A}})^n \to \mathrm{Dom}_{\mathscr{A}}$. The mapping $\mu_{\mathscr{E},n}$ produces, for each computation applying functions in Fun to n arguments, a corresponding computation applying functions in $\mathrm{Fun}_{\mathscr{A}}$ to n arguments; the new computation depends not only on the original computation but also on the arguments in $\mathscr{A}$. The mapping $\nu_{\mathscr{E},n}$ produces the results of the computations given by $\mu_{\mathscr{E},n}$. In the following let $\bar{x}$ denote $x_1,\ldots,x_n$ and $\bar{e}$ and $\bar{y}$ denote $e_1,\ldots,e_m$ and $y_1,\ldots,y_m$, respectively. Define $\mu_{\mathscr{E},n}(v_i,\bar{x}) = v_i$, $1 \leqslant i \leqslant n$, and $\mu_{\mathscr{E},n}(c,\bar{x}) = \mathscr{E}(c)(\,)$ for each constant symbol c. If $e_1,\ldots,e_m$ are in $V\,\mathrm{Exp}_n(\mathrm{Fun})$ and $f \in \mathrm{Fun}$ has m arguments, then $\mu_{\mathscr{E},n}(f(\bar{e}),\bar{x}) = (\mathscr{E}(f)(\bar{y}))(\mu_{\mathscr{E},n}(e_i,\bar{x}) \mid v_i)$, where each $y_i = \nu_{\mathscr{E},n}(e_i,\bar{x})$. Finally, $\nu_{\mathscr{E},n}(e,\bar{x}) = fn_{\mu_{\mathscr{E}}(e,\bar{x})}(\bar{x})$.

EXAMPLE 3.1. Consider Example 2.4. Let e be the expression $(v_1 + 1) + (-v_2)$ over $\mathscr{Z}$. Let $x_1 = 2$ and $x_2 = 6$. Then

$$\mu_{\mathscr{E},2}(v_1,2,6) = v_1, \qquad \nu_{\mathscr{E},2}(v_1,2,6) = 2,$$

$$\mu_{\mathscr{E},2}(v_2,2,6) = v_2, \qquad \nu_{\mathscr{E},2}(v_2,2,6) = 6,$$

$$\mu_{\mathscr{E},2}(1,2,6) = 1 + 1, \qquad \nu_{\mathscr{E},2}(1,2,6) = 2,$$

$$\begin{aligned}\mu_{\mathscr{E},2}(v_1+1,2,6) &= (\mathscr{E}(+)(2,2))(v_1 \mid v_1)(1+1 \mid v_2) \\ &= (v_1 + v_2)(v_1 \mid v_1)(1+1 \mid v_2) \\ &= v_1 + (1+1),\end{aligned} \qquad \nu_{\mathscr{E},2}(v_1+1,2,6) = 4,$$

$$\begin{aligned}\mu_{\mathscr{E},2}(-v_2,2,6) &= (\mathscr{E}(\text{unary}-)(6))(v_2 \mid v_1) \\ &= (v_1 \dot{-} 1)(v_2 \mid v_1) = v_2 \dot{-} 1,\end{aligned} \qquad \nu_{\mathscr{E},2}(-v_2,2,6) = 5,$$

$$\begin{aligned}\mu_{\mathscr{E},2}((v_1+1)+(-v_2),2,6) &= (\mathscr{E}(+)(4,5))(v_1+(1+1) \mid v_1)(v_2 \dot{-} 1 \mid v_2) \\ &= (v_2 \dot{-} v_1)(v_1+(1+1) \mid v_1)(v_2 \dot{-} 1 \mid v_2) \\ &= (v_2 \dot{-} 1) \dot{-} (v_1 + (1+1),\end{aligned}$$

$$\nu_{\mathscr{E},2}((v_1+1)+(-v_2),2,6) = 1.$$

The following theorem shows that the new mappings produce results which correspond naturally to the results of previously defined mappings.

THEOREM 3.1. *Assume* $\mathscr{A} \leqslant^{\exp}_{\tau,E} \mathscr{A}'$, $e_1,\ldots,e_n \in \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{A})}$, $e \in V\,\mathrm{Exp}_n(\mathrm{Fun}_{\mathscr{A}})$, *and* $e(e_i \mid v_i)_{i\in\{1,\ldots,n\}} \in \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{A})}$. Then (a) and (b) hold.

(a) $\mu_{\mathscr{E},n}(e, \rho_{\mathscr{E}}(e_1),\ldots,\rho_{\mathscr{E}}(e_n))(\sigma_{\mathscr{E}}(e_i) \mid v_i)_{i\in\{1,\ldots,n\}} = \sigma_{\mathscr{E}}(e(e_i \mid v_i)_{i\in\{1,\ldots,n\}})$.

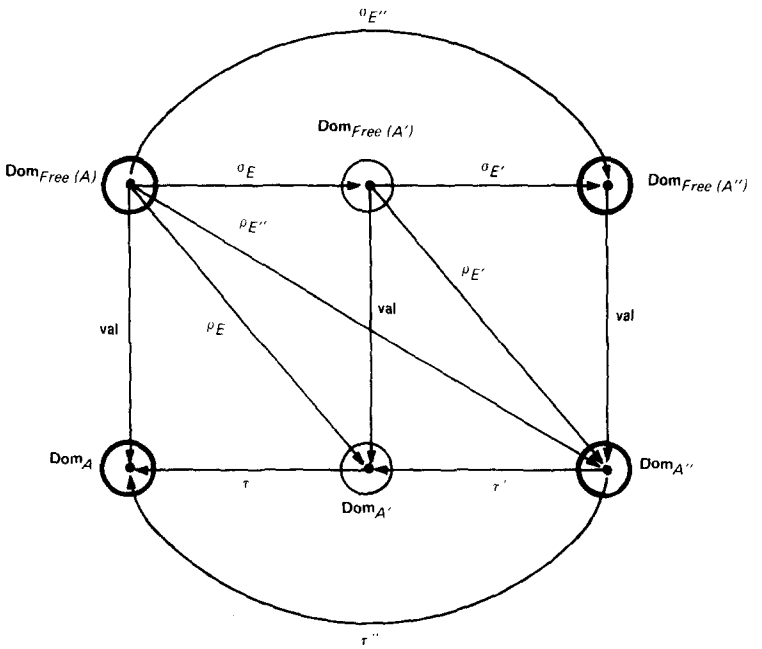
(b) $\nu_{\mathscr{E},n}(e, \rho_{\mathscr{E}}(e_1),\ldots,\rho_{\mathscr{E}}(e_n)) = \rho_{\mathscr{E}}(e(e_i \mid v_i)_{i\in\{1,\ldots,n\}})$.

Proof. Left to the reader. ∎

Now assume $\mathscr{A} \leqslant^{\exp}_{\tau,\mathscr{E}} \mathscr{A}'$ and $\mathscr{A}' \leqslant^{\exp}_{\tau',\mathscr{E}'} \mathscr{A}''$. Define $\mathscr{E} \oplus \mathscr{E}'$, a total, arity-preserving mapping from $\mathrm{Fun}_{\mathscr{A}}$ to the set of expression assignments over $\mathscr{A}''$, by substitution from $\mathscr{E}$ and $\mathscr{E}'$: given $f \in \mathrm{Fun}_{\mathscr{A}}$ of n arguments, $x_1,\ldots,x_n \in \mathrm{Dom}_{\mathscr{A}''}$, define $\mathscr{E} \oplus \mathscr{E}'(f)(x_1,\ldots,x_n) = \mu_{\mathscr{E}',n}(\mathscr{E}(f)(\tau'(x_1),\ldots,\tau'(x_n)), x_1,\ldots,x_n)$.

A straightforward definition (here omitted) can also be given for substitution of flowcharts; then if $\mathscr{A} \leqslant^{\mathrm{flow}}_{\tau,\mathscr{E},\mathscr{P}} \mathscr{A}'$ and $\mathscr{A}' \leqslant^{\mathrm{flow}}_{\tau',\mathscr{E}',\mathscr{P}'} \mathscr{A}''$, a total, arity-preserving mapping $\mathscr{P} \oplus \mathscr{P}'$ from $\mathrm{Fun}_{\mathscr{A}}$ to the set of flowcharts over $\mathscr{A}''$ can be given. These definitions satisfy the following:

THEOREM 3.2. (a) *If* $\mathscr{A} \leqslant^{\exp}_{\tau,\mathscr{E}} \mathscr{A}'$, $\mathscr{A}' \leqslant^{\exp}_{\tau',\mathscr{E}'} \mathscr{A}''$, $\tau'' = \tau \circ \tau'$ *and* $\mathscr{E}'' = \mathscr{E} \oplus \mathscr{E}'$, *then* $\mathscr{A} \leqslant^{\exp}_{\tau'',\mathscr{E}''} \mathscr{A}''$. *Moreover, the following diagram commutes*:



(b) *If* $\mathscr{A} \leqslant^{\text{flow}}_{\tau,\mathscr{E},\mathscr{P}} \mathscr{A}'$ *and* $\mathscr{A}' \leqslant^{\text{flow}}_{\tau',\mathscr{E}',\mathscr{P}'} \mathscr{A}''$, $\tau'' = \tau \circ \tau'$, $\mathscr{E}'' = \mathscr{E} \oplus \mathscr{E}'$, $\mathscr{P}'' = \mathscr{P} \oplus \mathscr{P}'$, *then* $\mathscr{A} \leqslant^{\text{flow}}_{\tau'',\mathscr{E}'',\mathscr{P}''} \mathscr{A}'$.

Proof. Left to the reader. ∎

Next, general theorems for composition of complexity-bounded expression assignment reducibilities are given.

THEOREM 3.3. *Assume* $\mathscr{A} \leqslant^{\text{exp}}_{\tau,\mathscr{E}} \mathscr{A}'$ *with strong complexity* t, *and that* $\text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(\sigma_{\mathscr{E}}(A)) \leqslant s(\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(A))$ *for all* $A \subseteq \text{Dom}_{\mathscr{F}_{free}(\mathscr{A})}$. *Assume* $\mathscr{A}' \leqslant^{\text{exp}}_{\tau',\mathscr{E}'} \mathscr{A}''$ *with strong complexity* t', *where* t' *is nondecreasing. Then* $\mathscr{A} \leqslant^{\text{exp}}_{\tau'',\mathscr{E}''} \mathscr{A}''$ *with strong complexity* t'', *where* $\tau'' = \tau \circ \tau'$, $\mathscr{E}'' = \mathscr{E} \oplus \mathscr{E}'$, *and* $t''(n) = \sum_{i=0}^{\lfloor t(n) \rfloor - 1} t'(\lfloor s(n) \rfloor + i)$.

Proof. $\mathscr{A} \leqslant^{\text{exp}}_{\tau'',\mathscr{E}''} \mathscr{A}''$, by Theorem 3.2(a). It remains to show the complexity bound.

Consider $f \in \text{Fun}_{\mathscr{A}}$ of n arguments and $e_1, \ldots, e_n \in \text{Dom}_{\mathscr{F}_{free}(\mathscr{A})}$ such that $f(e_1, \ldots, e_n)$ is defined. Write m for $\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(\{e_1, \ldots, e_n\})$.

By hypothesis, $L_{\mathscr{E}(f)}(\rho_{\mathscr{E}}(e_1), \ldots, \rho_{\mathscr{E}}(e_n)) \leqslant t(m)$. Thus, there exists a shortest sequence $\{A_i\}_{i=1}^{k}$ of subsets of $V\text{Exp}_n(\text{Fun}_{\mathscr{A}'})$, $k \leqslant t(m)$, with $A_0 = \{v_1, \ldots, v_n\}$, $\mathscr{E}(f)(\rho_{\mathscr{E}}(e_1), \ldots, \rho_{\mathscr{E}}(e_n)) \in A_k$, and $\text{size}_{(V\text{Exp}(\text{Fun}_{\mathscr{A}'}); \text{Fun}_{\mathscr{A}'} \cup \{v_i\}_{i=1}^{\infty});}(A_{i+1} : A_i) = 1$ for all i, $0 \leqslant i \leqslant k-1$. For each i, there exists $f_i \in \text{Fun}_{\mathscr{A}'}$ of l_i arguments, with $A_{i+1} = A_i \cup \{f_i(a_{i,1}, \ldots, a_{i,l_i})\}$ and $a_{i,1}, \ldots, a_{i,l_i}$ in A_i. If $e \in V\text{Exp}_n(\text{Fun}_{\mathscr{A}'})$, let $\text{sub}(e)$ denote $e(\sigma_{\mathscr{E}}(e_i) \mid v_i)_{i=1}^{n}$.

Let B_i denote $\text{sub}(A_i)$, $0 \leqslant i \leqslant k$. Then $B_{i+1} = B_i \cup \{\text{sub}(f_i(a_{i,1}, \ldots, a_{i,l_i}))\} = B_i \cup \{f_i(\text{sub}(a_{i,1}), \ldots, \text{sub}(a_{i,l_i}))\}$, where $\{\text{sub}(a_{i,1}), \ldots, \text{sub}(a_{i,l_i})\} \subseteq B_i$. Thus $\text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(B_{i+1} : B_i) \leqslant 1$. We claim that $\text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(B_i) \leqslant s(m) + i$, $0 \leqslant i \leqslant k$. This is because $\text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(B_0) = \text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(\text{sub}(A_0)) = \text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(\sigma_{\mathscr{E}}(\{e_1, \ldots, e_n\})) \leqslant s(m)$ by hypothesis; also, $\text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(B_{i+1}) \leqslant \text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(B_i) + \text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(B_{i+1} : B_i) \leqslant (s(m) + i) + 1$ for all i, $0 \leqslant i \leqslant k-1$.

Thus, by the hypotheses, $L_{\mathscr{E}'(f_i)}(\rho_{\mathscr{E}'}(\text{sub}(a_{i,1})), \ldots, \rho_{\mathscr{E}'}(\text{sub}(a_{i,l_i}))) \leqslant t'(\lfloor s(m) \rfloor + i)$.

Now, $L_{\mathscr{E}''(f)}(\rho_{\mathscr{E}''}(e_1), \ldots, \rho_{\mathscr{E}''}(e_n)) = \text{size}(\mu_{\mathscr{E}',n}(\mathscr{E}(f)(\tau'(\rho_{\mathscr{E}''}(e_1)), \ldots, \tau'(\rho_{\mathscr{E}''}(e_n))), \rho_{\mathscr{E}''}(e_1), \ldots, \rho_{\mathscr{E}''}(e_n))$ by definition of $\mathscr{E}''$, $= \text{size}(\mu_{\mathscr{E}',n}(\mathscr{E}(f)(\rho_{\mathscr{E}}(e_1), \ldots, \rho_{\mathscr{E}}(e_n)), \rho_{\mathscr{E}''}(e_1), \ldots, \rho_{\mathscr{E}''}(e_n)))$ by Theorem 3.2(a), $\leqslant \sum_{i=0}^{k-1} \text{size}(\mathscr{E}'(f_i)(\nu_{\mathscr{E}',n}(a_{i,1}, \rho_{\mathscr{E}''}(e_1), \ldots, \rho_{\mathscr{E}''}(e_n)), \ldots, \nu_{\mathscr{E}',n}(a_{i,l_i}, \rho_{\mathscr{E}''}(e_1), \ldots, \rho_{\mathscr{E}''}(e_n))))$ by definition of $\mu_{\mathscr{E}',n}$, $= \sum_{i=0}^{k-1} \text{size}(\mathscr{E}'(f_i)(\rho_{\mathscr{E}'}(\text{sub}(a_{i,1})), \ldots, \rho_{\mathscr{E}'}(\text{sub}(a_{i,l_i}))))$ by Theorems 3.1 and 3.2(a), $\leqslant \sum_{i=0}^{k-1} t'(\lfloor s(m) \rfloor + i)$ by the preceding paragraph, $\leqslant \sum_{i=0}^{\lfloor t(m) \rfloor - 1} t'(\lfloor s(m) \rfloor + i)$. ∎

The next theorem provides a bound on size growth allowed by a simulation map.

THEOREM 3.4. *Assume* $\mathscr{A} \leqslant^{\text{exp}}_{\tau,\mathscr{E}} \mathscr{A}'$ *with strong complexity* t, *where* t *is nondecreasing (and not identically equal to* 0*). Then for all* $A \subseteq \text{Dom}_{\mathscr{F}_{free}(\mathscr{A})}$, $\text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(\sigma_{\mathscr{E}}(A)) \leqslant \sum_{i=0}^{\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(A)-1} t(i)$.

Proof. If $\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(A) = \infty$, then the sum is infinite and thus the inequality holds by convention. For A with $\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(A)$ finite, we proceed by induction on $\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(A)$.

If $\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(A) = 0$, then $A = \emptyset$. Then $\sigma_{\mathscr{E}}(A) = \emptyset$ and so $\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(\sigma_{\mathscr{E}}(A)) = 0$.

If $\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(A) = k + 1$, $k \geqslant 0$, then there exists $C \subseteq \text{Dom}_{\mathscr{F}_{free}(\mathscr{A})}$, $\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(C) = k$, $f \in \text{Fun}_{\mathscr{A}}$ of n arguments, $e_1, \ldots, e_n \in C$, and $A = C \cup \{f(e_1, \ldots, e_n)\}$. Then $\text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(\sigma_{\mathscr{E}}(A)) \leqslant \text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(\sigma_{\mathscr{E}}(C)) + \text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(\sigma_{\mathscr{E}}(f(e_1, \ldots, e_n)) : \sigma_{\mathscr{E}}(C))$ by Theorem 2.1(a, c, d), which is at most $\sum_{i=0}^{\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(C)-1} t(i) + \text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(\sigma_{\mathscr{E}}(f(e_1, \ldots, e_n)) : \sigma_{\mathscr{E}}(C))$ by inductive hypothesis. The final term is at most $\text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(\sigma_{\mathscr{E}}(f(e_1, \ldots, e_n)) : \sigma_{\mathscr{E}}(\{e_1, \ldots, e_n\}))$ by Theorem 2.1(b), $\leqslant t(\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(\{e_1, \ldots, e_n\}))$ by hypothesis and Remark 2.1, $\leqslant t(\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(C))$ by Theorem 2.1(a) and the monotonicity of t. Collecting terms yields the result. ∎

Thus, a very general composition theorem for expression assignment programs can be stated. Although the bound seems messy (for instance, not in closed form), it suffices in [7] to yield reasonably good closed form upper and lower bounds.

THEOREM 3.5. *Assume* $\mathscr{A} \leqslant_{\tau}^{\exp} \mathscr{A}'$ *with complexity* t, *where* t *is nondecreasing (and not identically equal to* 0*). Assume* $\mathscr{A}' \leqslant_{\tau'}^{\exp} \mathscr{A}''$ *with complexity* t', *where* t' *is nondecreasing. Then* $\mathscr{A} \leqslant_{\tau\circ\tau'}^{\exp} \mathscr{A}''$ *with complexity* t'', *where* $t''(n) = \sum_{l=0}^{\lfloor t(n) \rfloor - 1} t'(\lfloor \sum_{j=0}^{n-1} t(j) \rfloor + i)$.

Proof. Choose $\mathscr{E}$ such that $\mathscr{A} \leqslant_{\tau,\mathscr{E}}^{\exp} \mathscr{A}'$ with strong complexity t. Let $s(n) = \sum_{i=0}^{n-1} t(i)$. By Theorem 3.4, $\text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(\sigma_{\mathscr{E}}(\mathscr{A})) \leqslant s(\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(A))$. By Theorem 3.3, $\mathscr{A} \leqslant_{\tau\circ\tau'}^{\exp} \mathscr{A}''$ with complexity t''. ∎

The following two theorems deal with composition of flowchart programs.

THEOREM 3.6. *Assume* $\mathscr{A} \leqslant_{\tau,\mathscr{E},\mathscr{P}}^{\text{flow}} \mathscr{A}'$ *with complexity* t *and that* $\text{size}_{\mathscr{F}_{free}(\mathscr{A}')}(\sigma_{\mathscr{E}}(A)) \leqslant s(\text{size}_{\mathscr{F}_{free}(\mathscr{A})}(A))$ *for all* $A \subseteq \text{Dom}_{\mathscr{F}_{free}(\mathscr{A})}$. *Assume* $\mathscr{A}' \leqslant_{\tau',\mathscr{E}',\mathscr{P}'}^{\text{flow}} \mathscr{A}''$ *with complexity* t', *where* t' *is nondecreasing. Then* $\mathscr{A} \leqslant_{\tau'',\mathscr{E}'',\mathscr{P}''}^{\text{flow}} \mathscr{A}''$ *with complexity* t'', *where* $\tau'' = \tau \circ \tau'$, $\mathscr{E}'' = \mathscr{E} \oplus \mathscr{E}'$, $\mathscr{P}'' = \mathscr{P} \oplus \mathscr{P}'$ *and* $t''(n) = \sum_{l=0}^{\lfloor t(n) \rfloor - 1} t'(\lfloor s(n) \rfloor + i)$.

Proof. A treatment similar to that for Theorem 3.3 is left to the reader. ∎

THEOREM 3.7. *Assume* $\mathscr{A} \leqslant_{\tau}^{\text{flow}} \mathscr{A}'$ *with complexity* t, *where* t *is nondecreasing (and not identically equal to* 0*). Assume* $\mathscr{A}' \leqslant_{\tau'}^{\text{flow}} \mathscr{A}''$ *with complexity* t', *where* t' *is nondecreasing. Then* $\mathscr{A} \leqslant_{\tau\circ\tau'}^{\text{flow}} \mathscr{A}''$ *with complexity* t'', *where* $t''(n) = \sum_{l=0}^{\lfloor t(n) \rfloor - 1} t'(\lfloor \sum_{j=0}^{n-1} t(j) \rfloor + i)$.

Proof. As for Theorem 3.5, using Theorems 3.4 and 3.6. ∎

The final result of this section is used in Section IV, in the proofs of Theorems 4.3 and 4.7.

THEOREM 3.8. *Let* $\mathscr{A}'$, $\mathscr{A}''$ *be algebras,* $\mathscr{A} = \langle \text{Dom}_{\mathscr{A}'}; \text{Fun}_{\mathscr{A}'} \cup \{c_1, \ldots, c_k\}; \text{Rel}_{\mathscr{A}'} \rangle$. *Assume* $\mathscr{A}' \leqslant_{\tau}^{\text{flow}} \mathscr{A}''$ *with complexity* t, *where* t *is nondecreasing and unbounded. Then* $\mathscr{A} \leqslant_{\tau}^{\text{flow}} \mathscr{A}''$ *with complexity* t', *where* $t'(n) = t(n + c)$ *for some constant* c.

Proof. Fix $\mathscr{E}$, $\mathscr{P}$ with $\mathscr{A} \leqslant^{\text{flow}}_{\iota,\mathscr{E},\mathscr{P}} \mathscr{A}'$, where ι is the identity function and $\mathscr{P}(p)$ is the obvious one-step flowchart for every $p \in (\text{Fun}_{\mathscr{A}} \cup \text{Rel}_{\mathscr{A}}) - \{c_1, ..., c_k\}$. Then there exists a constant d such that $\text{size}_{\mathscr{F}ree(\mathscr{A}')}(\sigma_{\mathscr{E}}(A)) \leqslant d + \text{size}_{\mathscr{F}ree(\mathscr{A})}(A)$ for all $A \subseteq \text{Dom}_{\mathscr{F}ree(\mathscr{A})}$. (This is because $\sigma_{\mathscr{E}}(A)$ is constructed from A by substitution of the expression $\mathscr{E}(c_i)(\,)$ for c_i, $1 \leqslant i \leqslant k$. The number of steps needed to generate all of the expressions in $\sigma_{\mathscr{E}}(A)$ is at most the number of steps needed to generate the finite number of expressions $\mathscr{E}(c_i)(\,)$, plus the number of steps required for A.)

Fix $\mathscr{E}'$, $\mathscr{P}'$ such that $\mathscr{A}' \leqslant^{\text{flow}}_{\tau,\mathscr{E}',\mathscr{P}'} \mathscr{A}''$ with complexity t. Let $\mathscr{E}'' = \mathscr{E} \oplus \mathscr{E}'$, $\mathscr{P}'' = \mathscr{P} \oplus \mathscr{P}'$. Choose $c \geqslant d$ so that $t(c) \geqslant L_{\mathscr{P}''(c_i)}(\,)$, $1 \leqslant i \leqslant k$. We show that c has the required property.

First, let $p \in \text{Fun}_{\mathscr{A}'} \cup \text{Rel}_{\mathscr{A}'}$ have n arguments, and let $e_1, ..., e_n \in \text{Dom}_{\mathscr{F}ree(\mathscr{A})}$ such that $p(e_1, ..., e_n)$ is defined. Then $\mathscr{P}''(p) = \mathscr{P}'(p)$. (Here, p is regarded as an operation both of $\mathscr{A}$ and of $\mathscr{A}'$.) Thus, $L_{\mathscr{P}''(p)}(\rho_{\mathscr{E}''}(e_1), ..., \rho_{\mathscr{E}''}(e_n)) = L_{\mathscr{P}'(p)}(\rho_{\mathscr{E}''}(e_1), ..., \rho_{\mathscr{E}''}(e_n)) = L_{\mathscr{P}'(p)}(\rho_{\mathscr{E}''}(e_1), ..., \rho_{\mathscr{E}''}(e_n)) = L_{\mathscr{P}'(p)}(\rho_{\mathscr{E}'}(\sigma_{\mathscr{E}}(e_1)), ..., \rho_{\mathscr{E}'}(\sigma_{\mathscr{E}}(e_n)))$, $\leqslant t(\text{size}_{\mathscr{F}ree(\mathscr{A}')}(\sigma_{\mathscr{E}}(\{e_1, ..., e_n\})))$ by hypothesis, $\leqslant t(d + \text{size}_{\mathscr{F}ree(\mathscr{A})}(\{e_1, ..., e_n\}))$ by the bound calculated above and the monotonicity of t, $\leqslant t(c + \text{size}_{\mathscr{F}ree(\mathscr{A})}(\{e_1, ..., e_n\}))$ by the choice of c.

On the other hand, if $p = c_i$ for some i, then the choice of c immediately yields the needed bound. ∎

IV. Flowchart Complexity of Groups—an Extended Example

In this section, suggestions are given for using flowchart time complexity and our size parameter to classify finitely-generated (abbreviated, f.g.) computable groups. Work in [2, 3, 4, 14, 16] on computable and complexity-bounded algebras is reconsidered to see if our definitions might yield sharper and more natural complexity results.

In [14, 16], the primary emphasis was on computability of the groups; where the latter paper does deal with complexity classification, it is only at a very high level (primitive recursive). In [2–4], the computability definitions of [16] are restricted in the most straightforward way (and also in a second, rather complicated way) to yield definitions for groups of different complexity. Application of very basic group-theoretic constructions seems to cause complexity in their straightforward definition to rise one level in the Grzegorczyk hierarchy, suggesting that this definition is not useful for low-order complexity.

Some definitions from the earlier papers are restated in our notation, for comparison. A group $\mathscr{G}$ is called *Rabin computable* if there is an injection $\kappa: \text{Dom}_{\mathscr{G}} \to \{0, 1\}^*$ such that $\kappa(\text{Dom}_{\mathscr{G}})$ is a recursive set, and such that $\lambda(\kappa(x), \kappa(y))[\kappa(x \circ y)]$ (and therefore $\lambda(\kappa(x))[\kappa(x^{-1})]$) is recursive. (Here λ is Church's lambda-notation: for example, $\lambda(\kappa(x), \kappa(y))[\kappa(x \circ y)]$ denotes the function mapping $\kappa(x)$ and $\kappa(y)$ to $\kappa(x \circ y)$. Also, $\text{Dom}_{\mathscr{G}}$ is the domain of $\mathscr{G}$.) Thus, Rabin's style of definition differs from ours in two ways. First, he specifies a recursiveness condition on the representing set, whereas we never place any constraints on that set.

Second, he does not permit multiple representations of elements, a situation we feel is very natural.

Mal'cev's work [14] is a general study of computable and primitive recursive algebras containing many different possible definitions and relationships between them. For computability, some of his definitions are equivalent to those we use; his definitions are generally more similar to ours than they are to Rabin's. To be specific, a group $\mathscr{G}$ is called *Mal'cev computable* (*Mal'cev primitive recursive*) if there exist a partial, surjective function $\tau: \{0, 1\}^* \to \mathrm{Dom}_{\mathscr{G}}$, two partial recursive (resp. primitive recursive) function $f_\circ$ and f_{-1}, and a partial recursive (resp. primitive recursive) relation $r_=$ such that $f_\circ$, f_{-1} and $r_=$ are τ-simulators of $\circ$, $^{-1}$ and $=$, respectively. As in our work, no recursiveness constraints are placed on the representing set and multiple representations are permitted. However, the simulators $f_\circ$, f_{-1} and $r_=$ in the primitive recursive version of Mal'cev's definition are required to be primitive recursive in the usual sense. Since primitive recursive functions are exactly those computable in primitive recursive time on a Turing machine, it follows that the primitive recursive restriction amounts to a complexity restriction on $\mathscr{G}$ with parameter residing in $\{0, 1\}^*$ rather than in $\mathscr{G}$, i.e., in the representing rather than the represented set. Thus, Mal'cev's style differs from ours.

Cannonito and Gatterdam [2, 3, 4] study complexity definitions based on a straightforward restriction of Rabin's definition. Taking some liberties with, but (we hope) preserving the spirit of, their definition, we say that a group $\mathscr{G}$ is *of* CG *complexity* t (for total $t: N \to N$) if $\mathscr{G}$ is Rabin computable, with $\kappa(\mathrm{Dom}_{\mathscr{G}})$'s characteristic function and total extensions of $\lambda(\kappa(x), \kappa(y))[\kappa(x \circ y)]$ and $\lambda(\kappa(x))[\kappa(x^{-1})]$ all of (Turing) complexity t. Thus, a complexity constraint is placed on the representing set, only single representations are used, and parameters are based in the representing rather than in the represented system. A more restrictive definition "$\mathscr{E}_\alpha$-standard" is not reproduced here because of its unwieldiness.

Our style of definition is now specialized to the present case. Let $\mathscr{B} = \langle \{0, 1\}^*; \varepsilon, 0, 1, 0\mathrm{suc}, 1\mathrm{suc}, \mathrm{head}, \mathrm{tail}, \mathrm{reverse}; =\varepsilon, =0, =1\rangle$, where ε is the empty string, $0\mathrm{suc}(x) = x0$, $1\mathrm{suc}(x) = x1$, $\mathrm{head}(x) = \varepsilon$ if $x = \varepsilon$, the first symbol of x otherwise, and $\mathrm{tail}(x) = \varepsilon$ if $x = \varepsilon$, all except the first symbol of x otherwise. All partial recursive functions are computable by flowcharts over $\mathscr{B}$, and, in fact, such flowcharts can be designed to simulate multi-tape Turing machines with at most a multiplicative constant increase in running time. (Two variables can be used for each tape.) Moreover, flowcharts over $\mathscr{B}$ can be simulated by multi-tape Turing machines, one tape per variable. Because of the sequential nature of Turing machines and the allowance of assignments in flowcharts, the running time may need to be (roughly) squared. For further discussion of the relationships between computability using $\mathscr{B}$ and using machine models, the reader is referred to [9, 11, 12].

A group $\mathscr{G} = \langle \mathrm{Dom}_{\mathscr{G}}; \{g_i\}_{i \in I} \cup \{\circ, ^{-1}\}; = \rangle$ (where $\{g_i\}_{i \in I}$ is a set of generators for $\mathscr{G}$) is *computable* if $\mathscr{G} \leqslant_\tau^{\mathrm{flow}} \mathscr{B}$ for some τ. A f.g. group $\mathscr{G} = \langle \mathrm{Dom}_{\mathscr{G}}; g_1, \ldots, g_k, \circ, ^{-1}; = \rangle$ (where $\{g_1, \ldots, g_k\}$ is a set of generators for $\mathscr{G}$) is *of complexity* t, $N \to N$, if $\mathscr{G} \leqslant_\tau^{\mathrm{flow}} \mathscr{B}$ with complexity t for some τ.

Results in this section deal primarily with finitely generated groups. For

computability, all of the definitions can be seen to be equivalent, and are equivalent to solvability of the word problem.

THEOREM 4.1. *A f.g. group is computable* iff *its word problem (with respect to any finite set of generators) is solvable.*

Proof. Assume $\mathcal{G}$ is computable and $\{g_1,..., g_k\}$ is a set of generators. Then $\mathcal{G} = \langle \mathrm{Dom}_{\mathcal{G}}; g_1,..., g_k, \circ, {}^{-1}; = \rangle \leqslant^{\mathrm{flow}}_{\tau,\mathcal{E},\mathcal{P}} \mathcal{B}$ for some $\tau, \mathcal{E}, \mathcal{P}$. Given two words over $g_1,..., g_k, \circ, {}^{-1}$, of the form $(g_{i_1}^{m_1})(g_{i_2}^{m_2}) \cdots (g_{i_l}^{m_l})$, where $1 \leqslant i_1,..., i_l \leqslant k$, $m_1,..., m_l \in \mathcal{Z} - \{0\}$, their equivalence is determined as follows. Their evaluation is simulated by applying the given τ-simulators of $g_1,..., g_k, \circ$, and ${}^{-1}$, and then the τ-simulator of $=$ is applied.

Conversely, assume the word problem with respect to $\{g_1,..., g_k\}$ is solvable, and let $\mathcal{G} = \langle \mathrm{Dom}_{\mathcal{G}}; g_1,..., g_k, \circ, {}^{-1}; = \rangle$. Elements of $V\,\mathrm{Exp}_0(\mathrm{Fun}_{\mathcal{G}})$ are coded into $\{0, 1\}^*$ by an obvious binary coding α. Define τ so that $\tau(\alpha(a)) = \mathrm{val}(a)$ for all $a \in V\,\mathrm{Exp}_0(\mathrm{Fun}_{\mathcal{G}})$. We claim that $\mathcal{G} \leqslant^{\mathrm{flow}}_{\tau} \mathcal{B}$. This is because flowcharts for τ-simulators of $\circ$ and ${}^{-1}$ are trivial to construct, and a τ-simulator of $=$ can be constructed using the solvability of the word problem. ∎

THEOREM 4.2. *Let $\mathcal{G} = \langle \mathrm{Dom}_{\mathcal{G}}; g_1,..., g_k, \circ, {}^{-1}; = \rangle$ be a group, $g_1,...,g_k$ a set of generators. Then $\mathcal{G}$ is computable* iff *$\mathcal{G}$ is Rabin computable* iff *$\mathcal{G}$ is Mal'cev computable.*

Proof. The equivalence of computability with Mal'cev computability is obvious. The equivalence of computability with Rabin computability follows from Theorem 4.1 together with a similar equivalence in [16] for Rabin computability. ∎

Some added insight into the definitions can be obtained by a direct proof of the harder direction for Theorem 4.2; i.e. of the fact that computability implies Rabin computability. Assume $\mathcal{G} \leqslant_{\tau,\mathcal{E},\mathcal{P}} \mathcal{B}$. Let α be as in the proof of Theorem 4.1. Then a partial recursive function β: $\{0, 1\}^* \to \{0, 1\}^*$ is defined, with $\rho_{\mathcal{G}}(a) = \beta(\alpha(a))$ for all $a \in \mathrm{Dom}_{\mathcal{F}_{ree}(\mathcal{G})}$. (To compute $\beta(y)$, determine the expression a represented by y (if any), and then evaluate a, following the given τ-simulators of $g_1,..., g_k, \circ, {}^{-1}$.)

Fix some effective enumeration of the elements of $\mathrm{Dom}_{\mathcal{F}_{ree}(\mathcal{G})}$. Define $\kappa(x)$, for $x \in \mathrm{Dom}_{\mathcal{G}}$, as $\alpha(a)$, where a is the first element in the enumeration for which $\mathrm{val}(a) = x$. Then κ has the necessary properties for Rabin computability. For instance, we show that $\kappa(\mathrm{Dom}_{\mathcal{G}})$ is recursive. Given $x \in \{0, 1\}^*$, check first that $x \in \alpha(\mathrm{Dom}_{\mathcal{F}_{ree}(\mathcal{G})})$. If not, then $x \notin \kappa(\mathrm{Dom}_{\mathcal{G}})$. If so, let $\alpha(a) = x$. Enumerate all elements of $\mathrm{Dom}_{\mathcal{F}_{ree}(\mathcal{G})}$ up to but not including a. For each expression b in turn, apply the given τ-simulator of $=$ to $\beta(\alpha(b))$ and $\beta(\alpha(a))$ to determine whether $\tau(\beta(\alpha(b))) = \tau(\beta(\alpha(a)))$. This is equivalent to determining if $\mathrm{val}(b) = \mathrm{val}(a)$. If a "yes" answer is obtained, then $x \notin \kappa(\mathrm{Dom}_{\mathcal{G}})$, but if not, $x \in \kappa(\mathrm{Dom}_{\mathcal{G}})$. Similar arguments show the other properties. ∎

Next, consider complexity definitions. Our definition is equivalent to the Mal'cev definition at the primitive recursive level. To state this equivalence, we first show that

the complexity of a f.g. group in our definition is independent of the choice of finite sets of generators.

THEOREM 4.3. *Let $\mathscr{G}$ be a group, $\{g_1,..., g_k\}$ and $\{g'_1,..., g'_{k'}\}$ two sets of generators. If $\langle \mathrm{Dom}_{\mathscr{G}}; g_1,..., g_k, \circ, {}^{-1}; = \rangle$ is of complexity t where is nondecreasing and unbounded, then $\langle \mathrm{Dom}_{\mathscr{G}}; g'_1,..., g'_{k'}, \circ, {}^{-1}; = \rangle$ is of complexity t', where $t'(n) = t(n + c)$, c some constant.*

Proof. Let $\mathscr{A} = \langle \mathrm{Dom}_{\mathscr{G}}; g_1,..., g_k, g'_1,..., g'_{k'}, \circ, {}^{-1}; = \rangle$, $\mathscr{A}' = \langle \mathrm{Dom}_{\mathscr{G}}; g_1,..., g_k, \circ, {}^{-1}; = \rangle$ and $\mathscr{A}'' = \mathscr{B}$. By Theorem 3.8, $\mathscr{A} \leqslant_{\tau}^{\mathrm{flow}} \mathscr{A}''$ with complexity t', where $t'(n) = t(n + c)$ for some c. Thus, $\langle \mathrm{Dom}_{\mathscr{G}}; g'_1,..., g'_{k'}, \circ, {}^{-1}; = \rangle \leqslant_{\tau} \mathscr{A}''$ with complexity t'. ∎

Now the desired equivalence is shown.

THEOREM 4.4. *A f.g. group $\mathscr{G}$ is of primitive recursive complexity (with respect to any finite set of generators) iff $\mathscr{G}$ is Mal'cev primitive recursive.*

Proof. Roughly, if $\mathscr{G}$ is Mal'cev primitive recursive, then the closure of the primitive recursive functions under unlimited iteration can 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 detail, assume $\mathscr{G}$ is Mal'cev primitive recursive. Let $\{g_1,..., g_k\}$ be any finite set of generators, and write $\mathscr{G} = \langle \mathrm{Dom}_{\mathscr{G}}, g_1,..., g_k, \circ, {}^{-1}; = \rangle$. Then $\mathscr{G} \leqslant_{\tau}^{\mathrm{flow}} \mathscr{B}$ for some τ, with the simulators of $\circ$, ${}^{-1}$, and $=$ all primitive recursive, hence computable in primitive recursive time on Turing machines. Thus, they are also computable in primitive recursive time by flowcharts over $\mathscr{B}$. (However, the bound uses primitive recursiveness *in the inputs to the flowcharts*; i.e. using a parameter based in $\mathscr{B}$.) Choose $\mathscr{P}$, $\mathscr{E}$ so that $\mathscr{P}(\circ)$, $\mathscr{P}({}^{-1})$, and $\mathscr{P}(=)$ are flowcharts over $\mathscr{B}$ with primitive recursive running time, and so that $\mathscr{G} \leqslant_{\tau,\mathscr{E},\mathscr{P}}^{\mathrm{flow}} \mathscr{B}$.

The closure of the primitive recursive functions under iteration implies that there exists primitive recursive p with $p(\mathrm{size}_{\mathscr{F}_{rec}(\mathscr{G})}(A)) \geqslant \mathrm{size}_{\mathscr{F}_{rec}(\mathscr{B})}(\sigma_{\mathscr{G}}(A)) \geqslant \mathrm{size}_{\mathscr{B}}(\rho_{\mathscr{G}}(A))$ for all $A \subseteq \mathrm{Dom}_{\mathscr{F}_{rec}(\mathscr{G})}$. The implication then follows.

Conversely, assume $\mathscr{G} = \langle \mathrm{Dom}_{\mathscr{G}}; g_1,..., g_k, \circ, {}^{-1}; = \rangle \leqslant_{\tau,\mathscr{E}}^{\mathrm{flow}} \mathscr{B}$ with primitive recursive complexity. A possible difficulty arises if $\mathrm{size}_{\mathscr{B}}(\rho_{\mathscr{G}}(a))$ is very much smaller than $\mathrm{size}_{\mathscr{F}_{rec}(\mathscr{G})}(a)$ for some values of a. τ is therefore modified to insure sufficiently long coding strings. Define α, β as in Theorems 4.1 and 4.2; that is, $\alpha: V\,\mathrm{Exp}_0(\mathrm{Fun}_{\mathscr{G}}) \to \{0, 1\}^*$ is a direct binary encoding of expressions and $\rho_{\mathscr{G}}(a) = \beta(\alpha(a))$ for all $a \in \mathrm{Dom}_{\mathscr{F}_{rec}(\mathscr{G})}$.

Define $\tau': \{0, 1\}^* \to \mathrm{Dom}_{\mathscr{G}}$ by $\tau'(\alpha(a)) = \mathrm{val}(a)$. $\circ$ and ${}^{-1}$ have trivial primitive recursive τ'-simulators, while simulation of $=$ on $\alpha(a)$ and $\alpha(b)$ involves computing $\beta(\alpha(a))$ and $\beta(\alpha(b))$ and using the given τ-simulator of $=$. The time required is primitive recursive in $\mathrm{size}_{\mathscr{F}_{rec}(\mathscr{G})}(\{a, b\})$ and therefore primitive recursive in $\alpha(a)$ and $\alpha(b)$. ∎

Similar arguments to those used so far can be used to show a similar equivalence with CG-primitive recursiveness [19].

The apparent need for iteration closure in Theorem 4.3 suggests that it is unlikely the Mal'cev style definition could extend to any lower levels of complexity with sharp results. Mal'cev did not attempt such a definition. The more straightforward Cannonito and Gatterdam attempt leads to an iteration difficulty. For instance, a relationship is shown in [2–4] between the complexity of a group and that of its word problem. Because of the flavor of their definitions, a group of CG-complexity in $\mathscr{E}_k$ is only shown to have a word problem of complexity in $\mathscr{E}_{k+1}$. (Classes refer to the Grzegorczyk hierarchy.) This gap arises because a word of length n might be represented in $\{0, 1\}^*$ by a string of length $f^n(0)$, $f \in \mathscr{E}_k$. This difficulty is discussed at length in [3]. (The reader is cautioned to refer to [3] for corrections to several of the results in [2] which are relevant to the present paper.) The more complicated "$\mathscr{E}_a$-standard" definitions are used to circumvent this difficulty. Our definition avoids the iteration difficulty because of its choice of parameter; thus, we obtain several simple, fairly sharp complexity results. For example, our definition of group complexity is reasonably closely related to the complexity of the word problem, as might be expected from the simple proof of Theorem 4.1.

Let $e_{\mathscr{G}}$ denote the identity for group $\mathscr{G}$.

THEOREM 4.5. *Let $\mathscr{G} = \langle \mathrm{Dom}_{\mathscr{G}}; g_1, \ldots, g_k, \circ, {}^{-1}; = \rangle$ be a group, $\{g_1, \ldots, g_k\}$ a set of generators, $t: N \to N$ total, nondecreasing and with $t(n) \geqslant n$ for all n.*

(a) *If $\mathscr{G}$ is of complexity t, then the word problem for $\mathscr{G}$ with respect to $g_1, \ldots, g_k$ is solvable with (multi-tape Turing machine) complexity t', where $t'(n) = cn^2t^2(cn)$ for some constant c.*

(b) *If the word problem for $\mathscr{G}$ with respect to $\{g_1, \ldots, g_k\}$ has complexity t (of the combined lengths of the words, on a multi-tape Turing machine), then $\mathscr{G}$ is of complexity t', where $t'(n) = t(c.2^n)$ for some constant c.*

Proof. (a) This follows roughly the proof of Theorem 4.1.

It suffices to show how to check for equivalence with $e_{\mathscr{G}}$. A string of the form $g_{i_1}^{m_1} \cdots g_{i_l}^{m_l}$ is coded naturally as a binary string x, with the m_i written in binary. Assume the length of string x is n. A relation flowchart is defined with $L_P(x) \leqslant cnt(cn)$ for some constant c, where P outputs the answer to whether x evaluates to the group identity. A Turing machine is then obtained from P.

Let $\mathscr{G} \leqslant^{\mathrm{flow}}_{\tau, \mathscr{E}, \mathscr{F}} \mathscr{B}$ with complexity t. For binary string x as above, choose $a \in \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{G})}$ representing a natural complete parenthesization of x so that $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{G})}(a) \leqslant cn$ for some constant c. P follows the construction of y to obtain $\rho_{\mathscr{B}}(y)$. There are at most (approximately) cn steps simulated, each involving application of τ-simulators to $\rho_{\mathscr{B}}$-images of subsets of $\mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{G})}$ of $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{G})} \leqslant cn$. Thus, the complexity of this part of P's operation is at most $cnt(cn)$. Then P applies the τ-simulator of $=$ to $\rho_{\mathscr{B}}(a)$ and $\rho_{\mathscr{B}}(e_{\mathscr{G}})$. This involves application of a τ-simulator to the $\rho_{\mathscr{B}}$-image of a set of elements of $\mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{G})}$ of $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{G})} \leqslant cn + c$, for some larger constant c. Thus, $L_P(x) \leqslant cn$ (for decoding x into its operations) $+ cnt(cn) + c$ (for generating $\rho_{\mathscr{B}}(e_{\mathscr{G}})$) $+ t(cn + c)$ for some larger c. Assuming $n \geqslant 1$, this sum is at most $cnt(cn)$ for some larger c, as needed.

A Turing machine is designed to simulate P, causing at most squaring of the runtime.

(b) Let elements of $\mathrm{Dom}_{\mathscr{G}}$ be represented in $\{0, 1\}^*$ by direct codings of free-group-reduced strings over $\{g_1, \ldots, g_k\}$, coded into binary as in (a). The simulators of $\circ$ and $^{-1}$ are the straightforward ones; let $\mathscr{E}$ yield the corresponding expression assignments. Any $a \in \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{G})}$ with $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{G})}(a) \leqslant n$ has $\rho_{\mathscr{E}}(a)$ of length $\leqslant c \cdot 2^n$ for some constant c. (See Example 4.1 below.) An upper bound on the complexity of the simulators of $\circ$ and $^{-1}$ is proportional to this length. The simulation of $=$ is essentially the solution of the word problem, on words each of length $c \cdot 2^n$; thus, time $t(2c \cdot 2^n)$ suffices, which is of the needed form for larger c. ∎

While an exponential difference between the complexities is not negligible, it appears to be unavoidable because of the incompatability of the parameters involved. Namely, an expression $a \in \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{G})}$ with $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{G})}(a) = n$, 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 legnth n is that it has a generating expression in $\mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{G})}$ of $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{G})}$ roughly n (not necessarily $\log n$).

EXAMPLE 4.1. Let $\mathscr{G} = \langle \mathrm{Dom}_{\mathscr{G}}; g_1, g_2, \circ, ^{-1}; = \rangle$, and let $\{a_n\}_{n=1}^{\infty} \subseteq \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{G})}$ be defined as follows: $a_1 = g_1 \circ g_2$, and $a_{n+1} = a_n \circ a_n$. Then $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{G})}(a_n) = n + 2$. However, the free-group reduced representation of a_n is $g_1 g_2 g_1 g_2 \cdots g_1 g_2$, a string with 2^n occurrences of generator symbols.

Theorem 4.5 and claim GH5 of [3] show that our definitions and the CG "$\mathscr{E}_\alpha$-standard" definitions agree for each class in the Grzegorczyk hierarchy from $\mathscr{E}_3$ upward.

In the same vein as Theorem 4.5, a simple classification result is obtained.

THEOREM 4.6. *The free group over a finite set of generators is of complexity* $\lambda n[c \cdot 2^n]$ *for some constant* c.

Proof. Let $\mathscr{G} = \langle \mathrm{Dom}_{\mathscr{G}}; g_1, \ldots, g_k, \circ, ^{-1}; = \rangle$ and code elements of $\mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{G})}$ as string representations of their free-group reduced forms. The length of the representation stays bounded as above, and that length dominates the complexity. ∎

Question. Can this bound be improved?

The fact that simple algebraic constructions preserve complexity provides further evidence for the naturalness of our definitions. In the remainder of the paper, it is shown 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 (up to a double exponential). By contrast, the first and last of these four constructions cause rises of a level in the Grzegorczyk hierarchy when the CG definition is used to measure complexity.

THEOREM 4.7. *Let* $\mathscr{G} = \langle \mathrm{Dom}_{\mathscr{G}}; g_1, \ldots, g_k, \circ, ^{-1}; = \rangle$ *be a group,* $\{g_1, \ldots, g_k\}$ *a set*

of generators. Assume $\mathscr{G}$ is of complexity t, where t is nondecreasing and unbounded. Let $\mathscr{H} = \langle \mathrm{Dom}_{\mathscr{H}}; h_1,..., h_l, \circ, {}^{-1}; = \rangle$ be subgroup of $\mathscr{G}$ with generating set $\{h_1,..., h_l\}$. Then $\mathscr{H}$ is of complexity t'; where $t'(n) = t(n + c)$ for some constant c.

Proof. Let $\mathscr{K} = \langle \mathrm{Dom}_{\mathscr{G}}; g_1,..., g_k, h_1,..., h_l, \circ, {}^{-1}; = \rangle$. By Theorem 3.8, $\mathscr{K} \leqslant_{\tau}^{\mathrm{flow}} \mathscr{B}$ with complexity t', where $t' = t(n + c)$ for some c. Thus $\mathscr{H} \leqslant_{\tau}^{\mathrm{flow}} \mathscr{B}$ with complexity t'. ∎

In the following, $\equiv_{\mathscr{H}}$ denotes equivalence modulo $\mathscr{H}$, for normal subgroup $\mathscr{H}$, $[x]$ denotes the equivalence class of x, and $\mathscr{G}/\mathscr{H}$ denotes the quotient group.

THEOREM 4.8. *Let $\mathscr{G}$ be a group, $\{g_1,..., g_k\}$ a set of generators, $\mathscr{H}$ a normal subgroup, $\langle \mathrm{Dom}_{\mathscr{G}}\, g_1,..., g_k, \circ, {}^{-1}; =, \equiv_{\mathscr{H}} \rangle \leqslant_{\tau}^{\mathrm{flow}} \mathscr{B}$ with complexity t. Then the quotient group $\langle \mathrm{Dom}_{\mathscr{G}/\mathscr{H}}; [g_1],..., [g_k], \circ, {}^{-1}; = \rangle$ is of complexity t.*

Proof. Define $\tau'(x) = [\tau(x)]$. The given τ-simulators for $g_1,..., g_k, \circ$, and ${}^{-1}$ are also τ'-simulators for $[g_1],..., [g_k]$, $\circ$ and ${}^{-1}$, respectively. The given τ-simulator of $\equiv_{\mathscr{H}}$ is the needed τ'-simulator of $=$. ∎

THEOREM 4.9. *Assume $\mathscr{G} = \langle \mathrm{Dom}_{\mathscr{G}}; g_1,..., g_k, \circ, {}^{-1}; = \rangle$ and $\mathscr{H} = \langle \mathrm{Dom}_{\mathscr{H}}; h_1,..., h_l, \circ, {}^{-1}; = \rangle$ (with $\{g_1,..., g_k\}$ and $\{h_1,..., h_l\}$ respective sets of generators) are each of complexity t, where t is nondecreasing and not identically equal to 0. Then the direct product of $\mathscr{G}$ and $\mathscr{H}$, $\mathscr{G} \times \mathscr{H} = \langle \mathrm{Dom}_{\mathscr{G}\times\mathscr{H}}; (g_i, e_{\mathscr{H}}), \quad 1 \leqslant i \leqslant k, \quad (e_{\mathscr{G}}, h_i), 1 \leqslant i \leqslant l, \circ, {}^{-1}; = \rangle$ is of complexity t', where $t'(n) = cnt(n)$ for some constant c.*

Proof. Let $\mathscr{G} \leqslant_{\tau,\mathscr{F},\mathscr{P}}^{\mathrm{flow}} \mathscr{B}$ with complexity t, and $\mathscr{H} \leqslant_{\tau',\mathscr{F}',\mathscr{P}'}^{\mathrm{flow}} \mathscr{B}$ with complexity t. Then τ'', $\mathscr{E}''$, $\mathscr{P}''$ will be defined so that $\mathscr{G} \times \mathscr{H} \leqslant_{\tau'',\mathscr{E}'',\mathscr{P}''}^{\mathrm{flow}} \mathscr{B}$ with complexity t'. Let $a \in \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{G}\times\mathscr{H})}$, and let b and b' be the component expressions of a for $\mathscr{G}$ and $\mathscr{H}$, respectively, with all explicit references to $e_{\mathscr{G}}$ and $e_{\mathscr{H}}$ eliminated. Then $\rho_{\mathscr{G}''}(a) = \langle x, x' \rangle$, where $x = \rho_{\mathscr{G}}(b)$ if $b \in \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{G})}$, a special code if $b = e_{\mathscr{G}}$, and similarly for x', b' and $\mathscr{E}'$. τ'' is defined from $\rho_{\mathscr{G}''}$ so that the diagram in Theorem 2.2 commutes. $\circ$, ${}^{-1}$, and $=$ are τ''-simulated using the given τ- and τ'-simulators on the components; definitions for $\mathscr{E}''$ and $\mathscr{P}''$ from $\mathscr{E}$, $\mathscr{E}'$, $\mathscr{P}$ and $\mathscr{P}'$ are straightforward.

Assume a, $a_1 \in \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{G}\times\mathscr{H})}$, with b, b', b_1, and b_1' their component expressions with $e_{\mathscr{G}}$ and $e_{\mathscr{H}}$ eliminated. Then the number of steps needed to compute the τ''-simulator of $\circ$ on $\rho_{\mathscr{G}''}(a)$ and $\rho_{\mathscr{G}''}(a_1)$ is bounded by the complexity of coding and decoding the components, plus a constant (in case b or $b_1 = e_{\mathscr{G}}$ or b' or $b_1' = e_{\mathscr{H}}$) plus the complexity of τ-simulating $\circ$ on $\rho_{\mathscr{G}}(b)$ and $\rho_{\mathscr{G}}(b')$ and $\rho_{\mathscr{G}}(b_1')$ (if b', $b_1' \in \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{H})}$).

The complexity of coding and decoding is proportional to the length of the representations, which is bounded by $c(1 + |\rho_{\mathscr{G}}(b)| + |\rho_{\mathscr{G}}(b_1)| + |\rho_{\mathscr{G}}(b')| + |\rho_{\mathscr{G}}(b_1')| + |\rho_{\mathscr{G}}(b \circ b_1)| + |\rho_{\mathscr{G}}(b' \circ b_1')|)$, where c is some constant. (Here, if any of the terms is undefined because the argument is $e_{\mathscr{G}}$ or $e_{\mathscr{H}}$, then that term can simply be eliminated from the expression.) These lengths can be bounded as follows. Consider $|\rho_{\mathscr{G}}(b)|$, for example; $|\rho_{\mathscr{G}}(b)| = \mathrm{size}_{\mathscr{B}}(\rho_{\mathscr{G}}(b)) \leqslant \mathrm{size}_{\mathscr{F}_{free}(\mathscr{B})}(\sigma_{\mathscr{G}}(b))$, $\leqslant \sum_{i=0}^{\mathrm{size}_{\mathscr{F}_{free}(\mathscr{G})}(b)-1} t(i)$ by Theorem 3.4, $\leqslant \mathrm{size}_{\mathscr{F}_{free}(\mathscr{G})}(b) \cdot t(\mathrm{size}_{\mathscr{F}_{free}(\mathscr{G})}(b)) \leqslant \mathrm{size}_{\mathscr{F}_{free}(\mathscr{G}\times\mathscr{H})}(a) \cdot t(\mathrm{size}_{\mathscr{F}_{free}(\mathscr{G}\times\mathscr{H})}(a)) \leqslant$

$\text{size}_{\mathcal{F}ree(\mathcal{G}\times\mathcal{H})}(\{a, a_1\}) \cdot t(\text{size}_{\mathcal{F}ree(\mathcal{G}\times\mathcal{H})}(\{a, a_1\}))$, as needed. The other lengths are similarly bounded. Thus, the total complexity of coding and decoding can be bounded by $c(1 + \text{size}_{\mathcal{F}ree(\mathcal{G}\times\mathcal{H})}(\{a, a_1\})) \cdot t(\text{size}_{\mathcal{F}ree(\mathcal{G}\times\mathcal{H})}(\{a, a_1\})))$ for come c. The complexity of τ-simulating $\circ$ on $\rho_{\mathcal{G}}(b)$ and $\rho_{\mathcal{G}}(b_1)$, is easily seen to be bounded by $t(\text{size}_{\mathcal{F}ree(\mathcal{G}\times\mathcal{H})}(\{a, a_1\}))$, and correspondingly for $\rho_{\mathcal{G}'}(b')$ and $\rho_{\mathcal{G}'}(b'_1)$. Thus, the total complexity for simulating $\circ$ can be bounded as needed.

Similar arguments for $^{-1}$ and $=$ yield the required bound. ∎

As an example of a more complex construction, the final theorem bounds the complexity of the amalgamated free product of two groups in terms of the complexity of the component groups and that of certain basic mappings and relations involving the component groups. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be disjoint groups, $\mathcal{H}$ a subgroup of $\mathcal{G}_1$, and α a monomorphism of $\mathcal{H}$ into $\mathcal{G}_2$. The *amalgamated free product*, $\mathcal{G}_1 * \mathcal{G}_{2(\alpha)}$, is the quotient group of the free product of $\mathcal{G}_1$ and $\mathcal{G}_2$ which results from identification of $\mathcal{H}$ with its image under α. Let canon_1: $\text{Dom}_{\mathcal{G}_1} \to \text{Dom}_{\mathcal{G}_1}$ and canon_2: $\text{Dom}_{\mathcal{G}_2} \to \text{Dom}_{\mathcal{G}_2}$ be any two maps producing canonical representations of right cosets of $\mathcal{H}$ and $\alpha(\mathcal{H})$, respectively. Then Theorem 4.4 of [15] says that each element of $\text{Dom}_{\mathcal{G}_1 * \mathcal{G}_{2(\alpha)}}$ has a unique representation in the form $hx_1 \cdots x_k$, where $h \in \text{Dom}_{\mathcal{H}}$ and each x_i is a canonical representative of a right coset in either $\mathcal{G}_1$ or $\mathcal{G}_2$, (but is not the representative of $\text{Dom}_{\mathcal{H}}$ or of $\alpha(\text{Dom}_{\mathcal{H}})$). Furthermore, two consecutive x_i are not both in $\text{Dom}_{\mathcal{G}_1}$ or both in $\text{Dom}_{\mathcal{G}_2}$. This representation is called the *canonical form* for the given element of $\text{Dom}_{\mathcal{G}_1 * \mathcal{G}_{2(\alpha)}}$ (with respect to canon_1 and canon_2).

In the following, εA denotes the membership relation for set A, and $\circ_{\mathcal{G}}$ and $^{-1}{}_{\mathcal{G}}$ denote the operations of group $\mathcal{G}$. If x is a free-group-reduced word over the generators $g_1, \ldots, g_k$, then $\text{gen}(x)$ denotes the sum of the absolute values of the exponents in x.

THEOREM 4.10. *Let* $\mathcal{K} = \langle \text{Dom}_{\mathcal{G}_1} \cup \text{Dom}_{\mathcal{G}_2};\ h_1, \ldots, h_k,\ g_1, \ldots, g_l,\ g'_1, \ldots, g'_m,\ \circ_{\mathcal{G}_1},\ \circ_{\mathcal{G}_2},\ ^{-1}{}_{\mathcal{G}_1},\ ^{-1}{}_{\mathcal{G}_2},\ \alpha,\ \alpha^{-1};\ =,\ \varepsilon\text{Dom}_{\mathcal{G}_1},\ \varepsilon\text{Dom}_{\mathcal{H}} \rangle$, where $\{h_1, \ldots, h_k\}$, $\{g_1, \ldots, g_l\}$, and $\{g'_1, \ldots, g'_m\}$ are sets of generators for $\mathcal{H}$, $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. Assume $\mathcal{K} \leqslant_{\tau}^{\text{flow}} \mathcal{B}$ with complexity t, where t is nondecreasing. Let $\mathcal{G}_1 * \mathcal{G}_{2(\alpha)} = \langle \text{Dom}_{\mathcal{G}_1 * \mathcal{G}_{2(\alpha)}};\ h_1, \ldots, h_k, g_1, \ldots, g_l, g'_1, \ldots, g'_m, \circ, ^{-1}; = \rangle$.

Then for some c, $\mathcal{G}_1 * \mathcal{G}_{2(\alpha)}$ is of complexity t', where $t'(n) = c^{c^{cn}} t(c^{cn})$.

Proof. Fix $\mathcal{E}$, $\mathcal{P}$ so that $\mathcal{K} \leqslant_{\tau,\mathcal{E},\mathcal{P}}^{\text{flow}} \mathcal{B}$ with complexity t. Then τ', $\mathcal{E}'$, $\mathcal{P}'$ must be defined so that $\mathcal{G}_1 * \mathcal{G}_{2(\alpha)} \leqslant_{\tau',\mathcal{E}',\mathcal{P}'}^{\text{flow}} \mathcal{B}$ with complexity t'. $\mathcal{E}'$ is defined so that for a in $\text{Dom}_{\mathcal{F}ree(\mathcal{G}_1 * \mathcal{G}_{2(\alpha)})}$, $\rho_{\mathcal{G}'}(a)$ is (a binary representation of) the free-group-reduced version of a. (In this proof, the binary representation of such a product and the product string itself will be referred to interchangeably. We rely on the reader to make the distinction when necessary.) $\circ$ and $^{-1}$ have the straightforward τ'-simulators. The τ'-simulation of $=$ uses Theorem 4.4 of [15], as follows.

Particular mappings, canon_1 and canon_2 are defined. Consider an enumeration of

elements of $\mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{H})}$ in order of $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{H})}$. Given $x \in \mathrm{Dom}_{\mathscr{G}_1}$ (resp. $\mathrm{Dom}_{\mathscr{G}_2}$), $\mathrm{canon}_1(x)$ (resp. $\mathrm{canon}_2(x)$) is defined to be $\mathrm{val}(a)$, where a is the first element of the enumeration having $\mathrm{val}(a)$ and x in the same right coset of $\mathrm{Dom}_{\mathscr{H}}$ (resp. $\alpha(\mathrm{Dom}_{\mathscr{H}})$).

A *coding* for a value $x \in \mathrm{Dom}_{\mathscr{G}_1 * \mathscr{G}_{2(a)}}$ is a binary string of the form $\langle h, x_1, \ldots, x_j \rangle$, where the string $\tau(h)\,\tau(x_1) \cdots \tau(x_j)$ is the canonical form of x with respect to the (fixed) mappings canon_1 and canon_2.

The τ'-simulator of $=$ uses the given τ-simulators to translate the two expressions into codings of their values, working from right to left in each expression as in the proof in [15]. Then it uses the τ-simulator of $=$ on the components of the two representations.

In order to accomplish the needed translation, τ-simulators of canon_1 and canon_2 will be helpful. We describe canon_1. Given $x \in \{0, 1\}^*$, apply the given τ-simulator of $\varepsilon\mathrm{Dom}_{\mathscr{G}_1}$ to determine if $\tau(x) \in \mathrm{Dom}_{\mathscr{G}_1}$. (It is possible that $\tau(x)$ is undefined, in which case the τ-simulator might diverge or give a meaningless answer.) If so, then begin enumerating the elements of $\mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{H})}$ in order of $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{H})}$. For each $a \in \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{H})}$ in turn, compute $\rho_{\mathscr{G}}(a)$ by simulating the operations appearing in the expression a. Then apply the given τ-simulators of $\varepsilon\mathrm{Dom}_{\mathscr{G}_1}$, ${}^{-1}{}_{\mathscr{G}_1}$, and $\varepsilon\mathrm{Dom}_{\mathscr{H}}$ to determine if $\tau(x) \circ_{\mathscr{G}_1} (\tau(\rho_{\mathscr{G}}(a)))^{-1_{\mathscr{G}_1}} \in \mathrm{Dom}_{\mathscr{H}}$, i.e., if $\tau(x)$ and $\mathrm{val}(a)$ are in the same right coset of $\mathscr{H}$. When a positive answer is obtained for an expression a, $\rho_{\mathscr{G}}(a)$ is the needed value.

If $x = \rho_{\mathscr{G}}(a)$ in this construction, where $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{H})}(a) = n$, then the simulator of canon_1 can be computed by a flowchart over $\mathscr{B}$ with a number of steps bounded by $cn^{cn}t(cn)$ for some constant c. This is because cn^{cn} is an upper bound on the number of expressions that must be generated before a positive answer is obtained, for some constant c. At most n operations are simulated in computing $\rho_{\mathscr{G}}(a)$ for each a; each of these operations requires at most $t(n)$ steps for its simulation. Also, $ct(cn)$ bounds the time needed to apply the given τ-simulators, for some c. The total complexity can be bounded as stated, for some (larger) c. Furthermore, the value $a \in \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{H})}$ found by this procedure has $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{H})}(a) \leqslant n$.

Next, the translation is described. Let $a \in \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{G}_1 * \mathscr{G}_{2(a)})}$, $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{G}_1 * \mathscr{G}_{2(a)})}(a) = n$, $x = \rho_{\mathscr{G}'}(a)$. x must be translated into a coding of its value. Note $\mathrm{gen}(x) \leqslant 2^n$ (see Example 4.1). x is processed from right to left, one generator (or generator inverse) symbol at a time. After any suffix of x has been processed, that suffix will have been translated into a coding of its value.

The rightmost symbol w is processed as follows. If w is h_i or h_i^{-1}, $1 \leqslant i \leqslant k$, then the coding is $\rho_{\mathscr{G}}(w)$. If w is g_i or g_i^{-1}, $1 \leqslant i \leqslant l$, then compute $\rho_{\mathscr{G}}(w)$ and apply the given τ-simulator of $\varepsilon\mathrm{Dom}_{\mathscr{H}}$ to the result. If the answer "yes" is obtained, then $\rho_{\mathscr{G}}(w)$ is the coding. If "no" is obtained, apply the τ-simulator of canon_1 described above to $\rho_{\mathscr{G}}(w)$, obtaining y, where $\tau(y)$ is the canonical representative of w's right coset. Next, use the τ-simulators of $\circ_{\mathscr{G}_1}$ and ${}^{-1}{}_{\mathscr{G}_1}$ to obtain h, where $\mathrm{val}(w) = \tau(h) \circ_{\mathscr{G}_1} \tau(y)$. The coding is $\langle h, y \rangle$. A similar construction is used if w is g_i' or $(g_i')^{-1}$, $1 \leqslant i \leqslant m$, except that when a τ-representation is obtained for a needed element of $\alpha(\mathrm{Dom}_{\mathscr{H}})$, the τ-simulator of α^{-1} must be applied to yield a τ-representation for the corresponding element of $\mathrm{Dom}_{\mathscr{H}}$.

Since there are only finitely many generators, the time to process the rightmost symbol is bounded by a constant.

Now consider the processing of a non-rightmost symbol w of x (assuming all symbols to its right have already been processed). Let $\langle h, x_1, ..., x_j \rangle$ be the assumed coding of the value of the suffix, and consider several cases.

(1) *w is h_i or h_i^{-1}.* Then combine $\rho_{\mathscr{F}}(w)$ with h using the τ-simulator of $\circ_{\mathscr{G}_1}$, obtaining $y \in \{0, 1\}^*$. The new coding is $\langle y, x_1, ..., x_j \rangle$.

(2) *w is g_i or g_i^{-1} and either $j = 0$ or $\tau(x_1) \in \mathrm{Dom}_{\mathscr{G}_2}$.* Then combine $\rho_{\mathscr{F}}(w)$ with h using the τ-simulator of $\circ_{\mathscr{G}_1}$, obtaining $y \in \{0, 1\}^*$. Apply the τ-simulator of $\varepsilon \mathrm{Dom}_{\mathscr{F}}$ to y.

(2a) $\tau(y) \in \mathrm{Dom}_{\mathscr{F}}$. Then the new coding is $\langle y, x_1, ..., x_j \rangle$.

(2b) $\tau(y) \notin \mathrm{Dom}_{\mathscr{F}}$. Then apply the τ-simulators of canon_1, $\circ_{\mathscr{G}_1}$ and $^{-1}{}_{\mathscr{G}_1}$ as above to obtain z and h', where $\tau(z)$ is the canonical representative of $\tau(y)$'s right coset, and $\tau(y) = \tau(h') \circ_{\mathscr{G}_1} \tau(z)$. The coding is $\langle h', z, x_1, ..., x_j \rangle$.

(3) *w is g_i or g_i^{-1} and $\tau(x_1) \in \mathrm{Dom}_{\mathscr{G}_1}$.* Then combine $\rho_{\mathscr{F}}(w)$ with h and also with x_1 using the τ-simulator of $\circ_{\mathscr{G}_1}$, obtaining $y \in \{0, 1\}^*$. Proceed as in case (2) except that x_1 is omitted from the final coding.

(4) *w is g_i' or $(g_i')^{-1}$.* Proceed analogously to (2) and (3), using the τ-simulators of α and α^{-1} where needed.

We bound the complexity of the preceding construction. We claim that after any number n' of the ($\leqslant 2^n$) symbols of x have been processed, the coding so far obtained is of the form $\langle h, x_1, ..., x_j \rangle$, where $j \leqslant n'$ and where each $x_i = \rho_{\mathscr{F}}(a_i)$, $h = \rho_{\mathscr{F}}(b)$, and $\sum_{i=1}^{j} \mathrm{size}_{\mathscr{F}_{free}(\mathscr{X})}(a_i) + \mathrm{size}_{\mathscr{F}_{free}(\mathscr{X})}(b) \leqslant c^{n'}$ for some constant c. This fact can be shown by induction on n', using the size bound noted in the construction of the simulators for canon_1 and canon_2.

Thus, the entire construction involves applying τ-simulators to elements of the form $\rho_{\mathscr{F}}(a)$, $a \in \mathrm{Dom}_{\mathscr{F}_{free}(\mathscr{X})}$, with $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{X})}(a) \leqslant c^{2^n}$. There are a constant number of τ-simulators applied for each symbol in the string, of which one (at most) is of canon_1 or canon_2 and all the others are of operations in $\mathrm{Fun}_{\mathscr{F}}$. Thus, the complexity is bounded above by $2^n[ct(c^{2^n}) + c(c^{2^n})^{(c \cdot c^{2^n})} t(c \cdot c^{2^n})]$ for some (larger) c. (The term outside the brackets bounds the number of symbols in x, the first term within brackets bounds the simulation of the operations in $\mathrm{Fun}_{\mathscr{F}}$, and the second term within brackets bounds the simulations of canon_1 or canon_2.) This expression is bounded by $c^{c^{c^n}} t(c^{c^n})$ for some c.

In order to complete the proof, we must bound the complexity of τ'-simulating $=$ on two expressions $x = \rho_{\mathscr{F}'}(a)$ and $x' = \rho_{\mathscr{F}'}(a')$, where $\mathrm{size}_{\mathscr{F}_{free}(\mathscr{G}_1 * \mathscr{G}_{2(a)})}(\{a, a'\}) = n$. Both x and x' are translated as above, and then the τ-simulator of $=$ is used on the components of the two codings. The total complexity is bounded roughly by $2 \cdot c^{c^{c^n}} t(c^{c^n})$ (to obtain the two codings) $+ 2^n \cdot t(c^{2^n})$ (to check equality on all the components). The total complexity is bounded as needed, for some c. ∎

The hypothesis of Theorem 4.10 bounds the complexity of algebra $\mathscr{X}$, which

includes as operations not only the operations of both $\mathscr{G}_1$ and $\mathscr{G}_2$, but also the correspondence between their subgroups. Thus, for example, the parameter $\text{size}_{\mathscr{F}ree(\mathscr{H})}(a)$ could be small for a with $\text{val}(a) \in \text{Dom}_{\mathscr{H}}$ requiring a large number of $\mathscr{H}$-operations for its generation. Since the hypothesized bound is given in terms of $\text{size}_{\mathscr{F}ree(\mathscr{H})}$, the hypothesis is fairly restrictive.

The computational complexity of a group measured according to our definitions seems to be related to the complexity of the group measured in terms of algebraic decomposition. Thus, our definitions might be useful as classification tools for group theory. Similar complexity classification of other mathematical structures should also be of some interest and value.

ACKNOWLEDGMENTS

The author thanks Ann Yasuhara, Chee Yap and Michael Loui for their meticulous readings of the original manuscript and their very valuable suggestions for its revision. This paper is very much improved for their efforts.

REFERENCES

1. A. V. AHO, J. HOPCROFT, AND J. ULLMAN, "The Design and Analysis of Computer Algorithms, Addison–Wesley, Reading, Mass., 1976.
2. F. B. CANNONITO, Hierarchies of computable groups and the word problem, *J. Symbolic Logic* **31**, No. 3, (1966).
3. F. B. CANNONITO, The algebraic invariance of the word problem in groups, *in* "*Word Problems*,, (W. W. Boone, F. B. Cannonito, and R. C. Lyndon, Eds.) Studies in Logic and the Foundations of Mathematics, Vol. 71, North-Holland, Amsterdam, 1973.
4. F. B. CANNONITO, AND R. W. GATTERDAM, The computability of group instructions, *I*, *in* "Word Problems" (W. W. Boone, F. B. Cannonito, and R. C. Lyndon, Eds.), Studies in Logic and the Foundations of Mathematics, Vol 71, North-Holland, Amsterdam, 1973.
5. J. GUTTAG, E. HOROWITZ, AND D. MUSSER, "Abstract Data Types and Software Validation," Research Report 76-48, Information Sciences Institute, August 1976.
6. J. GOGUEN, J. THATCHER, AND E. WAGNER, "An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types," IBM Technical Report.
7. N. A. LYNCH, Accessibility of Values as a Determinant of Relative Complexity of Algebras, submitted for publication.
8. N. A. LYNCH, Straight-Line Program Length as a Parameter for Complexity Measures. "Proceedings 10th Annual Symposium on Theory of Computing, pp. 150–161, 1978.
9. N. A. LYNCH, AND E. K. BLUM, Efficient reducibility between programming systems; preliminary report, *in* "Proceedings, 9th Annual Symposium on Theory of Computing, 1977," pp. 228–238.
10. N. A. LYNCH, AND E. K. BLUM, A difference in expressive power between flowcharts and recursion schemes, *Math. Systems Theory* **12**, (1979), 205–211.
11. N. A. LYNCH, AND E. K. BLUM, Relative computability and complexity of algebras. *Math. Systems Theory*, in press.
12. N. A. LYNCH, AND E. K. BLUM, Relative complexity of operations on numeric and bit string algebras. *Math. Systems Theory* **13**, (1980), 187–207.
13. B. LISKOV, AND S. ZILLES, Specification techniques for data abstractions, *Software Eng.* **SE-1**, No. 1 (1975) 7–19.

14. A. MAL'CEV, The Metamathematics of algebraic systems. "Studies in Logic and the Foundations of Mathematics," Vol. 66, North-Holland, Amsterdam, 1971.
15. W. MAGNUS, A. KARRASS, AND D. SOLITAR, "Combinatorial Group Theory, Presentations of Groups in Terms of Generators and Relations," Vol.13, Interscience, New York, 1966.
16. M. O. RABIN, Computable algebra, General theory and theory of computable fields, *Trans.* **95** (1960), 341–360.
17. J. SIMON, On feasable numbers, Ninth Annual ACM Symposium on Theory of Computing, 1977.
18. R. TARJAN, Reference machines requires non-linear time to maintain disjoint sets, Ninth Annual ACM Symposium on Theory of Computing, 1977.
19. M. VALIEV, personal communication.