Proc. 18th Symp. Foundations of Comp. Sci. Oct. 1977, 254-266.

LOWER BOUNDS FOR NATURAL PROOF SYSTEMS

Dexter Kozen

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

ABSTRACT

Two decidable logical theories are presented, one complete for deterministic polynomial time, one complete for polynomial space. Both have natural proof systems. A lower space bound of n/log(n) is shown for the proof system for the PTIME complete theory and a lower length bound of $2^{cn/\log(n)}$ is shown for the proof system for the PSPACE complete theory.

1 INTRODUCTION

1.1 Naturalness vs. Efficiency in Proof Systems and Computations

Most good lower bound results we have are the product of defining a simple, natural model of computation or computational complexity, one that would mirror the complexity of a straightforward implementation. Number of comparisons for sorting [Kn] and number of arithmetic operations for polynomial evaluation [AHU] are good examples. These models are more restricted than the more general models of Turing machine time and space and combinational complexity of Boolean circuits, for which our lower bound results are still poor. This behavior suggests a general principle: the more restricted the model, the better the bounds. We may explain this by considering computations as proofs in some proof system. According to Cook and Reckhow's definition [CR] an arbitrary Turing machine computation may be considered a proof in such a system, whose statements are the configurations of the machine, whose axioms are the accepting configurations, and whose rules of inference are determined by the transition function. The language involved would contain symbols for tape contents, states of the finite control, and position of the tape head. Compare this language to a language for expressing the computation of a straight line program for evaluating a polynomial, which might contain variable and constant symbols and symbols for the operations of addition and multiplication. The Turing machine language is much more primitive and expressive, as it is easy to see how to do a step-by-step simulation of a straight line program with a Turing machine, but not the reverse.

An algorithm for polynomial evaluation given by a straight line program using + and * is more *natural* than one given by a Turing machine, in the sense that each step in the computation contributes a much larger amount of progress toward the final result, and it is easier to measure the extent of this progress intuitively. With Turing machines, the steps are smaller, and it is harder to see how the later steps in the computation depend on the earlier ones, whether a particular step is unnecessary so that it can be deleted, or whether the steps can be rearranged to make the computation more efficient. Thus naturalness in a language correlates with greater understanding, and therefore better bounds.

The same situation arises in the study of the complexity of proof systems for propositional logic. Tseitin [TS] established a $2^{\sqrt{n}}$ lower bound for resolution proofs. Galil [Ga] later simplified and extended Tseitin's methods and established exponential lower bounds for related proof systems. Cook and Reckhow [CR] discussed a wide variety of proof systems for propositional logic and related the question of their complexity to the classic P=NP problem.

All the proof systems studied were *natural*, again in the sense that some kind of intuitive meaning could be attributed to the axioms and rules of inference. Each step in a proof contributed something in the way of progress toward the final conclusion, and it was easy to understand the extent of this progress. But giving a variety of natural proof systems for propositional logic and then showing that they all exhibit exponential worst case behavior does not say P ≠ NP, since there may be some fiendishly clever, unnatural proof system, in the form of a Turing machine computation, which is fast. It does however suggest that if such a fast procedure exists, it will probably be fiendishly clever and unnatural.

Although the concept of *natural* is intuitive and defies formalization (recall our description of a natural proof system or computation as one for which intuitive meaning can be ascribed to the axioms and rules), in practice there appears to be a tradeoff between naturalness and efficiency, of both proof systems for propositional logic and computation models for sorting or evaluation of polynomials. This phenomenon is well known to computer programmers: often straightforward, easily programmed algorithms are less efficient than tricky ones; when programming in PASCAL instead of assembly language, efficiency is sacrificed in favor of readability.

1.2 Outline of Main Results

In this paper we present more evidence supporting these ideas.

Instead of looking at the P=NP question which Cook and Reckhow approached, we consider the two open questions P=PSPACE and LOGSPACE=P. We give two logical theories, one $\leq_{\log}$-complete for P, one $\leq_{\log}$-complete for PSPACE. Both have natural proof systems. We establish a non-logspace lower bound for the proof system for the P complete theory, and a non-polynomial time lower bound for the proof system for the PSPACE complete theory. The two proof systems are so natural and simple that it is hard to imagine any other proof system which would not encode proofs in the given system in one way or another, in the same way that it is hard to imagine sorting without comparisons.

The first theory studied is an equational theory. Given a language consisting of symbols for constants and finitary operators (no variables), a finite set

$$\Gamma = \{x_1=y_1, \ldots, x_n=y_n\}$$

of identities between terms, and two terms x and y, is it the case that x=y in all models of Γ? This question is equivalent to the word problem for finitely presented algebras, shown in [Ko] to be complete for P.

A special case of this problem is the circuit value problem of Ladner [La], also complete for P. In the circuit value problem, we are given a list L of assignments to $c_1,\ldots,c_n$ of the form

$c_i=0$
$c_i=1$
$c_i=c_j \wedge c_k$, $j,k<i$
$c_i=c_j \vee c_k$, $j,k<i$
$c_i=\neg c_j$, $j<i$

such that each c_i occurs on the left exactly once. The problem is to determine the final Boolean value of c_n.

A natural proof system for the circuit value problem is the following:

Axioms: all statements appearing in L
Rules of inference:

$$\frac{c_i=c_j \vee c_k,\ c_j=1}{c_i=1} \qquad \frac{c_i=c_j \vee c_k,\ c_k=1}{c_i=1}$$

$$\frac{c_i=\neg c_j,\ c_j=1}{c_i=0} \qquad \frac{c_i=c_j \wedge c_k,\ c_j=1,\ c_k=1}{c_i=1}$$

and their duals.

This constitutes a complete proof system for true statements of the form "$c_i=1$" or "$c_i=0$", and it is hard to imagine any proof system for this theory which does not encode proofs in the above system in some way, since such a system would have to determine the value of some c_i without knowing the values of c_j and c_k previously, where $c_i=c_j \wedge c_k$ appears in L, for example. If it could be shown that all proof systems for this theory must encode proofs in the above system, then the question LOGSPACE=P, a major open question of computer science, would be settled in the negative.

The lower bound for this proof system of space n/log(n) follows from Paul, Tarjan, and Celoni's lower bound for pebbling of directed acyclic graphs [PTC]. This result extends to an n/log(n) space lower bound for a variety of natural proof systems for the equational theory mentioned above. One of them has been used extensively in [Ko] for the word problem for finitely presented algebras.

The second theory studied concerns functions from a set to itself. Let A be a set with n elements, and let $\{f_1,\ldots,f_k\}$ be a set of maps $A \rightarrow A$. Let $h:A \rightarrow A$ also be given. How hard is is to determine whether h is generated by $f_1,\ldots,f_k$ under composition? This problem is shown to be complete for PSPACE. It is easy to see that it is in nondeterministic linear space, as follows. Suppose functions $A \rightarrow A$ are represented by bipartite directed graphs with 2n vertices and n edges. Using this representation, the composition $f \circ g$ of two functions f and g can be computed easily, just by rearranging edges. We can start with the representation of the identity on A, then apply the f_i's nondeterministically in some order, and accept if h ever appears.

A natural proof system for statements of the form GEN(g), meaning "g is generated", consists of

Axioms: GEN(f_i), $1 \leq i \leq k$
Rule of inference: $\dfrac{\text{GEN}(f),\ \text{GEN}(g)}{\text{GEN}(f \circ g)}$

For infinitely many n, a set of cardinality n and functions f_1, f_2, and h are exhibited such that h is generated by f_1 and f_2 under composition, but the shortest proof of GEN(h) in the above proof system is of length at least $2^{cn/\log(n)}$. Again, if it could be shown that any proof system for this theory encodes proofs in the above system, then P≠PSPACE.

2 PRELIMINARIES

Definition 2.1. A *proof system*, for our purposes, will consist of

i) a *language* $S \subseteq \Sigma^*$ where Σ is a set of *symbols*. Elements of *S* are called *statements*.
ii) a subset of *S* called the *axioms*.
iii) a set of *rules of inference*, or relations between tuples of statements and statements.

If

$$\phi_1,\ldots,\phi_k \ R\ \phi$$

and *R* is a rule of inference, we write

$$\frac{\phi_1,\ldots,\phi_k}{\phi}$$

and say ϕ *follows from* $\phi_1,\ldots,\phi_k$ (*via R*).

A *proof* of ϕ in this system consists of a sequence of statements $\phi_1,\ldots,\phi_n$ such that each ϕ_i either is an occurrence of an axiom or follows from some $\phi_{j_1},\ldots,\phi_{j_m}$, all $j_k < i$, and $\phi=\phi_n$. If ϕ_i is the result of applying a rule *R* to $\phi_{j_1},\ldots,\phi_{j_m}$, $j_k < i$, $1 \leq k \leq m$, then ϕ_i is said to *reference* ϕ_{j_k}, $1 \leq k \leq m$.

□

Definition 2.2. The *length* of a proof π is the number of occurrences of statements in π and is denoted length(π). In order to define the *space* of a proof p, let π consist of the sequence of statements $\phi_1,\ldots,\phi_n$, and for $1 \leq i < n$, let W_i be the set of statements ϕ_j, $j \leq i$, referenced by some statement ϕ_k, $k > i$. By convention, $W_n = \{\phi_n\}$. Thus W_i is the set of statements which must be remembered across a line drawn between ϕ_i and ϕ_{i+1}. Let

$|W_i|$ = space required to represent the set W_i in some reasonable encoding,

$$\text{space}(\pi) = \max_{1 \leq i \leq n} |W_i| .$$

□

Example 2.3. Let M be a nondeterministic Turing machine accepting a set $L(M)$. As outlined in the introduction, we can view a computation history of M on input x as a proof π that $x \in L(M)$. Statements are of the form

$$a_1 a_2 \ldots a_i q a_{i+1} \ldots a_n$$

representing the contents of the tape, head position, and current state. Then length(π) is the number of steps M takes on input x, and space(π) is the number of tape cells M uses.

□

3 MAIN RESULTS

3.1 An Equational Theory Complete for P with Natural Proof Systems that Require More Than Logarithmic Space

Definition 3.1.1.. A language L is an *equational language* if it consists of

i) a set of finitary *function symbols*, denoted f, g, $f_1,\ldots$, including some nullary function symbols called *constants* and denoted a, b, $a_1,\ldots$, and an *equality symbol* $=$,

ii) *terms* built up from symbols inductively, according to the rules

a) every constant symbol is a term,

b) if $x_1,\ldots,x_m$ are terms and f is an m-ary function symbol, then $fx_1 \ldots x_m$ is a term.

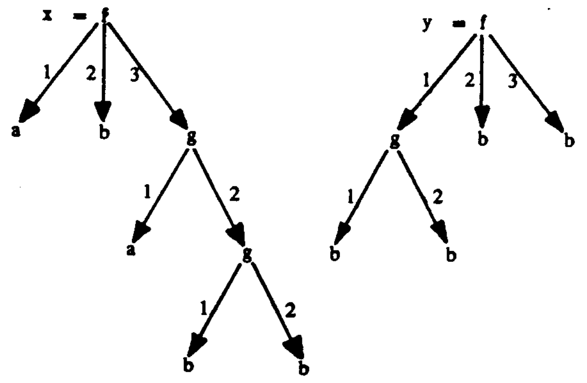
iii) *formulas* or *statements* $x = y$, where x and y are terms.

□

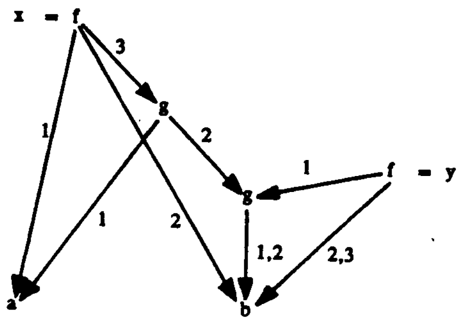
When working in an equational language L, we use a directed acyclic graph representation of terms and size measure given by the number of vertices in the graph. For example, let x and y be the terms

x = fabgagbb
y = fgbbbb

where f is ternary and g is binary. Then x and y may be represented first as trees, by



and then as directed acyclic graphs, by consolidating common subterms, so that x and y then become



If T is a set of terms, we take $|T|$ to be the number of vertices in the directed acyclic graph representation of T, as given by the above example. Note that if y is a subterm of x, then $|\{y,x\}| = |\{x\}|$. Equivalently, $|T|$ = the number of distinct subterms of terms in T.

If W is a set of formulas, we take $|W|$ to be $|T|$, where T is the set of all terms appearing in formulas of W.

Definition 3.1.2. Let

$$D = \{ fi \mid f \text{ is a function symbol}, 1 \leq i \leq \text{arity of } f\}.$$

A string $\alpha \in D^*$ will represent the position of a subterm in a term. For example, the term z = gbb occurs as a subterm of x above at position α = f3g2. λ represents the null string. We write $x\alpha z$ to denote that z occurs as a subterm of x at the position specified by α. Note that $x\lambda w$ iff $x = w$.

We write $x[\alpha/y]$ to denote the term x with the subterm at position α (if it exists) replaced by y.

□

When specifying a proof system in equational language L, we will often use *schemata* to specify an infinite set of axioms and rules For example, the rule representing transitivity of $\equiv$ is given by the schema

$$\frac{x\equiv y,\ y\equiv z}{x\equiv z}$$

where x, y, and z vary over terms of L.

Let P be a proof system in ℓ in which there is a symmetry rule

$$\frac{x\equiv y}{y\equiv x}$$

Syntactically, the symbol $\equiv$ is not symmetric, i.e. asserting $x\equiv y$ is not the same as asserting $y\equiv x$. In the presence of the symmetry rule, however, any interpretation of $\equiv$ must be a symmetric relation. For convenience, we would like to avoid the distinction between $x\equiv y$ and $y\equiv x$ on the syntactic level. The next theorem allows us to do this without loss of efficiency.

Let L' be formed from L by identifying formulas $x\equiv y$ and $y\equiv x$. We may think of the statements of L' as unordered pairs of terms $\{x,y\}$. Let P' be formed from P by replacing every statement $x\equiv y$ with $\{x,y\}$ in the axioms and rules, and removing the symmetry rule.

Lemma 3.1.3. To each proof π of $x\equiv y$ in P there corresponds a proof ρ of $\{x,y\}$ in P' with length length(π) $\leq$ length(π) and space(π) $\leq$ space(π).

Proof. Let P'' be a proof system in L' which is just P' with the addition of a rule R given by

$$\frac{\{x,y\}}{\{x,y\}}$$

To every proof π of $x\equiv y$ in P there corresponds a proof σ of $\{x,y\}$ in P'' with the same length and space requirements, just by mapping occurrences of $z\equiv w$ into occurrences of $\{z,w\}$. Applications of the symmetry rule of P go to applications of R of P''. It remains to delete all applications of R in σ.

Let ϕ_i be the result of some application of R in σ, and let ϕ_j be referenced by ϕ_i. Then $j<i$, and ϕ_i and ϕ_j are occurrences of the same statement. Alter σ by making all statements which reference ϕ_i reference ϕ_j instead. The space requirements are not increased, since no W_i increases. This may be repeated until all applications of R disappear, and the result is ρ.

□

In light of the above lemma, we will still use the symbol $\equiv$, but will consider it a syntactically symmetric relation, so that $x\equiv y$ and $y\equiv x$ are equivalent.

Definition 3.1.4. A *structure for L* is a pair $A = \langle A,I\rangle$ where A is a set, called the *domain*, and I is a map, called the *interpretation*, taking each constant symbol a of L to an element a_A of A and each m-ary function symbol f to a function $f_A: A^m \rightarrow A$. The interpretation extends inductively to terms by taking

$$(fx_1 \ldots x_m)_A = f_A(x_{1A},\ldots,x_{mA}).$$ □

Definition 3.1.5. Structure A is a *model of* $x\equiv y$, or $x\equiv y$ is *true in* A, if $x_A = y_A$.

□

If Γ is a finite set of formulas $\{x_1\equiv y_1,\ldots,x_n\equiv y_n\}$, how hard is it to determine, given x, y, and Γ, whether $x\equiv y$ is true in all models of Γ in which $\equiv$ is interpreted as equality? We answer this question with the following well-known construction.

Definition 3.1.6. The *free structure* T for L is the structure whose domain is the set of terms and whose interpretation is defined by

$$a_T = a$$

$$f_T(x_1,\ldots,x_m) = fx_1 \ldots x_m$$ □

Note that $x_T = x$ for all terms x, thus T is a model of $x\equiv y$ iff $x=y$.

Definition 3.1.7. The *Herbrand structure* for Γ, H_Γ (or just H, when Γ is understood), is defined as follows:

Let $\equiv_\Gamma$ be the smallest congruence relation on terms such that $x\equiv_\Gamma y$ for all formulas $x\equiv y$ in Γ. That is, $x\equiv_\Gamma y$ iff it is provable in the system E1 given below.

System E1.

i) $x\equiv_\Gamma x$ (reflexivity)

ii) $\dfrac{x\equiv_\Gamma y}{y\equiv_\Gamma x}$ (symmetry)

iii) $\dfrac{x\equiv_\Gamma y,\ y\equiv_\Gamma z}{x\equiv_\Gamma z}$ (transitivity)

iv) $\dfrac{x_1\equiv_\Gamma y_1,\ \ldots,\ x_m\equiv_\Gamma y_m}{fx_1\ldots x_m\equiv_\Gamma fy_1\ldots y_m}$ (congruence)

v) $x\equiv_\Gamma y$ for all $x\equiv y$ in Γ.

Rule iv) guarantees that the functions f_T are well-defined on $\equiv_\Gamma$-classes, so that we can form the quotient structure

$$H_\Gamma = T/\equiv_\Gamma$$

with domain

$$\{\ [x] \mid [x] \text{ is the } \equiv_\Gamma\text{-class of } x\ \}$$

and interpretation

$$a_H = [a]$$

$$f_H([x_1],\ldots,[x_m]) = [fx_1\ldots x_m]\ .$$ □

Note that $x_H = [x]$ for all terms x.

Theorem 3.1.8. $x\equiv y$ is true in all models of Γ iff $x\equiv y$ is true in H.

Proof. The only if part is trivial, since H is a model of Γ. Now suppose that H is a model of $x\equiv y$. Let A be any model of Γ. The function $f: T\rightarrow A$ defined by

$$f(x) = x_A$$

is a homomorphism, as may easily be checked. Define $x \approx y$ iff $f(x) \equiv f(y)$. It is easily verified that $\approx$ is a congruence relation such that $x \approx y$ for all $x \equiv y$ in Γ, so $\equiv_\Gamma$ is a refinement of $\approx$, by definition of $\equiv_\Gamma$. This says that f is well-defined on $\equiv_\Gamma$-classes, i.e. there is a homomorphism h such that

$$\begin{tikzcd} T \arrow[d, "{[\,]}"'] \arrow[rr, "f"] & & A \\ H \arrow[urr, "h"'] & & \end{tikzcd}$$

commutes. But now

$$\begin{aligned} x_H \equiv y_H &\rightarrow [x] \equiv [y] \\ &\rightarrow h([x]) \equiv h([y]) \\ &\rightarrow f(x) \equiv f(y) \\ &\rightarrow x_A \equiv y_A. \end{aligned}$$ □

The following theorem was proved in [Ko].

Theorem 3.1.9. The problem of determining whether $x \equiv y$ in H_Γ, given x,y, and Γ, is $\leq_{\log}$-complete for deterministic polynomial time. □

Corollary 3.1.10. The set

$$WP = \{ <\Gamma,x,y> \mid x \equiv y \text{ in all models of } \Gamma \}$$

is $\leq_{\log}$-complete for deterministic polynomial time. □

In addition to E1, the following are consistent and complete proof systems for WP, i.e. $x \equiv y$ is provable iff $x \equiv y$ is true in all models of Γ.

System E2.

i) $x \equiv x$ (reflexivity)

ii) $\dfrac{x \equiv y}{y \equiv x}$ (symmetry)

iii) $\dfrac{x \equiv y,\ y \equiv z}{x \equiv z}$ (transitivity)

iv) $x \equiv x[\alpha/y]$ where $x\alpha w$ and $y \equiv w$ appears in Γ (limited substitution)

System E3 is obtained from system E1 by adding a substitution rule

vi) $\dfrac{y \equiv w}{x \equiv x[\alpha/y]}$ where $x\alpha w$ (full substitution)

System E2 is a slight generalization of one used extensively in [Ko].

As shown in [Ko], a special case of the above problem is the circuit value problem of Ladner [La]. An instance of the circuit value problem consists of a list L of assignments to $c_1,\ldots,c_n$ of the form

$$c_i \equiv rhs_i$$

where rhs_i is one of

0,	
1,	
$c_j \wedge c_k$,	$j,k<i$,
$c_j \vee c_k$,	$j,k<i$,
$\neg c_j$,	$j<i$,

Such that each c_i, $1 \leq i \leq n$, appears on the left side of an assignment exactly once. The problem is to determine the Boolean value of c_n after the assignments are executed. If we take Γ to be the list L plus the axioms of the two-element Boolean algebra

$$\{ 0\wedge 0 \equiv 0,\ 0 \wedge 1 \equiv 0,\ 1 \wedge 0 \equiv 0,\ 1 \wedge 1 \equiv 1,\ 0 \vee 0 \equiv 0,\ 0 \vee 1 \equiv 1,\ 1 \vee 0 \equiv 1,\ 1 \vee 1 \equiv 1,\ \neg 0 \equiv 1,\ \neg 1 \equiv 0 \}$$

then, as shown in [Ko], H_Γ is a model of $c_n \equiv 1$ iff the Boolean value of c_n is 1.

A natural proof system for this special case of WP is

System E4.

i) all statements of L

ii) $\dfrac{c_i \equiv c_j \wedge c_k,\ c_j \equiv 1,\ c_k \equiv 1}{c_i \equiv 1}$

iii) $\dfrac{c_i \equiv c_j \vee c_k,\ c_j \equiv 1}{c_i \equiv 1}$

iv) $\dfrac{c_i \equiv c_j \vee c_k,\ c_k \equiv 1}{c_i \equiv 1}$

v) $\dfrac{c_i \equiv \neg c_j,\ c_j \equiv 1}{c_i \equiv 0}$

and their duals ii'), iii'), iv'), v') obtained by interchanging $\wedge$ and $\vee$, 0 and 1.

This constitutes a natural, consistent, and complete proof system for statements of the form $c_i \equiv 0$ or $c_i \equiv 1$. The non-logspace lower bound for this system follows easily from a result of Paul, Tarjan, and Celoni [PTC].

Definition 3.1.11. Let G be a rooted directed acyclic graph of out-degree 2. G ***can be pebbled with k pebbles*** if there is an algorithm to place pebbles on vertices of G and remove pebbles from G so that

i) a pebble is placed on a vertex only if all its immediate descendants have pebbles on them,
ii) the root of G is eventually pebbled,
iii) no more than k pebbles are ever on the graph simultaneously. □

Lemma 3.1.12. (Paul, Tarjan, Celoni [PTC]). For infinitely many n, there are graphs with n vertices which cannot be pebbled with less than cn/log(n) pebbles, where c>0 is independent of n.

□

Theorem 3.1.13. E4 requires space cn/log(n) for infinitely many n.

Proof. Let G be any directed acyclic graph with n vertices and out-degree at most 2. We may assume all interior vertices of G have out-degree 2, or else a new vertex can be added with edges to it from each interior vertex of out-degree one, without changing the number of pebbles required to pebble G by more than one. Assign a distinct symbol c_i to each vertex of G so that $c_1,\dots,c_n$ represents a topological sort of G, i.e. i<j iff c_i is a descendant of c_j. Let L be the list

$$\{\ c_i \equiv rhs_i \ |\ 1 \leq i \leq n\ \}$$

where $rhs_i \equiv 0$ if c_i is a leaf and $c_j \vee c_k$ if c_i is an interior vertex with descendants c_j and c_k. Let π be any proof in E4 of $c_n \equiv 0$. Let a pebble appear on vertex c_i of G at time t iff $c_i \equiv 0$ appears in W_t. It is easily shown that this algorithm obeys the pebbling rules, and the number of pebbles on G at time t is at most $|W_t|$, so G can be pebbled with at most space(π) pebbles. The result follows from Lemma 3.1.12. □

It is hard to imagine a proof system for the problem WP which would not encode pebblings in some way or other. The systems E1, E2, and E3 are all natural proof systems for WP, and apparently much stronger than E4, since large formulas can be built up. In E4, the size of formulas is at most 5, and there is no question which node of the graph G the statement $c_i \equiv 0$ represents. Nevertheless, proofs in E1, E2, and E3 can be shown to encode pebblings of graphs. We show a lower space bound for the system E2; the argument for the stronger systems E1 and E3 is significantly more complicated, and is deferred until a later paper.

In order to show a lower bound for the space requirements of E2, we define a weaker system E5 and show that proofs in E2 can be simulated by proofs in E5 with no loss of space efficiency. The statements of E5 will be the terms of E2. System $E5_u$ is defined with respect to a specific term u. Let Γ be any set of statements.

System $E5_u$.

i) u (reflexivity axiom)

ii) $\dfrac{x}{x[\alpha/y]}$ where $x\alpha w$ and $w \equiv y$ appears in Γ (limited substitution)

Theorem 3.1.14. $E5_u$ is a consistent and complete deductive system in that x is provable in $E5_u$ iff $x \equiv u$ is true in all models of Γ .

Proof. The proof is left to the reader. □

Lemma 3.1.15. To each proof π of $x \equiv y$ in E2 there corresponds a proof ρ of y in $E5_x$ such that space(ρ) ≤ space(π).

Proof. By Lemma 3.1.3 we may consider $x \equiv y$ and $y \equiv x$ as syntactically equivalent and disregard the symmetry rule of E2. The proof proceeds by induction on the length of π. If $x \equiv y$ is an instance of the reflexivity axiom, then $x = y$, so take ρ to be the single statement x. If $x \equiv y$ is an instance of the substitution axiom, then take ρ to be the proof consisting of the two statements x,y, where x is an instance of the reflexivity axiom of $E5_x$ and y follows from x by the substitution rule of $E5_x$. Then space(ρ) = $|\{x,y\}|$ = space(π). Finally, suppose $x \equiv y$ follows from $x \equiv z$, $z \equiv y$ by an application of the transitivity rule of E2. Encoded in π are shorter proofs π_1 of $x \equiv z$ and π_2 of $z \equiv y$, and space(π_1), space(π_2) ≤ space(π). By the induction hypothesis there are proofs ρ_1 of z in $E5_x$ and ρ_2 of y in $E5_z$ with space(ρ_i) ≤ space(π_i) , i=1,2. Placing these proofs end to end constitutes a proof ρ of y in $E5_x$, since the substitution rules of $E5_x$ and $E5_z$ are the same, and

$$\mathrm{space}(\rho) \leq \max_{i=1,2} \mathrm{space}(\pi_i) \leq \mathrm{space}(\pi).$$

□

Let G be a rooted directed acyclic graph such that each vertex has out-degree 0 or 2. Let $c_1,\dots,c_n$ be the vertices of G such that $c_1,\dots,c_n$ represents a topological sort of G. Let *L* be the equational language with constant symbols 0, $c_1,\dots,c_n$ and binary function symbol • (we write xy for x•y and parenthesize where necessary). Let

$$\Gamma' \ = \ \{\ c_i \equiv rhs_i \ |\ 1 \leq i \leq n\ \}$$

where

rhs_i = $c_j c_k$ if c_j and c_k are the immediate descendants of c_i in G,

rhs_i = 0 if c_i is a leaf

and let $\Gamma = \Gamma' \cup \{00 \equiv 0\}$. Then $<\Gamma, c_n, 0> \in$ WP. It is our intention to show that if π is a proof of c_n in $E5_0$, then G can be pebbled with at most space(π) pebbles. In this special case, we define a new proof system E6 in which to encode proofs in $E5_0$, and show that any proof of c_n in E6 directly encodes a pebbling of G.

System E6.

i) 0

ii) $\dfrac{x}{x[\alpha/00]}$ where $x\alpha 0$

iii) $\dfrac{x}{x[\alpha/c_i]}$ where $x\alpha(rhs_i)$.

In other words, we can start at 0 and derive new terms by substituting 00 for 0 anywhere 0 appears and substituting c_i for rhs_i anywhere rhs_i appears.

Let #edges(x) be the number of edges in the smallest directed acyclic graph representation of term x. If π is a proof $y_1,y_2,\dots,y_k$ of y in either $E5_0$ or E6, where $y_1 = 0$ and $y_k = y$, then let

$$\#\mathrm{edges}(\pi) = \max_{1 \leq i \leq k} \#\mathrm{edges}(y_i)\ .$$

Lemma 3.1.16. Let π be a proof of c_i in $E5_0$. Then there is a proof ρ of c_i in E6 with

$$\#\text{edges}(\rho) \leq \#\text{edges}(\pi).$$

Proof. Let $<$ be a binary relation on proofs in $E5_0$ of statements of the form c_i, some $1 \leq i \leq n$, where $\pi < \rho$ if π is a proof of c_i, ρ is a proof of c_j, and c_i is a descendant of c_j in G. Then $<$ is well-founded, so we may proceed by induction on $<$.

Basis. c_i is a leaf of G.

Let π be any proof of c_i in $E5_0$. It must be that the next to last statement is 0, since $c_i \equiv 0$ is in Γ and no other rule can yield c_i, so if ρ is the proof $0, c_i$ in E6, then

$$\#\text{edges}(\rho) \leq \#\text{edges}(\pi).$$

Induction step. c_i is an interior vertex of G, and the lemma holds for both descendants c_j and c_k of c_i.

Let π be a proof $y_1, \ldots, y_n$ of c_i in $E5_0$. It must be that $y_{n-1} = c_jc_k$ and $y_2 = 00$, since no other rules apply at those points. Thus π is the proof

$$0,\ 00,\ y_3, \ldots, y_{n-2},\ c_jc_k,\ c_i.$$

If any of the y_m, $3 \leq m \leq n-2$, is a single symbol c_ℓ, then π must look like

$$0,\ 00,\ \ldots,\ \text{rhs}_\ell,\ c_\ell,\ \text{rhs}_\ell,\ \ldots,\ c_jc_k,\ c_i$$

which can be shortened by deleting the two applications of the substitution rule of $E5_0$

$$\frac{\text{rhs}_\ell}{c_\ell} \quad \text{and} \quad \frac{c_\ell}{\text{rhs}_\ell}$$

without loss of space.

Similarly, if some y_m of π is 0, then the proof can be shortened and y_m deleted without loss of space. Finally we are left with the case in which each $y_j = x_jz_j$, all $2 \leq j \leq n-2$. For $2 \leq m \leq n-2$, it must be that $x_{m+1}z_{m+1}$ follows from x_mz_m by an application of the substitution rule to a *proper* subterm of x_mz_m; i.e., $x_{m+1}z_{m+1} = x_mz_m[\alpha/w]$, $w \equiv z$ in Γ, and $x_mz_m\alpha z$, but $\alpha \neq \lambda$. Rearrange π so that all applications to the left subterm are done first, followed by applications to the right subterm. Call this new proof π'. Then for some m, π' looks like

$$0,\ 00,\ u_3v_3,\ \ldots,\ u_{m-1}v_{m-1},\ c_j0,\ u_{m+1}v_{m+1},\ \ldots,\ u_{n-2}v_{n-2},\ c_jc_k,\ c_i$$

where $v_\ell = 0$, $\ell \leq m$, and $u_\ell = c_j$, $\ell \geq m$. But

$$\#\text{edges}(\pi') \leq \#\text{edges}(\pi),$$

since for all $\ell \leq m$, u_ℓ occurs as the left subterm of some term appearing in π, say $u_\ell w$, and

$$\#\text{edges}(u_\ell v_\ell) = \#\text{edges}(u_\ell 0) \leq \#\text{edges}(u_\ell w).$$

Similarly, for $\ell \geq m$,

$$\#\text{edges}(u_\ell v_\ell) \leq \#\text{edges}(wv_\ell),$$

where wv_ℓ occurs in π.

Break π' into two pieces

$$\pi_\ell = 0,\ 00,\ u_3v_3,\ \ldots,\ u_mv_m$$
$$\pi_r = u_mv_m, \ldots, u_{n-2}v_{n-2},\ c_jc_k,\ c_i$$

in which each term follows from the last by an application of the substitution rule of $E5_0$. By considering only the left subtrees of the terms in π_ℓ and the right subtrees of the terms in π_r, we get proofs

$$\pi_\ell' = 0,\ u_3,\ \ldots,\ u_{m-1},\ c_j$$

of c_j in $E5_0$ and

$$\pi_r' = 0,\ v_{m+1},\ \ldots,\ c_k$$

of c_k in $E5_0$. By the induction hypothesis, there are proofs

$$\rho_\ell' = 0,\ u_3',\ \ldots,\ u_{m-1}',\ c_j \quad \text{and}$$
$$\rho_r' = 0,\ v_{m+1}', \ldots,\ c_k$$

in E6, with no more edges than π_ℓ' and π_r', respectively. From this we get the sequences

$$\rho_\ell = 0,\ 00,\ u_3'0,\ \ldots,\ u_{m-1}'0,\ c_j0 \quad \text{and}$$
$$\rho_r = c_j0,\ c_jv_{m+1}',\ \ldots,\ c_jc_k,\ c_i$$

in which each term follows from the last by the rules of E6, and

$$\begin{aligned}\#\text{edges}(\rho_\ell) &= \#\text{edges}(\rho_\ell') + 2\\ &\leq \#\text{edges}(\pi_\ell') + 2\\ &= \#\text{edges}(\pi_\ell)\end{aligned}$$

and similarly,

$$\#\text{edges}(\rho_r) \leq \#\text{edges}(\pi_r).$$

By combining ρ_ℓ and ρ_r we get a proof ρ of c_i in E6, and

$$\begin{aligned}\#\text{edges}(\rho) &\leq \max\ \{\ \#\text{edges}(\rho_\ell),\ \#\text{edges}(\rho_r)\ \}\\ &\leq \max\ \{\ \#\text{edges}(\pi_\ell),\ \#\text{edges}(\pi_r)\ \}\\ &\leq \#\text{edges}(\pi).\end{aligned}$$

□

Let nodes(x) represent the set of symbols from $\{c_1, \ldots, c_n\}$ occurring term x, and let #nodes(x) be the cardinality of nodes(x). Let

$$\#\text{nodes}(\pi) = \max\ \{\ \#\text{nodes}(x)\ \mid\ x \text{ occurs in } \pi\ \}$$

where π is a proof in E6.

Lemma 3.1.17. If π is a proof of c_n in E6, then G can be pebbled with $\#\text{nodes}(\pi)$ pebbles.

Proof. Let π be the proof $y_1, \ldots, y_m$, where $0 = y_1$ and $y_m = c_n$. At time t, put pebbles on all the vertices of G which appear in nodes(y_t). Each set nodes(y_{t+1}) follows from nodes(y_t) according to the pebbling rules: if c_i becomes newly pebbled at time t + 1, i.e. if

$$c_i \in \text{nodes}(y_{t+1}) - \text{nodes}(y_t),$$

then c_i appeared in nodes(y_{t+1}) via an application of rule iv) of E6 of the form either

$$\frac{0}{c_i}$$

in which case c_i is a leaf, or

$$\frac{c_j c_k}{c_i}$$

in which case c_j, $c_k \in$ nodes(y_t), i.e. c_j and c_k were pebbled at time t. Since nodes(y_1) = ϕ and nodes(y_m) = $\{c_n\}$, π represents a pebbling of G in the desired way, and at most

$$\max \{ \#\text{nodes}(x) \mid x \text{ in } \pi \} = \#\text{nodes}(\pi)$$

pebbles are used. □

Theorem 3.1.18. System E2 requires cn/log(n) space.

Proof. For any proof π of c_n=0 in E2 there is a proof ρ of c_n in $E5_0$ and a proof σ of c_n in E6 with

$$\text{space}(\rho) \leq \text{space}(\pi) \text{ and}$$
$$\#\text{edges}(\sigma) \leq \#\text{edges}(\rho),$$

by Lemmas 3.1.15 and 3.1.16. By Lemmas 3.1.12 and 3.1.17,

$$\#\text{nodes}(\sigma) \geq \text{cn}/\log(n).$$

For any term x, $|x| \geq \frac{1}{2}\#\text{edges}(x)$, since each node has at most two descendants, and #edges(x) ≥ #nodes(x)-1, since there is a distinct leaf in the representation of x for each c_i in nodes(x), and each leaf has in-degree at least 1 (except for the case in which x is a single symbol). Combining these results, we have

$$\begin{aligned} \text{space}(\pi) &\geq \frac{1}{2} \#\text{edges}(\sigma) \\ &\geq \frac{1}{2} (\#\text{nodes}(\sigma) - 1) \\ &\geq \frac{1}{2} (\text{cn}/\log(n) - 1). \end{aligned}$$ □

It is expected that other natural proof systems for equational theories will behave similarly. For example, Cardoza, Lipton, and Meyer [CLM] have recently shown that the word problem for commutative semigroups is complete for exponential space, and so far the best know lower length bound for any proof system is a single exponential. Can the natural proof systems for this theory be shown to require 2^{2^n} in the worst case?

3.2 A Theory Complete for PSPACE with a Natural Proof System that Requires More Than Polynomial Time

Consider the following problem: given a finite set of functions $f_1,...,f_k$:A→A, where A is a set with n elements, and another function h:A→A, can the function h be obtained by some sequence of compositions of the f_i's? In other words, is h a member of the submonoid of F_A generated by $f_1,...,f_k$, where F_A is the monoid of functions A→A under composition?

Let functions A→A be represented by bipartite directed graphs with 2n vertices and n edges. It is easy to compute the composition f∘g of two functions f and g, just by rearranging edges.

Definition 3.2.1.

$$\text{GEN} = \{ <A, f_1,...,f_k, h> \mid h \text{ is generated by } f_1,...,f_k \text{ under composition} \}$$ □

We want to show first that GEN is complete for PSPACE. It is clearly in nondeterministic linear space, hence in deterministic n^2 space by a result of Savitch [Sa], since we can start with a representation of the identity on A, and then apply the f_i's nondeterministically in some order, accepting if h ever appears. It is unknown however whether GEN is a hardest context sensitive language.

Given f_i:A→A, 1≤i≤k, a natural complete proof system for proving that functions are generated by $f_1,...,f_k$ is the following, consisting of k axioms and one rule of inference:

System G1.

i) GEN(f_i), 1≤i≤k

ii) $\dfrac{\text{GEN(f), GEN(g)}}{\text{GEN(f}\circ\text{g)}}$

We will show later that this natural mechanism is slow, that is, requires proofs of length exponential in the size of A in the worst case. If all proof systems for GEN could be shown to encode proofs in G1, then P≠PSPACE.

In order to show GEN complete for PSPACE, we first show another problem complete for PSPACE, and then reduce instances of this problem to instances of GEN.

Definition 3.2.2. The finite automaton intersection problem. Let $F_1,...,F_k$ be k deterministic finite automata with a common alphabet Σ, and let $L(F_i)$ be the language accepted by F_i. The problem INT is to determine whether the F_i accept a common element of Σ^*, i.e.

$$\text{INT} = \{ <F_1,...,F_k> \mid \bigcap_{i=1}^{k} L(F_i) \text{ is nonempty} \}.$$ □

Lemma 3.2.3. INT is $\leq_{\log}$-complete for PSPACE.

Proof. To see that INT is in nondeterministic linear space, given $F_1,...,F_k$, place markers on the start states of $F_1,...,F_k$, and guess a string $x \in \Sigma^*$, moving markers according to the rules of $F_1,...,F_k$, and accept if at some point each F_i has a final state marked. Then INT ∈ PSPACE by a result of Savitch [Sa].

It remains to reduce any set in PSPACE to the set INT via a logspace reduction. Let M be a single tape, deterministic Turing machine with space bound p, where p is a polynomial, p(n)≥n. Let x be any input string over M's tape alphabet Σ, and let n = |x|, where |x| is the length of x. Let Q be a set of symbols representing the states of M's finite control and let #, ƀ be two other symbols, such that Σ, Q, and {#,ƀ} are pairwise disjoint. Let

$$\Delta = \Sigma \cup Q \cup \{ƀ\}.$$

A string

$$\#ID_0\#ID_1\# \ldots \#ID_m\#\# \in (\Delta \cup \{\#\})^*$$

represents a *valid computation* of M on input x if

i) each ID_i is an instantaneous description of M, consisting of the contents of M's tape (padded out to length p(n) with $\not b$'s), the position of M's tape head, and the state of M's finite control,

ii) each ID_{i+1} follows from ID_i in one step according to the transition rules of M,

iii) ID_0 is the start configuration of M on input x, and ID_m is an accepting configuration.

For example, if $\Sigma = \{0,1\}$, q_0 is M's start state, and x = 011001, then

$$ID_0 = q_0011001\not b \ldots \not b \ .$$

If the transition rules of M dictate that, when reading 0 on the tape in state q, M should print 1, move its head right, and enter state p, and if

$$ID_i = 01q0100\not b \ldots \not b \ ,$$

then

$$ID_{i+1} = 011p100\not b \ldots \not b \ .$$

Then M accepts $x \in \Sigma^*$ iff there is a valid computation

$$\text{VALCOMP} = \#ID_0\#ID_1\#\ldots\#ID_m\#\# \in (\Delta \cup \{\#\})^*$$

of M on input x. We will construct a set of finite automata with input alphabet $\Delta \cup \{\#\}$. The intersection of the sets accepted by these automata will be the singleton set consisting of the string VALCOMP above if it exists, and ϕ otherwise.

Assume that M always takes an even number of steps, and that M has a unique accept state, q_{acc}, and erases its tape before accepting, leaving the head at the left end of the tape. These assumptions are without loss of generality, since the finite control size is at most doubled.

Let F_i^{even} be a deterministic finite automaton accepting the set

$$(\#\Delta^{i-1}a_1a_2a_3\Delta^{p(n)-i-2}\#\Delta^{i-1}b_1b_2b_3\Delta^{p(n)-i-2})^*\#\#$$

where $a_i, b_i \in \Delta$, $1 \le i \le 3$, and $b_1b_2b_3$ can follow from $a_1a_2a_3$ according to the transition rules of M. I.e., F_i^{even} checks whether the ith, (i+1)st, and (i+2)nd symbols of ID_{k+1} follow from the ith, (i+1)st, and (i+2)nd symbols of ID_k, for even k.

For example, if M when reading 0 in state q must print 1, move its head left, and enter state p, and if $a_1a_2a_3 = 1q0$ then $b_1b_2b_3 = p11$; If $a_1a_2a_3 = q00$ then $b_1b_2b_3$ can be either 010 or 110.

A straightforward construction of F_i^{even} gives $2s^3p(n)$ states, where s is the cardinality of Δ, and F_i^{even} has a unique final state. The details of the construction are left to the reader. It is clear that

$$u \in \bigcap_{i=1}^{p(n)-2} L(F_i^{even})$$

iff

$$u = \#ID_0\#ID_1\#\ldots\#ID_{2m+1}\#\#$$

and ID_{2i+1} follows from ID_{2i} according to the transition rules of M, $0 \le i \le m$.

Similarly, construct F_i^{odd} to accept the set

$$\#\Delta^{p(n)}(\#\Delta^{i-1}a_1a_2a_3\Delta^{p(n)-i-2}\#\Delta^{i-1}b_1b_2b_3\Delta^{p(n)-i-2})^*\#\Delta^{p(n)}\#\#,$$

i.e. the F_i^{odd} do the same as the F_i^{even}, except they check that the even ID's follow from the odd ID's. F_i^{odd} may be constructed similar to F_i^{even}. F_i^{odd} has at most $2s^3p(n)$ states and a unique final state, and

$$u \in \bigcap_{i=1}^{p(n)-2} L(F_i^{odd})$$

iff

$$u = \#ID_0\#ID_1\#\ldots\#ID_{2m+1}\#\#$$

and ID_{2k} follows from ID_{2k-1} according to the transition rules of M, $1 \le k \le m$.

In addition, construct a deterministic finite automation f_{ends} which checks that ID_0 is the start configuration of M and the last ID the accept configuration of M, $q_{acc}\not b \not b \ldots \not b$. F_{ends} has $\le 2s^3p(n)$ states and a unique final state.

We have constructed 2p(n)-3 automata, each with at most $2s^3p(n)$ states, or $O(s^3p(n)^2)$ states in all. It is left to the reader to verify that the above construction can be done in space $\log(p(|x|) + |M|)$, where $|M|$ is the length of a standard encoding of M. Moreover,

$$u \in L(F_{ends}) \cap \bigcap_{i=1}^{p(n)-2} L(F_i^{even}) \cap L(F_i^{odd})$$

iff

$$u = \text{VALCOMP} \ ,$$

thus

$$L(F_{ends}) \cap \bigcap_{i=1}^{p(n)-2} L(F_i^{even}) \cap L(F_i^{odd}) \quad \text{is nonempty}$$

iff

M accepts x. □

Theorem 3.2.6. GEN is $\le_{log}$-complete for PSPACE.

Proof. As noted in the introduction to this section, GEN $\in$ PSPACE, so it remains to show GEN is hard for PSPACE. Since $\le_{log}$ is transitive, it suffices to reduce instances of the problem INT as constructed in Lemma 3.2.3 to instances of GEN.

Given M and x, let $F_1,\ldots, F_k$ be the instance of INT constructed in Lemma 3.2.3 such that

M accepts x

iff

$$\bigcap_{i=1}^{k} L(F_i) \text{ is nonempty.}$$

Observe from the construction of each F_i in the lemma that F_i has only one final state q_i^{final}. Let q_i^{start} be the start state of F_i and denote by q_i^j the j^{th} state of F_i. Let Σ be the common alphabet of the F_i.

Let A be the disjoint union of the sets of states of the k automata, plus three extra elements o_1, o_2, o_3. For each $a \in \Sigma$, define $f_a: A \to A$ as follows:

$f_a(q_i^j) =$ the state of F_i that q_i^j goes to under input symbol a, according to the transition rules of F_i

$f_a(o_1) = o_3$

$f_a(o_2) = o_2$

$f_a(o_3) = o_3$.

For $w \in \Sigma^*$, define f_w inductively by

$f_\lambda =$ the identity on A

$f_{wa} = f_a \circ f_w$.

It is easily demonstrated by induction on the length of w that

$f_w(q_i^j) =$ the state of F_i that q_i^j goes to under input string w, according to the transition rules of F_i.

Note in particular that

F_i accepts w

iff

$f_w(q_i^{start}) = q_i^{final}$.

Let f_{init} be a new function defined by

$f_{init}(q_i^j) = q_i^{start}$ for all states q_i^j of F_i

$f_{init}(o_1) = o_2$

$f_{init}(o_2) = o_3$

$f_{init}(o_3) = o_3$.

The set A will be the set on which our instance of GEN will be defined, and the set of generating functions $A \to A$ will be

$\{f_{init}\} \cup \{f_a \mid a \in \Sigma\}$.

Finally, let $h: A \to A$ be defined by

$h(q_i^j) = q_i^{final}$ for all states q_i^j of F_i

$h(o_i) = f_{init}(o_i)$, $1 \leq i \leq 3$.

It is left to the reader to verify that all the above functions $A \to A$ can be constructed in logspace, given $F_1, \ldots, F_k$.

Now we claim that

$$\bigcap_{i=1}^{k} L(F_i) \neq \phi$$

just in case h is generated by $\{f_{init}\} \cup \{f_a \mid a \in \Sigma\}$ under composition.

Suppose

$$\bigcap_{i=1}^{k} L(F_i) \neq \phi.$$

Then for some $w \in \Delta^*$ and for all $1 \leq i \leq k$,

$f_w(q_i^{start}) = q_i^{final}$

Moreover, by definition of the f_a,

$f_w(o_1) = o_3$,

$f_w(o_2) = o_2$,

$f_w(o_3) = o_3$.

Then $h = f_w \circ f_{init}$, as is easily verified by the definitions of f_{init} and h.

Now suppose h is generated by $\{f_{init}\} \cup \{f_a \mid a \in \Sigma\}$. h is not the identity on A, so there are g, f such that $h = g \circ f$, g is generated by $\{f_{init}\} \cup \{f_a \mid a \in \Sigma\}$, and f is one of f_{init}, f_a, $a \in \Sigma$. If $f \neq f_{init}$, then $f(o_1) = o_3$, and $g(o_3) = o_3$, a contradiction since $h(o_1) = o_2$. Thus $f = f_{init}$.

If $g = g_1 \circ f_{init} \circ g_2$, where g_2 is generated by $\{f_a \mid a \in \Sigma\}$ and g_1 is generated by $\{f_{init}\} \cup \{f_a \mid a \in \Sigma\}$, then

$$\begin{aligned} g \circ f(o_1) &= g_1(f_{init}(g_2(f_{init}(o_1)))) \\ &= g_1(f_{init}(g_2(o_2))) \\ &= g_1(f_{init}(o_2)) \\ &= o_3, \end{aligned}$$

again a contradiction. Thus g is generated by $\{f_a \mid a \in \Sigma\}$. This says that

$h = f_w \circ f_{init}$, $w \in \Sigma^*$.

Since $h(q_i^j) = q_i^{final}$ and $f_{init}(q_i^j) = q_i^{start}$, it must be that

$f_w(q_i^{start}) = q_i^{final}$, $1 \leq i \leq k$.

Thus

$$w \in \bigcap_{i=1}^{k} L(F_i).$$

□

It appears that the function f_{init} is essential to this particular construction. Is the problem still complete if we require the functions to be 1−1? In this case, we are asking a question about membership of a given permutation of n letters in a subgroup of the symmetric group on n letters.

The following result establishes a lower bound on the complexity of proofs in G1.

Theorem 3.2.7. For infinitely many n there is a set A, $|A| = n$, and functions f_0, f_1, and h such that h is generated by f_0 and f_1 under composition, but the shortest proof of GEN(h) in G1 is of length at least $2^{cn/\log(n)}$, where $c > o$ is independent of n.

Proof. There are two problems to overcome in the proof of this theorem. First, we can easily construct f_0, f_1, and h such that

$h = f_0 \circ f_1 \circ f_1 \circ \ldots \circ f_0$

for some exponential length string of f_0's and f_1's, but that is not to say that there is not a polynomial length proof of GEN($f_0 \circ f_1 \circ f_1 \circ \ldots \circ f_0$). For example, the string

$$h = \underbrace{f_0 \circ f_0 \circ \ldots \circ f_0}_{m}$$

in which each component is f_0 has a proof of length log(m) in G1. Second, even if we could insure that no proof of GEN(h) using a particular composition of f_0's and f_1's is less than exponential length, that is not to say that there is some other, perhaps shorter, sequence of f_0's and f_1's whose composition also yields h.

Let m be any power of 2. For any integer i, let b_i be the m-bit binary representation (low order bit first) of t, $0 \le t \le 2^m - 1$, where $t = i \bmod 2^m$, and let r_i be the log(m)-bit binary representation (low order bit first) of t, $0 \le t \le m-1$, where $t = i \bmod m$. Form a string x by starting with the string $b_0 b_1 \ldots b_{2m-1}$ and then inserting r_i before the ith digit of $b_0 b_1 \ldots b_{2m-1}$.

If w is any string in $\{0,1\}^*$, call an occurrence of a digit in w *blue* if its position in w is a multiple of log(m)+1, otherwise call it *red*. Thus in x, the digits of the b_i are blue and the r_i occur as consecutive blocks of log(m) red digits between two blue digits. Construct a sequence of finite automata F_i such that

$$\{x\} = \bigcap_{i=1}^{2m+2\log(m)+2} L(F_i)$$

as follows:

The first 2log(m)+1 automata will ignore the blue digits and check that the red digits occur in the proper order. For $1 \le i \le \log(m)$, automaton F_i will check that the ith bit of r_{j+1} in x follows properly from the first i bits of r_j, for even j. This is done by looking at each of the first (low order) i−1 bits of r_j and remembering whether any of them are zero. If so, the ith bit of r_{j+1} must be the same as the ith bit of r_j; if not, the ith bits of r_{j+1} and r_j must differ. The automaton remembers which, skips over log(m)+1 digits and checks whether the next digit is correct. If not it enters a dead state; if so it counts to the end of r_{j+1} and starts again. The automaton $F_{i+\log(m)}$, $1 \le i \le \log(m)$, will check whether the ith bit of the r_{j+1} in x follows from the ith bit of r_j, for odd j. $F_{i+\log(m)}$ is similar in construction to F_i The automaton $F_{2\log(m)+1}$ will check that the first log(m) bits are 0 and the last log(m)+1 bits are 1.

Each automaton F_i, $1 \le i \le 2\log(m)+1$, can be constructed with O(log(m)) states and a unique final state. The constructions are straightforward and are left to the reader.

The automata $F_{i+2\log(m)+1}$, $1 \le i \le 2m+1$, will check that the blue digits occur in the proper order. Automaton $F_{i+2\log(m)+1}$, $1 \le i \le m$, will check that the ith bit of b_{j+1} follows properly from the first i bits of b_j, for even j. These automata are similar in design to the F_i, $1 \le i \le \log(m)$, except they do not need to count to get from the ith bit of b_j to the ith bit of b_{j+1}. Instead, they assume that the string is accepted by all the F_i, $1 \le i \le 2\log(m)+1$, and use the red digits to locate the ith digit of b_{j+1}, by just looking for the next occurrence of r_i. This allows us to construct $F_{i+2\log(m)+1}$, $1 \le i \le m$, with O(log(m)) states. The automaton $F_{i+2\log(m)+1}$, $m+1 \le i \le 2m$, will check that the ith digit of b_{j+1} follows from the first i digits of b_j, for odd j.

Finally, the automaton $F_{2m+2\log(m)+2}$ will check that the first m blue digits are 0 and the last m are 1. This is done with O(log(m)) states by scanning the first log(m) bits of the input and then insuring that all blue digits before the next occurrence of log(m) consecutive red 0's are 0, then accepting at the end of the input string iff all the blue digits since the last occurrence of log(m) consecutive red 0's are 1.

The intersection of the sets accepted by these automata is exactly {x}, as desired, and each automaton can be constructed to have at most O(log(m)) states and a unique final state.

As in the proof of Theorem 3.2.6, let A be the disjoint union of the states of F_i, $1 \le i \le 2m+2\log(m)+2$, and let n be the cardinality of A. Then $m \ge cn/\log(n)$, for some c>0 independent of n. Let $f_0: A \to A$ and $f_1: A \to A$ be defined by

$f_a(q_i^j) =$ the state of F_i that q_i^j goes to under input symbol a according to the transition rules of F_i

for $a \in \{0,1\}$. Define f_w for $w \in \{0,1\}^*$ inductively, by

$$f_\lambda = \text{the identity on } A$$
$$f_{wa} = f_a \circ f_w, \quad a \in \{0,1\}, \quad w \in \{0,1\}^*.$$

Let $h = f_x$. As in Theorem 3.2.6, if q_i^{start} is the start state of F_i, and if q_i^{final} is the (unique) final state, then

$$f_w(q_i^{start}) = q_i^{final}$$

iff

F_i accepts w.

Since

$$\{x\} = \bigcap_{i=1}^{2m+2\log(m)+2} L(F_i),$$

we have for $w \in \{0,1\}^*$ that

$$h = f_w \to f_w(q_i^{start}) = q_i^{final}, \quad 1 \le i \le 2m+1$$
$$\to w \in \bigcap_{i=1}^{2m+2\log(m)+2} L(F_i)$$
$$\to w = x.$$

Thus x represents the *unique* composition f_x of functions f_0 and f_1 that yields h. For this reason, instead of considering proofs in G1 that f_x is generated by f_0 and f_1, we may consider proofs in G1 that x is generated by 0 and 1 in the monoid $\{0,1\}^*$. To see this, let $\sigma: \{0,1\}^* \to F_A$ (recall F_A is the monoid of functions A→A) be defined by

$$\sigma(w) = f_w.$$

It is easily verified that σ is a homomorphism. Then σ is 1−1 on substrings of x, otherwise there would be a $w \ne x$ with $f_w = f_x$. Hence for substrings w of x,

$$f_w = f_z \circ f_y \quad \text{iff} \quad w = yz.$$

This says that, for substrings w of x, an application of the rule

$$\frac{\mathrm{GEN}(f_z),\ \mathrm{GEN}(f_y)}{\mathrm{GEN}(f_w)}$$

of G1 is a valid application iff

$$\frac{\mathrm{GEN}(z),\ \mathrm{GEN}(y)}{\mathrm{GEN}(w)}$$

is a valid application. Moreover, if we consider only those proofs of G1 in which each intermediate statement is later referenced, then it is easily verified by induction on the length of the proof that if GEN(w) appears in a proof of GEN(x) in G1, then w is a substring of x, and if GEN(f_w) appears in a proof of GEN(f_x) in G1, then w is a substring of x.

By the above argument we see that

$$\mathrm{GEN}(w_1),\mathrm{GEN}(w_2),\ldots,\mathrm{GEN}(x)$$

is a proof of GEN(x) in G1 iff

$$\mathrm{GEN}(f_{w_1}),\mathrm{GEN}(f_{w_2}),\ldots,\mathrm{GEN}(f_x)$$

is a proof of GEN(f_x) in G1, i.e. σ provides a 1–1 length preserving correspondence between proofs of GEN(x) and proofs of GEN(f_x).

It then suffices to show that no proof of GEN(x) in G1 with axioms GEN(0) and GEN(1) is shorter than 2^{m-1}. We prove this by first showing that there is no $w\in\{0,1\}^*$, $|w|\geq 2m(\log(m)+1)$, with more than one occurrence as a substring of x. Note there is a proof of GEN(0^{2^m}) of length m, by doubling the length of the string of 0's in each step of the proof. The following claim allows us to circumvent this problem.

Claim. Let $d = 2m(\log(m)+1)$. Every string of length greater than or equal to d occurs at most once as a substring of x.

Proof of claim. Let $|y|\geq d$ and suppose there are two distinct occurrences of y in x, say y_1, y_2. The digits of y_1 and y_2 inherit their colors, red or blue, from their positions in x. Since $|y|\geq d$, all digits of some b_j are completely within y_1, and those of some b_k are completely within y_2. Suppose that the blue digits of y_1 and y_2 line up, i.e. the sth digit of b_j occurs in y_1 in the same position as the tth digit of b_k in y_2. If s=t, then $b_j=b_k$, contradicting the assumption that the two occurrences of y were distinct. If $s\neq t$, then at least one of y_1, y_2, say y_1, must contain the high order bit of some b_{j-1} and the low order bit of b_{j+1} (recall the representation of numbers is low order bit first). Since the low order bits of b_j and b_{j+1} must differ, the the corresponding digits of y_2 must differ. These are the ith bits, i>1, of some b_k, b_{k+1}. But in order for the ith digits of b_k and b_{k+1} to be different, all lower order digits of b_k must be 1 and all lower order digits of b_{k+1} must be 0. But this says that the high order digit of b_{j-1} is 1 and the high order digit of b_j is 0, which is impossible since the high order digit of b_j changes only once in the interval $0\leq j\leq 2^m-1$, and then only from 0 to 1.

Finally, suppose the blue digits of y_1 and y_2 do not line up. There are two blue digits in y_1 which are the lowest order digits of some b_j, b_{j+1}, and these digits must differ. Since these digits are mlog(m)+m apart, there are red digits in y_2 at a distance mlog(m)+m apart which are different. But any two red digits mlog(m)+m apart in x are the ith digits of some r_k and r_{k+m}, and $r_k = r_{k+m}$. This contradiction establishes the claim.

Let y be any string in $\{0,1\}^*$ and let π be a proof of GEN(y) in G1. Let #(y) be the number of occurrences of statements GEN(z) in π such that $|z|\geq d$ and GEN(z) occurs in π.

Claim. If $|y|\geq d$ and y is a substring of x, then

$$\#(y) \geq \lfloor |y|/d \rfloor.$$

Proof of claim. The proof is by induction on the length of y. For $d\leq |y| <2d-1$,

$$\lfloor |y|/d \rfloor = 1,$$

and the proof must contain the statement GEN(y).

For $|y|\geq 2d-1$, GEN(y) must follow from two statements GEN(u), GEN(v) occurring earlier in the proof, where y = uv.

Let z be arbitrary, $|z|\geq d$. Since u and v have nonoverlapping occurrences in x, z cannot occur as a substring of both u and v, by the previous claim. Thus

$$\#(y) = \#(u) + \#(v) + 1.$$

The extra 1 is for the statement GEN(y). If both $|u|\geq d$ and $|v|\geq d$, then by the induction hypothesis,

$$\begin{aligned}\#(u) + \#(v) + 1 &\geq \lfloor |u|/d \rfloor + \lfloor |v|/d \rfloor + 1\\ &\geq \lfloor (|u|+|v|)/d \rfloor\\ &= \lfloor |y|/d \rfloor .\end{aligned}$$

If one of $|u|,|v|<d$, say $|u|$, then $|v| = |y|-|u| \geq d$. By the induction hypothesis,

$$\begin{aligned}\#(y) = \#(v) + 1 &\geq \lfloor |v|/d \rfloor + 1\\ &= \lfloor (|y|-|u|)/d \rfloor + 1\\ &\geq \lfloor (|y|-(d-1))/d \rfloor + 1\\ &\geq \lfloor |y|/d \rfloor,\end{aligned}$$

and the claim is verified.

Thus any proof of GEN(x) must be of length at least

$$\begin{aligned}\lfloor |x|/d \rfloor &= \lfloor m(\log(m)+1)2^m/2m(\log(m)+1) \rfloor\\ &= 2^{m-1}\\ &\geq 2^{cn/\log(n)},\end{aligned}$$

for some c>0 independent of n. □

Acknowledgments

The author wishes to thank Alan Cobham and Merrick Furst for many helpful ideas and discussions. He is also indebted to Larry Stockmeyer for an observation which allowed the original $2^{c\sqrt{n}}$ lower bound for G1 to be improved to $2^{cn/\log(n)}$.

REFERENCES

[AHU] Aho, A.V., J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass, 1975.

[CLM] Cardoza, E., R. Lipton, and A.R. Meyer, "Exponential Space Complete Problems for Petri Nets and Commutative Semigroups," *Proc. 8th ACM STOC*, May 1976, pp. 50-54

[CR] Cook, S. and R. Reckhow, "On the Lengths of Proofs in the Propositional Calculus," *Proc. 6th ACM STOC*, May 1974, pp. 135-148.

[Ga] Galil, Z., *The Complexity of Resolution Procedures for Theorem Proving in the Propositional Calculus*, Ph.D. Thesis, Cornell University, June 1975.

[Kn] Knuth, D.E., *The Art of Computer Programming*, v. 3, *Sorting and Searching*, Addison-Wesley, Reading, Mass, 1975.

[Ko] Kozen, D., "Complexity of Finitely Presented Algebras," *Proc. 9th ACM STOC*, May 1977, pp. 164-177.

[La] Ladner, R.E., "The Circuit Value Problem is Logspace Complete for P," *SIGACT News* 7:1, January 1975.

[PTC] Paul, W.J., R.E. Tarjan, and J.R. Celoni, "Space Bounds for a Game on Graphs," *Proc. 8th ACM STOC*, May 1976, pp. 149-160.

[Sa] Savitch, W.J., "Relationships Between Nondeterministic and Determinstic Tape Complexities," *J. Comput. Syst. Sci. 4*, 1970.

[Ts] Tseitin, G.S., "On the Complexity of Derivations in the Propositional Calculus," *Studies in Constructive Mathematics and Mathematical Logic, Part II*, A.O. Silenko, ed., 1968, pp. 115-125.