JOURNAL OF COMPUTER AND SYSTEM SCIENCES 12, 222–268 (1976)

On the Equivalence, Containment, and Covering Problems for the Regular and Context-Free Languages*

HARRY B. HUNT III†

Aiken Computation Laboratory, Harvard University, Cambridge, Masaschusetts 02138

DANIEL J. ROSENKRANTZ

General Electric Company, Research and Development Center, Schenectady, New York 12345

AND

THOMAS G. SZYMANSKI†

Electrical Engineering Department, Princeton University, Princeton, New Jersey 08540

Received December 18, 1975

We consider the complexity of the equivalence and containment problems for regular expressions and context-free grammars, concentrating on the relationship between complexity and various language properties. Finiteness and boundedness of languages are shown to play important roles in the complexity of these problems. An encoding into grammars of Turing machine computations exponential in the size of the grammar is used to prove several exponential lower bounds. These lower bounds include exponential time for testing equivalence of grammars generating finite sets, and exponential space for testing equivalence of non-self-embedding grammars. Several problems which might be complex because of this encoding are shown to simplify for linear grammars. Other problems considered include grammatical covering and structural equivalence for right-linear, linear, and arbitrary grammars.

1. INTRODUCTION

There are many problems of interest concerning language descriptors such as regular expressions and context-free grammars (cfg's). These problems include the equivalence problem (Given two language descriptors, do they describe the

* Portions of this paper were presented at the 1974 Sixth Annual ACM Symposium on the Theory of Computing.

† The research of the first author was supported in part by NSF Grant GJ-35570. The research of the third author was supported in part by NSF Grants GJ-35570 and DCR74-21939.

same language?), the containment problem (Given two descriptors, does the language described by the first contain the language described by the second?), the grammatical covering problem [10], and the structural equivalence of grammars [21, 25]. In this paper we study the computational complexity of these and other problems. The complexity is analyzed as a function of simply stated properties of the descriptors or languages involved.

Section 2 concentrates on regular expression equivalence, and Section 3 on cfg equivalence. As an example of how complexity is related to language properties, testing a regular expression for inequivalence to a fixed regular language can be done in polynomial time if the fixed language is finite, is NP-complete if the fixed language is infinite but bounded, and is PSPACE-complete if the fixed language is unbounded. Testing two regular expressions for inequivalence is shown to be NP-complete for star-free regular expressions and PSPACE-complete for regular expressions of any fixed star height $\geqslant 1$. Testing both arbitrary and linear grammars for inequivalence to a fixed language can be done in polynomial time if the fixed language is finite, is NP-hard if the fixed language is infinite, is PSPACE-hard if the fixed language is unbounded, and is undecidable if the fixed language contains an unbounded regular subset.

In Section 4, exponential lower time bounds are proved for a variety of problems, including the equivalence problem for cfg's generating finite languages. Exponential lower space bounds are also proved for a variety of problems, including the equivalence problem for non-self-embedding cfg's. The results utilize an encoding, of Turing machine computations into grammars, based upon a "squaring" capability of cfg's that allows one to obtain cfg's for which the shortest or longest string in the generated language is exponential in the size of the grammar. This capability implies that decision algorithms which depend on the length of the shortest string will be exponential, and therefore complicates such problems as structural equivalence, equivalence of $LL(k)$ grammars and boundedness of cfg's. We show simplications when the grammars involved in these problems are restricted to be linear cfg's, so that the shortest string generated is linear in the size of the grammar.

In Section 5 we study grammatical covering and structural equivalence. We show these problems are PSPACE-complete for right-linear and for linear cfg's. Also for arbitrary cfg's, covering is undecidable, while structural equivalence is decidable, but PSPACE-hard.

We use two main approaches to establishing lower bounds on the complexity of various problems. One approach is to encode Turing machine computations of a given complexity into a particular problem of interest. This approach has been used by Stockmeyer and Meyer [24, 29, 30]. The other approach is to encode into the problem of interest, the problem of determining whether a given language descriptor describes the set of all strings over a fixed finite alphabet. This latter approach has been used by Greibach [11], Hopcroft [12], and Hunt [14].

Several preliminary definitions are needed to read this paper. We use $| x |$ to denote the length of x if x is a string, and the cardinality of x if x is a set.

See [13] for definitions of a *regular set*, *regular* or *type-3 grammar*, *finite automaton*, *nondeterministic finite automaton* (ndfa), *context-free grammar* (cfg), *context-free language* (cfl), *derivation*, *leftmost derivation*, *Turing machine* (Tm), and *linear bounded automaton* (lba).

DEFINITION 1.1. We use λ to denote the empty string and ϕ to denote the empty language.

Regular expressions over finite alphabet Σ are defined recursively as follows.

(a) λ, ϕ, and each a in Σ are regular expressions.

(b) If A and B are regular expressions, then so are $(A) \cup (B)$, $(A) \cdot (B)$, and $(A)^*$.

(c) Nothing else is a regular expression.

The language *denoted by* regular expression R is written $L(R)$.

The *star height* SH of a regular expression is defined recursively:

$$\begin{aligned} SH(a) &= 0 \quad \text{for } a \text{ in } \Sigma, \\ SH(\lambda) &= 0, \quad SH(\phi) = 0, \\ SH((A) \cup (B)) &= \max\{SH(A), SH(B)\}, \\ SH((A) \cdot (B)) &= \max\{SH(A), SH(B)\}, \\ SH((A)^*) &= SH(A) + 1. \end{aligned}$$

The *star height of a regular set* is the minimum of the star heights of the regular expressions denoting it. We call a regular expression of star height zero (i.e., with no occurrences of *) a $(\cup, \cdot)$-*expression*. ∎

The star height of a regular set can be viewed as a measure of the complexity of its looping structure.

For convenience, in writing regular expressions we frequently remove redundant parenthesis when no ambiguity can arise. However, the fully parenthesized version of these regular expressions are assumed to be processed by algorithms.

For cfg's, we use $\overset{*}{\Rightarrow}$ to mean derives (via a sequence of zero or more steps) and $\overset{*}{\Rightarrow}_L$ to mean derives via a leftmost derivation. For a sequence of productions π, we write $\Rightarrow^{\pi}$ or $\Rightarrow_L^{\pi}$ to indicate that π is the production sequence used in the derivation.

DEFINITION 1.2. For cfg $G = (N, \Sigma, P, S)$ and nonterminal A in N, $L(A) = \{w \mid w \text{ is in } \Sigma^* \text{ and } A \overset{*}{\Rightarrow} w\}$. The language generated by the grammar, $L(G)$, is defined to be $L(S)$.

G is called *linear* if the right side of each production is in $\Sigma^* \cup \Sigma^* N \Sigma^*$, and *right-linear* if the right side of each production is in $\Sigma^* \cup \Sigma^* N$. G is *reduced* if every nonterminal not equal to S can be used in some derivation of some string in $L(G)$. A language is called *linear* if it can be generated by some linear cfg. ▮

In discussing cfg's we generally make the following notational conventions. Let $G = (N, \Sigma, P, S)$ be a cfg. Then a, b, c, etc., denote elements of Σ; u, v, w, etc., denote elements of Σ^*; A, B, C, etc., denote elements of N; X, Y, Z, etc., denote elements of $N \cup \Sigma$; and α, β, γ, etc., denote elements of $(N \cup \Sigma)^*$.

DEFINITION 1.3. For grammar $G = (N, \Sigma, P, S)$, the size of grammar G, denoted $| G |$, is

$$| G | = \left(| N | + | \Sigma | + \sum_{A \to \alpha \text{ in } P} | A\alpha | \right) \cdot (\log(| N | + | \Sigma |)). \quad ▮$$

DEFINITION 1.4. A language $L \subseteq \Sigma^*$ is said to be *bounded* if and only if there exist strings $w_1, \ldots, w_k$ in Σ^* such that $L \subseteq w_1^* \cdots w_k^*$. A language that is not bounded is said to be *unbounded*.

A language $L \subseteq \Sigma^*$ is said to be *cofinite* if $\Sigma^* - L$ is finite.

A language $L \subseteq \Sigma^*$ is said to be *commutative* if and only if for all words u and v in L, $u \cdot v = v \cdot u$. ▮

DEFINITION 1.5. Let $L \subseteq \Sigma^*$ and x be in Σ^*. Then

$$x \backslash L = \{ y \mid x \cdot y \in L \} \qquad \text{and} \qquad L / x = \{ y \mid y \cdot x \in L \}. \quad ▮$$

DEFINITION 1.6. *P*, *NP*, *PSPACE*, and *CSL* are the classes of languages recognized by deterministic polynomially time bounded Tm, nondeterministic polynomially time bounded Tm, nondeterministic polynomially space bounded Tm, and nondeterministic lba, respectively. A language in P is said to be *p-decidable.* Ndtime($T(n)$) and Ndtape($T(n)$) are the classes of languages recognized by nondeterministic $T(n)$ time bounded Tm and nondeterministic $T(n)$ space bounded Tm, respectively. ▮

DEFINITION 1.7. L is *polynomially reducible* to M if there exists a function f computable by some deterministic polynomially time bounded Tm such that x is in L if and only if $f(x)$ is in M. A language is *NP-hard* if all languages in NP are polynomially reducible to it. A language is *NP-complete* if it is in NP and is NP-hard. A language is *PSPACE-hard* if all languages in PSPACE are polynomially reducible to it. A language is *PSPACE-complete* if it is in PSPACE and is PSPACE-hard. ▮

A language which is NP-hard can be considered to be "computationally intractable."

The complement of every language in P is also in P. We will use the following lemma, the first two parts of which are from [27].

LEMMA 1.8. (1) *Every language recognized by some nondeterministic polynomially space bounded Tm is also in PSPACE.*

(2) *Let M be in PSPACE. Then $\overline{M}$ is in PSPACE.*

(3) *Let M be PSPACE-complete. Then $\overline{M}$ is PSPACE-complete.*

Proof of (3). From (2), $\overline{M}$ is in PSPACE. Since M is PSPACE-complete, there exists a polynomially time bounded function g such that y is in $\overline{M}$ if and only if $g(y)$ is in M. Therefore, y is in M if and only if $g(y)$ is in $\overline{M}$. Now let L be any language in PSPACE. By the completeness of M, there exists a polynomially time bounded function f such that x is in L if and only if $f(x)$ is in M. But then x is in L if and only if $g(f(x))$ is in $\overline{M}$. Since $g \circ f$ is clearly a polynomially time bounded function, we conclude that $\overline{M}$ is PSPACE-hard. ☒

DEFINITION 1.9. A function f is *log-space computable* if there exists a deterministic Tm with a two-way read-only input tape, a one-way output tape and one two-way read–write working tape, which started with a string x on its input tape, will halt having written $f(x)$ on its output tape and having visited at most $\log(|x|)$ working tape squares. A function f is *log-lin computable* if it is log-space computable and there exists a constant $c > 0$ such that $|f(x)| \leqslant c \cdot |x|$. A Tm corresponding to a log-lin computable function is called a *log-lin transducer*. L is *log-lin reducible* to M if there exists a log-lin computable function f such that x is in L if and only if $f(x)$ is in M. A language is *CSL-hard* if all languages in CSL are log-lin reducible to it. A language is *CSL-complete* if it is in CSL and is CSL-hard. ▮

LEMMA 1.10. (1) *A language that is CSL-hard is also PSPACE-hard.*

(2) *A language that is CSL-complete is also PSPACE-complete.*

Proof. Part (1) follows from a proof technique used in [4, 32]. Let M be CSL-hard and let L in PSPACE be recognizable in space n^m. Let c be a new symbol and L' be the language

$$L' = \{xc^p \mid x \in M \text{ and } |xc^p| = |x|^m\}.$$

Then L' is in CSL and hence log-lin reducible to M. L is polynomially reducible to M by the function that first maps a string x into xc^p such that $|xc^p| = |x|^m$, and then applies to the resulting string the log-lin map from L' to M.

Part (2) follows from (1) and the fact that CSL $\subseteq$ PSPACE. ☒

The concepts of CSL-hard and CSL-complete are thus special case of PSPACE-hard

and PSPACE-complete. The CSL-hard languages require essentially linear space on any nondeterministic Tm, as indicated by the following simple fact.

PROPOSITION 1.11. *Let r be any positive rational <1. Let L be CSL-hard. Then L is not accepted by any n^r tape-bounded nondeterministic Tm.*

Proof. Ibarra [18] has shown that for all positive integers $p, q \geqslant 1$,

$$\text{Ndtape}(n^{p/q}) \supsetneq \text{Ndtape}(n^{p/(q+1)}).$$

This implies that for all positive rationals $r < 1$, $\text{Ndtape}(n^r) \subsetneq \text{CSL}$. But if L is an element of $\text{Ndtape}(n^r)$, then $\text{CSL} = \text{Ndtape}(n^r)$. ☒

Let M be a nondeterministic Tm with state set S, tape alphabet T, start state q_0 in S, and set of final accepting states $F \subseteq S$. Let $\Sigma = T \cup (S \times T) \cup \{\#\}$, where $\#$ is not in $T \cup (S \times T)$. Let $T(n)$ be a function such that $T(n) \geqslant n$ for all n.

DEFINITION 1.12. Let M be a $T(n)$ tape-bounded nondeterministic Tm as described above. Let $x = x_1 \cdots x_n$ be an input to M. An *instantaneous description* (of M on x), abbreviated i.d., is any word in $T^* \cdot (S \times T) \cdot T^*$ of length $T(n)$. The word $(q_0, x_1)\, x_2 \cdots x_n \not{b}^{T(n)-n}$, where $\not{b}$ is a special symbol denoting the blank tape square, is an *initial i.d.* Any i.d. (of M on x) that contains an accepting state is called a final i.d. We assume, without loss of generality, that M cannot make a move from a final i.d. Instantaneous description ID_{i+1} *follows* from i.d. ID_i if and only if there exists a move of M which changes ID_i into ID_{i+1} in one operation. A *valid computation* (of M on x) is a sequence of i.d.'s $\# ID_0 \# \cdots \# ID_t \#$, where ID_{i+1} follows from ID_i for $1 \leqslant i \leqslant t$, ID_0 is an initial i.d., ID_t is a final i.d., and $t \leqslant T(n)$. A string in Σ^* is an *invalid computation* (of M on x) if it is not a valid computation.

The transition function of M determines a certain function f_M from Σ^3 into 2^{Σ^3}, which we call the *i.d. transition function* and which is defined precisely in [14]. Roughly speaking, for three consecutive symbols in an i.d. delimited by $\#$'s, f_M specifies what the corresponding three symbols in a following i.d. can be. ▌

2. REGULAR EXPRESSION EQUIVALENCE

In this section, we study the complexity of various equivalence problems involving regular expressions. The regular expressions in this section are assumed to be over alphabet $\{0, 1\}$ unless we state otherwise. We use the following abbreviations for several predicates involving testing pairs of regular expressions for inequivalence:

RINEQ $= \{(R, S) \mid R$ and S are inequivalent regular expressions$\}$,

RINEQ-$(\cup, \cdot) = \{(R, S) \mid R$ and S are inequivalent $(\cup, \cdot)$-expressions$\}$,

RINEQ-(over 0) $= \{(R, S) \mid R$ and S are inequivalent regular expressions over $\{0\}\}$,

RINEQ-(one bounded) $= \{(R, S) \mid R$ and S are inequivalent regular expressions and at least one of $L(R)$ and $L(S)$ is a bounded language$\}$,

RINEQ-(both bounded) $= \{(R, S) \mid R$ and S are inequivalent regular expressions denoting bounded languages$\}$,

RINEQ-(star height k) $= \{(R, S) \mid R$ and S are inequivalent regular expressions of star height $k\}$,

RINEQ-cofinite $= \{(R, S) \mid R$ and S are inequivalent regular expressions denoting cofinite sets$\}$.

For any fixed regular language L_0, there is a set of regular expressions that do not denote L_0. Membership of an arbitrary regular expression in this set of regular expressions is a predicate which we abbreviate as follows.

$$\text{RINEQ}(L_0) = \{R \mid R \text{ is a regular expression and } L(R) \neq L_0\}.$$

The results in this section on the complexity of inequivalence problems are summarized below.

Problem	Complexity
RINEQ	CSL-complete
RINEQ-$(\cup, \cdot)$	NP-complete
RINEQ-(over 0)	NP-complete
RINEQ-(one bounded)	NP-complete
RINEQ-(both bounded)	NP-complete
For all $k \geqslant 1$, RINEQ-(star height k)	CSL-complete
RINEQ-cofinite	CSL-hard
RINEQ(L_0)	in P if L_0 is finite NP-complete if L_0 is infinite but bounded CSL-complete if L_0 is unbounded

Note that boundedness plays a major role in these results. In particular, the complexity of RINEQ(L_0) is completely characterized by whether L_0 is finite, infinite but bounded, or infinite.

First we explicitly state some well-known facts about regular expressions and regular sets.

LEMMA 2.1. (1) *Given an arbitrary regular expression R, a* ndfa *M such that $L(M) = L(R)$ can be constructed deterministically in time bounded by a polynomial in $|R|$.*

(2) *Given two arbitrary regular expressions R and S, the predicate "$L(R) \cap L(S) = \phi$" can be decided deterministically in time bounded by a polynomial in $|R| + |S|$.*

(3) *Given two inequivalent regular expressions R and S, there exists a string x such that x is in $L(R) \cap \overline{L(S)}$ or x is in $\overline{L(R)} \cap L(S)$ and such that $|x| \leqslant 2^{c(|R|+|S|)}$, where c is a constant independent of R and S.*

(4) *Given an arbitrary string x in $\{0, 1\}^*$ and an arbitrary regular expression R, membership of x in $L(R)$ can be decided deterministically in time bounded by a polynomial in $|x| + |R|$* [1].

(5) *A regular set R is infinite if and only if there exist strings u_1, u_2, u_3 in $\{0, 1\}^*$ with u_2 nonnull such that $u_1 \cdot u_2^* \cdot u_3 \subseteq R$* [13].

(6) *A regular set R over Σ is unbounded if and only if there exist strings r, s, x, and y in Σ^* and distinct a, b in Σ such that $r \cdot \{ax, by\}^* \cdot s \subseteq R$.*

(7) *Let Σ and Δ be nonempty finite alphabets, and let $h: \Sigma^* \to \Delta^*$ be a homomorphism. Then there exists a log-lin transducer that given as input a regular expression R over Σ, outputs a regular expression S such that $L(S) = h(L(R))$.*

Proof. (1) In [1, p. 322], after taking into account a variation in the definition of ndfa.

(2) Construct in polynomial time ndfa's $M_R = (K_1, \{0, 1\}, \delta_1, q_1, F_1)$ and $M_S = (K_2, \{0, 1\}, \delta_2, q_2, F_2)$ accepting $L(R)$ and $L(S)$, respectively. Construct a directed graph whose set of nodes is $K_1 \times K_2$. The graph has an edge from node (p_1, p_2) to (r_1, r_2) if and only if either r_1 is in $\delta_1(p_1, 0)$ and r_2 is in $\delta_2(p_2, 0)$, or if r_1 is in $\delta_1(p_1, 1)$ and r_2 is in $\delta_2(p_2, 1)$. Then $L(R) \cap L(S)$ is nonnull if and only if the graph has a path from node (q_1, q_2) to a node consisting of a pair of accepting states. The graph can be constructing and the path question answered in deterministic polynomial time.

(3) This follows from the fact that a ndfa with n states can be converted into a deterministic finite automaton with 2^n states that recognizes the same language. Therefore, there exist finite automata exponential in $|R| + |S|$ for recognizing $L(R) \cap \overline{L(S)}$ and $\overline{L(R)} \cap L(S)$.

(6) If: The proof in [8, pp. 142–143], that Σ^* is unbounded when Σ contains at least two elements, can be applied to show that R is unbounded when it contains $r \cdot \{ax, by\}^* \cdot s$.
Only if: From [12, Lemma 3.1].

(7) Let S be the regular expression that results from replacing each occurrence in R of each symbol a in Σ with $h(a)$. ☒

We next present without proof a result of Stockmeyer and Meyer [30].

PROPOSITION 2.2. *$\{R \mid R$ is a regular expression over $\{0\}$ and $L(R) \neq 0^*\}$ is NP-complete.*

We also note the following proposition, which is implicitly proved in [24].

PROPOSITION 2.3. *RINEQ is in CSL.*

The next proposition also is from [24]. Since we make several observations below about the regular expressions involved in the proof of this proposition, we sketch its proof.

PROPOSITION 2.4. *RINEQ($\{0, 1\}^*$) is CSL-complete.*

Proof. Observe that from Proposition 2.3, RINEQ($\{0, 1\}^*$) is in CSL.

The proof that RINEQ($\{0, 1\}^*$) is CSL-hard consists of two parts. First, for each lba M there exists a deterministic log-lin transducer that when given a string $y = a_1 \cdot \cdots \cdot a_n$ as input, outputs a regular expression β_y over a finite alphabet Σ such that $L(\beta_y) = \Sigma^*$ if and only if M does not accept y.

Let the tape alphabet, state set, accepting states, and start state of M be denoted by T, S, F, and q_0, respectively. Let $\#$ be a new symbol not an element of $T \cup (S \times T)$ and let f_M be the i.d. transition function for M. Then $\Sigma = \{\#\} \cup T \cup (S \times T)$ and $\beta_y = \beta_1 + \beta_2 + \beta_3$, where

$$\beta_1 = ((\Sigma - \{\#\}) \cup \# \cdot ((\Sigma - \{(q_0, a_1)\}) \cup (q_0, a_1) \cdot ((\Sigma - \{a_2\}) \cup a_2 \cdot (\cdots \cup a_n \cdot ((\Sigma - \{\#\})) \cdots)))) \cdot \Sigma^*;$$

$$\beta_2 = \left(\Sigma - \left(\bigcup_{q \in F} \{q\} \times T\right)\right)^*;$$

and

$$\beta_3 = \bigcup_{\sigma_1, \sigma_2, \sigma_3 \in \Sigma} \Sigma^* \cdot \sigma_1 \cdot \sigma_2 \cdot \sigma_3 \cdot \Sigma^{|y|-2} \cdot (\Sigma^3 - f_M(\sigma_1, \sigma_2, \sigma_3)) \cdot \Sigma^*.$$

The construction of β_y has an interpretation in terms of Definition 1.12 (with $T(n) = n$). $L(\beta_y)$ is the set of invalid computations of M on x. Expressions β_1, β_2, and β_3, respectively, denote strings that do not begin with the initial i.d. for y (bracketed by $\#$), that do not contain an accepting state, and that are not of the form of a sequence of i.d.'s (separated by $\#$) each of which (except for the first) follows from the previous i.d. The reader should note that the regular expression β_y has no nested occurrences of "*'s." Also, $|\Sigma^* - L(\beta_y)|$ equals the number of distinct sequences of moves of M that result in the acceptance of y.

Second, the regular expression β_y over Σ can be encoded into a regular expression $\hat{\beta}_y$ over $\{0, 1\}$ in such a way that $L(\hat{\beta}_y) = \{0, 1\}^*$ if and only if $L(\beta_y) = \Sigma^*$ and the encoding process is accomplished by a deterministic log-lin space transducer. More-

over, this encoding can be accomplished in such a way that $\hat{\beta}_y$ also has no nested occurrences of "*'s" and $|\{0, 1\}^* - L(\hat{\beta}_y)| = |\Sigma^* - L(\beta_y)|$. The details of the encoding are standard and are left to the reader. ☒

Combining Propositions 2.3 and 2.4 gives the following result from [24].

THEOREM 2.5. *RINEQ is CSL-complete.*

We now characterize the complexity of RINEQ(L_0) in terms of the complexity of L_0. Note that an algorithm to test for membership in RINEQ(L_0) has as its input a single regular expression R, and determines if $L(R) \neq L_0$. The time and space required by the algorithm are functions of the length of the input string, viz. $|R|$; these functions may involve constants that depend on L_0. The algorithm can be designed to incorporate implicitly properties of L_0. For instance, even though the algorithm does not receive a regular expression denoting L_0 as input, the algorithm can be designed to incorporate such a regular expression.

THEOREM 2.6. *Let L_0 be any fixed regular set.*

(1) *If L_0 is finite, then RINEQ(L_0) is in P.*

(2) *If L_0 is infinite but bounded, then RINEQ(L_0) is NP-complete.*

(3) *If L_0 is unbounded, then RINEQ(L_0) is CSL-complete.*

Proof. There exist regular expressions E and F such that $L(E) = L_0$ and $L(F) = \overline{L_0}$. An algorithm for RINEQ(L_0) can begin by testing if $L(R) \subseteq L_0$. By Lemma 2.1(2), this can be decided deterministically in time bounded by a polynomial in $|R|$ by deciding if $L(R) \cap L(F) = \phi$. The remainder of the proof diverges for the three parts of the theorem.

(1) $L(R)$ contains L_0 if and only if each of the finite number of strings in L_0 is also in $L(R)$. From Lemma 2.1(4), membership in $L(R)$ of each of these strings can be tested in time polynomial in the input to the algorithm, $|R|$. Therefore, "$L(R) \subseteq L_0$" and "$L(R) \supseteq L_0$" can each be tested deterministically in time bounded by a polynomial in $|R|$.

(2) Assume the algorithm has determined that $L(R) \subseteq L_0$. Since L_0 is fixed, we may assume that strings $w_1, \ldots, w_k$ are known such that $L_0 \subseteq w_1^* \cdot \cdots \cdot w_k^*$. From Lemma 2.1(3), L_0 is not a subset of $L(R)$ if and only if there exists a string in $L_0 \cap \overline{L(R)}$ of length $\leqslant 2^{c|R|}$. Here c is a constant that depends on L_0. Therefore L_0 is inequivalent to $L(R)$ if and only if there exist k integers $i_1, \ldots, i_k$ each $\leqslant 2^{c|R|}$ such that the string $x = w_1^{i_1} \cdot \cdots \cdot w_k^{i_k}$ is in $L_0 \cap \overline{L(R)}$. These k integers can be represented as k binary strings $\nu_1, \ldots, \nu_k$ each of length $\leqslant c|R|$. A nondeterministic polynomially time bounded algorithm can verify that L_0 is not a subset of $L(R)$ by working as follows.

Step 1. The algorithm nondeterministically guesses k binary strings $\nu_1, \ldots, \nu_k$ each of length $\leqslant c \cdot |R|$. These strings are to be interpreted as representing integers $i_1, \ldots, i_k$.

Step 2. The algorithm constructs an ndfa M_R such that $L(M_R) = L(R)$. From Lemma 2.1(1), this can be done deterministically in time polynomial in $|R|$.

Step 3. Let δ be the transition function of M_R. For all positive integers i and for $1 \leqslant j \leqslant k$, define a Boolean matrix $A^i_{w_j}$ whose element in row m and column n equals 1 if q_n is in $\delta(q_m, w_j{}^i)$, and equals 0 otherwise. Also, let B be the Boolean matrix defined by the Boolean matrix product

$$B = A^{i_1}_{w_1} \cdot A^{i_2}_{w_2} \cdot \cdots \cdot A^{i_k}_{w_k}$$

so that the element in row m and column n of B equals 1 if and only if q_n is in $\delta(q_m, x)$. The algorithm calculates each of $A^{i_1}_{w_1}, A^{i_2}_{w_2}, \ldots, A^{i_k}_{w_k}$ by successive squaring. The algorithm then multiplies these matrices together to compute the matrix B. By inspection of B, the algorithm verifies that x is not in $L(R)$.

Step 4. Let N be some fixed ndfa such that $L(N) = L_0$. The algorithm verifies that x is in L_0 by a method analogous to that used in Step 3.

The number of matrix multiplications in Steps 3 and 4 is $O(|\nu_1| + \cdots + |\nu_k|) \leqslant O(k \cdot c \cdot |R|)$. The dimensions of the matrices calculated in Step 3 equal the number of states of M_R. Both k and the dimensions of the matrices calculated in Step 4 depend on L_0, and are independent of R. Therefore the algorithm operates in polynomial time, and so RINEQ(L_0) is in NP.

Now we show that RINEQ(L_0) is NP-hard. Since L_0 is infinite, from Lemma 2.1(5) there exist strings u_1, u_2, and u_3 in $\{0, 1\}^*$ with u_2 nonnull such that $u_1 \cdot u_2{}^* \cdot u_3 \subseteq L_0$. Let $h: 0^* \to \{0, 1\}^*$ be the lambda-free homomorphism defined by $h(0) = u_2$. By Lemma 2.1(7), for any regular expression R over $\{0\}$, a regular expression S over $\{0, 1\}$ can be constructed deterministically in time bounded by a polynomial in $|R|$ such that

$$L(S) = u_1 \cdot h(L(R)) \cdot u_3 \cup (L_0 - u_1 \cdot u_2{}^* \cdot u_3).$$

But $L(S) = L_0$ if and only if $L(R) = 0^*$. Therefore, from Proposition 2.2, RINEQ(L_0) is NP-hard.

(3) RINEQ(L_0) is easily seen to be in CSL. The lba corresponding to Proposition 2.3 can be supplied as input (R, E) where E is a regular expression such that $L(E) = L_0$.

We now show that RINEQ(L_0) is CSL-hard. Since L_0 is unbounded, by Lemma 2.1(6) there exist strings r, s, x, and y in $\{0, 1\}^*$ such that $r \cdot \{0x, 1y\}^* \cdot s \subseteq L_0$. Let

h: $\{0, 1\}^* \to \{0, 1\}^*$ be the lambda-free homomorphism defined by $h(0) = 0x$ and $h(1) = 1y$. By Lemma 2.1(7) there exists a log-lin transducer that given as input any regular expression R, produces as output a regular expression S such that

$$L(S) = r \cdot h(L(R)) \cdot s \cup (L_0 \cap \overline{r \cdot \{0x, 1y\}^* \cdot s}).$$

But $L(S) = L_0$ if and only if $L(R) = \{0, 1\}^*$. Thus, by Proposition 2.4 and the transitivity of log-lin reducibility, RINEQ(L_0) is CSL-hard. ∎

The proof in (2) above that RINEQ(L_0) is in NP is an extension of the proof in [30] that "$L(R) \neq 0^*$" is in NP.

Next we consider the effect on the complexity of the regular expression equivalence problem of cofiniteness and fixed star height, two properties of the regular sets extensively studied in the literature (see [6, 22, 23]). We note that part (1) of the following theorem has been stated in [15, 30].

THEOREM 2.7. (1) *RINEQ-$(\cup, \cdot)$ is NP-complete.*

(2) *For all $k \geqslant 1$, RINEQ-(star height k) is CSL-complete.*

(3) *RINEQ-cofinite is CSL-hard.*

Proof. (1) Cook [7] has shown that the set of nontautological D_3 Boolean forms, i.e., nontautological Boolean forms in disjunctive normal form with at most three literals per clause, is NP-complete. Let $f = c_1 \vee c_2 \vee \cdots \vee c_m$ be any arbitrary D_3-Boolean form. Then each clause c_i is the Boolean product of at most three literals. Let the number of variables appearing in f be n.

Without loss of generality, we assume that no variable occurs both complemented and uncomplemented in the same clause. For each c_i let $\beta_i = \beta_{i1} \cdot \beta_{i2} \cdot \cdots \cdot \beta_{in}$, where

$$\begin{aligned} \beta_{ij} &= (0 \cup 1), && \text{if } x_j \text{ and } \bar{x}_j \text{ are not literals in } c_i, \\ &= (0), && \text{if } \bar{x}_j \text{ is a literal in } c_i, \\ &= (1), && \text{if } x_j \text{ is a literal in } c_i. \end{aligned}$$

Let $\beta = (\beta_1) \cup (\beta_2) \cup \cdots \cup (\beta_m)$. Then $L(\beta) = \{0, 1\}^n$ if and only if f is a tautology. Clearly β and a $(\cup, \cdot)$-expression denoting $\{0, 1\}^n$ can be constructed deterministically in time bounded by a polynomial in $|f|$. Thus, $\{(R, S) \mid R$ and S are inequivalent $(\cup, \cdot)$ expressions$\}$ is NP-hard. But this set is easily seen to be in NP.

(2) The regular expressions constructed in the proof of Proposition 2.4 are all of star height equal to 1. For any $k \geqslant 1$ it is easy to modify $\hat{\beta}_y$ of the proof of Proposition 2.4 in such a way that the resulting regular expression has star height equal to k and the language denoted by the resulting expression equals $L(\hat{\beta}_y)$.

(3) Every context-sensitive language is accepted by some lba M such that for any input y all computations of M on y terminate in time $\leqslant 2^{c|y|}$, where c is a constant independent of y. Therefore, there are only a finite number of distinct sequences of moves of M that result in the acceptance of y. Consequently, $L(\beta_y)$ of the proof of Proposition 2.4 is a cofinite set. ☒

Note that in the proof of Theorem 2.7(3), the regular expressions involved are known to denote cofinite sets. Thus, testing for inequivalence is hard even when one is told that the input regular expressions denote cofinite sets.

Next we consider the inequivalence problem when bounded languages are involved. Note the following simple relationship between star height and boundedness.

PROPOSITION 2.8. *The star height of any infinite but bounded regular set is equal to* 1.

Proof. This result follows trivially from the result in [9] that every bounded regular set is a member of the smallest family of sets which contain all finite sets, all sets of the form w^* (w a finite length string), and which is closed with respect to finite union and finite product. ☒

We next use the following lemma, from [8], relating commutativity and boundedness for the regular sets.

LEMMA 2.9. (1) *A language L is commutative if and only if there exists a word w such that $L \subseteq w^*$.*

(2) *A language of the form A^* is bounded if and only if A is commutative.*

Proof. For a proof of (1), see [8, p. 169]. Statement (2) follows from the fact in [8, p. 171] that, letting u, v be strings over $\{0, 1\}$, $u \cdot v \neq v \cdot u$ implies $\{u, v\}^*$ is not bounded. ☒

THEOREM 2.10. *There exists a deterministic polynomially time bounded Tm M such that M, when given a regular expression R as input, determines if $L(R)$ is bounded and if so outputs strings $w_1, w_2, \ldots, w_k$ such that $L(R) \subseteq w_1^* \cdot w_2^* \cdot \cdots \cdot w_k^*$ and such that $|w_1| + |w_2| + \cdots + |w_k| \leqslant |R|$.*

Proof. The proof is by induction on the depth d of nesting of regular operators in the regular expression R. If $d = 0$, then $L(R) = \phi$, λ, 0, or 1; and $L(R) \subseteq 0^*$, 0^*, 0^*, or 1^*, respectively. Assume that the theorem holds for all regular expressions of depth $\leqslant k$. Let $d = k + 1$. Then $R = (A) \cup (B)$, $(A) \cdot (B)$, or $(A)^*$.

Case 1. If $R = (A) \cup (B)$, then $L(R)$ is bounded if and only if both $L(A)$ and $L(B)$ are bounded. If $L(A) \subseteq x_1^* \cdot \cdots \cdot x_m^*$ and $L(B) \subseteq y_1^* \cdot \cdots \cdot y_n^*$ with

$$|x_1| + \cdots + |x_m| \leqslant |A| \qquad \text{and} \qquad |y_1| + \cdots + |y_n| \leqslant |B|,$$

then $L(R) \subseteq x_1^* \cdot \cdots \cdot x_m^* \cdot y_1^* \cdot \cdots \cdot y_n^*$ and

$$| x_1 | + \cdots + | x_m | + | y_1 | + \cdots + | y_n | \leqslant | R |.$$

Case 2. If $R = (A) \cdot (B)$, then $L(R)$ is bounded if and only if $L(A)$ is empty, $L(B)$ is empty, or both $L(A)$ and $L(B)$ are bounded. But the emptiness problem for regular expressions is trivially p-decidable. If neither $L(A)$ nor $L(B)$ is empty, then the proof is similar to that of Case 1.

Case 3. If $R = (A)^*$, by Lemma 2.9(2) $L(R)$ is bounded if and only if $L(A)$ is commutative. If $L(A) = \phi$ or λ, then $L(A)$ is trivially commutative. Otherwise a nonempty string x in $L(A)$ can be found deterministically in time bounded by a polynomial in $| A |$ such that $| x | \leqslant | A |$. But by Lemma 2.9(1), $L(A)$ is commutative if and only if there exists a string w such that $L(A) \subseteq w^*$. But any such w must have the property that $x = w^n$ for some nonnegative integer n. M finds each of the prefixes w of x, such that $x = w^n$ for some n. For each such w, M tests whether $L(A) \cap \overline{w^*} = \phi$. From Lemma 2.1(2), this test can be done deterministically in time bounded by a polynomial in $| A |$. ☒

We can now state our results on inequivalence of bounded languages.

THEOREM 2.11. (1) *RINEQ-(one bounded) is NP-complete.*

(2) *RINEQ-(both bounded) is NP-complete.*

(3) *RINEQ-(over 0) is NP-complete.*

Proof. By Proposition 2.2, these languages are NP-hard. That they are in NP can be seen by combining Theorem 2.10 and the proof of Theorem 2.6(2). ☒

We note that all of our results about equivalence apply equally to containment. For example, the following analogs of Theorems 2.6 and 2.11 hold.

THEOREM 2.12. *Let L_0 be any fixed regular set.*

(a) *If L_0 is finite, then $\{R \mid R$ is a regular expression and $L(R) \supseteq L_0\}$ is in P.*

(b) *If L_0 is infinite but bounded, then $\{R \mid R$ is a regular expression and $\neg[L(R) \supseteq L_0]\}$ is NP-complete.*

(c) *If L_0 is unbounded, then $\{R \mid R$ is a regular expression and $\neg[L(R) \supseteq L_0]\}$ is CSL-complete.*

THEOREM 2.13. *$\{(R, S) \mid R$ and S are regular expressions, at least one of $L(R)$ and $L(S)$ is bounded, and $\neg[L(R) \supseteq L(S)]\}$ is NP-complete.*

The proofs are almost identical to those of Theorems 2.6 and 2.11 and are left to the reader.

The results in this section and [16] suggest that the reason there has been little success in finding "canonical" forms for arbitrary regular expressions may be due to the difficulty of testing for equivalence. Moreover, our results here and in [16] show that the equivalence problem for most of the subclasses of the regular sets or regular expressions studied in the literature are as hard as the equivalence problem for arbitrary regular expressions. However, the equivalence problem may be simpler for unambiguous regular expressions, i.e., regular expressions for which every string in the denoted language is described by the regular expression in only one way (see [5]). We note that all of the proofs in this section involve highly ambiguous regular expressions. Moreover, we are unable to show anything about the complexity of the equivalence problem for unambiguous regular expressions. We feel that the unambiguous regular expressions are a reasonable candidate for a nontrivial subclass of the regular expressions with a deterministic polynomially time bounded equivalence problem.

OPEN PROBLEM 1. Is the equivalence problem for the unambiguous regular expressions in P, NP-complete, etc.?

3. CONTEXT-FREE LANGUAGE EQUIVALENCE

In this section, we concentrate on the complexity of testing a cfg to determine if it generates some fixed cfl. For any fixed cfl L_0 we define

$$\text{GINEQ}(L_0) = \{G \mid G \text{ is a cfg and } L(G) \neq L_0\}.$$

Also, for any fixed linear cfl L_0, we define

$$\text{LGINEQ}(L_0) = \{G \mid G \text{ is a linear cfg and } L(G) \neq L_0\}.$$

The main result of this section is that:

(1) If L_0 is finite, then GINEQ(L_0) is in P.

(2) If L_0 is infinite, then GINEQ(L_0) is NP-hard.

(3) If L_0 is unbounded, then GINEQ(L_0) is PSPACE-hard.

(4) If L_0 contains an unbounded regular subset, then GINEQ(L_0) is not recursive.

This result is analogous to Theorem 2.6, except that (2) and (3) above give only lower bounds on complexity. Obtaining upper bounds is an open problem.

Identical results hold for LGINEQ(L_0). Moreover, for the linear cfg's, these lower bounds are "tight" in the sense that they are exact for certain fixed languages. For

the infinite but bounded linear language 0^*, LGINEQ(0^*) is in NP. For the unbounded linear language $L = \{w \# w^{\text{rev}} \mid w \text{ in } \{0, 1\}^*\}$, LGINEQ($L$) is in PSPACE.

We also show that if a cfg L_0 contains an infinite regular subset, then GINEQ(L_0) is as hard as GINEQ(0^*).

We now explicitly state some well-known facts about cfg's. We assume in this section that the grammars involved have some finite nonnull terminal set Σ.

LEMMA 3.1. (1) *Given an arbitrary regular expression R, a right-linear* cfg G *can be found deterministically in time bounded by a polynomial in* $|R|$ *such that* $L(G) = L(R)$ [2].

(2) *Given an arbitrary* cfg G *and an arbitrary* ndfa M, *a* cfg G' *can be found deterministically in time bounded by a polynomial in* $|G| + |M|$ *such that* $L(G') = L(G) \cap L(M)$. *Furthermore, if G is linear, then so is G'.*

(3) *There exists a deterministic polynomially time bounded Tm that decides whether an arbitrary* cfg *generates the empty set* [2].

(4) *There exists a deterministic polynomially time bounded Tm that, given an arbitrary* cfg G *and an arbitrary string w, decides if w is in $L(G)$.*

(5) *Let Σ and Δ be nonempty finite alphabets, and let $h: \Sigma^* \to \Delta^*$ be a homomorphism. Then there exists a log-lin transducer that given as input a* cfg G *with terminal set Σ, outputs a* cfg G' *with terminal set Δ such that $L(G') = h(L(G))$.*

(6) *Given an arbitrary* cfg G, *an equivalent reduced* cfg G' *can be found deterministically in time bounded by a polynomial in G. Furthermore, if G is linear, then so is G'* [2].

(7) *Let G be a* cfg *for which $L(G)$ is infinite. Then there exist strings u, v, w, x, y in Σ^* with $v \cdot x$ nonnull, such that*

$$\{u \cdot v^i \cdot w \cdot x^i \cdot y \mid i \geqslant 0\} \subseteq L(G) \qquad [8].$$

(8) *GINEQ($\{0, 1\}^*$) and LGINEQ($\{0, 1\}^*$) are both not recursive.*

(9) *Let G be a reduced* cfg *with $L(G) \neq \phi$. Then $L(G)$ is bounded if and only if for all nonterminals A of G, both $L_A(G) = \{u \mid A \overset{*}{\Rightarrow} uAv\}$ and $R_A(G) = \{v \mid A \overset{*}{\Rightarrow} uAv\}$ are commutative* [8].

Proof. (2) In polynomial time, modify G so that the maximum length of the right-hand side of any production is 2. Then obtain G' by using the method of [8, p. 88], with the extension that for each production $A \to \lambda$ in G and each state p in M, G' has the production

$$(p, A, p) \to \lambda.$$

(4) If $w = \lambda$, check if w is in $L(G)$ by testing for emptiness the grammar obtained from G by deleting all productions with a terminal in the right-hand side.

If $w \neq \lambda$, then modify G to eliminate lambda-productions, as follows. Let $G = (N, \Sigma, P, S)$. Find the set N_λ of nonterminals whose generated languages contain λ. Consider the production

$$A \to \alpha_0 B_1 \alpha_1 B_2 \cdots B_k \alpha_k , \qquad k \geqslant 0,$$

where each B_i is in N_λ and each α_i is in $((N - N_\lambda) \cup \Sigma)^*$. For this production, new nonterminals $X_1, X_2, \ldots, X_k$ are introduced and the production is replaced by the set of productions

$$\begin{aligned}
&A \to \alpha_0 \alpha_1 \cdots \alpha_k && \text{if } \alpha_0 \alpha_1 \cdots \alpha_k \neq \lambda,\\
&A \to \alpha_0 X_1 , &&\\
&X_i \to \alpha_i X_{i+1} , && 1 \leqslant i < k,\\
&X_i \to B_i \alpha_i X_{i+1} , && 1 \leqslant i < k,\\
&X_i \to B_i \alpha_i \alpha_{i+1} \cdots \alpha_k && 1 \leqslant i \leqslant k
\end{aligned}$$

We note that a polynomial time procedure for eliminating lambda-productions has been independently obtained by Yehudai [31].

The grammar is then converted to Chomsky normal form by the methods of [2], and membership of w in the language generated by the grammar is determined by the Cocke–Younger–Kasami algorithm [2].

(5) Let G' be the cfg that results from replacing each occurrence in G of each symbol a in Σ with $h(a)$.

(8) GINEQ($\{0, 1\}^*$) is shown not to be recursive in [13]. The languages involved in that proof are all linear, and so the same construction also proves that LGINEQ($\{0, 1\}^*$) is not recursive. ☒

We now prove a series of technical lemmas. We begin with a cfl analog of Lemma 2.1(6). A result closely related to Proposition 3.2 appears in [12].

PROPOSITION 3.2. *A* cfl L *is unbounded if and only if there exist strings* w, x, *and* y *in* Σ^* *and distinct* a, b *in* Σ *such that every string in* $w \cdot \{ax, by\}^*$ *is a prefix of some string in* L, *or every string in* $\{xa, yb\}^* \cdot w$ *is a suffix of some string in* L.

Proof. Only if: Assume L is unbounded. From Lemma 3.1(9), for some reduced cfg G with start symbol S generating L, there is a nonterminal A such that $L_A(G) = \{u \mid A \overset{*}{\Rightarrow} uAv\}$ or $R_A(G) = \{v \mid A \overset{*}{\Rightarrow} uAv\}$ is noncommutative. Assume $L_A(G)$ is noncommutative. Then there exist strings u_1, u_2 in $L_A(G)$ such that $u_1 \cdot u_2 \neq u_2 \cdot u_1$. By Lemma 2.9(2), $\{u_1, u_2\}^*$ is an unbounded regular set. Thus by Lemma 2.1(6) there exist strings r, s, x, and y in Σ^* and distinct a, b in Σ such that $r \cdot \{ax, by\}^* \cdot s \subseteq \{u_1, u_2\}^* \subseteq L_A(G)$. Since G is reduced, there exist strings z_1, z_2 in Σ^* such that

$S \overset{*}{\Rightarrow} z_1 A z_2$. Let $w = z_1 \cdot r$. Then every string in $w \cdot \{ax, by\}^*$ is a prefix of some string in L. If $R_A(G)$ is noncommutative, the proof is similar.

If: This follows from an application of the proof in [8, pp. 142–143] that if Σ contains at least two elements, then Σ^* is unbounded. ☒

DEFINITION 3.3. A cfl L_0 is said to be *deciperable* if and only if for all strings u, v, w, x and y in Σ^* such that $v \cdot x \neq \lambda$ and such that $\{u \cdot v^i \cdot w \cdot x^i \cdot y \mid i \geqslant 0\} \subseteq L_0$,

$$u \cdot v^i \cdot w \cdot x^i \cdot y = u \cdot v^j \cdot w \cdot x^k \cdot y \qquad \text{implies} \quad i = j = k$$

for all integers $i, j, k \geqslant 0$. ▮

We now prove several lemmas, culminating in the result that every infinite cfl either has an infinite regular subset, or is decipherable.

LEMMA 3.4. *Consider x, y, z in Σ^* with x nonnull. If $x \cdot y = y \cdot z$, then there exist strings r, s in Σ^* and integer $p \geqslant 0$ such that $x = r \cdot s$, $z = s \cdot r$, and $y = (r \cdot s)^p \cdot r$.*

Proof. The proof is by induction on the length of $| x \cdot y |$. If $| x \cdot y | = 1$, then since x is nonnull, $x = z$ and $y = \lambda$. Let $r = \lambda$, $s = x$, and $p = 0$.

Let $| x \cdot y | = k > 1$ and assume that the lemma holds for all strings x', y', z' in Σ^* with $x' \cdot y' = y' \cdot z'$ and $| x' \cdot y' | < k$.

Case 1. If $| x | = | y |$, then $x = y = z$. Let $r = x$, $s = \lambda$, and $p = 0$.

Case 2. If $| x | > | y |$, then $x = y \cdot d$ for some d in Σ^+. But $x \cdot y = y \cdot z$ implies $y \cdot d \cdot y = y \cdot z$, which implies $z = d \cdot y$. Let $r = y$, $s = d$, and $p = 0$.

Case 3. If $| x | < | y |$, then $y = x \cdot d$ for some string d in Σ^+. But $x \cdot y = y \cdot z$ implies $x \cdot x \cdot d = x \cdot d \cdot z$, which implies $x \cdot d = d \cdot z$. Since x is nonnull by assumption, $| x \cdot d | < | x \cdot y |$. Thus by the inductive hypothesis, there exist strings r, s in Σ^* and integer $p' \geqslant 0$ such that $x = r \cdot s$, $z = s \cdot r$, and $d = (r \cdot s)^{p'} \cdot r$. Let $y = (r \cdot s)^{p'+1} \cdot r$. ☒

LEMMA 3.5. *Consider u, v, w, x, y in Σ^* with both v and x nonnull. If there exist integers $i, j, k \geqslant 0$ such that $u \cdot v^i \cdot w \cdot x^i \cdot y = u \cdot v^j \cdot w \cdot x^k \cdot y$ with $i \neq j$ or $i \neq k$, then there exist strings r, s in Σ^* and integers $m, n, p \geqslant 0$ such that $v^m = r \cdot s$, $x^n = s \cdot r$, and $w = (r \cdot s)^p \cdot r$.*

Proof. Assume $u \cdot v^i \cdot w \cdot x^i \cdot y = u \cdot v^j \cdot w \cdot x^k \cdot y$. Then $v^i \cdot w \cdot x^i = v^j \cdot w \cdot x^k$. Since both v and x are nonnull, $i \neq j$ if and only if $i \neq k$.

Case 1. If $i > j$, then $i = j + m$ with $m > 0$. Then $k > i$. Thus $k = i + n$ with $n > 0$. Thus $v^{j+m} \cdot w \cdot x^i = v^i \cdot w \cdot x^{i+n}$ implies $v^m \cdot w = w \cdot x^n$, and the conclusion follows from Lemma 3.4.

Case 2. If $j > i$, then $j = i + m$ with $m > 0$ and $k < i$. Thus $i = k + n$ with $n > 0$. Thus $v^i \cdot w \cdot x^{k+n} = v^{i+m} \cdot w \cdot x^k$ implies $v^m \cdot w = w \cdot x^n$, and the conclusion follows from Lemma 3.4. ☒

LEMMA 3.6. *Consider u, v, w, x, y in Σ^* with both v and x nonnull. If there exist integers $i, j, k \geqslant 0$ such that $u \cdot v^i \cdot w \cdot x^i \cdot y = u \cdot v^j \cdot w \cdot x^k \cdot y$ with $i \neq j$ or $i \neq k$, then $\{u \cdot v^l \cdot w \cdot x^l \cdot y \mid l \geqslant 0\}$ is regular.*

Proof. Suppose there exist integers $i, j, k > 0$ such that $u \cdot v^i \cdot w \cdot x^i \cdot y = u \cdot v^j \cdot w \cdot x^k \cdot y$ with $i \neq j$ or $i \neq k$. From Lemma 3.5 there exist integers $m, n, p \geqslant 0$ and strings r, s in Σ^* such that $v^m = r \cdot s$, $x^n = s \cdot r$, and $w = (r \cdot s)^p \cdot r$. Thus,

$$\{u \cdot v^l \cdot w \cdot x^l \cdot y \mid l \geqslant 0\} = \bigcup_{0 \leqslant a < mn} \{u \cdot v^{a+b \cdot mn} \cdot w \cdot x^{a+b \cdot mn} \cdot y \mid b \geqslant 0\}$$

$$= \bigcup_{0 \leqslant a < mn} \{u \cdot v^a \cdot (r \cdot s)^{bn} \cdot (r \cdot s)^p \cdot r \cdot (s \cdot r)^{bm} \cdot x^a \cdot y \mid b \geqslant 0\}$$

$$= \bigcup_{0 \leqslant a < mn} \{u \cdot v^a \cdot (r \cdot s)^p \cdot ((r \cdot s)^{m+n})^b \cdot r \cdot x^a \cdot y \mid b \geqslant 0\}$$

$$= \bigcup_{0 \leqslant a < mn} u \cdot v^a \cdot (r \cdot s)^p \cdot ((r \cdot s)^{m+n})^* \cdot r \cdot x^a \cdot y,$$

which is a finite union of regular sets and hence is regular. ☒

THEOREM 3.7. *Let L be any infinite* cfl. *Then either L has an infinite regular subset, or L is decipherable.*

Proof. Suppose L is not decipherable. Then there exist strings u, v, w, x and y in Σ^* with $v \cdot x \neq \lambda$ such that $\{u \cdot v^l \cdot w \cdot x^l \cdot y \mid l \geqslant 0\} \subseteq L_0$, and there exist integers $i, j, k \geqslant 0$ such that $u \cdot v^i \cdot w \cdot x^i \cdot y = u \cdot v^j \cdot w \cdot x^k \cdot y$ with $i \neq j$ or $j \neq k$. If either $v = \lambda$ or $x = \lambda$, then $\{u \cdot v^l \cdot w \cdot x^l \cdot y \mid l \geqslant 0\}$ is an infinite regular set and L has an infinite regular subset. If both v and x are nonnull, then by Lemma 3.6, $\{u \cdot v^l \cdot w \cdot x^l \cdot y \mid l \geqslant 0\}$ is an infinite regular set and L has an infinite regular subset. ☒

We now prove a result on the complexity of GINEQ(L_0).

THEOREM 3.8. *Let L_0 be any* cfl *that contains an infinite regular subset. Then GINEQ(0*) is polynomially reducible to GINEQ(L_0).*

Proof. Since L_0 has an infinite regular subset, there exist strings w_1, w_2, and w_3 in Σ^* with w_2 nonnull such that $w_1 \cdot w_2^* \cdot w_3 \subseteq L_0$. Thus, L_0 is the disjoint union of $w_1 \cdot w_2^* \cdot w_3$ and $L_1 = L_0 \cap (\Sigma^* - w_1 \cdot w_2^* \cdot w_3)$. Let $h: 0^* \to \Sigma^*$ be the lambda-

free homomorphism defined by $h(0) = w_2$. Since w_1, w_2, and w_3 are fixed strings and L_1 is a fixed cfl, by Lemma 3.1(5) given any cfg G, a cfg G' can be found deterministically in time bounded by a polynomial in $| G |$ such that

$$L(G') = w_1 \cdot h(L(G)) \cdot w_3 \cup L_1 .$$

But $L(G') = L_0$ if and only if $L(G) = 0^*$. ☒

It follows immediately from Proposition 2.2 and Lemma 3.1(1) that GINEQ(0*) is NP-hard. However, we do not know if it is an element of either NP or PSPACE. If GINEQ(0*) is a member of neither of these classes, then by Theorem 3.8, neither is GINEQ(L_0) whenever L_0 has an infinite regular subset. The proof in [8] that every cfl contained in 0^* is regular implies that GINEQ(0*) is decidable, but does not directly address its complexity.

OPEN PROBLEM 2. Is GINEQ(0*) an element of either NP or PSPACE?

We now prove the main result of this section.

THEOREM 3.9. *Let L_0 be an arbitrary* cfl.

(1) *If L_0 is finite, then GINEQ(L_0) is in P.*

(2) *If L_0 is infinite, then GINEQ(L_0) is NP-hard.*

(3) *If L_0 is unbounded, then GINEQ(L_0) is PSPACE-hard.*

(4) *If L_0 contains an unbounded regular subset, then GINEQ(L_0) is not recursive.*

Proof. Throughout the proof the reader is reminded that L_0 is considered to be a *fixed* cfl. (1) There exists an ndfa M such that $L(M) = \overline{L_0}$. If $L(G) \cap L(M)$ is not empty, then $L(G) \neq L_0$. By Lemma 3.1(2) and (3), this can be determined deterministically in time bounded by a polynomial in $| G |$. If $L(G) \cap L(M) = \phi$, then $L(G) \subseteq L_0$. Thus, to verify that $L(G) = L_0$ it suffices to check that all strings w in L_0 are also elements of $L(G)$. Since there are only a finite number of such strings, Lemma 3.1(4) implies that this can be accomplished deterministically in time bounded by a polynomial in $| G |$.

(2) If L_0 has an infinite regular subset, then Theorem 3.8, Lemma 3.1(1) and Proposition 2.2 imply that GINEQ(L_0) is NP-hard. If L_0 has no infinite regular subsets, then by Proposition 3.7 it is decipherable. Since L_0 is an infinite cfl, from Lemma 3.1(7) there exist strings u, v, w, x, y in Σ^* with $v \cdot x$ nonnull, such that $\{u \cdot v^i \cdot w \cdot x^i \cdot y \mid i \geq 0\} \subseteq L_0$. Since L_0 contains no infinite regular subsets, both v and x are nonnull.

Let $h: 0^* \to \Sigma^*$ be the lambda-free homomorphism defined by $h(0) = v$. For any regular expression R over $\{0\}$, by Lemma 2.1(1) and (7), a ndfa accepting the language $u \cdot h(L(R)) \cdot w \cdot x^* \cdot y$ can be found deterministically in time bounded

by a polynomial in $|R|$. Therefore, from Lemma 3.1(2), a cfg G can be found deterministically in time bounded by a polynomial in $|R|$, such that

$$L(G) = (u \cdot h(L(R)) \cdot w \cdot x^* \cdot y \cap L_0) \cup (L_0 \cap (\Sigma^* - u \cdot v^* \cdot w \cdot x^* \cdot y)).$$

We now show that $L(G) = L_0$ if and only if $L(R) = 0^*$.

If $L(R) = 0^*$, then $h(L(R)) = v^*$ and $L(G)$ clearly equals L_0. If $L(R) \subsetneq 0^*$, then there exists a nonnegative integer i such that 0^i is not in $L(R)$. Let $z = u \cdot v^i \cdot w \cdot x^i \cdot y$ and suppose z is in $L(G)$. Then since z is clearly not an element of

$$L_0 \cap (\Sigma^* - u \cdot v^* \cdot w \cdot x^* \cdot y),$$

z is in $u \cdot h(L(R)) \cdot w \cdot x^* \cdot y$. Thus there exist nonnegative integers j and k with $i \neq j$ such that $z = u \cdot v^i \cdot w \cdot x^i \cdot y = u \cdot v^j \cdot w \cdot x^k \cdot y$. But this contradicts the decipherability of L_0. Thus z is not in $L(G)$ and $L(G) \neq L_0$. Therefore, RINEQ$\{R \mid R$ is a regular expression over 0 and $L(G) \neq 0^*\}$ is polynomially reducible to GINEQ(L_0). From Proposition 2.2, GINEQ(L_0) is therefore NP-hard.

(3) By Proposition 3.2, we assume there exist strings w, r, s in Σ^* and distinct a, b in Σ such that every string in $w \cdot \{ax, by\}^*$ is a prefix of some string in L_0. An analogous proof holds for the suffix case. Let $h: \{0, 1\}^* \to \Sigma^*$ be the lambda-free homomorphism defined by $h(0) = axax$ and $h(1) = axby$. By Lemma 2.1(1) and (7) and Lemma 3.1(2), for every regular expression R, a cfg G can be found deterministically in time bounded by a polynomial in $|R|$ such that

$$L(G) = (w \cdot h(L(R)) \cdot byax \cdot \Sigma^* \cap L_0) \cup (L_0 \cap (\Sigma^* - w \cdot \{axax, axby\}^* \cdot byax \cdot \Sigma^*)).$$

We now show that $L(G) = L_0$ if and only if $L(R) = \{0, 1\}^*$.

If $L(R) = \{0, 1\}^*$, then $h(L(R)) = \{axax, axby\}^*$ and $L(G)$ clearly equals L_0. If $L(R) \subsetneq \{0, 1\}^*$, then there exists z in $\{0, 1\}^* - L(R)$. Suppose $w \cdot h(z) \cdot byax$ is a prefix of some string in $L(G)$. Then there exists a string v in Σ^* such that $w \cdot h(z) \cdot byax \cdot v$ is in $L(G)$. Since $w \cdot h(z) \cdot byax \cdot v$ is not in

$$L_0 \cap (\Sigma^* - w \cdot \{axax, axby\}^* \cdot byax \cdot \Sigma^*),$$

$w \cdot h(z) \cdot byax \cdot v$ is in

$$w \cdot h(L(R)) \cdot byax \cdot \Sigma^* \cap L_0.$$

But this implies $h(z)$ is in $h(L(R))$, which implies z is in $L(R)$. Thus $w \cdot h(z) \cdot byax$ is not a prefix of any string in $L(G)$; and $L(G) \neq L_0$. Therefore, RINEQ($\{0, 1\}^*$) is polynomially reducible to GINEQ(L_0). Hence from Proposition 2.4, GINEQ(L_0) is PSPACE-hard.

(4) Since L_0 contains an unbounded regular subset, by Lemma 2.1(6) there exist strings r, s, x, and y in Σ^* and distinct a, b in Σ such that $r \cdot \{ax, by\}^* \cdot s \subseteq L_0$. Thus L_0 can be expressed as the disjoint union

$$L_0 = r \cdot \{ax, by\}^* \cdot s \cup (L_0 \cap (\Sigma^* - r \cdot \{ax, by\}^* \cdot s)).$$

Let $h: \{0, 1\}^* \to \Sigma^*$ be the lambda-free homomorphism defined by $h(0) = ax$ and $h(1) = by$. By Lemma 3.1(5) and (2), given any cfg G with terminal alphabet $\{0, 1\}$, a cfg G' can be found effectively such that

$$L(G') = r \cdot h(L(G)) \cdot s \cup (L_0 \cap (\Sigma^* - r \cdot \{ax, by\}^* \cdot s)).$$

But $L(G') = L_0$ if and only if $L(G) = \{0, 1\}^*$, which from Lemma 3.1(8) is undecidable. ☒

Theorem 3.9(4) extends Hopcroft's result in [12] that GINEQ(L_0) is undecidable if L_0 is any unbounded regular set. We also note the result [8] that it is decidable to test equivalence of a pair of cfg's when one of them generates a bounded language.

The proofs of Theorems 3.8 and 3.9 go through for several proper subclasses of the cfl's as well. These subclasses include the linear cfl's, the metalinear cfl's [8], and the languages accepted by nondeterministic 1-reversal bounded 1-counter machines (see [3]). In particular, the following two theorems have proofs identical to those of Theorems 3.8 and 3.9.

THEOREM 3.10. *Let L_0 be any linear* cfl *that contains an infinite regular subset. Then LGINEQ(0*) is polynomially reducible to LGINEQ(L_0).*

THEOREM 3.11. *Let L_0 be an arbitrary linear* cfl.

(1) *If L_0 is finite, then LGINEQ(L_0) is in P.*

(2) *If L_0 is infinite, then LGINEQ(L_0) is NP-hard.*

(3) *If L_0 is unbounded, then LGINEQ(L_0) is PSPACE-hard.*

(4) *If L_0 contains an unbounded regular subset, then LGINEQ(L_0) is undecidable.*

Moreover, for linear cfl's the "lower bounds" in Theorem 3.11 are "tight" as discussed above.

PROPOSITION 3.12. *LGINEQ(0*) is in NP.*

Proof. For a linear cfg G with terminal alphabet $\{0\}$, an equivalent right-linear cfg G' can be found by replacing every production $A \to uBv$ of G, where A and B are nonterminals, by $A \to u \cdot vB$. Moreover, given any right-linear grammar G, a ndfa M such that $L(M) = L(G)$ can be found deterministically in time bounded

by a polynomial in $|G|$. But $L(M) \neq 0^*$ can be verified nondeterministically in time bounded by a polynomial in $|M|$ as in the proof of Theorem 2.6(2). ☒

Hopcroft [12] has presented an algorithm for deciding GINEQ($\{w \# w^{\text{rev}} \mid w$ in $\{0, 1\}^*\}$) for arbitrary cfg's G. This algorithm can be modified to show that GINEQ(L_0) is decidable for a variety of different unbounded cfl's L_0. We do not know if any equivalent algorithm is executable by some polynomially space-bounded Tm. However, such a polynomially space-bounded algorithm does exist for the linear cfg's.

PROPOSITION 3.13. *Let* $L_0 = \{w \# w \mid w$ *in* $\{0, 1\}^*\}$. *Then LGINEQ*(L_0) *is in PSPACE.*

Proof. The polynomially space-bounded algorithm that tests a given linear cfg $G = (N_1, \Sigma, P_1, S)$ for membership in LGINEQ(L_0) is presented below.

Step 1. Determine whether λ is in $L(G)$. If so, G is in LGINEQ(L_0) and the algorithm halts.

Step 2. Verify that $L(G)$ is infinite. If not, G is in LGINEQ(L_0).

Step 3. Rewrite G if necessary so that it is reduced, has no lambda-productions, and has no production whose right side is a single nonterminal. Call the resulting linear cfg $G_2 = (N_2, \Sigma, P_2, S)$.

Step 4. Verify that $L(G_2) \cap (\Sigma^* - \{0, 1\}^* \cdot \# \cdot \{0, 1\}^*)$ is empty. Otherwise G is in LGINEQ(L_0).

Comment. If $L(G) = L_0$, then for each A in N_2, either $L(A) \subseteq \{0, 1\}^* \cdot \# \cdot \{0, 1\}^*$ or $L(A)$ equals a unique terminal string.

Step 5. For each A in N_2 verify that $L(A) \subseteq \{0, 1\}^* \cdot \# \cdot \{0, 1\}^*$ or that $|L(A)| = 1$. Otherwise G is in LGINEQ(L_0).

Step 6. For each A in N_2 such that $|L(A)| = 1$, substitute the unique terminal string in $L(A)$ for every occurrence of A in the right side of a production of G_2. From G_2, delete A and every production whose left side is A. Call the resulting linear cfg G_3.

Comment. $G_3 = (N_3, \Sigma, P_3, S)$ and each production of G_3 has the form $A \to t$ or $A \to t_1 B t_2$, with A, B in N_3, t in $\{0, 1\}^* \cdot \# \cdot \{0, 1\}^*$, and t_1, t_2 in $\{0, 1\}^*$.

Step 7. For each A in N_3 find a string in $L(A)$ and call this string X_A. If $L(G_3) = L_0$, then X_A must be of the form $u\#u^{\text{rev}}v^{\text{rev}}$ or $vu\#u^{\text{rev}}$, for u, v in $\{0, 1\}^*$. Verify that each X_A has one of the two above forms. Otherwise G is in LGINEQ(L_0).

Comment. If $X_A = u\#u^{\text{rev}}v^{\text{rev}}$ and $L(G_3) = L_0$, then A can only appear in sentential forms of form $zvAz^{\text{rev}}$. Similarly if $X_A = vu\#u^{\text{rev}}$ and $L(G_3) = L_0$, then A can only appear in sentential forms of form $zAv^{\text{rev}}z^{\text{rev}}$.

Step 8. For each production $A \to t$ such that $X_A = u\#u^{\text{rev}}v^{\text{rev}}$, verify that $t = w\#w^{\text{rev}}v^{\text{rev}}$, for some w in $\{0, 1\}^*$; and for each production $A \to t$ such that $X_A = vu\#u^{\text{rev}}$, verify that $t = vw\#w^{\text{rev}}$, for some w in $\{0, 1\}^*$. Otherwise G is in LGINEQ(L_0).

Step 9. For each production $A \to t_1Bt_2$ such that $X_A = u\#u^{\text{rev}}v^{\text{rev}}$, verify that $t_1X_Bt_2 = w\#w^{\text{rev}}v^{\text{rev}}$, and for each production $A \to t_1Bt_2$ such that $X_A = vu\#u^{\text{rev}}$, verify that $t_1X_Bt_2 = vw\#w^{\text{rev}}$, for some w in $\{0, 1\}^*$. Otherwise G is in LGINEQ(L_0).

Step 10. Verify that $X_S = u\#u^{\text{rev}}$. Otherwise G is in LGINEQ(L_0).

Comment. If G_3 satisfies the tests in Steps 8–10, then $L(G_3) \subseteq L_0$. This can be verified by induction; the inductive hypothesis is: If $X_A = u\#u^{\text{rev}}v^{\text{rev}}$, then $L(A) \subseteq \{w\#w^{\text{rev}}v^{\text{rev}} \mid w \in \{0, 1\}^*\}$ and if $X_A = vu\#u^{\text{rev}}$ then $L(A) \subseteq \{vw\#w^{\text{rev}} \mid w \in \{0, 1\}^*\}$ for all nonterminals A of G_3.

Step 11. Construct the right-linear grammar $G_4 = (N_3, \Sigma, P_4, S)$, where $A \to yB$ is in P_4 if and only if $A \to yBy'$ is in P_3 and $A \to y$ is in P_4 if and only if $A \to y\#y'$ is in P_3, where A, B are in N_3 and y, y' are in $\{0, 1\}^*$.

Step 12. Construct a nondeterministic finite automaton M such that $L(M) = L(G_4)$.

Step 13. Verify that $L(M) = \{0, 1\}^*$. If so, then $L(G) = L_0$; otherwise G is in LGINEQ(L_0).

Since G is linear, each of Steps 1–13 is executable in space bounded by a polynomial in $|G|$. Note that Steps 1–12 are executable in time bounded by a polynomial in $|G|$. From Proposition 2.3, Step 13 can be done in polynomial space. ☒

This algorithm can be modified to show that such sets as LGINEQ($\{a^nb^n \mid n \geqslant 1\}$) are elements of NP.

OPEN PROBLEM 3. (1) Does there exist an infinite but bounded cfl L_0 such that GINEQ(L_0) is an element of NP?

(2) Does there exist an unbounded cfl L_0 such that GINEQ(L_0) is an element of PSPACE?

Finally, as in Section 2, analogous results hold for containment as well.

4. An Exponential Combinatorial Structure

A natural analog of the "squaring" operator introduced in [24] that complicates several problems related to the cfg's is presented. We exploit this structure to prove that the equivalence and containment problems for cfg's generating finite sets require time at least $2^{cn/(\log n)}$ on any nondeterministic Tm. We also prove that the equivalence and containment problems for non-self-embedding cfg's require tape at least $2^{cn/(\log n)}$.

The exponential lower time bounds for equivalence and containment immediately apply to cfg's generating bounded languages. We also prove a metatheorem that implies an exponential lower time bound for a variety of predicates on cfg's generating bounded languages.

We present several problems that may be exponential due to the cfg "squaring" structure. These problems include structural equivalence of cfg's [21, 25], s-grammar equivalence [20], and $LL(k)$ grammar equivalence [26]. We note subexponential upper bounds on certain of these problems when restricted to the linear cfg's.

We now illustrate the cfg "squaring" structure with the grammar G_4 generating the language

$$L(G_4) = \{w \mid w \in \{0, 1\}^* \text{ and } |\, w \,| \leqslant 2^4\}.$$

The grammar has start symbol S_4 and contains the following productions.

$$\begin{aligned} S_4 &\to S_3S_3\,, \\ S_3 &\to S_2S_2\,, \\ S_2 &\to S_1S_1\,, \\ S_1 &\to S_0S_0\,, \\ S_0 &\to 0 \mid 1 \mid \lambda. \end{aligned}$$

Proposition 4.1. *There exists a constant $c > 0$ such that for all integers $n \geqslant 1$, there exists a* cfg G_n *with* $L(G_n) = \{w \mid w$ *in* $\{0, 1\}^*$ *and* $|\, w \,| \leqslant 2^n\}$ *and* $|\, G_n \,| \leqslant c \cdot n \cdot (\log n)$.

Proof. The productions of G_n are $S_n \to S_{n-1}S_{n-1}$, $S_{n-1} \to S_{n-2}S_{n-2}$,..., and $S_0 \to 0 \mid 1 \mid \lambda$. ☒

Definition 4.2 [28]. A function $T(m)$ is said to be a *running time* if there is a deterministic Tm M such that, for each input word x to M, M computes on x for precisely $T(|\, x \,|)$ steps. ▮

Theorem 4.3 [28]. *$Ndtime(T_2(n)) - Ndtime(T_1(n)) \neq \phi$, whenever $T_2(n)$ is a running time and* $\lim_{n\to\infty} (T_2(n)/T_1(n + 1)) = \infty$.

We next present an encoding of *Tm* computations into cfg's generating finite sets. This encoding is the basis of our exponential time lower bounds.

PROPOSITION 4.4. *Let M be any arbitrary nondeterministic 2^{cn} time bounded single tape Tm, for some integer constant $c > 0$. Let the state set, tape alphabet, start state, and set of accepting states of M be denoted by S, T, q_0, and F, respectively. Let $\Sigma = T \cup (S \times T) \cup \{\#\}$, where $\#$ is a symbol not in $T \cup (S \times T)$. Then there exists a deterministic log-space transducer M' such that M', when given a nonnull input $x = a_1 \cdot \cdots \cdot a_n$, outputs a* cfg G_x *such that*

(i) $L(G_x) \subseteq (\Sigma \cup \{\lambda\})^{2^{2cn+1}+3\cdot 2^{cn}+8}$;

(ii) $L(G_x) = (\Sigma \cup \{\lambda\})^{2^{2cn+1}+3\cdot 2^{cn}+8}$ *if and only if x is not in $L(M)$*;

(iii) $|G_x| \leqslant c_M \cdot n \cdot (\log n)$, *where c_M is a constant depending only upon M not x; and*

(iv) *M' is $O(n \cdot (\log n))$ time bounded.*

Proof. We refer to the instantaneous descriptions defined in Definition 1.12, taking $T(n) = 2^{cn}$. Also we let $(\Sigma)^{**}(k)$ denote $\Sigma^{[k]}$, where Σ is a finite alphabet and k is an expression denoting the positive integer $[k]$.

G_x is constructed so that $L(G_x)$ equals the set of invalid computations of M on x of length $\leqslant 2^{2cn+1} + 3 \cdot 2^{cn} + 8$. Any valid computation of M on x is a string over Σ of the form $\#\cdot[[(\Sigma - \{\#\})^{**}\, 2^{cn}] \cdot \#]^k$, where $k \leqslant 2^{cn} + 1$. Thus all valid computations of M on x have length $\leqslant 1 + (2^{cn} + 1) \cdot (2^{cn} + 1) = 2^{2cn} + 2^{cn+1} + 2$.

A string γ of length $\leqslant 2^{2cn+1} + 3 \cdot 2^{cn} + 8$ is an invalid computation if and only if

(1) $|\gamma|$ is less than the length of the initial i.d., i.e.,

$$\gamma \in S_1 = (\Sigma \cup \{\lambda\})^{**}(2^{cn} + 1);$$

(2) $2^{2cn} + 2^{cn+1} + 3 \leqslant |\gamma| \leqslant 2^{2cn+1} + 3 \cdot 2^{cn} + 8$, i.e.,

$$\gamma \in S_2 = [\Sigma^{**}(2^{2cn} + 2^{cn+1} + 3)] \cdot [(\Sigma \cup \{\lambda\})^{**}(2^{2cn} + 2^{cn} + 5)];$$

(3) $|\gamma| \leqslant 2^{2cn} + 2^{cn+1} + 2$ but γ does not end in "#", i.e.,

$$\gamma \in S_3 = [(\Sigma \cup \{\lambda\})^{**}(2^{2cn} + 2^{cn+1} + 1)] \cdot (\Sigma - \{\#\});$$

(4) γ contains an error between two consecutive i.d.'s, i.e.,

$$\gamma \in S_4 = \bigcup_{\sigma_1,\sigma_2,\sigma_3 \in \Sigma} [(\Sigma \cup \{\lambda\})^{**}(2^{2cn} + 2^{cn} + 1)] \cdot \sigma_1 \cdot \sigma_2 \cdot \sigma_3$$
$$\cdot [\Sigma^{**}(2^{cn} - 2)] \cdot (\Sigma^3 - f_M(\sigma_1, \sigma_2, \sigma_3)) \cdot [(\Sigma \cup \{\lambda\})^{**}(2^{2cn} + 2^{cn} + 1)];$$

(5) $|\gamma| \leqslant 2^{2cn} + 2^{cn+1} + 2$ but γ contains no symbol of form (q_f, t) with $q_f \in F$ and $t \in T$, i.e.,

$$\gamma \in S_5 = \left(\left(\Sigma - \bigcup_{\substack{q_f \in F \\ t \in T}} \{(q_f, t)\}\right) \cup \{\lambda\}\right)^{**} (2^{2cn} + 2^{cn+1} + 2);$$

or

(6) $|\gamma| \leqslant 2^{2cn} + 2^{cn+1} + 2$ but γ does not begin with

$$\#\cdot(q_0, x_1) \cdot x_2 \cdot \cdots \cdot x_n \cdot [(\not b)^{**}(2^{cn} - n)] \cdot \#.$$

If γ satisfies (6), then

(6.1) γ does not begin with $\#\cdot(q_0, x_1) \cdot x_2 \cdot \cdots \cdot x_n$; or

(6.2) γ does not contain two occurrences of $\#$; or

(6.3) γ does contain two occurrences of $\#$, but the first and second $\#$'s in γ are not separated by 2^{cn} characters; or

(6.4) γ has a nonblank character in its jth position, where $n + 2 \leqslant j \leqslant 2^{cn} + 1$.

All strings satisfying (6.1) are elements of

$$\begin{aligned} S_6 = [(\Sigma - \{\#\}) \cup \# \cdot [(\Sigma - \{(q_0, x_1)\}) \cup (q_0, x_1) \cdot [(\Sigma - \{x_2\}) \cup x_2 \cdot [\ldots \\ \cdot [(\Sigma - \{x_{n-1}\}) \cup x_{n-1} \cdot [(\Sigma - \{x_n\})]] \cdots]]]] \\ \cdot [(\Sigma \cup \{\lambda\})^{**}(2^{2cn} + 2^{cn+1} + 1)]. \end{aligned}$$

All strings satisfying (6.2) also satisfy (1), (3), or (6.1). All strings satisfying (6.3) are elements of

$$\begin{aligned} S_7 = \# \cdot [(\Sigma \cup \{\lambda\})^{**}(2^{cn} - 1)] \cdot \# \cdot [(\Sigma \cup \{\lambda\})^{**}(2^{2cn} + 2^{cn+1} + 1)] \\ \cup \# \cdot [(\Sigma - \{\#\})^{**}(2^{cn} + 1)] \cdot [(\Sigma \cup \{\lambda\})^{**}(2^{2cn} + 2^{cn})]. \end{aligned}$$

All strings satisfying (6.4) are elements of

$$\begin{aligned} S_8 = \Sigma^{n+1} \cdot [(\Sigma \cup \{\lambda\})^{**}(2^{cn} - n - 1)] \cdot (\Sigma - \{\not b\}) \\ \cdot [(\Sigma \cup \{\lambda\})^{**}(2^{2cn} + 2^{cn+1})]. \end{aligned}$$

We note that all strings in $S_1 \cup \cdots \cup S_8$ have length $\leqslant 2^{2cn+1} + 3 \cdot 2^{cn} + 8$. All strings in $S_6 \cup S_7 \cup S_8$ that do not satisfy (6.1), (6.2), or (6.3) have length greater than $2^{2cn} + 2^{cn+1} + 2$; and thus they are not valid computations. Finally, any string of length $\leqslant 2^{2cn+1} + 3 \cdot 2^{cn} + 8$ not in $S_1 \cup \cdots \cup S_8$ is a valid computation of M on x.

For fixed Tm M and variable x with $n = |x|$, cfg's $G_1, \ldots, G_8$ can be constructed such that $G_i = (N_i, \Sigma, \Pi_i, S); L(G_i) = S_i; |G_i| \leqslant k \cdot n \cdot (\log n)$, where k depends only upon M; $N_i \cap N_j = \{S\}$ for $1 \leqslant i < j \leqslant 8$; and S does not occur on the right side of any production. We only present G_1 and G_6. The other cfg's are constructed analogously.

Π_1 is defined by

$$
\begin{aligned}
S &\to A_0^{(1)} A_{cn}^{(1)}, \\
A_{cn}^{(1)} &\to A_{cn-1}^{(1)} A_{cn-1}^{(1)}, \\
A_{cn-1}^{(1)} &\to A_{cn-2}^{(1)} A_{cn-2}^{(1)}, \\
&\cdots \\
A_0^{(1)} &\to \lambda \mid \sigma_1 \mid \cdots \mid \sigma_m, \qquad \text{where} \quad \Sigma = \{\sigma_1, \ldots, \sigma_m\}.
\end{aligned}
$$

Π_6 is defined by

$$
\begin{aligned}
S &\to A_0^{(6)} B_0^{(6)}, \\
A_0^{(6)} &\to \sigma_i \mid \# A_1^{(6)} \qquad \text{for} \quad \sigma_i \in \Sigma \text{ and } \sigma_i \neq \#, \\
A_1^{(6)} &\to \sigma_j \mid (q_0, x_1) A_2^{(6)} \qquad \text{for} \quad \sigma_j \in \Sigma \text{ and } \sigma_j \neq (q_0, x_1), \\
&\cdots \\
A_{n-1}^{(6)} &\to \sigma_k \mid x_{n-1} A_n^{(6)} \qquad \text{for} \quad \sigma_k \in \Sigma \text{ and } \sigma_k \neq x_{n-1}, \\
A_n^{(6)} &\to \sigma_l \qquad \text{for} \quad \sigma_l \in \Sigma \text{ and } \sigma_l \neq x_n, \\
B_0^{(6)} &\to C_{2cn}^{(6)} C_{cn+1}^{(6)} C_0^{(6)}, \\
C_{2cn}^{(6)} &\to C_{2cn-1}^{(6)} C_{2cn-1}^{(6)}, \\
&\cdots \\
C_0^{(6)} &\to \lambda \mid \sigma_1 \mid \cdots \mid \sigma_m.
\end{aligned}
$$

Let $G_x = (N_1 \cup \cdots \cup N_8, \Sigma, \Pi_1 \cup \cdots \cup \Pi_8, S)$. Then $L(G_x) = S_1 \cup \cdots \cup S_8$; $L(G_x) \subseteq (\Sigma \cup \{\lambda\})^{**}(2^{2cn+1} + 3 \cdot 2^{cn} + 8)$; $L(G_x) = (\Sigma \cup \{\lambda\})^{**}(2^{2cn+1} + 3 \cdot 2^{cn} + 8)$ if and only if x is not in $L(M)$; and $|G_x| \leqslant c_M \cdot n(\log n)$, where c_M is a constant depending only upon M not x. We leave the proof that (iv) also holds to the reader. ☒

THEOREM 4.5. *There exists a constant $c > 0$ such that both the equivalence problem and the containment problem for* cfg's *generating finite sets require time at least $2^{cn/(\log n)}$ infinitely often (i.o.) on any nondeterministic multitape Tm.*

Proof. By Theorem 4.3 and the fact that a $2^{d_1 n}$ time bounded nondeterministic multitape Tm can be converted into a $2^{d_2 n}$ time bounded nondeterministic single-tape Tm with $d_2 \geqslant d_1$, there exists a constant $d > 1$ and a 2^{dn} time bounded nondeterministic single-tape Tm M such that the recognition of $L(M)$ on any nondeterministic multitape Tm requires time $\geqslant 2^n$ i.o. By Proposition 4.4 there exists an $O(n(\log n))$ time bounded deterministic log-space transducer M' such that M', when given an input x to M with $|x| = n$, outputs a cfg G_x such that $L(G_x) \subseteq L(n) = (\Sigma \cup \{\lambda\})^{2^{2dn+1}+3\cdot 2^{dn}+8}$, where Σ is a fixed finite alphabet depending upon M not x. Moreover, $L(G_x) \neq L(n)$ if and only if x is in $L(M)$. Thus $L(n)$ is not a subset of $L(G_x)$ if and only if x is in $L(M)$.

The following nondeterministic algorithm verifies that x is in $L(M)$, if this is true.

Step 1. Apply M' to x, outputting G_x.

Step 2. Verify that $L(G_x) = L(G_n)$, where $L(G_n) = L(n)$ and G_n contains the following productions (where an obvious shorthand is used to represent the productions whose righthand sides are terminal strings).

$$A \to A_{2dn+1}A_{dn}A_{dn}A_{dn}A_3, \qquad A_{2dn+1} \to A_{2dn}A_{2dn}, \ldots,$$

$$A_1 \to A_0A_0, \qquad A_0 \to \Sigma \mid \lambda.$$

Clearly $n \cdot (\log n) \leqslant |G_x| + |G_n| \leqslant c_1 \cdot n \cdot (\log n)$, where c_1 is a constant depending only upon M not x. Let $m = |G_x| + |G_n|$. The time required to execute this algorithm nondeterministically is no greater than $c_2 \cdot n \cdot (\log n)$, the time to execute Step 1, plus the time to test G_x and G_n for equivalence or containment.

Let $T(k)$ be the time required by some fixed nondeterministic multitape Tm to test for equivalence or containment of two cfg's G and H that generate finite sets, where $k = |G| + |H|$. Because time 2^n is required for verifying membership in $L(M)$, $2^n \leqslant c_2 \cdot n \cdot (\log n) + T(m)$ i.o. Thus, $T(m) \geqslant 2^{c_3 n}$ i.o. for some c_3 greater than zero. The fact that $m \leqslant c_1 \cdot n \cdot (\log n)$ implies $n \geqslant (1/c_1) \cdot (m/\log n)$. The fact that $n \cdot (\log n) \leqslant m$ implies $\log n \leqslant \log m$, so that $n \geqslant (1/c_1) \cdot (m/\log m)$. Letting $c_4 = c_3/c_1$, we conclude that

$$T(m) \geqslant 2^{c_4 m/(\log m)} \qquad \text{i.o.} \quad \boxtimes$$

Since all finite cfl's are bounded, the following immediate corollary of Theorem 4.5 holds.

THEOREM 4.6. *There exists a constant $c > 0$ such that determining if*

(a) $L(G) \subseteq L(H)$, *or*

(b) $L(G) \supseteq L(H)$, *or*

(c) $L(G) = L(H)$,

where G is an arbitrary cfg *and H is a* cfg *generating a bounded language, requires time at least $2^{cn/\log n}$ i.o. on any nondeterministic multitape Tm.*

These problems are shown to be decidable in [8].

Another analog of Theorem 4.5 holds for the non-self-embedding cfg's.

DEFINITION 4.7. A reduced cfg G is said to be *self-embedding* if there exists a nonterminal A of G such that $A \overset{*}{\Rightarrow} uAv$, where both u and v are nonnull strings of terminals. A cfg that is not self-embedding is said to be *non-self-embedding.*

PROPOSITION 4.8. *Let M be any arbitrary nondeterministic 2^{cn}* tape *bounded Tm, for some constant $c > 0$. Let the state set, tape alphabet, start state, and set of accepting states of M be denoted by S, T, q_0, and F, respectively. Let $\Sigma = T \cup (S \times T) \cup \{\#\}$, where $\#$ is a symbol not in $T \cup (S \times T)$. Then there exists a deterministic log-space transducer M' such the M', when given a nonnull input $x = x_1 \cdot \cdots \cdot x_n$, outputs a non-self-embedding* cfg *G_x such that*

(i) *$L(G_x) = \Sigma^*$ if and only if x is not in $L(M)$;*

(ii) *$| G_x | \leqslant C_M \cdot n \cdot (\log n)$, where C_M is a constant depending only upon M not x; and*

(iii) *M' is $O(n \cdot (\log n))$ time bounded.*

Proof. G_x is constructed so that $L(G_x)$ equals the set of invalid computations of M on x. We note that any valid computation of M on x is a string over Σ of the form $\# \cdot [[(\Sigma - \{\#\})^{**} 2^{cn}] \cdot \#]^k$ with $k \geqslant 1$.

A string γ in Σ^* is an invalid computation of M on x if and only if

(1) no $\#$'s appear or only one $\#$ appears in γ, i.e.,

$$\gamma \in S_1 = (\Sigma - \{\#\})^* \cup (\Sigma - \{\#\})^* \cdot \# \cdot (\Sigma - \{\#\})^*;$$

(2) two consecutive $\#$'s in γ are separated by less than or greater than 2^{cn} characters, i.e.,

$$\gamma \in S_2 = \Sigma^* \cdot \# \cdot [((\Sigma - \{\#\}) \cup \{\lambda\})^{**}(2^{cn} - 1)] \cdot \# \cdot \Sigma^* \cup \Sigma^* \cdot \# \cdot [(\Sigma - \{\#\}^{**}(2^{cn} + 1)] \cdot (\Sigma - \{\#\})^* \cdot \# \cdot \Sigma^*;$$

(3) γ does not end in $\#$, i.e., $\gamma \in S_3 = \Sigma^* \cdot (\Sigma - \{\#\})$;

(4) γ does not contain any symbols $(q_f, t) \in F \times T$, i.e.,

$$\gamma \in S_4 = (\Sigma - (F \times T))^*;$$

(5) γ contains an error between two consecutive i.d.'s, i.e.,

$$\gamma \in S_5 = \bigcup_{\sigma_1,\sigma_2,\sigma_3 \in \Sigma} \Sigma^* \cdot \sigma_1 \cdot \sigma_2 \cdot \sigma_3 \cdot [\Sigma^{**}(2^{cn} - 2)] \cdot (\Sigma^3 - f_M(\sigma_1, \sigma_2, \sigma_3)) \cdot \Sigma^*;$$

or

(6) γ does not begin with $\# \cdot (q_0, x_1) \cdot x_2 \cdot \cdots \cdot x_n \cdot [\not\phi^{**}(2^{cn} - n)] \cdot \#$. Any string satisfying (6) that does not satisfy (1) or (2) must be of the form $x \cdot y \cdot \# \cdot w$, where $|x| = n + 1$ and $x \neq \# \cdot (q_0, x_1) \cdot x_2 \cdot \cdots \cdot x_n$, or $|y| = 2^{cn} - n$ and $y \neq \not\phi^{**}(2^{cn} - n)$ with x, y, w in Σ^*. If $x \neq \# \cdot (q_0, x_1) \cdot x_2 \cdot \cdots \cdot x_n$ then

$$\gamma \in S_6 = [(\Sigma - \{\#\}) \cup \# \cdot [(\Sigma - \{(q_0, x_1)\}) \cup (q_0, x_1) \cdot [\cdots [(\Sigma - \{x_{n-1}\}) \cup x_{n-1} \cdot [(\Sigma - \{x_n\})]] \cdots]]] \cdot \Sigma^*.$$

If $y \neq \not\phi^{**}(2^{cn} - n)$ then

$$\gamma \in S_7 = \# \cdot \Sigma^n \cdot (\Sigma - \{\#\})^* \cdot (\Sigma - \{\not\phi, \#\}) \cdot (\Sigma - \{\#\})^* \cdot \# \cdot \Sigma^*.$$

But non-self-embedding cfg's $G_1, \ldots, G_7$ can be constructed such that $G_i = (N_i, \Sigma, \Pi_i, S)$; $L(G_i) = S_i$; $|G_i| \leqslant k \cdot n(\log n)$; $N_i \cap N_j = \{S\}$ for $1 \leqslant i < j \leqslant 7$, and S does not occur on the right side of any production. The construction of the grammars is straightforward and is left to the reader.

$$G_x = (N_1 \cup \cdots \cup N_7, \Sigma, \Pi_1 \cup \cdots \cup \Pi_7, S); \qquad L(G_x) = S_1 \cup \cdots \cup S_7;$$

$L(G_x) \neq \Sigma^*$ if and only if x is not in $L(M)$; and $|G_x| \leqslant C_M \cdot n \cdot (\log n)$, where C_M is a constant depending only upon M not x. We leave the remainder of the proof to the reader. ☒

THEOREM 4.9. *There exists a constant $c > 0$ such that both the equivalence problem and the containment problem for the non-self-embedding* cfg's *require tape at least $2^{cn/(\log n)}$ i.o. on any* Tm.

Theorem 4.9 follows from Proposition 4.8 and known space hierarchy results (see [13]) by an argument very similar to that of the proof of Theorem 4.5. Therefore, it is left to the reader.

It is well known (see [13]) that non-self-embedding cfg's only generate regular sets. Moreover, given a non-self-embedding cfg G, an equivalent regular grammar H can be effectively obtained. Thus the equivalence and containment problems for the non-self-embedding cfg's are easily seen to be decidable. Analogous to Theorem 2.6(3), the following corollary of Proposition 4.8 holds.

PROPOSITION 4.10. *There exists a constant $c > 0$ such that for any fixed unbounded*

regular set L_0, $\{G \mid G$ is a non-self-embedding cfg *and $L(G) \neq L_0\}$ requires tape at least $2^{cn/(\log n)}$ i.o. on any nondeterministic multitape Tm.*

The proof is exactly analogous to the proof in Theorem 2.6(3) that RINEQ(L_0) is CSL-hard and is left to the reader.

Another corollary of Theorem 4.5 is the following metatheorem about the complexity of predicates on the bounded cfl's.

THEOREM 4.11. *Let P be any nontrivial predicate on the bounded* cfl's *such that*

(a) *P is true for all bounded regular sets, and*

(b) *either $\mathscr{L} = \{L' \mid L' = x\backslash L, x \in \{0, 1\}^*, P(L)$ is true$\}$ or $\mathscr{R} = \{L' \mid L' = L/x, x \in \{0, 1\}^*, P(L)$ is true$\}$ is a proper subset of the bounded* cfl's.

Then there exists a constant $c > 0$ such that any nondeterministic multitape Tm *that recognizes $\{G \mid G$ is a* cfg *generating a bounded language and $P(L(G))$ is false$\}$ requires time at least $2^{cn/(\log n)}$ i.o.*

Proof. We only prove the theorem for the case when $\mathscr{L}$ is a proper subset of the bounded cfl's. The proof when $\mathscr{R}$ is a proper subset of the bounded cfl's is similar.

Let L be a bounded cfl not in $\mathscr{L}$. Since L is bounded, there exist strings $w_1, \ldots, w_k$ in $\{0, 1\}^+$ such that $L \subseteq w_1^* \cdot \cdots \cdot w_k^*$. Let Σ, G_x, and G_n be as in the proofs of Proposition 4.4 and Theorem 4.5. Let \$ be a symbol not in Σ. Let $\rho: \Sigma \cup \{\$\} \to \{0, 1\}^+$ be some 1–1 function with the property that for all s in $\Sigma \cup \{\$\}$, $|\rho(s)|$ is a constant. Let $\hat{\rho}: (\Sigma \cup \{\$\})^* \to \{0, 1\}^*$ be the homomorphic extension of ρ. Let

$$L_x = \hat{\rho}(L(G_x)) \cdot \rho(\$) \cdot w_1^* \cdot \cdots \cdot w_k^* \cup \hat{\rho}(L(G_n)) \cdot \rho(\$) \cdot L.$$

If $L(G_x) = L(G_n)$, then $L_x = \hat{\rho}(L(G_n)) \cdot \rho(\$) \cdot w_1^* \cdot \cdots \cdot w_k^*$. Thus L_x is a regular bounded language; and by (a), $P(L_x)$ is true. Otherwise let z be in $L(G_n) - L(G_x)$. By the properties of $\hat{\rho}$, $\hat{\rho}(z) \cdot \rho(\$)\backslash L_x = L$. Thus by (b), $P(L_x)$ is false. Hence $P(L_x)$ is true if and only if $L(G_x) = L(G_n)$. Finally, given L, G_x, and G_n, a cfg G_x' can be constructed deterministically in $O(|G_x| + |G_n|)$ time such that $L(G_x') = L_x$. Clearly L_x is bounded. The remainder of the proof exactly parallels that of Theorem 4.5 and is left to the reader. ☒

Thus, such predicates as regularity, linearity, inherent ambiguity, and "$L(G)$ is a deterministic cfl" for the bounded cfl's all require at least exponential time on any nondeterministic multitape Tm. Moreover the proof of Theorem 4.11 shows that these lower bounds hold even for cfg's that are known in advance to generate bounded languages.

A similar theorem holds for the bounded linear cfl's as well.

THEOREM 4.12. *Let P be any nontrivial predicate on the bounded linear* cfl's *such that*

(a) *P is true for all bounded regular sets, and*

(b) *either $\mathscr{L} = \{L' \mid L' = x\backslash L,\ x \in \{0, 1\}^*,\ P(L)$ is true$\}$ or $\mathscr{R} = \{L' \mid L' = L/x,\ x \in \{0, 1\}^*,\ P(L)$ is true$\}$ is a proper subset of the bounded linear* cfl's.

Then $\{G \mid G$ is a linear cfg *generating a bounded language and $P(L(G))$ is false$\}$ is NP-hard.*

Proof. We only sketch the proof for the case when $\mathscr{L}$ is a proper subset of the bounded linear cfl's.

Let L be a bounded linear cfl not in $\mathscr{L}$. Then there exist strings $w_1, \ldots, w_k$ in $\{0, 1\}^+$ such that $L \subseteq w_1^* \cdot \cdots \cdot w_k^*$. Let f be a D_3-Boolean form with m clauses and n variables. Then as in the proof of Theorem 2.7(1), a $(\cup, \cdot)$ regular expression β can be constructed deterministically in time bounded by a polynomial in $|f|$ such that $L(\beta) \subseteq \{0, 1\}^n$ and $L(\beta) = \{0, 1\}^n$ if and only if f is a tautology.

Let the homomorphism $\rho: \{0, 1\} \to \{0, 1\}$ be defined by $\rho(0) = 00$ and $\rho(1) = 01$. Let

$$L_\beta = \rho(L(\beta)) \cdot 10 \cdot w_1^* \cdot \cdots \cdot w_k^* \cup \{00, 01\}^n \cdot 10 \cdot L.$$

Then $P(L_\beta)$ is true if and only if $L(\beta) = \{0, 1\}^n$, which is true if and only if f is a tautology. Clearly, given β (as constructed in the proof of Theorem 2.7(1)), a linear grammar G_β can be constructed deterministically in time bounded by a polynomial in $|\beta|$ such that $L(G_\beta) = L_\beta$. Again, clearly L_β is a bounded cfl. ☒

Finally, we list several other problems which might be exponential in time or space due to the cfg "squaring" structure. Since this combinatorial structure is the essence of nonlinearity for cfg's, we also study the complexity of several of these problems when restricted to the linear cfg's.

1. *Structural equivalence of cfg's* (for the definition see Definition 5.3 below):

OPEN PROBLEM 4. Is structural equivalence of cfg's decidable by some polynomially space bounded Tm?

We show in Section 5 that for linear cfg's, structural equivalence is decidable in polynomial space.

2. *Boundedness of cfg's*:

OPEN PROBLEM 5. Is the predicate "$L(G)$ is bounded" for cfg's p-decidable?

PROPOSITION 4.13. *Boundedness for linear* cfg's *is decidable by some deterministic polynomially time bounded Tm.*

Proof. Let G be reduced with $L(G) \neq \phi$. From [9, Lemma 3.1], $L(G)$ is bounded if and only if for each nonterminal A of G, both $L_A(G) = \{u \mid A \overset{*}{\Rightarrow} uAv\}$ and $R_A(G) = \{v \mid A \overset{*}{\Rightarrow} uAv\}$ are commutative. For linear cfg G and nonterminal A of G, cfg's $\mathscr{L}_A$ and $\mathscr{R}_A$ such that $L(\mathscr{L}_A) = L_A(G)$ and $L(\mathscr{R}_A) = R_A(G)$ are constructible in time polynomial in $| G |$. Moreover, if G is linear, then so are $\mathscr{L}_A$ and $\mathscr{R}_A$.

The following algorithm tests a cfg G for commutativity.

Step 1. Reduce G. Let G' be the resulting reduced grammar.

Step 2. Test if $L(G') = \phi$. If $L(G') = \phi$; then $L(G)$ is commutative.

Step 3. Find a string x in $L(G')$. By Lemma 2.9(1), $L(G')$ is commutative if and only if there exists a string w such that $L(A) \subseteq w^*$. But any such w must have the property that $x = w^n$ for some nonnegative integer n. Find each of the prefixes w of x, such that $x = w^n$ for *some* n. For each such w, test whether $L(G') \cap \overline{w^*}$ is empty.

From Lemma 3.1(6) and (3), Steps 1 and 2 are executable in polynomial time. For a nonempty linear grammar G, there exists an x in $L(G)$ such that $| x | \leqslant | G |$ and x can be found in polynomial time. For each w to be tested in Step 3, a deterministic finite automaton recognizing w^* can be constructed in polynomial time. Therefore, a ndfa recognizing $\overline{w^*}$ can be found in polynomial time. From Lemma 3.1(2) and (3), Step 3 is therefore executable in polynomial time. ☒

Analogs of Proposition 4.13 also hold for other classes of "linear-like" cfg's such as the metalinear cfg's. However, for arbitrary cfg's G, the length of the shortest string x in $L(G)$ is not polynomially bounded in $| G |$.

3. *Equivalence of s and LL grammars* [20, 26]:

OPEN PROBLEM 6. Are the equivalence problems for s-grammars or $LL(k)$ grammars p-decidable?

DEFINITION 4.15. A cfg is an s-grammar (simple grammar) if and only if

(1) The right side of every production begins with a terminal symbol; and

(2) if $A \to a\alpha$ and $B \to a\beta$ are distinct productions, where a is a terminal symbol, then $A \neq B$. ▮

The following result is shown in [17].

THEOREM 4.16. *The equivalence problem for linear s-grammars is p-decidable.*

In each of Open Problems (4)–(6) the nonlinear structure of Proposition 4.1 causes the known algorithms to be exponential.

5. Grammatical Covering and Structural Equivalence

In Sections 2 and 3 we saw that for all infinite L_0, RINEQ(L_0) and GINEQ(L_0) are computationally intractable even when decidable. In this section we show that these results carry over to several more practically significant problems about cfg's, especially grammatical covering and structural equivalence. We investigate the complexity of grammatical covering and structural equivalence for the right-linear, linear, and arbitrary cfg's.

The main results of this section are summarized in Table I.

TABLE I

Class	Problem		
	Language equivalence	Covering	Structural equivalence
Right-linear cfg	PSPACE-complete	PSPACE-complete	PSPACE-complete
Linear cfg	Undecidable	PSPACE-complete	PSPACE-complete
Arbitrary cfg	Undecidable	Undecidable	Decidable, but PSPACE-hard

Definition 5.1 [10]. Let $G_1 = (N_1, \Sigma, P_1, S_1)$ and $G_2 = (N_2, \Sigma, P_2, S_2)$ be cfg's. Let h be a map from P_1 to $P_2 \cup \{\lambda\}$. Let h be extended to a map from P_1^* to P_2^* by defining $h(\lambda) = \lambda$ and $h(\pi p) = h(\pi)\, h(p)$ for all π in P_1^* and p in P_1. We say that G_1 *left covers* G_2 under h if for all w in Σ^*

(1) whenever $S_1 \Rightarrow_L^{\pi} w$, then $S_2 \Rightarrow_L^{h(\pi)} w$; and

(2) whenever $S_2 \Rightarrow_L^{\pi} w$ then there exists a π' such that $h(\pi') = \pi$ and $S_1 \Rightarrow_L^{\pi'} w$.

The notion of *right covers* is defined analogously except rightmost rather than leftmost deviations are considered. ∎

Throughout this section we frequently use the word *cover* as a generic term referring to both kinds of covering. We note the following obvious facts.

Proposition 5.2. (1) *If G_1 covers G_2, then $L(G_1) = L(G_2)$.*

(2) *If G_1 covers G_2, then for any x in Σ^*, the number of leftmost derivations of x in G_1 is greater than or equal to the number of derivations of x in G_2.*

The notion of covering arose from the fact that compiler writers are frequently forced to parse according to a grammar more complex than the one they are actually

interested in. A covering map allows them to parse according to one grammar, but build a tree or call semantic routines according to another.

Another notion of similarity of cfg's is provided by

DEFINITION 5.3. Let $G = (N, \Sigma, P, S)$ be a cfg. Define the *parenthesized grammar* corresponding to G by

$$\mathscr{P}(G) = (N, \Sigma \cup \{[\ ,\ ]\}, P', S),$$

where "[" and "]" are new symbols and $P' = \{A \to [\alpha] \mid A \to \alpha \in P\}$. Two grammars G_1 and G_2 are called *structurally equivalent* if $L(\mathscr{P}(G_1)) = L(\mathscr{P}(G_2))$.

PROPOSITION 5.4 [25]. *If G_1 and G_2 are structurally equivalent, then $L(G_1) = L(G_2)$.*

Structural equivalence essentially says that if you examine the sets of trees generated by the grammars in question, then these sets of trees are identical if intermediate nodes are considered to be unlabeled. The following result has appeared several times in the literature [19, 21, 25].

PROPOSITION 5.5. *It is decidable whether two arbitrary* cfg's *are structurally equivalent.*

A moment's reflection will reveal that these concepts (structural equivalence and grammatical covering) are less general than language equivalence. Consider the grammars of

$$\begin{aligned} G_1\colon\ & S \to A \mid B & \qquad G_2\colon\ & S \to C0 \mid 0 \\ & A \to 0A \mid 0 & & C \to D0 \mid 0 \\ & B \to 0B \mid 0, & & D \to E0 \mid 0 \\ & & & E \to S0 \mid 0. \end{aligned}$$

G_1 generates right-linear trees and G_2 generates left-linear trees so G_1 and G_2 cannot be structurally equivalent. Moreover, since G_1 has fewer productions than G_2 and every production of G_2 is used in some derivation, G_1 cannot cover G_2. Finally, note that G_1 is ambiguous but G_2 is not; hence G_2 cannot cover G_1. Thus, neither of the concepts of covering and structural equivalence applies to G_1 and G_2 even though $L(G_1) = L(G_2) = 0^+$.

In this section, we reduce several problems to the equivalence problem for right-linear grammars. Let

EQUIV-(right-linear)
$= \{(G_1, G_2) \mid G_1$ and G_2 are equivalent right-linear grammars$\}$.

INEQUIV-(right-linear)
$= \{(G_1, G_2) \mid G_1$ and G_2 are inequivalent right-linear grammars$\}$.

We use the following lemma.

LEMMA 5.6. *EQUIV-(right-linear) and INEQUIV-(right-linear) are both in PSPACE.*

Proof. Inequivalence of two right-linear grammars can be verified (in a manner similar to the verification procedure in [24] for inequivalence of ndfa's) by a nondeterministic lba that guesses a string which is in one language, but not in the other. Therefore, from Lemma 1.8(1), INEQUIV-(right-linear) is in PSPACE. From Lemma 1.8(2), EQUIV-(right-linear) is also in PSPACE. ∎

We now consider the complexity of problems involving covering. We first establish several upper bounds, using the equivalence test for linear s-grammars (Theorem 4.16) as a chief tool.

LEMMA 5.7. $\{(G_1, G_2, h) \mid G_1$ *and* G_2 *are right-linear* cfg's *and* G_1 *covers* G_2 *under* $h\}$ *is an element of PSPACE.*

Proof. Let $G_1 = (N_1, \Sigma, P_1, S_1)$ and $G_2 = (N_2, \Sigma, P_2, S_2)$. We need efficient tests for

(*) For all π ($S_1 \Rightarrow^{\pi} w$ implies $S_2 \Rightarrow^{h(\pi)} w$),

and

(**) For all π ($S_2 \Rightarrow^{\pi} w$ implies there exists π' such that $S_1 \Rightarrow^{\pi'} w$ and $h(\pi') = \pi$).

First we will show how (*) may be reduced to linear s-grammar equivalence. Define $G' = (N_1, P_1 \cup \Sigma, P', S_1)$, where

$$P' = \{A \to pBw^{\text{rev}} \mid p \in P_1 \text{ and } p = A \to wB\} \cup \{A \to pw^{\text{rev}} \mid p \in P_1 \text{ and } p = A \to w\}.$$

Thus, $L(G') = \{\pi w^{\text{rev}} \mid S_1 \Rightarrow^{\pi} w\}$.

Next define $G'' = (N_1 \times (N_2 \cup \{\lambda\}), P_1 \cup \Sigma, P'', (S_1, S_2))$, where

$$\begin{aligned} P'' = {} & \{(A, B) \to p(C, D)\, w^{\text{rev}} \mid p \in P_1, \text{ there exists } y \text{ such that} \\ & \quad p = A \to yC \text{ and } h(p) = B \to wD\} \\ & \cup \{(A, B) \to p(C, B) \mid p \in P_1, \text{ there exists } y \text{ such that} \\ & \quad p = A \to yC \text{ and } h(p) = \lambda\} \\ & \cup \{(A, B) \to p(C, \lambda)\, w^{\text{rev}} \mid p \in P_1, \text{ there exists } y \text{ such that} \\ & \quad p = A \to yC \text{ and } h(p) = B \to w\} \\ & \cup \{(A, \lambda) \to p(C, \lambda) \mid p \in P_1, \text{ there exists } y \text{ such that} \\ & \quad p = A \to yC \text{ and } h(p) = \lambda\} \end{aligned}$$

$$\cup \{(A, B) \to pw^{\text{rev}} \mid p \in P_1 \text{, there exists } y \text{ such that } p = A \to y \text{ and } h(p) = B \to w\}$$
$$\cup \{(A, \lambda) \to p \mid p \in P_1 \text{, there exists } y \text{ such that } p = A \to y \text{ and } h(p) = \lambda\}.$$

Thus,

$$L(G'') = \{\pi w^{\text{rev}} \mid S_2 \overset{h(\pi)}{\Rightarrow} w \text{ and there exists } y \text{ such that } S_1 \overset{\pi}{\Rightarrow} y\}.$$

Observe that G' and G'' are linear s-grammars and that

$$G' \mid + \mid G'' \mid = O(n^2), \qquad \text{where} \quad n = \mid G_1 \mid + \mid G_2 \mid + \mid h \mid.$$

We claim that

$$L(G') = L(G'') \qquad \text{if and only if (*).}$$

Proof of claim. Only if: Let $S_1 \Rightarrow^{\pi} w$ be a derivation of G_1. Then πw^{rev} is in $L(G')$ and hence πw^{rev} is in $L(G'')$. By the definition of G'' this implies $S_2 \Rightarrow^{h(\pi)} w$.

If: $L(G') \neq L(G'')$ if and only if there exists π such that $S_1 \Rightarrow^{\pi} w$ but not $S_2 \Rightarrow^{h(\pi)} w$, which would violate (*).

Hence, (*) is polynomially reducible to linear s-grammar equivalence, which in turn by Theorem 4.16 is in P. Thus, (*) can be tested in PSPACE. Second, we will show how (**) can be reduced to containment of right-linear grammars.

Define $\bar{G} = (N_1, P_2, \bar{P}, S_1)$, where

$$\bar{P} = \{A \to pB \mid \text{there exists } y \text{ such that } A \to yB \text{ is in } P_1 \text{ and } p = h(A \to yB)\}$$
$$\cup \{A \to p \mid \text{there exists } y \text{ such that } A \to y \text{ is in } P_1 \text{ and } p = h(A \to y)\}.$$

Clearly $L(\bar{G}) = \{h(\pi) \mid \text{there exists } w \text{ such that } S_1 \Rightarrow^{\pi} w\}$.

Define $\hat{G} = (N_2, P_2, \hat{P}, S_2)$, where

$$\hat{P} = \{A \to pB \mid p \in P_2 \text{ and there exists } y \text{ such that } p = A \to yB\}$$
$$\cup \{A \to p \mid p \in P_2 \text{ and there exists } y \text{ such that } p = A \to y\}.$$

Clearly $L(\hat{G}) = \{\pi \mid \text{there exists } w \text{ such that } S_2 \Rightarrow^{\pi} w\}$.

The reader may easily verify that $\bar{G}$ and $\hat{G}$ are right-linear and that $\mid \bar{G} \mid + \mid \hat{G} \mid = O(\mid G_1 \mid + \mid G_2 \mid)$.

We claim that once (*) is established,

$$L(\bar{G}) \supseteq L(\hat{G}) \qquad \text{if and only if (**).}$$

Proof of claim. Only if: Suppose $S_2 \Rightarrow^{\pi} w$. Then π is in $L(\bar{G})$ and thus π is in $L(\bar{G})$. Hence by definition of $\bar{G}$, there exist π', y such that $h(\pi') = \pi$ and $S_1 \Rightarrow^{\pi'} y$. But by (*), y must be w.

If: Suppose π is in $L(\bar{G})$. By definition of $\bar{G}$, there exists w such that $S_2 \Rightarrow^{\pi} w$. By (**), there exists π' such that $h(\pi') = \pi$ and $S_1 \Rightarrow^{\pi'} w$. By definition of $\bar{G}$, $h(\pi')$ is in $L(\bar{G})$ and so π is in $L(\bar{G})$.

Thus (**) is polynomially reducible to right-linear grammar containment. $L(\bar{G}) \supseteq L(\bar{\bar{G}})$ if and only if $L(\bar{G}) \cup L(\bar{\bar{G}}) = L(\bar{G})$. A right-linear grammar generating $L(\bar{G}) \cup L(\bar{\bar{G}})$ can be found in polynomial time. From Lemma 5.6, the equivalence problem can be solved in polynomial space. Thus (**) can be tested in polynomial space. ∎

A more detailed argument is used in [17] to show the stronger result:

LEMMA 5.8. *$\{(G_1, G_2, h) \mid G_1$ and G_2 are linear* cfg's *and G_1 covers G_2 under $h\}$ is in PSPACE.*

Since any map between the sets of productions of two grammars can be expressed in an amount of space proportional to the sizes of the grammars, we have

COROLLARY 5.9. *$\{(G_1, G_2) \mid G_1$ and G_2 are linear* cfg's *and G_1 covers G_2 under some map $h\}$ is in PSPACE.*

Proof. We simply test all possible maps from P_1 to $P_2 \cup \{\lambda\}$ using the polynomially space-bounded algorithm of Lemma 5.8 as a "subroutine." ∎

In the next lemma, we provide a lower bound on the complexity of covering problems.

LEMMA 5.10. *$\{(G_1, G_2, h) \mid G_1$ and G_2 are type-3 grammars over $\{0, 1\}$ and G_1 covers G_2 under $h\}$ is PSPACE-hard.*

Proof. Let $G = (N, \{0, 1\}, P, S_1)$ be an arbitrary type-3 grammar. Let $\bar{G} = (\{S\}, \{0, 1\}, \bar{P}, S)$ with

$$\bar{P} = \{S \to 0S, S \to 1S, S \to 0, S \to 1\}.$$

Let $h: P \to \bar{P}$ be defined by

$$\begin{aligned} h(A \to \alpha) &= S \to 0S \quad \text{if} \quad \alpha \in \{0\} \cdot N, \\ &= S \to 1S \quad \text{if} \quad \alpha \in \{1\} \cdot N, \\ &= S \to 0 \quad \text{if} \quad \alpha \in \{0\}, \\ &= S \to 1 \quad \text{if} \quad \alpha \in \{1\}. \end{aligned}$$

We claim that G covers $\bar{G}$ under h if and only if $L(G) = \{0, 1\}^*$. There are two distinct cases to consider.

Case 1. $L(G) \neq \{0, 1\}^*$. Since $L(\bar{G}) = \{0, 1\}^*$, Proposition 5.2 guarantees that no covering map exists.

Case 2. $L(G) = \{0, 1\}^*$. By induction on the number of steps in a derivation,

$$S_1 \overset{\pi}{\Rightarrow} wA \text{ for some nonterminal } A \text{ implies } S \overset{h(\pi)}{\Rightarrow} wS.$$

Thus,

$$(*) \qquad S_1 \overset{\pi}{\Rightarrow} w \text{ implies } S \overset{h(\pi)}{\Rightarrow} w.$$

Next, suppose $S \Rightarrow^{\pi} w$. Since $L(G) = L(\bar{G})$, there exists π' such that $S_1 \Rightarrow^{\pi'} w$. By (*), $S \Rightarrow^{h(\pi')} w$. Since $\bar{G}$ is unambiguous $h(\pi') = \pi$ and we have

$$(**) \quad S \overset{\pi}{\Rightarrow} w \text{ implies there exists } \pi' \text{ such that } h(\pi') = \pi \text{ and } S_1 \overset{\pi'}{\Rightarrow} w.$$

Thus, G covers $\bar{G}$ under h.

Since $|G| + |\bar{G}| = O(|G|)$ and the equivalence to $\{0, 1\}^*$ problem is PSPACE-hard (Proposition 2.4 and Lemma 1.10), the covering problem is also PSPACE-hard.

Next, we show that freedom to choose the map h in any possible way does not make this problem easier.

COROLLARY 5.11. *$\{G_1, G_2) \mid G_1$ and G_2 are type-3 grammars and G_1 covers $G_2\}$ is PSPACE-hard.*

Proof. In the proof of the previous lemma we saw that if $L(G) \neq L(\bar{G})$ then no covering map exists. On the other hand, if $L(G) = \{0, 1\}^* = L(\bar{G})$, then there exists a covering map, namely, h of the proof of Lemma 5.10. Thus, a cover exists if and only if $L(G) = \{0, 1\}^*$.

These results can be combined to show that all of our bounds are tight.

THEOREM 5.12. *Let $\mathscr{S}_1(T) = \{(G_1, G_2, h) \mid G_1, G_2 \in T$ and G_1 covers G_2 under $h\}$. Let $\mathscr{S}_2(T) = \{(G_1, G_2) \mid G_1, G_2 \in T$ and G_1 covers $G_2\}$. Then $\mathscr{S}_1(T)$ and $\mathscr{S}_2(T)$ are PSPACE-complete for*

(a) $T = \{G \mid G$ *is type-3*$\}$,

(b) $T = \{G \mid G$ *is right-linear*$\}$,

(c) $T = \{G \mid G$ *is linear*$\}$.

Proof. Lemma 5.10 and Corollary 5.11 establish lower bounds for $\mathscr{S}_1$ and $\mathscr{S}_2$, respectively. Lemma 5.8 and Corollary 5.9 provide the necessary upper bounds. ∎

Let us next consider the general case of determining whether an arbitrary cfg covers another arbitrary cfg. Using a technique reminiscent of that employed in the proof of Lemma 5.10, we show Theorem 5.13. The proof of Theorem 5.13 assumes that "cover" means left cover; an analogous proof holds for right cover.

THEOREM 5.13. *Let* $\mathscr{S} = \{(G_1, G_2, h) \mid G_1$ *and* G_2 *are arbitrary* cfg's *over* $\{0, 1\}$ *and* G_1 *covers* G_2 *under* $h\}$. *Then* $\mathscr{S}$ *is not recursively enumerable.*

Proof. Let G be any cfg. We will construct G', G'', and h such that G' covers G'' under h if and only if $L(G) = \{0, 1\}^*$. Since this latter problem is well known to be undecidable and the complement of $\mathscr{S}$ is clearly recursively enumerable we can conclude that $\mathscr{S}$ is not recursively enumerable.

An arbitrary cfg can be converted in polynomial time into an equivalent cfg $G = (N, \{0, 1\}, P, S)$, for which the right side of each production is in $(0 + 1 + \lambda)N^*$.

Define $G' = (N \cup \{S', A\}, \{0, 1\}, P', S')$, where S' and A are new characters and $P' = P_0 \cup P_1 \cup P_2 \cup P_3 \cup P_4$, with

$$P_0 = \{X \to 0\alpha \mid X \to 0\alpha \in P\},$$

$$P_1 = \{X \to 1\alpha \mid X \to 1\alpha \in P\},$$

$$P_2 = \{X \to \alpha \mid X \to \alpha \in P \text{ and } \alpha \in N^*\},$$

$$P_3 = \{A \to 1\},$$

$$P_4 = \{S' \to SA\}.$$

(Note that $P = P_0 \cup P_1 \cup P_2$).

Define $G'' = (\{S''\}, \{0, 1\}, P'', S'')$, where $P'' = \{S'' \to 0S'', S'' \to 1S'', S'' \to 1\}$.

Finally, define $h: P' \to P'' \cup \{\lambda\}$ by

$$\begin{aligned} h(p) &= S'' \to 0S'' && \text{if } p \in P_0, \\ &= S'' \to 1S'' && \text{if } p \in P_1, \\ &= S'' \to 1 && \text{if } p \in P_3, \\ &= \lambda && \text{if } p \in P_2 \cup P_4. \end{aligned}$$

We claim that G' covers G'' under h if and only if $L(G) = \{0, 1\}^*$.

Case 1. $L(G) \neq \{0, 1\}^*$. Then $L(G') = L(G) \cdot \{1\} \neq \{0, 1\}^* \cdot \{1\} = L(G'')$. Thus by Proposition 5.2, G' cannot cover G''.

Case 2. $L(G) = \{0, 1\}^*$. Then $L(G') = L(G'') = \{0, 1\}^* \cdot \{1\}$. It is readily seen by induction on the number of steps in a leftmost derivation that

$$S' \overset{\pi}{\Rightarrow} wA \text{ implies } S'' \overset{h(\pi)}{\Rightarrow} wS''.$$

This in turn yields

(*) $$S' \overset{\pi}{\Rightarrow} w \text{ implies } S'' \overset{h(\pi)}{\Rightarrow} w.$$

Thus every derivation of G' maps onto a derivation of G''. We must next show that every derivation in G'' is mapped onto by some derivation in G'.

Suppose $S'' \Rightarrow^{\pi} w$ for some π. Since $L(G') = L(G'')$ there exists a derivation π' of w in G'. Moreover, by (*), $h(\pi')$ is a derivation of w in G''. Finally, since G'' is unambiguous, we conclude that $h(\pi') = \pi$. We thus conclude

(**) $S'' \overset{\pi}{\Rightarrow} w$ implies there exists π' such that $h(\pi') = \pi$ and $S' \overset{\pi'}{\Rightarrow} w$.

Thus G' covers G'' under h. ∎

Since the map h described above works as a cover if any map does, we have the following

COROLLARY 5.14. *The set* $\mathscr{S} = \{(G_1, G_2) \mid G_1$ *and* G_2 *are arbitrary* cfg's *over* $\{0, 1\}$ *and* G_1 *covers* $G_2\}$ *is not recursively enumerable.*

In summary then, the covering problem is PSPACE-complete for type-3, right-linear, and linear grammars, but is undecidable for arbitrary cfg's.

Next we consider structural equivalence.

We first dispose of a degenerate case, namely, structural equivalence of type-3 grammars. Recall that every production of such a grammar is either of the form $A \to bC$ or $A \to b$. Thus the tree structure corresponding to any particular string is preordained by the fact that the grammar is type-3 and completely independent of the structure of the grammar itself! We thus are led to

LEMMA 5.15. *Two type-3 grammars are structurally equivalent if and only if their generated languages are equivalent.*

Proof. The "only if" portion is simply a restatement of Proposition 5.4. To establish the "if" portion, let G_1 and G_2 be the relevant grammars and $\mathscr{P}(G_1)$ and $\mathscr{P}(G_2)$ their parenthesis grammars. Observe that $a_1 \cdots a_n$ is in $L(G_i)$ if and only if $[a_1[a_2[a_3 \cdots [a_n]^n$ is in $L(\mathscr{P}(G_i))$. Thus $L(G_1) = L(G_2)$ implies $L(\mathscr{P}(G_1)) = L(\mathscr{P}(G_2))$, which is simply the definition of structural equivalence. ∎

An immediate consequence of the preceding lemma is

THEOREM 5.16. *Let* $\mathscr{S}_1 = \{(G_1, G_2) \mid G_1$ *and* G_2 *are structurally inequivalent type-3 grammars over* $\{0\}\}$. *Let* $\mathscr{S}_2 = \{(G_1, G_2) \mid G_1$ *and* G_2 *are structurally inequivalent type-3 grammars over* $\{0, 1\}\}$. *Then*

(1) *$\mathscr{S}_1$ is NP-complete.*

(2) *$\mathscr{S}_2$ is PSPACE-complete.*

Proof. (1) By Lemma 5.15, $\mathscr{S}_1 = \{(G_1, G_2) \mid G_1$ and G_2 are type-3 grammars over $\{0\}$ and $L(G_1) \neq L(G_2)\}$. From Lemma 3.1(1) a regular expression can be converted in polynomial time into a right-linear cfg generating the same language. This right-linear grammar can be converted in polynomial time into an equivalent type-3 grammar. Therefore RINEQ-(over 0) is polynomially reducible to $\mathscr{S}_1$. Since RINEQ-(over 0) is NP-hard (from Theorem 2.11(3)), $\mathscr{S}_1$ is also NP-hard. In [30] it is shown that the inequivalence problem for ndfa's over $\{0\}$ is solvable in nondeterministic polynomial time. Since type-3 grammars are clearly transformable to equivalent ndfa's in polynomial time, we conclude that $\mathscr{S}_1$ is in NP.

(2) The proof of (2) is similar, using Proposition 2.3 to establish that $\mathscr{S}_2$ is PSPACE-hard. From Lemmas 5.15 and 5.6, $\mathscr{S}_2$ is a member of PSPACE. ∎

Next let us consider the more interesting cases, namely, right-linear, linear, and arbitrary context-free grammar structural equivalence. It turns out that the size of the terminal alphabet is irrelevant to the complexity of these problems because arbitrary symbols can be encoded by subtrees using a single terminal symbol.

LEMMA 5.17. *Let $\mathscr{S} = \{(G_1, G_2) \mid G_1$ and G_2 are structurally equivalent right-linear grammars over $\{0\}\}$. Then $\mathscr{S}$ is PSPACE-hard.*

Proof. Let

$$\mathscr{T} = \{G \mid G \text{ is a type-3 grammar and } L(G) = \{1, 2\}^+\}.$$

We will show that $\mathscr{T}$ is polynomially reducible to $\mathscr{S}$. Let $G = (N, \{1, 2\}, P, S)$ be an arbitrary type-3 grammar. Define $G' = (N, \{0\}, P', S)$, where

$$\begin{aligned} P' = &\{A \to 0B \mid A \to 1B \in P\} \\ &\cup \{A \to 00B \mid A \to 2B \in P\} \\ &\cup \{A \to 0 \mid A \to 1 \in P\} \\ &\cup \{A \to 00 \mid A \to 2 \in P\}. \end{aligned}$$

Define $G'' = (\{S''\}, \{0\}, P'', S'')$, where

$$P'' = \{S'' \to 0S'', S'' \to 00S'', S'' \to 0, S'' \to 00\}.$$

Observe that $a_1a_2 \cdots a_n$ is in $L(G)$ if and only if $[0^{a_1}[0^{a_2}[\cdots[0^{a_n}]^n$ is in $L(\mathscr{P}(G'))$. Moreover, $a_1a_2 \cdots a_n$ is in $\{1, 2\}^+$ if and only if $[0^{a_1}[0^{a_2}[\cdots[0^{a_n}]^n$ is in $L(\mathscr{P}(G''))$. This means that $L(G) = \{1, 2\}^+$ if and only if $L(\mathscr{P}(G')) = L(\mathscr{P}(G''))$, which is true if and only if G' and G'' are structurally equivalent. Since $|G'| + |G''| = O(|G|)$ and

the translation is quite obviously executable in polynomial time, we conclude that $\mathscr{T}$ is polynomially reducible to $\mathscr{S}$. But $\mathscr{T}$ is PSPACE-hard, by Proposition 2.4, Lemma 1.10(1), and Lemma 1.8(2). Thus $\mathscr{S}$ is also PSPACE-hard. ∎

LEMMA 5.18. *Let* $\mathscr{S} = \{(G_1, G_2) \mid G_1$ *and* G_2 *are structurally equivalent linear* cfg's*}. Then* $\mathscr{S}$ *is in PSPACE.*

Proof. We can reduce the membership problem for $\mathscr{S}$ to the equivalence problem for right-linear grammars in polynomial time. Let $G_i = (N_i, \Sigma, P_i, S_i)$ for $i = 1, 2$ be the relevant input grammars. Define $G_i' = (N_i, \Sigma \cup \{\$, ¢\}, P_i', S_i)$, in which

$$P_i' = \{A \to \$u¢vB \mid A \to uBv \in P_i\} \cup \{A \to \$u \mid A \to u \in P_i\}.$$

Observe that

$$L(P_i) \subseteq (\$\Sigma^*¢\Sigma^*)^* \cdot \{\$\} \cdot \Sigma^*.$$

Moreover,

$$[u_1[u_2[\cdots[u_n[v]\, w_n] \cdots w_2]\, w_1] \in L(\mathscr{P}(G_i))$$

if and only if

$$\$u_1¢w_1\$u_2¢w_2\$ \cdots \$u_n¢w_n\$v \in L(G_i').$$

It immediately follows that $L(G_1') = L(G_2')$ if and only if $L(\mathscr{P}(G_1)) = L(\mathscr{P}(G_2))$, which is true if and only if G_1 and G_2 are structurally equivalent.

Since $|G_1'| + |G_2'| = O(|G_1| + |G_2|)$ and the translation is clearly polynomial in time, we can apply Lemma 5.6 to see that $\mathscr{S}$ is in PSPACE. ∎

We combine the previous two lemmas, yielding

THEOREM 5.19. *The following sets are PSPACE-complete*:

(a) $\{(G_1, G_2) \mid G_1$ *and* G_2 *are structurally equivalent type-*3 *grammars over* Σ *with* $|\Sigma| \geqslant 2\}$,

(b) $\{(G_1, G_2) \mid G_1$ *and* G_2 *are structurally equivalent right-linear grammars over* Σ *with* $|\Sigma| \geqslant 1\}$,

(c) $\{(G_1, G_2) \mid G_1$ *and* G_2 *are structurally equivalent linear* cfg's *over* Σ *with* $|\Sigma| \geqslant 1\}$.

Proof. The lower bounds on the complexity of these sets are provided by Theorem 5.16 (and Lemma 1.8), Lemma 5.17, and Lemma 5.17, respectively. The upper bound in each case is provided by Lemma 5.18. ∎

We have not been able to find a polynomially space-bounded algorithm for structural equivalence of arbitrary cfg's. The problem is clearly PSPACE-hard and is known to be decidable (Proposition 5.5); but an exact characterization of the complexity of this problem remains open. Related results appear in [33]. We conjecture, however, that no such polynomially space-bounded algorithm exists.

6. Conclusion

We have seen that the complexity of a variety of decidable questions about regular expressions and context-free grammars can be understood in terms of very simple underlying properties. A natural combinatorial structure for the context-free grammars was presented, which complicates several related problems including structural equivalence of context-free grammars and *LL*-equivalence. Several problems were presented whose time and space complexity is provably exponential due to this structure. As a corollary we showed that many predicates on the bounded context-free languages require exponential time, even on nondeterministic machines.

Acknowledgment

The authors are grateful to L. H. Landweber, P. M. Lewis, R. E. Stearns, and J. D. Ullman for helpful discussions.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, Mass., 1974.
2. A. V. Aho and J. D. Ullman, "The Theory of Parsing, Translation and Compiling," Vol. I, Prentice–Hall, Englewood Cliffs, N.J., 1972.
3. B. S. Baker and R. V. Book, Reversal-bounded multi-pushdown machines, *in* "Proceedings of the 13th Annual IEEE Symposium on Switching and Automata Theory" (1972), pp. 207–211.
4. R. V. Book, On languages accepted in polynomial time, *SIAM J. Computing* 1 (1972), 281–287.
5. R. Book, S. Even, S. Greibach, and G. Ott, Ambiguity in graphs and expressions, *in* "Proceedings of the Third Annual Princeton Conference on Information Sciences and Systems" (1969), pp. 345–349.
6. R. S. Cohen and J. A. Brzozowski, On the star-height of regular events, *in* "Proceedings of the Eighth Annual IEEE Symposium on Switching and Automata Theory" (1967), pp. 265–279.
7. S. A. Cook, The complexity of theorem-proving procedures, *in* "Proceedings of the Third Annual ACM Symposium on the Theory of Computing" (1971), pp. 151–158.

8. S. GINSBURG, "The Mathematical Theory of Context-Free Languages," McGraw–Hill, New York, 1966.
9. S. GINSBURG AND E. H. SPANIER, Bounded regular sets, *Proc. Amer. Math. Soc.* **17** (1966), 1043–1049.
10. J. N. GRAY AND M. A. HARRISON, On the covering and reduction problems for context-free grammars, *J. Assoc. Comput. Mach.* **19** (1972), 675–698.
11. S. GREIBACH, A note on undecidable properties of formal languages, *Math. Systems Theory* **2** (1968), 1–6.
12. J. E. HOPCROFT, On the equivalence and containment problems for context-free languages, *Math. Systems Theory* **3** (1969), 119–124.
13. J. E. HOPCROFT AND J. D. ULLMAN, "Formal Languages and Their Relation to Automata," Addison–Wesley, Reading, Mass., 1969.
14. H. B. HUNT III, On the time and tape complexity of languages, Ph.D. Thesis, Cornell University, 1973.
15. H. B. HUNT III, On the time and tape complexity of languages, I, *in* "Proceedings of the Fifth Annual ACM Symposium on the Theory of Computing" (1973), pp. 10–19.
16. H. B. HUNT III AND D. J. ROSENKRANTZ, Computational parallels between the regular and context-free languages, *in* "Proceedings of the Sixth Annual ACM Symposium on the Theory of Computing" (1974), pp. 64–73.
17. H. B. HUNT III, D. J. ROSENKRANTZ, AND T. G. SZYMANSKI, The covering problem for linear context-free grammars, *Theoret. Comput. Sci.*, to appear.
18. O. H. IBARRA, A note concerning nondeterministic tape complexities, *J. Assoc. Comput. Mach.* **19** (1972), 608–612.
19. D. E. KNUTH, A characterization of parenthesis languages, *Inform. Contr.* **11** (1967), 269–289.
20. A. J. KORENJAK AND J. E. HOPCROFT, Simple deterministic languages, *in* "Proceedings of the Seventh Annual IEEE Symposium on Switching and Automata Theory" (1966), pp. 36–46.
21. R. MCNAUGHTON, Parenthesis grammars, *J. Assoc. Comput. Mach.* **14** (1967), 490–500.
22. R. MCNAUGHTON, The loop complexity of regular events, *Inform. Sci.* **1** (1969), 305–328.
23. R. MCNAUGHTON AND S. PAPERT, "Counter-Free Automata," MIT Press, Cambridge, Mass., 1971.
24. A. R. MEYER AND L. J. STOCKMEYER, The equivalence problem for regular expressions with squaring requires exponential space, *in* "Proceedings of the 13th Annual IEEE Symposium on Switching and Automata Theory" (1972), pp. 125–129.
25. M. C. PAULL AND S. H. UNGER, Structural equivalence of context-free grammars, *J. Comput. System Sci.* **2** (1968), 427–468.
26. D. J. ROSENKRANTZ AND R. E. STEARNS, Properties of deterministic top-down grammars, *Inform. Contr.* **17** (1970), 226–256.
27. W. SAVITCH, Relationships between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.* **4** (1970), 177–192.
28. J. I. SEIFERAS, M. J. FISCHER, AND A. R. MEYER, Refinements of the nondeterministic time and space hierarchies, *in* "Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory" (1973), pp. 130–137.
29. L. J. STOCKMEYER, The complexity of decision problems in automata theory and logic, Report TR-133, M.I.T., Project MAC, Cambridge, Mass., 1974.
30. L. J. STOCKMEYER AND A. R. MEYER, Word problems requiring exponential time, *in* "Proceedings of the Fifth Annual ACM Symposium on the Theory of Computing" (1973), pp. 1–9.

31. A. YEHUDAI, private communication.
32. H. B. HUNT III, On the time and tape complexity of languages, I, Cornell University, Department of Computer Science, TR73-156, January 1973.
33. H. B. HUNT III AND T. G. SZYMANSKI, On the complexity of grammar and related problems, *in* "Proceedings of the Seventh Annual ACM Symposium on the Theory of Computing" (1975), pp. 54–65.

Printed in Belgium