Single Tree Grammars[1]

Sheila Greibach[2], Weiping Shi[3] and Shai Simonson[4]

Abstract

A context-free grammar is a single-tree grammar (STG) if every nonterminal symbol has at most one production rule whose right hand side contains nonterminal symbols. Various properties of this class of grammars are studied. Although most properties of STG's are undecidable, every linear STG generates a bounded language and a deterministic context-free language, so most properties of linear STG's are decidable. Finally, we show that STG grammars have natural applications for logic database query languages.

Keywords. Formal languages, context-free grammars, closure properties, decidability, logic databases.

[1]Research of Weiping Shi was supported in part by the Semiconductor Research Corporation under contract 88-DP-109 and in part by the National Science Foundation under grant CCR-8710730. Research of Shai Simonson was supported by the National Science Foundation under grant CCR-8710730.

[2]Department of Computer Science, University of California at Los Angeles, Los Angeles, CA 90024.

[3]Coordinated Science Laboratory and Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801.

[4]Department of Mathematics and Computer Science, Stonehill College, North Easton, MA 02357.

1 Introduction

Every context-free grammar can be translated into an equivalent grammar (by introducing unit productions) such that every nonterminal has at most two rules whose right hand sides contain nonterminals. Our research concerns the class of languages obtained by insisting that each non-terminal in the context-free grammar has at most one such rule, rather than two.

Definition 1.1 A *context-free grammar* is a quadruple $G = (V, T, P, S)$ where V is a finite set of variables (nonterminals), T is a disjoint finite set of terminals, $S \in V$ is the start symbol and P is a finite set of production rules of the form $Z \to y$ where $Z \in V$ and $y \in (V \cup T)^*$. A rule $Z \to y$ is called a *Z-rule* and is *terminating* if y contains only terminals and otherwise *continuing*. The relation $\Rightarrow$ (directly yields) is defined by $uZv \Rightarrow uyv$ if $Z \to y$ is a rule and u, v are strings; and $\overset{*}{\Rightarrow}$ is the transitive reflexive closure of $\Rightarrow$. The language generated by G is

$$L(G) = \{w \in T^* \mid S \overset{*}{\Rightarrow} w\}.$$

We use $Z \to y_1|y_2|\cdots|y_m$ to abbreviate m rules $Z \to y_i, i = 1, \ldots, m$.

Definition 1.2 A *single-tree grammar* (STG) is a context-free grammar such that for every nonterminal Z there is at most one continuing Z-rule. A language is a *single-tree language* (STL) if it is the language generated by a single-tree grammar.

We call it single-tree, because there is a unique (possibly infinite) derivation tree from which parse trees for the words in the language are obtained by either deleting a subtree rooted at a nonterminal or replacing it with a finite number of leaves (terminal symbols). Some simple examples of STL's include Σ^*, $a^n b^n$ and the set of strings with an equal number of a's and b's. On the other hand, neither $a^* + b^*$ nor ww^R is an STL. The reader may wish to verify these facts before proceeding in order to develop some intuition for the STG restrictions.

We investigate the properties of STL's and certain restricted kinds of STL's. In section 2, we introduce a "pumping" style lemma which serves as a tool for proving that certain context-free languages are not

STL's. We also investigate the closure properties of STL's. Section 3 contains a proof that it is undecidable whether an arbitrary context-free language is an STL. In section 4, we prove that most of the decision problems are undecidable for STL's. Section 5 contains a discussion of linear and ultralinear STL's. We show that every linear STL is bounded and deterministic context-free, hence most properties of linear STG's are decidable. We also define k-ultralinear STG's and prove that although there exist 2-ultralinear STG's that do not generate deterministic context-free languages, most properties of k-ultralinear STG's are decidable. Finally in section 6, we introduce an application of STG's in logic database theory.

Throughout the paper, we use the terminology from Ginsburg [3], and Hopcroft and Ullman [6].

2 Intercalation Lemma and Closure Properties

The next lemma gives us a method for proving that a language is not an STL. It is like a pumping lemma in this regard, but strings get intercalated rather than pumped, hence the name. It says that if a single-tree language contains two large enough words, then they can be cut in two such that all four recombinations are in the language.

Lemma 2.1 (Intercalation Lemma for STL's). If L is an STL, then there exists an n such that for all $x, y \in L$, $|x| \geq n$, $|y| \geq n$, we can write $x = uv$ and $y = wz$, where $|u| + |w| > 0$ and $|v| + |z| > 0$, such that $uz, wv \in L$.

Proof. Let G be an STG for L with start symbol S and let n be greater than the maximum length of any terminating rule of G. Consider derivations of x and y from S, with $|x|, |y| \geq n$:

$$S \overset{*}{\Rightarrow} \alpha \overset{*}{\Rightarrow} x$$
$$S \overset{*}{\Rightarrow} \alpha \overset{*}{\Rightarrow} y.$$

Since G is single-tree and $|x|, |y| \geq n$, we assume without loss of generality that both derivations reach the same string α which is not a single nonterminal symbol. Split α into nonempty parts α_1 and α_2 such that $\alpha = \alpha_1\alpha_2$. Again, without loss of generality, we assume that neither

α_1 nor α_2 generates the empty string in both derivations. If $\alpha_1 \overset{*}{\Rightarrow} \epsilon$ in both derivations, then split α_2. Similarly, if $\alpha_2 \overset{*}{\Rightarrow} \epsilon$ in both derivations, then split α_1. If either of these is unsplittable because it is a single nonterminal then just replace it with other nonterminal(s) using its unique continuing rule. Continue until we no longer have a single nonterminal, and then proceed with the splitting.

Let the part of x generated by α_1 be u and the part by α_2 be v. That is, $\alpha_1 \overset{*}{\Rightarrow} u$ and $\alpha_2 \overset{*}{\Rightarrow} v$, and $x = uv$. Similarly, let $y = wz$ such that $\alpha_1 \overset{*}{\Rightarrow} w$ and $\alpha_2 \overset{*}{\Rightarrow} z$. Since α_1 does not generate ϵ in both derivations, we have $|u| + |w| > 0$. Similarly, $|v| + |z| > 0$. Finally, $uz, wv \in L$. □

Many context-free languages (even regular sets) can be shown not to be STL's using this lemma.

Theorem 2.2 STL's are not closed under the following operations:
1) Union.
2) Intersection.
3) Intersection with regular sets.
4) Complementation (even of regular sets).
5) Inverse homomorphism.
6) Substitution by or into regular sets.

Proof.
1) Consider $a^* + b^*$. Let $x = a^n$ and $y = b^n$, from the intercalation lemma, $u = a^i$, $v = a^{n-i}$, $w = b^j$ and $z = b^{n-j}$ where $0 < i + j < 2n$. It is easy to verify that either $uz = a^i b^{n-j}$ or $wv = b^j a^{n-i}$ would contain both a and b, a contradiction.

A very similar proof shows that $L = \{ww^R \mid w \in \{a,b\}^*\}$ is not an STL. Since $a^n, b^n \in L$, then $a^i b^{n-j}$ and $b^j a^{n-i}$ are in L, a contradiction.

2) Consider the language L_1 generated by the following grammar $G = (\{S, A, C\}, \{a, b, c\}, P, S)$ where P consists of

$$\begin{array}{rcl} S & \rightarrow & AC \\ A & \rightarrow & aAb \mid \epsilon \\ C & \rightarrow & Cc \mid \epsilon. \end{array}$$

Then $L_1 = \{a^i b^i c^j \mid i, j \geq 0\}$. We can similarly define an STL $L_2 = \{a^i b^j c^j \mid i, j \geq 0\}$. But $L_1 \cap L_2$ is not even context-free.

3) Since $a^* + b^* = \Sigma^* \cap (a^* + b^*)$, and Σ^* is an STL.

4) Consider the following STG $G = (\{S, A, B, C\}, \{a, b\}, P, S)$, where P consists of

$$\begin{array}{lcl} S & \rightarrow & AB \\ A & \rightarrow & CaCbC \mid \epsilon \\ B & \rightarrow & CbCaC \mid \epsilon \\ C & \rightarrow & CC \mid a \mid b \mid \epsilon. \end{array}$$

But $\overline{L(G)} = a^+ + b^+$ is not an STL, since STL's are closed under union with finite sets and $a^* + b^*$ is not an STL.

5) Let $L = (b + \epsilon)(ab)^*(a + \epsilon)$ and $h(a) = ab$ and $h(b) = ba$, then $h^{-1}(L) = a^+ + b^+$.

6) Let $L = a + b$, $s(a) = a^*$ and $s(b) = b^*$, then $s(L) = a^* + b^*$. $\Box$

Theorem 2.3 STL's are closed under the following operations:

1) Kleene transitive closures ($*$ and $+$).
2) Concatenation.
3) Homomorphism.
4) Reversal.
5) Substitution by finite sets.
6) Union with finite sets.
7) Union with concatenation ($L_1 \cup L_2 \cup L_1L_2 \cup \{\epsilon\}$).

Proof. We omit the straightforward proofs 1) – 6).

7) Given two STG's with start symbols S_1 and S_2, simply add the new start symbol S and rules

$$\begin{array}{ll} S & \rightarrow AB \\ A & \rightarrow S_1 \mid \epsilon \\ B & \rightarrow S_2 \mid \epsilon. \end{array}$$

$\Box$

The result in 7) is extensively used in the rest of the paper to simulate union. It is this result which, in some sense, makes STG's powerful, and hence difficult to understand.

3 Recognition

In this section, we show it is undecidable whether an arbitrary context-free language is an STL. We start with the following lemma.

Lemma 3.1 If L is an STL and for some $w \in \Sigma^*$ and symbols a and b, both $L \cap wa^*$ and $L \cap wb^*$ are infinite, then L must contain a word of the form wz, where z contains both a and b.

Proof. The proof is very similar to the proof of the intercalation lemma. Consider $wa^n \in L$ and $wb^m \in L$, where n and m are sufficiently large. We know that

$$S \overset{*}{\Rightarrow} \alpha \overset{*}{\Rightarrow} wa^n \qquad (1)$$

$$S \overset{*}{\Rightarrow} \alpha \overset{*}{\Rightarrow} wb^m. \qquad (2)$$

Split α into two parts α_1 and α_2 such that $\alpha = \alpha_1\alpha_2$. Without loss of generality, we assume that neither α_1 nor α_2 generates empty strings in both derivations. (The reasoning is the same as we used in the intercalation lemma). Furthermore, we can assume that in derivation (1), $\alpha_1 \overset{*}{\Rightarrow} wa^p$ for some $p \geq 1$, or in derivation (2), $\alpha_1 \overset{*}{\Rightarrow} wb^q$ for some $q \geq 1$. We just need to keep splitting until this occurs. In case we cannot split because the last nonterminal, say X, in α generates ua^n and vb^m, where u and v are suffixes of w, we expand X. This can be done because X must use its unique continuing rule in both derivation (1) and derivation (2).

We now consider all possible cases and show that a word of the form wz is in L, where z contains both a and b. If α_1 generates wa^p in (1) and wb^q in (2), where $p \geq 1$ and $q \geq 1$, then since α_2 does not generate the empty string in both derivations, we can have $\alpha_1\alpha_2$ together generate wz where z contains both a and b. If α_1 generates wa^p in (1) and u in (2), where $p \geq 1$ and u is a prefix of w, then α_2 must generate some b's. Therefore we will have $\alpha_1\alpha_2 \overset{*}{\Rightarrow} wz$, where z contains both a and b. Similarly, if α_1 generates u in (1) and wb^q in (2), where u is a prefix of w and $q \geq 1$, then α_2 must generate some a's. Therefore we also have $\alpha_1\alpha_2 \overset{*}{\Rightarrow} wz$, where z contains both a and b. $\square$

Lemma 3.1 can be strengthened to show that there must be infinitely many words of the form wz, where z contains both a and b. However, we do not need the stronger result.

Theorem 3.2 It is undecidable whether an arbitrary context-free language is an STL.

Proof. We use reduction from the universe problem for context-free languages, and in particular from the problem "$L = \{a,b\}^*$?". Given an arbitrary context-free language $L \subseteq \{a,b\}^*$, let

$$L' = Lc\{a,b\}^* \cup \{a,b\}^* c(a^* \cup b^*)$$

for a new symbol c. We claim $L = \{a,b\}^*$ if and only if L' is an STL.

If $L = \{a,b\}^*$, then $L' = \{a,b\}^* c\{a,b\}^*$ which is an STL.

If $L \neq \{a,b\}^*$, then let $w \notin L$. Assume that L' is an STL. Since both $L' \cap wcb^*$ and $L' \cap wca^*$ are infinite, from Lemma 3.1, L' must contain a word of the form wcy, where y contains both a and b. But this is impossible by the format of L', since $w \notin L$.

Since the universe problem is undecidable for context-free languages, and L' is effectively constructible, the theorem follows. □

4 Undecidable Properties of STL's

4.1 Intersection and Ambiguity

We show it is undecidable to determine if the intersection of two STL's is empty. The proof requires some definitions and lemmas.

The *Post Correspondence Problem* (PCP) for two lists

$$U = (u_1, u_2, \ldots, u_n), Y = (y_1, y_2, \ldots, y_n)$$

of nonempty strings over a vocabulary Σ with at least two symbols, asks whether there is a sequence of integers $i_1, i_2, \ldots, i_m$ such that

$$u_{i_1} u_{i_2} \cdots u_{i_m} = y_{i_1} y_{i_2} \cdots y_{i_m}?$$

The *Modified Post Correspondence Problem* (MPCP) of Hopcroft and Ullman [6] requires that $i_1 = 1$, i.e., the solution is required to start with the first string on each list. We use a variant we call the *Special Post Correspondence Problem* (SPCP) with the four added constraints:

1) u_1 and y_1 start with #, which appears nowhere else,
2) u_n and y_n end with \$, which appears nowhere else,
3) for $i > 1, u_i$ starts with c and y_i does not, and
4) for $i < n, y_i$ ends with c and u_i does not.

Lemma 4.1 SPCP is undecidable.

Proof. See [6] for a proof of the undecidability of MPCP. SPCP can be shown undecidable by the following reduction from MPCP. Let X and Z be the MPCP lists of length n, and $\#, \$$ and c be new symbols. Lists U and Y for SPCP are defined by defining for $1 < i < n+1$, u_{i+1} as x_i with c inserted after each symbol and y_{i+1} as z_i with c inserted before each symbol. In addition, let

$$u_1 = \#u_2,\ u_{n+2} = \$,\ y_{n+2} = c\$$$

and y_1 is formed from $\#z_1$ by inserting c before each symbol except for the first two. We provide an example of the reduction below. Let

$$\begin{array}{ll} x_1 = 1 & z_1 = 111 \\ x_2 = 10111 & z_2 = 10 \\ x_3 = 10 & z_3 = 0 \end{array}$$

be an instance of MPCP. Construct an instance of SPCP:

$$\begin{array}{ll} u_1 = \#1c & y_1 = \#1c1c1 \\ u_2 = 1c & y_2 = c1c1c1 \\ u_3 = 1c0c1c1c1c & y_3 = c1c0 \\ u_4 = 1c0c & y_4 = c0 \\ u_5 = \$ & y_5 = c\$. \end{array}$$

$\square$

The special constraints of SPCP are necessary in the proofs that follow, particularly in Lemma 4.3.

Theorem 4.2 It is undecidable whether the intersection of two STL's is empty.

Proof. The reduction is from SPCP. Let

$$\mathbf{A} = (u_1, u_2, \ldots, u_n),\ \mathbf{B} = (y_1, y_2, \ldots, y_n)$$

be an instance of SPCP. Let $d_1, d_2, \ldots, d_n$ be new symbols. Let G_A be an STG defined by the rules:

$$\begin{array}{lll} S_A & \rightarrow & d_n A u_n \\ A & \rightarrow & A_1 A_2 \cdots A_n \\ A_i & \rightarrow & d_i A u_i \mid \epsilon, \quad \text{for } i = 1, 2, \ldots, n. \end{array}$$

Let G_B be an STG defined by the rules:

$$\begin{array}{lll} S_B & \rightarrow & d_n B y_n \\ B & \rightarrow & B_1 B_2 \cdots B_n \\ B_i & \rightarrow & d_i B y_i \mid \epsilon, \quad \text{for } i = 1, 2, \ldots, n. \end{array}$$

We claim $L(G_A) \cap L(G_B) \neq \emptyset$ if and only if the instance $(\mathbf{A}, \mathbf{B})$ of SPCP has a solution. If the instance of SPCP has a solution then it must be of the form $1 i_1 i_2 \cdots i_m n$. If so, then the word

$$d_n d_{i_m} \cdots d_{i_2} d_{i_1} d_1 u_1 u_{i_1} u_{i_2} \cdots u_{i_m} u_n$$

in $L(G_A)$ is equal to the word

$$d_n d_{i_m} \cdots d_{i_2} d_{i_1} d_1 y_1 y_{i_1} y_{i_2} \cdots y_{i_m} y_n$$

in $L(G_B)$. If $L(G_A) \cap L(G_B) \neq \emptyset$, then let $x \in L(G_A) \cap L(G_B)$. If x is of the form

$$d_n \{d_1, d_2, \ldots, d_n\}^* d_1 \Sigma^+$$

then the instance of SPCP has a solution $1 i_1 i_2 \cdots i_m n$, where

$$x = d_n d_{i_m} \cdots d_{i_2} d_{i_1} d_1 w$$

for some $w \in \Sigma^+$. Lemma 4.5 proves that x must be in the form $d_n \{d_1, d_2, \ldots, d_n\}^* d_1 \Sigma^+$. $\square$

Lemma 4.3 If there exist two sequences $i_1 i_2 \cdots i_r$ and $j_1 j_2 \cdots j_s$ such that $u_{i_1} u_{i_2} \cdots u_{i_r} = y_{j_1} y_{j_2} \cdots y_{j_s}$, then $i_1 = j_1 = 1$ and $i_r = j_s = n$.

Proof. Immediate from the special constraints of SPCP. $\square$

Corollary 4.1 If $u_i d_j$ is a substring of x, where $x \in L(G_A) \cap L(G_B)$, then $i = n$.

Proof. Since the d_j's must match exactly in the two derivations of x (from G_A and G_B), there is a sequence of u's which match a sequence of y's, where i is the last integer in the sequence of u's. $\square$

Lemma 4.4 If $x \in L(G_A) \cap L(G_B)$, then x must be of the form

$$d_n\{d_1, d_2, \ldots, d_n\}^* d_1 \Sigma^+.$$

Proof. It suffices to show that in the derivation $S_A \overset{*}{\Rightarrow} x$, every time the production $A \rightarrow A_1 A_2 \cdots A_n$ is used, $A_i \rightarrow \epsilon$ will be used later for all but at most one of the A_i's. We prove this by contradiction. Assume that i and j are the smallest integers for which the productions $A_i \rightarrow d_i A u_i$ and $A_j \rightarrow d_j A u_j$ are used, where $i < j$. Then $u_i d_j$ is a substring of x where $i < n$. By Corollary 4.4, we have a contradiction. $\square$

For the next theorem we need the following lemmas.

Lemma 4.5 The grammars G_A and G_B are unambiguous.

Proof. Without loss of generality we consider G_A. The reader can verify that every unique sequence of d_i's and u_i's which equals a string x in $L(G_A)$, has a unique parse tree. It remains to be shown that every distinct sequence of d_i's and u_i's generated by G_A gives a different string.

Assume there are two distinct sequences of d_i's and u_i's which both equal x in $L(G_A)$. Each sequence consists of a combination of blocks of d_i's and blocks of u_i's. (A block of d_i's is a consecutive sequence of d_i symbols surrounded by u_i symbols on the left and right ends. A block of u_i's is similarly defined). The indices in the blocks of d_i's must be identical in each sequence since each block contains single symbols. Each block of u_i's in each sequence must equal the same string, but perhaps the indices are different. Consider the leftmost block of u_i's in one sequence whose indices are different from those in the second sequence. It is easy to check by inspecting G_A that if this occurs, then the indices in one block must properly contain the other. This implies that two different strings appear between two d_i symbols of each sequences. Therefore, the two sequences derive different strings, a contradiction. $\square$

A language has the prefix property if it does not contain any word w and a nontrivial extension wu with u nonempty.

Lemma 4.6 The languages $L(G_A)$ and $L(G_B)$ have the prefix property.

Proof. We can construct a deterministic pushdown automata (DPDA) that accepts $L(G_A)$ by empty stack. Clearly a DPDA that accepts by empty stack cannot accept a word and a nonempty extension of the word. $\square$

Theorem 4.7 Ambiguity of STG's is undecidable.

Proof. The proof is very similar to the proof of Theorem 4.2. The reduction is from SPCP for (A, B) as before with a new start symbol S and new rules $S \to S_1S_2$, $S_1 \to S_A \mid \epsilon$ and $S_2 \to S_B \mid \epsilon$. Call this new grammar G_S. We claim that G_S is ambiguous if and only if the SPCP for (A, B) has a solution.

If the SPCP for (A, B) has a solution $i_1i_2 \cdots i_m$, then

$$S \overset{*}{\Rightarrow} S_1 \overset{*}{\Rightarrow} d_{i_m} \cdots d_{i_2}d_{i_1}u_{i_1}u_{i_2} \cdots u_{i_m}.$$

Moreover, this string can be derived from S in a different way:

$$S \overset{*}{\Rightarrow} S_2 \overset{*}{\Rightarrow} d_{i_m} \cdots d_{i_2}d_{i_1}y_{i_1}y_{i_2} \cdots y_{i_m}.$$

If G_S is ambiguous, let x be a string in $L(G_S)$ with two different leftmost derivations. It is impossible, in both derivations, for x to be generated strictly from either S_1 or S_2, with the other generating ϵ. This is because G_A and G_B are both unambiguous, see Lemma 4.6.

It is also impossible, in one derivation, for x to be generated strictly from S_1 (or S_2), and in the other derivation, by a combination of S_1 and S_2. Without loss of generality assume S_1 generates x and w, and S_2 generates v such that $wv = x$, where w and v are not empty. Then w is a proper prefix of x, where w and x are in $L(G_A)$. This is impossible by Lemma 4.7.

Therefore, a string x in $L(G_S)$ with two distinct leftmost derivations has the property that S_1 generates ϵ in one derivation, and S_2 generates ϵ in the other derivation. That is, $x \in L(G_A) \cap L(G_B)$. As in Theorem 4.2, there must be a solution to the SPCP for (A, B). $\square$

4.2 Universe

Theorem 4.8 Given an STL L and vocabulary Σ, it is undecidable whether $L = \Sigma^*$.

Proof. The proof is by a reduction from PCP, and follows the standard idea in the proof for general context-free languages. The difficult part is to construct the appropriate languages using only STL's. For this, the closure of STL's under union with concatenation will be helpful.

Let $\mathbf{A} = (u_1, u_2, \ldots, u_n)$ and $\mathbf{B} = (y_1, y_2, \ldots, y_n)$, be an instance of PCP, where $u_i, y_i \in \Sigma^*$. Let $Q(A)$ and $Q(B)$ be the standard PCP languages defined respectively by the following context-free grammars:

$$\begin{array}{ll} A \rightarrow & d_1Au_1 \mid d_2Au_2 \mid \cdots \mid d_nAu_n \mid d_1u_1 \mid d_2u_2 \mid \cdots \mid d_nu_n \\ B \rightarrow & d_1By_1 \mid d_2By_2 \mid \cdots \mid d_nBy_n \mid d_1y_1 \mid d_2y_2 \mid \cdots \mid d_ny_n \end{array}$$

where $D = \{d_1, d_2, \ldots, d_n\}$ is a set of new symbols not in Σ.

It is well known that $Q(A) \cap Q(B) = \emptyset$ iff the PCP instance $(\mathbf{A}, \mathbf{B})$ has no solution. We construct an STL $T = \overline{Q(A) \cap Q(B)}$, so that $T = \Sigma^*$ iff $Q(A) \cap Q(B) = \emptyset$.

We first show that both $\overline{Q(A)}$ and $\overline{Q(B)}$ are STL's. Then we prove that their union is also an STL.

Without loss of generality, consider $\overline{Q(A)}$. Every string in $\overline{Q(A)}$ is contained in at least one of the sets below, and every string from any set below is in $\overline{Q(A)}$.

1)The empty string.
2)Strings which only contain symbols from Σ.
3)Strings which only contain symbols from D.
4)Strings which contain at least one symbol from both D and Σ, but not of the form $D^*\Sigma^*$.
5)Strings of the form $D^*\Sigma^*$ where there is a mismatch between some d_i and some u_i as we move from the center outward.

Each of the sets is an STL: (1) $\{\epsilon\}$, (2) Σ^+, (3) D^+, (4) $R = (\Sigma + D)^*\Sigma D(\Sigma + D)^*$. Only (5) requires some explanation.

(5) Consider a string of the form $D^*\Sigma^*$ which is not in $Q(A)$. Starting from the rightmost d_j and the leftmost symbol from Σ, we move left from d_j and right from the Σ symbol comparing successive d_j's with successive strings of length u_j. There are three cases to consider.

(a)We reach a d_i and there are $|u_i|$ symbols which do not equal u_i.
(b)We reach a d_i and there are fewer than $|u_i|$ symbols left.
(c)All d_j match successfully, but there are leftover symbols from Σ after all the d_j symbols are exhausted.

Let

$$\begin{aligned} N(i, A) &= \{x \mid x \in \Sigma^*, |x| = |u_i| \text{ and } x \neq u_i\}, \\ M(i, A) &= \{x \mid x \in \Sigma^*, |x| < |u_i|\}, \end{aligned}$$

and let $P(A)$, which is a subset of $Q(A)$, be the STL defined by

$$\begin{aligned} A &\to A_1 A_2 \cdots A_n \\ A_i &\to d_i A u_i \mid \epsilon \quad \text{for} \quad i = 1, 2, \ldots, n. \end{aligned}$$

Then,

Case (a) is contained in: $F_i(A) = D^* d_i P(A) N(i, A)\Sigma^*$, for $i = 1, 2, \ldots, n$.

Case (b) is contained in: $E_i(A) = D^* d_i P(A) M(i, A)$, for $i = 1, \ldots, n$.

Case (c) is contained in: $P(A)\Sigma^+$.

Notice that we use $P(A)$ here instead of $Q(A)$ because $Q(A)$ is not an STL. Furthermore, by introducing $P(A)$, we add no new strings. Any new strings introduced in (5) are already in (4).

$\overline{Q(A)}$ is equal to the union of (1)–(5). In fact, (2) is contained in (5c), and (3) is contained in (5b). Hence,

$$\overline{Q(A)} = \{\epsilon\} \cup R \cup P(A)\Sigma^+ \cup \left(\bigcup_{i=1}^{n} (F_i(A) \cup E_i(A)) \right).$$

The last step is to show that the union of these STL's is also an STL. We claim that

$$\overline{Q(A)} = (P(A)\Sigma^+ \cup \{\epsilon\})(R \cup \{\epsilon\})F(A)E(A).$$

where

$$\begin{aligned} F(A) &= (F_1(A) \cup \{\epsilon\})(F_2(A) \cup \{\epsilon\}) \cdots (F_n(A) \cup \{\epsilon\}), \\ E(A) &= (E_1(A) \cup \{\epsilon\})(E_2(A) \cup \{\epsilon\}) \cdots (E_n(A) \cup \{\epsilon\}). \end{aligned}$$

Clearly this is an STL (because STL's are closed under concatenation), and it contains $\overline{Q(A)}$ (the union of (1)–(5)). We show that it is also contained in $\overline{Q(A)}$.

Any string that contains a non-empty substring in R is not in $Q(A)$. Any string that has a proper prefix in $P(A)\Sigma^+$ is not in $Q(A)$. Any string that contains a non-empty substring in $F(A)$ is not in $Q(A)$. Any string that ends with a non-empty substring in $E(A)$ is not in $Q(A)$.

Hence, $\overline{Q(A)} = (P(A)\Sigma^+ \cup \{\epsilon\})(R \cup \{\epsilon\})F(A)E(A)$.

Similarly, $\overline{Q(B)} = (P(B)\Sigma^+ \cup \{\epsilon\})(R \cup \{\epsilon\})F(B)E(B)$.

Finally, by a similar trick as before:

$$\begin{aligned} T &= \overline{Q(A)} \cup \overline{Q(B)} \\ &= (P(A)\Sigma^+ \cup \{\epsilon\})(P(B)\Sigma^+ \cup \{\epsilon\})(R \cup \{\epsilon\})F(A)F(B)E(A)E(B). \end{aligned}$$

T is an STL, and $T = \Sigma^*$ iff $\overline{Q(A)} \cap \overline{Q(B)} = \emptyset$, iff the PCP instance $(\mathbf{A}, \mathbf{B})$ has a solution. $\square$

Corollary 4.2 Let L_1 and L_2 be arbitrary STG languages and R an arbitrary regular set. The following problems are undecidable.

1) $L_1 = L_2$?
2) $L_1 \supseteq L_2$?
3) $L_1 = R$?
4) $L_1 \supseteq R$?

Proof. Since Σ^* is an STL, fix L_2 and R to be Σ^*, then they are all equivalent to "$L_1 = \Sigma^*$?" $\square$

4.3 Other Properties

The following theorem is a specialization to STL's of the Rice's-theorem type of results established by Greibach for context-free languages [4]. A property is nontrivial on a family $\mathcal{F}$ if there is at least one member of $\mathcal{F}$ for which it holds and at least one member of $\mathcal{F}$ for which it does not hold. The *right derivative* operation takes a language L and string y into $L/y = \{w \mid wy \in L\}$ and if y is a single symbol it is called *right derivative by a single symbol.*

Theorem 4.9 Let $\mathcal{P}$ be any property nontrivial on STL's such that $\mathcal{P}$ is preserved under intersection with regular sets and right derivative by a single symbol and $\mathcal{P}$ is true for every regular set which is also an STL. Then $\mathcal{P}$ is undecidable.

Proof. Let $L_0 \subseteq \Sigma^*$ be an STL where $\mathcal{P}(L_0)$ is false. Given any STL $L \subseteq \Sigma^*$, construct $L_1 = (L_0 c\Sigma^*)^*(\Sigma^* cL)^*$, where $c \notin \Sigma$. Clearly L_1 is an STL.

We claim that $L = \Sigma^*$ iff $\mathcal{P}(L_1)$ is true. If $L = \Sigma^*$, then $L_1 = (\Sigma^* c\Sigma^*)^*$ which is regular and an STL, so $\mathcal{P}(L_1)$ is true. If $L \neq \Sigma^*$, then let x be a string not in L. If $\mathcal{P}(L_1)$ were true, then $\mathcal{P}(L_1 \cap \Sigma^* cx) = \mathcal{P}(L_0 cx)$ would be true. But if $\mathcal{P}(L_0 cx)$ were true, then $\mathcal{P}(L_0)$ would also be true, by preservation of $\mathcal{P}$ under $/a$ and an inductive argument on the length of cx. But $\mathcal{P}(L_0)$ is false, so $\mathcal{P}(L_1)$ must be false. □

Corollary 4.3 Let L be an arbitrary STL. The following problems are undecidable.
1) Is L regular?
2) Is L linear?
3) Is L ultralinear?

Proof. Since all these properties are nontrivial, preserved under intersection with regular sets and under, right derivative by a single symbol, and these properties are true for every regular set which is also an STL.

For readers not familiar with the definition of linear and ultralinear, please refer to Definition 5.1. □

5 Relationship with Other Classes of Context-free Languages

5.1 Linear STL's and Ultralinear STL's

Definition 5.1 A *linear grammar* is one where the right side of every production contains at most one nonterminal symbol. A *k-ultralinear* *grammar* is one where every sentence generated by the grammar contains at most k nonterminals. Linear and k-ultralinear languages are those generated by the respective grammars. A language is ultralinear if it is k-ultralinear for some k.

The ultralinear languages are one generalization of linear languages. These definitions and other generalizations can be found in [3, 6].

Definition 5.2 A linear (k-ultralinear) STL is one that is generated by a grammar that is both linear (k-ultralinear) and an STG.

We show that every linear STL is deterministic context-free, but that there exists a 2-ultralinear STL which is not deterministic context-free.

Definition 5.3 A language L is *bounded* if there exist words $w_1, \ldots, w_n$ such that $L \subseteq w_1^* w_2^* \cdots w_n^*$.

Bounded context-free languages were characterized by Seymour Ginsburg and co-authors who showed that many questions – including ambiguity and equivalence – which are undecidable for general context-free languages are decidable for bounded context-free languages and that it is decidable whether a context-free grammar generates a bounded language. Details of these and other results on bounded languages can be found in [3].

We show that every ultralinear STL is bounded. Since it is undecidable whether an arbitrary STL is ultralinear, there must exist a bounded STL which is not ultralinear.

Theorem 5.1 There exists a 2-ultralinear STL which is not a deterministic context-free language.

Proof. Consider the following 2-ultralinear STG $G = (\{S, A, B\}, \{a, b\}, P, S)$ where P consists of

$$\begin{array}{lcl} S & \rightarrow & AB \\ A & \rightarrow & aAb \mid \epsilon \\ B & \rightarrow & aBbb \mid \epsilon. \end{array}$$

Then, $L(G) = \{a^i b^i a^j b^{2j} \mid i, j \geq 0\}$. If $L(G)$ were deterministic context-free, then since deterministic context-free languages are closed under intersection with regular sets, we would have

$$L(G) \cap \{a^i b^j \mid i, j \geq 0\} = \{a^i b^j \mid j = i \;\text{ or }\; j = 2i\}$$

is deterministic context-free, which is a contradiction. $\square$

The next few lemmas will help us to show that every linear STL is a deterministic context-free language.

Lemma 5.2 The language L generated by grammar $G = (\{S\}, \Sigma, P, S)$ where P consists of

$$S \to \alpha_1 S \alpha_2 \mid \beta$$

is deterministic context-free, where $\alpha_1, \alpha_2, \beta \in \Sigma^*$.

Proof. We show that based on the strings α_1, α_2 and β, L is either recognizable by a DPDA, or L is regular.

A simple-minded way to accept the strings in L is as follows. We read $|\alpha_1|$ symbols at a time, test if they are equal to α_1, and hold the number of α_1's in the stack. If we reach a string equals to β, we start reading α_2's and popping the stack. The problem with this naive method is that one might not be able to distinguish α_1 from a prefix of $\beta\alpha_2^*$.

This problem can be solved by considering whether α_1^n is in $\beta\alpha_2^*$ for some n. If α_1^n is not in $\beta\alpha_2^*$, then we can modify the strategy above to work by looking ahead $n|\alpha_1|$ symbols. If α_1^n is in $\beta\alpha_2^*$ for all n, L must be regular.

We now describe the details. Consider whether

$$\alpha_1^{\lceil |\beta|/|\alpha_1| \rceil + |\alpha_2|} = \beta \alpha_2^{|\alpha_1|} \gamma$$

where γ is a proper prefix of α_2.

1) Not equal: Then use a DPDA which looks ahead $n|\alpha_1|$ symbols where $n = \lceil |\beta|/|\alpha_1| \rceil + |\alpha_2|$. This will allow us to find β and decide where to stop pushing and start popping.

2) Equal: Consider Figure 1 where the symbols of each row are identical.

The arrow comes up from the end of the $\lceil |\beta|/|\alpha_1| \rceil$th copy of α_1. The distance from the arrow to the right end of both strings is $|\alpha_1||\alpha_2|$.

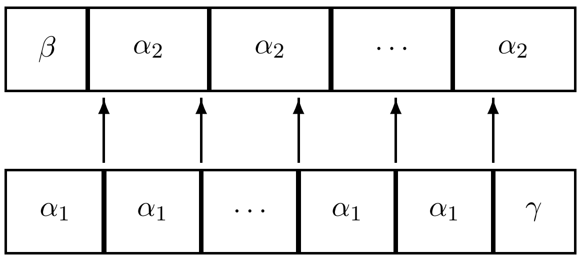

Figure 1: Correspondence between α_1 and α_2.

Therefore the arrow partitions α_2 into 2 parts, the first is equal to γ and let the second part be δ. Then, $\delta\alpha_2^{|\alpha_1|-1}\gamma = \alpha_1^{|\alpha_2|}$.

For any $k \geq 0$, we have

$$\begin{aligned}
\delta\alpha_2^{k|\alpha_1|-1}\gamma &= \delta\,\overbrace{(\alpha_2^{|\alpha_1|-1}\alpha_2)(\alpha_2^{|\alpha_1|-1}\alpha_2)\cdots(\alpha_2^{|\alpha_1|-1}\gamma)}^{k} \\
&= (\delta\alpha_2^{|\alpha_1|-1}\gamma)(\delta\alpha_2^{|\alpha_1|-1}\gamma)\cdots(\delta\alpha_2^{|\alpha_1|-1}\gamma) \\
&= \alpha_1^{k|\alpha_2|}.
\end{aligned}$$

Because $\alpha_1^{\lceil|\beta|/|\alpha_1|\rceil+|\alpha_2|} = \beta\alpha_2^{|\alpha_1|}\gamma$, we have $\beta\gamma = \alpha_1^{\lceil|\beta|/|\alpha_1|\rceil}$. Therefore, each string in L can be represented as follows, where $i, j \geq 0$ and $j < |\alpha_1|$.

$$\begin{aligned}
\alpha_1^{i|\alpha_1|+j}\beta\alpha_2^{i|\alpha_1|+j} &= \alpha_1^{i|\alpha_1|+j}\beta\gamma\delta\alpha_2^{i|\alpha_1|+j-1} \\
&= \alpha_1^{i|\alpha_1|+j+\lceil|\beta|/|\alpha_1|\rceil}\delta\alpha_2^{i|\alpha_1|-1}\alpha_2^{j} \\
&= \alpha_1^{i|\alpha_1|+j+\lceil|\beta|/|\alpha_1|\rceil+i|\alpha_2|}\delta\alpha_2^{j-1} \\
&= (\alpha_1^{|\alpha_1|+|\alpha_2|})^*\alpha_1^{j+\lceil|\beta|/|\alpha_1|\rceil}\delta\alpha_2^{j-1}.
\end{aligned}$$

Thus L is a regular set since there are only finitely many j's. (Note: the case where $j = 0$ can be fixed by the fact that L is regular iff $L\alpha_2$ is regular.) $\square$

Lemma 5.3 Every linear STG with a single nonterminal symbol generates a deterministic context-free language.

Proof. Any linear STG with a single nonterminal symbol has the following production rules:

$$S \to \alpha_1 S \alpha_2 \mid \beta_1 \mid \cdots \mid \beta_n.$$

The proof is by induction on n. Lemma 5.2 is the case for $n = 1$. Assume it is true for $n \leq k$ and consider grammar $S \to \alpha_1 S \alpha_2 \mid \beta_1 \mid \cdots \mid \beta_{k+1}$. Without loss of generality, assume that $|\beta_{k+1}| \geq |\beta_i|$ for $1 \leq i \leq k$.

The argument is similar to the proof of Lemma 5.2. Consider whether

$$\alpha_1^{\lceil |\beta_{k+1}|/|\alpha_1| \rceil + |\alpha_2|} = \beta_{k+1} \alpha_2^{|\alpha_1|} \gamma$$

where γ is a proper prefix of α_2. If equal, then $S \to \alpha_1 S \alpha_2 \mid \beta_{k+1}$ is regular, which implies $S \to \alpha_1 S \alpha_2 \mid \beta_1 \mid \cdots \mid \beta_{n+1}$ is deterministic context-free. If not equal, then for every $1 \leq i \leq k$, consider whether

$$\beta_i \alpha_2^{\lceil |\beta_{k+1}|/|\beta_i| \rceil + |\alpha_2|} = \beta_{k+1} \alpha_2^{|\alpha_2|} \gamma_i$$

where γ_i is a proper prefix of α_2. If none of these are equal, then we can distinguish β_{k+1} from $\beta_1, \ldots, \beta_k$ by looking ahead at most $|\beta_{k+1}| + |\alpha_2|^2$ symbols. If equal for some i, then the language L_i generated by $S \to \alpha_1 S \alpha_2 | \beta_i$ is a prefix of the language L_{k+1} generated by $S \to \alpha_1 S \alpha_2 | \beta_{k+1}$.

Therefore, whenever we find β_i, we remember it in the finite control. After the stack is empty, we expect either the end of input or a string δ, such that $\beta_{k+1} = \beta_i \delta$. $\square$

Theorem 5.4 Every linear STG generates a deterministic context-free language.

Proof. After eliminating useless productions, we have the following production rules:

$$\begin{array}{lcl} S & \to & \alpha_{0,1} X_1 \alpha_{0,2} \mid \beta_{0,1} \mid \cdots \mid \beta_{0,n_0} \\ X_1 & \to & \alpha_{1,1} X_2 \alpha_{1,2} \mid \beta_{1,1} \mid \cdots \mid \beta_{1,n_1} \\ & & \cdots \\ X_k & \to & \alpha_{k,1} X_{k+1} \alpha_{k,2} \mid \beta_{k,1} \mid \cdots \mid \beta_{k,n_k} \end{array}$$

If $X_{k+1} = S$, then we can rewrite the production rules as

$$S \to \alpha_{0,1} \alpha_{1,1} \cdots \alpha_{k,1} S \alpha_{0,2} \alpha_{1,2} \cdots \alpha_{k,2} \mid \beta_{0,1} \mid \cdots$$

and by Lemma 5.3, it generates a deterministic context-free language.

Otherwise, if $X_{k+1} = X_i$, for some $1 \leq i \leq k$, then the language is equal to $uL(X_i)v$ plus some finite sets, where $L(X_i)$ generates a deterministic context-free language, and u and v are fixed strings. This language is also deterministic context-free.

Finally, if X_{k+1} is a new symbol then the language is finite. □

Theorem 5.5 Every ultralinear STG generates a bounded language.

Proof. From the proof in Theorem 5.4, the language generated by linear STG's can be characterized as

$$F \cup \{uv^n wx^n z \mid n \geq 0, w \in W\}$$

for finite sets F and W and fixed strings u, v, x and z. Every language in the above characterization is bounded. Furthermore, it is not hard to see that ultralinear-STL's are just the closure of linear-STL's under concatenation, and union with finite sets. Therefore, they are also bounded. □

Theorem 5.6 Let L_1 and L_2 be arbitrary ultralinear STG languages. The following problems are decidable.

1) $L_1 = \Sigma^*$?

2) $L_1 = L_2$?

3) Is L_1 inherently ambiguous?

Proof. The universe problem 1) is trivial for ultralinear-STL's since Σ^* is not bounded if $|\Sigma| \geq 2$. If $|\Sigma| = 1$, the grammar generates a regular set.

The proofs for 2) and 3) are immediate from Theorem 5.5, and the fact that the equivalence and ambiguity problems are decidable for bounded languages [3]. □

We conjecture that it is decidable whether an arbitrary context-free language is a linear STL, and whether it is an ultralinear STL. It should be pointed out that this does not conflict with Corollary 4.12, because a linear STL is a language generated by a grammar that is both linear and an STG.

5.2 Regular Sets

There are regular sets which are not STL's. A simple example is $a^* + b^*$. We would like to characterize exactly which regular sets are STL's. A grammar is *left linear* if all continuing rules are of the form $Z \to Yu$ where Y is a nonterminal symbol and u is a terminal string, and *right linear* if all continuing rules are of the form $Z \to uY$. We consider a grammar to be regular if it is either left linear or right linear. The following theorem characterizes which regular languages can be generated by grammars that are simultaneously STG's and regular.

Theorem 5.7 Every left linear STG generates a regular set of the form $F + Wv^*u$, and every right linear STG generates a regular set of the form $F + uv^*W$, where F and W are finite sets, u and v are fixed strings.

Proof. Similar to the proof of Theorems 5.4 and 5.5. $\square$

Of course, there are languages which are both regular and STL but not generated by any grammar that is both regular and STL. A simple example is a^*b^*.

We conjecture that the intersection of regular sets and STL's is the smallest family of languages containing the finite sets which is closed under concatenation, Kleene closure, and union with finite sets. (It is not hard to see that you get strictly less if union with finite sets is not allowed. For example, consider $a^*b^* + cb^* = b^*(a^* + c)$.)

5.3 Inherent Ambiguity

Theorem 5.8 There is a 4-ultralinear STG that generates an inherently ambiguous context-free language.

Proof. Consider $G = (\{S, A, B, C, D\}, \{a, b, c\}, P, S)$ where P is defined as

$$\begin{aligned} &S \to ABCD \\ &A \to aAb \mid \epsilon \\ &B \to Bc \mid \epsilon \\ &C \to Ca \mid \epsilon \\ &D \to bDc \mid \epsilon. \end{aligned}$$

Clearly the grammar is 4-ultralinear. Furthermore, it generates the language $a^n b^n c^* a^* b^m c^m$, where $m, n \geq 0$, which is inherently ambiguous. The idea is that one never knows how to parse the string $a^n b^n c^n$. $\square$

We conjecture that all 3-ultralinear STG's generate unambiguous languages.

(Note: If an STL L is contained in $a_1^* \cdots a_k^*$ for distinct symbols a_i, then L is unambiguous. The proof above gives a counterexample when the symbols are not distinct, since the language above is contained in $a^* b^* c^* a^* b^* c^*$.)

5.4 Dyck Sets

A 1-sided Dyck set or Dyck language is a language with k types of balanced parenthesis, see Hopcroft and Ullman [6]. A 2-sided Dyck set is similar except that the parenthesis can be balanced in either direction, i.e. close-open or open-close.

For example, the set of strings with an equal number of a's and b's, is the two-sided Dyck set with one kind of parentheses.

Dyck sets are important because they give examples of hard context-free languages. It is known that every context-free language L is $h(L_D \cap R)$ where h is a homomorphism, R is a regular set, and L_D is a 1-sided Dyck set [6]. Also, see Greibach [5], where a variation of a Dyck set is proved to be as hard to parse as any context-free language. The result in [5] implies that there exist STL's which are as hard to parse as any context-free language.

It is an open question whether there is a characterization of STL's by Dyck sets. A possible conjecture is that STL's are the smallest family of languages which contains the finite sets, 1-sided and 2-sided Dyck sets; and is closed under concatenation, Kleene closure, and substitution by finite sets.

Theorem 5.9 All 1-sided and 2-sided Dyck sets are STL's.

Proof. The 1-sided Dyck set with k kinds of parentheses is generated by the STG:

$$\begin{aligned} S \rightarrow &\quad S A_1 A_2 \cdots A_k \mid \epsilon \\ A_i \rightarrow &\quad a_i S b_i \mid \epsilon \quad \text{for } i = 1, 2, \ldots, k \end{aligned}$$

and the 2-sided Dyck set with k kinds of parentheses is generated by the STG:

$$\begin{array}{ll} S \rightarrow & SA_1A_2\cdots A_kB_1B_2\cdots B_k \mid \epsilon \\ A_i \rightarrow & a_iSb_i \mid \epsilon \quad \text{for } i=1,2,\ldots,k \\ B_i \rightarrow & b_iSa_i \mid \epsilon \quad \text{for } i=1,2,\ldots,k. \end{array}$$

The details of why these grammars generate the claimed sets are left to the reader. We provide the details for a special case below. □

Theorem 5.10 The set of strings with an equal number of a's and b's is an STL.

Proof. Consider the following STG:

$$\begin{array}{lll} S & \rightarrow & SAB \mid \epsilon \\ A & \rightarrow & aSb \mid \epsilon \\ B & \rightarrow & bSa \mid \epsilon. \end{array}$$

Clearly, this language is a subset of equal number of a's and b's, since every production contains the same number of a's and b's.

To see that every string x with equal number of a's and b's is in L, use induction on the length of x. The case $|x| = 0$ is trivially true. Assume $x \in L$, for all x which have an equal number of a's and b's, and $|x| \leq 2(n-1)$. When $|x| = 2n$, assume without loss of generality the first symbol of x is a. Then look for the b which *matches* the a. This is the first b moving from left to right for which the number of of a's and b's in the string up to that point are equal.

If the b is not the last symbol, then we are done since then $x = uv$ where both u and v are equal number of a's and b's, where $|u|, |v| \leq 2(n-1)$, so x can be generated by a concatenation of derivations for u and v. (Simply let $S \overset{*}{\Rightarrow} S(AB)^k$ for sufficiently large k, and then use S for u and $(AB)^k$ for v.) If the b is the last symbol, then we can write $x = ayb$. Since $S \Rightarrow A \Rightarrow aSb$ and by the induction hypothesis, $y \in L$, we know $x \in L$. □

6 Applications

Besides the theoretical motivation of understanding the structure of restricted context-free grammars, there are applications of STG's in logic database theory. One of these applications is for a class of logic queries called H queries defined by Chandra and Harel [2]. We review some terminology from that paper. For more information about logic database query processing, see [2, 7].

Let U be a countable universal domain. A relational database of type $\bar{a} = (a_1, \ldots, a_k), k \geq 0, a_i \geq 0$, is a tuple $B = (D, R_1, R_2, \ldots, R_k)$ where D is a finite nonempty subset of U, and for $1 \leq i \leq k$, $R_i \subseteq D^{a_i}$. A query Q of type $\bar{a} \rightarrow b$ is a partial function from the set of databases of type $\bar{a}$ to subsets of U^b such that $Q(B) \subseteq D^b$ whenever it is defined.

In order to define the language H, we call elements of U constants and assume we have an unlimited supply of terminal relational symbols $R_0, R_1, \ldots$, and nonterminal relational symbols $S_0, S_1, \ldots$, and variables $x_1, x_2, x_3, \ldots$ We assume that "=" is a special binary terminal relation symbols. For an n-ary relation symbol S (resp. $R, =$), and variables $x_1, \ldots, x_n$, $S(x_1, \ldots, x_n)$ (resp. $R(x_1, \ldots, x_n), x_1 = x_2$) is a nonterminal (resp. terminal) atomic formula. A clause C is an expression of the form $A : -B_1, \ldots, B_n$, where A is a nonterminal atomic formula, and $B_1, \ldots, B_n$ are atomic formulas. A program P of H is a finite set of clauses.

The intuition behind a program P can be described as follows. P represents the conjunction of its clauses. Each clause $A : -B_1, \ldots, B_n$ is taken to stand for the universal closure of the implication $(B_1 \wedge B_2 \wedge \cdots \wedge B_n) \supset A$, and the set of tuples in a nonterminal relation S is taken to be those d appearing in any atomic formula of the form $S(d)$ whose truth is a consequence of P.

Query Q_1 contains query Q_2, written $Q_1 \subseteq Q_2$, if for all databases D, $Q_1(D) \supseteq Q_2(D)$. Queries Q_1 and Q_2 are equivalent, written $Q_1 = Q_2$, if for all databases D, $Q_1(D) = Q_2(D)$.

Let H_1 be the set of H programs with the restriction that each nonterminal relation has at most one clause whose right hand side contains a nonterminal atomic formula. The undecidability of linearity, equivalence and containment problems about H_1 logic programs follows from the undecidability of these problems for STG's. Here is a proof of one of the properties. Other properties are similar.

Theorem 6.1 Given H_1 queries Q_1 and Q_2, it is undecidable whether $Q_1 = Q_2$.

Proof. Given two STG's G_1 and G_2, construct the following logic programs: For each nonterminal symbol X_i, construct a nonterminal relation $S_i(x, y)$. For each terminal symbol x_j, construct a terminal relation $R_j(x, y)$. For a production rule such as $X_1 \to X_2 X_1 x_3$, construct a clause $S_1(t_1, t_4) : -S_2(t_1, t_2) S_1(t_2, t_3) R_3(t_3, t_4)$. For an ϵ-production $X_i \to \epsilon$ we have $S_i(t_1, t_2) : -(t_1 = t_2)$. Now it is easy to see that the two queries so constructed are equivalent if and only if $L(G_1) = L(G_2)$. $\square$

7 Conclusions

We have analyzed the class of STL's, the languages that are generated by context-free grammars where every nonterminal symbol has at most one production rule that contains a nonterminal. This class is interesting from its own theoretical standpoint, and for applications in logic database theory.

We proved the intercalation lemma, a method for showing that a language is not an STL. Future research can focus on finding stronger versions of this lemma, taking into account the ideas in Boasson [1].

The STL's are in general a difficult class of languages to analyze. We showed that most properties of STL's are undecidable. We also proved that the Dyck sets are all STL's, implying that there are STL's that are very difficult to parse.

We gave a characterization for linear and ultralinear STL's. We proved that linear STG's all generate deterministic context-free languages; there is a 2-ultralinear STG which does not generate a deterministic context-free language; but ultralinear STL's are all bounded. This implied that most questions are decidable for these sets. We conjectured that it is decidable whether an arbitrary context-free language is a linear STL, and whether it is an ultralinear STL.

We showed that there is a 4-ultralinear STG which generates an inherently ambiguous language, and conjectured that all 3-ultralinear STG's generate unambiguous languages.

We characterized the (simultaneous) regular STG languages, and conjectured a characterization for the intersection of regular sets and STL's. This characterization is interesting because it is essentially just the regular sets without union (only concatenation, Kleene star, and union with finite sets is allowed).

It is an open question whether one can characterize the bounded STL's. Similarly, it is an open question whether one can characterize the STL's using Dyck sets.

Finally, one might consider a generalization of STL's. Parse trees for the words in an STL are obtained by taking a unique (possibly) infinite derivation tree; and deleting subtrees rooted at nonterminals, or replacing them with a finite number of leaves (terminal symbols). One can consider other languages where again we have a unique infinite tree plus truncation and substitution rules, but we define another (still highly computable and patterned) scheme to generate the infinite tree.

Acknowledgments The authors thank Michael Harrison and Juris Hartmanis for their early comments and encouragement, and Moshe Vardi for information about applications of STL's.

References

[1] L. Boasson. Two iteration theorems for some families of languages. *J. Computer and System Sciences* **7**:6, Dec. 1973, 583–596.

[2] A. K. Chandra and D. Harel. Horn Clauses, Queries and Generalizations. *J. Logic Programming* **1**, 1985, 1–15.

[3] S. Ginsburg. *The Mathematical Theory of Context-Free Languages.* McGraw-Hill, 1966.

[4] S. A. Greibach. A note on undecidable properties of formal languages. *Math. Systems Theory* **2**:1, 1968, 1–6.

[5] S. A. Greibach. The hardest context-free language. *SIAM J. Computing* **2**:4, 1973, 304–310.

[6] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[7] O. Shumueli. Decidability and expressiveness aspects of logic queries. *Proc. ACM SIGMOD-SIGACT Symp. Principles of Database Systems,* 1987, 237–249.