2014

Ordinary Generating Functions of Context-Free Grammars

Tanner Swett
Grand Valley State University

Edward Aboufadel
Grand Valley State University, aboufade@gvsu.edu

ORDINARY GENERATING FUNCTIONS OF CONTEXT-FREE GRAMMARS

TANNER SWETT AND EDWARD ABOUFADEL, ADVISOR

1. Introduction

A *context-free grammar* is a mathematical construct that classifies strings (sequences of symbols) as either "valid" or "invalid", by specifying a set of "production rules" which determine the ways in which valid strings can be formed. A "language" (that is, a set of strings) generated by a context-free grammar is known as a *context-free language*.

According to Sipser ([4] p. 99):

> Context-free grammars were first used in the study of human languages. One way of understanding the relationship of terms such as *noun*, *verb*, and *preposition* and their respective phrases leads to a natural recursion because noun phrases may appear inside verb phrases and vice versa. Context-free grammars can capture important aspects of these relationships.

Although context-free grammars were initially used to study human language, this paper investigates context-free languages in a more theoretical context. Specifically, we are interested in "counting sequences": given a language, its counting sequence is a sequence of natural numbers stating how many strings of each length are elements of the language.

The *ordinary generating function* of a sequence is the power series whose coefficients are the elements of the sequence. This paper investigates the properties of ordinary generating functions of counting sequences of context-free languages.

We will begin by defining a context-free grammar and an ordinary generating function, and discussing some examples. We will then discuss the Chomsky–Schützenberger Theorem, an important theorem about the generating functions of context-free languages. Finally, we will extend the notion of a context-free grammar by defining *integer-labeled context-free grammars*, where each rule has a (possibly negative) label indicating the multiplicity of that rule. We will prove that an extended version of the Chomsky–Schützenberger Theorem also holds for integer-labeled context-free grammars.

2. Context-free grammars and their generating functions

2.1. **Context-free grammars.** We will now define a **context-free grammar**. This definition is based on the definition of a context-free grammar found in Sipser ([4] p. 100).

Date: June 3, 2014.

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Arts at Grand Valley State University.

The purpose of a CFG is to define a *language*, that is, a set of "words" (strings, or sequences of symbols). Words can be formed by applying "production rules"; a word is in the language if and only if there is some way of forming it by applying these rules, starting with the "start symbol". A language which can be defined by a context-free grammar is a *context-free language*.

The main component of a context-free grammar is the set of its production rules, which are of the form $A \to x$, where A is a "nonterminal symbol" and x is a string of symbols (either "terminal" or "nonterminal"). The meaning of this rule is that while forming a word, you may replace the symbol A with the string x. A *terminal symbol* is one that will appear in the final string, and a *nonterminal symbol* is one that must be replaced using a production rule.

Here is an example of a context-free grammar from Sipser [4]:

$$A \to 0A1$$
$$A \to B$$
$$B \to \#$$

One string that can be generated by this grammar is 000#111. This string can be generated using the following steps:

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

We start with the string A, and replace A with $0A1$ three times; then we replace A with B, and, finally, we replace B with #.

Formally, a context-free grammar is a tuple (V, Σ, R, S), where

- V is a finite set, called the *variables* or the *nonterminal symbols*;
- Σ is a finite set (disjoint from V), called the *terminal symbols* or the *alphabet*;
- R is a finite set of *rules* of the form $A \to x$, where A is a nonterminal symbol and x is a string of symbols; and
- S is an element of V, the *start symbol*.

We say that a string s *yields* a string t (written $s \Rightarrow t$) if and only if s can be written in the form uAv, and t can be written in the form uwv, such that $A \to w$ is a rule of the grammar. Given a sequence $s_1, s_2, \ldots, s_k$, for $k \geq 1$, we say that this sequence is a *derivation* from s_1 to s_k (written $s_1 \overset{*}{\Rightarrow} s_k$) if $s_1 \Rightarrow s_2 \Rightarrow \cdots \Rightarrow s_k$.

Finally, the **language** of the context-free grammar is defined as the set of all strings s such that $S \overset{*}{\Rightarrow} s$.

2.2. **Generating functions for context-free grammars.** A *formal power series* is defined as a series of the form $\sum_{j=0}^{\infty} a_j x^j$, where $\{a_j\}$ is a sequence of complex numbers. (However, all of the formal power series used in this paper have only integer coefficients.) An example of a formal power series is

$$1 + x + 2!\, x^2 + 3!\, x^3 + 4!\, x^4 + \cdots .$$

We do not particularly care about the fact that this series diverges for all values of x other than 0; we are more interested in the series itself than in the values it takes on. However, formal power series are often identified with the functions that they define; for example, the expression $\frac{1}{(1-x)}$ is understood to represent the series $1 + x + x^2 + x^3 + \cdots$.

It turns out that formal power series can be useful in studying context-free grammars. The *ordinary generating function* of a sequence $\{a_j\}$ is defined as the series $\sum_{j=0}^{\infty} a_j x^j$. We define the **univariate ordinary generating function** of a language as the ordinary generating function of its "counting sequence". That is, the univariate ordinary generating function of a language L is defined as

$$\mathbf{S}_L(x) = \sum_{j=0}^{\infty} \Big|\{s : s \text{ is a string in } L \text{ of length } j\}\Big| \cdot x^j.$$

Sometimes, we are interested not only in the length of each string in the language, but also in the number of instances of each character occurring in each string. Therefore, we also define the **multivariate ordinary generating function** of a language as follows. Suppose that a language L is defined over an alphabet such as $\{a, b, c\}$. The multivariate OGF of L is

$$\mathbf{S}_L(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty}\sum_{k=0}^{\infty} \Big|\{s : s \in L, s \text{ contains exactly } i \text{ } a\text{'s, } j \text{ } b\text{'s, and } k \text{ } c\text{'s}\}\Big| \cdot \mathbf{a}^i\mathbf{b}^j\mathbf{c}^k.$$

This definition can be extended to alphabets of any size.

2.3. **Example: a language from Du & Ko.** We will consider the multivariate generating function of the context-free language $\{a^m b^n c^p : m + 2n - p \geq 0\}$, taken from Du and Ko ([2] p. 101). (It is not necessarily obvious from the definition that this language is context-free, but Du and Ko show that it is.) In order to do so, we will separately consider the cases $p \leq m$, $m \leq p$, and $m = p$.

First, we consider the case where $p \leq m$. In this case, it is always true that $m - p \geq 0$, and therefore $m + 2n - p \geq 0$. Thus, the only constraint on n and p is that they be nonnegative integers, and the only constraint on m is that it be an integer greater than or equal to p. So this case is represented by the following multivariate generating function:

$$\mathbf{S}_1(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \sum_{n=0}^{\infty}\sum_{p=0}^{\infty}\sum_{m=p}^{\infty} \mathbf{a}^m\mathbf{b}^n\mathbf{c}^p$$

We perform a change of variables, letting $q = m - p$, and replacing m with $q + p$:

$$\begin{aligned}
&= \sum_{n=0}^{\infty}\sum_{p=0}^{\infty}\sum_{q=0}^{\infty} \mathbf{a}^{q+p}\mathbf{b}^n\mathbf{c}^p \\
&= \sum_{n=0}^{\infty}\sum_{p=0}^{\infty}\sum_{q=0}^{\infty} \mathbf{a}^q\mathbf{b}^n(\mathbf{ac})^p \\
&= \frac{1}{1-\mathbf{a}} \cdot \frac{1}{1-\mathbf{b}} \cdot \frac{1}{1-\mathbf{ac}} \\
&= \frac{1}{(1-\mathbf{a})(1-\mathbf{b})(1-\mathbf{ac})}
\end{aligned}$$

Second, we consider the case where $m \leq p$. Here, the constraint $m + 2n - p \geq 0$ can be written as $p \leq m + 2n$. Thus, here, the only constraint on m and n is that they be nonnegative integers, and the constraint on p is that it be an integer

between m and $m + 2n$ inclusive. So this case is represented by the following function:

$$\mathbf{S}_2(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \sum_{p=m}^{m+2n} \mathbf{a}^m \mathbf{b}^n \mathbf{c}^p$$

We will perform a change of variables, representing n and p in terms of new variables x and y. We will treat m as a constant.

Let $x = 2n - (p - m)$ and $y = p - m$. Notice that the constraint $m + 2n - p \geq 0$ can be rewritten as $2n - (p - m) \geq 0$, or $x \geq 0$, and the constraint $m \leq p$ can be rewritten as $p - m \geq 0$, or $y \geq 0$. Within these constraints, we can make x and y take on any integer values with $x + y$ even, by letting $n = \frac{x+y}{2}$ and $p = y + m$. Therefore, the constraints here are that m be any nonnegative integer, and that x and y be nonnegative integers with $x + y$ even.

With this change of variables, we can rewrite this summation as follows:

$$\begin{aligned}
\mathbf{S}_2(\mathbf{a}, \mathbf{b}, \mathbf{c}) &= \sum_{m=0}^{\infty} \left(\sum_{\text{even } x=0}^{\infty} \sum_{\text{even } y=0}^{\infty} \mathbf{a}^m \mathbf{b}^{(x+y)/2} \mathbf{c}^{y+m} + \sum_{\text{odd } x=1}^{\infty} \sum_{\text{odd } y=1}^{\infty} \mathbf{a}^m \mathbf{b}^{(x+y)/2} \mathbf{c}^{y+m} \right) \\
&= \sum_{m=0}^{\infty} \left(\sum_{\text{even } x=0}^{\infty} \sum_{\text{even } y=0}^{\infty} \mathbf{a}^m \mathbf{b}^{(x+y)/2} \mathbf{c}^{y+m} + (\mathbf{bc}) \sum_{\text{even } x=0}^{\infty} \sum_{\text{even } y=0}^{\infty} \mathbf{a}^m \mathbf{b}^{(x+y)/2} \mathbf{c}^{y+m} \right) \\
&= \sum_{m=0}^{\infty} \left((1 + \mathbf{bc}) \sum_{\text{even } x=0}^{\infty} \sum_{\text{even } y=0}^{\infty} \mathbf{a}^m \mathbf{b}^{(x+y)/2} \mathbf{c}^{y+m} \right) \\
&= \sum_{m=0}^{\infty} \left((1 + \mathbf{bc}) \sum_{x=0}^{\infty} \sum_{y=0}^{\infty} \mathbf{a}^m \mathbf{b}^{x+y} \mathbf{c}^{2y+m} \right) \\
&= \sum_{m=0}^{\infty} \left((1 + \mathbf{bc}) \sum_{x=0}^{\infty} \sum_{y=0}^{\infty} (\mathbf{ac})^m \mathbf{b}^x (\mathbf{bc}^2)^y \right) \\
&= \frac{(1 + \mathbf{bc})}{(1 - \mathbf{ac})(1 - \mathbf{b})(1 - \mathbf{bc}^2)}
\end{aligned}$$

Third and finally, we consider the case where $m = p$. In this case (like the first case), it is always true that $m - p \geq 0$, and therefore $m + 2n - p \geq 0$. Thus, the only constraints on the quantity $m = p$ and the quantity n is that they be nonnegative integers. Thus, this case is represented by the multivariate generating function

$$\mathbf{S}_\Omega(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \mathbf{a}^m \mathbf{b}^n \mathbf{c}^m = \frac{1}{(1 - \mathbf{ac})(1 - \mathbf{b})}.$$

Since the third case describes the overlap between the first two cases, the multivariate generating function for the original language is the sum of the first two functions, minus the third:

$$\begin{aligned}\mathbf{S}(\mathbf{a},\mathbf{b},\mathbf{c}) &= \mathbf{S}_1(\mathbf{a},\mathbf{b},\mathbf{c}) + \mathbf{S}_2(\mathbf{a},\mathbf{b},\mathbf{c}) - \mathbf{S}_\Omega(\mathbf{a},\mathbf{b},\mathbf{c}) \\ &= \frac{1}{(1-\mathbf{a})(1-\mathbf{b})(1-\mathbf{ac})} + \frac{(1+\mathbf{bc})}{(1-\mathbf{ac})(1-\mathbf{b})(1-\mathbf{bc}^2)} - \frac{1}{(1-\mathbf{ac})(1-\mathbf{b})} \\ &= \frac{1+\mathbf{bc}-\mathbf{abc}-\mathbf{abc}^2}{(1-\mathbf{ac})(1-\mathbf{a})(1-\mathbf{b})(1-\mathbf{bc}^2)}.\end{aligned}$$

2.4. **The Dyck language.** The Dyck language is the "language of correctly matched parentheses": its strings are those admitting a one-to-one correspondence between opening parentheses and closing parentheses, such that no closing parenthesis precedes its corresponding opening parenthesis. (An example of such a string is `(())()`.) The Dyck language has the context-free grammar $S \to aSbS \mid \epsilon$ (using a and b to represent the parentheses).

We will find the univariate ordinary generating function $\mathbf{S}$ of the Dyck language. Since the language admits exactly one string of length zero, the constant term of $\mathbf{S}(x)$ is 1. For $n \geq 2$, each valid string of length n consists of an a, a b, and two valid strings of length p and q, such that $p+q+2 = n$. This means that the coefficient of the x^n term of $\mathbf{S}(x)$ is the sum of the coefficients of x^p and x^q, where $p+q = n-2$.

Notice, however, that the coefficient of the x^{n-2} term in the function $\mathbf{S}(x)^2$ is *also* the sum of the coefficients of x^p and x^q in $\mathbf{S}(x)$, where $p+q = n-2$. Therefore, the coefficient of the x^n term of $\mathbf{S}(x)$, for $n \geq 2$, is simply the coefficient of the x^{n-2} term of $\mathbf{S}(x)^2$, or, in other words, the coefficient of the x^n term of $x^2\mathbf{S}(x)^2$.

We have shown that the constant term of $\mathbf{S}(x)$ is 1, and that for $n \geq 2$, the coefficient of the x^n term of $\mathbf{S}(x)$ is the same as the coefficient of the x^n term of $x^2\mathbf{S}(x)^2$. Since the constant term of $x^2\mathbf{S}(x)^2$ is 0, this means that

$$\mathbf{S}(x) = x^2\mathbf{S}(x)^2 + 1.$$

This equation is quadratic in $\mathbf{S}(x)$, meaning that it has the two solutions

$$\mathbf{S}(x) = \frac{1 \pm \sqrt{1-4x^2}}{2x^2}.$$

Of course, of these two solutions, only one is actually the function $\mathbf{S}$. If we choose $+$ for $\pm$, the resulting series contains negative coefficients, so the correct solution must be the one with $-$. Thus, we have found that

$$\mathbf{S}(x) = \frac{1 - \sqrt{1-4x^2}}{2x^2} = 1 + x^2 + 2x^4 + 5x^6 + 14x^8 + \cdots.$$

This series does, in fact, describe the number of strings of each length the Dyck language admits. For example, the language admits exactly 5 strings of length 6: namely, *aaabbb*, *aababb*, *aabbab*, *abaabb*, and *ababab*. Likewise, the language admits 14 strings of length 8, and so on.

Admittedly, this expression for $\mathbf{S}(x)$ looks somewhat uninspiring; it does not have any obvious connections to interesting properties of the Dyck language. It is interesting to note, however, that the power series expansion of $\sqrt{1-4x^2}$ has only integer coefficients (as does the power series expansion of $\sqrt{1-4x}$).

2.5. **The Chomsky–Schützenberger Theorem.** Notice that the equation that we came up with for the univariate ordinary generating function of the Dyck language looks quite similar to the context-free grammar for the Dyck language:

$$S \to aSbS \mid \epsilon$$

$$\mathbf{S}(x) = x^2\mathbf{S}(x)^2 + 1$$

It looks as if the equation for $\mathbf{S}(x)$ could have been obtained from the grammar simply by replacing S with $\mathbf{S}(x)$, a and b with x, ϵ with 1, the $\mid$ operator with the $+$ operator, and the $\to$ symbol with the $=$ symbol. Indeed, this is, in fact, the case: the Chomsky–Schützenberger Theorem states that given any *unambiguous* context-free grammar, it is possible to transform the grammar in this way into a set of polynomial equations describing its (univariate or multivariate) ordinary generating function. [1]

The general idea is that it is possible to find an ordinary generating function for each of the nonterminal symbols in a grammar, and the grammar's production rules describe how these functions relate to each other. As expected, when a nonterminal symbol has multiple alternative rules, these alternatives are simply added together. The important realization is that, because of the way that multiplication of power series works, concatenation of two components corresponds to multiplication of the corresponding generating functions.

Later in this paper, we present a proof of an extended version of the Chomsky–Schützenberger theorem.

2.6. **Examples from Flajolet.** The contrapositive form of the Chomsky–Schützenberger Theorem is also useful. Namely: given a context-free language, if its ordinary generating function cannot be described by any set of polynomial equations (i.e. the function is not algebraic), then the language has no unambiguous context-free grammar (the language is *inherently ambiguous*).

Flajolet [3] gives many examples of such languages. One of these examples is the *Goldstine language* $G_{\neq}$, consisting of all strings of the form $a^{n_1}ba^{n_2}ba^{n_3}b\cdots a^{n_p}b$, such that for some j, $n_j \neq j$. Flajolet shows that the generating function of this language is an algebraic function minus the function

$$B(z) = \sum_{n=1}^{\infty} z^{n(n+1)/2-1},$$

which is a "lacunary series", meaning that its sequence of coefficients contains arbitrary long strings of 0s. Flajolet cites a theorem stating that a lacunary series cannot be an algebraic function, which means that B is not algebraic. This, in turn, means that the generating function of $G_{\neq}$ is not an algebraic function, from which Flajolet concludes that $G_{\neq}$ is an inherently ambiguous language.

[1] This theorem is presented in [1], section 2, "Grammars as Generators of Formal Power Series." The theorem as presented in the original paper is actually more powerful than the version given here, as Chomsky and Schützenberger do not assume that variables in a power series commute with each other.

3. Integer-labeled context-free grammars

3.1. **Definitions.** We define an **integer-labeled context-free grammar** as follows. This definition, like our definition of a context-free grammar, is based on the definition of a context-free grammar found in Sipser ([4] p. 100).

An integer-labeled context-free grammar is a tuple (V, Σ, R, S), where

- V is a finite set of symbols, the *nonterminal symbols*;
- Σ is a finite set of symbols, the *terminal symbols*, disjoint from V;
- R is a *function* from the set $V \times (V \cup \Sigma)^*$ to the integers, whose value is 0 for all but finitely many inputs; and
- S is an element of V, the *start symbol*.

For each input (A, w) for which the function R has a nonzero value n, we say that $A \to (n)w$ is a **rule** of the grammar. Alternatively, we say that $A \to w$ is a rule of the grammar with multiplicity n.

The idea here is that we permit a rule to "count multiple times" (including a negative number of times). Thus, when counting the number of derivations of a string, if a derivation uses a rule with a multiplicity other than 1, then we count the derivation multiple times (specifically, the multiplicity of the derivation is the product of the multiplicities of the rules it uses). We will now define this notion more formally.

We define a function Y from the set $(V \cup \Sigma)^* \times (V \cup \Sigma)^*$ to the integers. Given strings s and t, if $Y(s, t) = n$, then we say that s **left-yields** t with multiplicity n (written $s \Rightarrow (n)t$). The definition of Y is

$$Y(s,t) = \sum_{(u,v,w)} R(A, w),$$

where the sum runs over all triples of strings (u, v, w) such that $uAv = s$, $uwv = t$, and u contains no nonterminal symbols.

Given a sequence $s_1, s_2, \ldots, s_k$ of strings, for $k \geq 1$, we say that this sequence is a **leftmost derivation** from s_1 to s_k with multiplicity n, where $n = Y(s_1, s_2) \cdot Y(s_2, s_3) \cdot \cdots \cdot Y(s_{k-1}, s_k)$. (The one-element sequence s_1 is always a leftmost derivation with multiplicity 1.)

Next, we define a partial function D from the set $(V \cup \Sigma)^* \times (V \cup \Sigma)^*$ to the set $\mathbb{Z} \cup \{+\infty, -\infty\}$. For strings s and t, if $D(s, t) = n$, then we say that s **derives** t with multiplicity n (written $s \stackrel{*}{\Rightarrow} (n)t$). The function $D(s, t)$ is defined as the sum of the multiplicities of all leftmost derivations $s_1 \Rightarrow s_2 \Rightarrow \cdots \Rightarrow s_k$, where $k \geq 1$, such that $s_1 = s$ and $s_k = t$. (If this sum has infinitely many positive terms but only finitely many negative terms, we say that the sum is $+\infty$. Likewise, if the sum has infinitely many negative terms but finitely many positive terms, we say that the sum is $-\infty$. If the sum has infinitely many positive terms and infinitely many negative terms, the sum is left undefined.)

Finally, we define the multiplicity of a string s in the language as $D(S, s)$. (Note that we are not yet defining the language of an integer-labeled context free grammar as a set.)

As an example, consider the integer-labeled context-free grammar

$$S \to (2)aS \mid (3)Sb \mid (1)\epsilon.$$

We will calculate the multiplicity of the string ab. This string has two derivations: $S \Rightarrow aS \Rightarrow aSb \Rightarrow ab$, and $S \Rightarrow Sb \Rightarrow aSb \Rightarrow ab$.

First, we consider the first derivation, $S \Rightarrow aS \Rightarrow aSb \Rightarrow ab$. The string S only yields the string aS via one rule, namely $S \to aS$, so $Y(S, aS)$ is simply the multiplicity of this rule, which is 2. Likewise, aS only yields aSb via the rule $S \to Sb$, so $Y(aS, aSb)$ is the multiplicity of this rule, or 3. Finally, aSb only yields ab via the rule $S \to (1)\epsilon$, so $Y(aSb, ab)$ is 1. Putting this together, we find that the multiplicity of this derivation is $2 \cdot 3 \cdot 1 = 6$.

The second derivation, $S \Rightarrow Sb \Rightarrow aSb \Rightarrow ab$, is the same, except that the rules are applied in a different order. This does not affect our multiplication; the multiplicity of this derivation is again $3 \cdot 2 \cdot 1 = 6$.

Finally, the multiplicity of the string ab is the sum of the multiplicities of all of its derivations: $D(S, ab) = 6 + 6 = 12$. The significance of this number, essentially, is that it is the number of leftmost derivations the string ab *would* have in a context-free grammar containing two copies of the rule $S \to aS$ and three copies of the rule $S \to Sb$, if context-free grammars were allowed to contain the same rule multiple times.

We will now define the language of an integer-labeled context-free grammar as a set. Clearly a string with positive multiplicity should be included, and a string with zero multiplicity should be excluded, but it is not clear whether a string with negative or undefined multiplicity should be included or excluded. Therefore, we will focus on grammars where every multiplicity is nonnegative:

A *nonnegative integer-labeled context-free grammar* (or *nilCFG*) is an integer-labeled context-free grammar where the multiplicity of each string is defined and nonnegative. The language of a nilCFG is defined as the set of all strings with positive multiplicity.

We then define an *unambiguous nilCFG* as a nilCFG in which the multiplicity of each string is either 0 or 1. Note that a string in an "unambiguous" nilCFG may still have multiple leftmost derivations, as long as their multiplicities sum to 1.

3.2. **Example from Du & Ko revisited.** Recall the language from Du and Ko [2] we discussed earlier, namely, $L = \{a^m b^n c^p : m + 2n - p \geq 0\}$. We will create an unambiguous nilCFG for this language.

Again, we separately consider the cases $p \leq m$, $m \leq p$, and $m = p$.

First, we consider the case where $p \leq m$. In this case, it is always true that $m + 2n - p \geq 0$. Thus, the only constraint on n and p is that they be nonnegative integers, and the constraint on m is that it be greater than or equal to p.

It is straightforward to come up with an unambiguous context-free grammar describing this case. Essentially, someone who wishes to build a string satisfying this grammar is permitted to add as many as and bs as they like; they may also add as many cs as they like, but for each c they add, they must add an a as well. The grammar describing this is

$$T \to aT \mid U$$
$$U \to bU \mid V$$
$$V \to aVc \mid \epsilon.$$

We then consider the case where $m \leq p$. Now the only constraint on m and n is that they be nonnegative integers, and the constraint on p is that it be an integer between m and $m + 2n$ inclusive.

Again, creating an unambiguous context-free grammar describing this case is not too difficult. We will say that a person is allowed to use as many as and bs as they wish. For each a that they add, they must also add a c. For each b that they add, they may add zero, one, or two cs; however, to avoid ambiguity, they may only use the "add one c" option once. The grammar which describes all of these rules is

$$\begin{aligned} W &\to aWc \mid X \\ X &\to Y \mid bYc \\ Y &\to bY \mid Z \\ Z &\to bZcc \mid \epsilon. \end{aligned}$$

Finally, we consider the case where $m = p$. In this case, the inequality $m + 2n - p \geq 0$ simplifies to $2n \geq 0$. Thus, the only constraint, apart from $m = p$, is that all variables be non-negative. To create an unambiguous CFG for this case, we will say that a person may add as many as and bs as they wish, as long as for each a, they add exactly one c.

$$\begin{aligned} R &\to aRc \mid Q \\ Q &\to bQ \mid \epsilon \end{aligned}$$

A context-free grammar for the entire language L is simply $S \to T \mid W$ (along with the above production rules, of course). However, since the case described by T and the case described by W overlap, this grammar is not unambiguous. But we can construct an unambiguous nilCFG for L simply by subtracting out the overlap: $S \to T \mid W \mid (-1)\ R$.

We can now derive the multivariate generating function for this language by solving the following system of (linear) polynomial equations:

$$\begin{aligned} \mathbf{S} &= \mathbf{T} + \mathbf{W} + (-1)\mathbf{R} \\ \mathbf{T} &= \mathbf{aT} + \mathbf{U} \\ \mathbf{U} &= \mathbf{bU} + \mathbf{V} \\ \mathbf{V} &= \mathbf{aVc} + 1 \\ \mathbf{W} &= \mathbf{aWc} + \mathbf{X} \\ \mathbf{X} &= \mathbf{Y} + \mathbf{bYc} \\ \mathbf{Y} &= \mathbf{bY} + \mathbf{Z} \\ \mathbf{Z} &= \mathbf{bZcc} + 1 \\ \mathbf{R} &= \mathbf{aRc} + \mathbf{Q} \\ \mathbf{Q} &= \mathbf{bQ} + 1 \end{aligned}$$

Omitting the algebraic details, we find that

$$\mathbf{T} = \frac{1}{(1-\mathbf{ac})(1-\mathbf{a})(1-\mathbf{b})}$$
$$\mathbf{W} = \frac{(1+\mathbf{bc})}{(1-\mathbf{ac})(1-\mathbf{b})(1-\mathbf{bc}^2)}$$
$$\mathbf{R} = \frac{1}{(1-\mathbf{ac})(1-\mathbf{b})}$$
$$\begin{aligned}\mathbf{S} &= \frac{1}{(1-\mathbf{ac})(1-\mathbf{a})(1-\mathbf{b})} + \frac{(1+\mathbf{bc})}{(1-\mathbf{ac})(1-\mathbf{b})(1-\mathbf{bc}^2)} - \frac{1}{(1-\mathbf{ac})(1-\mathbf{b})} \\ &= \frac{1+\mathbf{bc}-\mathbf{abc}-\mathbf{abc}^2}{(1-\mathbf{ac})(1-\mathbf{a})(1-\mathbf{b})(1-\mathbf{bc}^2)}\end{aligned}$$

As we can see, this method of deriving an ordinary generating function can be simpler than the method we used for this language earlier.

3.3. **The power of nilCFGs?** Since nilCFGs are a generalization of context-free grammars, one may expect nilCFGs to be more powerful. However, the author has not been able to prove or disprove the existence of languages that can be (unambiguously or otherwise) described by a nilCFG but not by a CFG. The "best case" is that there exist languages which can be described by unambiguous nilCFGs, but cannot be described even by ambiguous CFGs. We conjecture that this is the case.

One seemingly promising approach is to find an unambiguous context-free language L whose complement is not a context-free language. If such a language is found, then the complement of L will be an unambiguous nilCFG, which can be constructed from the original unambiguous CFG by adding a new rule $S' \to A \mid (-1)S$ (where A is a nonterminal which generates every string exactly once, and S is the start symbol of the original CFG), and using S' as the new start symbol.

There does not seem to be any obvious reason to believe that no such language L exists. After all, in general, the complement of a context-free language is not a context-free language; there is no clear reason that requiring L to be unambiguous would force the complement of L to be context-free. However, we have not been able to find an example of such a language L.

3.4. **Chomsky–Schützenberger Theorem, extended.** Recall that the Chomsky–Schützenberger Theorem states that given an unambiguous context-free grammar, the grammar can be transformed into a set of polynomial equations describing the language's multivariate ordinary generating function. We will show that the same is true of unambiguous nilCFGs.

This proof is based on the proof found in Chomsky and Schützenberger ([1] pp. 126–127).

Suppose that we have an unambiguous nilCFG G. We will begin by "normalizing" G, in order to create another nilCFG G' that generates the same language (except that G' will not generate ϵ regardless of whether or not G generates ϵ).

(In the below process, at any point where we would create a rule of the form $A \to (p)s$, where the rule $A \to (q)s$ already exists, we instead replace the latter rule with the rule $A \to (p+q)s$.)

First, suppose that there is a cycle $A_1, A_2, \ldots, A_q$ of nonterminals, such that $A_1 \to (n_1)A_2$, $A_2 \to (n_2)A_3$, $\ldots$, $A_{q-1} \to (n_{q-1})A_q$, and $A_q \to (n_q)A_1$ are all rules

of G. Then every nonterminal in the cycle is unreachable, because if a nonterminal in the cycle were reachable, then any derivation involving this nonterminal could be extended to create infinitely many derivations of the same string (by repeating the cycle arbitrarily many times). But an unambiguous nilCFG can't derive the same string infinitely many times. Since all of the nonterminals in the cycle are unreachable, we delete every rule mentioning any of these nonterminals. We repeat this process until no such cycles remain.

Next, suppose that there is a pair (A, B) of nonterminals in G such that $A \to (n)B$ is a rule of G. Of all such pairs, select one such that B does not have any rules of the form $B \to (n)C$ where C is a nonterminal. (There must exist such a pair; the only way such a pair could fail to exist is if $A \to (n)B$ were part of a cycle, but all cycles have been deleted.) We delete the rule $A \to (n)B$, and then for every rule of the form $D \to (p)sAt$, we add an additional rule $D \to (p)sBt$. We repeat this process until no such pairs (A, B) remain.

Next, suppose that there is some nonterminal A in G such that $A \to (n)\epsilon$ is a rule of G. We create a new nonterminal A' that has the same rules as A, except that the rule $A \to (n)\epsilon$ is removed. Then, for each rule of the form $B \to (p)sAt$, we replace this rule with a pair of rules $B \to (p)sA't$ and $B \to (np)st$; observe that this does not change the behavior of B. (If a rule has multiple As on the right hand side, then we will continue performing this replacement until A no longer appears on the right hand side of any rule.) We then delete the original nonterminal A and all of its rules. (This process may introduce rules of the form $B \to (p)C$, so at this point, we repeat the above two paragraphs.) We repeat this process until no rules of the form $A \to (n)\epsilon$ remain.

It is possible that in the above paragraph, we deleted the start symbol S and replaced it with a new symbol S', perhaps multiple times. If this is the case, we simply use the new symbol as the new start symbol for G'; the only potential difference is that G', perhaps unlike G, does not generate ϵ.

As a final modification, we delete every nonterminal A such that A does not derive any terminal strings.

We now have an unambiguous nilCFG G' with the property that the right-hand side of each rule contains at least one terminal or at least two (not necessarily distinct) nonterminals. Now, for each nonterminal A in G', we can write the equation

$$\mathbf{A} = \sum_{A \to (n)s} (n \cdot \mathbf{s}),$$

where $\mathbf{s}$ is the product of the variables $\mathbf{x}$ for each terminal symbol x in the string s. These equations describe the multivariate ordinary generating function of G' by (possibly recursively) writing the generating function for each nonterminal symbol in terms of the generating functions for itself and the other nonterminal symbols.

We can now define a sequence of polynomials $\mathbf{S}_0, \mathbf{S}_1, \ldots$, defined by letting $\mathbf{S}_0 = \mathbf{S}$, and letting $\mathbf{S}_{n+1}$ be the polynomial formed from $\mathbf{S}_n$ by replacing every occurrence of a nonterminal variable $\mathbf{A}$ with the right-hand side of its corresponding equation.

Define the "string-degree" of a term as the number of nonterminal variables in the term, plus twice the number of terminal variables in the term. We will now show, using induction, that for all nonnegative integers n, every term of string-degree less than or equal to n in $\mathbf{S}_n$ consists entirely of terminal variables. In the

base case, $n = 0$, so we are considering $\mathbf{S}_0 = \mathbf{S}$. This polynomial has no terms of string-degree 0, so the property is vacuously true.

For the inductive step, we will assume that (for some specific n) every term of string-degree less than or equal to n in $\mathbf{S}_n$ consists entirely of terminal variables, and we will show that the same is true for $n + 1$ and $\mathbf{S}_{n+1}$. We know that, in $\mathbf{S}_n$, every term containing nonterminal variables has string-degree at least $n + 1$. When we form $\mathbf{S}_{n+1}$, in each new term, either a nonterminal is replaced with at least a terminal (increasing the string-degree), or a nonterminal is replaced with at least two nonterminals (also increasing the string-degree). So every new term in $\mathbf{S}_{n+1}$ must have string-degree at least $n + 2$. Since every *old* term in $\mathbf{S}_{n+1}$ consists entirely of terminals (otherwise it would have been replaced), we can conclude that every term of string-degree less than or equal to $n + 1$, being an old term, must consist entirely of terminals.

We now know that we can construct the limit $\mathbf{G}'$ of this sequence by taking all the terms of string-degree n from $\mathbf{S}_n$, for each nonnegative integer n, and the result will be a solution to our system of polynomial equations, and thus will be the multivariate ordinary generating function for G'.

Finally, since the only possible difference between the languages of G' and G is that G' does not contain ϵ whereas G may, the generating function $\mathbf{G}$ for G is either $\mathbf{G}'$ (if G does not contain ϵ) or $\mathbf{G}' + 1$ (if G does contain ϵ).

Acknowledgements

This paper is a senior thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Arts at Grand Valley State University. I would like to thank my advisor, Prof. Edward Aboufadel, for his support and advice, without which this paper would not have been possible.

References

[1] N. Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Compuer Programming and Formal Systems*, pages 118–161. North-Holland, 1963.

[2] Ding-Zhu Du and Ker-I Ko. *Problem Solving in Automata, Languages, and Complexity*. John Wiley & Sons, 2001.

[3] Philippe Flajolet. Analytic models and ambiguity of context-free languages. In *Theoretical Computer Science*, pages 283–309. North-Holland, 1987.

[4] Michael Sipser. *Introduction to the Theory of Computation, Second Edition*. Thomson Course Technology, second edition, 2006.