AN ALGEBRAIC VIEW OF AMBIGUITY IN CONTEXT-FREE GRAMMARS AND LANGUAGE EXPRESSIONS

TOM ALTMAN
Department of Computer Science and Engineering
University of Colorado Denver, Denver, CO 80217, United States

GEORGE LOGOTHETIS
R&D Bloomberg L.P.
New York, N.Y. 10022, United States

Context-free grammars are viewed as systems of algebraic equations incorporating the algebraic operations of union and concatenation on languages. This algebraic view also extends to encompass ambiguity in context-free grammars and language expressions, and provides significant advantages over the traditional view of ambiguity. In the context-free environments, ambiguity can be decomposed into union, concatenation, and closure ambiguity. Previously, we had shown that the traditional proof of the undecidability of ambiguity in context-free grammars addresses only one part of ambiguity, namely the union-ambiguity problem. Along with some practical applications of the algebraic approach, such as the detection, identification, and elimination of certain ambiguity-causing grammar constructs, we extend and generalize this result by showing that, for ϵ-free context-free languages, both the concatenation ambiguity as well as the two types of the closure ambiguity problems are also undecidable. It is not surprising, therefore, that even the most restricted versions of this problem are NP-complete.

Keywords: algebraic language theory; operations on languages; decision problems; ambiguity detection and/or reduction; formal power series; approximating languages.

1991 Mathematics Subject Classification: 03D35, 20M35, 40G99, 41A99, 68Q25, 68Q70

1. Introduction

Context-free grammars (CFGs) and the languages they generate, the context-free languages (CFLs), are useful in defining the syntax of programming languages, formalizing the notion of parsing, and numerous other string-processing applications. Chomsky [4] originated the CFG formalism which he later expanded in [5,6]. The related Backus-Naur form (BNF) notation was used for the description of ALGOL in [2,16]. Ginsburg and Rice [10] showed the relationship between CFGs and BNF. Ambiguity in CFGs was first studied by Cantor [3], Floyd [9], and Greibach [13]. For a thorough investigation of algebraic properties of formal languages, the reader can refer to [7,11,12,15,17,18,19].

Traditionally, the definition of context-free languages from CFGs is expressed in terms of derivations or, equivalently, parse trees. Let $G = (N, \Sigma, P, S)$ be a CFG,

where N and Σ are the sets of nonterminals and terminals, respectively, P is the set of productions, and $S \in N$ is the start symbol. The productions in P have the form $A \to \alpha$, where A is a nonterminal and α is either a string $X_1 \cdots X_n$ of grammar symbols from $(N \cup \Sigma)^+$ or the empty string ϵ. Each nonterminal A defines a language $\mathrm{L}(A)$ over the alphabet Σ. String x is a sentence of $\mathrm{L}(A)$ iff there exists a parse tree with root A and frontier x. By definition, the language $\mathrm{L}(G)$ of the grammar is the language $\mathrm{L}(S)$. A grammar is called *ambiguous* iff there is a sentence of $\mathrm{L}(G)$ that has more than one parse tree, and *unambiguous*, otherwise.

Example 1. The CFG $G_1 = (N, \Sigma, P, S)$, where

$$
\begin{aligned}
N &= \{\texttt{E}, \texttt{F}\},\\
\Sigma &= \{\texttt{n}, \texttt{+}, \texttt{*}, \texttt{(}, \texttt{)}\},\\
P &= \{\texttt{E} \to \texttt{E+E}, \texttt{E} \to \texttt{E*F}, \texttt{E} \to \texttt{F}, \texttt{F} \to \texttt{(E)}, \texttt{F} \to \texttt{n}\},\\
S &= \texttt{E},
\end{aligned}
$$

will be used as a running example throughout the paper. This grammar is ambiguous, as can be demonstrated by several distinct parse trees for the sentence `n*n+n+n*n` of $\mathrm{L}(G_1)$, shown in Figure 1. □

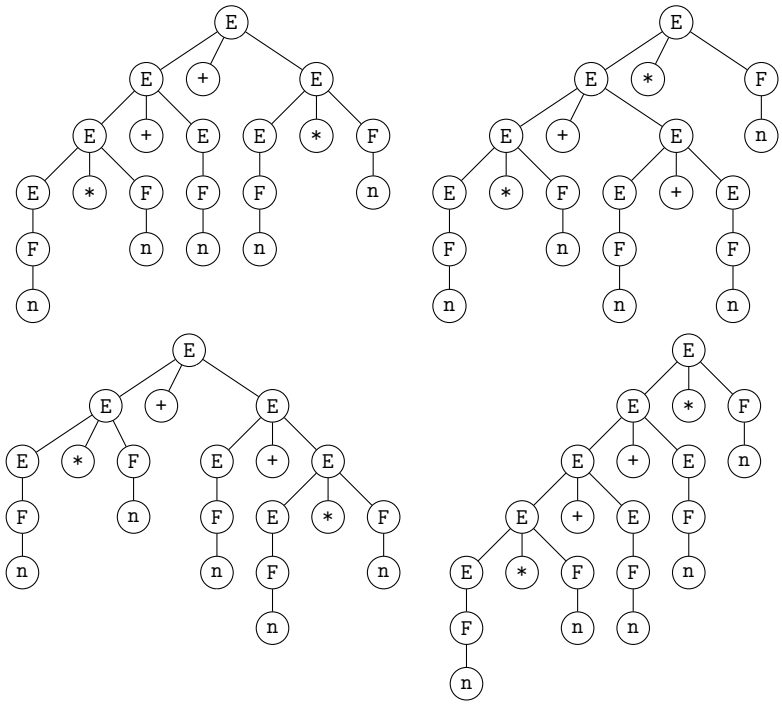

Fig. 1. Four parse trees for the string `n*n+n+n*n`

The issue of ambiguity in CFGs is very important, since the grammars utilized in describing the syntax of programming languages are expected to be unambiguous. Therefore, it would be desirable to have available algorithms, properties, and methodologies that can assist in the following endeavors:

- Decide whether or not any given CFG is ambiguous.
- Identify and locate ambiguity within the structure of a CFG.
- Identify constructs guaranteed to cause ambiguity in CFGs.
- Transform an ambiguous CFG into an unambiguous or, at least, less ambiguous one that generates the same language.

Unfortunately, the traditional treatment of ambiguity in CFGs provides little help along these lines. Indeed, the major result is negative: Specifically, it is well known that the ambiguity problem in CFGs is undecidable [3,9,13]. The objective of this paper is to present an algebraic characterization of CFGs which provides a much more precise, refined, and purposeful way of describing ambiguity, its manifestation in terms of grammar constructs, and transformations that can help in eliminating ambiguity from CFGs. In Sections 2-4, we present the algebraic view of CFGs and CFLs, followed by an algebraic view of ambiguity in CFGs, and its advantages over the traditional approach. In Section 5, we address the undecidability of ambiguity in CFGs and CF language expressions and show that even in the most restricted (i.e., bounded) instances, CFG ambiguity problems are NP-complete or harder. We propose a practical algorithm that detects the presence of ambiguity, although not its absence, in Section 6. The issues of grammars and their numerical power series as well as transformations that can eliminate certain ambiguous constructs from CFGs are addressed in Section 7, with concluding remarks in Section 8.

2. Algebraic View of CFGs and Languages

Let Σ be an alphabet, Σ^* the set of strings over Σ, and $\mathcal{P}(\Sigma^*)$ the set of languages over Σ (i.e., the powerset of Σ^*), and $L, M \in \mathcal{P}(\Sigma^*)$. Algebraic operations of *union*, *concatenation*, and (*positive*) *Kleene closure* are defined as follows:

(1) $L \cup M = \{x \in \Sigma^* \mid x \in L \text{ or } x \in M\}$.
(2) $L{\cdot}M = \{xy \in \Sigma^* \mid x \in L \text{ and } y \in M\}$.
(3) $L^+ = \{x \in \Sigma^* \mid x \in L \text{ or } x \in L{\cdot}L \text{ or } x \in L{\cdot}L{\cdot}L \text{ or } ...\}$; Also, $L^* = L^+ \cup \{\epsilon\}$.

Union and concatenation operations have a number of important algebraic properties: Let A, B, C, and D be arbitrary languages over Σ.

- Both union and concatenation are associative: $A \cup (B \cup C) = (A \cup B) \cup C$ and $A{\cdot}(B{\cdot}C) = (A{\cdot}B){\cdot}C$.
- Union is commutative, but concatenation is not: $A \cup B = B \cup A$, but $A{\cdot}B \neq B{\cdot}A$.
- The languages $\varnothing$ and $\{\epsilon\}$ are the identity operands of union and concatenation, respectively: $A \cup \varnothing = \varnothing \cup A = A$ and $A{\cdot}\{\epsilon\} = \{\epsilon\}{\cdot}A = A$. Also, $\varnothing$ is the zero operand of concatenation: $A{\cdot}\varnothing = \varnothing{\cdot}A = \varnothing$.

- Concatenation distributes over union from both left and right: $A{\cdot}(B \cup C) = (A{\cdot}B) \cup (A{\cdot}C)$ and $(B \cup C){\cdot}A = (B{\cdot}A) \cup (C{\cdot}A)$.
- Both union and concatenation are monotonic: $A \subseteq C$ and $B \subseteq D$ imply that $A \cup B \subseteq C \cup D$ and $A{\cdot}B \subseteq C{\cdot}D$.

A finite length algebraic formula built in terms of these operations on languages will be called a *language expression*. Parentheses may be used to indicate the proper order in which the various operations are applied. Also, the associativity of union and concatenation makes the use of parentheses for grouping successive unions or successive concatenations redundant, unless there are other reasons for indicating a specific order. Furthermore, the distributivity of concatenation over union may be used to transform a language expression into one formed as unions of concatenations, without parentheses. Finally, a language expression involving CFLs can always be simplified, or transformed, into an equivalent one that contains no empty operands.

The definition of a context-free language by a CFG can be expressed as the solution to a system of algebraic equations. The unknowns in these equations are the languages of the various nonterminals in the grammar. The equations contain language expressions that are formed from the right-hand sides of the productions in the grammar. We first group all productions $A \to \alpha_1, A \to \alpha_2, ..., A \to \alpha_m$ of a nonterminal A into the form $A \to \alpha_1 \mid \alpha_2 \mid ... \mid \alpha_m$. This composite production for A can then be transformed into an equation defining the language $\mathrm{L}(A)$, as follows:

- $\mathrm{L}(A) = \mathrm{L}(\alpha_1 \mid \alpha_2 \mid ... \mid \alpha_m)$.
- $\mathrm{L}(\alpha_1 \mid \alpha_2 \mid ... \mid \alpha_m) = \mathrm{L}(\alpha_1) \ \cup \ \mathrm{L}(\alpha_2) \cup \cdots \cup \mathrm{L}(\alpha_m)$.
- $\mathrm{L}(X_1X_2 \cdots X_n) = \mathrm{L}(X_1){\cdot}\mathrm{L}(X_2){\cdot} \ \cdots \ {\cdot}\mathrm{L}(X_n)$.
- For $\mathtt{t} \in \Sigma, \ \mathrm{L}(\mathtt{t}) = \{\mathtt{t}\}$.
- $\mathrm{L}(\epsilon) = \{\epsilon\}$.

To each nonterminal A in a CFG corresponds an equation $\mathrm{L}(A) = le$ defining $\mathrm{L}(A)$. This language expression le, the defining expression of $\mathrm{L}(A)$, has the form of unions of concatenations. Collectively, the definitions of the nonterminal languages form a complete system of equations uniquely defining each of these languages.

Example 2. Grouping together the productions for the two nonterminals of G_1 yields the composite productions:

$\mathtt{E} \to \mathtt{E{+}E} \mid \mathtt{E{*}F} \mid \mathtt{F}$
$\mathtt{F} \to \mathtt{(E)} \mid \mathtt{n}$,

that may be interpreted as the following system of equations:

$\mathrm{L}(\mathtt{E}) = \mathrm{L}(\mathtt{E}){\cdot}\{\mathtt{+}\}{\cdot}\mathrm{L}(\mathtt{E}) \cup \mathrm{L}(\mathtt{E}){\cdot}\{\mathtt{*}\}{\cdot}\mathrm{L}(\mathtt{F}) \cup \mathrm{L}(\mathtt{F})$
$\mathrm{L}(\mathtt{F}) = \{\mathtt{(}\}{\cdot}\mathrm{L}(\mathtt{E}){\cdot}\{\mathtt{)}\} \cup \{\mathtt{n}\}$. □

The interpretation of a CFG as a system of algebraic equations yields definitions of the languages of the various nonterminals that are equivalent to those obtained via the traditional techniques that make use of derivations, parse trees, or even the less traditional approaches that use, for example, the formal power series in associative, but noncommutative variables [7,17,19], or the morphic generator for context-free languages [18]. Specifically, for any nonterminal A, the language $\mathrm{L}(A)$ is the limit of successive *approximating languages* $\mathrm{L}_i(A)$, $i = 1, 2, ...$, that satisfy the following properties, see also [7,10,11]:

(1) $\mathrm{L}_0(A) = \varnothing$.
(2) $\mathrm{L}_i(A) \subseteq \mathrm{L}_{i+1}(A) \subseteq \mathrm{L}(A)$, for $i \geq 0$.
(3) The approximating languages $\mathrm{L}_i(A)$ converge monotonically to $\mathrm{L}(A)$, in the sense that for every sentence x of $\mathrm{L}(A)$ there exists a unique $k > 0$, such that $x \notin \mathrm{L}_i(A)$ for $i < k$ and $x \in \mathrm{L}_i(A)$ for $i \geq k$.

For a given A, the definitions of the approximating languages $\mathrm{L}_i(A)$ are obtained from the equation $\mathrm{L}(A) = le$ defining $\mathrm{L}(A)$. Specifically, $\mathrm{L}_0(A) = \varnothing$, and for each $i \geq 0$, $\mathrm{L}_{i+1}(A) = (le)_i$, where $(le)_i$ denotes the language expression le in which each instance of a nonterminal language $\mathrm{L}(B)$ is replaced with $\mathrm{L}_i(B)$.

Example 3. Starting with $\mathrm{L}_0(\mathtt{E}) = \varnothing$ and $\mathrm{L}_0(\mathtt{F}) = \varnothing$, the approximating languages for $\mathrm{L}(\mathtt{E})$ and $\mathrm{L}(\mathtt{F})$ in Example 2 can now be defined iteratively as:

$$\mathrm{L}_{i+1}(\mathtt{E}) = \mathrm{L}_i(\mathtt{E})\cdot\{\mathtt{+}\}\cdot\mathrm{L}_i(\mathtt{E}) \cup \mathrm{L}_i(\mathtt{E})\cdot\{\mathtt{*}\}\cdot\mathrm{L}_i(\mathtt{F}) \cup \mathrm{L}_i(\mathtt{F})$$
$$\mathrm{L}_{i+1}(\mathtt{F}) = \{\mathtt{(}\}\cdot\mathrm{L}_i(\mathtt{E})\cdot\{\mathtt{)}\} \cup \{\mathtt{n}\}.$$

Now, we can construct the first few approximating languages, as shown below.

$$\mathrm{L}_1(\mathtt{E}) = \varnothing\cdot\{\mathtt{+}\}\cdot\varnothing \cup \varnothing\cdot\{\mathtt{*}\}\cdot\varnothing \cup \varnothing = \varnothing$$
$$\mathrm{L}_1(\mathtt{F}) = \{\mathtt{(}\}\cdot\varnothing\cdot\{\mathtt{)}\} \cup \{\mathtt{n}\} = \{\mathtt{n}\}$$

$$\mathrm{L}_2(\mathtt{E}) = \varnothing\cdot\{\mathtt{+}\}\cdot\varnothing \cup \varnothing\cdot\{\mathtt{*}\}\cdot\{\mathtt{n}\} \cup \{\mathtt{n}\} = \{\mathtt{n}\}$$
$$\mathrm{L}_2(\mathtt{F}) = \{\mathtt{(}\}\cdot\varnothing\cdot\{\mathtt{)}\} \cup \{\mathtt{n}\} = \{\mathtt{n}\}$$

$$\mathrm{L}_3(\mathtt{E}) = \{\mathtt{n}\}\cdot\{\mathtt{+}\}\cdot\{\mathtt{n}\} \cup \{\mathtt{n}\}\cdot\{\mathtt{*}\}\cdot\{\mathtt{n}\} \cup \{\mathtt{n}\} = \{\mathtt{n+n}, \mathtt{n*n}, \mathtt{n}\}$$
$$\mathrm{L}_3(\mathtt{F}) = \{\mathtt{(}\}\cdot\{\mathtt{n}\}\cdot\{\mathtt{)}\} \cup \{\mathtt{n}\} = \{\mathtt{(n)}, \mathtt{n}\}$$

$$\mathrm{L}_4(\mathtt{E}) = \{\mathtt{n+n}, \mathtt{n*n}, \mathtt{n}\}\cdot\{\mathtt{+}\}\cdot\{\mathtt{n+n}, \mathtt{n*n}, \mathtt{n}\} \cup \{\mathtt{n+n}, \mathtt{n*n}, \mathtt{n}\}\cdot\{\mathtt{*}\}\cdot\{\mathtt{(n)}, \mathtt{n}\} \cup \{\mathtt{(n)}, \mathtt{n}\} =$$
$$\{\mathtt{n+n+n+n}, \mathtt{n+n+n*n}, \mathtt{n+n+n}, \mathtt{n*n+n+n}, \mathtt{n*n+n*n}, \mathtt{n*n+n}, \mathtt{n+n+n}, \mathtt{n+n*n}, \mathtt{n+n},$$
$$\mathtt{n+n*(n)}, \mathtt{n+n*n}, \mathtt{n*n*(n)}, \mathtt{n*n*n}, \mathtt{n*(n)}, \mathtt{n*n}, \mathtt{(n)}, \mathtt{n}\}$$
$$\mathrm{L}_4(\mathtt{F}) = \{\mathtt{(}\}\cdot\{\mathtt{n+n}, \mathtt{n*n}, \mathtt{n}\}\cdot\{\mathtt{)}\} \cup \{\mathtt{n}\} = \{\mathtt{(n+n)}, \mathtt{(n*n)}, \mathtt{(n)}, \mathtt{n}\}.$$

Note that in $\mathrm{L}_4(\mathtt{E})$, the strings `n+n+n` and `n+n*n` are each listed twice in order to emphasize the fact that they can be constructed in more than one way. □

The equivalence of the traditional and algebraic interpretations of CFGs, encapsulated in the Properties (1)-(3) of the approximating languages, can be easily justified as follows: Property (1) is trivially true from the definition of $\mathrm{L}_0(A)$; Property (2) is a direct consequence of (1) and the monotonicity of union and concatenation; Property (3) is true due to the fact that, by construction, $\mathrm{L}_k(A)$ consists of precisely the sentences of $\mathrm{L}(A)$ that are the frontiers of parse trees of *height* $\leq k$.

3. Algebraic View of Ambiguity

We now proceed to describe ambiguity in CFGs in algebraic terms, namely in terms of the standard algebraic operations of union, concatenation, and closure on languages. Let us start by formally defining the notions of ambiguous union, concatenation, language expression, and closure operations.

Definition 1. The union operation in the expression $A \cup B$ is *ambiguous* if $A \cap B \neq \varnothing$. In other words, at least one string in $A \cup B$ has an ambiguous origin.□

Definition 2. The concatenation operation in the expression $A{\cdot}B$ is *ambiguous* if there exist $x_1, x_2 \in A$, $y_1, y_2 \in B$, $x_1 \neq x_2$ and $y_1 \neq y_2$ such that $x_1 y_1 = x_2 y_2$. In other words, at least one string in $A{\cdot}B$ can be expressed in more than one way as a concatenation of strings from A and B. □

Definitions 1 and 2 can be extended to (finite length) language expressions as well as (infinite length) language closure operations.

Definition 3. A language expression with no empty operands is *ambiguous* if one or more of the operations into which it is decomposed is ambiguous. □

Definition 4. For any language L, the positive and Kleene closure operations, denoted by L^+ and L^*, respectively, are *ambiguous* if for some $k \geq 1$, the language expression $\{\epsilon\} \cup L \cup L{\cdot}L \cup \cdots \cup L^k$ is ambiguous. □

Note that $\epsilon \in L$ implies that both L^+ and L^* are unconditionally ambiguous. If $L = \varnothing$, then $L^+ = \varnothing$ and $L^* = \{\epsilon\}$, and both, by definition, are not ambiguous. Determining whether each of the three algebraic operations is ambiguous will be called the *union-*, *concatenation-*, and *closure-ambiguity problem*, respectively.

Example 4. Consider the language expression $((((A{\cdot}B){\cdot}(C \cup D)) \cup E) \cup (F{\cdot}G))$ over nonempty languages $A - G$, in a fully parenthesized form to indicate clearly its breakdown into six language operations, three unions and three concatenations, as shown in Figure 2. This language expression is ambiguous iff at least one of the six operations is ambiguous. □

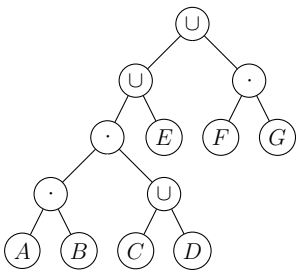

Fig. 2. Language Expression $((((A{\cdot}B){\cdot}(C \cup D)) \cup E) \cup (F{\cdot}G))$ represented schematically

As stated, the ambiguity of a language expression is tied to the specific decomposition of the expression into binary operations. A very important fact is that the ambiguity of a language expression is preserved under language containment as well as the algebraic transformations based on the associativity of union and concatenation, and the distributivity of concatenation over union. This can be established from the following properties.

Property 1. Both union-ambiguity and concatenation-ambiguity are monotonic: Let and $A \subseteq C$ and $B \subseteq D$. If $A \cup B$ is ambiguous then $C \cup D$ is ambiguous. Also, if $A{\cdot}B$ is ambiguous then $C{\cdot}D$ is ambiguous.

Proof. Follows from the monotonicity of union and concatenation operations. □

Property 2. $A \cup_1 (B \cup_2 C)$ is ambiguous iff $(A \cup_3 B) \cup_4 C$ is ambiguous.

Proof. ($\Rightarrow$) If the first union operation $\cup_1$ is ambiguous, then $\cup_3$ or $\cup_4$ (possibly both) must be ambiguous, and if $\cup_2$ is ambiguous, then $\cup_4$ must be ambiguous.

($\Leftarrow$) In a similar fashion, if $\cup_4$ is ambiguous, then $\cup_1$ or $\cup_2$ (possibly both) must be ambiguous, and if $\cup_3$ is ambiguous, then $\cup_1$ must be ambiguous. □

An immediate implication of the last property, and the commutativity of union, is that a language expression of the form $A_1 \cup A_2 \cup \cdots \cup A_n$ is ambiguous iff $A_i \cap A_j \neq \varnothing$, for some $1 \leq i < j \leq n$.

Property 3. $A{\cdot}(B{\cdot}C)$ is ambiguous iff $(A{\cdot}B){\cdot}C$ is ambiguous.

Proof. Similar to the proof of Property 2. □

An immediate implication of the last property is that a language expression of the form $A_1{\cdot}A_2{\cdot}\ \cdots\ {\cdot}A_n$ is ambiguous iff there exist strings $x_i, y_i \in A_i,\ i = 1, ..., n$, such that $x_1 x_2 \cdots x_n = y_1 y_2 \cdots y_n$ and $x_j \neq y_j$, for some $1 \leq j \leq n$.

Property 4. $A\cdot_1(B \cup_2 C)$ is ambiguous iff $(A\cdot_3 B) \cup_4 (A\cdot_5 C)$ is ambiguous.

Proof. ($\Rightarrow$) If $\cup_2$ is ambiguous, then $\cup_4$ must be ambiguous. If $\cdot_1$ is ambiguous, then there must exist $x_1, x_2 \in A$ and $y_1, y_2 \in B \cup C$ such that $x_1 \neq x_2,\ y_1 \neq y_2$, and $x_1y_1 = x_2y_2$. Now, if $y_1, y_2 \in B$ then $\cdot_3$ is ambiguous; if $y_1, y_2 \in C$ then $\cdot_5$ is ambiguous; or, if $y_1 \in B$ and $y_2 \in C$ or $y_1 \in C$ and $y_2 \in B$ then $\cup_4$ is ambiguous.

($\Leftarrow$) If $\cdot_3$ or $\cdot_5$ is ambiguous, then $\cdot_1$ must be ambiguous. If $\cup_4$ is ambiguous, then there must exist $x_1, x_2 \in A,\ y \in B,\ z \in C$ such that $x_1y = x_2z$. Now, if $x_1 = x_2$ then $y = z$ and $\cup_2$ is ambiguous; if $x_1 \neq x_2$ then $y \neq z$ and $\cdot_1$ is ambiguous. □

Property 5. $(A \cup B)\cdot C$ is ambiguous iff $(A\cdot C) \cup (B\cdot C)$ is ambiguous.

Proof. Analogous to the proof of Property 4. □

Figure 3 contains four algebraic transformations of the language expression shown in Figure 2. Based on the properties just described, the ambiguity of any one of the five expressions implies the ambiguity of the other four.

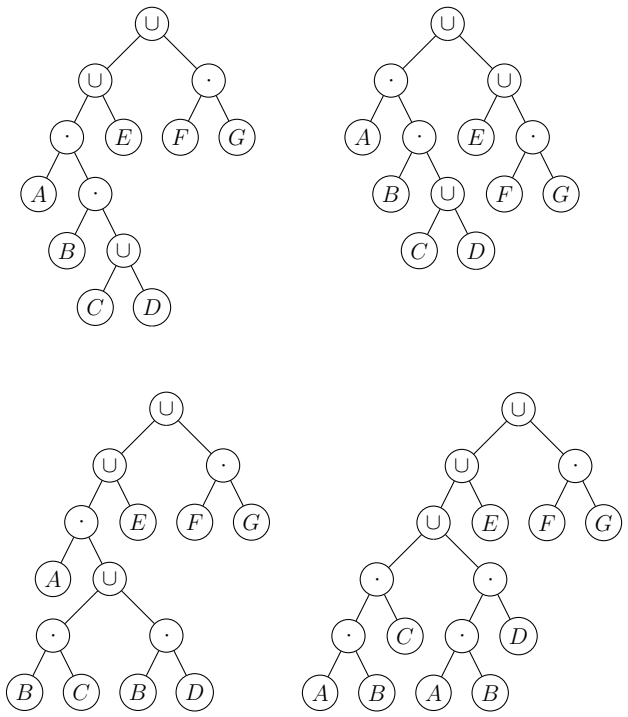

Fig. 3. Four algebraic transformations of the expression in Figure 2

As defined above, the ambiguity of language expressions can now be used to provide an alternative algebraic characterization of ambiguity in CFGs.

Definition 5. The language $\mathrm{L}(A)$ of a nonterminal A in a CFG is *ambiguously defined* if the language expression in the equation defining $\mathrm{L}(A)$ is itself ambiguous. A CFG is *algebraically ambiguous* if the language of one or more of its nonterminals is ambiguously defined. □

Altman and Logothetis [1] proved that a CFG without useless or unreachable nonterminals is algebraically ambiguous iff it is ambiguous in the traditional sense.

4. Advantages of the Algebraic View of Ambiguity

The traditional view of ambiguity characterizes a CFG to be ambiguous as a whole. The algebraic view, however, pinpoints ambiguity down to the level of each separate language definition in the grammar. Better yet, the ambiguity of each language definition can be pinpointed down to individual operations, that is, unions and/or concatenations, in the language expression.

As an example, let us look again at the ambiguous grammar G_1. One can show that $\mathrm{L}(\mathtt{E})$ is defined ambiguously, whereas $\mathrm{L}(\mathtt{F})$ is defined unambiguously. The 'fault', therefore, lies with the definition of $\mathrm{L}(\mathtt{E})$, and not the definition of $\mathrm{L}(\mathtt{F})$. This is important to know when facing the task of transforming G_1 into an unambiguous grammar that generates the same language. In the definition $\mathrm{L}(A) = le$ of a language $\mathrm{L}(A)$, the defining expression le has the form $\mathrm{L}(\alpha_1) \cup \mathrm{L}(\alpha_2) \cup \cdots \cup \mathrm{L}(\alpha_m)$, each $\mathrm{L}(\alpha_i)$ representing the language contributed by the production $A \to \alpha_i$ for A. Each pair of $\alpha_i, \alpha_j,\ 1 \leq i < j \leq n$, for which $\mathrm{L}(\alpha_i) \cap \mathrm{L}(\alpha_j) \neq \varnothing$ manifests an instance of a union-ambiguity in the definition of $\mathrm{L}(A)$, and we will say that there is a *union-ambiguity* between the pair of productions $A \to \alpha_i$ and $A \to \alpha_j$.

By examining G_1 and the definition $\mathrm{L}(\mathtt{E}) = \mathrm{L}(\mathtt{E+E}) \cup \mathrm{L}(\mathtt{E*F}) \cup \mathrm{L}(\mathtt{F})$ we see that the string $\mathtt{n+n*n} \in \mathrm{L}(\mathtt{E+E}) \cap \mathrm{L}(\mathtt{E*F})$; making the union operation in $\mathrm{L}(\mathtt{E+E}) \cup \mathrm{L}(\mathtt{E*F})$ ambiguous. Both $\mathrm{L}(\mathtt{E+E}) \cap \mathrm{L}(\mathtt{F}) = \varnothing$ and $\mathrm{L}(\mathtt{E*F}) \cap \mathrm{L}(\mathtt{F}) = \varnothing$. The one instance on union-ambiguity corresponds to the pair of productions $\mathtt{E} \to \mathtt{E+E}$ and $\mathtt{E} \to \mathtt{E*F}$.

Next, in the definition of $\mathrm{L}(A)$, each $\mathrm{L}(\alpha_i)$ has the form of $L(X_1) \cdot \ \cdots \ \cdot L(X_n)$. If this expression is ambiguous, then we say that there is a *concatenation-ambiguity* within the production rule $A \to X_1 \cdots X_n$.

For example, in the definition $\mathrm{L}(\mathtt{E}) = \mathrm{L}(\mathtt{E}) \cdot \{\mathtt{+}\} \cdot \mathrm{L}(\mathtt{E}) \cup \mathrm{L}(\mathtt{E}) \cdot \{\mathtt{*}\} \cdot \mathrm{L}(\mathtt{F}) \cup \mathrm{L}(\mathtt{F})$, the sub-expression $\mathrm{L}(\mathtt{E}) \cdot \{\mathtt{+}\} \cdot \mathrm{L}(\mathtt{E})$ is ambiguous. This is demonstrated by the strings: $x_1 = \mathtt{n+n} \in \mathrm{L}(\mathtt{E}), x_2 = \mathtt{+} \in \{\mathtt{+}\}, x_3 = \mathtt{n} \in \mathrm{L}(\mathtt{E}),\ y_1 = \mathtt{n} \in \mathrm{L}(\mathtt{E}), y_2 = \mathtt{+} \in \{\mathtt{+}\}$, and $y_3 = \mathtt{n+n} \in \mathrm{L}(\mathtt{E})$, giving us $x_1x_2x_3 = y_1y_2y_3 = \mathtt{n+n+n}$. On the other hand, $\mathrm{L}(\mathtt{E}) \cdot \{\mathtt{*}\} \cdot \mathrm{L}(\mathtt{F})$ is not ambiguous. Therefore, the concatenation-ambiguity in the definition of $\mathrm{L}(\mathtt{E})$ is manifested only within the production $\mathtt{E} \to \mathtt{E+E}$.

As shown, the algebraic view of ambiguity provides a much more detailed picture than that provided by the traditional view:

- The ambiguity of a CFG is manifested in terms of ambiguous definitions for the language $\mathrm{L}(A)$ of each nonterminal A. These, in turn, are manifested in terms of ambiguous union and concatenation operations of the expression defining $\mathrm{L}(A)$.
- Union-ambiguities in the definition of $\mathrm{L}(A)$ appear between the pairs of production rules for A.
- Concatenation-ambiguities surface within individual production rules for A.

A naive systematic process for detecting ambiguity in a CFG, based on the discussion above, is shown in Figure 4.

For each nonterminal A in the grammar, check the definition of $\mathrm{L}(A)$ for ambiguity.
Let $A \to \alpha_1, \ldots, A \to \alpha_n$ be the productions for A.
Union:

```
for i:=1 to n−1 do
  for j:=i+1 to n do
    if L(α_i) ∩ L(α_j) ≠ ∅ then
      announce union-ambiguity for A → α_i and A → α_j;
```

Concatenation:

```
for i:=1 to n do
  let α_i = X_1 ··· X_m
  L:=L(X_1);
  for j:=1 to m−1 do
    if L·L(X_{j+1}) is ambiguous then
      announce ambiguity in the j^th concatenation of A → α_i;
    L:=L·L(X_{j+1});
```

Fig. 4. A Naive Approach for Detecting Ambiguity

In computational terms, the problem of detecting ambiguity in a CFG is reduced to that of detecting ambiguous unions and concatenations of CFLs. Since the ambiguity problem in CFGs is undecidable, it follows that the process described above will not work. However, it is still valuable on two distinct counts:

- It helps expose the fact that the undecidability proof for the traditional ambiguity is incomplete, as only the problem of union-ambiguity for CFLs had been shown to be undecidable.
- It provides the basis for a computationally plausible and practical algorithm that accurately detects the presence of union, concatenation, and closure ambiguities, although not their absence.

5. Undecidability of Union, Concatenation, and Closure Ambiguity

The problem of ambiguity in CFGs has traditionally been shown to be undecidable by reducing Post's Correspondence Problem (PCP) to it. The proof below is similar to the one found in [14].

An instance PCP(A, B) of Post's Correspondence Problem consists of two lists of strings over a finite alphabet Σ, $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots, b_k$. PCP$(A, B)$ is said to have a solution iff there exists a sequence $i_1, i_2, \ldots, i_n$ of integers such that $1 \leq i_j \leq k$ for $j = 1, \ldots, n$, and

$$a_{i_1} a_{i_2} \cdots a_{i_n} = b_{i_1} b_{i_2} \cdots b_{i_n}.$$

Let $A = a_1, a_2, \ldots, a_k$ and $B = b_1, b_2, \ldots, b_k$ be the two lists of strings over Σ forming an instance of a PCP(A, B), and let $c_1, c_2, \ldots, c_k$, and $\#$ be distinct symbols not in Σ. Consider G_{Post}, a CFG corresponding to the PCP(A, B):

$$\begin{aligned} S &\rightarrow S_A \mid S_B \\ S_A &\rightarrow a_i S_A c_i \mid a_i c_i, i = 1, \ldots, k \\ S_B &\rightarrow b_i S_B c_i \mid b_i c_i, i = 1, \ldots, k. \end{aligned}$$

For languages L(S_A) and L(S_B), each set of production rules is clearly not ambiguous. Since $c_1, c_2, \ldots, c_k$ are distinct symbols not in Σ, G_{Post} can be ambiguous *if and only if* the union $L(S_A) \cup L(S_B)$, implied by the two production rules for S, is ambiguous, i.e., $L(S_A) \cap L(S_B) \neq \varnothing$. In other words, since the production rules for S_A and S_B are not ambiguous, the ability to decide whether the union of two context-free languages is ambiguous is *all* that is needed to solve the PCP(A, B). The above proof leaves open the decidability question for the problems of ambiguous concatenation as well as ambiguous closure operations on CFLs. As we show next, these problems are undecidable as well.

Theorem 1. [1] *Union-ambiguity is reducible to concatenation-ambiguity.*

Proof. Let L and M be two languages over Σ. Suppose we wish to determine if the union operation in $L \cup M$ is ambiguous, that is, whether or not $L \cap M \neq \varnothing$. Consider the languages $L' = L \cdot \{\#\} \cup \{\epsilon\}$ and $M' = M \cdot \{\#\} \cup \{\epsilon\}$ over $\Sigma \cup \{\#\}$. We will show that $L \cup M$ is ambiguous iff $L' \cdot M'$ is ambiguous.

($\Rightarrow$) Assume $L \cup M$ is ambiguous and let $x \in L \cap M$. Consider $y_1 = x\# \in L'$, $z_1 = \epsilon \in M'$ and $y_2 = \epsilon \in L'$, $z_2 = x\# \in M'$. Clearly, $y_1 z_1 = y_2 z_2 = x\#$, and $y_1 \neq y_2, z_1 \neq z_2$. Therefore, $L' \cdot M'$ is ambiguous.

($\Leftarrow$) Assume $L' \cdot M'$ is ambiguous. Consider strings $y_1, y_2 \in L'$, $y_1 \neq y_2$, and $z_1, z_2 \in M'$, $z_1 \neq z_2$, such that $y_1 z_1 = y_2 z_2$. Since $y_1 \neq y_2$, w.l.g. let y_1 be the longer of the two strings. So $y_1 \neq \epsilon$ and y_1 must have the form $y\#$, where $y \in L$. Now y_2 must be a prefix of y, and y cannot contain $\#$, so y_2 can only be ϵ. Therefore, we have $y\#z_1 = z_2$. Since $z_2 \neq \epsilon$, z_2 must have the form $z\#$, where $z \in M$. We have $y\#z_1 = z\#$. Since z cannot contain $\#$, z_1 must be ϵ. Finally, we have $y\# = z\#$, which implies $y = z$, making $L \cup M$ ambiguous. □

If L and M are CFLs, then so are L' and M', making the concatenation-ambiguity problem for Type 2, 1, and 0 languages undecidable. Note, however, that this proof was obtained by introducing the empty string ϵ into the languages L' and M', which played a pivotal role in the reduction argument.

Below, we show that for ϵ-free CFLs, the concatenation-ambiguity problem as well as the positive and Kleene closure-ambiguity problems are undecidable, even if the languages themselves were generated by unambiguous grammars.

Theorem 2. *For an ϵ-free CFL L generated by an unambiguous CFG, the union-ambiguity problem within the language expression $L \cup L{\cdot}L$ is undecidable.*

Proof. In G_{Post}, let us replace the production rules $S \to S_A \mid S_B$ by

$$\begin{aligned} S &\to S_1 \mid S_1S_1 \\ S_1 &\to \# \mid \#S_A \mid S_B. \end{aligned}$$

The grammar for $L = \mathrm{L}(S_1)$ is unambiguous. String $x \in L \cup L{\cdot}L = \mathrm{L}(S)$, where $S \to S_1 \to \#S_A \to x \in L$ and $S \to S_1S_1 \to \#S_1 \to \#S_B \to x \in L{\cdot}L$, will exist and have an ambiguous origin (union) iff $L(S_A) \cap L(S_B) \neq \varnothing$, i.e., iff $\mathrm{PCP}(A, B)$ has a solution. □

Theorem 3. *For an ϵ-free CFL L generated by an unambiguous CFG, the concatenation-ambiguity problem in the expression $L{\cdot}L$ is undecidable.*

Proof. In G_{Post}, let us replace the production rules $S \to S_A \mid S_B$ by

$$\begin{aligned} S &\to S_1S_1 \\ S_1 &\to \# \mid \#S_A \mid S_B\#. \end{aligned}$$

Again, the grammar for $L = \mathrm{L}(S_1)$ is unambiguous. String $x \in L{\cdot}L = \mathrm{L}(S)$, where $S \to S_1S_1 \to \#S_AS_1 \to \#S_A\# \to x$ and $S \to S_1S_1 \to \#S_1 \to \#S_B\# \to x$, will exist and have an ambiguous concatenation iff $\mathrm{PCP}(A, B)$ has a solution. □

It follows that for ϵ-free CFLs L and M, each generated by an unambiguous CFG, the concatenation-ambiguity problem in the expression $L{\cdot}M$ is undecidable.

Theorem 4. *For each $k \geq 1$ there exists an unambiguous CFG generating an ϵ-free CFL L such that language expressions of the form $\{\epsilon\} \cup L \cup L{\cdot}L \cup \cdots \cup L^j$, $j \leq k$, are not ambiguous; nevertheless, the problem of determining whether union-ambiguities exist within L^+ is undecidable.*

Proof. Consider the following set of unambiguous production rules for $L = \mathrm{L}(S_1)$

$$S_1 \to \# \mid \#^k S_A \mid S_B.$$

The shortest string in L^+ with a possible ambiguous origin (union) will be $S_1 \to \#^k S_A \to x \in L$ and $\#_1{\cdot}\ \cdots\ {\cdot}\#_k{\cdot}S_B = \#^k S_B \to x \in L^{k+1}$. String x will exist iff $\mathrm{PCP}(A, B)$ has a solution. N.B.: L^+ has no concatenation-ambiguities. □

Theorem 5. *For each $k \geq 1$ there exists an unambiguous CFG generating an ϵ-free CFL L such that language expressions of the form $\{\epsilon\} \cup L \cup L \cdot L \cup \cdots \cup L^j,\ j \leq k$, are not ambiguous; nevertheless, the problem of determining whether concatenation-ambiguities exist within L^+ is undecidable.*

Proof. Consider the following set of unambiguous production rules for $L = \mathrm{L}(S_1)$

$$S_1 \to \# \mid \#^k S_A \mid S_B \#^k.$$

The shortest string in L^+ with an ambiguous concatenation will be $x \in L^{k+1}$, where $\#^k S_A \cdot \#_1 \cdot \ \cdots \ \cdot \#_k = \#^k S_A \#^k \to x$ and $\#_1 \cdot \ \cdots \ \cdot \#_k \cdot S_B \#^k = \#^k S_B \#^k \to x$, and x will exist iff $\mathrm{PCP}(A, B)$ has a solution. N.B.: L^+ has no union-ambiguities. □

It is not by coincidence that Theorems 2, 3, and their proofs are, de facto, instances ($k = 1$) of Theorems 4 and 5, respectively. The next result follows immediately.

Theorem 6. *For CFLs, the closure-ambiguity problem is undecidable.* □

5.1. *Bounded Ambiguity Problem*

The undecidability of ambiguity in the union, concatenation, and closure operations can be traced back to the fact that for infinite CFLs, the empty-intersection problem is undecidable. There are two obvious ways to restrict the domain of the ambiguity problem to make it solvable: (1) put a limit on the *length* of the derivations for the strings which are to be examined for ambiguous unions and/or concatenations, or (2) put a limit on the *height* of the parse trees, i.e., check for ambiguities within and between the (finite) approximating languages of the nonterminals.

Constable et al. [8] had shown that $\mathrm{BPCP}(A, B)$, a bounded version of PCP in which the length of the solution sequence cannot exceed $K \leq |A|$, is NP-complete. Let us introduce two related bounded ambiguity problems, starting with BAP_1:

> INSTANCE: A Context-free grammar $G = (N, \Sigma, P, S)$, integer $K' \leq |P|$.
> QUESTION: Does there exist a string $x \in \mathrm{L}(G)$ that has more than one derivation of length $\leq K'$?

Clearly $\mathrm{BAP}_1 \in$ NP. Next, BPCP $\leq_p \mathrm{BAP}_1$, where $G = G_{Post}$ will be constructed exactly as in the original reduction of the unbounded PCP to the CFG ambiguity problem, and $K' = K + 1$. Note that for strings in $\mathrm{L}(G_{Post})$, the lengths of their derivations and the heights of their parse trees are related linearly. While BAP_1 is NP-complete, the second problem, BAP_2, appears to be even more challenging:

> QUESTION: Does there exist a string $x \in \mathrm{L}(G)$ that has more than one parse tree of height $\leq K'$?

CFGs exist where the shortest such string x is exponentially long in $|P|$ and, hence, in the problem size (G's encoding). BPCP is also reducible to BAP_2, so the problem is at least NP-hard. However, BAP_2 is not known to be in NP or even in PSPACE.

6. A Practical Algorithm for Detecting Ambiguity

The ambiguity detection process outlined in Figure 4 is not computationally feasible, because the sets involved in the language expressions containing the union and/or concatenation operations may be infinite. However, a variant of the algorithm, based on the monotonicity of union and concatenation operations and their ambiguity, is. Specifically, let us consider the definition of $\mathrm{L}(A)$ for some nonterminal A, and its approximating languages $\mathrm{L}_i(A)$, which are always finite. If, for some $k \geq 1$, the definition of $\mathrm{L}_k(A)$ demonstrates an instance of union or concatenation-ambiguity, then the definitions of $\mathrm{L}_i(A)$, $i \geq k$, and the definition of $\mathrm{L}(A)$ will demonstrate the same instance of ambiguity. Conversely, if the definition of $\mathrm{L}(A)$ demonstrates an instance of union or concatenation-ambiguity, then there must exist a $k' \geq 1$ such that every $\mathrm{L}_i(A)$, $i \geq k'$, will demonstrate the same instance of ambiguity. Hence, a practical pseudo-algorithm for detecting ambiguities can operate as follows:

---

for $k := 1, \ldots$ **do**
For each nonterminal A compute $\mathrm{L}_k(A)$ (* for closure, compute $\bigcup_{i=1}^{k} \mathrm{L}_k^i(A)$ *)
identifying union and concatenation ambiguities as depicted in Figure 4;

This is indeed a computationally feasible algorithm, in that the languages involved in the unions and concatenations are always finite and, therefore, the ambiguity of union and concatenation can be detected quite easily: Let A and B be finite languages, and let $|A|$ denote the number of sentences in A. Then $A \cup B$ is ambiguous iff $|A \cup B| < |A| + |B|$ and $A \cdot B$ is ambiguous iff $|A \cdot B| < |A||B|$. Furthermore, the closure ambiguities will naturally manifest themselves as union and/or concatenation ambiguities during the appropriate iterations of the k-loop. Notice that the loop '**for** $k := 1, \ldots$' is open-ended. For any given grammar, this algorithm will detect all union, concatenation, and closure ambiguities if that loop is allowed to iterate far enough. The problem, of course, is that one cannot predetermine an appropriate value of k by which ambiguities, if any, are detected.

The algorithm for detecting the presence of ambiguities, via the use of approximating languages (ALs), has been implemented and it is relatively useful in identifying the ambiguity-causing productions in CFGs. One concern is the size of the ALs, which for some grammars grow exponentially in k, the approximation index. Various approaches, e.g., using incrementally-growing prefix trees for the implicit representation of ALs, have significantly reduced the memory requirements (although not necessarily the running time) of the ambiguity detecting algorithm.

When this algorithm is applied to G_1, the union-ambiguity between productions `E` $\rightarrow$ `E+E` and `E` $\rightarrow$ `E*F` (for the string `n+n*n`) as well as the concatenation-ambiguity within the production `E` $\rightarrow$ `E+E` (for the string `n+n+n`) are first detected during the execution of the loop with value $k = 4$. AL L_4(`E`) in Example 3 confirms this fact.

7. Elimination of Certain-to-Occur Ambiguities

So far, we have focused primarily on the issues of classification and detection of ambiguity, both from a theoretical and a practical perspective. Equally important is the issue of elimination or reduction of ambiguity, i.e., the finding of language-preserving transformations of an ambiguous grammar into an unambiguous or less ambiguous one. An implication of Properties 2-5 is that language-preserving grammar transformations based on the associativity of union and concatenation, and the distributivity of concatenation over union are of no direct help, since they also preserve the ambiguity of the grammar. One can also show that language-preserving transformations that involve the substitution of a nonterminal in the right-hand side of a production with its defining expression, or vice versa, do not change the ambiguity of the definition. Instead, one needs transformations based on the *idempotency* properties of language operations. A very simple such transformation, for example, is based on the idempotency property "$A \cup A \equiv A$" of union, shown below.

Example 5. For any nonempty language A, the concatenation operation in the expression $(A \cup \{\epsilon\}) \cdot (A \cup \{\epsilon\})$ is ambiguous. This expression can be transformed as follows: $(A \cup \{\epsilon\}) \cdot (A \cup \{\epsilon\}) = A \cdot A \cup \{\epsilon\} \cdot A \cup A \cdot \{\epsilon\} \cup \{\epsilon\} = A \cdot A \cup A \cup A \cup \{\epsilon\} = A \cdot A \cup A \cup \{\epsilon\}$. This last transformation eliminated the one certain-to-occur union-ambiguity in "$A \cup A$". The resulting language expression is not unconditionally ambiguous, as the original one was, although it can still be ambiguous, depending on the content of A. □

Identifying such algebraic transformations that eliminate certain-to-occur union and concatenation-ambiguities appears to be an important step in the construction of less ambiguous CFGs. Very prominent in that effort is the algebraic operation of *closure*, which has many idempotency properties that help both in the detection and in the elimination of ambiguous constructs commonly encountered in CFGs. Of particular importance are the properties relating closure and language definitions:

- The definitions $L = fL \,|\, g$ and $L = Lf \,|\, g$ are equivalent to the definitions $L = f^* g$ and $L = g f^*$, respectively.

The above is instrumental in converting extended context-free definitions into the Backus Naur Form. It can be used to convert repetition to left or right recursion (and vice versa, if there is a need for it). Notice that it says that $\mathtt{a}^*$ is the solution to the language equation $X = \mathtt{a}X \,|\, \epsilon$. In other words, the repetition operator, sometimes also called the *quasi-inverse* [7,17,19], is the closest one has to an inverse element, as it could be obtained for the above by the following manipulation:

$$X = \mathtt{a}X \mid \epsilon \Rightarrow X = X\mathtt{a} + \epsilon \Rightarrow X - X\mathtt{a} = \epsilon \Rightarrow X(\epsilon - \mathtt{a}) = \epsilon \Rightarrow$$

$$X = \frac{\epsilon}{\epsilon - \mathtt{a}} = \bigcup_{i=0}^{\infty} \mathtt{a}^i = \epsilon + \mathtt{a} + \mathtt{a}^2 + \cdots = \mathtt{a}^*.$$

Although the operations of inverse union and inverse concatenation used in this manipulation may not formally exist, the analogy with the numeric equation

$$\mathcal{X} = \frac{1}{1-a} = \sum_{i=0}^{\infty} a^i = 1 + a + a^2 + \cdots$$

is clear. One is left to wonder whether the definition of 'imaginary' inverses of union and concatenation could be used to advantage as stepping stones for more general formula manipulations that, like the one above, start and end in 'real' elements.

7.1. *Power Series and Ambiguity*

It was observed by Schützenberger that any context-free language over a finite alphabet Σ can be defined in terms of formal power series (ps) in associative but noncommutative variables, which are simply functions from Σ^* to the integers [7]. Each ps is determined by assigning an integral coefficient to each word w in Σ^*. The collection of all ps is a ring, and the equations of the grammar become honest polynomial equations that are satisfied by the n-tuple of ps $(f_1, \ldots, f_n)$. The equations give recursive rules (based on grammar's productions) that allow for the complete solution $(f_1, \ldots, f_n)$ to be constructed by successive approximations [7,11,19].

If the variables in Σ commute (as in Example 6, below), the system becomes an ordinary algebraic system. In the context-free case, if the coefficients (of all the monomials) of the solutions $f_1, \ldots, f_n$ are all 0 or 1, then the CFG is unambiguous. Otherwise, these coefficients indicate the number of distinct derivations, or parse trees, for a given string w. Therefore, if a coefficient for any string exceeds 1, the corresponding CFG is ambiguous.

Example 6. Let us examine the grammar G_2 below.

$$\begin{aligned} Z &\to X \mid Y \\ X &\to \mathtt{aaa}X \mid \mathtt{aaa} \\ Y &\to \mathtt{aaaaaaa}Y \mid \epsilon. \end{aligned}$$

$\mathrm{L}(X)$ contains all nonempty strings over $\{\mathtt{a}\}$ of lengths that are divisible by 3 and $\mathrm{L}(Y)$ has strings whose lengths are divisible by 7, including ϵ. Solving for X and Y,

$$X = X\mathtt{aaa} + \mathtt{aaa} \Rightarrow X - X\mathtt{aaa} = \mathtt{aaa} \Rightarrow X(\epsilon - \mathtt{aaa}) = \mathtt{aaa} \Rightarrow X = \frac{\mathtt{a}^3}{\epsilon - \mathtt{a}^3}, \text{ and}$$

$$Y = Y\mathtt{aaaaaaa} + \epsilon \Rightarrow Y - Y\mathtt{aaaaaaa} = \epsilon \Rightarrow Y(\epsilon - \mathtt{aaaaaaa}) = \epsilon \Rightarrow Y = \frac{\epsilon}{\epsilon - \mathtt{a}^7}.$$

The concatenation operation is commutative for languages defined over one-symbol alphabets and $\{\epsilon\} \equiv 1$ is always the concatenation $\equiv$ multiplication identity operand. Also, recall that languages over one-symbol alphabets are context-free iff they are regular, hence, $\mathrm{L}(Z)$ does have an unambiguous Type 3 grammar.

Let $\mathtt{a} = 1/2$ and examine the numeric power series for $\mathrm{L}(X)$ and $\mathrm{L}(Y)$, where

$$\mathcal{X} = \frac{\mathtt{a}^3}{\epsilon - \mathtt{a}^3} = \frac{1/8}{1-(1/8)} = \frac{1}{7} = \sum_{i=1}^{\infty}(1/8)^i = 0.14285714\ldots, \text{ and}$$

$$\mathcal{Y} = \frac{\epsilon}{\epsilon - \mathtt{a}^7} = \frac{1}{1-(1/128)} = \frac{128}{127} = \sum_{i=0}^{\infty}(1/128)^i = 1.00787401\ldots, \text{ respectively.}$$

Converting into base two, we obtain the coefficients for each $\mathtt{a}^j$ in $\mathrm{L}(X)$ and $\mathrm{L}(Y)$

$$\mathcal{X} = 0.14285714\ldots = 0.001001001001001001001\mathbf{1}00100_{(2)}\ldots, \text{ and}$$

$$\mathcal{Y} = 1.00787401\ldots = 1.000000100000010000001\mathbf{1}00000_{(2)}\ldots, \text{ respectively.}$$

Since $Z \to X \mid Y$, we have $\mathcal{Z} = \mathcal{X} + \mathcal{Y}$. So, the $\mathtt{a}^j$ coefficients in $\mathrm{L}(Z)$ are:

$$\mathcal{Z} = 1.15073115\ldots = 1.001001101001011001002\mathbf{2}00100_{(2)}\ldots$$

$$\mathrm{L}(Z) = \{\epsilon,\ \ \mathtt{a}^3,\ \ \mathtt{a}^6,\mathtt{a}^7,\mathtt{a}^9,\ \mathtt{a}^{12},\mathtt{a}^{14},\mathtt{a}^{15},\mathtt{a}^{18},\ \mathtt{a}^{\mathbf{21}},\ \mathtt{a}^{24},\ \ldots\ldots\}.\ N.B.\colon\ \mathtt{a}^0 = \epsilon \equiv 1.$$

The coefficient of "**2**" in the 21^{st} position past the binary point refers to $(1/2)^{21}$, that is, the two derivations of the string $\mathtt{a}^{21} = (\mathtt{aaa})^7$ in $\mathrm{L}(X)$ and $(\mathtt{aaaaaaa})^3$ in $\mathrm{L}(Y)$. This makes the union in the equation $\mathrm{L}(Z) = \mathrm{L}(X) \cup \mathrm{L}(Y)$ and, hence, G_2 ambiguous. If in G_2, we replaced $Z \to X \mid Y$ with $W \to XY$, and performed multiplication instead of addition, then $\mathcal{W} = 0.14398200\ldots = 0.00100100110110111111111\mathbf{2}1121_{(2)}\ldots$, would identify $\mathtt{a}^{24}$ as the shortest string with a concatenation ambiguity in $\mathrm{L}(W)$, where $\mathtt{a}^{24} = \mathtt{a}^3 \cdot (\mathtt{a}^7)^3 = (\mathtt{a}^3)^8 \cdot \epsilon$. □

The simple example above shows how numeric ps can sometimes be used to detect the presence of ambiguity by recognizing the specific strings with multiple derivations. Unfortunately, both the formal ps and numeric ps approaches do very little in terms of locating and possibly eliminating, or at least reducing, ambiguities in CFGs. This becomes very apparent when one investigates more complex grammars and their ps (see, e.g, [7], p.128-9).

7.2. *Language formulas with closures*

The definitions of positive and Kleene closure are based on combinations of union and concatenation operations. This makes the notion of ambiguous closure operations well defined and quite intuitive (Definition 4). The following identities in language formulas with closures are very useful in the detection and elimination of ambiguities:

- $A^*A^* \equiv A^*$.
  The concatenation in the left side of the identity is always ambiguous. Transforming such a construct into A^* eliminates that source of ambiguity.
- $A^+A^* \equiv A^*A^+ \equiv A^+$.
  The concatenation in both the left and middle sides of the identity is always ambiguous. Transforming either construct into A^+ eliminates that ambiguity.

- $(A^*)^* \equiv (A^+)^* \equiv (A^*)^+ \equiv A^*$.
  The top (outside) closures in the left three sides of the identity are all always ambiguous, and their transformation into A^* eliminates that ambiguity.
- $(A^+)^+ \equiv A^+$.
  The top closure in the left side is ambiguous, right side eliminates that particular source of ambiguity.
- $A(BA)^* \equiv (AB)^*A;\ A(BA)^+ \equiv (AB)^+A;\ (A\,|B)^* \equiv A^*(BA^*)^*$.
  The last three identities actually preserve the ambiguity status of the two sides. Their application, however, can help facilitate the application of one or more of the other identities involving closures, particularly the first one.

7.3. *Ambiguity-causing production rules and their transformations*

As shown, the algebraic characterization of CFGs provides a much more precise, refined, and purposeful way of describing ambiguity, its manifestation in terms of grammar constructs, and transformations that help in eliminating ambiguity from CFGs. Besides closure and its idempotency properties, additional observations and grammar transformations can be identified.

Observation 1. *$L{\cdot}M \subseteq L$ and $K{\cdot}L \subseteq L \Rightarrow$ union in $L{\cdot}M \cup K{\cdot}L$ is ambiguous.*

Proof. Let $x \in K$, $y \in L$, $z \in M$. Then, $xyz \in K{\cdot}L{\cdot}M = (K{\cdot}L){\cdot}M \subseteq L{\cdot}M$. Also, $xyz \in K{\cdot}L{\cdot}M = K{\cdot}(L{\cdot}M) \subseteq K{\cdot}L$. Therefore, $xyz \in L{\cdot}M \cap K{\cdot}L$. □

Corollary 1. *A CFG containing production rules of the form*

$$A \to \beta A \mid A\gamma \mid \delta$$

is ambiguous. They can be replaced with equivalent production rules

$$A \to \beta A \mid T$$
$$T \to T\gamma \mid \delta,$$

where T is a new nonterminal. □

Example 7. The production rules in G_1,

$$\texttt{E} \to \texttt{E+E} \mid \texttt{E*F} \mid \texttt{F},$$

have the form described in Corollary 1, where $A = \texttt{E}$, $\beta = \texttt{E+}$, $\gamma = \texttt{*F}$, and $\delta = \texttt{F}$. Therefore, they can be replaced with the production rules

$$\texttt{E} \to \texttt{E+E} \mid \texttt{T}$$
$$\texttt{T} \to \texttt{T*F} \mid \texttt{F},$$

and one can now show that the new grammar has no union ambiguities. □

Observation 2. $L{\cdot}M{\cdot}L \subseteq L \Rightarrow$ *language expression* $L{\cdot}M{\cdot}L$ *is ambiguous.*

Proof. Let $x_1, x_2, x_3 \in L$, and $y_1, y_2 \in M$. Consider $w_1 = x_1y_1x_2,\ u_1 = y_2x_3$ and $w_2 = x_1,\ u_2 = y_1x_2y_2x_3$. Now, we have $w_1u_1 = w_2u_2$, and $w_1 \neq w_2,\ u_1 \neq u_2$. Also, $w_1 = x_1y_1x_2 \in L{\cdot}M{\cdot}L \subseteq L,\ u_1 \in M{\cdot}L$, and $w_2 \in L,\ u_2 = y_1x_2y_2x_3 \in M{\cdot}(L{\cdot}M{\cdot}L) \subseteq M{\cdot}L$. So, $L{\cdot}M{\cdot}L$ is ambiguous. □

Corollary 2. *A CFG containing production rules of the form*

$$A \to A\beta A \mid \gamma$$

is ambiguous. They can be replaced with equivalent production rules

$$A \to A\beta\gamma \mid \gamma.$$ □

Example 8. The production rules introduced in Example 7,

$$\texttt{E} \to \texttt{E+E} \mid \texttt{T},$$

have the form described in Corollary 2, where $A = \texttt{E},\ \beta = \texttt{+}$, and $\gamma = \texttt{T}$.
Therefore, they can be replaced with the production rules

$$\texttt{E} \to \texttt{E+T} \mid \texttt{T}.$$

The final resulting grammar, which is equivalent to G_1, is shown below. It is provably unambiguous, since all of its union and concatenation operations are unambiguous.

$$\begin{aligned}
&\texttt{E} \to \texttt{E+T} \mid \texttt{T}\\
&\texttt{T} \to \texttt{T*F} \mid \texttt{F}\\
&\texttt{F} \to \texttt{(E)} \mid \texttt{n}.
\end{aligned}$$ □

8. Conclusion

An algebraic view of context-free grammars and languages providing new insights into the issue of ambiguity was presented. Our primary focus has been on the issues of undecidability, and the classification and detection of ambiguity, from theoretical and practical, i.e., algorithmic, perspectives, respectively. The topic of elimination and/or reduction of ambiguity has also been addressed. A number of useful transformations, which eliminate constructs that are certain-to-cause ambiguities in CFGs, language expressions, and closure operations were identified and analyzed.

The algebraic approach to CFGs and CFLs allows for a more precise and thorough investigation of ambiguity, including its reduction and/or outright elimination. The partition of ambiguity into its union, concatenation, and closure components also exposed the fact that the traditional ambiguity undecidability proofs were somewhat incomplete, as they did not address the undecidability results regarding the ambiguity of the concatenation and closure operations on ϵ-free CFLs.

Acknowledgments

This work was partially supported by NSF Grant #CCR-9215731.

References

[1] T. Altman and G. Logothetis, A note on ambiguity in Context-Free Grammars, *Information Processing Letters*, 35: **3**, (1990), pp. 111-114.
[2] J. Backus, The syntax and semantics of the proposed international algebraic language, *Proc. Intl. Conf. on Information Processing*, (1959), pp. 125-132.
[3] D. Cantor, On the ambiguity problem of Backus systems, *J. ACM* 9: **4**, (1962), pp. 477-479.
[4] N. Chomsky, Three models for the description of language, *IRE Trans. on Information Theory* 2: **3**,(1956), pp. 113-124.
[5] N. Chomsky, On certain formal properties of grammars, *Information and Control* 2: **2**, (1959), pp. 137-167.
[6] N. Chomsky, Formal properties of grammars, *Handbook of Math. Psych.*, Vol. 2, (John Wiley and Sons, New York, 1963).
[7] N. Chomsky and M. Schützenberger, The algebraic theory of context-free languages, in: P. Braffort and D. Hirschberg (Eds.), *Computer Programming and Formal Systems*, (North Holland, Amsterdam, 1975), pp. 118-161.
[8] R. Constable, H. Hunt, and S. Sahni, On the computational complexity of scheme equivalence, *Report No. 74-201*, Dept. of Comp. Science, Cornell University (1974).
[9] R. Floyd, On ambiguity in phrase structure languages, *CACM* 5: **10**, (1962), pp. 526-534.
[10] S. Ginsburg and H.G. Rice, Two families of languages related to ALGOL, *J. ACM* 9: **3**, (1962), pp. 350-371.
[11] S. Ginsburg, *The Mathematical Theory of Context-free Languages*, (McGraw-Hill, 1966).
[12] S. Ginsburg, *Algebraic and Automata-Theoretic Properties of Formal Languages*, (North Holland, Amsterdam, 1975).
[13] S. Greibach, The undecidability of the ambiguity problem for minimal linear grammars, *Information and Control* 6: **2**, (1963), pp. 119-125.
[14] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, (Addison-Wesley, 1979).
[15] W. Kuich, Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata, in: G. Rozenberg and A. Salomaa (Eds.), *Handbook of formal languages*, Vol. 1, Word, Language, Grammar, (Springer, 1997), pp. 609-677.
[16] P. Naur, et al., Report on the algorithmic language ALGOL 60, *Comm. ACM* 3: **5**,(1960), pp. 299-314, revised in *Comm. ACM* 6: **1**, (1963), pp. 1-17.
[17] A. Salomaa, *Formal Languages*, (Academic Press, 1973).
[18] A. Salomaa, *Jewels of Formal Language Theory*, (Computer Science Press, 1981).
[19] E. Shamir, Algebraic, Rational, and Context-Free Power Series in Noncommuting Variables, in: M. Arbib (Ed.), *Algebraic Theory of Machines, Languages, and Semigroups*, (Academic Press, 1968), pp. 329-341.