www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 10 October, 2013 Page No. 2921-2926

Analyzing Ambiguity of Context-Free Grammars

Akshay Kanwar, Aditi Khazanchi, Lovenish Saluja

Abstract

In computer science, an ambiguous grammar is a formal grammar for which there exists a string that can have more than one leftmost derivation, while an unambiguous grammar is a formal grammar for which every valid string has a unique leftmost derivation. For real-world programming languages, the reference CFG is often ambiguous, due to issues such as the dangling else problem. If present, these ambiguities are generally resolute by adding precedence rules or other context-sensitive parsing rules, so the overall phrase grammar is unambiguous.

It has been known since 1962 that the ambiguity problem for context-free grammars is undesirables. Ambiguity in context-free grammars is a frequent problem in language design and parser invention, as well as in applications where grammars are used as models of real-world physical structures.

We observe that there is a simple linguistic categorization of the grammar ambiguity problem, and we show how to develop this to conservatively approximate the problem based on local regular approximations and grammar unfolding. As an application, we consider grammars that occur in RNA analysis in bioinformatics, and we demonstrate that our static analysis of context-free grammars is sufficiently precise and efficient to be sensibly useful.

Introduction

In formal language theory, a context-free grammar (CFG) is a formal grammar in which every production rule is of the form

$V \rightarrow w$

Where V is a single non-terminal symbol, and w is a string of terminals and/or non-terminals (w can be empty). A formal grammar is considered "context free" when its production rules can be applied apart from of the context of a non-terminal. It does not matter which symbols the non-terminal is surrounded by, the single non-terminal on the left hand side can always be replaced by the right hand side.

Languages generated by context-free grammars are known as context-free languages (CFL). Different Context Free grammars can generate the same context free language. It is important to discriminate properties of the language (intrinsic properties) from properties of a particular grammar (extrinsic properties). Given two context free grammars, the language equality question (do they generate the same language?) is undesirable.

Context-free grammars are important in linguistics for describing the structure of sentences and words in natural language, and in computer science for recitation the structure of programming languages and other formal languages.

In linguistics, some authors use the term phrase structure grammar to refer to context-free grammars, whereby phrase structure grammars are distinct from craving grammars. In computer science, a popular notation for context-free grammars is Backus–Naur Form, or BNF.

1.1.Overview:

Characterization of grammar ambiguity that allows us to reason about the language of the non-terminals in the grammar rather than the structure of the grammar. In particular, we reformulate the ambiguity problem in terms of language roundabout and overlap operations. Based on this characterization, we formulate a general frame- work for conservatively approximating the ambiguity problem. In another section we show how regular approximations can be used to obtain a particular decidable approximation. Next section discusses applications in the area of bio sequence analysis where context-free grammars are used to describe RNA structures. It also summarizes a number of experiments that test the accuracy and performance of the analysis. In the appendices, we show how the precision can be improved by selectively unfolding parts of the given grammar, and we provide proofs of the propositions.

A context-free grammar *G* is defined by the 4-tuple $G = (V\,, \Sigma\,, R\,, S\,)$ where

1. V is a finite set; each element $v \in V$ is called *a non-terminal character* or a *variable*. Each variable represents a different type of phrase or clause in the sentence. Variables are also sometimes called syntactic categories. Each variable defines a sub-language of the language defined by G.
2. Σ is a finite set of *terminal*s, disjoint from V, which make up the actual content of the sentence. The set of terminals is the alphabet of the language defined by the grammar G.
3. R is a finite relation from V to $(V \cup \Sigma)^*$, where the asterisk represents the KLeene star operation. The members of R are called the *(rewrite) rule*s or *production*s of the grammar. (also commonly symbolized by a P)
4. S is the start variable (or start symbol), used to represent the whole sentence (or program). It must be an element of V.

Production rule notation

A production rule in R is formalized mathematically as a pair $(\alpha, \beta) \in R$, where $\alpha \in V$ is a non-terminal and $\beta \in (V \cup \Sigma)^*$ is a string of variables and/or terminals; rather than using ordered pair notation, production rules are usually written using an arrow operator with α as its left hand side and β as its right hand side: $\alpha \rightarrow \beta$.

It is allowed for β to be the empty string, and in this case it is customary to denote it by ε. The form $\alpha \rightarrow \varepsilon$ is called an ε-production.

It is common to list all right-hand sides for the same left-hand side on the same line, using | (the pipe symbol) to separate them. Rules $\alpha \rightarrow \beta_1$ and $\alpha \rightarrow \beta_2$ can hence be written as $\alpha \rightarrow \beta_1 \mid \beta_2$.

Rule application

For any strings $u, v \in (V \cup \Sigma)^*$, we say u directly yields v, written as $u \Rightarrow v$, if $\exists (\alpha, \beta) \in R$ with $\alpha \in V$ and $u_1, u_2 \in (V \cup \Sigma)^*$ such that $u \,= u_1 \alpha u_2$ and $v \,= u_1 \beta u_2$. Thus, v is the result of applying the rule (α, β) to u.

Repetitive rule application

For any $u, v \in (V \cup \Sigma)^*$, we say u yields v written as $u \overset{*}{\Rightarrow} v$ (or $u \Rightarrow\Rightarrow v$ in some textbooks), if $\exists\ u_1, u_2, \cdots u_k \in (V \cup \Sigma)^*, k \geq 0$ such that $u \Rightarrow u_1 \Rightarrow u_2 \cdots \Rightarrow u_k \Rightarrow v$

Context-free language

The language of a grammar $G = (V\,, \Sigma\,, R\,, S\,)$ is the set

$$L(G) = \{w \in \Sigma^* : S \overset{*}{\Rightarrow} w\}$$

A language L is said to be a context-free language (CFL), if there exists a CFG G, such that $L \,=\, L(G)$.

Proper CFG's

A context-free grammar is said to be *proper*, if it has

- no *inaccessible* symbols: $\forall N \in V : \exists \alpha, \beta \in (V \cup \Sigma)^* : S \overset{*}{\Rightarrow} \alpha N \beta$
- no *unproductive* symbols: $\forall N \in V : \exists w \in \Sigma^* : N \overset{*}{\Rightarrow} w$
- no ε-productions: $\forall N \in V, w \in \Sigma^* : (N, w) \in R \Rightarrow w \neq$ ε
  (the right-arrow in this case denotes logical "implies" and not grammatical "yields")
- no cycles: $\neg \exists N \in V : N \overset{*}{\Rightarrow} N$

Example

The grammar $G = (\{S\}, \{a, b\}, P, S)$, with productions

$S \rightarrow aSa,$

$S \rightarrow bSb,$

$S \rightarrow \varepsilon,$

is context-free. It is not proper since it includes an ε-production. A typical derivation in this grammar is

$S \rightarrow aSa \rightarrow aaSaa \rightarrow aabSbaa \rightarrow aabbaa.$

This makes it clear that $L(G) = \{ww^R : w \in \{a, b\}^*\}$. The language is context-free, however it can be proved that it is not regular.

2. A Characterization of Grammar Ambiguity

We begin by briefly recapitulating the basic terminology about context-free grammars.

Definition 1 (Context-free grammar and ambiguity).

A context-free gram- mar (CFG) G is defined by G = (N , Σ , s, π) where N is a finite set of non-terminals, Σ is a finite set of alphabet symbols (or terminals), s ∈N is the start non-terminal, and $\pi : N \rightarrow P(E*)$ is the production function where $E = \Sigma \cup N$.

We write $\alpha n \omega \Rightarrow \alpha\theta\omega$ when $\theta \in \pi(n)$ and $\alpha, \omega \in E*$, and $\Rightarrow*$ is the reflexive transitive closure of ⇒. We assume that every non-terminal n ∈N is reachable from s and derives some string that is, $\exists\alpha, \varphi, \omega \in \Sigma*$: $s \Rightarrow* \alpha n\omega \Rightarrow* \alpha\varphi\omega$. The language of a sentential form $\alpha \in E*$ is $LG(\alpha) = \{x \in \Sigma * \mid \alpha \Rightarrow* x\}$, and the language of G is L (G) = LG (s).

Assume that x ∈ L (G), that is, $s = \varphi 0 \Rightarrow \varphi 1 \Rightarrow ... \Rightarrow \varphi n = x$. Such a

derivation sequence gives rise to a derivation tree where each node is labelled with a symbol from E, the root is labelled s, leaves are labelled from Σ , and the labels of children of a node with label e are in π(e). G is ambiguous if there exists a string x in L(G) with multiple derivation trees, and we then say that x is ambiguous relative to G.

We now introduce the properties vertical and horizontal unambiguity and show that they together characterize grammar unambiguity.

Definition 2 (Vertical and horizontal unambiguity).

A grammar G is vertically unambiguous if

$\forall n \in N , \alpha, \alpha 0 \in \pi(n), \alpha = \alpha 0 :\ LG(\alpha) \cap LG(\alpha 0) = \emptyset$

A grammar G is horizontally unambiguous if

$\forall n \in N, \alpha \in \pi(n), i \in \{1... |\alpha|-1\}:\ LG(\alpha 0 \cdots \alpha i-1) \cap LG(\alpha i \cdots \alpha|\alpha|-1) = \emptyset$

Where ∩ is the language overlap operator defined by

$X \cap Y = \{ xay \mid x, y \in \Sigma * \wedge a \in \Sigma + \wedge x, xa \in X \wedge y, ay \in Y \}$

Intuitively, vertical unambiguity means that, during parsing of a string, there is never a choice between two different productions of a non-terminal. The overlap X ∩ Y is the set of strings in XY that can be split non-uniquely in an X part and a Y part. For example, if X = {x, xa} and Y = {a, ay} then X ∩ Y = {xay}.

Horizontal unambiguity then means that, when parsing a string according to a production, there is never any choice of how to split the string into substrings corresponding to the entities in the production.

Definition 3 (Characterization of Ambiguity).

G is vertically and horizontally unambiguous ⇔ G is unambiguous

Proof: Intuitively, any ambiguity must result from a choice between two productions of some non-terminal or from a choice of how to split a string according to a single production.

This scheme essentially means that we have transformed the problem of context-free grammar ambiguity from a grammatical property to a linguistic property dealing solely with the languages of the non-terminals in the grammar rather than with derivation trees. As we shall see in the next section, this characterization can be exploited to obtain a good conservative approximation for the problem without violating the two requirements described. Note that this linguistic characterization of grammar ambiguity should not be perplexed with the notion of inherently ambiguous languages/ (A language is inherently ambiguous if all its grammars are ambiguous.)

3. A Framework for Conservative Approximation

The characterization of ambiguity presented above can be used as a foundation for a agenda for obtaining decidable, conservative approximations of the ambiguity problem. When the analysis says “unambiguous grammar”, we know that this is indeed the case. The key to this technique is that the linguistic characterization allows us to reason about languages of non-terminals rather than derivation trees.

Definition 6 (Grammar over-approximation).

A grammar over-approximation relative to a CFG G is a function AG : E∗ → P (Σ ∗) where LG (α) ⊆ AG (α) for every α ∈ E∗. An approximation strategy A is a function that returns a grammar over approximation AG given a CFG G.

Definition 7 (Approximated vertical and horizontal unambiguity).

A grammar G is vertically unambiguous relative to a grammar over-approximation AG if

∀n ∈ N, α, α0 ∈ π(n),α = α0 : AG (α) ∩ AG (α0) = ∅

Similarly, G is horizontally unambiguous relative to AG if

∀n ∈ N ,α ∈ π(n),i ∈ {1,... , |α|−1} : AG (α0 ••• αi−1) ∩ AG (αi ••• α|α|−1) = ∅ Finally, we say that an approximation strategy A is decidable if the following problem is decidable: “Given a grammar G, is G vertically and horizontally unambiguous relative to AG ?”

Definition 8 (Approximation soundness).

If G is vertically and horizon- tally unambiguous relative to AG then G is unambiguous.

4. Regular Approximation

One approach for obtaining decidability is to consider regular approximations, that is, ones where AG (α) is a regular language for each α: the family of regular languages is closed under both connection and overlap, and emptiness on regular languages is decidable. Also, shortest examples can easily be extracted from non-empty regular languages. As a concrete approximation strategy we propose using Mohri and Nederhof’s algorithm for constructing regular approximations of context-free grammars.

We will not repeat their algorithm in detail, but some important properties are worth mentioning. Given a CFG G, the approximation results in another

CFG G0 which is right linear (and hence its language is regular), L(G) ⊆ L(G0), and G0 is at most twice the size of G. Whenever n ⇒∗ αnω and n ⇒∗ θ in G for some α, ω, θ ∈ E and n ∈ N , the grammar G0 has the property that n ⇒∗ αm θωk for any m, k. Intuitively, G0 keeps track of the order that alphabet symbols may appear in, but it loses track of the fact that α and ω must appear in balance.

Definition 9 (Mohri-Nederhof approximation strategy).

Let MN be the approximation strategy that given a CFG G = (N , Σ , s, π) returns the grammar over-approximation MN G defined by MN G (α) = L(Gα) where Gα is the Mohri- Nederhof approximation of the grammar (N ∪{sα}, Σ , sα, π[sα 7→ {α}]) for some sα 6∈ N .

In other words, whenever we need to compute AG (α) for some α ∈ E∗, we apply Mohri and Nederhof’s approximation algorithm to the grammar G modified to derive α as the first step.

5. Application to Bio-sequence Analysis

The languages of bio-sequences are trivial from the formal language point of view. The alphabet of DNA is ΣDNA = {A, C, G, T}, of RNA it is ΣRNA = {A, C, G, U}, and for proteins it is a 20 letter amino acid code. In each case, the language of bio-sequences is Σ∗. Bio-sequence analysis relates two sequences to each other or one sequence to itself. The latter is our application domain-RNA structure analysis.

RNA is a chain molecule, built from the four bases adenine (A), cytosine (C), guanine (G), and uracil (U), connected via a backbone of sugar and phosphate. Mathematically, it is a string over ΣRNA of moderate length (compared to genomic DNA), ranging from 20 to 10,000 bases.

RNA forms structure by folding back on itself. Certain bases, located at different positions in the backbone, may form hydrogen bonds. Such bonded base pairs arise between complementary bases G − C, A − U, and G − U. By forming these bonds, the two pairing bases are arranged in a plain, and this inturn enables them to stack very densely onto adjacent bases also forming pairs. Helical structures arise, which are energetically stable and mechanically rather stiff. They enable RNA to perform its wide variety of functions.

Because of the backbone turning back on itself, RNA structures can be viewed as palindromic languages.

Starting from palindromes in the traditional we can characterize palindrome languages for RNA structure via five generalizations:

a. A letter does not match to itself but to a complementary.
b. The two arms of a palindrome may be separated by a non-palindrome string called a loop.
c. The two arms of the palindrome may hold non-pairing bases called bulges.
d. A string may hold several adjacent palindromes separated by unpaired bases
e. Palindromes can be recursively nested, that is, a loop or a bulge may contain further palindromes.

A language is a context-free language (CFL) if all of its strings are generated by a context-free grammar.

A 4-tuple G = < V , Σ, S , P > is a context-free grammar (CFG) if V and Σ are finite sets sharing no elements between them, S $\in$ V is the start symbol, and P is a finite set of productions of the form X -> α, where X $\in$ V , and $\alpha \in (V \cup \Sigma)^*$.

Example 1: $L_1 = \{ a^n b^n$ | n is a positive integer } is a context-free language. For the following context-free grammar G_1 = < V_1 , Σ, S , P_1 > generates L_1 : V_1 = { S } , Σ = { a , b } and P_1 = { S -> aSb , S -> ab }.

Example 2: $L_2 = \{ ww^r | w \in \{a, b\}^+ \}$ is a context-free language , where w is a non-empty string and w^r denotes the reversal of string w, that is, w is spelled backward to obtain w^r . For the following context-free grammar G_2 = < V_2 , Σ, S , P_2 > generates L_2 : V_2 = { S } , Σ = { a , b } and P_2 = { S -> aSa , S -> bSb , S -> aa , S -> bb }.

Example 3: Let L_3 be the set of algebraic expressions involving identifiers x and y, operations + and * and left and right parentheses. Then L_3 is a context-free language. For the following context-free grammar G_3 = < V_3 , Σ_3, S , P_3 > generates L_3 : V_3 = { S } , Σ_3 = { x , y , (,) , + , * } and P_3 = { S -> (S + S) , S -> S*S , S -> x , S -> y }.

6. Properties of Context-Free Language

Theorem 1: Let L_1 and L_2 be context-free languages. Then $L_1 \cup L_2$, L_1L_2 , and L_1^* are context-free languages.

Outline of Proof:

This theorem can be verified by constructing context-free grammars for union, concatenation and Kleene star of context-free grammars as follows:

Let G_1 = < V_1, Σ, S_1 , P_1 > and G_2 = < V_2 , Σ , S_2 , P_2 > be context-free grammars generating L_1 and L_2 , respectively.

Then for $L_1 \cup L_2$, first re-label symbols of V_2, if necessary, so that V_1 and V_2 don't share any symbols. Then let S_u be a symbol which is not in $V_1 \cup V_2$. Next define $V_u = V_1 \cup V_2 \cup \{ S_u \}$ and $P_u = P_1 \cup P_2 \cup \{ S_u \text{->} S_1 , S_u \text{->} S_2 \}$.

Then it can be easily seen that G_u = < V_u, Σ, S_u , P_u > is a context-free grammar that generates the language $L_1 \cup L_2$.

Similarly for L_1L_2, first re-label symbols of V_2, if necessary, so that V_1 and V_2 don't share any symbols. Then let S_c be a symbol which is not in $V_1 \cup V_2$. Next define $V_c = V_1 \cup V_2 \cup \{ S_c \}$ and $P_c = P_1 \cup P_2 \cup \{ S_c \text{->} S_1S_2 \}$. Then it can be easily seen that G_c = < V_c , Σ , S_c , P_c > is a context-free grammar that generates the language L_1L_2 .

For L_1^*, let S_s be a symbol which is not in V_1. Then let $P_s = P_1 \cup \{S_s \text{->} S_sS_1, S_s \text{->} \Lambda \}$. It can be seen that the grammar G_s = < V_s, Σ, S_s , P_s > is a context-free grammar that generates the language L_1^* .

Pushdown Automata

Like regular languages which are accepted by finite automata, context-free languages are also accepted by automata but not finite automata. They need little more complex automata called pushdown automata. Let us consider a context-free language $a^n b^n$. Any string of this language can be tested for the membership for the language by a finite automaton if there is a memory such as a pushdown stack that can store a's of a given input string. For example, as a's are read by the finite automaton, push them into the stack. As soon as the symbol b appears stop storing a's and start popping a's one by one every time a b is read. If another (or anything other than b) is read after the first b, reject the string. When all the symbols of the input string are read, check the stack. If it is empty, accept the string. Otherwise reject it. This automaton behaves like a finite automaton except the following two points: First, its next state is determined not only by the input symbol being read, but also by the symbol at the top of the stack. Second, the contents of the stack can also be changed every time an input symbol is read. Thus its transition function specifies the new top of the stack contents as well as the next state.

Let us define this new type of automaton formally.

A pushdown automata (or PDA) is a 7-tuple M = < Q , Σ , Γ , q_0 , Z_0 , A , δ >,where Q is a finite set of states, Σ and Γ are finite sets (the input and stack alphabet,respectively).
q_0 is the initial state, Z_0 is the initial stack symbol and it is a member of Γ , A is the set of accepting states δ is the transition function and

9.

$$\delta : Q \times (\Sigma \cup (\Lambda \} \times \Gamma \to 2^{Q \times \Gamma^*}.$$

Conclusion

We have obtainable a technique for statically analyzing ambiguity of context-free grammars. Based on a linguistic characterization, the technique allows the use of grammar transformations, in particular regular approximation and recitation, without sacrificing reliability. Moreover, the analysis is often able to identify sources of ambiguity through existing examples being automatically generated. The analysis may be used when LR(k) and related techniques are insufficient, for example in bio-sequence analysis, as our examples show. Our experiments indicate that the precision, the speed, and the quality of warning messages are ample to be practically useful.

References

1. http://en.wikipedia.org/wiki/Ambiguous_grammar
2. http://en.wikipedia.org/wiki/Context-free_grammar
3. http://www.cs.odu.edu/~toida/nerzic/390teched/cfl/cfg.html
4. www.cs.rochester.edu/~nelson/courses/csc_173/**grammars/cfg**.html
5. lambda.uta.edu/cse5317/notes/node12.html
6. cs.union.edu/~striegnk/courses/nlp-with-prolog/html/node37.html
7. http://www.cs.cornell.edu/people/tj/svm_light/svm_cfg.html
8. http://www.cse.ohio-state.edu/~gurari/theory-bk/theory-bk-threese3.html