Analyzing Ambiguity of Context-Free Grammars

Claus Brabrand[1], Robert Giegerich[2], and Anders Møller[1]

[1] BRICS, Department of Computer Science (DAIMI), University of Aarhus, Denmark
{brabrand,amoeller}@brics.dk
[2] Practical Computer Science, Faculty of Technology, Bielefeld University, Germany
robert@techfak.uni-bielefeld.de

Abstract. It has been known since 1962 that the ambiguity problem for context-free grammars is undecidable. Ambiguity in context-free grammars is a recurring problem in language design and parser generation, as well as in applications where grammars are used as models of real-world physical structures.
We observe that there is a simple linguistic characterization of the grammar ambiguity problem, and we show how to exploit this to conservatively approximate the problem based on local regular approximations and grammar unfoldings. 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 practically useful.

1 Introduction

When using context-free grammars to describe formal languages, one has to be aware of potential ambiguity in the grammars, that is, the situation where a string may be parsed in multiple ways, leading to different parse trees. We propose a technique for detecting ambiguities in a given grammar. As the problem is in general undecidable [9] we resort to conservative approximation. This is much like, for example, building an LR(k) parse table for the given grammar and checking for conflicts. The analysis we propose has two significant advantages:

1. The LR(k) condition has since its discovery by Knuth in 1965 been known as a powerful test for unambiguity [11]. An example of an even larger class of unambiguous grammars is LR-Regular [10] but using that requires manually providing regular lookahead sets. Moreover, not even LR-Regular is sufficient for a considerable class of grammars involving palindromic structures, which our technique can handle.
2. The ambiguity warnings that our approach can produce if a potential ambiguity is detected are more human readable than those typically produced by, for example, Bison [19]. (Any user of Bison or a similar tool will recognize the difficulty in finding the true cause of a conflict being reported.) Our technique is in many cases able to produce shortest possible examples of strings that may be parsed in multiple ways and precisely identify the relevant location in the grammar, thereby pinpointing the cause of the ambiguity.

In recent years an increasing number of parser generators, for example, Bison [19], Elkhound [13], and SDF [21], support general context-free grammars rather than unambiguous subclasses, such as, LL(k), LR(k), or LALR(k). Such tools usually handle ambiguities by dynamically (that is, during parsing) disambiguating or merging the resulting parse trees [20, 3]. In contrast, our approach is to *statically* analyze the grammars for potential ambiguities.

In bioinformatics, context free grammars in various guises have important applications, for example in sequence comparison, motif search, and RNA secondary structure analysis [6, 8]. Recently, ambiguity has gained attention in this field, as several important algorithms (such as the Viterbi algorithm on stochastic CFGs) have been shown to deliver incorrect results in the presence of ambiguity [7, 5]. The ambiguity problem arises in biosequence analysis from the necessity to check a static property of the dynamic programming algorithms employed – the question whether or not an element of the search space may be evaluated more than once. If so, probabilistic scoring schemes yield incorrect results, and enumeration of near-optimal solutions drowns in redundancy. It may seem surprising that the static analysis of this program property can be approached as a question of language ambiguity on the formal language level. We will explain this situation in some detail in Section 5.

Before we start presenting our method, we want to state two requirements on a practical ambiguity checker that result from this application domain, and must be kept in mind in the sequel: First, the grammars to be checked are actually abstractions from richer programming concepts. They may look strange from a formal language design point of view – for example, they may contain "redundant" nonterminal symbols generating the same language. However, different nonterminals model different physical structures with different semantics that are essential for subsequent algorithmic processing. Hence, the grammar must be checked as is, and cannot be transformed or simplified by, for instance, coalescing such nonterminal symbols. Second, the domain experts are typically molecular biologists with little programming expertise and no training in formal language theory. Hence, when ambiguity is discovered, it must be reported in a way that is meaningful to this category of users.

Besides the applications to biosequence analysis, our motivation behind the work we present here has been analyzing reversibility of transformations between XML and non-XML data [2].

Contributions Despite decades of work on parsing techniques, which in many cases involve the problem of grammar ambiguity, we have been unable to find tools that are applicable to grammars in the area of biosequence analysis. This paper contributes with the following results:

- We observe that there is a simple linguistic characterization of grammar ambiguity. This allows us to shift from reasoning about grammar derivations to reasoning about purely linguistic properties, such as, language inclusion.
- We show how Mohri and Nederhof's regular approximation technique for context-free grammars [14] can be adapted in a local manner to detect many

common sources of ambiguity, including ones that involve palindromic structures. Also, a simple grammar unfolding trick can be used to improve the precision of the analysis.
– We demonstrate that our method can handle "real-world" grammars of varying complexity taken from the bioinformatics literature on RNA analysis, acquitting the unambiguous grammars and pinpointing the sources of ambiguity (with shortest possible examples as witnesses) in two grammars that are in fact ambiguous.

We here work with plain, context-free grammars. Generalizing our approach to work with parsing techniques that involve interoperability with a lexical analyzer, precedence/associativity declarations, or other disambiguation mechanisms is left to future work.

Overview We begin in Section 2 by giving a characterization of grammar ambiguity that allows us to reason about the *language* of the nonterminals in the grammar rather than the *structure* of the grammar. In particular, we reformulate the ambiguity problem in terms of language intersection and overlap operations. Based on this characterization, we then in Section 3 formulate a general framework for conservatively approximating the ambiguity problem. In Section 4 we show how regular approximations can be used to obtain a particular decidable approximation. Section 5 discusses applications in the area of biosequence analysis where context-free grammars are used to describe RNA structures. It also summarizes a number of experiments that test the precision 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.

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 grammar (CFG) *G is defined by $G = (\mathcal{N}, \Sigma, s, \pi)$ where $\mathcal{N}$ is a finite set of* nonterminals*, Σ is a finite set of alphabet symbols (or* terminals*), $s \in \mathcal{N}$ is the* start nonterminal*, and $\pi : \mathcal{N} \to \mathcal{P}(E^*)$ is the* production function *where $E = \Sigma \cup \mathcal{N}$. 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 $\Rightarrow$. We assume that every nonterminal $n \in \mathcal{N}$ is reachable from s and derives some string, that is, $\exists \alpha, \phi, \omega \in \Sigma^* : s \Rightarrow^* \alpha n \omega \Rightarrow^* \alpha\phi\omega$. The* language *of a sentential form $\alpha \in E^*$ is $\mathcal{L}_G(\alpha) = \{x \in \Sigma^* \mid \alpha \Rightarrow^* x\}$, and the language of G is $\mathcal{L}(G) = \mathcal{L}_G(s)$.*

Assume that $x \in \mathcal{L}(G)$, that is, $s = \phi_0 \Rightarrow \phi_1 \Rightarrow \ldots \Rightarrow \phi_n = x$. Such a derivation sequence gives rise to a derivation tree *where each node is labeled with a symbol from E, the root is labeled s, leaves are labeled from Σ, and the*

labels of children of a node with label e are in $\pi(e)$. *G is* ambiguous *if there exists a string x in* $\mathcal{L}(G)$ *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 *iff*

$$\forall n \in \mathcal{N}, \alpha, \alpha' \in \pi(n), \alpha \neq \alpha' : \ \mathcal{L}_G(\alpha) \cap \mathcal{L}_G(\alpha') = \emptyset$$

A grammar G is horizontally unambiguous *iff*

$$\forall n \in \mathcal{N}, \alpha \in \pi(n), i \in \{1, \ldots, |\alpha|-1\} : \ \mathcal{L}_G(\alpha_0 \cdots \alpha_{i-1}) \between \mathcal{L}_G(\alpha_i \cdots \alpha_{|\alpha|-1}) = \emptyset$$

where $\between$ *is the* language overlap *operator defined by* $X \between Y = \{\ xay \mid x, y \in \Sigma^* \wedge a \in \Sigma^+ \wedge x, xa \in X \wedge y, ay \in Y\}$

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

Proposition 3 (Characterization of Ambiguity).

G is vertically and horizontally unambiguous $\quad\Leftrightarrow\quad$ *G is unambiguous*

Proof. Intuitively, any ambiguity must result from a choice between two productions of some nonterminal or from a choice of how to split a string according to a single production. A detailed proof is given in Appendix B.

This proposition 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 nonterminals 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 in Section 1.

Note that this linguistic characterization of grammar ambiguity should not be confused with the notion of *inherently ambiguous languages* [9]. (A language is inherently ambiguous if all its grammars are ambiguous.)

We now give examples of vertical and horizontal ambiguities.

Example 4 (a vertically ambiguous grammar).

```
Z   :   A 'y'
    |   'x' B    ;
A   :   'x' 'a'  ;
B   :   'a' 'y'  ;
```

The string `xay` can be parsed in two ways by choosing either the first or the second production of `Z`. The name *vertical* ambiguity comes from the fact that productions are often written on separate lines as in this example.

Example 5 (a horizontally ambiguous grammar).

```
Z   :   ['x' A] ↔ [B]   ;
A   :   'a'
    |   ε               ;
B   :   'a' 'y'
    |   'y'             ;
```

Also here, the string `xay` can be parsed in two ways, by parsing the `a` either in `['x' A]` (using the first production of `A` and the second of `B`) or in `[B]` (using the second production of `A` and the first of `B`). Here, the ambiguity is at a split-point between entities on the right-hand side of a particular production, hence the name *horizontal* ambiguity.

3 A Framework for Conservative Approximation

The characterization of ambiguity presented above can be used as a foundation for a framework 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 nonterminals rather than derivation trees.

Definition 6 (Grammar over-approximation). *A* grammar over-approximation *relative to a CFG G is a function $\mathcal{A}_G : E^* \to \mathcal{P}(\Sigma^*)$ where $\mathcal{L}_G(\alpha) \subseteq \mathcal{A}_G(\alpha)$ for every $\alpha \in E^*$. An* approximation strategy $\mathcal{A}$ *is a function that returns a grammar over-approximation $\mathcal{A}_G$ given a CFG G.*

Definition 7 (Approximated vertical and horizontal unambiguity). *A grammar G is* vertically unambiguous relative to a grammar over-approximation $\mathcal{A}_G$ *iff*

$$\forall n \in \mathcal{N}, \alpha, \alpha' \in \pi(n), \alpha \neq \alpha' : \ \mathcal{A}_G(\alpha) \cap \mathcal{A}_G(\alpha') = \emptyset$$

Similarly, G is horizontally unambiguous relative to $\mathcal{A}_G$ *iff*

$$\forall n \in \mathcal{N}, \alpha \in \pi(n), i \in \{1, \ldots, |\alpha|-1\} : \ \mathcal{A}_G(\alpha_0 \cdots \alpha_{i-1}) \between \mathcal{A}_G(\alpha_i \cdots \alpha_{|\alpha|-1}) = \emptyset$$

Finally, we say that an approximation strategy $\mathcal{A}$ is decidable *if the following problem is decidable: "given a grammar G, is G vertically and horizontally unambiguous relative to $\mathcal{A}_G$?"*

Proposition 8 (Approximation soundness). *If G is vertically and horizontally unambiguous relative to $\mathcal{A}_G$ then G is unambiguous.*

Proof. The result follows straightforwardly from Definitions 2, 6, and 7 and Proposition 3. For details, see Appendix C.

As an example of a decidable but not very useful approximation strategy, the one which returns the constant Σ^* approximation corresponds to the trivial analysis that reports that every grammar may be (vertically and horizontally) ambiguous at all possible locations. In the other end of the spectrum, the approximation strategy which for every grammar G returns $\mathcal{L}_G(\alpha)$ for each α has full precision but is generally undecidable (since it involves checking emptiness of intersections of context-free languages).

Also note that two different approximations, $\mathcal{A}_G$ and $\mathcal{A}'_G$, may be combined: the function $\mathcal{A}''_G$ defined by $\mathcal{A}''_G(\alpha) = \mathcal{A}_G(\alpha) \cap \mathcal{A}'_G(\alpha)$ is a grammar over-approximation that subsumes both $\mathcal{A}_G$ and $\mathcal{A}'_G$. Such a pointwise combination is generally better than running the two analyses independently as one of the approximations might be good in one part of the grammar, and the other in another part.

4 Regular Approximation

One approach for obtaining decidability is to consider *regular* approximations, that is, ones where $\mathcal{A}_G(\alpha)$ is a regular language for each α: the family of regular languages is closed under both intersection and overlap, and emptiness on regular languages is decidable (for an implementation, see [15]). 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 [14].

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 G' which is right linear (and hence its language is regular), $\mathcal{L}(G) \subseteq \mathcal{L}(G')$, and G' is at most twice the size of G. Whenever $n \Rightarrow^* \alpha n \omega$ and $n \Rightarrow^* \theta$ in G for some $\alpha, \omega, \theta \in E$ and $n \in \mathcal{N}$, the grammar G' has the property that $n \Rightarrow^* \alpha^m \theta \omega^k$ for any m, k. Intuitively, G' 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 $G = (\mathcal{N}, \Sigma, s, \pi)$ returns the grammar over-approximation MN_G defined by $MN_G(\alpha) = \mathcal{L}(G_\alpha)$ where G_α is the Mohri-Nederhof approximation of the grammar $(\mathcal{N} \cup \{s_\alpha\}, \Sigma, s_\alpha, \pi[s_\alpha \mapsto \{\alpha\}])$ for some $s_\alpha \notin \mathcal{N}$.*

In other words, whenever we need to compute $\mathcal{A}_G(\alpha)$ for some $\alpha \in E^*$, we apply Mohri and Nederhof's approximation algorithm to the grammar G modified to derive α as the first step.

Example 10 (palindromes). A classical example of an unambiguous grammar that is not LR(k) (nor LR-Regular) is the following whose language consists of all palindromes over the alphabet $\{\mathtt{a}, \mathtt{b}\}$:

```
P : 'a' P 'a' | 'b' P 'b' | 'a' | 'b' | ε ;
```

Running our analysis on this grammar immediately gives the result "`unambiguous grammar!`". It computes MN_G for each of the five right-hand sides of productions and all their prefixes and suffixes and then performs the checks described in Definition 7. As an example, MN_G(`'a' P 'a'`) is the regular language $\mathtt{a}(\mathtt{a}+\mathtt{b})^*\mathtt{a}$, and MN_G(`'b' P 'b'`) is $\mathtt{b}(\mathtt{a}+\mathtt{b})^*\mathtt{b}$. Since these two languages are disjoint, there is no vertical ambiguity between the first two productions.

A variant of the grammar above is the following language, AntiPalindromes, which our analysis also verifies to be unambiguous:

```
R : 'a' R 'b' | 'b' R 'a' | 'a' | 'b' | ε ;
```

As we shall see in Section 5, this grammar is closely related to grammars occurring naturally in biosequence analysis.

Example 11 (ambiguous expressions). To demonstrate the capabilities of producing useful warning messages, let us run the analysis on the following tiny ambiguous grammar representing simple arithmetical expressions:

```
Exp[plus]   :   Exp '+' Exp
   [mult]   |   Exp '*' Exp
   [var]    |   'x'          ;
```

(Notice that we allow productions to be labeled.) The analysis output is

```
*** vertical ambiguity: E[plus] <--> E[mult]
    ambiguous string: "x*x+x"
*** horizontal ambiguity at E[plus]: Exp <--> '+' Exp
    ambiguous string: "x+x+x"
*** horizontal ambiguity at E[plus]: Exp '+' <--> Exp
    ambiguous string: "x+x+x"
*** horizontal ambiguity at E[mult]: Exp <--> '*' Exp
    ambiguous string: "x*x*x"
*** horizontal ambiguity at E[mult]: Exp '*' <--> Exp
    ambiguous string: "x*x*x"
```

Each source of ambiguity is clearly identified, even with example strings that have been verified to be non-spurious (of course, it is easy to check with a CFG parser whether a concrete string is ambiguous or not). Obviously, these messages are more useful to a non-expert than, for example, the shift/reduce conflicts and reduce/reduce conflicts being reported by Bison.

5 Application to Biosequence Analysis

The languages of biosequences are trivial from the formal language point of view. The alphabet of DNA is $\Sigma_{\mathtt{DNA}} = \{A, C, G, T\}$, of RNA it is $\Sigma_{\mathtt{RNA}} = \{A, C, G, U\}$, and for proteins it is a 20 letter amino acid code. In each case, the language of biosequences is Σ^*. Biosequence analysis relates two sequences to each other (sequence alignment, similarity search) or one sequence to itself (folding). 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 phos-

phate. Mathematically, it is a string over $\Sigma_{\texttt{RNA}}$ 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 in turn 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 sense (as described in Example 10) we can characterize palindromic languages for RNA structure via five generalizations: (1) a letter does not match to itself but to a complementary base (cf. Example 12); (2) the two arms of a palindrome may be separated by a non-palindromic string (of length at least 3) called a *loop*; (3) the two arms of the palindrome may hold non-pairing bases called *bulges*; (4) a string may hold several adjacent palindromes separated by unpaired bases; and (5) palindromes can be recursively nested, that is, a loop or a bulge may contain further palindromes.

Example 12 (RNA "palindromes" – base pairs only).

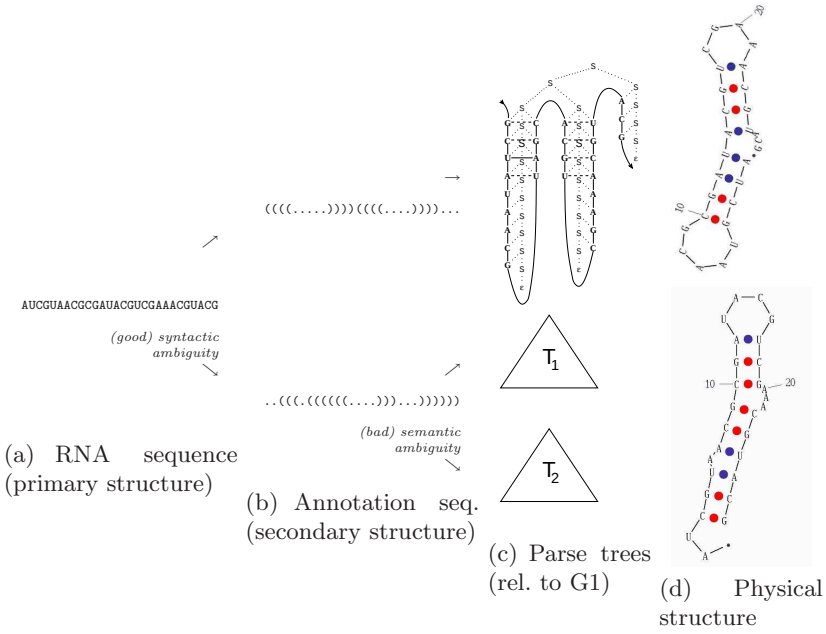
```
R   :   'C' R 'G'   |   'G' R 'C'
    |   'A' R 'U'   |   'U' R 'A'
    |   'G' R 'U'   |   'U' R 'G'    |   ε    ;
```

Context-free grammars are used to describe the structures that can be formed by a given RNA sequence. (The grammars G1 through G8, which we describe later, are different ways to achieve this.) All grammars generate the full language $\Sigma^*_{\texttt{RNA}}$, the different derivations of a given RNA string corresponding to its possible physical structures in different ways.

Figure 1 shows an RNA sequence and two of its possible structures, presented in the graphical form commonly used in biology, and as a so-called Vienna (or "dot-bracket") string, where base pairs are represented as matching parentheses and unpaired bases as dots.

The number of possible structures under the rules of base pairing is exponentially related to the length of the molecule. In formal language terms, each string has an exponential number of parse trees. This has been termed the "good" ambiguity in a grammar describing RNA structure. The set of all structures is the search space from which we want to extract the "true" structure. This is achieved by evaluating structures under a variety of scoring schemes. A CYK-style parser [1] constructs the search space and applies dynamic programming along the way.

The problem at hand arises when different parse trees correspond to the same physical structure. In Figure 1, parse trees T_1 and T_2 denote different parse trees for the same physical structure, shown to their right. In this case, the scoring schemes are misled. The number of structures is wrongly counted, and the most likely parse does not find the most likely structure. We say that the algorithm



(a) RNA sequence (primary structure)

(b) Annotation seq. (secondary structure)

(c) Parse trees (rel. to G1)

(d) Physical structure

Fig. 1. Good and bad ambiguity in RNA folding

exhibits the "bad" kind of ambiguity. It makes no sense to check the grammar for ambiguity *as is*, since (using a phrase from [18]) the bad ambiguity hides within the good.

Fortunately, the grammar can be transformed such that the good ambiguity is eliminated, while the bad persists and can now be checked by formal language techniques such as ours. The grammar remains structurally unchanged in the transformation, but is rewritten to no longer generate RNA sequences, but Vienna strings. They represent structures uniquely, and if one of them has two different parse trees, then the original grammar has the bad type of ambiguity.

We applied our ambiguity checker to several grammars that were obtained by the above transformation from stochastic grammars used in the bioinformatics literature [5, 18, 22]. Grammars G1 and G2 were studied as ambiguous grammars in [5], and our algorithm nicely points out the sources of ambiguity by indicating shortest ambiguous words. In [5], G2 was introduced as a refinement of G1, to bring it closer to grammars used in practice. Our ambiguity checker detects an extra vertical ambiguity in G2 (see Table 1) and clearly reports it by producing the ambiguous word "`()`" for the productions `P[aPa]` and `P[S]`. Grammars G3 through G8 are unambiguous grammars, taken from the same source. Our approach demonstrates their unambiguity.

Grammars used for thermodynamic RNA folding are rather large in order to accommodate the elaborate free energy model where the energy contribution of a single base or base pair strongly depends on its context. Grammars with bad ambiguity can still be used to find the minimum free energy structure, but not

Grammar	Bytes	(n, v, h)	LR(k)	Our
Palindromes (Ex. 10)	125	(1,5,3)	**non-LR(k)**	*unamb.*
AntiPalindromes (Ex. 10)	125	(1,5,3)	**non-LR(k)**	*unamb.*
Base pairs (Ex. 12)	144	(1,7,3)	**non-LR(k)**	*unamb.*
G1 [5]	91	(1,5,3)	24/12, 70/36, 195/99, ...	5V + 1H
G2 [5]	126	(2,5,3)	25/13, 59/37, 165/98, ...	6V + 1H
G3 [5]	154	(3,4,3)	**non-LR(k)**	*unamb.*
G4 [5]	115	(2,3,4)	LALR(1)	*unamb.*
G5 [5]	59	(1,3,4)	LALR(1)	*unamb.*
G6 [5]	116	(3,2,3)	LALR(1)	*unamb.*
G7 [5]	261	(5,4,3)	**non-LR(k)**	*unamb.*
G8 [5]	227	(4,3,4)	LALR(1)	*unamb.*
Voss-Light (Ex. 13)	243	(6,4,3)	**non-LR(k)**	*unamb.*
Voss [22]	2,601	(28,9,7)	**non-LR(k)**	*unamb.*

Table 1. Benchmark results

for the enumeration of near-optimal structures, and not for Boltzmann statistics scoring.

The grammar *Voss* from [22] has 28 nonterminals and 65 productions. This grammar clearly asks for automatic support (even for experts in formal grammars). We demonstrate this application in two steps. First, we study a grammar, *Voss-Light*, which demonstrates an essential aspect of the *Voss* grammar: unpaired bases in bulges and loops (the dots in the transformed grammar) must be treated differently, and they hence are derived from different nonterminal symbols even though they recognize the same language. This takes the grammar *Voss-Light* (and consequently also *Voss*) beyond the capacities of, for example, LR(k) parsing, whereas our technique succeeds in verifying unambiguity.

Example 13 (Voss-Light).

```
P  :  '(' P ')'  |  '(' O ')'  ;          // P: closed structure
O  :  L P  |  P R  |  S P S  |  H  ;       // O: open structure
L  :  '.' L  |  '.'  ;                     // L: left bulge
R  :  '.' R  |  '.'  ;                     // R: right bulge
S  :  '.' S  |  '.'  ;                     // S: singlestrand
H  :  '.' H  |  '.' '.' '.'  ;             // H: hairpin 3+ loop
```

As the second step, we took the full grammar, which required four simple unfolding transformations (see Appendix A) due to spurious ambiguities related to multiloops. Our method succeeded to show unambiguity, which implies that the Boltzmann statistics computed according to [22] are indeed correct.

Summary of Biosequence Analysis Experiments

Table 1 summarizes the results of running our ambiguity analysis and that of LR(k) on the example grammars from biosequence analysis presented in this

paper. The first column lists the name of the grammar along with a source reference. The second column quantifies the size of a grammar (in bytes). The third column elaborates this size measure where n is the total number of nonterminals, v is the maximum number of productions for a nonterminal, and h is the maximum number of entities on the right-hand-side of a production. The fourth column shows the results of running LR(k) and LALR(1) analysis: if the grammar is ambiguous, we list the number of shift/reduce and reduce/reduce conflicts as reported by LR(k) for increasing k, starting at $k = 1$. The last column shows the verdict from our analysis, reporting no false positives.

All example grammars, except *Voss*, take less than a second to analyze (including two levels of unfolding, as explained in Appendix A, in the case of G7 and G8). The larger *Voss* grammar takes about a minute on a standard PC. Note that in 7 cases, our technique verifies unambiguity where LR(k) fails.

With the recent *Locomotif* system [17], users draw graphical representation of physical structures (cf. Figure 1(d)), from which in a first step CFGs augmented with scoring functions are generated, which are subsequently compiled into dynamic programming algorithms coded in C. With this system, biologists may generate specialized RNA folding algorithms for many RNA families. Today more than 500 are known, with a different grammar implied by each – and all have to be checked for unambiguity.

6 Conclusion

We have presented 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 unfolding, without sacrificing soundness. Moreover, the analysis is often able to pinpoint sources of ambiguity through concrete examples being automatically generated. The analysis may be used when LR(k) and related techniques are inadequate, for example in biosequence analysis, as our examples show. Our experiments indicate that the precision, the speed, and the quality of warning messages are sufficient to be practically useful.

References

1. Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation and Compiling, Volume 1: Parsing*. Prentice Hall, 1972.
2. Claus Brabrand, Anders Møller, and Michael I. Schwartzbach. Dual syntax for XML languages. In *Proc. 10th International Workshop on Database Programming Languages, DBPL '05*, volume 3774 of *LNCS*, pages 27–41. Springer-Verlag, August 2005.
3. Claus Brabrand, Michael I. Schwartzbach, and Mads Vanggaard. The metafront system: Extensible parsing and transformation. In *Proc. 3rd ACM SIGPLAN Workshop on Language Descriptions, Tools and Applications, LDTA '03*, April 2003.

4. Anne Brüggemann-Klein and Derick Wood. Balanced context-free grammars, hedge grammars and pushdown caterpillar automata. In *Proc. Extreme Markup Languages*, 2004.
5. Robin D. Dowell and Sean R. Eddy. Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. *BMC Bioinformatics*, 5(71), 2004.
6. Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 1998.
7. Robert Giegerich. Explaining and controlling ambiguity in dynamic programming. In *Proc. 11th Annual Symposium on Combinatorial Pattern Matching*, volume 1848 of *LNCS*, pages 46–59. Springer-Verlag, 2000.
8. Robert Giegerich, Carsten Meyer, and Peter Steffen. A discipline of dynamic programming over sequence data. *Science of Computer Programming*, 51(3):215–263, 2004.
9. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, April 1979.
10. Karel Culik II and Rina S. Cohen. LR-regular grammars - an extension of LR(k) grammars. *Journal of Computer and System Sciences*, 7(1):66–96, 1973.
11. Donald E. Knuth. On the translation of languages from left to right. *Information and Control*, 8:607–639, 1965.
12. Donald E. Knuth. A characterization of parenthesis languages. *Information and Control*, 11:269–289, 1967.
13. Scott McPeak and George C. Necula. Elkhound: A fast, practical GLR parser generator. In *Proc. 13th International Conference on Compiler Construction, CC '04*, 2004.
14. Mehryar Mohri and Mark-Jan Nederhof. *Robustness in Language and Speech Technology*, chapter 9: Regular Approximation of Context-Free Grammars through Transformation. Kluwer Academic Publishers, 2001.
15. Anders Møller. dk.brics.automaton – finite-state automata and regular expressions for Java, 2001-2006. `http://www.brics.dk/automaton/`.
16. Mark-Jan Nederhof. Practical experiments with regular approximation of context-free languages. *Computational Linguistics*, 26(1):17–44, 2000.
17. Janina Reeder and Robert Giegerich. A graphical programming system for molecular motif search. In *Proc. 5th International Conference on Generative Programming and Component Engineering, GPCE '06*, pages 131–140. ACM, October 2006.
18. Janina Reeder, Peter Steffen, and Robert Giegerich. Effective ambiguity checking in biosequence analysis. *BMC Bioinformatics*, 6(153), 2005.
19. Elizabeth Scott, Adrian Johnstone, and Shamsa Sadaf Hussein. Tomita style generalised parsers. Technical Report CSD-TR-00-A, Royal Holloway, University of London, 2000.
20. Mark van den Brand, Jeroen Scheerder, Jurgen J. Vinju, and Eelco Visser. Disambiguation filters for scannerless generalized LR parsers. In *Proc. 11th International Conference on Compiler Construction, CC '02*, 2002.
21. Eelco Visser. *Syntax Definition for Language Prototyping*. PhD thesis, University of Amsterdam, September 1997.
22. Bjorn Voss, Robert Giegerich, and Marc Rehmsmeier. Complete probabilistic analysis of RNA shapes. *BMC Biology*, 4(5), 2006.

A Improving Precision with Grammar Unfolding

We here demonstrate how precision can be improved by a simple grammar unfolding technique. A related approach has been used by Nederhof [16] but not in the context of analyzing ambiguity.

Example 14 (unambiguous expressions). Although the grammar in Example 11 is ambiguous, its language is not inherently ambiguous. By introducing parentheses and fixing precedence and associativity of the operators, '+' and '*', an unambiguous grammar can easily be constructed:

```
Exp[plus]       :   Exp '+' Term
   [term]       |   Term              ;
Term[mult]      :   Term '*' Factor
    [factor]    |   Factor            ;
Factor[var]     :   'x'
      [par]     |   '(' Exp ')'       ;
```

While even the LR(k) technique is perfectly capable of acquitting this grammar as unambiguous, our analysis, as presented so far, reports the following spurious errors:

```
*** potential vertical ambiguity: Exp[plus] <--> Exp[term]
*** potential horizontal ambiguity at Exp[plus]: Exp <--> '+' Term
*** potential horizontal ambiguity at Exp[plus]: Exp '+' <--> Term
*** potential horizontal ambiguity at Term[mult]: Term <--> '*' Factor
*** potential horizontal ambiguity at Term[mult]: Term '*' <--> Factor
```

Note that, in contrast to the output being generated in Example 11, there are no example strings and the word "`potential`" has been added. For example, $MN_G(\texttt{Exp '+' Term}) \cap MN_G(\texttt{Term})$ is nonempty, but any example string in this intersection turns out to be unambiguous (our implementation tests just a single string in the intersection). This means that the analysis, as presented in the previous sections, cannot say with certainty that this grammar is ambiguous, nor that it is unambiguous.

By investigating the potential vertical ambiguity reported above, we see that a string $x = \alpha\texttt{+}\omega \in \mathcal{L}_G(\texttt{Exp})$ must have free parentheses in α if and only if x matches `Exp[term]` and not `Exp[plus]`. Hence, if we can distinguish between operators that appear within parentheses from ones that appear outside, we can eliminate this kind of spurious error.

Fortunately, the ambiguity characterization provided by Proposition 3 allows us to employ language-preserving transformations on the grammar without risking violations of the requirements set up in Section 1. A simple language preserving grammar transformation is *unfolding* recursive nonterminals, and, as explained in the following, this happens to be enough to eliminate all five spurious errors in the example above.

More generally, for *balanced grammars* [4] where parentheses are balanced in each production we can regain some precision that is lost by the Mohri-Nederhof approximation by *unfolding* the grammar to distinguish between inside and outside parentheses.

Definition 15 (Balanced grammar). *A grammar* $G = (\mathcal{N}, T, s, \pi)$ *is* balanced *if (1) the terminal alphabet has a decomposition* $T = \Sigma \cup \Gamma \cup \widehat{\Gamma}$ *where* Γ *is a set of left parentheses and* $\widehat{\Gamma}$ *a complementary set of right parentheses, and (2) all productions in are of the form* α *or* $\alpha\gamma\phi\widehat{\gamma}\omega$*, where* $\alpha, \phi, \omega \in (\Sigma \cup \mathcal{N})^*, \gamma \in \Gamma, \widehat{\gamma} \in \widehat{\Gamma}$ *(that is,* γ *is the complementary parenthesis of* $\widehat{\gamma}$*).*

Definition 16 (Unfolded balanced grammar). *Unfolding a balanced grammar* $G = (\mathcal{N}, \Sigma \cup \Gamma \cup \widehat{\Gamma}, s, \pi)$ *produces the grammar* $G_{\mathsf{u}} = (\mathcal{N} \cup \underline{\mathcal{N}}, \Sigma \cup \Gamma \cup \widehat{\Gamma} \cup \underline{\Sigma} \cup \underline{\Gamma} \cup \underline{\widehat{\Gamma}}, s, \pi_{\mathsf{u}})$ *where* $\underline{\mathcal{N}}$*,* $\underline{\Sigma}$*,* $\underline{\Gamma}$*, and* $\underline{\widehat{\Gamma}}$ *are copies of* $\mathcal{N}$*,* Σ*,* Γ*, and* $\widehat{\Gamma}$*, respectively, with all symbols underlined, and*

$$\pi_{\mathsf{u}}(n) = \begin{cases} \alpha & \text{if } \pi(n) = \alpha \\ \alpha\gamma\underline{\phi}\widehat{\gamma}\omega & \text{if } \pi(n) = \alpha\gamma\phi\widehat{\gamma}\omega \text{ where } \gamma \in \Gamma \text{ and } \widehat{\gamma} \in \widehat{\Gamma} \text{ match} \end{cases}$$

$$\pi_{\mathsf{u}}(\underline{n}) = \begin{cases} \underline{\alpha} & \text{if } \pi(n) = \alpha \\ \underline{\alpha\gamma\phi\widehat{\gamma}\omega} & \text{if } \pi(n) = \alpha\gamma\phi\widehat{\gamma}\omega \text{ where } \gamma \in \Gamma \text{ and } \widehat{\gamma} \in \widehat{\Gamma} \text{ match} \end{cases}$$

where $\underline{\theta}$ *is* θ *with all symbols underlined.*

Clearly, $\mathcal{L}(G)$ and $\mathcal{L}(G_{\mathsf{u}})$ are identical, except that in all strings in $\mathcal{L}(G_{\mathsf{u}})$ a symbol is underlined if and only if it is enclosed by parentheses. Thus, when checking vertical unambiguity of two productions in G it is sound to check the corresponding productions in G_{u} instead, and similarly for horizontal unambiguity. As the following example shows, this may improve precision of the analysis.

Example 17 (unambiguous expressions, unfolded). The grammar from Example 14 is balanced with $\Gamma = \{\texttt{(}\}$ and $\widehat{\Gamma} = \{\texttt{)}\}$. Unfolding yields this grammar:

```
Exp    : Exp '+' Term          Exp    : Exp '+' Term
       | Term          ;              | Term          ;
Term   : Term '*' Factor       Term   : Term '*' Factor
       | Factor        ;              | Factor        ;
Factor : 'x'                   Factor : 'x'
       | '(' Exp ')'   ;              | '(' Exp ')'   ;
```

A string parsed in the original grammar is parsed in exactly the same way in the new unfolded grammar, except that everything enclosed by parentheses is now underlined. For example, the string `x+(x+(x)+x)+x` from the original grammar corresponds to `x+(x+(x)+x)+x` with the resulting grammar. Now, every string in MN_G(`Exp '+' Term`) contains the symbol '`+`' whereas no strings in MN_G(`Term`) have this property (all '`+`' symbols are here underlined), so the potential vertical ambiguity warning is eliminated, and similarly for the other warnings.

The grammars G7, G8, and Voss-Light introduced in Section 5 need two unfoldings in order to be rightfully acquitted as unambiguous, and Voss needs four. However, these grammars all share the property of having many different nonterminals producing the same pairs of balanced parentheses, which is precisely why the regular approximation needs a couple of unfoldings to be able to discern these cases. We have not encountered grammars that need more than four levels of unfolding.

Example 18 (A non-balanced grammar). Grammars that exhibit parenthesis-like structures but are not balanced with any suitable choice of Γ and $\widehat{\Gamma}$ do not appear to be common in practice. An artificial example is the following:

```
S : A A ;
A : 'x' A 'x' | y ;
```

Our present technique is not able to detect that there is no horizontal ambiguity in the first production. The grammar is LR(1), however, so this example along with Example 10 establish the incomparability of our approach and LR(k). Still, our ambiguity characterization allows further language-preserving grammar transformations to be applied beyond the unfolding technique described above; we leave it to future work to discover other practically useful ones. In this tiny example, a simple solution would be to apply a transformation that underlines all symbols enclosed by `x`'s. In other cases one can use Knuth's algorithm for transforming a non-balanced grammar into an equivalent balanced one [12]. An alternative approach would be to combine our technique with, for example, LR(k). Of course, one can just run both techniques and see if one of them returns "`unambiguous grammar!`", but there may be a way to combine them more locally (that is, for individual checks of vertical/horizontal unambiguity) to gain precision, which is also left to future work.

B Proof of Proposition 3

We use the notation $\Vdash G$ when a grammar G is vertically unambiguous according to Definition 2, $\models G$ when it is horizontally unambiguous, and $\Vdash\!\models G$ means that it is both vertically and horizontally unambiguous.

To show

$$\Vdash\!\models G \quad \Leftrightarrow \quad G \textit{ is unambiguous}$$

we consider each direction in turn.

We prove $\Vdash\!\models G \Rightarrow G$ *is unambiguous* by contrapositively establishing that if G is ambiguous, then $\not\Vdash G \vee \not\models G$ (that is, $\not\Vdash\!\models G$). Assume that G is ambiguous; that is, that there are two derivation trees for some string, ω. We now proceed by induction in the maximum height h of the two derivation trees for ω (where the height of a derivation tree is the maximum number of edges from the root to a leaf).

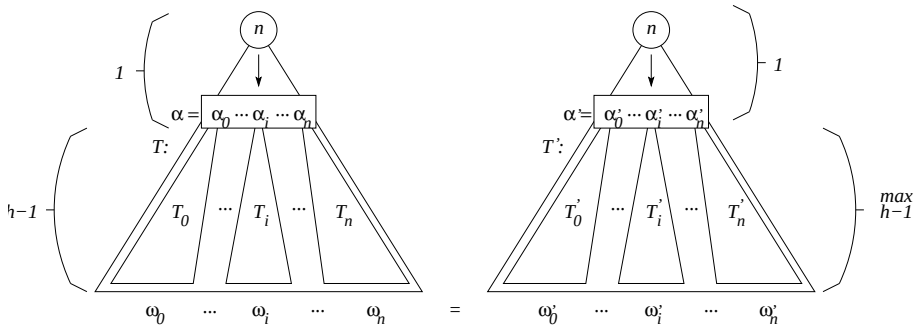


Fig. 2. Two derivation trees, T and T', both deriving ω from n.

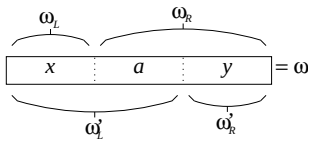


Fig. 3. Overlap: $\mathcal{L}_G(\alpha_0..\alpha_k \between \mathcal{L}_G(\alpha_{k+1}..\alpha_n)) \supseteq \{\omega\} \neq \emptyset$.

Base case (h=1): The result holds vacuously since there cannot be two different derivation trees of height one; a derivation tree of height one necessarily has terminals as leaves meaning that two such trees would be the same.

Inductive case: We assume the property holds for all derivation trees of maximum height $h-1$ and show that it also holds for height h. Assume that G ambiguously has two derivation trees, T and T', the heigher of which has height h, as depicted in Figure 2 (assuming T is heigher). There are now two cases depending on whether or not the top-most production in the tree is the same:

[*Case* $\alpha \neq \alpha'$]: If the trees differ due to the different productions α and α' ultimately producing the same string ω we have that: $\mathcal{L}_G(\alpha) \cap \mathcal{L}_G(\alpha') \supseteq \{\omega\} \neq \emptyset$, and hence $\not\parallel G$ as required.

[*Case* $\alpha = \alpha'$]: This case again splits into two cases depending on whether the difference in the two trees is at the top-level or further down the trees:

[*Case* $\exists i : \omega_i \neq \omega'_i$]: In this case, we let $k = min\{\ i \mid \omega_i \neq \omega'_i\ \}$ and, depending on this k, let $\omega_L = \omega_0 \cdots \omega_k$, $\omega'_L = \omega'_0 \cdots \omega'_k$, $\omega_R = \omega_{k+1} \cdots \omega_n$, and $\omega'_R = \omega'_{k+1} \cdots \omega'_{n'}$. Since k was chosen as the minimum we must have that $\omega_L \neq \omega'_L$ and since $\omega_L \omega_R = \omega = \omega'_L \omega'_R$, the strings must be organized according to Figure 3 (assuming without loss of generality that $|\omega_L| < |\omega'_L|$ which means that ω_L is a prefix of ω'_L) in that $\omega = \omega_L a \omega'_R$ for some $a \in \Sigma^+$. We thus have a language overlap $X \between Y \supseteq \{\omega_L a \omega'_R\} \neq \emptyset$ for the two languages $X = \mathcal{L}_G(\alpha_0 \cdots \alpha_k)$ and $Y = \mathcal{L}_G(\alpha_{k+1} \cdots \alpha_n)$, and hence $\not\parallel G$ as required.

[*Case* $\forall i : \omega_i = \omega'_i$]: In this case, the difference between the two trees T and T' must be further down. Pick any i such that the subtrees T_i and T'_i (where T_i is the subtree corresponding to α_i) are different (such an i must exist since T and T' were different in the first place). The induction hypothesis applied to these smaller trees, ambiguously deriving ω_i from α_i, gives us $\nVdash G$ as required.

We now consider the other direction, $\Vdash G \Leftarrow G$ *is unambiguous*. This is shown by contrapositively establishing that if $\nVdash G \vee \nvDash G$, then G is ambiguous. We split into two cases; one for horizontal ambiguity and one for vertical ambiguity.

[*Case* $\nVdash G$]: Since $\nVdash G$, we have that $\mathcal{L}_G(\alpha) \cap \mathcal{L}_G(\alpha') \neq \emptyset$ for some n where $\alpha, \alpha \in \pi(n)$ and $\alpha \neq \alpha'$. Let ω be an element of this intersection; meaning that we have $\alpha \Rightarrow^* \omega$ and $\alpha' \Rightarrow^* \omega$. Since n is reachable in G, we can construct two different derivation trees starting at s, corresponding to the following derivations:

$$s \Rightarrow^* \gamma_L\ n\ \gamma_R \Rightarrow \gamma_L\ \alpha\ \gamma_R \Rightarrow^* \gamma_L\ \omega\ \gamma_R \Rightarrow^* \omega_L\ \omega\ \omega_R$$
$$s \Rightarrow^* \gamma_L\ n\ \gamma_R \Rightarrow \gamma_L\ \alpha'\ \gamma_R \Rightarrow^* \gamma_L\ \omega\ \gamma_R \Rightarrow^* \omega_L\ \omega\ \omega_R$$

These derivations are possible since we assumed that all nonterminals are derivable and all nonterminals derive something.

[*Case* $\nvDash G$]: Since $\nvDash G$, we have that $x, xa \in \mathcal{L}_G(\alpha_L)$ and $y, ay \in \mathcal{L}_G(\alpha_R)$ for some production $\pi(n) = \alpha = \alpha_L \alpha_R$. Similar to the case above, we make two different derivation trees corresponding to the following two derivations:

$$s \Rightarrow^* \gamma_L\ n\ \gamma_R \Rightarrow^* \gamma_L\ \alpha_L\ \alpha_R\ \gamma_R \Rightarrow^* \gamma_L\ x\ \alpha_R\ \gamma_R \Rightarrow^* \gamma_L\ x\ ay\ \gamma_R \Rightarrow^* \omega_L\ x\ ay\ \omega_R$$
$$s \Rightarrow^* \gamma_L\ n\ \gamma_R \Rightarrow^* \gamma_L\ \alpha_L\ \alpha_R\ \gamma_R \Rightarrow^* \gamma_L\ xa\ \alpha_R\ \gamma_R \Rightarrow^* \gamma_L\ xa\ y\ \gamma_R \Rightarrow^* \omega_L\ xa\ y\ \omega_R$$

C Proof of Proposition 8

We use the notation $\Vdash_{\mathcal{A}_G} G$ when a grammar G is both vertically and horizontally unambiguous relative to a grammar over-approximation $\mathcal{A}_G$ according to Definition 7.

It is enough to show that $\Vdash_{\mathcal{A}_G} G \Rightarrow \Vdash G$ since the result then follows from Proposition 3. However, this is immediate from Definitions 2, 6, and 7: since $\mathcal{A}_G$ is a conservative approximation, we have that for all $\alpha, \beta \in E^*$:

$$\mathcal{A}_G(\alpha) \cap \mathcal{A}_G(\beta) = \emptyset \quad \Rightarrow \quad \mathcal{L}_G(\alpha) \cap \mathcal{L}_G(\beta) = \emptyset$$
$$\mathcal{A}_G(\alpha) \between \mathcal{A}_G(\beta) = \emptyset \quad \Rightarrow \quad \mathcal{L}_G(\alpha) \between \mathcal{L}_G(\beta) = \emptyset$$