Analyzing Ambiguity of Context-Free Grammars

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

[1] Department of Computer Science, 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.

Keywords: CFG ambiguity, regular approximation, RNA analysis.

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 [1] 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 [2]. An example of an even larger class of unambiguous grammars is LR-Regular [3]. However, not even LR-Regular is sufficient for a considerable class of grammars involving palindromic structures, which our technique can handle. Additionally, unlike LR(k), our approach works well without separating lexical descriptions from the grammars.
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 [4]. (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

Jan Holub and Jan Žd'árek (Eds.): CIAA 2007, LNCS 4783, pp. 214–225, 2007.

may be parsed in multiple ways and precisely identify the relevant location in the grammar, thereby pinpointing the cause of the ambiguity.

An increasing number of parser generators, for example, Bison [4], SDF [5], and Elkhound [6], 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 [7,8]. In contrast, our approach is to *statically* analyze the grammars for potential ambiguities. Also, we aim for a *conservative* algorithm, unlike many existing ambiguity detection techniques (e.g. [9,10]). The recent approach by Schmitz is conservative; for a comparison with our approach see the paper [11]. Another conservative approach, expressed in terms of pushdown acceptors, is described in an article by Kuich [12].

In bioinformatics, context-free grammars in various guises have important applications, for example in sequence comparison, motif search, and RNA secondary structure analysis [13,14]. 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 [15,16]. 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 state two requirements on a practical ambiguity checker that result from the biosequence analysis 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 [17]. This involves scannerless parsing, that is, lexical descriptions are not separated from the grammars, so LR(k) is not applicable.

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 areas mentioned above. 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 [18] 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 technical report [19] 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 $X = \{\mathtt{x}, \mathtt{xa}\}$ and $Y = \{\mathtt{a}, \mathtt{ay}\}$ then $X \between Y = \{\mathtt{xay}\}$. 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 $\Leftrightarrow$ *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 [19]. □

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* [1]. (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 [19]. □

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 undecidable (since it involves checking language disjointness for context-free grammars).

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 a different 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 [20]). 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 [18].

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 {a, 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 a(a+b)*a, and MN_G(`'b' P 'b'`) is b(a+b)*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 phosphate. 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.

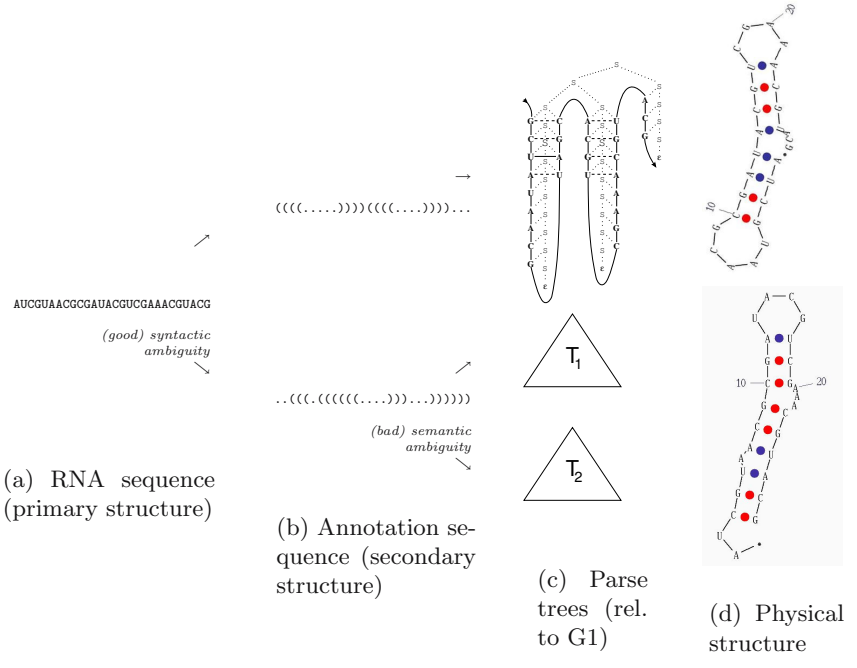

Fig. 1. Good and bad ambiguity in RNA folding

A CYK-style parser [21] 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 exhibits the "bad" kind of ambiguity. It makes no sense to check the grammar for ambiguity *as is*, since (using a phrase from [22]) 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 [16,22,23]. Grammars G1 and G2 were studied as ambiguous grammars in [16], and our algorithm nicely points out the sources of ambiguity by indicating shortest ambiguous words. In [16], 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 for the enumeration of near-optimal structures, and not for Boltzmann statistics scoring.

The grammar *Voss* from [23] 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 [19]) due to spurious ambiguities related to multiloops. Our method succeeded to show unambiguity, which implies that the Boltzmann statistics computed according to [23] are indeed correct.

5.1 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 automatic LR(k) and LALR(1) analyses: 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 $k = 1$. We have manually inspected that the ones marked as non-LR(k) are in fact non-LR-Regular. 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 [19], in the case of G7 and G8).

Table 1. Benchmark results

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 [16]	91	(1,5,3)	24/12, 70/36, 195/99, ...	5V + 1H
G2 [16]	126	(2,5,3)	25/13, 59/37, 165/98, ...	6V + 1H
G3 [16]	154	(3,4,3)	**non-LR(k)**	*unamb.*
G4 [16]	115	(2,3,4)	LALR(1)	*unamb.*
G5 [16]	59	(1,3,4)	LALR(1)	*unamb.*
G6 [16]	116	(3,2,3)	LALR(1)	*unamb.*
G7 [16]	261	(5,4,3)	**non-LR(k)**	*unamb.*
G8 [16]	227	(4,3,4)	LALR(1)	*unamb.*
Voss-Light (Ex. 13)	243	(6,4,3)	**non-LR(k)**	*unamb.*
Voss [23]	2,601	(28,9,7)	**non-LR(k)**	*unamb.*

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 [24], 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. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (1979)
2. Knuth, D.E.: On the translation of languages from left to right. Information and Control 8, 607–639 (1965)

3. Culik II, K., Cohen, R.S.: LR-regular grammars - an extension of LR(k) grammars. Journal of Computer and System Sciences 7(1), 66–96 (1973)
4. Scott, E., Johnstone, A., Hussein, S.S.: Tomita style generalised parsers. Technical Report CSD-TR-00-A, Royal Holloway, University of London (2000)
5. Visser, E.: Syntax Definition for Language Prototyping. PhD thesis, University of Amsterdam (1997)
6. McPeak, S., Necula, G.C.: Elkhound: A fast, practical GLR parser generator. In: Duesterwald, E. (ed.) CC 2004. LNCS, vol. 2985, Springer, Heidelberg (2004)
7. van den Brand, M., Scheerder, J., Vinju, J.J., Visser, E.: Disambiguation filters for scannerless generalized LR parsers. In: Horspool, R.N. (ed.) CC 2002 and ETAPS 2002. LNCS, vol. 2304, Springer, Heidelberg (2002)
8. Brabrand, C., Schwartzbach, M.I., Vanggaard, M.: The metafront system: Extensible parsing and transformation. In: LDTA 2003. Proc. 3rd ACM SIGPLAN Workshop on Language Descriptions, Tools and Applications, ACM Press, New York (2003)
9. Gorn, S.: Detection of generative ambiguities in context-free mechanical languages. Journal of the ACM 10(2), 196–208 (1963)
10. Cheung, B.S.N., Uzgalis, R.C.: Ambiguity in context-free grammars. In: SAC 1995. Proc. ACM Symposium on Applied Computing, ACM Press, New York (1995)
11. Schmitz, S.: Conservative ambiguity detection in context-free grammars. In: ICALP 2007. Proc. 34th International Colloquium on Automata, Languages and Programming (2007)
12. Kuich, W.: Systems of pushdown acceptors and context-free grammars. Elektronische Informationsverarbeitung und Kybernetik 6(2), 95–114 (1970)
13. Durbin, R., Eddy, S.R., Krogh, A., Mitchison, G.: Biological Sequence Analysis. Cambridge University Press, Cambridge (1998)
14. Giegerich, R., Meyer, C., Steffen, P.: A discipline of dynamic programming over sequence data. Science of Computer Programming 51(3), 215–263 (2004)
15. Giegerich, R.: Explaining and controlling ambiguity in dynamic programming. In: Giancarlo, R., Sankoff, D. (eds.) CPM 2000. LNCS, vol. 1848, pp. 46–59. Springer, Heidelberg (2000)
16. Dowell, R.D., Eddy, S.R.: Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. BMC Bioinformatics 5(71) (2004)
17. Brabrand, C., Møller, A., Schwartzbach, M.I.: Dual syntax for XML languages. In: Bierman, G., Koch, C. (eds.) DBPL 2005. LNCS, vol. 3774, Springer, Heidelberg (2005)
18. Mohri, M., Nederhof, M.J.: 9: Regular Approximation of Context-Free Grammars through Transformation. In: Robustness in Language and Speech Technology, Kluwer Academic Publishers, Dordrecht (2001)
19. Brabrand, C., Giegerich, R., Møller, A.: Analyzing ambiguity of context-free grammars. Technical Report RS-07-10, BRICS (2007)
20. Møller, A.: dk.brics.automaton – finite-state automata and regular expressions for Java (2007), `http://www.brics.dk/automaton/`
21. Aho, A.V., Ullman, J.D.: The Theory of Parsing, Translation and Compiling, vol. 1: Parsing. Prentice-Hall, Englewood Cliffs (1972)
22. Reeder, J., Steffen, P., Giegerich, R.: Effective ambiguity checking in biosequence analysis. BMC Bioinformatics 6(153) (2005)
23. Voss, B., Giegerich, R., Rehmsmeier, M.: Complete probabilistic analysis of RNA shapes. BMC Biology 4(5) (2006)
24. Reeder, J., Giegerich, R.: A graphical programming system for molecular motif search. In: GPCE 2006. Proc. 5th International Conference on Generative Programming and Component Engineering, pp. 131–140. ACM Press, New York (2006)