Ambiguity Analysis of RNA Secondary Structure Prediction Grammars

Bachelor Thesis

Raphael Reitzig
Bachelor Informatik

Supervised by Prof. Dr. Markus E. Nebel,
AG Algorithmen und Komplexität,
Fachbereich Informatik,
Technische Universität Kaiserslautern

Summer Term 2009

Abstract

In [DE04], several context free grammars for RNA secondary structure prediction are presented. Some of them are claimed to be unambiguous with respect to secondary structures. Although this property is crucial for the correctness of prediction algorithms using those grammars, it is only shown empirically, but not formally proven. This gap is filled in the work at hand.

In [DE04] werden mehrere kontextfreie Grammatiken für die Vorhersage von RNA Sekundärstrukturen vorgestellt. Von einigen wird behauptet, dass sie eindeutig in Bezug auf Sekundärstrukturen seien. Obwohl diese Eigenschaft wichtig für die Korrektheit von Algorithmen ist, die diese Grammatiken benutzen, wird sie zwar empirisch belegt, aber nicht formal bewiesen. Diese Lücke wird in der vorliegenden Arbeit geschlossen.

Contents

1 Formal Tools

In order to handle the upcoming problems in a convenient way, some formal notations and tools are needed. In this section, very basic definitions and some properties of formal languages, (stochastic) context free grammars and grammar analysis with generating functions are presented. Notations are mostly based on those used in [Sch01] and [Wil90].

1.1 Formal Languages

A finite set $\Sigma \neq \{\}$ is called *alphabet*, any element $a \in \Sigma$ is a *symbol*. Individual symbols can be *concatenated* with the operation $\cdot$, creating finite chains of symbols. Such a chain is called *word* (or *string*). Since $\cdot$ is associative, it is often left out and it naturally extends to words, e.g. $ab \cdot ab = abab$. The *length* of a word α, that is the number of symbols in α, is denoted by $|\alpha|$. The word of length 0 is called the *empty word* ε. Words that consist of a repeated pattern can be written in short as

$$\alpha^n = \underbrace{\alpha \cdot \ldots \cdot \alpha}_{n \text{ times}}.$$

Also, to access the i-th symbol of a word α, the form α_i is used. For a whole *subword* (or *substring*) $\alpha' = \alpha_i \ldots \alpha_j$ of α, write $\alpha_{i,j}$. To denote the set of all words over Σ of a given length, use

$$\Sigma^n = \{\alpha : |\alpha| = n, \alpha_i \in \Sigma \text{ for } 1 \leq i \leq n\}.$$

The famous Kleene Star, defined by

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n,$$

deals the set of all words over Σ. Often also $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ is used. Note that $(\Sigma^*, \cdot)$ is a monoid with identity element ε.

Any set $\mathcal{L} \subseteq \Sigma^*$ is called a *formal language over* Σ. A handy notation for certain sublanguages is $\mathcal{L}_{\circ n} = \{\alpha \in \mathcal{L} : |\alpha| \circ n\}$, with $\circ \in \{=, \neq, <, \leq, >, \geq\}$. $\mathcal{L}_n$ is short for $\mathcal{L}_{=n}$.

Example 1. Let $\Sigma = \{a, b\}$. Then, $\Sigma^2 = \{aa, ab, ba, bb\}$, $(ab)^3 = ababab$, $|(ab)^3| = 6$ and $ababab_{2,4} = bab$. If furthermore $\mathcal{L} = \{a^n b^n : n \geq 0\} \subseteq \Sigma^*$, then $\mathcal{L}_{<5} = \{\varepsilon, ab, aabb\}$.

1.2 Context Free Grammars

To describe formal languages in a convenient and finite way, Chomsky introduced the means of *formal grammars*. Assume you want to describe the formal language $\mathcal{L} \subseteq \Sigma^*$. A formal grammar G uses additional symbols not in Σ and production rules that translate words over both symbol kinds to other such words. Starting with a distinguished starting symbol, successive application of

such rules generates words from $\mathcal{L}$. Formal grammars are classified alongside the classes of formal languages they can generate. As a general rule, the more powerful or expressive a grammar is, the harder it is to decide algorithmically wether a word is generated by it or not. The class of context free grammars is fairly easy to handle yet powerful enough for many applications. In formal terms, this class is defined like this:

Let N, T be two disjoint, finite sets of symbols, $\delta \subseteq N \times (N \cup T)^*$ a set of *production rules* and $S \in N$ the *start symbol*. Then $G = (N, T, \delta, S)$ is a *context free grammar* (CFG). The elements of N are called *non-terminals*, those of T *terminals*. Rules (A, α) can be written $A \to \alpha$. Several alternatives $A \to \alpha_1$, $A \to \alpha_2$ are often shortend to $A \to \alpha_1 \mid \alpha_2$.

$\alpha A \beta \Rightarrow_G \alpha\gamma\beta$ denotes a *generation step in* G if and only if $\alpha, \beta, \gamma \in (N \cup T)^*$ and $(A, \gamma) \in \delta$. An arbitrarily long sequence of generation steps $\alpha \Rightarrow_G \ldots \Rightarrow_G \beta$ can be written $\alpha \Rightarrow_G^* \beta$ in short. $\Rightarrow_G$ can be abbreviated $\Rightarrow$ if there can be no misunderstanding as to which grammar is meant.

$\Rightarrow_G$ is also often read as relation on $(N \cup T)^*$. $\alpha \Rightarrow_G \beta$ reads 'β can be *directly derived from* α'. Consequently, $\Rightarrow_G^*$ is the reflexive and transitive closure of $\Rightarrow_G$, reading '*can be derived from*'.

$\mathcal{L}(G) \subseteq T^*$ is called the *language generated by* G. $\alpha \in \mathcal{L}(G)$ if and only if $S \Rightarrow_G^* \alpha$. Every sequence $S \Rightarrow_G^* \alpha$ is called a *generation of* α *in* G. Any language created by a CFG is called a *context free language* (CFL).

Each generation can be written as a uniquely determined tree[1]. This tree is also called *syntax tree* or *parse tree*, depending on the respective author's background. Its front, read from left to right, is the generated word. If there is more than one syntax tree for at least one word in $\mathcal{L}(G)$, G is called *ambiguous*. The maximum number of syntax trees for any $\alpha \in \mathcal{L}(G)$ is called G's *degree of ambiguity*. If no such maximum exists, G is said to be ambiguous with infinite degree.

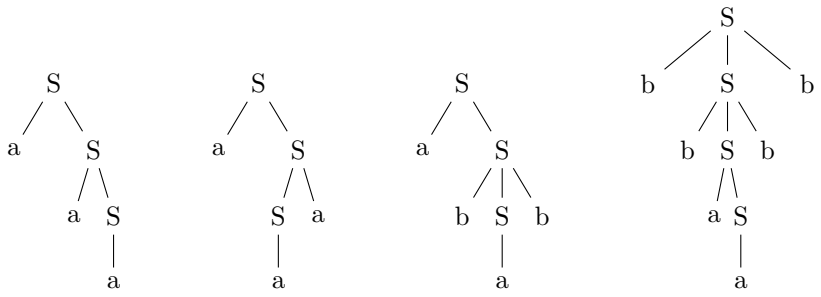
Example 2. Given the grammar $G = (\{S\}, \{a, b\}, \{S \to aS \mid Sa \mid bSb \mid a\}, S)$, for example the generations

$$S \Rightarrow_G aS \Rightarrow_G aaS \Rightarrow_G aaa,$$
$$S \Rightarrow_G aS \Rightarrow_G aSa \Rightarrow_G aaa,$$
$$S \Rightarrow_G aS \Rightarrow_G abSb \Rightarrow_G abab \text{ and}$$
$$S \Rightarrow_G bSb \Rightarrow_G bbSbb \Rightarrow_G bbaSbb \Rightarrow_G bbaabb$$

can be done in G. Consequently, $\{aaa, abab, bbaabb\} \subseteq \mathcal{L}(G)$. See Figure 1 for the corresponding syntax trees. Note that there are at least two syntax trees for aaa. Therefore, G is ambiguous.

The syntax tree can be given a special meaning. In the context of programming languages, it determines the program's semantics. Hence, there should be only one syntax tree for any given program. In the world of RNA molecules, however, syntax trees of a molecule's primary structure describe its possible secondary structures. How this can be used for structure prediction will be shown later on.

[1] Different generations may correspond to the same syntax tree, though.



(a) *aaa* (b) *aaa* (c) *abab* (d) *bbaabb*

Figure 1: Syntax trees corresponding to the generations in Example 2.

1.3 Stochastic Context Free Grammars

CFGs as described above either generate a word or do not. This property is sufficient for lots of tasks, for example the construction of programming language compilers. But sometimes a probability distribution on words is needed. Consider, for instance, the generation of test input data for an algorithm. That data should fit the inputs that occur in real use but should nonetheless be randomized. Well researched and often used tools are the Markov Chain and its predicting cousin, the Hidden Markov Model (HMM). These can describe probability distributions on *regular* languages which are, unfortunately, substantially less powerful than context free languages. Luckily, their concepts can be transported to CFLs. A proper model to describe and work with distributions over CFLs are *stochastic context free grammars* (SCFGs). Work on SCFGs is not as popular as that on HMMs and is a bit hard to come by. A short introduction with many further references is given in [HF71].

A SCFG $G = (N, T, \delta, p, S)$ is defined by the following properties. First, $G' = (N, T, \delta, S)$ is a CFG and $\mathcal{L}(G) = \mathcal{L}(G')$. Assume without loss of generality that for every $A \in N$, at least one rule $(A, \alpha) \in \delta$ exists. p is a function $p : \delta \to [0, 1]$ that assigns each rule a probability. For better readability, p is often given as annotation in δ, like $(A, \alpha, p(A \to \alpha)) \in \delta$ or $A \overset{p(A\to\alpha)}{\longrightarrow} \alpha$. Additionally, for each non-terminal $A \in N$,

$$\sum_{(A,\alpha)\in\delta} p(A \to \alpha) = 1$$

must hold.

Let $S \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n$ be a generation for $\alpha_n \in \mathcal{L}(G)$ and $R = \{r_1, \ldots, r_n\}$ the multiset of rules $r_i \in \delta$ used in that generation. The syntax tree T that is implied by the given generation has the probability in G

$$P(T \mid G) = \prod_{r\in R} p(r).$$

Note that different generations that lead to the same syntax tree T all use the rules in R if in different order. This is not considered a worthy contribution,

therefore R is only taken into account once for T. Now assume that a word $\alpha \in \mathcal{L}(G)$ has m pairwise distinct syntax trees $T_1, \ldots, T_m$ in G. Then, the probability of α in G is given by

$$P(\alpha \mid G) = \sum_{1 \leq i \leq m} P(T_i \mid G).$$

G is *consistent with* $\mathcal{L}(G)$, that is to say it describes a probability distribution on $\mathcal{L}(G)$, if and only if it fulfills the constraints stated above and furthermore

$$\sum_{\alpha \in \mathcal{L}(G)} P(\alpha \mid G) = 1$$

holds.

Example 3. Consider G from Example 2 and extend it to a SCFG with $p(S \to aS) = 0.4$, $p(S \to Sa) = 0.2$, $p(S \to bSb) = 0.3$ and $p(S \to a) = 0.1$. G obviously is well-defined. It is also consistent; a proof of this fact is given in the next section. The syntax trees given in Figure 1 have the probabilities

$$P((\text{a}) \mid G) = 0.4 \cdot 0.4 \cdot 0.1 = 0.016,$$
$$P((\text{b}) \mid G) = 0.4 \cdot 0.2 \cdot 0.1 = 0.008,$$
$$P((\text{c}) \mid G) = 0.4 \cdot 0.3 \cdot 0.1 = 0.012 \text{ and}$$
$$P((\text{d}) \mid G) = 0.3 \cdot 0.3 \cdot 0.4 \cdot 0.1 = 0.0036.$$

There are two more syntax trees for aaa, one with probability $0.2 \cdot 0.4 \cdot 0.1 = 0.008$ and one with probability $0.2 \cdot 0.2 \cdot 0.1 = 0.004$. So, the probability of aaa is $P(aaa \mid G) = 0.016 + 0.008 + 0.008 + 0.004 = 0.036$.

1.4 Grammar Analysis Using Generating Functions

A thorough introduction to generating functions and many of their uses is given in [Wil90]. Techniques using generating functions utilise that functions have a meaning in both continuous and discrete mathematics. While manipulation of functions in the world of analysis is often easily done, their series expansions, interpreted as formal power series, might solve a problem in the discrete universe. Adept switching between and intertwining of both points of view on the same function often results in surprisingly simple solutions to otherwise hard to solve problems. In this context, mainly one application is of interest. Suppose you can represent a counting problem as a recurrence formula in a suitable way and the function $A(x)$ is the solution of this recurrence. Then, A is called the problem's *generating function* and the coefficients $a_n = [x^n]A$ of A's series expansion (meaning, $A(x) = \sum_{n \geq 0} a_n x^n$) solve the counting problem. Thus, combinatorial problems can be reduced to solving a recurrence and finding the series expansion of its solution. Of course, both steps can be arbitrarily complicated. In many cases, however, at least the second step is easy because for many typically occurring types of functions corresponding series expansions are known.

In particular, ambiguity of context free grammars can be examined using generating functions. Unambiguousness is often a very important feature of

CFGs, for example in the context of programming languages. Unfortunately, the general problem is algorithmically undecidable and manual proofs are often hard to find. Using a CFG's *structure function*, which is a generating function, you can calculate the number of syntax trees of words with length n in many cases. If you know the number of such words in the generated language, you can conclude wether the grammar is ambiguous or not and, if it is, even give the average degree of ambiguity. The theoretical background has been developed in [CS63]. Clearer notation and proofs about some major aspects, in particular the derivation of the specific technique used in the following, can be found in later works, for instance in [Kui70].

The structure function is obtained by solving a functional equation system that is created directly from the set of productions. Assuming $G = (N, T, \delta, S)$, solve

$$\left[A(x) = \sum_{(A, a_0 \ldots a_k) \in \delta} \prod_{i=0}^{k} \tau(a_i) \ : A \in N \right] \text{ with } \tau(a) = \begin{cases} a(x) & , a \in N \\ x & , a \in T \end{cases}.$$

If the system has a solution, $S(x)$ is the structure function of G and $[x^n]S(x)$ is the number of syntax trees in G for words of length n. It might not be solvable, though. For example, if there is an infinite amount of syntax trees for but one word, there can not be a structure function with real or even natural coefficients. Other problems arise because solving functional equations can be very hard, especially when higher powers of the functional variables occur. In practice, often only numerically computed approximations can be determined.

Example 4. Consider the grammar $G = (\{B\}, \{a, b\}, \{B \to aB \mid bB \mid \varepsilon\}, B)$. Apparently, it generates the language $\{a, b\}^*$. To find its structure function, you have to solve the equation (system)

$$B(x) = xB(x) + xB(x) + 1.$$

The solution and structure function of G is quite obviously[2]

$$B(x) = \frac{1}{1 - 2x} = \sum_{n \geq 0} 2^n x^n$$

and therefore $[x^n]B(x) = 2^n$ is the number of syntax trees for words of length n. Since there are exactly 2^n such words, this grammar is unambiguous.

If you take $G' = (\{A, B\}, \{a, b\}, \{A \to BB, B \to aB \mid bB \mid \varepsilon\}, A)$, which also generates $\{a, b\}^*$, and solve the resulting equation system

$$A(x) = B(x)^2$$
$$B(x) = xB(x) + xB(x) + 1,$$

the structure function of G' is

$$A(x) = \frac{1}{(1 - 2x)^2} = \sum_{n \geq 0} (n + 1) 2^n x^n.$$

Thus it is proven fact that G' is ambiguous with degree ∞.

[2] Of course, this might be less obvious if it were not for [Wil90], p. 52 ff.

If you look at the steps made in [Kui70] that show the feasability of a grammar's structure function $S(x) = \sum_{n\geq 0} a_n x^n$, you can easily see that it can be extended to SCFGs quite naturally. If each addend in the grammar's equation system is multiplied with its corresponding rule's probability, each word is not counted with 1, but with its probability. Therefore, a_n is not the number of words of length n but the sum of probabilities of all those words, i.e. the probability that a word has length n. As a consequence, $S(1) = \sum_{n\geq 0} a_n$ is the sum of all word probabilites. This can be used to prove that the SCFG from Example 3 is consistent and to provide another example for the use of generating functions.

Example 5. The SCFG given in Example 3 translates to the equation

$$S(x) = 0.6 \cdot xS(x) + 0.3 \cdot x^2 S(x) + 0.1 \cdot x$$

which has the solution

$$S(x) = \frac{0.1 \cdot x}{1 - 0.6 \cdot x - 0.3 \cdot x^2}.$$

Since obviously $S(1) = 1$ holds, the grammar in question is consistent.

2 Biological Basics

2.1 RNA

The term *ribonucleic acid* (RNA) names a class of molecules that is essential for any organism's metabolism. RNA is, for the most part, a catalyst for complex reaction chains. For example, messenger RNA, transfer RNA and ribosomal RNA are part of protein synthesis, one of the most central processes in our cells. There is some special RNA in the mitochondria, the main suppliers of energy in cells, that varies far more between species than the well protected DNA in the cell core and can thus be used to identify organisms. There are some viruses whose genetic information is carried entirely by RNA. For some more functions of RNA, see [DEKM98]. Even so, biologists only start to discover the hole scope of uses RNA has.

An RNA molecule is a polymer, a chain, of nucleotides. There are four distinct nucleotides in RNA, distinguished by their nitrogenous bases: adenine (`A`), cytosine (`C`), guanine (`G`) and uracil (`U`). The sequence of nucleotides is also called *primary structure*. But the primary structure of a molecule seems to be less important in determining its function than its folding, called *secondary structure*, is. Different sequences that fold in the same way may serve the same purpose. On the other hand, the same sequence might fold to entirely different structures given different environmental conditions.

Biologists typically extract RNA from dead organisms and obtain the primary structure, meaning the linear sequence of nucleotides, by sequencing. To guess what effect a given RNA sequence might have, they need an idea of how it may look like folded in the living cell. Also, they might want to find similar RNA in a database. Since RNA is less protected against mutation than DNA

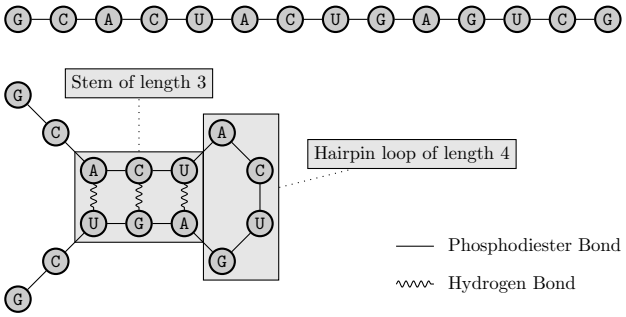


Figure 2: A sample RNA primary (top) and secondary structure (bottom).

for it is only single-stranded, RNA primary structure changes much through evolution. Therefore, results are often better when looking for a similar structure rather than for a similar sequence. In both (and probably many more) cases a wellfounded prediction of RNA secondary structures is needed. This can be done manually or by computers where obviously the latter method is of interest for computer scientists. There are databases of painstakingly found and confirmed corresponding pairs of RNA primary and secondary structures, for instance [CRW09]. One possible way to attack the prediction problem is to choose the *most probable* structure given the knowledge you have about similar species' RNA. In order to do that, you need a feasible probability model for RNA structures. This has been done using SCFGs, for example in [DE04].

The type of secondary structures considered here forms because of nucleotides pairing up out of order with hydrogen bonds. Those are much weaker than the phosphodiester bonds that form the molecule's backbone, but strong enough to bend the chain into more complex forms. Usually, only `A` and `U` as well as `C` and `G` pair up. Other combinations are technically not impossible, but highly improbable under normal conditions due to the structure of the distinct nitrogenous bases. Figure 2 shows a small sample of both RNA primary and secondary structure. You can see two typical patterns highlighted: a chain of paired nucleotides, called *stem* (or *stack*), and a little *hairpin loop* at its end. Figure 3 contains a secondary structure with a *multi-branched loop* in its center. Biologists often consider further patterns, such as *bulges* or *interior loops*, but those as well as hairpin loops are only special cases of the multi loop. Consequently, it is sufficient to model stems and loops that contain unpaired nucleotides and branch into stems.

Note also that there are more possible structures that do not fit this construction kit. Other connections between nucleotides, so called *pseudoknots*, occur if nucleotides in a loop pair up with bases from outside that loop. Thus, structures with pseudoknots will never be predicted by the procedure explained in the following. Even though this restriction assures certain failure in many cases, it in a way makes sense: pseudoknot prediction is in general NP-complete (see [Spe08]).

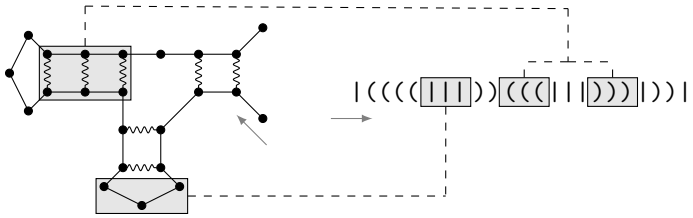


Figure 3: A secondary structure and the corresponding Motzkin word.

2.2 Secondary Structures as Strings

The set of RNA primary structures can be considered the formal language Σ^* over the alphabet $\Sigma = \{A, C, G, U\}$. That is to say that a molecule is a concatenation of symbols from Σ. Secondary structures, on the other hand, can not be represented as easily. Since sketches as seen in Figure 3 are unhandy and hard to grasp formally, many authors use a convenient string notation for secondary structures (e.g. [Neb01]) that goes by the name *Motzkin words*. Here, a recursive definition serves best.

Definition 1. The language of Motzkin words $\mathcal{M} \subseteq \{\texttt{|}, \texttt{(}, \texttt{)}\}^*$ is defined by the following rules:

- $\varepsilon, \texttt{|} \in \mathcal{M}$,
- $\alpha \in \mathcal{M} \Rightarrow \texttt{(}\alpha\texttt{)} \in \mathcal{M}$ and
- $\alpha, \beta \in \mathcal{M} \Rightarrow \alpha\beta \in \mathcal{M}$.

Every symbol in a Motzkin word represents one nucleotide. While a matching pair of parenthesis represents a base pair connected by a hydrogen bond, a pipe stands for an unpaired nucleotide. Thus, `||||` corresponds to a linear chain of four bases whereas `((|||))` represents a stem of length two with a hairpin loop containing three bases attached. See Figure 3 for a more sophisticated example. It turns out that not all Motzkin words describe structures that are physically feasible. For example, very short loops and stems can not occur due to the tension strongly angled backbones effect. Secondary structure prediction methods should take care not to predict such impossible structures. In some cases, this is solved by simply assigning such structures very low or zero probabilities. Other models rule out certain structures beforehand. In the case of [DE04], both minimal loop length $h \geq 0$ and minimum stem length $s \geq 1$ are restricted. Formally, $\mathcal{M}^{(s,h)} \subseteq \mathcal{M}$ and for every $\alpha \in \mathcal{M}^{(s,h)}$ the conditions

- $\alpha_j = \texttt{(} \;\Rightarrow\; \alpha_{i,k} = \texttt{(}^{k-i+1}$ with $i \leq j \leq k$ and $k - i + 1 \geq s$,
- $\alpha_j = \texttt{)} \;\Rightarrow\; \alpha_{i,k} = \texttt{)}^{k-i+1}$ with $i \leq j \leq k$ and $k - i + 1 \geq s$ and
- $\texttt{(}\beta\texttt{)}$ subword of $\alpha \;\Rightarrow\; |\beta| \geq h$

hold. Of course, $\mathcal{M}^{(1,0)} = \mathcal{M}$ holds.

Note that Motzkin words abstract from the fact that RNA contains four different nucleotides[3]. This is not problematic, though. Any Motzkin word of length n represents 4^n pairwise different RNA structures. Loads of them are impossible because of pairing contraints, but this matter is handled in the same manner as impossible loop or stem lengths. As long as all concerned models abstract in the same way, the results obtained on Motzkin words can be transferred to RNA secondary structures. In accordance with the cited authors, the term *secondary structure* is used as a synonym for Motzkin words from now on.

3 Secondary Structure Prediction Using SCFGs

The basic idea of RNA secondary structure prediction with stochastic context free grammars is simple. Assuming a grammar feasible to model the type of RNA and the environmental conditions in question has been found, you choose the most probable syntax tree for a given sequence and return the corresponding structure as your prediction. But as always, the devil is in the details. The following explanations are taken from both [DE04] and [DEKM98].

First, a feasible grammar has to be found. There seems to be no grammar that is universally fit to model every kind of RNA, so rather specific grammars are built. Usually, a CFG is chosen with careful consideration of structural elements and dependencies. For example, a very simple grammar might not distinguish between distinct loop types. Another might and had therefore the potential to model a certain class of RNA more precisely. Some grammars might introduce so called *stacking parameters* that assign a nucleotide pair different probabilities depending on the pair that occurs right before it. That allows to look for certain patterns in stems. When a fitting CFG is found, it is extended with probability parameters to a SCFG. These parameters have to be fixed in a process called *training* before structure prediction itself can start. This can be done manually, algorithmically or using a combination of both. Manual approaches contain guessing and using empirical values. On the other hand, a typical algorithmic approach is to count the number of rule invocations in a set of known syntax trees and to assign the relative weights as probabilities. Tweaking the training process, however implemented, can take care of ambiguities already. Simple precedence rules can shift weights in such a way that the resulting grammar is unambiguous in practice, if not formally.

Once the grammar and its parameters are fixed, actual prediction can start. There are several algorithms based on the famous dynamic programming algorithm by Cocke, Younger and Kasami (CYK). The original CYK algorithm decides wether a word α is generated by a context free grammar G in Chomsky normal form or not. A syntax tree can be found by backtracking through the created matrix. There are variations that extend that algorithm to work on grammars that are not in Chomsky normal form and even on stochstic CFGs. Two versions are available of the latter. One calculates the probability of the input word α in G, that is to say the sum of the probabilites of all syntax trees for α. This is of no use for the means of structure prediction. The other ver-

[3] But any RNA molecule without pseudoknots is uniquely described by a pair from $\Sigma^* \times \mathcal{M}$.

sion calculates the maximum of all probabilities of syntax trees that generate α and allows to find that optimal tree with backtracking. This is exactly what is needed. Assuming that each possible secondary structure of α is described by exactly one syntax tree, probabilities of syntax trees and secondary structures are equivalent and the prediction problem is solved by returning the structure corresponding to the optimal syntax tree. Yet if there were several syntax trees per structure (that is, if the grammar were *ambiguous with respect to secondary structures*), this procedure might produce a wrong result. Consider the following example for the reason.

Example 6. Let

$$G = (\{S\}, \{a\}, \{S \xrightarrow{0.1} a, S \xrightarrow{0.3} aS, S \xrightarrow{0.3} Sa, S \xrightarrow{0.3} aS\hat{a}, \}, S).$$

You can show G's consistency with a proof analogue to Example 5. The mapping from syntax trees to secondary structures is defined as follows. Invocations of the first three rules (those with a single a on the right hand side) correspond to an unpaired nucleotide. The last rule (that with two a, one of them marked) stands for a pair of nucleotides. Thus, stems are built with the last rule and single strands, for instance in loops, with the others. Since empty hairpin loops `()` are forbidden, the sequence aaa has only two possible secondary structures, `|||` and `(|)`. The structure `(|)` is created by exactly one syntax tree with probability $0.3 \cdot 0.1 = 0.03$, while `|||` is created by four syntax trees with probability $0.3^2 \cdot 0.1 = 0.009$ each (see Figures 1(a) and 1(b) for two of them). Thus, `(|)` is predicted although `|||` has the greater probability of altogether $4 \cdot 0.009 = 0.036$.

It might be possible to circumvent this problem by modifying CYK further. If it is, that has not been considered in [DE04]. But obviously, the problem dissolves if the used grammar is unambiguous with respect to secondary structures, that is to say every secondary structure is represented by exactly one syntax tree. This is why in [DE04], the authors check ambiguity of their grammars empirically. The main purpose of the thesis at hand is to examine those grammars' structural ambiguity more formally.

4 Number of Secondary Structures

The next section requires the numbers $\mathcal{S}_n^{(s,h)}$ of secondary structures of length n, restricted to such structures that have at least s pairs in any stem and at least h bases in any hairpin loop. That is to say, $\mathcal{S}_n^{(s,h)} = |\mathcal{M}_n^{(s,h)}|$. In [HSS96], the authors provide an equation whose solutions are generating functions for exactly these numbers. For $s = 1$, this equation is

$$0 = x^2(\mathcal{S}^{(1,h)}(x))^2 - \left(1 - x + x^2 \sum_{k=0}^{h-1} x^k\right) \mathcal{S}^{(1,h)}(x) + 1.$$

It can be solved like any quadratic equation, yielding two solutions. To choose the correct one, consider their series expansions around $x = 0$. Since the coefficients of $\mathcal{S}^{(s,h)}(x)$ count secondary structures, they have to be non-negative

integers. Only one solution fulfills that requirement. For those h needed later on, the following generating functions are obtained:

$$\mathcal{S}^{(1,0)}(x) = \frac{1 - x - \sqrt{1 - 2x - 3x^2}}{2x^2}$$
$$\mathcal{S}^{(1,1)}(x) = \frac{1 - x + x^2 - \sqrt{1 - 2x - x^2 - 2x^3 + x^4}}{2x^2}$$
$$\mathcal{S}^{(1,2)}(x) = \frac{1 - x + x^2 + x^3 - \sqrt{1 - 2x - x^2 - x^4 + 2x^5 + x^6}}{2x^2}$$

For $s = 2, h = 1$, the equation is given by

$$0 = x^4(\mathcal{S}^{(2,1)}(x))^2 - \left((1-x)(1-x^2+x^4) + x^4\right)\mathcal{S}^{(2,1)}(x) + (1 - x^2 + x^4),$$

which evaluates to

$$\mathcal{S}^{(2,1)}(x) = \frac{1 - x - x^2 + x^3 + 2x^4 - x^5}{2x^4} - \frac{\sqrt{1 - 2x - x^2 + 4x^3 - x^4 - 8x^5 + 3x^6 + 6x^7 - 2x^8 - 4x^9 + x^{10}}}{2x^4}.$$

Note that $\mathcal{S}_n^{(s,h)} = [x^n]\mathcal{S}^{(s,h)}(x)$ holds.

5 Analysis of Structure Prediction Grammars

In [DE04], the authors present a total of nine grammars that can be used for RNA secondary structure prediction. Their complexity ranges from three to 13 rule types, some are ambiguous while others are not and some consider stacking[4] while others do not. All grammars are tested for their prediction accuracy on a training set. The results range from 3% to 49%, justifying the consideration.

Mind that all those grammars are syntactically ambiguous. If there was only one syntax tree and therewith only one secondary structure generated for each sequence, that grammar would be useless for secondary structure prediction. Still, the technique shown in 1.4 can be applied. The coefficients of the structure functions just are not compared with the number of words, but the number of structures.

Since all necessary preparations are done, the presented grammars can now be formally analysed with regard to their structural ambiguity. That is to say that for each grammar it is first proven that it generates at least the correct set of structures. After that, the number of that structures is compared with the number of syntax trees generated by the respective grammar. If the numbers equal, the grammar is structurally unambiguous. On the other hand, if the numbers differ significantly, there might arise problems during structure prediction as described above.

All the grammars use the same structural interpretation as given in Example 6. Lone occurrences of a translate to `|`, an unpaired nucleotide, and double occurrences $a \ldots \hat{a}$ translate to `(...)`, a pair.

[4] G_{6S}, G_7 and G_8 support pair probabilites dependent on the pair occurring right before.

For notational ease, assume the same alphabet and starting symbol for all grammars:

$$G_i = \{N_i, \{a\}, \delta_i, S\}.$$

The production rules δ_i will be given for each grammar G_i, and the respective set of non-terminals N_i is the set of all upper case symbols occurring in δ_i. Note that $\hat{a}$ is not a new symbol, but a marked instance of a. This is due to the fact that although RNA is made of four different bases, the grammars abstract from alphabet size and use only one symbol. For formal analysis, this is fine for the most time. The marked copies of a, however, are a reminder that, if extending the grammar to the actual alphabet of four letters, rules with two a's (that is, an a and an $\hat{a}$) have to be instantiated once for each combination of letters, resulting in a total of 16 alternatives where here only one stands. Occurrences of $P^{b\hat{b}} \to aS\hat{a}$ have to be understood in a similar manner: Not only will there be 16 different instances of P, but each has to be combined with all 16 possible right sides, resulting in a total of 256 production rules.

5.1 $G_{\{1,2\}}$

The first two grammars presented have the following sets of rules:

$$\delta_1 = \{\ S \to aS\hat{a} \mid aS \mid Sa \mid SS \mid \varepsilon\ \},$$

and

$$\begin{aligned}\delta_2 = \{\ & S \to aP^{a\hat{a}}\hat{a} \mid aS \mid Sa \mid SS \mid \varepsilon, \\ & P^{b\hat{b}} \to aP^{a\hat{a}}\hat{a} \mid S\ \}.\end{aligned}$$

These two candidates need no in-depth investigation. Due to the alternatives $S \to SS \mid \varepsilon$ both G_1 and G_2 possess, each word has infinitely many syntax trees. Assume there would be finitely many, take one and replace an arbitrary generation step $\alpha S\beta \Rightarrow \alpha\gamma\beta$ with $\alpha S\beta[\Rightarrow \alpha SS\beta \Rightarrow \alpha S\beta] \Rightarrow \alpha\gamma\beta$. Obviously, you can iterate the part in brackets arbitrarily often, thus creating infinitely many syntax trees for one word[5].

Because there are only finitely many possible structures for any sequence, this means that both G_1 and G_2 are structurally ambiguous with degree ∞. This result agrees with [DE04].

5.2 G_3

With G_3, matters start to become interesting. But first of all, the set of rules:

$$\begin{aligned}\delta_3 = \{\ & S \to aS\hat{a} \mid aL \mid Ra \mid LS, \\ & L \to aS\hat{a} \mid aL, \\ & R \to Ra \mid \varepsilon\ \}.\end{aligned}$$

[5] This is why the technique used on the other grammars won't work here, anyway.

There is no apparent reason to regard this grammar ambiguous, so the technique presented in Section 1.4 is used on it. As outlined above, the number of syntax trees this grammar generates is compared with the number of structures they imply. To select the correct number to compare with it has to be proven which class of structures G_3 generates. To do this, the equivalency of secondary structures and Motzkin words is used. G_3 can easily be transformed into the grammar $G_3^{\mathcal{M}}$ that clearly creates exactly the Motzkin words that correspond to the secondary structures G_3 generates according to the interpretation given above:

$$\begin{aligned}\delta_3^{\mathcal{M}} = \{\ & S \to (S) \mid \mid L \mid R\mid \mid LS,\\ & L \to (S) \mid \mid L,\\ & R \to R\mid \mid \varepsilon\ \}.\end{aligned}$$

Now, if $\mathcal{L}(G_3^{\mathcal{M}}) = \mathcal{M}_{>0}^{(1,1)}$ holds, G_3 generates exactly those secondary structures that are not of length zero and whose stems and loops have at least length one. In fact, $\mathcal{M}_{>0}^{(1,1)} \subseteq \mathcal{L}(G_3^{\mathcal{M}})$ is sufficient. If

1. at least the structures from $\mathcal{M}_{>0}^{(1,1)}$ can be generated and
2. exactly $\mathcal{S}_n^{(1,1)}$ syntax trees for words of length $n > 1$ are generated,

it can be concluded that for each structure in $\mathcal{M}_{>0}^{(1,1)}$ there is exactly one syntax tree in G_3. That would be the desired result. The first of the required facts can be proven using a standard technique, induction over word length n.

Claim: $\mathcal{M}_{>0}^{(1,1)} \subseteq \mathcal{L}(G_3^{\mathcal{M}})$

Induction Basis: $\mid \in \mathcal{L}(G_3^{\mathcal{M}}) \cap \mathcal{M}^{(1,1)}$ since $\mid$ can be generated in $G_3^{\mathcal{M}}$ using the generation $S \Rightarrow R\mid \Rightarrow \mid$.

Induction Hypothesis: $(\mathcal{M}_{<n}^{(1,1)} \setminus \{\varepsilon\}) \subseteq \mathcal{L}(G_3^{\mathcal{M}})_{<n}$.

Inductive Step: Let $\alpha \in \mathcal{M}_n^{(1,1)}$ be chosen arbitrarily, but fixed. Then one of the following cases applies:

- $\alpha = \mid^n$:
  Consider the generation $S \Rightarrow R\mid \Rightarrow^{n-1} R\mid^n \Rightarrow \mid^n$.
- $\alpha = (\beta)$ with $\beta \in \mathcal{M}_{n-2}^{(1,1)} \setminus \{\varepsilon\}$:
  Consider $S \Rightarrow (S) \Rightarrow^* (\beta)$, with $S \Rightarrow^* \beta$ given by induction hypothesis.
- $\alpha = \mid^k(\beta)$ with $k \geq 1$ and $\beta \in \mathcal{M}_{<n}^{(1,1)} \setminus \{\varepsilon\}$:
  Consider $S \Rightarrow^k \mid^k L \Rightarrow \mid^k(S) \Rightarrow^* \mid^k(\beta)$, with $S \Rightarrow^* \beta$ again given by induction hypothesis.
- $\alpha = \mid^k(\beta)\gamma$ with $k \geq 0$ and $\beta, \gamma \in \mathcal{M}_{<n}^{(1,1)} \setminus \{\varepsilon\}$:
  Consider $S \Rightarrow LS \Rightarrow^k \mid^k LS \Rightarrow \mid^k(S)S \Rightarrow \mid^k(\beta)\gamma$ using the induction hypothesis for both $S \Rightarrow^* \beta$ and $S \Rightarrow^* \gamma$.

Since in all cases $\alpha \in \mathcal{L}(G_3^{\mathcal{M}})_n$ has been shown, the inductive step is complete.

Now the first fact is shown, the number of syntax trees G_3 creates can be determined. First, translate G_3 to the equation system

$$\begin{aligned} S &= x^2S + xL + xR + LS \\ L &= x^2S + xL \\ R &= xR + 1 \end{aligned}$$

Note that to be completely correct, every S, L and R should be written $S(x), L(x)$ and $R(x)$ since they are rather functions in x than symbols here. But because they can not be mistaken for symbols in this context, readability is given precedence over literal correctness. Using either trivial algebra or a computer algebra system of choice, the system can be solved to obtain

$$S(x) = \frac{1 - x - x^2 - \sqrt{1 - 2x - x^2 - 2x^3 + x^4}}{2x^2}.$$

Note that this solution has been chosen out of two for the same reasons as were explained in Section 4.

$S(x)$ is suspiciously similar to $\mathcal{S}^{(1,1)}(x)$. It also strikes that $\varepsilon \notin \mathcal{L}(G_3)$ whereas $\mathcal{S}^{(1,1)}(x)$ does count ε. And indeed,

$$\mathcal{S}^{(1,1)}(x) - 1 = S(x)$$

holds, implying the second fact needed. Combining the facts, it is proven that G_3 is unambiguous with regards to secondary structures. That confirms the empirical results in [DE04].

5.3 G_4

The next grammar in question is G_4 with the rules

$$\begin{aligned} \delta_4 = \{\ & S \to aS \mid T \mid \varepsilon, \\ & T \to Ta \mid aS\hat{a} \mid TaS\hat{a}\ \}. \end{aligned}$$

The claim that $\mathcal{M} \subseteq \mathcal{L}(G_4^{\mathcal{M}})$ is proven by induction over word length n. $S \Rightarrow \varepsilon$ makes up the base case. Assuming $\mathcal{M}_{<n} \subseteq \mathcal{L}(G_4^{\mathcal{M}})_{<n}$, there are the following possibilities of what $\alpha \in \mathcal{M}_n$ can look like:

- $\alpha = \,|\beta$ with $\beta \in \mathcal{M}_{<n}$:
  Since $S \Rightarrow |S \Rightarrow^* |\beta$ holds by induction hypothesis, $\alpha \in \mathcal{L}(G_4^{\mathcal{M}})_n$.
- $\alpha = (\beta)\gamma$ with $\beta, \gamma \in \mathcal{M}_{<n}$:
  Now $S \Rightarrow T \Rightarrow^* T\gamma \Rightarrow (S)\gamma \Rightarrow^* (\beta)\gamma$ holds with one application of the induction hypothesis and the fact that for all $\gamma \in \mathcal{M}_{<n}$, $T \Rightarrow^* T\gamma$. This is also shown by induction. For $\gamma = \varepsilon$, the claim obviously holds. Assuming that it also holds for all words in $\mathcal{M}_{<k}$, the following cases for $\gamma \in \mathcal{M}_k$ can occur:
  - $\gamma = \nu|$ with $\nu \in \mathcal{M}_{<k}$:
    The claim holds with $T \Rightarrow T| \Rightarrow^* T\nu|$, using the induction hypothesis of this inner proof.

– $\gamma = \mu(\nu)$ with $\nu, \mu \in \mathcal{M}_{<k}$:
Here, $T \Rightarrow T(S) \Rightarrow^* T\mu(S) \Rightarrow^* T\mu(\nu)$ with both the inner and the outer induction hypothesis.

For $\alpha \in \mathcal{L}(G_4^{\mathcal{M}})_n$ holds in all cases, the claim is proven. G_4 is now transformed to the equation system

$$S = xS + T + 1$$
$$T = xT + x^2S + x^2TS$$

whose solution includes

$$S(x) = \frac{1 - x - \sqrt{1 - 2x - 3x^2}}{2x^2}.$$

Since $S(x) = \mathcal{S}^{(1,0)}(x)$, G_4 is unambiguous with regards to secondary structures, again in accordance with [DE04].

5.4 G_5

Next comes G_5, the smallest grammar presented:

$$\delta_5 = \{\ S \to aS \mid aS\hat{a}S \mid \varepsilon\ \}.$$

First, it has to be proven that $\mathcal{M} \subseteq \mathcal{L}(G_5^{\mathcal{M}})$. That can again be done by induction over word length n. The base case, $\mathcal{M}_0 \subseteq \mathcal{L}(G_5^{\mathcal{M}})_0$, is easy since clearly $\varepsilon \in \mathcal{L}(G_5^{\mathcal{M}})$. Now assume $\mathcal{M}_{<n} \subseteq \mathcal{L}(G_5^{\mathcal{M}})_{<n}$ and let $\alpha \in \mathcal{M}_n$. Then, one of the following cases applies:

- $\alpha = \,|\,\beta$ with $\beta \in \mathcal{M}_{<n}$:
Since $S \Rightarrow \,|\,S \Rightarrow^* \,|\,\beta$ holds by induction hypothesis, $\alpha \in \mathcal{L}(G_5^{\mathcal{M}})_n$.
- $\alpha = (\beta)\gamma$ with $\beta, \gamma \in \mathcal{M}_{<n}$:
Since $S \Rightarrow (S)S \Rightarrow^* (\beta)S \Rightarrow^* (\beta)\gamma$ holds with two applications of the induction hypothesis, $\alpha \in \mathcal{L}(G_5^{\mathcal{M}})_n$.

So the claim is proven. Now transform G_5 to

$$S = xS + x^2S^2 + 1,$$

which also results in $S(x) = \mathcal{S}^{(1,0)}(x)$, proving G_5 to be unambiguous with regards to secondary structures as claimed in [DE04].

5.5 G_6

G_6 is given by

$$\begin{aligned}\delta_6 = \{\ & S \to LS \mid L,\\ & L \to aF\hat{a} \mid a,\\ & F \to aF\hat{a} \mid LS\ \}.\end{aligned}$$

To show that $\mathcal{M}_{>0}^{(1,2)} \subseteq \mathcal{L}(G_6^{\mathcal{M}})$, some special cases have to be eliminated. First of all, $S \Rightarrow L \Rightarrow |$ holds. Now assume that also

$$LS \Rightarrow^* \alpha \quad \text{for all } \alpha \in \mathcal{M}_{>1}^{(1,2)} \setminus \{(\beta) : \beta \in \mathcal{M}\}$$

holds. If so, then clearly those two generations are sufficient to proof the original claim:

$$S \Rightarrow LS \text{ and}$$
$$S \Rightarrow^* (F) \Rightarrow^{k-1} ({}^kF)^k \Rightarrow ({}^kLS)^k.$$

Note that k is assumed to be maximal for any word generated via the second option here[6]. Of course, the hasty assumption from above has to be proven. The induction base is $LS \Rightarrow |S \Rightarrow |L \Rightarrow ||$ and the induction hypothesis is

$$LS \Rightarrow^* \alpha' \quad \text{for all } \alpha' \in \mathcal{M}_{<n}^{(1,2)} \setminus \{\varepsilon, |, (\beta) : \beta \in \mathcal{M}\}.$$

Then, for any $\alpha \in \mathcal{M}_n^{(1,2)}$ with $n > 2$, one of the following cases applies:

- $\beta = |\beta$ with $\beta \in \mathcal{M}_{>1}^{(1,2)} \cap \mathcal{M}_{<n}^{(1,2)}$:
  Consider $LS \Rightarrow LLS \Rightarrow |LS \Rightarrow^* |\beta$ and use the induction hypothesis for $LS \Rightarrow^* \beta$.
- $\alpha = ({}^k\beta)^k|$ with k maximal and $\beta \in \mathcal{M}_{>1}^{(1,2)} \cap \mathcal{M}_{<n}^{(1,2)}$:
  Consider

  $$\begin{aligned} LS &\Rightarrow (F)S \Rightarrow^{k-1} ({}^kF)^kS \Rightarrow ({}^kLS)^kS \\ &\Rightarrow ({}^kLS)^kL \Rightarrow ({}^kLS)^k| \Rightarrow^* ({}^k\beta)^k| \end{aligned}$$

  using the induction hypothesis for $LS \Rightarrow^* \beta$ again.
- $\alpha = ({}^k\beta)^k\gamma$ with k maximal and $\beta, \gamma \in \mathcal{M}_{>1}^{(1,2)} \cap \mathcal{M}_{<n}^{(1,2)}$:
  Consider

  $$\begin{aligned} LS &\Rightarrow LLS \Rightarrow (F)LS \Rightarrow^{k-1} ({}^kF)^kLS \\ &\Rightarrow ({}^kLS)^kLS \Rightarrow^* ({}^k\beta)^k\gamma \end{aligned}$$

  with double application of induction hypothesis.

Thus, $\mathcal{M}_{>0}^{(1,2)} \subseteq \mathcal{L}(G_6^{\mathcal{M}})$ is proven and G_6 can be translated to its equation system.

$$S = LS + L$$
$$L = x^2F + x$$
$$F = x^2F + LS$$

Solving the system produces

$$S(x) = \frac{1}{2}\left(\frac{1}{x^2} - \frac{1}{x} - 1 + x - \frac{\sqrt{1 - 2x - x^2 - x^4 + 2x^5 + x^6}}{x^2}\right)$$

[6] In this context, k is maximal for $\alpha = ({}^k\beta)^k$ if there is no $m > k$ with $\alpha = ({}^m\beta')^m$.

With the same observation as made for G_3,

$$S(x) = \mathcal{S}^{(1,2)}(x) - 1$$

implies that G_6 is unambiguous with regards to secondary structures. Again, [DE04] is confirmed.

5.6 G_{6S}

This grammar is an extension of G_6 to support stacking parameters. Is has the set of rules

$$\begin{aligned}\delta_{6S} = \{\ & S \to LS \mid L, \\ & L \to aF^{a\hat{a}}\hat{a} \mid a, \\ & F^{b\hat{b}} \to aF^{a\hat{a}}\hat{a} \mid LS\ \}.\end{aligned}$$

G_{6S} generates the same structures as G_6. This can be shown with a proof that is analoguous to that provided for G_6 and is therefore left out here. It is clear that at this point, abstracting from $|\Sigma|$ will no longer work since the number of non-terminals (and therewith the number of equations) depends on the number of terminals. So, the resulting equation system is

$$\begin{aligned}S &= LS + L \\ L &= \sum_{(a,\hat{a})\in\Sigma^2} x^2 F^{a\hat{a}} + |\Sigma|x \\ \big[F^{b\hat{b}} &= \sum_{(a,\hat{a})\in\Sigma^2} x^2 F^{a\hat{a}} + LS : (b,\hat{b}) \in \Sigma^2\ \big]\end{aligned}$$

Fortunately, since all $F^{a\hat{a}}$ are equal they can be identified in the equations and merged into one alias F. The system can be significantly simplified, and now the number of equations is independent of $|\Sigma|$:

$$\begin{aligned}S &= LS + L \\ L &= |\Sigma|^2 x^2 F + |\Sigma|x \\ F &= |\Sigma|^2 x^2 F + LS\end{aligned}$$

But now, all dependance on $|\Sigma|$ can be removed again with the same arguments as when it got removed from the non-stacking grammars in the first place. This leads to the system

$$\begin{aligned}S &= LS + L \\ L &= x^2 F + x \\ F &= x^2 F + LS\end{aligned}$$

which is exactly the same as the one that occurred during the analysis of G_6. Thus, the further procedure can be omitted here and it can be concluded directly that again, [DE04] is right and G_{6S} is unambiguous with regards to secondary structures.

5.7 G_7

Grammar G_7 is given by the set of rules

$$\begin{aligned}\delta_7 = \{\ & S \to aP^{a\hat{a}}\hat{a} \mid aL \mid Ra \mid LS,\\ & L \to aP^{a\hat{a}}\hat{a} \mid aL,\\ & R \to Ra \mid \varepsilon,\\ & P^{b\hat{b}} \to aP^{a\hat{a}}\hat{a} \mid aN\hat{a},\\ & N \to aL \mid Ra \mid LS\ \}.\end{aligned}$$

Here, $\mathcal{M}_{>0}^{(2,1)} \subseteq \mathcal{L}(G_7^{\mathcal{M}})$ holds. The induction base is given by $S \Rightarrow R| \Rightarrow |$ and as induction hypothesis assume that $\mathcal{M}_{<n}^{(1,2)} \setminus \{\varepsilon\} \subseteq \mathcal{L}(G_7^{\mathcal{M}})$ holds. Then for any $\alpha \in \mathcal{M}_n^{(2,1)}$, one of the following cases applies:

- $\alpha = |^n$:
  Use $S \Rightarrow R| \Rightarrow^* R|^n \Rightarrow |^n$ directly.

- $\alpha = (^k\beta)^k$ with $k > 1$ maximal and $\beta \in \mathcal{M}_{<n}^{(2,1)} \setminus \{\varepsilon\}$:
  Generate

  $$S \Rightarrow (P^{||}) \Rightarrow^* (^{k-1}P^{||})^{k-1} \Rightarrow (^kN)^k \Rightarrow^* (^k\beta)^k.$$

  It is obvious that the *correct* $P^{a\hat{a}}$ can be chosen in any case. $N \Rightarrow^* \beta$ holds because $S \Rightarrow^* \beta$ does so by induction hypothesis. More elaborately, N has all alternatives S has but $(P^{||})$ while $S \Rightarrow (P^{||})$ can not be the first generation step in $S \Rightarrow^* \beta$ for k is maximal in α. Therefore, the induction hypothesis extends to N if the generated word is restricted appropriately, and it indeed is.

- $\alpha = |^m(^k\beta)^k$ with $m > 0, k > 1$ maximal and $\beta \in \mathcal{M}_{<n}^{(2,1)} \setminus \{\varepsilon\}$:
  Generate

  $$\begin{aligned}S &\Rightarrow |L \Rightarrow^* |^mL \Rightarrow |^m(P^{||})\\ &\Rightarrow^* |^m(^{k-1}P^{||})^{k-1} \Rightarrow |^m(^kN)^k\\ &\Rightarrow^* |^m(^k\beta)^k\end{aligned}$$

  using the induction hypothesis on N as above for $N \Rightarrow^* \beta$.

- $\alpha = |^m(^k\beta)^k\gamma$ with $m \geq 0, k > 1$ maximal and $\beta, \gamma \in \mathcal{M}_{<n}^{(2,1)} \setminus \{\varepsilon\}$:
  Generate

  $$\begin{aligned}S &\Rightarrow LS \Rightarrow^* |^mLS \Rightarrow |^m(P^{||})S\\ &\Rightarrow^* |^m(^{k-1}P^{||})^{k-1}S \Rightarrow |^m(^kN)^kS\\ &\Rightarrow^* |^m(^k\beta)^k\gamma\end{aligned}$$

  with two applications of the induction hypotheses, one of them done indirectly like above.

Going through the same steps as for G_{6S} above, the rules of G_7 can be translated to the equation system

$$\begin{aligned}
S &= x^2P + xL + xR + LS \\
L &= x^2P + xL \\
R &= xR + 1 \\
P &= x^2P + x^2N \\
N &= xL + xR + LS
\end{aligned}$$

As solution, this system yields

$$\begin{aligned}
S(x) = &\frac{1 - x - x^2 + x^3 - x^5}{2x^4} \\
&- \frac{\sqrt{1 - 2x - x^2 + 4x^3 - x^4 - 8x^5 + 3x^6 + 6x^7 - 2x^8 - 4x^9 + x^{10}}}{2x^4},
\end{aligned}$$

what equals $\mathcal{S}^{(2,1)}(x) - 1$. With the same argumentation as for G_3 it can be concluded that G_7 is unambiguous with regard to secondary structures. [DE04] is once more proven correct.

5.8 G_8

The next and last grammar G_8 is given by

$$\begin{aligned}
\delta_8 = \{\ & S \to aS \mid T \mid \varepsilon, \\
& T \to Ta \mid aP^{a\hat{a}}\hat{a} \mid TaP^{a\hat{a}}\hat{a}, \\
& P^{b\hat{b}} \to aP^{a\hat{a}}\hat{a} \mid aN\hat{a}, \\
& N \to aS \mid Ta \mid TaP^{a\hat{a}}\hat{a}\ \}.
\end{aligned}$$

For G_8, $\mathcal{M}^{(2,1)} \subseteq \mathcal{L}(G_8^{\mathcal{M}})$ holds which is of course proven with induction over word length n. The induction base is $S \Rightarrow \varepsilon$. Assume as induction hypothesis as usual that $\mathcal{M}_{<n}^{(2,1)} \subseteq \mathcal{L}(G_8^{\mathcal{M}})$ holds. For $\alpha \in \mathcal{M}_n^{(2,1)}$, one of the following cases applies:

- $\alpha = \mid^n$:
  Generate α directly with $S \Rightarrow \mid S \Rightarrow^* \mid^n S \Rightarrow \mid^n$.
- $\alpha = \mid^m \beta$ with $m \geq 0, \beta = (^{k_t}\gamma_t)^{k_t} \mid^{l_t} \ldots (^{k_1}\gamma_1)^{k_1} \mid^{l_1}$ and $k_i > 1$ maximal, $l_i \geq 0, \gamma_i \in \mathcal{M}_{<n}^{(2,1)} \setminus \{\varepsilon\}$ for $1 \leq i \leq t, t > 0$:
  Generate $S \Rightarrow^* \mid^m S \Rightarrow \mid^m T \Rightarrow^* \mid^m \beta$ where $T \Rightarrow^* \beta$ has to be shown seperately. The induction base for $t = 1$ is given by the generation

  $$\begin{aligned}
  T &\Rightarrow^* T\mid^{l_1} \Rightarrow (P^{\mid\mid})\mid^{l_1} \Rightarrow^* (^{k_1-1}P^{\mid\mid})^{k_1-1}\mid^{l_1} \\
  &\Rightarrow (^{k_1}N)^{k_1}\mid^{l_1} \Rightarrow^* (^{k_1}\gamma_1)^{k_1}\mid^{l_1}.
  \end{aligned}$$

  $N \Rightarrow^* \gamma_1$ follows from the outer induction hypothesis in the same manner as explained in the proof for G_7. Here, the first two generation steps for γ_1

have to be considered. Assuming T can generate any such β with $t < t_0$, generate β for $t = t_0$ like that:

$$\begin{aligned} T &\Rightarrow^* T\,|^{l_1} \Rightarrow T\,(P^{||})\,|^{l_1} \Rightarrow^* T\,(^{k_1-1}P^{||})^{k_1-1}\,|^{l_1} \\ &\Rightarrow T\,(^{k_1}N)^{k_1}\,|^{l_1} \Rightarrow^* T\,(^{k_1}\gamma_1)^{k_1}\,|^{l_1} \\ &\Rightarrow^* (^{k_{t_0}}\gamma_{t_0})^{k_{t_0}}\,|^{l_{t_0}} \ldots (^{k_1}\gamma_1)^{k_1}\,|^{l_1}, \end{aligned}$$

using both induction hypotheses.

From here on, G_8 can be treated in the same manner as G_7, translating the set of rules to

$$\begin{aligned} S &= xS + T + 1 \\ T &= xT + x^2P + x^2TP \\ P &= x^2P + x^2N \\ N &= xS + xT + x^2TP \end{aligned}$$

This system's solution contains $S(x) = \mathcal{S}^{(2,1)}(x)$, implying that G_8 is unambiguous with respect to secondary structures and [DE04] continues to be right.

6 Conclusion

In [DE04], the authors use several grammars for RNA secondary structure prediction and claim most were unambiguous with regards to secondary structures. Formal proofs for the claims are not given, leaving behind some doubt. If a grammar failed to be unambiguous, prediction algorithms would fail. In this work, it is proven that all grammars that are claimed to be unambiguous in [DE04] are indeed so. This result ensures that RNA secondary structure prediction algorithms using any of the grammars G_3 through G_8 work indeed correctly in every case.

Acknowledgements

I thank Prof. Dr. Markus Nebel for continuously supporting me not only in the making of this thesis, but also in my studies in general. Further thanks go to Lars Hüttenberger, who has proven to be a valuable discussion partner throughout the concurrent making of our theses, as well as Reiner Hüchting, Simon Schröder, Frank Weinberg and Uli Laube, who have each helped me out once in a while.

Last but not least, I send warm regards and thanks to my proofreaders, namely Hannah Bayer, Katharina Heil, Isabel Reitzig and Lars Scherer.

References

[CRW09] Comparative RNA Web Site and Project, August 2009. http://www.rna.ccbb.utexas.edu/DAT/.

[CS63] N. Chomsky and M. P. Schützenberger. *Computer Programming and Formal Systems*, chapter The Algebraic Theory of Context-Free Languages, pages 118–161. Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Company, third printing, 1970 edition, 1963.

[DE04] Robin Dowell and Sean Eddy. Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. *BMC Bioinformatics*, 5(1):71, 2004. ISSN 1471-2105. URL `http://www.biomedcentral.com/1471-2105/5/71`.

[DEKM98] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis: Probabilistic models of proteins and nucleic acids*. Press Syndicate of the University of Cambridge, 1998.

[HF71] T. Huang and K.S. Fu. On Stochastic Context-Free Languages. *Information Sciences*, 3(3):201–224, 1971.

[HSS96] Ivo L. Hofacker, Peter Schuster, and Peter F. Stadler. Combinatorics of RNA Secondary Structures. *Discr. Appl. Math*, 89, 1996.

[Kui70] Werner Kuich. On the Entropy of Context-Free Languages. *Information and Control*, 16(2):173–200, 1970.

[Neb01] Markus E. Nebel. Combinatorial Properties of RNA Secondary Structures. *Journal of Computational Biology*, 9:541–573, 2001.

[Sch01] Uwe Schöning. *Theoretische Informatik - kurzgefasst*. Spektrum Akademischer Verlag, 4th edition, 2001.

[Spe08] Volker Sperschneider. *Bioinformatics - Problem Solving Paradigms*. Springer-Verlag, 2008.

[Wil90] Herbert S. Wilf. *generatingfunctionology*. Academic Press, second edition, internet edition edition, 1990. URL `http://www.math.upenn.edu/~wilf/DownldGF.html`.